Systematische Analyse und Konstruktion integrierter Sicherheitsarchitekturen für mobile verteilte Systeme

Harald Görl

Fakultät für Informatik
der Technischen Universität München

Systematische Analyse und Konstruktion integrierter Sicherheitsarchitekturen für mobile verteilte Systeme

Harald Görl

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Peter Paul Spies
Prüfer der Dissertation:	
	1. Univ.-Prof. Dr. Uwe Baumgarten
	2. Univ.-Prof. Dr. Claudia Eckert Technische Universität Darmstadt

Die Dissertation wurde am 21.12.2004 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 27.6.2005 angenommen.

Abstract

This work is about the development of a secure operating system, while taking the special threats against mobile devices into consideration. Known attacks, threats and vulnerabilities are classified by using a developed scheme and that provides a basis for the generic security analysis. Given this, a catalog of known counter measures is created, containing all detectable attacks to avoid and prevent. A special method for systematic security analysis will be presented, which is applied on a system model and will detect attacks, threats and vulnerabilities in the system. The security analysis method does contain a vulnerability analysis and a threat analysis and the attack analysis. Some tools are described that support the development and the analysis of an operating system kernel. While using the developed security analysis method, the security target is protected against well know classes of threats, attacks and vulnerabilities. The method also describes how to document the steps for successful attacks and counter measures against them and thus creates a comprehensible security architecture.

Kurzfassung

Heutige mobile Endgeräte besitzen ein hohes Maß an Rechen-, Speicher- und Kommunikationsfähigkeit, was durch komplexer werdende Hardware und Betriebssysteme ermöglicht wird. Die eingesetzte Betriebssoftware unterstützt spezielle Aspekte der Mobilität und besitzt immer mehr Funktionen klassischer Betriebssysteme. Obwohl aus dem Bereich der klassischen Betriebssysteme viele Mechanismen bekannt sind, sind kaum wirkungsvolle Sicherheitsmechanismen integriert. Meist werden nur abgesicherte Kommunikationsprotokolle und Kryptografiebibliotheken angeboten. Selbst wenn diese von den Anwendungen und den Benutzern korrekt eingesetzt werden, sind dennoch eine Vielzahl von Bedrohungen auf das System vorhanden. Trotz der Existenz vieler wirkungsvoller Sicherheitskonzepte mangelt es an deren Integration in ein Gesamtsystem.

Das Ziel dieser Arbeit ist die Realisierung eines abgesicherten Betriebssystems für mobile Endgeräte. Die besonderen Aspekte der Mobilität, wie etwa die sich ändernden Umgebungseigenschaften, werden dabei berücksichtigt. Besonderer Schwerpunkt liegt auf der bedrohungsorientierten Sicht bei der Entwicklung. Aufgrund der Ergebnisse einer Angriffs- und Bedrohungsanalyse werden bestehende Sicherheitsmaßnahmen ausgewählt. Neben der Neuentwicklung wird auch die Abhärtung bestehender Systeme betrachtet.

Als Basis dient eine ausführliche Analyse bestehender Angriffe, Bedrohungen und Schwachstellen, die durch ein entwickeltes Schema klassifiziert werden. Daneben wird ein Katalog mit existierenden Gegenmaßnahmen aufgebaut, in dem jeweils die Klassen der vermeidbaren oder verhinderbaren Angriffe beschrieben werden. Es wird ein Verfahren vorgestellt, mit dem auf Basis eines Systemmodells eine systematische Sicherheitsanalyse durchgeführt werden kann. Diese Analyse besteht aus der Schwachstellen-, Bedrohungs- und Angriffsanalyse. Die Schwachstellenanalyse kann auch auf bestehendem Quelltext durchgeführt werden. Dazu wurde ein Werkzeug zur Programmanalyse entwickelt, das mit dem Wissen über bekannte Angriffe Schwachstellen findet. Wesentliche Bedrohungen werden durch sogenannte Bedrohungsbäume dargestellt und dokumentiert. Zu diesem Zweck wurde ein Werkzeug zur Generierung, Verwaltung und Transformation von Angriffs- und Bedrohungsbäumen erstellt. Mit den entwickelten Werkzeugen und der Systematik zur Sicherheitsanalyse können durch das gewonnene Wissen über Schwachstellen, mögliche Angriffe und Bedrohungen auf ein System gezielt bestehende Sicherheitsmechanismen integriert und angewendet werden. Dadurch entsteht eine den Anforderungen angepasste, sehr wirkungsvolle Sicherheitsarchitektur, die aufgrund der Dokumentation nachvollziehbar und transparent ist. Abschließend wird das beschriebene Verfahren bei der Realisierung eines abgesicherten Betriebssystems für mobile Endgeräte angewandt.

Danksagung

Herzlichst danken möchte ich an dieser Stelle vor allem Herrn Prof. Dr. Uwe Baumgarten, der mir die Möglichkeit gegeben hat, an diesem Thema lange Zeit zu arbeiten, mir trotzdem den nötigen Spielraum gegeben hat und immer für Fragen und Diskussionen zur Verfügung stand. Weiterer Dank gebührt Prof. Dr. Claudia Eckert, die mich zum Kernthema dieser Arbeit hingeführt hat, diese Arbeit begutachtet hat, aber leider während meiner Arbeit die Hochschule gewechselt hat. Herrn Prof. Dr. Peter Paul Spies danke ich für die Bereitschaft, den Vorsitz der Prüfungskomission zu übernehmen.

Ich möchte mich hier bei allen Kollegen bedanken, die, jeder für sich, seinen Beitrag zum Gelingen dieser Arbeit geleistet haben. Dazu zählen Alexander Buchmann und Sven Lachmund, die sich trotz schwieriger Aufgaben immer an langen Diskussionen über diese Arbeit beteiligt und viele wichtige Anregungen gegeben haben. Zudem möchte ich noch Christian Rehn danken, der durch einen anderen Blickwinkel der Arbeit gut Impulse gegeben hat. Für alle studentischen Hilfkräfte soll hier stellvertretend Ingo Schnabel erwähnt werden, der ausdauerndes Interesse für diese Arbeit bewiesen hat.

Herzlicher Dank geht an alle meine Freunde, die mir immer mit guten Ratschlägen beiseite standen und oft ermutigende Worte gefunden haben. Natürlich danke ich meinen Eltern, die mir schon sehr früh die Möglichkeiten gegeben haben, mich für Naturwissenschaften und für die Informatik zu begeistern.

Mein wichtigster Dank gilt jedoch meiner Frau Bettina für die unermüdliche Bereitschaft mich bei dieser Arbeit zu unterstützen und Peter dafür, dass er gerade bei der Fertigstellung dieser Arbeit seinen Papa manchmal zwar sehen, dann aber nur wenig mit ihm spielen konnte.

Inhaltsverzeichnis

Kapitel 1

Einführung

Die IT-Sicherheit ist schon seit einigen Jahrzehnten ein wichtiges Forschungsgebiet sowohl in den theoretischen, als auch in den angewandten praktischen und technischen Bereichen der Informatik. In diesem einleitenden Kapitel wird zunächst motiviert, in welchem Bereich der IT-Sicherheit die vorliegende Arbeit eingeordnet ist und mit welchen anderen Arbeiten sie verwandt ist. Zudem wird erklärt, wie sich das Vorgehen in dieser Arbeit von den bereits bestehenden Entwicklungsmethoden zur Konstruktion sicherer Systeme unterscheidet. Nachdem anschließend die Ziele erläutert wurden, werden der Aufbau der Arbeit und die Struktur der einzelnen Kapitel im Überblick vorgestellt.

1.1 Motivation

Sicherheit[1] stellt eine Eigenschaft dar, die beim Einsatz von IT-Systemen vorausgesetzt wird. Meist rückt sie allerdings erst dann in den Vordergrund, wenn gefährliche Anlässe, Unfälle oder Katastrophen eingetreten sind. Viele dieser Ereignisse sind durch präventive Maßnahmen einschränkbar und vermeidbar, eine wünschenswerte, vollständige Verhinderung hängt jedoch im Wesentlichen von der Komplexität des betrachteten Systems ab.

1.1.1 Der System-Begriff

Ein System ist eine begrenzte, funktionale Einheit. Gegenstand der Untersuchung in dieser Arbeit sind allgemein **informationsverarbeitende Systeme**, die aus einer beliebigen Kombination von Hardware und Software bestehen und für bestimmte Zwecke eingesetzt werden. Informationsverarbeitende Systeme dienen allgemein zur Speicherung, Kommunikation oder Weiterverarbeitung von Information, wobei meist auch Personen, beispielsweise

[1] Der Sicherheitsbegriff wird in Abschnitt 1.1.3 genauer betrachtet

die Anwender, mit in den Systembegriff einbezogen werden. Ein technisches informationsverarbeitendes System, auch als **IT-System** oder **Rechensystem** bezeichnet, dagegen beschränkt die Sicht auf die technischen Einheiten des Systems, bestehend aus Hardware und Software. Es existieren verschiedene Aspekte, die bei der Entwicklung und im Betrieb des IT-Systems ein Rolle spielen, wie etwa Mobilität und die Sicherheit. Andere Eigenschaften, wie beispielsweise die Geschwindigkeit, sind oft wenig exakt verwendet, teilweise sogar subjektiv und können nur dann verstanden werden, wenn neben dem IT-System auch seine Umgebung betrachtet wird, wie etwa dessen Benutzer.

Definition 1.1 (Rechensystem)
Ein Rechensystem *ist ein technisches, dynamisches System mit Fähigkeiten zur Speicherung und Verarbeitung von Information und der Kommunikation. Es besteht aus aktiven und passiven Komponenten.*

□

Innerhalb des Rechensystems werden Informationen in Form von Datenobjekten gespeichert und verarbeitet. Dabei existieren einerseits die *passiven Objekte*, die Daten speichern, wie etwa Dateien oder programmiersprachliche Datenstrukturen. Andererseits gibt es *aktive Objekte* mit der zusätzlichen Fähigkeit, diese Daten weiterzuverarbeiten. Alle aktiven Objekte, auch die, die im Auftrag von Benutzern eine Aufgabe erledigen, werden *Subjekte* genannt. Die Benutzer selbst sind nicht Bestandteil des IT-Systems, müssen aber bei der Betrachtung verschiedener Eigenschaften, insbesondere der Sicherheitseigenschaften, mit berücksichtigt werden.

Der Begriff Rechensystem wird in dieser Arbeit synonym zu *IT-System* verwendet und umfasst sowohl integrierte Hardware als auch Software. Mit dem allgemeineren Begriff *System* werden IT-Systeme verbunden mit ihrer Umgebung bezeichnet, also vor allem auch unter Einbeziehung der Benutzer.

IT-Systeme sind Werkzeuge, die für das Speichern, Weiterverarbeiten und den Transport von Daten zuständig sind. Dennoch sind nicht die Daten als solche, sondern die Informationen, die diese repräsentieren, für den Benutzer schließlich interessant und wertvoll. Informationen werden durch Interpretation aus den verarbeiteten Daten gewonnen, wobei dieser Schritt im Allgemeinen nur durch den Menschen geschehen kann. Eine grundlegende Einführung dazu findet sich beispielsweise in [Bro98].

1.1.2 Daten und Informationen

Wenn etwa der aktuelle Kontostand eines Kontos von 1234,56 € in einem IT-System als Zahlenwert „1234,56“ gespeichert wird, geht die echte Bedeutung dieser Zahl als monetärer Betrag verloren. Nur die richtige Anwendung, bzw. die Interpretation der Zahl im richtigen Kontext liefert wieder die Information über den Kontostand.

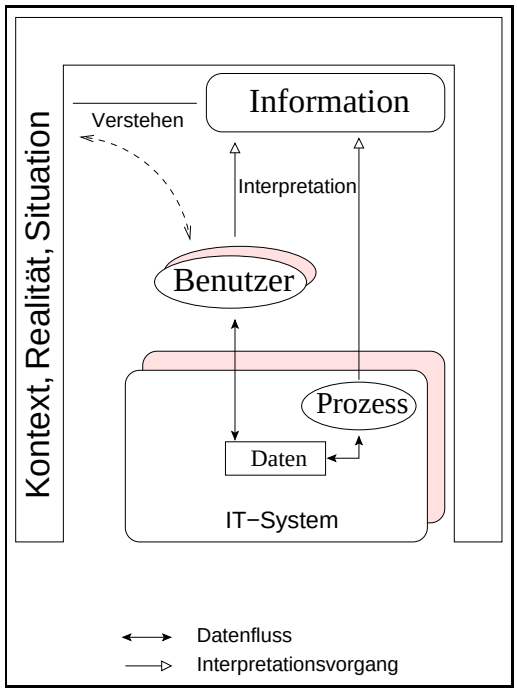

Abbildung 1.1: Darstellung der Dateninterpretation

Abbildung 1.1 veranschaulicht nochmals grob die Zusammenhänge zwischen den Daten, den Informationen und der dazwischenliegenden Interpretation. Zudem gibt es noch den gedanklichen Schritt des Verstehens, der die ermittelten Informationen in den Zusammenhang zur Realität setzt. Dieses Verstehen jedoch ist kaum mit den Mitteln der Informatik (Algorithmen, Prozesse, mathematische oder informelle Modelle usw.) näher zu beschreiben. Die Abbildung abstrahiert dabei von den genauen Vorgängen bei der Interpretation, die etwa durch formale Vorschriften oder Sematikmodelle beschrieben werden können.

Wie im Bild ersichtlich, kann die Interpretation nicht ausschließlich durch den Menschen, sondern auch durch andere Subjekte erfolgen, wie etwa Prozesse. Ein Prozess kann durch die Angabe von zusätzlicher Information einen Wert in einen bestimmten Zusammenhang stellen. Beispielsweise wird zu einer Zahl eine Währung und eine entsprechende Bedeutung angegeben, wie etwa der Begriff „Kontostand“ eine Zahl als Betrag eines Kontos ausweist. Der Zusammenhang und die Informationen daraus sind dann sofort klar. An diesem Beispiel auch zu erkennen, dass Veränderungen der Daten und auch der zusätzlichen Informationen zu einer völlig anderen Interpretation und somit zu einer falschen Information führen können.

Ein Ziel der Forschung über IT-Sicherheit muss also der Schutz vor den drei folgenden skizzierten Angriffsklassen sein.

1. *Schutz vor Manipulation der Daten:* Das Verändern der gespeicherten Daten zur

Darstellung neuer Informationen, wie etwa eines anderen Kontostandes im obigen Beispiel.

2. *Schutz vor unberechtigtem Zugriff auf die Daten:* Schon die reine Informationsgewinnung ohne die Veränderung der Daten stellt einen wichtigen Angriff dar.

3. *Sicherstellung der Verfügbarkeit der Daten:* Berechtigte Benutzer dürfen nicht am Zugang zu den Daten und damit den Informationen gehindert werden.

Dabei muss beachtet werden, dass diese drei Klassen nur eine grobe Einteilung der Schutzbedürfnisse vor bekannten Angriffen darstellt und dass die daraus erkennbaren Angriffe nicht nur alleine, sondern auch in Kombinationen auftreten können. Die genannten drei Schutzklassen sind für das Verständnis der allgemeinen Angriffsklassifizierung im folgenden Kapitel notwendig.

Die Vollständigkeit der drei Schutzklassen kann allerdings nicht bewiesen werden. Auch wenn zunächst alle naheliegenden Angriffe durch diese Schutzarten indirekt beschrieben werden können, muss diskutiert werden, ob nicht doch jederzeit eine neue Angriffsklasse oder ein konkreter neuartiger Angriff gefunden werden kann, der nicht in diese Einteilung fällt. Mittels einer einfachen Systematik anhand Abbildung 1.1 wird nun kurz versucht, einen neuartigen Angriff zu finden.

Dazu wird systematisch anhand der Grafik der Vorgang der Interpretation auf mögliche Schwachstellen hin untersucht. Die Manipulation oder Beobachtung der Daten und das Verhindern der Verfügbarkeit der Daten sind schon bekannte Angriffe. Der Vorgang der Interpretation der Daten für den eigentlichen Informationsgewinn kann jedoch auch angegriffen werden. Dazu könnte beispielsweise das Subjekt, dass die Interpretation vornimmt, behindert oder verfälscht werden. Es hängt einerseits von der Definition des Begriffes der Verfügbarkeit ab, ob ein solcher Angriff darunter fällt oder eine Art von neuem Angriff darstellt. Andererseits hängt es auch vom Grad der Abstraktion ab, wie allgemein sich Angriffsklassen fassen lassen und beispielsweise der genannte Fall noch abgedeckt wird.

1.1.3 Klassische Schutzziele

Für die folgenden Diskussionen werden die Begriffe Vertraulichkeit, Integrität und Verfügbarkeit derart bestimmt, dass alle bekannten Angriffe sich als Angriff auf eines dieser drei Schutzziele einteilen lässt. Eine exakte Definition aller Begriffe und die Begründung dieser Einteilung findet sich im Kapitel 2 wieder.

Ein gegebenes System erfüllt das Schutzziel der *Vertraulichkeit* genau dann, wenn keine unauthorisierte Informationsgewinnung ermöglicht wird.

In der Literatur wird manchmal auch zwischen Daten- und Informationsgewinnung unterschieden, was dann zur Definition der Datenvertraulichkeit und Informationsvertraulichkeit

führt. In dieser Arbeit ist mit dem Begriff der Vertraulichkeit immer die Vertraulichkeit der Informationen gemeint, wenn es nicht ausdrücklich anders angegeben ist. Die exakte Trennung zwischen dem Daten- und Informationsbegriff ist selbstverständlich nötig, wie in den vorausgegangenen Abschnitten bereits erläutert wurde. Bei der Betrachtung der Vertraulichkeit genügt es jedoch, stets die unauthorisierte Gewinnung der Informationen, im Sinne von interpretierten Daten zu beachten. Die unauthorisierte Gewinnung von Daten stellt im Allgemeinen keinen Angriff dar, wenn keinerlei Information über deren korrekte Interpretationen vorliegt. Das Abhören eines verschlüsselten Kanals etwa liefert stets die Daten einer schützenswerten Kommunikation. Ohne die richtige Interpretation, wie den Schlüssel zur Dechiffrierung der Informationen, sind die Daten im Allgemeinen jedoch für einen Angreifer wertlos.

Die *Integrität* eines Systems ist genau dann gewährleistet, wenn keine unauthorisierte Änderung, Zerstörung und der Verlust von Daten möglich ist.

Anders als bei der Definition der Vertraulichkeit wird hier explizit die Integrität der Daten, nicht der Informationen betrachtet. Die unauthorisierte Veränderung von Daten, selbst ohne Kenntnisse der darin enthaltenen Informationen, zerstören das Schutzziel der Integrität.

Das Schutzziel der *Verfügbarkeit* ist erfüllt, wenn jeder authorisierte Zugriff auf ein System oder eine Systemressource durchgeführt werden kann.

Diese Definitionen stützen sich einerseits auf das Buch [Eck02], in dem eine nützliche Trennung zwischen dem Begriff der Daten und der Informationen beibehalten wird, andererseits werden die Definitionen aus [Shi00] berücksichtigt. In Kapitel 2 werden die Begriffe für die weitere Entwicklung und Anwendung der hier vorgestellten Methodik genauer definiert.

Die drei Schutzziele beschreiben jeweils nur mögliche Schutzbedürfnisse, welche exakt dann verletzt werden, wenn ein Angriff auf ein solches Schutzziel durchgeführt wird. Daneben existieren noch weitere Schutzziele, wie beispielsweise die „Zurechenbarkeit“. Dadurch wird ausgedrückt, dass eine durchgeführte Aktion im nachhinein nicht abgestritten werden kann. Somit kann jede geschehene Aktion in einem System stets einem Subjekt zugeordnet werden, was beispielsweise für ein Bezahlsystem notwendig ist. Dieses Schutzziel unterscheidet sich jedoch von den drei oben definierten dadurch, dass es aus der Sicht des attackierten Systems erst im nachhinein angegriffen oder verletzt wird und letztlich ein Angriff auch immer durch eine Verletzung der Vertraulichkeit, Integrität oder Verfügbarkeit beschrieben werden kann. Anderes Beispiel ist das Schutzziel der „Anonymität“, die die nicht mögliche eindeutige Identifikation eines Subjektes beschreibt und eigentlich eine Umschreibung der Vertraulichkeit der privaten oder persönlichen Daten bezeichnet. Es gilt für alle bekannten Schutzziele, dass sie sich auf die drei klassischen Schutzziele abbilden lassen. Andere Schutzbedürfnisse der Benutzer lassen sich mit den Mitteln der Informatik nicht beschreiben und sind meist subjektiv einordenbar, dazu zählt beispielsweise der wichtige Begriff „Vertrauen“.

Der Sicherheitsbegriff

Für den Begriff der Sicherheit im Deutschen existieren in der Informatik im Englischen zwei unterschiedliche Bezeichnungen. Zum einen ist der ganze Bereich der Zuverlässigkeit, Ausfallsicherheit gemeint, der mit dem englischen Wort „Safety“ beschrieben wird, nähere Informationen finden sich beispielsweise in [Sto96]. Zum anderen wird das Gebiet des Datenschutzes und der Datensicherheit beschrieben, was im Englischen mit dem Begriff „Security“ bezeichnet wird. Je nach vorhandenem Wissensstand und der jeweiligen Betrachtungsweise sind diese beiden Bereiche mehr oder weniger voneinander zu trennen. In dieser Arbeit wird mit dem Sicherheitsbegriff immer der Bereich der „Security“ gemeint, wenn es nicht ausdrücklich anders beschrieben wird. Als deutscher Begriff wird dabei stets „IT-Sicherheit“ verwendet, der in der Literatur als Synonym zu „Security“ zu verstehen ist. Grundsätzlich sind diese beiden Forschungsbereiche heute noch weit voneinander entfernt, aber es sind etliche Konzepte von dem einen Bereich auf den anderen übertragbar, was auch später in dieser Arbeit noch sichtbar wird.

In den beiden Bereichen werden unterschiedliche Begriffe für sehr ähnliche Zusammenhänge verwendet. Ein Beispiel hierfür sind Monitoring-Systeme, die im Safety-Bereich die Überwachung zur Laufzeit bezeichnen. Etwas vergleichbares wird in der Security-Ecke Detektions-System genannt, wobei die konzeptionellen Aufgaben und Schwierigkeiten bei der Konzeption und Entwicklung sehr ähnlich sind. Ein anderes Beispiel wäre die Verwendung von Fehler- und Bedrohungsbäumen, auf die im Kapitel 5 noch näher eingegangen wird.

Generell gilt es, die funktionalen Eigenschaften der Systeme durch die Einhaltung von Sicherheitsanforderungen so wenig wie wöglich zu beeinflussen und zu stören. Allgemein sollte ein System daher wie in Abbildung 1.2 betrachtet werden. Die Sicherheit, sowohl im Sinne der Security, als auch der Safety, muss dann unter Berücksichtigung der eigentlichen Anforderungen an die Funktionalität, des Kontextes und der verarbeitenden Daten und Informationen in das System geeignet eingefügt werden. Aus diesem Blickwinkel heraus werden Sicherheitseigenschaften oft auch als Aspekte eines Systems verstanden. Ob allerdings das moderne, aspektorientierte Entwickeln von Systemen auch für die Realisierung von Sicherheitsanforderungen wirkungsvoll eingesetzt werden kann, ist heute ungeklärt und wird hier nicht weiter diskutiert. Nähere Informationen findet man beispielsweise in [VBC01].

Die Struktur der Probleme im Safety-Bereich unterscheidet sich von den untersuchten Problemen im Security-Bereich. Dies lässt sich am einfachsten anhand einfacher Skizzen der betrachteten Problematik darstellen.

Die beiden folgenden Skizzen stellen ein zu untersuchendes System dar, das aus verschiedenen Komponenten und gewisser Kommunikation dazwischen und nach außen besteht. Anhand dieser Ansicht kann man ganz allgemein die untersuchten Probleme in der Safety, als auch in der IT-Sicherheit darstellen.

Bei den Problematiken, die in der Safety untersucht werden und in Abbildung 1.3 (a)

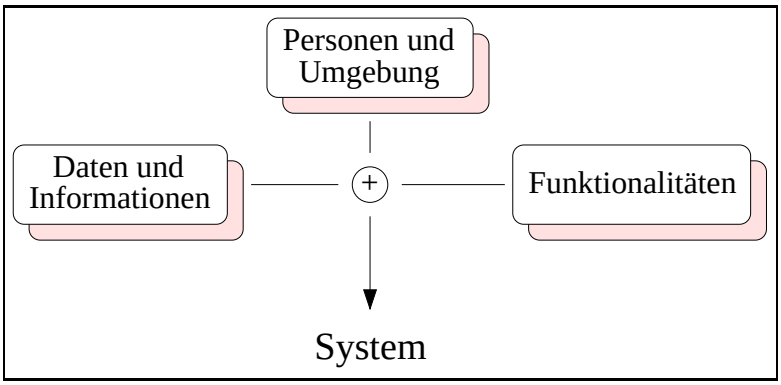

Abbildung 1.2: Einheiten eines informationsverarbeitenden Systems

allgemein dargestellt sind, werden immer die Gefahren gesucht und analysiert, die von dem bekannten, zu untersuchenden System selbst ausgehen. Im Wesentlichen geht es also dabei um Auswirkungen des IT-Systems auf andere Systemteile oder auch auf seine Umwelt. Dies wird in der Abbildung durch die Bedrohungspfeile verdeutlicht. Da man das IT-System, das letztlich der Verursacher der „Probleme“ ist, sehr genau kennt, kann man zumindest theoretisch alle möglichen Auswirkungen sowohl finden, als auch verhindern. Letztlich ist das ein Hauptgrund, warum man etwa mit formalen Methoden in diesem Bereich hohe Erfolge erzielen kann. Vollständig beschreiben lässt sich allerdings auch in diesem Bereich niemals alles, da zumindest Dinge wie Umwelteinflüsse usw. stets auf ein IT-System wirken und das Verhalten der Systeme beeinflussen.

Die Abbildung 1.3 (b) zeigt daneben das selbe zu untersuchende IT-System, das nun auf IT-Sicherheit hin untersucht werden soll. Dabei werden allerdings nicht die Gefahren analysiert, die vom System ausgehen, sondern die „Angriffe“[2] gesucht und analysiert, die auf das System einwirken können. Da man im Allgemeinen die Verursacher dieser Angriffe nicht kennen kann, ist die Quelle des Angriffs unbekannt. Man kann damit auf naheliegende Weise auch keine exakten Aussagen über die möglichen Angriffe auf das System treffen. Dies ist in der Abbildung auch daran zu erkennen, dass ein Pfeil den Angriff kennzeichnet, der keinen exakten Ursprung zu haben scheint. In Wirklichkeit existiert natürlich der Angreifer, er wird aber bei ernsten Angriffen erst bekannt, wenn der Angriff (meist erfolgreich) durchgeführt worden ist. In der Praxis kennt man die Auswirkungen dieser großen Problematik beispielsweise daher, dass Abwehrmaßnahmen gegen konkrete Computerviren erst nach einem Auftreten des Virus entwickelt und eingesetzt werden können. Oft wird der Unterschied zwischen den beiden Begriffen aber nicht richtig herausgestellt, wie auch in [HRU76] zu sehen ist.

[2]Die Begriffe werden im nächsten Kapitel ausführlich definiert.

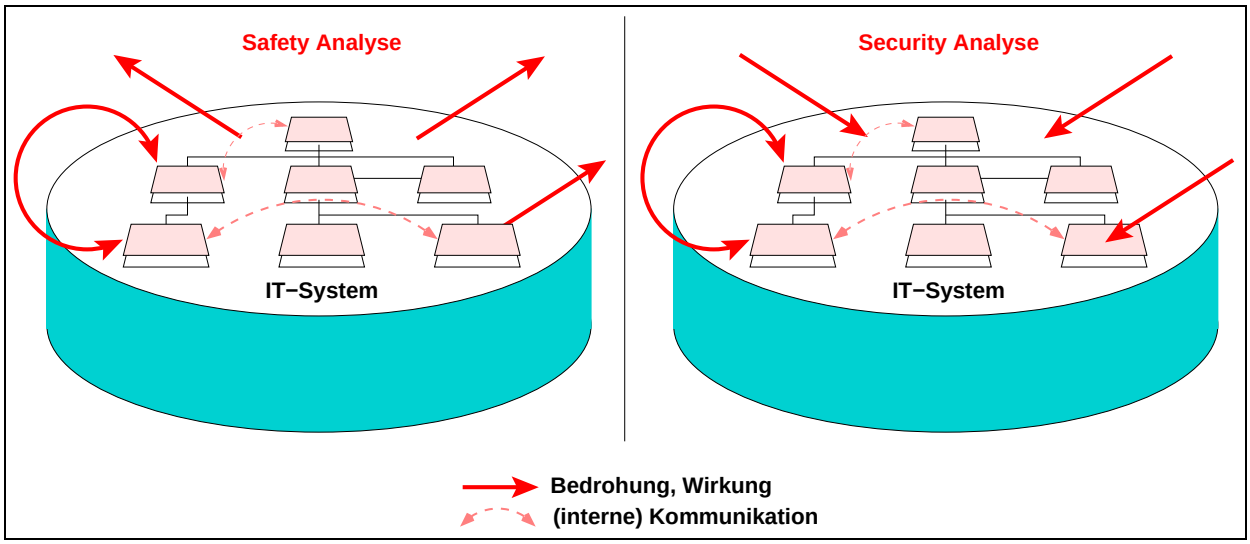

Abbildung 1.3: (a) Problemstellung bei Safety (b) und bei IT-Sicherheit

1.1.4 Hauptproblematik der IT-Sicherheit

Daraus kann eine wesentliche Folgerung für den allgemeinen Umgang mit Angriffen in der IT-Sicherheit gezogen werden. Der eigentliche Verursacher eines Angriffs wird im Allgemeinen erst hinterher bekannt, nachdem der Angriff erfolgreich durchgeführt worden ist. Es ist zudem unmöglich, einen Angreifer zu analysieren und präventive Gegenmaßnahmen gegen alle Arten von Angreifern zu installieren, wenn man den Angriffsverursacher überhaupt nicht kennt. Genau das ist die Hauptproblematik im IT-Sicherheitsbereich und führt letztlich dazu, dass exakte Methoden heute immer nur einen kleinen Teil der möglichen Angriffe erfassen und verhindern können. Formale Methoden für die Entwicklung und Analyse von IT-Systemen, die alle Arten der Gefährdungen erfassen können, wären zwar wünschenswert, wurden aber bis heute aus diesem Grund nicht gefunden. Grundsätzlich muss bezweifelt werden, dass solche Verfahren überhaupt möglich sein werden. Damit wird herausgestellt, dass die exakten Methoden an, aus heutiger Sicht, prinzipielle Grenzen stoßen und deshalb für die Lösung in dieser Arbeit nicht als alleinige Antwort dienen können.

1.1.5 Charakteristika der IT-Sicherheit

Bei der Betrachtung der Sicherheit von IT-Systemen ist es für Lösungen naheliegend, die Sicherheit im „täglichen Leben“ der Menschen zu betrachten. Dazu können Technik-nahe Bereiche analysiert werden, wie etwa der Straßen- und Luftverkehr, bei denen die technische Sicherheit eine sehr große Rolle spielt. Zudem können auch persönliche Schutzbedürfnisse und Verhalten der Menschen selbst analysiert werden und daraus versucht werden, Schlüsse auf ein geeignetes Verhalten von IT-Systemen zu ziehen.

Diese Sicherheitsbereiche weisen allerdings gewaltige Unterschiede auf, die nicht ohne Weiteres übersehen werden dürfen und führen oft dazu, dass ein direkter Vergleich zur IT-Sicherheit nicht mehr möglich ist. Das Besondere an IT-Sicherheit im Gegensatz zur Sicherheit in der „Natur“ sind folgende Tatsachen und werden auch in [Sch00] ausführlich besprochen.

Automatisierbarkeit

Ein Angriff kann automatisiert stattfinden, der eigentliche Verursacher muss zur Angriffszeit nicht beteiligt sein.

Durch Programmierung kann eine beliebige Zeitspanne zwischen dem Verursachen und dem eigentlichen Auftreten des Angriffs entstehen. Beispielsweise werden Viren erst einige Zeit (heute meist einige Tage oder Wochen) sich nur verbreiten und starten ihren Angriffscode erst zu einem festgelegten Datum, wenn das Virus genügend weit verbreitet wurde. Die Person, die als eigentlicher Verursacher das Virus programmiert hat, ist dann längst nicht mehr aktiv an den Angriffen beteiligt.

Unbestimmbarkeit von Ursprung und Angriffszeitpunkt

Ein Angriff kann jederzeit und von überall aus stattfinden.

Da für die Attacke technische Systeme verwendet werden, die miteinander vernetzt sind, kann ein Angriff von jeder Stelle dieses Netzwerks ausgeführt werden. Da Netze wie etwa das Internet weltweit arbeiten, kann jeder Angriff praktisch von jeder Stelle der Erde aus erfolgen. Technisch gesehen ist der Zugriff zweier Rechner, die in einem Raum stehen, der selbe, wie ein Zugriff zwischen Rechnern, die in unterschiedlichen Ländern stehen. Es ist im Allgemeinen nicht möglich, den physikalischen Standort eines Rechners durch Überwachung seiner Kommunikation festzustellen. Weil Angriffe automatisiert durchgeführt werden können, kann ein Angriff praktisch jederzeit erfolgen und muss nicht direkt von dem System aus stattfinden, an dem der eigentliche Angreifer (die Person) sich befindet. Zudem kann der Angriff derart programmiert sein, dass er zeitverzögert zu der eigentlichen Arbeit des Angreifers auftritt.

Wiederholbarkeit

Ein Angriff kann praktisch beliebig oft wiederholt werden.

Physikalische Güter werden bei einem Angriff auf die IT-Sicherheit selten verbraucht, deshalb ist ein Angriff immer ohne Verluste von Gütern beliebig oft wiederholbar. Allerdings werden die Angriffsziele meist nach dem Bekanntwerden von Angriffen verändert oder entfernt, so dass ein Angriff vermieden werden kann.

Verlust der Originalität

Die „Güter“, die die Ziele darstellen (Informationen usw.) sind verlustfrei und ohne späteren Nachweis kopierbar.

Die Daten, die ein Angreifer erhält, sind stets beliebig oft kopierbar und können deshalb an beliebig vielen Orten aufbewahrt und auch an beliebig viele Personen weitergegeben werden.

Diese Eigenschaften der IT-Sicherheitsprobleme ändern zumindest die Gewichtung der einsetzbaren Schutzmechanismen stark ab. Beispielsweise geht man im täglichen Leben immer davon aus, dass der Täter bei einem Diebstahl direkt vor Ort gefasst werden kann, zu dem Zeitpunkt an dem er den Diebstahl ausführt. Ein Angriff auf Computersysteme kann zu einem Zeitpunkt von praktisch überall aus erfolgen, im Allgemeinen ist der Angreifer nicht direkt am Tatort und könnte dort auch nie gefasst werden. Die Schutzmechanismen, präventiv wie auch reaktiv (also unmittelbar nach dem Angriff) sind deshalb auch anders auszulegen, wie etwa in [Sto88] diskutiert wird.

1.1.6 Entwicklung abgesicherter Systeme

Der Entwicklungsprozess für abgesicherte informationsverarbeitende Systeme lässt sich wie in Abbildung 1.4 einteilen und wird in der Literatur oft mit dem Begriff „Security Engineering“[3] bezeichnet, wie etwa in [And01]. Der Begriff der Absicherung von Systemen bedeutet, dass funktonaler beziehungsweise organisatorischer Aufwand zur Einhaltung der Schutzziele betrieben wird und wird später in Kapitel 2 definiert.

Im Unterschied zu vielen Quellen in der Literatur über derartige Entwicklungsprozesse, wie etwa in [Eck02], wird in dieser Arbeit Wert darauf gelegt, dass neben der Neuentwicklung sicherer Software auch bestehende Software einen möglichen Ausgangspunkt bei der Entwicklung darstellt. Selbstverständlich bietet eine Neuentwicklung mit Berücksichtigung geforderter Sicherheitseigenschaften den besten Ausgangspunkt zur Erhaltung der Schutzziele. Es ist jedoch kein Garant für ein sicheres System, so dass die oftmals praktikablere und meist billigere Methode eingesetzt wird, Systeme im Nachhinein abzusichern. Es ist für ein allgemeines System nicht möglich, bestimmte Sicherheitseigenschaften zu finden, die nur bei einer Neuentwicklung eingehalten werden können. Zudem sind keine Sicherheitsmechanismen bekannt, die sich nicht auch nachträglich in ein System integrieren ließen. Die Höhe des Aufwand allerdings kann je nach Mechanismus erheblich unterschiedlich sein, da im Einzelfall die Integration sehr aufwändig werden kann und eine Umstrukturierung notwendig wird. Dies läßt sich aber nicht allgemein, sondern nur im konkreten Fall zeigen.

Wie in der Abbildung 1.4 gezeigt, kann das Systemmodell, das als funktionale Basis des Security Engineering dient, sowohl aus einer Analyse der Anforderungen, als auch aus der

[3] Der Begriff ist analog zum Begriff des „Software Engineering“ zu verstehen.

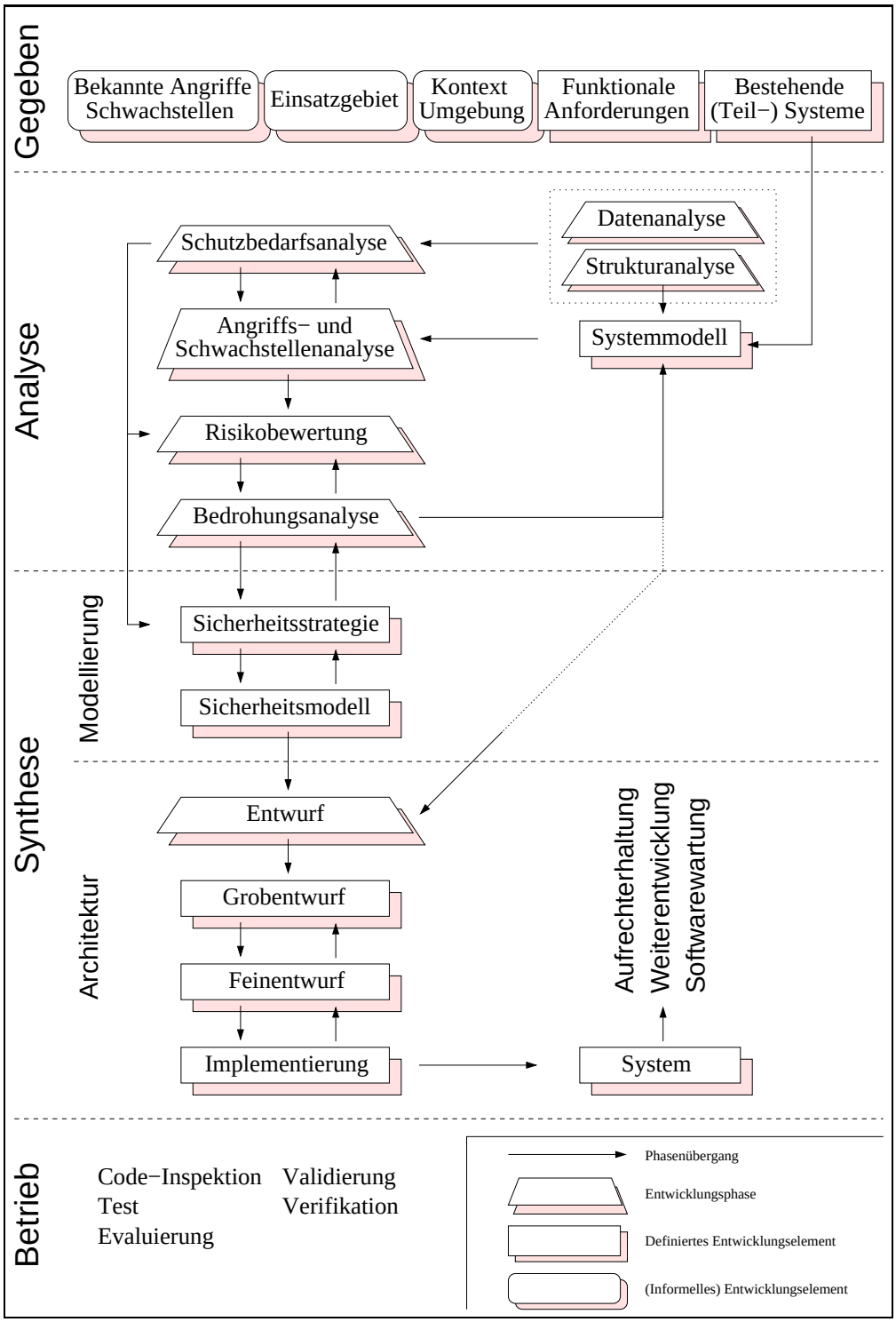

Abbildung 1.4: Der Entwicklungsprozess des Security Engineering

Analyse bestehender Teilsysteme entwickelt werden. Es gibt im Wesentlichen zwei Gründe für die Verwendung bestehender Systeme im Security Engineering. Erstens können dadurch Fehler, insbesondere Sicherheitsschwachstellen, vermieden werden, wenn sie in ähnlicher Art bereits aus anderen Systemen bekannt wurden. Zweitens existieren heute unzählige Softwaresysteme, die im Security Engineering auch berücksichtigt werden müssen und zunächst ohne besondere Rücksicht auf die Sicherheit entwickelt worden sind. In dieser Arbeit spielen Betriebssysteme eine zentrale Rolle. Diese wichtigen Systembestandteile sind große, komplexe Softwaregebilde, die sich besonders dadurch auszeichnen, dass sie zentraler Vermittler zwischen den Anwendungen und der Hardware sind. Standardbetriebssysteme gelten heute als unsicher, was sich auch durch die immer häufigeren Meldungen von erfolgreichen Einbrüchen von Viren und Würmern bestätigen lässt. Neben diesen Angriffen gibt es natürlich noch eine Vielzahl anderer, unbekannterer erfolgreicher Angriffe. Diese Betriebssysteme und darauf angepasste Hardware und Software werden heute überall eingesetzt. Es ist nur mit riesigem Aufwand möglich, diese Systeme durch komplette Neuentwicklungen zu ersetzen, die mehr Sicherheit versprechen. Wichtig ist demnach, dass neben einer vollständigen Neuentwicklung von Software auch die Einbeziehung bestehender Systemteile beim Security Engineering berücksichtigt wird.

1.2 Ziele

Ziel der Arbeit war die Entwicklung eines Betriebssystems für mobile Endgeräte. Da einerseits die komplette Neuentwicklung zu aufwändig wäre und andererseits viele Basismechanismen bestehender Betriebssysteme existieren und für ein abgesichertes Gesamtsystem sinnvoll weiterverwendet werden können, werden in dieser Arbeit wichtige Teilsysteme entwickelt und in ein bestehendes Betriebssystem integriert. Daraus ergibt sich, dass sowohl neue Teile entwickelt, als auch bestehende abgehärtet werden müssen, wie es nach dem oben dargestellten Entwicklungsprozess möglich ist.

Dazu werden vor allem bestehende Systematiken für die Analyseteile des Entwicklungsprozesses betrachtet und diskutiert und eine neue entwickelt. Ziel dabei war es, die Ergebnisse der einzelnen Analysephasen möglichst verständlich und nachvollziehbar darzustellen, damit sowohl schnell auf neue Angriffe reagiert werden kann, als auch Vertrauen in das System und seine Sicherheitsmechanismen entstehen kann. Besonderer Schwerpunkt in dieser Arbeit liegt auf der Systematik der Bedrohungsanalyse, die in bisherigen Ansätzen weder ausreichend nachvollziehbar war, noch deren Ergebnisse exakt dokumentiert waren. Eine bekannte Diskussion um den formal nicht fassbaren Begriff Vertrauen findet sich in [Tho84].

Für den hier entwickelten Lösungsweg ist die modellhafte Sicht auf das IT-System und die systematische Suche nach möglichen Bedrohungen notwendig. Man kann über den Angreifer als eigentliche Ursache nur insoweit Aussagen treffen, die man aufgrund bereits durchgeführter Angriffe kennt. Deshalb werden in dieser Arbeit Aussagen über die möglichen Angriffspunkte des Systems getroffen. Diese Aussagen kann man nur durch Kenntnisse

des Zielsystems machen, ohne den Angreifer zu kennen.

Der entwickelte Lösungsweg, der in dieser Arbeit beschrieben wird, stellt eine zielgerichtete und systematische Analysemethode bei der Entwicklung abgesicherter IT-Systeme dar. Grundsätzliches Problem dabei ist, dass die Angreifer unbekannt sind und deren mögliche Angriffe zunächst nicht vorliegen. Allerdings kann man durch eine Sicht auf das System die möglichen Angriffspunkte ermitteln und systematisch deren mögliche Gefährdung analysieren. Dazu werden in Kapitel 2 zunächst Klassen bekannter Angriffe und Schwachstellen beschrieben. Das Systemmodell, das in Kapitel 6 entwickelt wird, wird dann systematisch auf diese möglichen Angriffs- und Schwachstellenmuster hin untersucht. Dabei können sowohl bestehende Systeme, als auch Systeme entwicklungsbegleitend auf eventuelle Schwachstellen hin analysiert werden, und man erhält eine nachvollziehbare, später anpassbare Darstellung der möglichen Bedrohungen auf das IT-System. Die entwickelte Systematik wird in Kapitel 5 dargestellt.

Wurden die möglichen Bedrohungen analysiert, so werden genau die bestehenden Methoden und Mechanismen für die Absicherung von Systemen als Sicherheitsbausteine integriert, die die gefundenen Bedrohungen abwehren. Diese Bausteine werden in Kapitel 2 aufgezeigt und bieten bereits einen hohen Schutz gegen bestimmte Arten von Bedrohungen und Angriffen. Diese Arbeit bietet eine Methode, die eine Kombination dieser Verfahren darstellt und durch die geeignete Integration ein sehr hohes Maß an Sicherheit für ein System liefert. Dies kann sowohl bei Systemen bei der Entwicklung, als auch bei bestehenden, zu verbessernden Systemen angewendet werden.

Im Folgenden werden die einzelnen Aspekte des Themas im Hinblick auf den Stand der Technik im IT-Sicherheitsbereich betrachtet. Dabei ist zu beachten, dass der Fokus dieser Arbeit nicht die Entwicklung von neuen Sicherheitsmechanismen ist. Stattdessen wird die oben angesprochene Systematik entwickelt, mit der analysierte, bestehende Mechanismen und Konzepte wirkungsvoll in ein Gesamtsystem integriert werden. Die folgenden vier Abschnitte beschreiben die Eckpunkte dieser Systematik, die Details folgen in späteren Kapiteln. Die bearbeiteten Themen werden nach dem Zeitaufwand für diese Arbeit in einem Übersichtsgraphen dargestellt. Diese grobe, quantitative Wertung hilft als Überblick über die unterschiedlichen Teilthemen.

1.2.1 Themengraph

In dem Übersichtsgraphen in Abbildung 1.5 werden die bearbeiteten Gebiete einerseits und jeweils die Tiefe der Bearbeitung für diese Arbeit andererseits aufgetragen. Die Darstellung gibt einen ungefähren Überblick über die durchgeführten Arbeiten, eine Maßeinheit für den Beitrag kann nicht genau festgelegt werden. Hier finden sich die vier im Folgenden diskutierten Teilgebiete wieder, zu denen diese Arbeit einen Beitrag leistet und die auch mit bearbeiteten Unterbereichen differenziert dargestellt werden. Damit wird ein möglichst anschaulicher Überblick über die Bearbeitung des gesamten Themas gegeben.

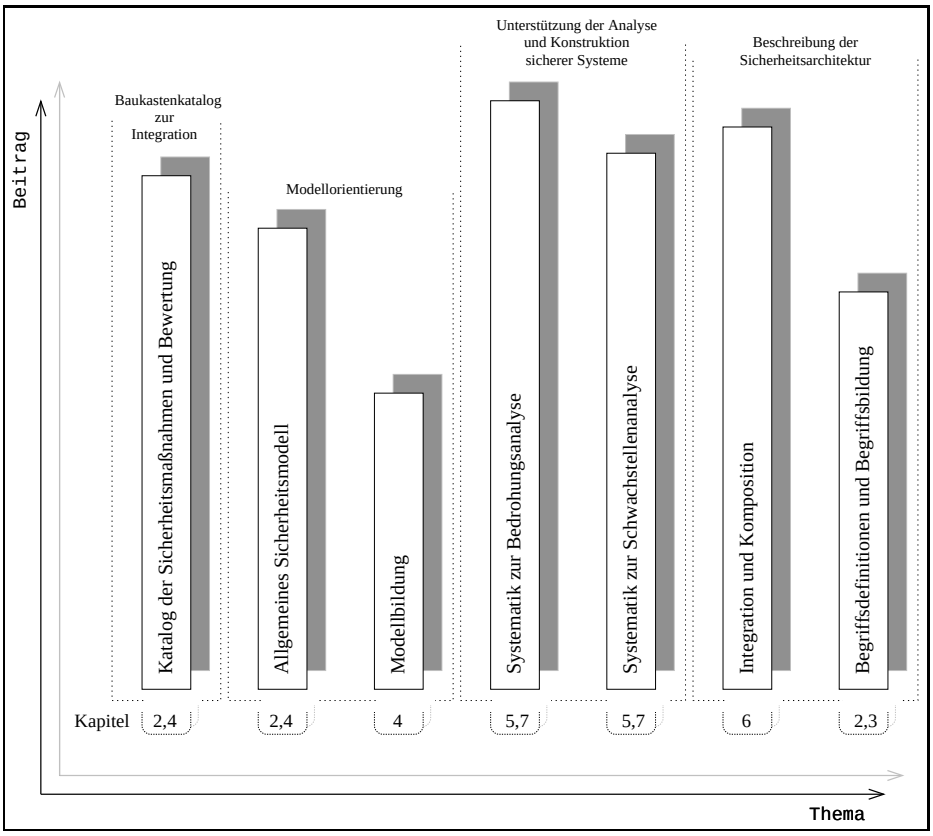

Abbildung 1.5: Darstellung der Bearbeitungstiefe der einzelnen Themen

Baukastenkatalog zur Integration

Sicherheitsmechanismen gibt es bereits für das Erreichen unterschiedlicher Schutzziele und Schutzbedürfnisse. Allerdings werden diese Mechanismen in der Forschung meistens streng voneinander getrennt betrachtet und stets jeweils für genau ein Schutzziel entwickelt. Die Kryptographie bespielsweise liefert sehr fundierte Möglichkeiten, die Schutzziele Vertraulichkeit oder Integrität durchzusetzen. Es gibt dabei unterschiedlichste Kryptoverfahren, die eingesetzt werden können. Soll aber etwa die Anonymität gesichert werden, oder ein Schutz vor Angriffen durch Informationsflussanalysen, dann genügt der einfache Einsatz noch so guter Kryptomechanismen nicht mehr. In den veröffentlichten Arbeiten wird dann meist der Raum der erreichbaren Schutzziele eingeschränkt, ohne Hinweis darauf, dass damit nicht automatisch ein sicheres Gesamtsystem entsteht, sondern allenfalls einige Angriffe abgewehrt werden. Als Beispiel sei hier [MvOV96] als Standardwerk der aktuellen Kryptographie erwähnt, in dem nicht zwischen den Zielen und Möglichkeiten der Kryptographie und der Konstruktion eines sicheren Gesamtsystems unterschieden wird. Damit kann leicht der Eindruck entstehen, dass gute Kryptoverfahren auch automatisch sichere Systeme liefern.

Hauptziel dieser Arbeit ist es daher, die bestehenden Sicherheitsmechanismen gemeinsam zu betrachten und für die Konstruktion eines sicheren Betriebssystems für mobile Endgeräte zu integrieren. Dafür wurde ein Baukasten der unterschiedlichen Verfahren erstellt und jeweils Wirkung und mögliche Schutzziele beschrieben. Durch Wiederholung dieser Analyse- und Syntheseschritte können die Probleme, die durch das Zusammenwirken der integrierten Sicherheitsmechanismen entstehen können, wiederum erkannt und beseitigt werden.

Modellorientierung

Grundlage für die Betrachtung sowohl der Sicherheitseigenschaften und Mechanismen, als auch der funktionalen Eigenschaften der Systeme, sind in dieser Arbeit Modelle. Ein Modell als Abbild des „realen“ Systems ermöglicht die einfache Untersuchung der Strukturen und Abläufe. Ziel dieser Arbeit ist es, ein Modell für sichere Systeme zu entwickeln, mit dem die Zusammenhänge und Wirkungen zwischen den unterschiedlichen Sicherheitsmechanismen darstellbar sind. An diesem Mechanismenmodell sind sowohl die einzelnen Phasen der möglichen Verteidigung, als auch die Angriffe selbst wiederzufinden.

Für die systematische Bedrohungsanalyse wird ebenfalls ein Modell des untersuchten Systems entwickelt, an dem die schrittweise Analyse und das Finden der möglichen Bedrohungen durchgeführt wird. Modelle der richtigen Darstellungsform bieten dafür die geeignete Grundlage. Zu unterscheiden ist dabei einerseits zwischen dem Systemmodell für die systematische Bedrohungsanalyse und dem Bedrohungsmodell andererseits. Für das Systemmodell wird eine Beschreibungssprache ausgewählt, die für das Zielsystem der Analyse passend erscheint. Durch Anpassungen des vorgestellten Verfahrens sind jedoch auch andere Modellbeschreibungssprachen denkbar. Das Bedrohungsmodell dagegen ist ein Teilergebnis der Analyse, die ja dem Zweck dient, neue Bedrohungen zu finden, die auf ein System wirken können.

Unterstützung der Analyse und Konstruktion sicherer Systeme

Ein Ziel dieser Arbeit ist es, das bekannte Vorgehen der Bedrohungsanalysen zu verbessern und dafür eine geeignete Systematik zu entwickeln. Das Ergebnis dieser Analysen wird meistens als Eingabe für die Bewertung der Risiken eingesetzt und dient so schließlich als die eigentliche Grundlage für die gewählte Sicherheitsarchitektur.

Bisher wurden diese systemweiten Analysen entweder auf Basis von Programm-Quellcode oder auf unsystematisches Anwenden von bekannten Angriffen auf das System durchgeführt. Die hier vorgestellte Systematik liefert einen Weg, der am Modell die möglichen Angriffe schrittweise finden lässt. Dadurch kann zwar keine Vollständigkeit über alle möglichen Angriffe garantiert werden, es unterstützt allerdings bei der Analyse derart, dass man auch neue, nicht triviale Angriffe finden kann.

Das Vorgehen bei der systematischen Analyse ist sowohl für die Betrachtung von bereits bestehenden Systemen nützlich, als auch bei der Entwicklung und Konstruktion neuer Systeme einsetzbar. Allerdings muss an dieser Stelle betont werden, dass ohne ein solides Grundverständnis über IT-Sicherheit kein sicheres System solide entwickelt werden kann, egal, auf welche Art auch immer die Sicherheit integriert wird.

Beschreibung der Sicherheitsarchitektur

Basierend auf den Analyseergebnissen, dem Bedrohungsmodell und den entsprechenden Risiken wird eine geeignete Sicherheitsarchitektur für das System entwickelt. Diese Architektur beinhaltet nur wichtige Funktionen des Systems und legt den Fokus auf die Sicherheitsmechanismen und Verfahren zum Erreichen der Schutzziele. Der Vorteil der hier vorgestellten Beschreibungen ist zum einen, dass sie neben der Entwicklung direkt eingesetzt und verwendet werden können. Die Sicherheitsanalyse kann parallel zur Entwicklung stattfinden und die Ergebnisse können direkten Einfluss auf das System haben. Zum anderen kann die Sicherheitsarchitektur getrennt vom System analysiert werden.

1.3 Struktur und Aufbau

Im Folgenden werden Struktur und Aufbau der einzelnen Teile und Kapitel dieser Arbeit erläutert.

1.3.1 Grobstruktur

Die gesamte Arbeit lässt sich in drei große Abschnitte einteilen. Teil I „Analyse“ umfasst Kapitel 2–5. Er enthält die notwendigen Definitionen der Begriffe und beschreibt, welche Bausteine für die Konstruktion sicherer Systeme existieren. Neben einem allgemeinen Modell zur IT-Sicherheit liefern diese Kapitel einen Vergleich bekannter abgesicherter Betriebssysteme, sowohl aus dem klassischen, als auch dem mobilen Bereich. Der Teil schließt mit einem erarbeiteten Konzept zur systematischen Sicherheitsanalyse, das durch die Analyse anderer Verfahren in dieser Arbeit entwickelt wird.

In dem zweiten Teil „Synthese“ mit Kapitel 6 und 7 wird zunächst ein Verfahren zur systematischen Sicherheitsanalyse entwickelt und detailiert erklärt. Dabei werden alle bekannten Teilanalysen, wie Schwachstellen-, Angriffs- und Bedrohungsanalyse der vorherigen Kapitel integriert. Anschließend werden die dafür entwickelten Werkzeuge vorgestellt.

Im dritten Teil „Umsetzung“ der Ergebnisse dieser Arbeit wird anhand konkreter Beispiele der Einsatz der entwickelten Konzepte und Werkzeuge beschrieben. Dazu werden Teilsysteme des entwickelten abgesicherten Betriebssystems für mobile Endgeräte analysiert.

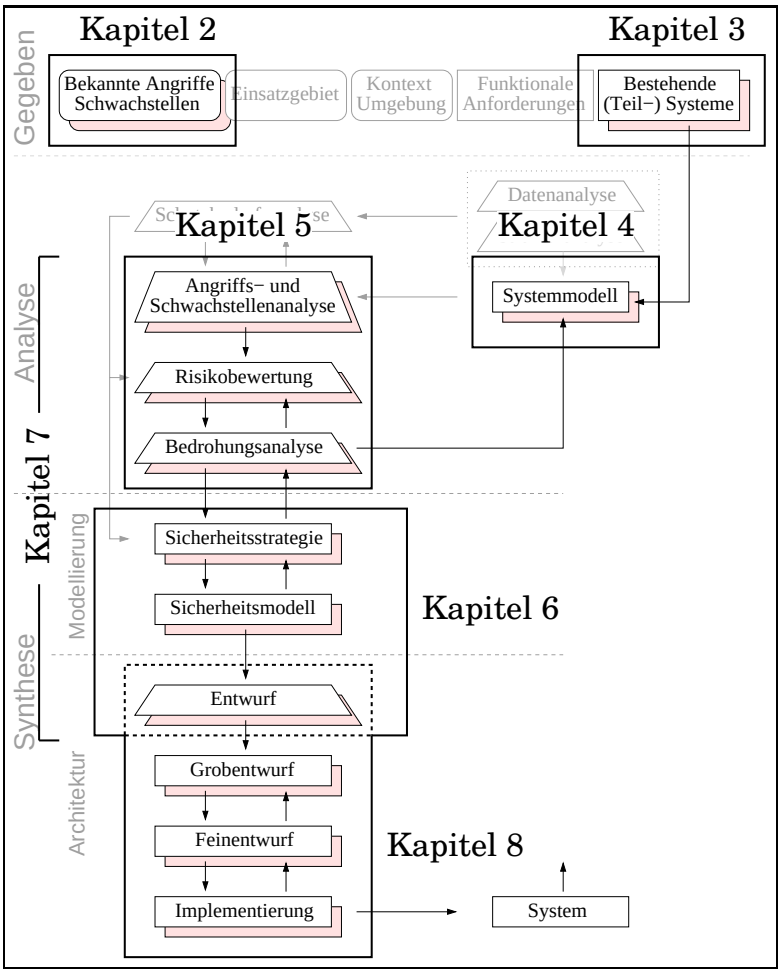

Abbildung 1.6: Einordnung der Kapitel in den Entwicklungsprozess

Anhand der Abbildung 1.6 kann man die Kapitelstruktur nochmals in Bezug zum Entwicklungsprozess verfolgen. Die Abbildung richtet sich nach der Darstellung des Entwicklungsprozesses aus Abbildung 1.4.

1.3.2 Feinstruktur

Kapitel 2 liefert die für die Arbeit notwendigen Begriffe unter Berücksichtigung der verschiedenen Definitionen in der Literatur. Um das komplexe Thema für die später entwickelte Lösung zu strukturieren werden zunächst konkrete und bekannte Schwachstellen, Angriffe und Bedrohungen untersucht. Auf dieser Basis werden anschließend Klassen von Bedrohungen und Angriffen gebildet und eine sinnvolle Phaseneinteilung der Angriffsdurchführung angegeben. Neben einer Übersicht über die Breite der möglichen Abwehrmaßnahmen werden auch die Gegenmaßnahmen klassifiziert.

Ziel des Kapitel 3 ist ein Überblick über bestehende Betriebssysteme zu geben, die einen besonderen Sicherheitsaspekt berücksichtigen. Dazu werden die wesentlichen Archtekturprinzipien für Betriebssysteme besprochen, die Sicherheitseigenschaften bieten oder beeinflussen. Beispiele hierfür wären Kernarchitekturen und Mechanismen für Speicherschutz. Anschließend werden bestehende abgesicherte klassische Betriebssysteme vorgestellt. Deren Entwicklung liegt bereits teilweise einige Jahrzehnte zurück, bieten aber dennoch Konzepte, die heute noch oder wieder aktuell sind. Abschließend werden abgesicherte Betriebssysteme für mobile Endgeräte besprochen, die allerdings aus Indurstrieentwicklungen stammen und im Wesentlichen von den Archtiekturen klassischer Systeme abgeleitet sind.

Kapitel 4 liefert eine Diskussion über mögliche Modellierungsmethoden, die zur Systembeschreibung für die Sicherheitsanalyse verwendet werden können. Es werden nur einige bekannte Modellierungen besprochen, die charakteristisch für unterschiedliche Modellierungsarten sind, dennoch gibt es sehr viele weitere Methoden, die hier nicht berücksichtigt werden. Die am Ende des Kapitels ausgewählte Methode liefert eine brauchbare Möglichkeit für das später entwickelte Verfahren, könnte aber auch durch andere ersetzt werden.

In Kapitel 5 werden zunächst die möglichen Analysetechniken für die verschiedenen Aspekte Schwachstellen, Angriffe und Bedrohungen beschrieben und deren prinzipiellen Möglichkeiten und Grenzen aufgezeigt. Anschließend werden einzelne bestehende Verfahren für jede dieser Analysen verglichen und bewertet.

Kapitel 6 stellt die in dieser Arbeit entwickelten Methode zur systematischen Sicherheitsanalyse vor. Ausgehend von einer Beschreibung der allgemeinen Ziele der Analyse wird die systematische Suche von Schwachstellen anhand von Modellen und an Quelltexten beschrieben. Das Auffinden möglicher Angriffe wird danach durch allgemeine Angriffsmuster und die beschriebenen Phasen von Angriffen beschrieben. Zur Integration der Analyseergebnisse und Darstellung der Sicherheitseigenschaften des analysierten Systems werden spezielle Strukturen entwickelt. Diese T_{AV}-Bäume werden anschließend diskutiert. Abschließend wird das Vorgehen bei der Sicherheitsanalyse zur Konstruktion der T_{AV}-Bäume beschrieben, die durch eine Risikoanalyse bewertet und als Basis zur Suche entsprechender Gegenmaßnahmen herangezogen werden.

Die Beschreibung der Werkzeuge, die für die beschriebene Methodik entwickelt wurden, folgt in Kapitel 7. Nach der Beschreibung der allgemeinen Anforderung wird zunächst das Tool zur Generierung von Aufrufgraphen aus gegebenen Quelltexten vorgestellt. Anschließend wird das Werkzeug zur Erzeugung, Verwaltung und Berechnung von T_{AV}-Bäumen beschrieben. Schließlich wird ein Werkzeug zur modellhaften Analyse beliebiger Sprachen diskutiert.

Die Analysemethode und die Werkzeuge werden in Kapitel 8 eingesetzt. Wichtige Systemteile eines entwickelten Betriebssystems dienen dabei als Ziel zur Analyse. Unter anderem werden der abgesicherte Startvorgang, die Erweiterbarkeit der Kerns oder ein gesicherter persistenter Speicher analysiert.

Teil I

Analyse

Kapitel 2

IT-Sicherheit

In diesem Kapitel werden zunächst wesentliche Begriffe aus dem Bereich der IT-Sicherheit definiert, die in der Literatur ungenau, manchmal sogar widersprüchlich verwendet werden. Anschließend werden bekannte Angriffe und Bedrohungen dargestellt, wobei ein generisches Angriffsmuster entwickelt wird, auf das sich alle bekannten Angriffe abbilden lassen. Neben der konkreten Darstellung von Angriffen werden Klassen von Angriffen und Bedrohungen entwickelt. Dadurch können sowohl Angriffe als auch Bedrohungen allgemein dargestellt werden. Darüber hinaus entsteht die Möglichkeit, über Angriffe und Bedrohung zu diskutieren, die weder bereits aufgetreten, noch genau bekannt sind. Es werden bekannte Sicherheitsmechanismen beschrieben, die erfolgreich gegen bestimmte Klassen von Angriffen wirksam sind. Anhand der entwickelten Phasen eines Angriffs werden die beschriebenen Gegenmaßnahmen ebenso klassifiziert, so dass schließlich ein Baukasten entwickelt werden kann, der für bestimmte Angriffe Abwehrmaßnahmen anbietet.

2.1 Begriffe und Definitionen

Schützenswerte Güter innerhalb eines IT-Systems sind Daten und Informationen. Vorgänge und Prozesse, im Sinne von Abläufen, sind keine schützenswerten Güter. Obwohl die Entwicklung von Prozessen aufwändig ist und sie damit einen gewissen Wert darstellen, werden in dieser Arbeit nur die Einheiten als schützenswert angesehen, die verarbeitet werden, die Daten und Informationen.

Definition 2.1 (Daten und Repräsentation)
Daten *bilden die Verarbeitungseinheiten, die in Objekten und Subjekten sowohl gespeichert als auch von beiden verarbeitet werden. Die Art der Speicherung und Darstellung der Daten innerhalb des Rechensystems nennt man* Repräsentation.

□

Wie bereits im ersten Kapitel erklärt wurde, besteht ein Schutzbedürfnis nicht nur gegenüber den Daten, sondern vor allem auch gegenüber Information.

Definition 2.2 (Information)
Die Information *ist das Ergebnis der Interpretation gespeicherter oder auch verarbeiteter Daten.*

□

Das Durchsetzen der Schutzbedürfnisse und deren detailierte Beschreibung wird durch sogenannte Sicherheits-Regeln festgelegt, die Policies genannt werden, wie sie etwa in [GM82] beschrieben werden. Das Durchsetzen der einzelnen Policies geschieht mit speziellen Sicherheits-Mechanismen, was zu folgenden Definitionen führt.

Definition 2.3 (Sicherheits-Policy)
Eine Sicherheits-Policy *legt fest, was erlaubt ist und was nicht und kann organisatorische Maßnahmen festlegen.*

□

Durch diese Definition sind die oft auftauchenden Diskussionen über das häufig falsch verwendete Schlagwort Sicherheits-Policy hinfällig, näheres findet sich in [Ste91]. Meistens werden als organisatorische Maßnahmen die Reaktionen auf mögliche Angriffe festgelegt, da diese nicht automatisiert erfolgen können.

Definition 2.4 (Sicherheits-Mechanismus)
Der Sicherheits-Mechanismus *ist ein Hilfsmittel oder eine Methode, welche Sicherheits-Politiken durchsetzt.*

□

Nach [Bis03] werden Sicherheits-Mechanismen wie folgt in ihrer Exaktheit klassifiziert. Es sei $\mathcal{Z}$ die Menge aller möglichen Zustände eines Systems, $\mathcal{Z}_\mathcal{S}$ die Menge aller sicheren Zustände, wie sie von den Sicherheits-Policies beschrieben werden. Durch die eingesetzten Sicherheitsmechanismen kann das System nur die Zustände $\mathcal{Z}_\mathcal{R}$ annehmen, wobei gilt, dass $\mathcal{Z}_\mathcal{R} \subseteq \mathcal{Z}$ ist. Zur genauen Unterscheidung der Güte eines Sicherheitsmechanismus werden diese wie folgt klassifiziert.

Definition 2.5 (Klassifizierung eines Sicherheits-Mechanismus)
Man nennt einen Sicherheits-Mechanismus `sicher`*, wenn* $\mathcal{Z}_\mathcal{R} \subseteq \mathcal{Z}_\mathcal{S}$ *gilt; er heißt* `exakt`*, wenn* $\mathcal{Z}_\mathcal{R} = \mathcal{Z}_\mathcal{S}$*; und man nennt ihn* `unscharf`*, wenn mindestens ein Zustand* $r \in \mathcal{Z}_\mathcal{R}$ *existiert, mit* $r \notin \mathcal{Z}_\mathcal{S}$*.*

□

In unterschiedlichen Anwendungen der Sicherheit werden die Sicherheitsbedürfnisse durch *Schutzziele* beschrieben und festgelegt. Sowohl deren Interpretation als auch der Kontext, in dem sie verwendet werden, können sehr unterschiedlich sein, so dass eine exakte Definition sinnvoll ist.

Definition 2.6 (Schutzziel)
Ein Schutzziel *beschreibt eine Eigenschaft des Systems, die in allen erreichbaren Zuständen $\mathcal{Z}_\mathcal{R}$ aufgrund der Sicherheits-Policy $\mathcal{P}$ gelten soll.*

□

Gegeben sei ein beiliebiges zeitdiskretes System $\mathcal{S}$ mit den möglichen Zuständen $\mathcal{Z}$, einer Menge von Subjekten $\mathcal{A}$, den Informationen $\mathcal{I}$ und einer beliebigen Sicherheits-Policy $\mathcal{P}$, die durch einen exakten Sicherheits-Mechanismus durchgesetzt wird.

Definition 2.7 (Vertraulichkeit)
Wir sagen, dass das gegebene System $\mathcal{S}$ das Schutzziel der Vertraulichkeit *(engl. Confidentiality) genau dann erfüllt, wenn für alle Zustände $\mathcal{Z}$ gilt, dass ein $\mathcal{I}$ nur dann einem $\mathcal{A}$ zur Verfügung steht, wenn $\mathcal{P}$ es erlaubt.*

□

Definition 2.8 (Integrität)
Die Integrität *(engl. Integrity) eines Systems $\mathcal{S}$ ist genau dann gewährleistet, wenn für alle Zustände $\mathcal{Z}$ gilt, dass jedes $\mathcal{I}$ nur genau dann verändert werden kann, wenn $\mathcal{P}$ es erlaubt.*

□

Definition 2.9 (Verfügbarkeit)
Das Schutzziel der Verfügbarkeit *(engl. Availability) eines Systems $\mathcal{S}$ ist genau dann erfüllt, wenn für alle Zustände $\mathcal{Z}$ gilt, dass ein Zugriff von einem beliebigen $\mathcal{A}$ auf ein $\mathcal{I}$ mindestens dann möglich ist, wenn $\mathcal{P}$ es erlaubt.*

□

Wie wichtig jedes einzelne dieser Schutzziele ist, hängt immer von der Anwendung selbst ab. Die Reihenfolge der Definitionen ist hier willkürlich gewählt. Jede Art von Sicherheitsverletzung kann als beliebige Kombination der Verletzung der drei Schutzziele betrachtet werden, wie später in diesem Kapitel noch gezeigt wird.

Die Schutzziele werden meist während des Designs und der Entwicklung von Rechensystemen und der Software festgelegt. Oft wird deren Gewichtung auch nachträglich geändert oder zusätzliche Schutzziele für ein laufendes System festgelegt, wenn beispielsweise neue Angriffe bekannt werden. Im Allgemeinen müssen alle drei Schutzziele berücksichtigt werden, nur in Sonderfällen oder bei der Betrachtung von Teilsystemen kann ein Schutzziel unwichtig werden.

In der Praxis werden häufig noch andere Schutzziele benannt, die für bestimmte Anwendungen wichtig sein können. Zwei zusätzliche, aktuell wichtige Schutzziele sind die Verbindlichkeit und die Originalität.

Verbindlichkeit (engl. non repudiation) beschreibt die Eigenschaft, dass eine Aktion von dem ausführenden Subjekt hinterher nicht geleugnet oder bestritten werden kann. Wichtig ist dieses Ziel beispielsweise beim elektronischen Geschäftsverkehr, damit eine Handlung später nachweisbar bleibt und einen rechtsverbindlichen Charakter behält.

Unter *Originalität* (engl. originality) versteht man die Eigenschaft, dass gespeicherte Daten nicht unberechtigt weitergegeben und kopiert wurden. Dieses Schutzziel ist heute sehr aktuell, weil es beispielsweise den Schutz vor Raubkopien beschreibt, der sowohl bei klassischen Computerprogrammen und -daten als auch bei Unterhaltungsdaten (Musik, Videos) ein sehr großes Problem darstellt.

Die Verbindlichkeit kann man allerdings auch dadurch beschreiben, dass Vertraulichkeit, Integrität und Verfügbarkeit im Sinne der angegebenen Definitionen berücksichtigt werden. Das nachträgliche Abstreiten einer durchgeführten Aktion ist nur dann möglich, wenn diese anonym oder mit gefälschter Identität durchgeführt wurde. Das widerspricht aber der dem Schutzziel Vertraulichkeit. Darin wird nach oben gezeigter Definition gefordert, dass die Policy bestimmt, wer Zugriff auf Daten haben darf und schließlich folgt daraus, dass die Sicherheitsregeln unzureichend definiert wurden. Ebenso kann die Originalität durch das Schutzziel Vertraulichkeit ausgedrückt werden.

Definition 2.10 (Gegenmaßnahmen)
Sicherheits-Mechanismen und andere Maßnahmen zur Erreichung der festgelegten Schutzziele können anhand ihrer Wirkzeiten klassifiziert werden.

1. Prävention *(engl. prevention): Maßnahmen, die vor der Ausführung des Systems die Sicherheit erhöhen, wie bei der Entwicklung, der Distribution oder der Installation.*
2. Erkennung *(engl. detection): Maßnahmen, die stattfindende und durchgeführte Angriffe zur Laufzeit des Systems erkennen.*
3. Reaktion und Wiederherstellung *(engl. recovery): Maßnahmen, die auf einen erkannten Angriff reagieren und einen abgesicherten Systemzustand wiederherstellen.*

Unter Gegenmaßnahmen *(engl. Countermeasure) oder* Sicherheitsmaßnahmen *versteht man die Summe aller präventiven, erkennenden und reaktiven Maßnahmen zur Erhöhung der Sicherheit.*

□

Aufgrund der hohen Komplexität und der Größe der Software sind Softwaresysteme praktisch nie fehlerfrei. Insbesondere die Fehler, die eine Umgehung der Gegenmaßnahmen ermöglichen, sind für diese Arbeit interessant.

Definition 2.11 (Schwachstelle)
Eine Schwachstelle *(engl. Vulnerability) ist eine Eigenschaft eines Systems im Design, in der Implementierung, während des Betriebs oder beim Management, die zur Umgehung der Policy ausgenutzt werden kann.*

□

Bei der Diskussion über die Sicherheit eines Gesamtsystems wird alles erwogen, was zur Umgehung der Policy führen kann. Das können neben Schwachstellen auch organisatorische Mängel und Benutzerfehler sein. Dies führt zur Definition der Angriffe und Bedrohungen.

Definition 2.12 (Angriff)
Ein Angriff *(engl. Attack, auch Threat Action) ist eine Aktion eines Subjektes $\mathcal{A}$ gegen die Sicherheitsmaßnahmen eines Systems zur Umgehung der Sicherheits-Policy. $\mathcal{A}$ nennt man den Angreifer.*

Man charaktierisiert Angriffe anhand des Ortes von $\mathcal{A}$ als interne und externe Angriffe und anhand seiner Aktivitäten als aktive und passive Angriffen.

Ein interner Angriff *wird von einem Subjekt innerhalb des abgesicherten Systems durchgeführt, ein* externer Angriff *von außerhalb.*

Aktive Angriffe *beeinflussen Systemressourcen oder beeinträchtigen den Betrieb des Systems.* Passive Angriffe *umgehen die Schutzziele durch Beobachtung oder Abhören ohne eine Aktion durchzuführen, die direkt gegen das System gerichtet ist.*

□

Die Umgehung der Sicherheits-Policy, die aus verschiedensten Gründen geschehen kann, nennt man auch Verletzung der Sicherheit.

Definition 2.13 (Bedrohung)
Eine Bedrohung *(engl. Threat) ist eine existierende, mögliche Verletzung der IT-Sicherheit eines Systems.*

□

Bedrohungen ergeben sich vor allem aus der Bewertung von aktiven und passiven Angriffen auf ein System. Man sagt auch, dass eine Bedrohung eine existierende Gefahr für ein System darstellt. Der Unterschied zwischen Bedrohungen und Angriffen wird im Folgenden kurz diskutiert.

Angriffe können aufgrund der Existenz der passenden Bedrohung ermittelt und durchgeführt werden. Bedrohungen existieren, weil bekannte Angriffe eine Gefahr für ein System darstellen und deshalb als Bedrohung angesehen werden.

Allerdings können einerseits Angriffe vorhanden sein, die nicht als Bedrohung angesehen werden müssen. Beispielsweise stellt die permanente Anfrage an einen Netzwerkdienst im

Allgemeinen einen Angriff dar. Ist das System, das diesen Dienst implementiert, jedoch so konstruiert, dass häufige Anfragen technisch kein Problem darstellen, so ist dieser Angriff nicht als Bedrohung gegen das System zu bewerten.

Andererseits können Bedrohungen existieren, deren Gefahr nicht durch Angriffe ausgehen. Der Verlust der Spannungsversorgung eines Rechners stellt zum Beispiel eine Bedrohung gegenüber der Verfügbarkeit dar. Allerdings muss dafür nicht unbedingt ein Angriff durchgeführt werden, ein leerer Akku kann ebenfalls eine Ursache dafür sein.

In der Abbildung 2.1 werden die Zusammenhänge zwischen den einzelnen Begriffen grafisch dargestellt. Das unten dargestellte IT-System besitzt im Allgemeinen gewisse Schwachstellen, die Wirkungen und Folgen nach sich ziehen können. Die Schwachstellen können von Angreifern durch aktive oder passive Angriffe ausgenutzt und Systemressourcen verwendet werden. Neben Anderen stellen externe und interne Angreifer die größten Bedrohungen dar.

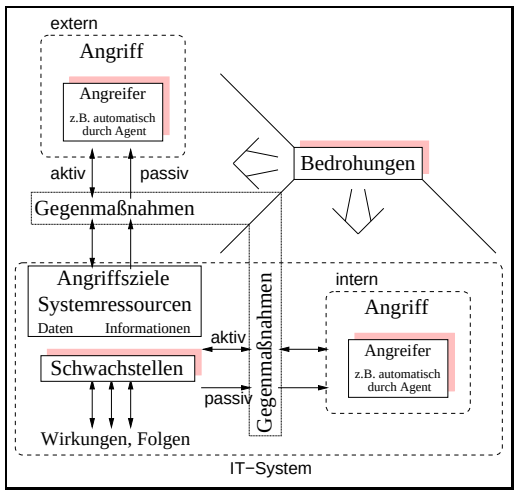

Abbildung 2.1: Darstellung der Begriffe

Ziel der Entwicklung von abgesicherter Software muss es daher immer sein, die Menge der möglichen Bedrohungen feststellen zu können. Der Teilprozess in der Softwaretechnik, der sich damit beschäftigt, wird *Bedrohungsanalyse* (engl. Threat Analysis) genannt.

Der eigentliche Begriff der Sicherheit wird in dieser Arbeit zunächst als Schutz der Informationen eines Rechensystems verstanden. Dieses Gebiet der *Informationssicherheit* lässt sich in zwei Untergebiete unterteilen. Die *Computersicherheit* auf der einen Seite beschäftigt sich mit der sicheren Speicherung und Verarbeitung von informationstragenden Daten in Rechensystemen. Auf der anderen Seite existiert das Teilgebiet der *Kommunikationssicherheit*, das sich mit der sicheren Übertragung von informationstragenden Daten in und zwischen Rechensystemen befasst.

Problematisch bei diesen Begriffsbildungen ist, dass die Sicherheitsbegriffe wiederum mit einer *sicheren* Eigenschaft beschrieben werden. Besser ist demnach folgende Definition:

Nach [Eck02] ist die *Informationssicherheit* die Eigenschaft eines funktionssicheren Systems, welches nur Systemzustände annehmen kann, die zu keiner unauthorisierten Informationsveränderung oder -gewinnung führen.

Dabei wird die Informationssicherheit auf den Schutz vor unauthorisierter Veränderung und Informationsgewinnung beschränkt. Neben diesen möglichen Angriffen auf Schutzziele existieren noch weitere Ziele, allen voran die Verfügbarkeit, die kein reines Schutzziel der Funktionssicherheit darstellt.

Andere Definitionen gehen genauer auf die Verfahren der Informationssicherheit ein, wie etwa in [Gol99]: „Computersicherheit behandelt präventive und erkennende Maßnahmen unauthorisierter Aktionen durch die Benutzer eines Rechensystems."

Dies führt nun zur Definition der IT-Sicherheit, wie sie dieser Arbeit zugrunde liegt.

Definition 2.14 (IT-Sicherheit)
Die IT-Sicherheit *(engl. Security) ist die Eigenschaft eines Rechensystems sowohl präventive, erkennende, wie auch reaktive Maßnahmen einzusetzen, um die festgelegten Schutzziele im Rahmen der Sicherheit-Policies durchzusetzen.*

□

Definition 2.15 (Absicherung)
Die Absicherung *beschreibt alle technischen und organisatorischen Mittel, die zur Einhaltung der IT-Sicherheit dienen können.*

□

Neben den klassischen Schutzzielen wird im Umfeld der IT-Sicherheit noch der Aspekt des Datenschutzes betrachtet. Dabei werden die Rechte der Personen betrachtet, nicht nur der technische Schutz des Systems.

Definition 2.16 (Datenschutz)
Der Datenschutz *(engl. Privacy) behandelt die Rechte der Person, meist des Benutzers eines Systems, über alle Aktionen bestimmen zu können, die seine persönlichen Daten und Informationen betreffen.*

□

Der Bereich des Datenschutzes enthält spezielle Schutzziele, wie etwa die *Anonymität*, die ausdrückt, dass keine Zuordnung zwischen einer Aktion und dem ausführenden Benutzer möglich ist. Diese Art von Schutzzielen werden in dieser Arbeit durch den gemeinsamen Begriff Vertraulichkeit ausgedrückt. Andere Belange des Datenschutzes, wie etwa juristische Fragestellungen, werden hier nicht näher diskutiert, können aber in vielen Papieren nachgelesen werden, beispielsweise in [Smi93], [ML00], [Pfl97] und [Sha79].

2.2 Konkrete Bedrohungen und Angriffe

Im Folgenden werden bekannte Angriffe und damit entstehende Bedrohungen für ein System dargestellt. Diese werden genau analysiert und stellen die Basis für die in Abschitt 2.3 folgende Klassifikation dar. Da nur ein Ausschnitt der möglichen Angriffe dargestellt werden kann, werden vor allem aktuelle Angriffe berücksichtigt, die heute für die Praxis relevant sind. Es wird aus den bekannten Angriffsbereichen je ein Angriff erklärt, damit ein großes Spektrum für die Klassifikation vorhanden ist.

2.2.1 Schwachstellenklassifikation

In [LBMC94] werden 50 bekannte Angriffe analysiert und die zugrundeliegenden Schwachstellen nach folgenden Kriterien dargestellt:

1. Ursprung und Absicht der Schwachstelle,
2. Entstehungszeitpunkt der Schwachstelle und
3. Ort der Schwachstelle.

Die analysierten Angriffe sind in drei Matrizen für jede der Kriterien der Schwachstellen eingeteilt. Hier wird die Einteilung als Hierarchie in den drei Baumdiagrammen dargestellt. In den folgenden Diagrammen 2.2, 2.3 und 2.4 werden die drei Kriterien bis hin zum konkreten Beispiel verfeinert. Im Diagramm 2.2 werden die Schwachstellen nach ihrem Ursprung eingeteilt. Grundsätzlich wird zwischen dem unabsichtlichen (2.2:1.1) und dem absichtlichen Einbringen einer Schwachstelle (2.2:1.2) unterschieden, wobei Letzteres dann einen Angriff darstellt, wenn es böswillig (2.2:1.2.2) geschieht. Die numerischen Werte, die in den Blättern in Klammern mit angegeben sind, zeigen die Anzahl der untersuchten Angriffe für den jeweiligen Bereich.

In der Abbildung 2.3 werden die Schwachstellen nach der Zeit ihres Entstehens eingeteilt. Auffällig ist, dass die meisten Schwachstellen bereits während der Entwicklungszeit (2.3:1.1) in das System gebracht werden. Dabei beruhen die meisten wiederum auf fehlerhafter Spezifikation oder fehlerhaftem Design (2.3:1.1.1), nur eine Schwachstelle wird während der Erzeugung des Objektcodes (2.3:1.1.3) generiert. Das ist hauptsächlich dadurch zu erklären, dass der Übersetzungsvorgang vom Quellcode zum ausführbaren Programm vom Compiler automatisch durchgeführt wird und deshalb bei diesem Schritt nur wenige Fehlermöglichkeiten vorhanden sind.

Der Ort, an dem die Schwachstelle innerhalb des Systems vorhanden ist, ist die Grundlage für das Diagramm 2.4. Charakteristisch ist, dass nur wenige Angriffe auf Schwachstellen in der Hardware (2.4:1.2) beruhen, viele werden im Bereich des Betriebssystems (2.4:1.1.1)

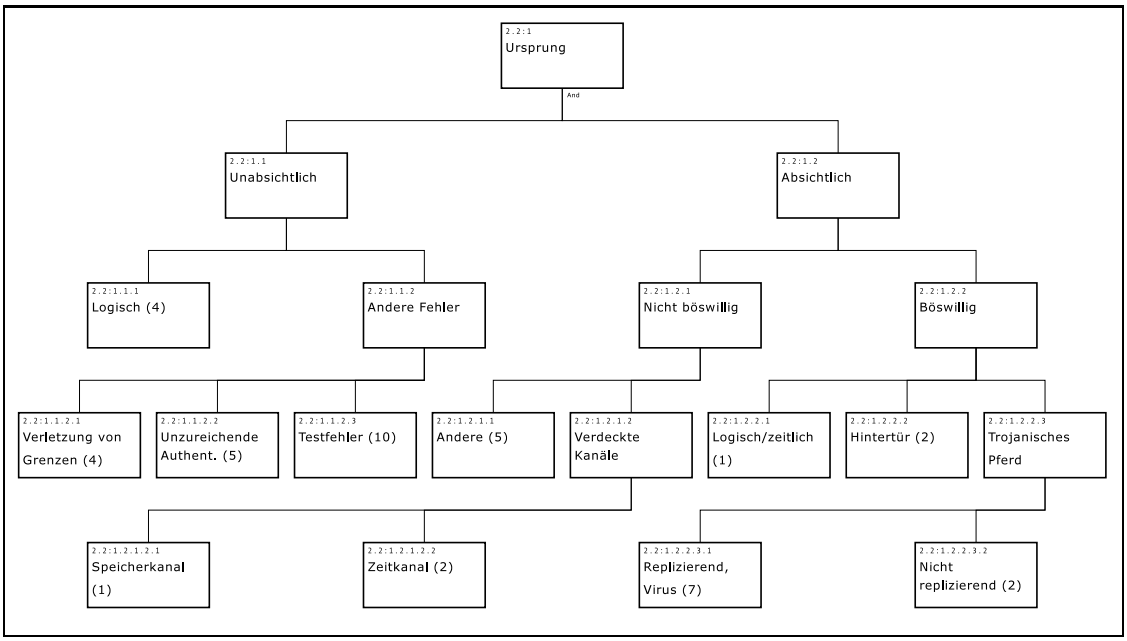

Abbildung 2.2: Klassifikation des Ursprungs bekannter Schwachstellen

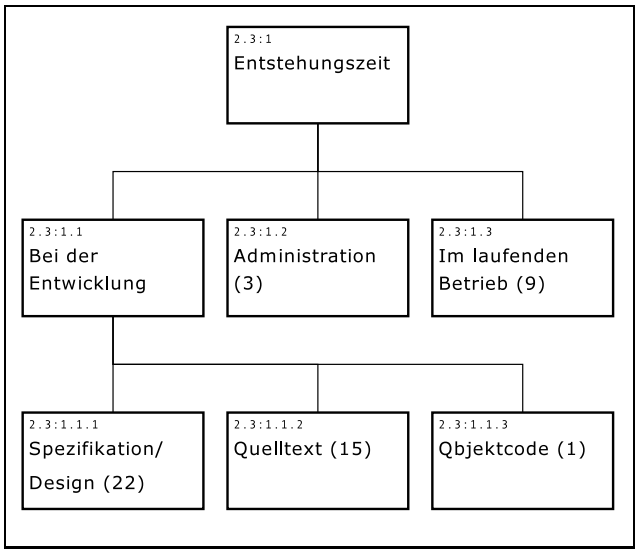

Abbildung 2.3: Klassifikation der Entstehungszeit bekannter Schwachstellen

gefunden. Dass allerdings nur ein Angriff durch Anwendungsschwachstellen (2.4:1.1.3) möglich ist, ist hauptsächlich auf die Auswahl der 50 untersuchten Angriffe zurückzuführen und darf nicht als allgemeine Tatsache verstanden werden. Aktuelle Angriffe sind vielmehr mindestens in gleichem Maße auf fehlerhafte Anwendungssoftware zurückzuführen als auf Schwachstellen im Betriebssystemkern.

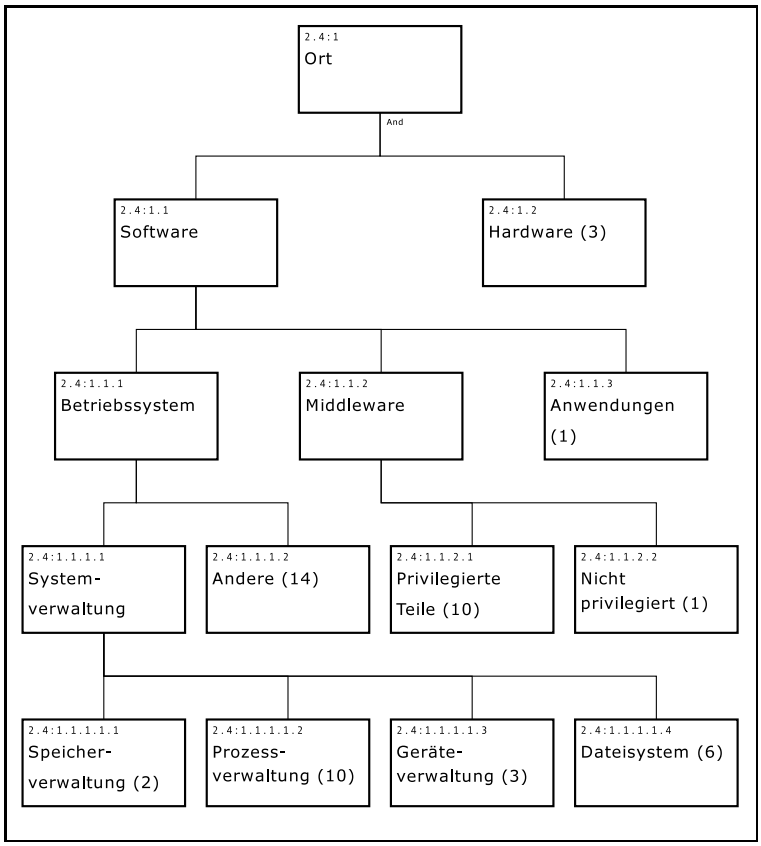

Abbildung 2.4: Klassifikation des Ortes bekannter Schwachstellen

2.2.2 Einordnung von Bedrohungen

Neben der Einordnung möglicher Schwachstellen ist Wissen über bekannte Angriffe wesentlich, die ernsthafte Bedrohungen darstellen können. Im Folgenden werden deshalb Bedrohungen auf Basis bekannter Angriffe in die drei zeitlichen Entstehungsphasen eingeteilt. Diese Phasen werden später im Abschnitt 2.3.1 noch genauer beschrieben. Sie decken alle bekannten Bedrohungen ab und dienen als Basis zur Klassifikation.

2.2.2.1 Bedrohungen bei der Konstruktion

Die erste Phase, in der Bedrohungen für ein System entstehen können, ist während der Konstruktion. Eine Diskussion dazu findet sich in [Cor91]. Einerseits können bei der Entwicklung des Konzeptes logische Fehler auftreten, die zu Bedrohungen auf das System führen können. Die Behebung dieser Fehler ist meist mit hohem Aufwand verbunden. Andererseits können bei der Entwicklung programmiertechnische Fehler oder Fehler des Übersetzers zu Schwachstellen führen, die ein Angreifer ausnutzen kann. Es folgen drei

kleine Beispiele für Fehler bei der Konstruktion, die zu großen Bedrohungen der Systeme geführt haben.

Syntaktische, unerkannte Fehler

In der Abbildung 2.5 ist in Zeile 1 ein Teil eines Fortran-Programms zu sehen, der zur Steuerung einer amerikanischen Weltraumsonde gehört. Die Anweisung beschreibt den Beginn einer Do-Schleife mit dem Intervall $I = [1, 100]$.

```
1 DO 20 I = 1,100
2 DO 20 I = 1.100
```

Abbildung 2.5: Ungewollter Fehler in einem Fortran-Programm

Tatsächlich wurde vom Programmierer allerdings die Zeile 2 eingegeben, die die Zuweisung der implizit deklarierten Variablen $DO20I$ mit dem Wert $1,100$ bewirkt. Dieser absichtlich oder unabsichtlich eingebrachte Fehler bewirkte den Absturz der Sonde und zeigt, wie schnell kritische Fehler in Software entstehen und wie schwierig es auch trotz manueller Code-Inspektion ist, derartige Fehler zuverlässig zu finden. Zudem ist trotz des Auffindens des Fehlers danach nur schwer zu ermitteln, ob der Fehler absichtlich oder aus Versehen in das System eingebracht wurde. Ob tatsächlich ein Angriff stattfand ist bei dieser Art von Fehlern deshalb nur schwer nachzuvollziehen.

Codemanagement bei der Entwicklung

Bei großen Softwareprojekten ist der Aufwand zur Verwaltung der Quelltexte hoch. Meistens werden zur Unterstützung Versionsverwaltungssysteme eingesetzt. Nicht zu unterschätzen sind jedoch die Bedrohungen, die durch Verwendung derartiger Tools entstehen können, da die Entwickler neben der Sicherheit des eigenen Produktes auch auf die Sicherheit der Verwaltungswerkzeuge vertrauen müssen. Durch das Ausnutzen von Schwachstellen könnte ein Angreifer den Quelltext verändern und beispielsweise Hintertüren einbauen. Ist die dadurch entstandene Bedrohung nur einem Angreifer bekannt, so ist es unwahrscheinlich, dass sie überhaupt erkannt wird. Gerade im Open Source Bereich sind immer wieder Gerüchte um derartig entstandene Schwachstellen im Umlauf, und es ist vorstellbar, dass zahlreiche Softwaresysteme bereits mit Schwachstellen ausgerüstet sind, die nur wenigen bekannt sind. Beispielsweise ist in der GNU-Implementierung von `crontab` seit Jahren ein Fehler enthalten gewesen[1], der es einem einfachen Unix-Benutzer erlaubt, innerhalb

[1] bis 3/2003

von Sekunden ohne Programmierkenntniss eine Befehlszeile zu erhalten, die Systemrechte freigibt. Es ist wahrscheinlich, dass dieser Fehler nicht zufällig entstanden ist, sondern unter Umgehung der Sicherheitsmaßnahmen des Verwaltungswerkzeugs absichtlich in den Quelltext eingebaut wurde.

Fehler in Kryptoverfahren

Neben erwähnten programmiertechnischen Fehlern kommen noch logische Fehler hinzu. Klassische Beispiele hierfür sind Fehler bei kryptografischen Verfahren. Das Needham-Schroeder Public-Key Protokoll zum Aushandeln eines gemeinsamen Sitzungsschlüssels wurde lange Jahre als einfaches Protokoll beschrieben. Der Fehler, der darin enthalten ist, wurde allerdings erst viele Jahre später gefunden und veröffentlicht, obwohl das Protokoll sehr einfach ist. Dieses Beispiel zeigt, dass logische Fehler durchaus entstehen können und auch ein jahrelanges, fehlerfreies Anwenden von Verfahren keine Sicherheit garantieren kann. Trotzdem stellen Kryptoverfahren eine der Bauteile für sichere Systeme dar, die mathematisch formal meist gut analysiert und damit vertrauenswürdig sind. Zudem muss erwähnt werden, dass nicht nur kryptografische Protokolle, sondern ebenso Kommunikationsprotokolle als Angriffsziele geeignet sind, wie etwa für die TCP/IP Protokollfamilie in [Bel89] gezeigt wird. Kryptographie wird als Baustein für sichere Systeme im Abschnitt 2.4.3.2 katalogisiert.

2.2.2.2 Bedrohungen zur Laufzeit

Bedrohungen, die zur Laufzeit auftreten, basieren meist auf Schwachstellen, die schon zur Konstruktionszeit in das System eingeflossen sind. Die Trennung ist daher nicht immer einfach und strikt. In dieser Arbeit wird eine Bedrohung dann zur Laufzeit angesehen, wenn der Großteil der Arbeit beim Ausnutzen der Schwachstelle bzw. bei der Durchführung des Angriffs zur Laufzeit des Systems geschieht. Wenn die wesentlichen Aktionen zur Entstehung der Bedrohung bei der Konstruktion durchgeführt wurden, liegt dagegen eine Bedrohung zur Entwicklungszeit vor.

Pufferüberläufe

Die meisten der heutigen Angriffe basieren auf Schwachstellen in Programmen, die durch den Überlauf begrenzter Puffer (Bounded Buffer) entstehen. Können logisch Angriffe auf IT-Systeme schon in abstrakten Modellen erkannt werden, sind die Angriffe durch Pufferüberlauf nur am fertigen Code, also an der konkreten Realisierung, zu erkennen. Sie hängen zudem sehr stark von der Einsatzumgebung ab. Das bedeutet einerseits, dass auch bei sorgfältiger Entwicklung derartige Schwachstellen durch kleinste Fehler in der Realisierung auftreten. Andererseits bestehen daher auch gute Möglichkeiten, diese Fehler auf

Quelltextebene automatisch bzw. Tool-unterstützt zu finden. Obwohl diese Angriffe schon seit langem bekannt sind, werden sie heute in zunehmendem Maße erfolgreich durchgeführt, weshalb im Folgenden die wesentlichen Mechanismen vorgestellt werden.

Grundsätzlich treten die Probleme durch Pufferüberlauf-Angriffe nur an den Stellen auf, an denen Datenpuffer im Speicher abgelegt werden, die eine definierte, feste Größe haben und damit eine feste Anfangs- und Endeadresse im Speicher besitzen. Erstes Ziel von Pufferüberlauf-Angriffen ist stets die Grenze der Anfangs- oder Endeadresse zu übertreten und somit die Integrität bestimmter, angrenzender Daten zu beeinflussen, wie etwa in [Lee88] beschrieben. Ob die Programmausführung in Folge der Integritätsverletzung Code des Angreifers ausführt, hängt davon ab, ob der Angriff auf genau die Versionen der beteiligten Programme abgestimmt war. Als Beteiligte sind neben den Prozessen auch Middleware- und Kommunikationsplattformen und das Betriebssystem zu sehen. Zudem ist ein Angriff sowohl vom Speicherlayout des Prozesses abhängig, das vom Übersetzer festgelegt wird als auch vom dynamischen Laufzeitsystems des Compilers, das die dynamische Speicherverwaltung realisiert.

Das Speicherlayout eines Prozesses $\mathcal{P}$ in Betriebssystemen mit virtueller Speicherverwaltung sieht wie folgt aus. Betrachtet wird das Programm in Abbildung 2.6.

```
1 char globalString[] = "Text";
2 int  globalInteger;
3
4 void main (int argc, char* argv[])
5 {
6   int   localInteger;
7   char *localString = (char *)malloc(sizeof(char)*4);
8   localInteger++;
9 }
```

Abbildung 2.6: Beispielprogramm in C

Ein Übersetzer erzeugt verschiedene Speicherregionen des Prozesses $\mathcal{P}$ jeweils für unterschiedliche Aufgaben. Diese sind in der folgenden Tabelle 2.1 mit Verweisen auf die entsprechende Stelle im abgebildeten Programm dargestellt.

Bei einem Angriff wird versucht, sicherheitskritische Bereiche durch Daten des Angreifers, die beispielsweise als Argumente oder durch Protokolle übergeben werden, zu überschreiben. Überläufe können in allen Segmenten eines Prozesses durchgeführt werden, die beschreibbar sind.

Segment	Programmzeile
Datensegment für globale, initialisierte Daten	1
Globale, uninitialisierte Daten (Block Storage Segment BSS)	2
Programmargumente und Umgebung (Environment)	4
Keller (Stack)	6
Halde (Heap)	7
Textsegment, nur lesbar, für den Programmcode und Konstanten	8

Tabelle 2.1: Segmente eines Unix-Prozesses

Überlauf auf dem Keller

Das am häufigsten genutze Segment für derartige Überläufe ist der Keller (Stack Overflow). Es ist nicht kompliziert ein sicherheitskritisches Objekt auf dem Keller zu finden, das überschrieben werden kann. Einfaches Beispiel ist die Rücksprungadresse von Funktionen, die stets auf dem Keller zwischengespeichert wird.

Ein Angreifer wird versuchen, die Rücksprungadresse so zu überschreiben, dass nach dem Verlassen der Funktion nicht mehr zum Aufrufer, dem dynamischen Vorgänger, sondern in einen Code des Angreifers gesprungen wird. Dieser Programmcode kann im einfachsten Fall zusammen mit dem Überschreiben der Rücksprungadresse auf den Keller gebracht worden sein. Grundsätzlich gefährden Angriffe dieser Art stets zunächst die Integrität des Kellerinhaltes. Durch das Einbringen und Ausführen von fremdem Programmcode des Angreifers kann zudem die Vertraulichkeit der gespeicherten Daten innerhalb des Systems, als auch dessen Verfügbarkeit gefährdet werden.

Der Keller eines Prozesses befindet sich in gängigen Betriebssystemen in Seiten des virtuellen Speichers, die sowohl lese- und schreibbar sind, als auch Ausführungsberechtigung besitzen. Diese Attribute werden vom Kern in der Seitenkacheltabelle verwaltet und vom Kern festgelegt. Gegen die Ausführung von eventuell gefährlichem Code auf dem Keller existieren in verschiedenen Betriebssystemen bereits Lösungen. Dazu wird lediglich das Ausführungsattribut der entsprechenden Seiten entfernt. Jeder Versuch, den Programmzähler auf diese Seiten zu setzen, wie etwa durch einen direkten Sprungbefehl, wird durch die Speicherverwaltungseinheit, die in Hardware realisiert ist, abgefangen wird. Diese löst einen Seitenfehler aus, unterbricht damit den kompromitierten Prozess und startet eine Behandlungsroutine im Kern.

Als Beispiel dient das Programm 2.7, in dem die Argumente, die der Benutzer beim Programmstart angegeben hat, an eine Funktion in einen lokalen, begrenzten Puffer kopiert

werden. Die Bereichsgrenzen werden dabei nicht geprüft, weshalb jeder Aufrufer des Programms grundsätzlich die Möglichkeit hat, sowohl den Basiszeiger als auch die Rücksprungadresse zu verändern. Er muss lediglich mit den Argumenten die Länge 32 überschreiten und kann so ungewünschtes Verhalten erzeugen.

```
1  void verbinde(int anzahl, char* strings[])
2  {
3    char lPuffer[32];
4    char *zeiger = lPuffer;
5    int  lInteger;
6
7    for (lInteger=1; i<anzahl, lInteger++)
8    {
9      strcpy(zeiger, strings[lInteger]);
10     zeiger += strlen(strings[lInteger]);
11   }
12 }
13
14 int main(int argc, char* argv[])
15 {
16   verbinde(argc, argv);
17 }
```

Abbildung 2.7: Beispielprogramm zur Schwachstelle eines Pufferüberlaufs auf dem Keller

Der Keller ist beim Unterprogrammaufruf so aufgebaut, wie in Abbildung 2.8 zu sehen ist. Klassische Angriffe füllen den Bereich `lPuffer` mit einem Maschinencode des Angreifers, beispielsweise dem Ausführen einer Befehlszeile. Nach dem Basepointer wird die Rücksprungadresse mit dem Wert auf `lPuffer` überschrieben. Wenn der Rücksprung geschehen soll, beispielsweise durch einen `RET`-Befehl, wird der Code des Angreifers auf dem Keller ausgeführt.

In der Praxis müssen dabei einige Details berücksichtigt werden, wie etwa der exakte Mechanismus der Kopierfunktion oder die Anzahl der Argumente, was aber prinzipiell nichts ändert.

Diese Schwachstellen sind leicht auszunutzen, wenn sowohl der Quelltext als auch der Übersetzer und das Laufzeitsystem des Prozesses dem Angreifer bekannt sind. In Softwaremonokulturen ist das Ausnützen derartiger Schwachstellen leichter, als wenn Programme jeweils mit unterschiedlichen Laufzeitsystemen und mit verschiedenen Compilern übersetzt werden. In einem späteren Abschnitt dieses Kapitels werden mögliche Gegenmaßnahmen dargestellt.

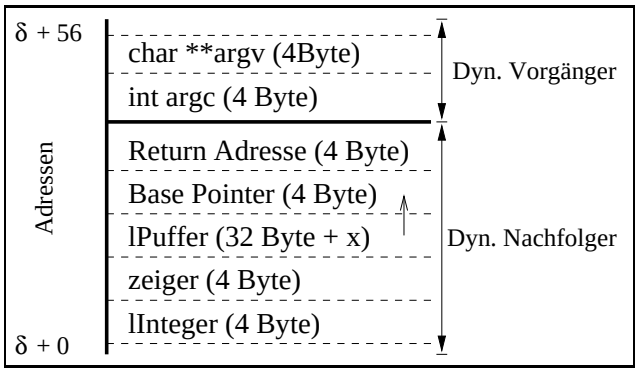

Abbildung 2.8: Aufbau des Kellers nach Programm 2.7

Überlauf auf der Halde

Es existieren auch Schwachstellen, die auf einem Überlauf von Puffern auf der Halde basieren (Heap Overflow). Diese sind weitaus schwieriger auszunutzen als Kellerüberläufe, da die Organisation der Halde komplizierter ist als die des Kellers und es selbst mit verfügbarem Quellcode schwer ist, Objekte auf der Halde zu finden, die an einem überschreibbaren Puffer anschließen und sicherheitskritisch sind.

Die Probleme mit Pufferüberläufen werden oft als Verwundbarkeiten in Abhängigkeit der verwendeten Programmiersprache dargestellt. Tatsächlich sind Angriffe hauptsächlich in C oder C++ Programmen zu finden, da diese effizienten Hochsprachen keine automatische Bereichsprüfung durchführen. Die Prüfung kann sowohl zur Übersetzungszeit teilweise, als auch zur Laufzeit vollständig durchgeführt werden. Dazu müssen nur die Unter- und Obergrenzen der Puffer bekannt sein, damit der Compiler automatisch Prüfsequenzen in den Programmcode einfügen kann. Problematisch bei C und C++ Programmen sind Pointer, die auf einen begrenzten Puffer zeigen und nicht automatisch identifiziert werden können. Schreibzugriffe durch diese Pointer über einen begrenzten Bereich hinaus können nicht erkannt werden.

Viele andere Sprachen, allen voran dynamische Sprachen wie Java, führen eine Bereichsprüfung sowohl bei der Übersetzung als auch während der Laufzeit durch und können Überläufe erkennen. Das setzt einerseits eine fehlerfreie Sprachimplementierung voraus und gilt andererseits dann nicht mehr, wenn der Programmierer eigene Datenstrukturen benutzt, deren Bereichsgrenzen nicht auf Überschreitung hin geprüft werden. Es existieren Beispiele für derartige Schwachstellen auch unter Java. Wichtig ist also, dass Programmierer jeder Sprache sich stets den Gefahren bewusst ist, völlig verhindern kann man diese Schwachstellen nicht, wie auch in [SUN] gewarnt wird. Nähere Informationen zu Pufferüberläufen finden sich auch unter [JV02].

Als Beispiel dient das Programm 2.9, welches ein Argument in einen begrenzten Puffer auf der Halde nach `lString1` kopiert. Es finden keinerlei Bereichsprüfungen statt, so dass ein

anschließendes Datum überschrieben werden kann. Zwar liegt die Vermutung nahe, dass das Datum `lStringRootUser` direkt im Anschluß daran auf der Halde zu finden ist, tatsächlich hängt das aber von der Laufzeitumgebung des Prozesses und damit auch vom verwendeten Übersetzer ab. Ein Angriff geht deshalb zwangsläufig von speziellen Übersetzer- und Betriebssystemversionen aus, um wirkungsvoll zu sein.

```
1  int main (int argc, char* argv[])
2  {
3    char *lString1 = (char *)malloc(sizeof(char)*4);
4    char *lStringRootUser = (char *)malloc(sizeof(char)*10);
5
6    strcpy(lStringRootUser, "Huber");
7    strcpy(lString1, argv[1]);
8
9    printf("Name of Rootuser is %s", lStringRootUser);
10 }
```

Abbildung 2.9: Beispiel zur Schwachstelle eines Pufferüberlaufs auf der Halde

Ein Aufruf des Programms mit einem Argument, das aus einem Feld von 20mal dem Zeichen „a“ besteht, wird das Datum `lStringRootUser` mit dem Wert „aaaa“ überschreiben[2]. Das für das Datum `lString1` auf der Halde 16 Byte reserviert werden, obwohl nur vier gebraucht werden, liegt am Übersetzer, die Halde wird durch die Verwendung eines anderen Compilers unterschiedlich verwaltet.

Viren und Würmer

Die verbreitetsten und populärsten Angriffe im Internet sind die von Viren oder Würmern und sollen deshalb im Folgenden diskutiert werden. Viren sind möglichst unauffällige kleine Programme, die sich in anderen Programmen, sogenannten Wirtsprogrammen, einnisten. Durch die Verbreitung und den Start der infizierten Wirtsprogramme wird der Virus auf neuen Rechensystemen aktiv und kann sich ausbreiten.

Würmer dagegen sind eigenständige Programme in einem Rechensystem, die Schwachstellen anderer Systeme für ihre Verbreitung ausnutzen. Sie hängen sich nicht wie Viren an ein bestehendes Programm an, sondern arbeiten als selbständiger Prozess. Die Integrität von Programmen wird, ganz im Gegensatz zu Virenattacken, durch einen erfolgten Wurmangriff nicht beeinträchtigt. Weil Integritätsverletzungen mit den Mitteln leicht entdeckt werden können, die vertrauenswürdige Plattformen bieten, wird im Folgenden über Wurmangriffe und deren Vermeidung diskutiert.

[2] Übersetzt mit dem gcc Version 3.3 unter Linux glibc Version 2.3.2

Die Angriffe durch Würmer treten heute in wechselnden Abständen auf. Deren Verlauf und Schadensgröße ist weitgehend unvorhersagbar und die Ausbreitungsgeschwindigkeit enorm. Zudem ist der Aufwand, einen wirkungsvollen Wurm herzustellen, eher gering. Die Kenntnisse über Schwachstellen können leicht nachgelesen werden und Tests auf die Wirksamkeit lassen sich ohne Konsequenzen beliebig oft am eigenen Rechner wiederholen. Sämtliche benötigten Werkzeuge sind heute für jedermann im Internet frei verfügbar.

Direkter Schaden durch diese bösartige Software besteht im Wesentlichen aus dem Schaden, den der fremde Programmcode selbst anrichtet, wie etwa das unkontrollierte Herunterfahren des Systems, die Zerstörung der Integrität von Programmen und Daten und der erhöhte Ressourcenverbrauch. Der indirekte Schaden ist oftmals wesentlich höher, der beispielsweise aus dem präventiven Abschalten eines kompletten Netzwerks oder der Arbeitszeit für das Bereinigen der Systeme nach einem erfolgten Angriff besteht.

Voraussetzungen für Wurmangriffe

Voraussetzungen für nennenswerte Wurmangriffe und Schäden sind vor allem Software-Monokulturen, die eine große Anzahl an gleichen Systemen darstellen. Wird eine Schwachstelle der eingesetzten Software entdeckt, so ist diese auf allen Systemen ausnutzbar. Dabei ist zu beachten, dass jede Softwareschicht gleichermaßen durch Angriffe gefährdet ist, der Betriebsystemkern ebenso wie Bibliotheken oder Anwendungsprozesse. Wäre die Softwarelandschaft gemischt, dann könnten entdeckte Schwachstellen nur wenige Rechner infizieren und der Schaden wäre gering. Dabei muss allerdings beachtet werden, dass schon der erste Internet-Wurm bereits verschiedene Schwachstellen unterschiedlicher Software ausgenutzt hat (`fingerd, sendmail` und Passwort-Attacke). Eine ausführliche Diskussion findet sich unter [Spa89b] und [Spa89a].

Zudem ist auch die Aktualität der Schwachstellen für eine Attacke interessant. Zwar verkürzt sich die Zeit zwischen dem Erkennen der Schwachstelle und der Installation der Gegenmaßnahmen immer mehr, dagegen wird aber die Verbreitungsgeschwindigkeit der Würmer durch schnellere Verbindungen immer größer. Das reine reaktive Vorgehen gegen diese schädliche Software, also die Reaktion nach erfolgtem Angriff beispielsweise mittels Virenscanner, muss durch wirksame präventive Maßnahmen erweitert werden. Ob der Einsatz von vertrauenswürdigen Modulen in Rechensystemen eine solche wirksame Maßnahme sein kann, wird im Folgenden nach der Beschreibung der Wirkungsweise eines Wurmes untersucht.

Arbeitsweise eines Wurms

Es existieren viele Beispiele für aktiv gewordene Würmer, die großen Schaden im Internet verursacht haben. In letzter Zeit waren dies vor allem der Windows-Wurm `W32/Sobig.F` und der `W32/Lovesan`, deren Funktionsweise aus diesem Grund kurz erwähnt wird.

Sobig.F

Erster verteilt sich sowohl als Anhang in E-Mails als auch über den Mechanismus der Windows-Laufwerksfreigabe. Auf infizierten Systemen wird systematisch nach bestimmten Dateien gesucht. Diese Dateien werden systematisch auf Strings durchsucht, die eine gültige E-Mail-Adresse darstellen. An die gefundenen Adressen werden infizierte E-Mails geschickt, ebenso werden alle auffindbaren Autostartordner, auch im Netzwerk, mit dem Wurm verseucht. Der etwa 100 KByte große `Sobig.F` verbreitet sich nur dann, wenn das Infektionsprogramm beispielsweise durch das Öffnen des E-Mail-Anhangs durch den Benutzer ausgeführt wird.

Lovesan

Demgegenüber existieren auch Würmer, die Schwachstellen in Systemen dazu ausnutzen, um Programme auf entfernte Rechner zu laden und dort auszuführen. Wichtig ist dabei, dass Benutzer keinerlei Aktionen ausführen müssen, damit sich der Wurm in einem System einnistet. Aktueller und bekannter Vertreter dafür ist der `W32/Lovesan` oder auch als MSBlast bekannte Wurm.

Die Verbreitung des Wurms geschieht über eine Schwachstelle, die am Port 135 von Windows 2000/XP Rechnern (DCOM RPC-Dienst) bekannt ist. Diese Schwachstelle ist auch deshalb für Würmer geeignet, weil sie sowohl auf Windows 2000 Systemen als auch auf Windows XP Systemen unabhängig des Service-Packs vorhanden war.

Durch einen Trick wird die Ein-/Ausgabe des Kommandozeileninterpreters `cmd.exe` an den Port 4444 gebunden. Der Angreifer hat nun eine Eingabeaufforderung, die er entfernt nutzen kann. Der Wurm startet dann sofort folgende Kommandos:

- `tftp -i <ip-source> GET msblast.exe`
- `start msblast.exe`

Auf dem angegriffenen Rechner wurde damit der Wurm übertragen und gestartet. Der Schädling versucht zuerst einen Eintrag in der Systemdatenbank zu erzeugen, damit er beim nächsten Neustart des Systems automatisch gestarten wird. Danach wählt er zufällig eine Klasse-C IP-Netzwerkadresse aus, die er nach möglichen Opfern durchsucht. Dieses Durchsuchen wird durch den Start von 20 weiteren Threads beschleunigt.

Wenn das aktuelle Datum nach dem 15.8. jeden Jahres liegt, werden speziell erzeugte Pakete an die Adresse `windowsupdate.com` geschickt. Hinter dieser Adresse verbrigt sich der Dienst von Microsoft, der auch für Sicherheitsupdates verantwortlich ist. Das eigentliche Ziel des Wurms sollte offensichtlich ein DDoS-Angriff auf genau die Adresse sein, die eigentlich vor dem Wurm schützen sollte. Genauere Analysen wie etwa Rückübersetzungen des Quelltextes kann man unter [Els03] und [MSB03] finden.

An diesen konkreten Wurmangriffen kann man mehreres erkennen. Erstens basieren diese Arten von Angriff auf Schwachstellen, die längst bekannt sind und deshalb keineswegs neuartige Angriffe darstellen. Die besprochenen Würmer nutzen lediglich Pufferüberläufe um Zugriff auf das Zielsystem zu bekommen, eine Schwachstelle, die seit langem bekannt ist und gegen die etliche wirkungsvolle Gegenmaßnahmen bekannt sind. Das aktuelle Beispiel und die hohe Schadenswirkung der Würmer zeigt, dass diese nicht konsequent eingesetzt werden. Zweitens kann man daran erkennen, dass die Konsequenzen der Angriffe sehr gering sind. Grundsätzlich stellt die Bereinigung der infizierten Rechner natürlich einen gewissen Aufwand dar, allerdings wird mit den bisher bekannten Angriffen nur selten Datendiebstahl oder Datenzerstörung betrieben. Das deutet darauf hin, dass die Angriffe kaum von wirtschaftlichen Interessen geleitet werden. Diese Aussagen treffen selbstverständlich nur auf die Angriffe zu, die bekannt werden und deren Schaden veröffentlicht wird. Dass viele Angriffe nie veröffentlicht werden, die ernsthafte Schäden und Verluste erzeugen, ist nicht auszuschließen. Es scheint aber unwahrscheinlich, da der Aufwand, der von kommerziellen Unternehmen für Sicherheit betrieben wird, mit dem Auftreten dieser Angriffe nicht spürbar steigt.

Verteilter DoS-Angriff

DoS-Angriffe (Denial-of-Service) stellen eine Bedrohung gegen das Schutzziel der Verfügbarkeit dar. Da der Angriff darauf abzielt, einen bestehenden, angebotenen Dienst derart intensiv zu nutzen, dass er aus Kapazitätsgründen nicht mehr gewohnt verfügbar bleibt, ist er leicht zu erkennen. Trotzdem werden derartige Bedrohungen leicht übersehen. DoS-Angriffe sind in den unterschiedlichsten Ausprägungen bekannt. Es sind sowohl Angriffe auf Netzwerkdienste von entfernten Rechnern und das Zerstören deren Verfügbarkeit bekannt als auch Angriffe innerhalb eines lokalen Rechners auf die Verfügbarkeit von Betriebssystemdiensten. Ein Beispiel für einen entfernten Angriff ist der vorher beschriebene Wurm Lovesan, lokale Angriffe zielen beispielsweise auf das Scheduling.

In der Praxis ist die Abwehr von Denial of Service-Attacken heute noch sehr problematisch, wie auch in [Gli83] beschrieben wird. Dies liegt unter anderem daran, dass Angriffe auch von vertrauenswürdigen Benutzern ausgeführt werden können. Hier kann die Angriffsabsicht schlecht oder überhaupt nicht bewiesen werden. Die einfache, intensive Verwendung eines Dienstes kann schon zu Zuständen führen, die von denen eines DoS-Angriffs nicht zu unterscheiden sind. Zudem könnten diese Angriffe ebenso völlig anonym und mit sehr einfachen Mitteln durchgeführt werden. Für die Praxis wesentlich ist vor allem die Tatsache, dass es momentan keinerlei präventive Maßnahmen gegen DoS-Angriffe ohne Einschränkungen des Betriebs gibt. Eine ausführliche Diskussion und Klassifizierung von DoS-Gegenmaßnahmen findet sich beispielsweise in [HHP03].

Fazit

In diesem Abschnitt wurden drei konkrete Angriffe beschrieben, von denen jeder eines der drei allgemeinen Schutzziele verletzt. Ein Angriff durch Pufferüberläufe verletzt das Schutzziel der Vertraulichkeit, oft auch die Schutzziele Verfügbarkeit und Integrität, und stellt meist nur einen ersten Schritt als Einstieg in ein fremdes System dar, um weitere Angriffe durchzuführen. Ein Denial-of-Service Angriff ist die klassische Form der Verletzung der Verfügbarkeit und ein Angriff von Viren und Würmern kann die Integrität eines Systems verletzen.

Die erwähnten Attacken stellen selbstverständlich nur einen kleinen Ausschnitt der bekannten Möglichkeiten von Angreifern dar, sie zeigen aber die Komplexität, die sich bei der Vermeidung von Schwachstellen und Angriffen ergibt. Oft treten die Angriffe nicht einzeln auf, sondern Teilangriffe ergänzen sich zu einer großen Bedrohung. Bei der Sicherheitsanalyse müssen also neben der Kenntniss der einzelnen möglichen Angriffe auch die Zusammenhänge möglicher Angriffe erkannt werden.

2.2.2.3 Bedrohungen bei der Durchführung von Gegenmaßnahmen

Sicherheitsbedrohungen eines Systems können nie völlig ausgeschlossen werden. Daher ist es wesentlich, dass zur Laufzeit des Systems auf auftretende Bedrohungen, Angriffe und bekanntwerdende Schwachstellen reagiert wird. Beim Auftreten einer bisher unbekannten Gefährdung des Systems oder eines Angriffs müssen Gegenmaßnahmen gestartet werden. Das erste Auftreten des Angriffs und seine Folgen können so nicht verhindert werden, allerdings müssen weitere Angriffe der selben Art abgehalten werden. Von der Geschwindigkeit und Qualität dieser Reaktion hängt es ab, ob ein Angriff mehrmals erfolgreich ist.

Administrative Fehler

Der weite Bereich der administrativen Fehler ist allgemein kaum zu fassen. Die Möglichkeiten gehen von einer vergessenen offenen Befehlszeile mit Hauptbenutzerrechten an einer Konsole über den Verlust von Sekundärspeichermedien (Bänder, CDs usw.) bis hin zu der meist unabsichtlichen Weitergabe von Passwörtern. Tatsächlich kann diesen Fehlern entweder durch strikte Vorschriften begegnet werden, was aber keine präventive Maßnahme darstellt. Oder man muss das Verhalten der jeweiligen Umgebung individuell anpassen und beispielsweise Verhaltenskataloge nutzen, wie sie etwa vom BSI herausgegeben werden. Eine rein technische Lösung dieser Probleme ist jedoch unmöglich, da der Mensch eine wesentliche Rolle dabei spielt und selbst eine große Schwachstelle darstellt.

Angriffe beim Datenmanagement

Das Datenmanagement beschreibt alle Aktionen, die die Integrität und Verfügbarkeit der Daten eines Systems sicherstellen. Wesentliche Bestandteile sind die Backupstrategien und sämtliche Arten der Integritätsprüfung der Daten. Werden bei diesen Aufgaben Fehler gemacht, so ist sowohl die Integrität und Verfügbarkeit als auch die Vertraulichkeit der Daten betroffen.

Angriffe beim Lebenszeitmanagement

Unter dem Lebenszeitmanagement wird jeder Vorgang verstanden, der dem weiteren Betrieb von Software behilflich ist. Dieses umfassende Thema besteht heute im Wesentlichen aus dem Einbringen von Fehlerkorrekturen in existierende Software (Softwareupdate) und der Pflege der Konfiguration dieser Systeme. Fehler während dieser administrativen Aufgaben können Systeme auf der Betriebssystemebene, wie auf Anwendungsebene kompromitieren.

2.3 Klassifikation von Bedrohungen und Angriffen

In diesem Abschnitt werden Bedrohungen modelliert, so dass die bekannten Angriffsklassen des vorigen Abschnittes durch dieses Modell beschrieben werden können. Ziel ist es dabei, ein Modell zu entwickeln, so dass während der Entwicklung eines Systems einzelne Systemteile mit bekannten Bedrohungsmustern verglichen werden können und das System damit auf das mögliche Vorhandensein von Bedrohungen getestet wird. Andere Versuche Modelle für Angriffe zu finden, sind beispielsweise in [MEL01] und [PS98] zu lesen. Zunächst besteht das Modellbild ganz allgemein aus einem IT-System (für diese Arbeit ein Mobiles Verteiltes System) und einem weitgehend unbekannten, allgemeinen Angreifer. Zu beachten ist dabei die besondere Ausprägung der Angriffe in der IT-Sicherheit, dass der Angriff beispielsweise gleichzeitig von verschiedenen Orten aus durchgeführt werden kann. Eine Diskussion über Sicherheitsmodelle findet sich unter anderem in [McL87], [McL94], [McL90b] und [McL90a].

2.3.1 Phasen

Wie beispielsweise in [Amo94] beschrieben ist, gibt es nur drei Klassen auftretender Bedrohungen, die allgemein angewandt werden: die unerlaubte Informationserreichbarkeit (engl. Disclosure Threat), die Verletzung der Datenintegrität (engl. Integrity Threat) und die Behinderung der Verfügbarkeit (engl. Denial-of-Service Threat). Im Folgenden werden die Phasen beschrieben, in die man Angriff und Verteidigung grundsätzlich einteilen kann.

2.3.1.1 Der Angriff

Der Angriff, der auf eine vorhandene Bedrohung abzielt, kann in drei zeitliche Phasen eingeteilt werden.

1. Vorbereitungsphase

   Die Vorbereitungsphase wird wiederum in die einzelnen Schritte aufgeteilt, die der Angreifer zur erfolgreichen Durchführung benötigt. Beeinflussende Schritte sind Aktionen, die vom Angreifer ausgehen und das Zielsystem des Angriffs und seine Umgebung verändern. Nicht beeinflussende Schritte dagegen bedeuten die reine Informationsgewinnung, wie etwa durch Abhören oder Beobachten. Die Vorbereitungsphase ist dadurch gekennzeichnet, dass deren einzelnen Schritte nicht als Angriff gewertet werden können.

2. Durchführungsphase

   In der Durchführungsphase versucht der Angreifer eines oder mehrere Schutzziele zu verletzen. Auch diese Phase wird in Unterphasen aufgeteilt und kann aus nicht beeinflussenden und beeinflussenden Schritten bestehen. Im Gegensatz zur Vorbereitungsphase kann aus der Kenntniss dieser Schritte ein Angriff erkannt werden.

3. Konsequenzen

   Nachdem ein Angriff erfolgreich durchgeführt wurde, kann der Angreifer die Ergebnisse nutzen. Das können einerseits gestohlene Daten sein, die in dieser Phase beliebig kopiert und weitergegeben werden, es kann sich aber auch um Zugänge zu Systemen handeln, ohne dass ein direkter Nutzen sichtbar ist.

Die drei Phasen des Angriffs bedeuten auch für das Opfer gewisse Möglichkeiten zur Erkennung der Vorbereitung und der Durchführung des Angriffs und zur Reaktion auf den Angriff. In der Vorbereitungsphase des Angriffs könnte das potentielle Opfer Beobachtungen registieren und erkennen, wenn Tätigkeiten auf das Suchen von Schwachstellen hinweisen. Das Erkennen eines Angriffs in der Durchführungsphase ermöglicht dem Opfer eine schnelle Reaktion und eine mögliche Identifizierbarkeit und Verfolgbarkeit des Täters. Nach einem erfolgreichen Angriff sind für das Opfer die Schäden meist nur mit hohem organisatorischem Aufwand zu begrenzen. Oft ist nicht einmal vollständig klar, welche Daten kompromitiert wurden, so dass davon ausgegangen wird, dass alle Daten betroffen sind.

Es ist möglich, dass nur Teile von Aktionen in den einzelnen Phasen wirkungsvoll sind und vom Angreifer durchgeführt werden. Dies kann beispielsweise dann auftreten, wenn die Vorbereitung oder die Durchführung des Angriffs erkannt und zur Laufzeit behindert wurde. Dann ist vom konkreten Fall abhängig, ob der Angreifer bereits erfolgreich war oder ob er rechtzeitig von einer Schutzzielverletzung abgehalten wurde. Sind konkrete Angriffe mit allen Phasen bekannt, so ist es sinnvoll, sie als Baum darzustellen, wie in Abbildung

2.10 gezeigt wird. Das Ziel des gesamten Angriffs wird dabei in der Wurzel des Baums beschrieben. Es folgen die drei Phasen mit den einzelnen Teilschritten.

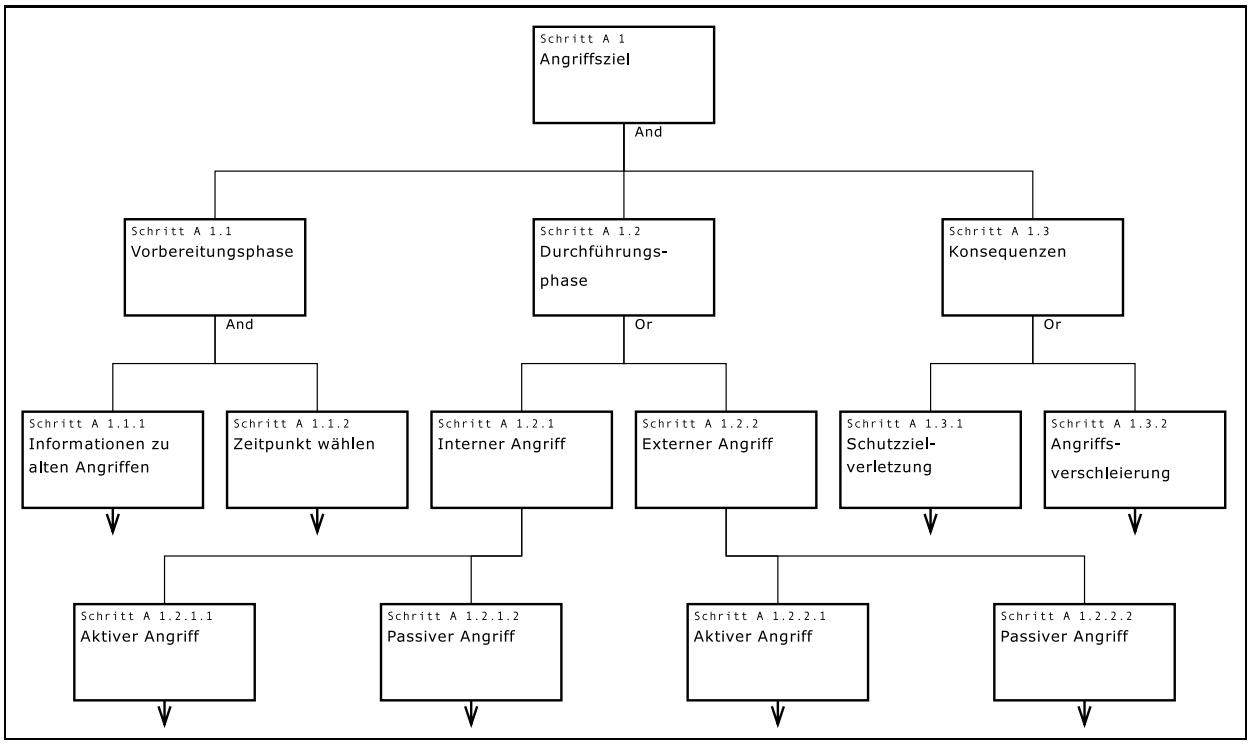

Abbildung 2.10: Phaseneinteilung allgemeiner Angriffe

Zudem ist es möglich, dass der Angriff ohne weiteres Ziel durchgeführt wird. Die Vorbereitungs- und Durchführungsphase wird für einen ernsthaften Angriff durchgeführt, es folgen aber keinerlei Konsequenzen. Obwohl dies sehr unüblich aussieht, sind derartige Angriffe heute gängig. Angriffe durch sogenannte Script-Kiddies, die eher aus Spieltrieb durchgeführt werden, als ernsthafte Konsequenzen bewirken zu wollen, sind heute der Großteil der bekannt werdenden Angriffe. Auch wenn dieser Begriff relativ harmlos klingt, sind die Schäden durch diese Angriffe extrem hoch.

Ein Ziel eines Angreifers ist es, während der Beobachtung einer Attacke, also bei der Vorbereitungs- und Durchführungsphase, nicht erkannt zu werden. Erst mit dem Bekanntwerden der Konsequenzen wird das Ziel klar. Zudem kommt hinzu, dass das Ziel des Angriffs tatsächlich oft überhaupt nicht feststeht. Wie der Fehler eines Fortran-Programms, der im vorigen Abschnitt dargestellt wurde, kann teilweise nicht einmal im Nachhinein festgestellt werden, ob der Fehler aus Versehen oder absichtlich eingefügt wurde und welches das eigentliche Ziel des Angreifers gewesen sein kann.

Grundsätzlich gilt, dass innerhalb und nach jeder der drei Phasen ein Angriff abgebrochen und ein erneuter Versuch durchgeführt werden kann. Bei Angriffen auf IT-Systeme kann

dies beliebig wiederholt werden und zunächst wirkungslos bleiben. Auf diese Weise lassen sich Teilangriffe testen, meist ohne erkannt zu werden.

2.3.1.2 Die Verteidigung

Wie eine Phaseneinteilung allgemeiner Angriffe möglich ist, so werden im folgenden Abschnitt die Möglichkeiten der Verteidigung ebenfalls in drei zeitliche Phasen aufgeteilt.

1. Vermeidung und Verhinderung (engl. Prevention und Avoidance)

   Es ist wünschenswert, schon in der Designphase bevor Angriffe möglich werden, wirkungsvolle, präventive Maßnahmen einzusetzen, die bestimmte Angriffe prinzipiell vermeiden. Es existieren einige Klassen von Gegenmaßnahmen, die, wenn sie korrekt eingesetzt werden, dies tatsächlich gegen spezielle Angriffe leisten können. Allerdings können im Allgemeinen vorab nicht alle möglichen Bedrohungen und Angriffe erkannt werden. Deshalb ist die Verhinderungsphase nötig und sinnvoll, muss aber durch die folgenden Phasen ergänzt werden.

2. Angriffserkennung (engl. Detection)

   Die Angriffserkennung beschreibt alle technischen und administrativen Dienste, die zur Laufzeit eines Systems Angriffe erkennen können. Zum einen existieren sowohl Mechanismen, die Angriffsmuster automatisch erkennen, wie beispielsweise IDS-Systeme. Zum anderen überwachen Administratoren Netzwerke und kritische Systeme zur Laufzeit.

3. Reaktion auf einen Angriff (engl. Reaction)

   Nachdem ein Angriff erkannt wurde, muss in geeigneter Weise reagiert werden. Diese Reaktion kann in verschiedenster Form geschehen, wie ein Abschalten des Systems oder die Identifizierung des Angreifers. Auch wenn bereits erste Versuche zur automatischen Reaktion existieren, gibt es in diesem Bereich die wenigsten Verfahren.

Es ist davon auszugehen, dass die Vorbereitungsphase eines Angriffs notwendig ist. Deshalb sind die einzelnen Verteidigungsschritte in jeder Phase des Angriffs wirkungsvoll. Bei der Angriffsvorbereitung können präventiv öffentliche Informationen geschützt werden, die dem Angreifer eine Hilfe sind. Darunter sind beispielsweise Daten zu verstehen, die über das potentielle Zielsystem, über Kommunikationswege oder über die Benutzer Wissen vermitteln. Zudem kann die Überwachung des Systems und der Benutzer erschwert werden. Sind bestimmte Schritte bekannt, die ein Angreifer während der Angriffsvorbereitung durchführen muss, wie beispielsweise das Herunterladen spezieller Software, so kann dies erkannt und darauf reagiert werden. Wird die Durchführung erkannt und werden die Konsequenzen klar, kann beispielsweise der Nutzen des Angreifers durch gestohlene Daten behindert werden.

2.3.2 Charakteristika von Angriffen in den einzelnen Phasen

Zur Verfeinerung der allgemeinen Angriffsbeschreibung werden wesentliche Eigenschaften der einzelnen Phasen in der Tabelle 2.2 aufgezählt. Diese sind durch die Analyse bestehender Angriffe entstanden, die in den vorigen Abschnitten zusammengefasst wurde. Die einzelnen Eigenschaften sind der Ort des Angreifers und des Ziels, die beteiligten Subjekte und Objekte, der speziell notwendige Kontext für die Durchführung des Angriffs, die Beschreibung des Ablaufs in den entwickelten drei Phasen, die Intention des Angreifers und mögliche Schwachstellen des Ziels. Zu den beteiligten Subjekten gehören auch eingesetzte Mittel wie ein Budget oder das erforderliche Know-How des Angreifers. Alle diese charakteristischen Punkte sind nicht unbedingt orthogonal zueinander, enthalten aber zumindest alle Informationen um die bekannten Angriffe eindeutig zu beschreiben.

Eigenschaft	Bedeutung
LOC_T	Ort des Ziels
LOC_A	Ort des Angreifers
SUB_A	Beteiligte Subjekte des Angreifers
SUB_T	Beteiligte Subjekte des Angriffsziels
OBJ_A	Beteiligte Objekte des Angreifers
OBJ_T	Beteiligte Objekte des Angriffsziels
CON_A	Kontext, Randbedingungen für den Angriff
TIM_A	Zeitaspekt und Abläufe
INT_A	Absicht und Ziel der Aktion
VUL_T	Schwachstellen des Angriffsziels

Tabelle 2.2: Klassifikationsschema

Je nach Angriff besitzen die einzelnen Eigenschaften unterschiedliche Ausprägungen, so dass damit eine Angriffsbeschreibung möglich ist. Mit Hilfe dieser Beschreibungen über Angriffe werden Abwehrmaßnahmen ausgewählt, indem man die für den Angriff verwendeten Eigenschaften durch die Gegenmaßnahme behindert oder deren Verfügbarkeit beeinträchtigt. Eine Möglichkeit wäre etwa, Mechanismen einzusetzen, die die beteiligten Subjekte und Objekte des Angreifers zerstören und dem Angriff so die nötigen Mechanismen zu rauben.

Ein weiterer Punkt für den Einsatz der oben beschriebenen Eigenschaften ist die Suche nach neuen Angriffsklassen. Gegeben sei ein System, das auf mögliche Angriffe zu untersuchen ist. Das System ist dazu in einer beliebigen, handhabbaren Form vorhanden, wie etwa in

Form eines Systemmodells oder durch vorhandenen Quelltext[3]. Es ist möglich, ohne das exakte Wissen über Angriffe die einzelnen Eigenschaften des Systems zu testen und so Schwachstellen zu finden. So ist es beispielsweise möglich, ein System auf einen Angriff hin zu untersuchen, bei dem im Wesentlichen der Ort des Angreifers LOC_A bekannt ist und der die beteiligten Objekte des Angriffsziels OBJ_T ausspioniert. Ohne dass dieser Angriff eine konkrete Form oder einen Namen hat, kann ein System auf diese Verletzbarkeit getestet werden.

2.3.2.1 Vorbereitung und Ursprung

Bei bisherigen Ansätzen zur Bedrohungsanalyse, die in Kapitel 5 beschrieben werden, wurde die Vorbereitungsphase nicht gesondert betrachtet. Umgangssprachlich wurde gesagt, dass der Angreifer dabei festlegt, woher die Angriffsdurchführung kommt (Eigenschaft LOC_A) und auf welchem Weg sie stattfinden wird (Eigenschaften SUB_T und OBJ_T). Mit den oben beschriebenen Eigenschaften ist nun eine detailiertere und systematische Beschreibung möglich. Grundsätzlich werden in dieser vorbereitenden Phase alle Eigenschaften weitgehend festgelegt. Der Angreifer wird das Zielsystem (LOC_T) und seinen Standort (LOC_A) aussuchen, er bereitet alle Subjekte und Objekte auf die Durchführung vor (SUB_A, OBJ_A), lernt Schwachstellen des Ziels kennen (VUL_T, CON_A) und analysiert Abläufe (TIM_A).

2.3.2.2 Durchführung

Während der Durchführung ändern sich die Eigenschaften kaum noch. Sind mobile Systeme beteiligt, so kann sich sowohl der Ort des Ziels (LOC_T) als auch des Angreifers (LOC_A) ändern. Für einen wirkungsvollen Angriff bleiben jedoch die Subjekte und Objekte (SUB_A, OBJ_A) konstant.

2.3.2.3 Konsequenz und Ziele des Angreifers

Als Folge wird der Angreifer versuchen, den Angriff zu verschleiern. Dazu kann er versuchen, seinen Standort LOC_A zu verändern. Daneben kann das Rückverfolgen des Angriffs behindert werden, indem alle am Angriff beteiligten Subjekte SUB_A und Objekte OBJ_A entfernt werden. Datenobjekte OBJ_T des Angriffsziels oder Kopien davon können im Besitz des Angreifers sein. Ebenso können OBJ_T und SUB_T vom Angriff beschädigt sein, beispielsweise können neue Schwachstellen VUL_T in das Zielsystem eingebracht worden sein.

[3]Bei der später entwickelten Analyse in Kapitel 6 ist aber ein deutlicher Unterschied erkennbar, in welcher Form das System beschrieben wird.

2.4 Allgemeine Sicherheitsmaßnahmen

Es existieren zahlreiche Gegenmaßnahmen, die wirksam gegen bestimmte Klassen von Angriffen sind. Sie werden im Folgenden beschrieben, um daraus einen Maßnahmenkatalog zu entwickeln. Verschiedene Maßnahmen werden auch in [Lan83] verglichen. Die Sicherheitsmaßnahmen können prinzipiell in den drei Phasen des Angriffs wirksam sein, präventiv bei der Vorbereitung, erkennend während des Angriffs und reaktiv nach dem Angriff. Für den ersten Bereich, den verhindernden Maßnahmen, existieren zahlreiche Konzepte. Im erkennenden und reaktiven Bereich gibt es dagegen heute kaum Gegenmaßnahmen, so dass sich diese Gliederungsstruktur nicht eignet. Deshalb werden sie im Folgenden nach der Art der Umsetzung geordnet dargestellt. Neben den einzelnen Konzepten und Mechanismen werden auch die Angriffsklassen angegeben, gegen welche sie wirksam sind. Basis hierfür sind die Eigenschaften von Angriffen, wie sie im letzten Abschnitt beschrieben wurden.

Die Aufzählung ist nicht vollständig, sondern soll nur einen großen Bereich bekannter Mechanismen abdecken, damit die gewählte Strukturierung plausibel wird. Dadurch lässt sich der Mechanismenkatalog später leicht durch andere, vielleicht neu entstandene Verfahren erweitern und die Methodik weiterhin nutzen.

2.4.1 Konstruktionsmethoden für sichere Systeme

In [Amo94] werden vier Teilprozesse bei der Konstruktion sicherer Software festgelegt. Diese Teile sind für alle Entwicklungsmethoden aus dem Security-Engineering Bereich gültig, die heute eingesetzt werden. Unabhängig vom Vorgehensmodell wird auch die Reihenfolge dieser Teilprozesse stets so eingehalten.

1. *Bestimmung der Sicherheitsanforderungen:* In diesem Schritt werden die Sicherheitsanforderungen definiert. Sie ergeben sich einerseits aus den intuitiven Bedürfnissen der Benutzer und Administratoren des Systems. Darunter fallen beispielsweise Anforderungen an die physikalische Umgebung des Systems, wie etwa das Aufstellen in einem eigenen Raum, oder eine regelmäßige Abfrage des Passworts und die Visualisierung der aktuellen Sicherheitseigenschaften. Andererseits werden dabei auch technische Sicherheitsanforderungen aufgestellt, wie der regelmäßige Austausch von Benutzerpasswörtern oder der Einsatz von abgesicherten externen Datenträgern.

   Nicht bestimmt werden können Anforderungen, die erst durch eine Schwachstellenanalyse des gesamten, fertiggestellten Systems oder durch auftretende Bedrohungen während des Betriebs entdeckt werden. So kann beispielsweise der Einsatz eines kryptografischen Protokolls problematisch werden, wenn Schwachstellen in dem kryptografischen Verfahren entdeckt werden. Moderne Entwicklungsmethoden bezeichnen deshalb die Bestimmung der Sicherheitsanforderungen als iterativen Prozess, der letztlich auch während des Betriebs des Systems weiter durchgeführt werden muss.

2. *Auswahl der Sicherheitsverfahren und Mechanismen:* Durch die Auswahl der eingesetzten Sicherheitsverfahren wird die Systemarchitektur beeinflusst beziehungsweise teilweise festgelegt. Dabei soll auf eine erweiterbare Bibliothek von Mechanismen zugegriffen werden, damit diese wiederverwendet werden und die Entwicklung und Prüfung der Verfahren nicht mehrmals durchgeführt wird. Es ist nicht immer möglich, bei bekannter Anforderung genau ein Verfahren eindeutig auszuwählen. Meist kommen mehrere in Frage, wie beispielsweise bei der Auswahl des einzusetzenden symmetrischen oder asymmetrischen krytografisches Verfahrens. Selbst bei iterativem Vorgehen können die Verfahren meistens nicht eindeutig gewählt werden, es bleibt dann immer den Softwarearchitekten und Entwicklern vorbehalten, eines auszuwählen und diese Entscheidung zu begründen.

3. *Beweis der Sicherheitseigenschaften:* Es wäre beim Einsatz eines Systems wünschenswert, wenn die Sicherheitseigenschaften so dokumentiert sind, dass für einen Nichtentwickler klar erkennbar ist, dass sie korrekt sind und die Sicherheitsanforderungen erfüllen. Es existieren nur wenige Methoden, die in diese Richtung gehen, allerdings hat keine bisher damit Erfolg. Durch Tests können zwar bestimmte Eigenschaften nachgewiesen werden, es ist jedoch sehr schwer, damit das Nichtvorhandensein von Eigenschaften, wie etwa Schwachstellen, zu zeigen. Der Einsatz von Systemen und die Abhärtung durch konsequente Analyse der Angriffe und Weiterentwicklung kann nicht in sicherheitskritischen Bereichen durchgeführt werden. Hier ist der Einsatz von Methoden gewünscht, die von vorneherein bestimmte Sicherheitseigenschaften garantieren können. Das Gebiet der Formalen Methoden könnte in seiner Gesamtheit einen verlässlichen Weg dafür darstellen, allerdings hat es bis heute bei der Entwicklung großer Systeme keine großen Erfolge zeigen können. Es ist prinzipiell möglich, funktionale Eigenschaften und auch deren korrektes Verhalten zu zeigen. Das Zeigen des Nichtvorhandenseins von Eigenschaften jedoch, wie es bei der Prüfung von Sicherheitseigenschaften nötig wäre, gelingt heute nicht. Ein Beispiel für den Aufwand alleine der zu Grunde liegenden Spezifikation kann in [WSO+75] nachgelesen werden. Es existiert eine Fülle von Papieren, die die Verifikation von Sicherheitseigenschaften in Betriebssystemen behandeln, wichtige sind [CGHM81], [Mil76], [Moo90], [Neu73], [WKP80], [PF78], [Rus81] und [Sil83].

4. *Nachrüsten der Sicherheit:* Als Folge davon, dass der Beweis von Sicherheitseigenschaften nicht gelingt, muss man Sicherheitsmechanismen und die Ausbesserung von gefundenen Fehlern und Schwachstellen während des Betriebs durchführen. Es hängt stark vom jeweiligen möglichen Angriff ab, ob diese Nachbesserung einfach durchzuführen ist, oder Teile des Systems neu entwickelt werden müssen und sie so einen hohen Aufwand darstellt. Zudem ist es auch nötig, Sicherheit in bestehende Systeme nachzurüsten, soweit dies möglich ist. Hauptgrund dafür ist das Vorhandensein von großer Software, die nicht mit vertretbarem Aufwand durch eine abgesicherte Neuentwicklung, wie es nötig wäre, ersetzt wird. Einerseits spricht der hohe Aufwand, andererseits die Unterbrechung des Betriebs dagegen.

Die in dieser Arbeit entwickelte Methode berücksichtigt alle erwähnten Punkte, so dass sowohl die Sicherheitsanforderungen und die eingesetzten Mechanismen nachvollziehbar und die Sicherheitseigenschaften des Systems verständlich bleiben.

2.4.1.1 Designprinzipien

In [SSd75] werden unter anderem 10 Designprinzipien für sichere Systeme entwickelt. Im Gegensatz zu den ersten 8 Prinzipien sind die letzten beiden aus dem Gebiet der physikalischen Sicherheit übernommen und lassen sich nur bedingt auf den Bereich der IT-Sicherheit anwenden. In der deutschsprachigen Literatur werden die einzelnen Prinzipien teilweise übersetzt, teilweise auch im Englischen belassen, wobei es je nach Quelle jeweils unterschiedliche Übersetzungen gibt. Im Folgenden werden die Begriffe deshalb im Englischen belassen.

1. *Economy of mechanism:* Das Design der angewendeten Gegenmaßnahmen soll so einfach und klein wie möglich gehalten werden. Dadurch wird sowohl die Benutzbarkeit und Benutzerakzeptanz erhöht als auch die Überprüfung des Systems erleichtert.

2. *Fail-safe defaults:* Zugriffskontrollen sollen den Zugriff auf Objekte durch Prüfung der Policy erlauben, nicht verbieten. Die Alternative dazu wäre, dass Kontrollen bestimmten Subjekten einen expliziten Zugriff verbieten, grundsätzlich aber Zugriffe erlaubt sind. Es ist einfacher zu prüfen, ob die Zugriffskontrollen korrekt sind, wenn zunächst alle Zugriff verboten sind, nur die erlaubten nach Prüfung zugelassen werden, als umgekehrt.

3. *Complete mediation:* Jeder Zugriff muss stets auf Basis der aktuellen Policies geprüft werden. Dieses Prinzip stellt die Grundlage für jede Art von Zugriffsschutz dar. Mechanismen, die zur Verbesserung der Geschwindigkeit Ergebnisse von Zugriffskontrollen zwischenspeichern, müssen bei Änderung der Rechte stets die aktuellen Werte kennen.

   Interessanterweise wird genau dieses Prinzip von herkömmlichen Unix-Dateisystemen nicht eingehalten. Dies ist für die Analyse des abgesicherten Dateisystems wichtig, das in Kapitel 8 diskutiert wird. Deshalb wird die Verletzung des *Complete mediation*-Prinzips im Folgenden kurz an einem Beispiel gezeigt.

Fallbeispiel: *Unix-Dateisysteme und das Complete mediation-Prinzip:*
In einem Unix-System arbeiten 2 Benutzer zu selben Zeit, Benutzer R ist Administrator und Benutzer U normaler Anwender, dessen aktuelles Arbeitsverzeichnis sich irgendwo unterhalb des Verzeichnisses `/U/working/` befindet. Benutzer U hat Lese- und Schreibrechte für alle Verzeichnisse einschließlich und unterhalb von `/U/`. Zu einem Zeitpunkt t_1 nimmt der Adminstrator dem Benutzer U die Rechte des Verzeichnisses `/U/` weg (entfernt beispielsweise die Lese- und Schreibberechtigung für alle Benutzer) und glaubt, dass kein Benutzer mehr in einem Verzeichnis unterhalb von `/U/` arbeiten kann.
Das ist eine falsche Annahme. Der Benutzer U, der sich bereits in einem Unterverzeichnis unterhalb von `/U/working/` befindet, kann ohne Beeinflussung auch zu Zeitpunkten $t > t_1$ weiterarbeiten, obwohl dadurch das *Complete mediation*-Prinzip eindeutig verletzt wird. Dies kommt daher, dass ein Unix-Dateisystem nur die Rechte des direkt übergeordneten Verzeichnisses beim Zugriff prüft und diese sind in dem Beispiel ausreichend. Benutzer U kann deshalb solange in dem Verzeichnis und Unterverzeichnissen weiterarbeiten, solange er es nicht verlässt. Ein Wechsel in das Verzeichnis ist allerdings nach dem Zeitpunkt t_1 nicht möglich. Ursache für den Effekt ist die Effizienz der Kontrollen, deren Zeitaufwand bei einer ständigen Prüfung aller übergeordneten Verzeichnisse sehr hoch werden würde. Standard Unix-Dateisysteme bieten dafür keine Lösung.

4. *Open design:* Die Sicherheit der Verfahren darf nicht auf deren Geheimhaltung beruhen. Nur Variablen für die Verfahren stellen das Geheimnis dar, wie etwa Passwörter, private Schlüssel und PINs oder persönliche Merkmale. Dadurch wird es ermöglicht, dass jeder die Sicherheitsmechanismen eines Systems kontrollieren kann und so das Vertrauen in ein System gesteigert wird, ohne dass der Schutz der Benutzer verloren geht.

   Open Source Software liefert die Einhaltung dieses Prinzips immer, trotzdem erhält man dadurch natürlich nicht automatisch sichere Software, wie es oft in Diskussionen angedeutet wird. Eine Diskussion darüber findet sich auch in C.

5. *Separation of privilege:* Wenn sinnvoll und möglich, sollen die Geheimnisse zur Gewährung von Zugriffen auf mehrere Subjekte verteilt werden. Das Versagen eines Subjektes hat dann nicht sofort den Verlust der Datensicherheit zur Folge.

6. *Least privilege:* Jedes Subjekt soll nur genau die Rechte besitzen, die es zur Erfüllung seiner jeweiligen Aufgaben tatsächlich benötigt. Viele der heutigen Angriffe nutzen die Nichteinaltung dieses Prinzips aus, um beispielsweise Rechte des Administrators zu erhalten. Das Vorhandensein eines im System allmächtigen Superusers ist ein klassisches Beispiel für die Verletzung dieses Prinzips.

7. *Least common mechanism:* Es sollen möglichst wenige Mechanismen existieren, die für die Kontrolle mehrerer Subjekte gleichzeitig verwendet werden und die von allen Subjekten abhängen. Jeder solche Mechanismus kann Angreifern als ungewollter Informationspfad dienen. Zudem müssen Mechanismen, die für alle Benutzer verwendet werden, auch entsprechend flexibel und anpassbar sein können. Dies kann beispielsweise wiederum dem Prinzip *Economy of mechanism* entgegenstehen.

8. *Psychological acceptability:* Wesentlich für die Wirksamkeit der Schutzmechanismen ist die Benutzbarkeit. Dabei spielt es ebenso eine Rolle, dass die Konfiguration und Bedienung einfach ist, wie es auch wesentlich ist, dass der Benutzer seine Schutzbedürfnisse ohne nenneswerte Verluste auf das System abbilden kann, die Bedienung also weitgehend intuitiv ist.

9. *Work factor:* Bei der Konstruktion sollen stets der Aufwand eines Angriffs und die zur Verfügung stehenden Ressourcen des Angreifers verglichen werden. Die derart durchgeführte Risikoanalyse kann das Gefährdungspotential bestimmter Angriffe verändern. Allerdings ist es oft nicht leicht möglich, den Aufwand eines Angriffs zu quantifizieren, da die Durchführung auch von unsicheren Faktoren abhängt, wie etwa Schwachstellen im Code oder Fehler bei der Administration.

10. *Compromise recording:* Bei jedem erkannten Informationsverlust muss ein nicht mehr zu entfernendes Signal gesetzt werden. Dadurch kann zwar der Informationsverlust selbst nicht verhindert werden, jedoch ist der Schaden danach bekannt. Es ist allerdings sehr schwer in IT-Systemen einerseits den Informationsverlust überhaupt zu entdecken, andererseits ein vom Angreifer nicht manipulierbares Signal zu setzen, da der Angreifer das System meist unter seiner Kontrolle hat.

Es gibt zahlreiche andere Versuche, Designprinzipien dieser Art festzulegen oder die genannten zu erweitern. Das Besondere an den in [SSd75] aufgeführten im Gegensatz zu den anderer Quellen ist jedoch, dass sie jeweils unabhängig voneinander sind und damit je eine eigene Dimension bei der Qualifizierung von Sicherheitsarchitekturen eröffnen.

2.4.1.2 Allgemeines Vorgehensmodell

Eine frühe Beschreibung für das Vorgehen bei der Entwicklung sicherer Systeme findet sich in [Amo94]. Dieses als „Systems Security Engineering“ bezeichnete, sehr allgemeine Verfahren liefert die Grundlage für die meisten der heute angewendeten Vorgehensmodelle. Wesentlich dabei ist die frühe Identifizierung der möglichen Gefahren auf das System.

Erster Schritt bei der Entwicklung muss die Festlegung der Architektur des Systems sein, da dessen Funktionalität im Vordergrund des Designs steht. Naheliegende Angriffe können durch den Einsatz von bekannten Sicherheitsverfahren bereits pauschal abgewehrt werden,

ohne dass man besondere Sicherheitsanalysen dafür benötigt. Beispielsweise wird die Integrität und Vertraulichkeit der Kommunikation durch den Einsatz von kryptografischen Verfahren abgesichert, was schon zu Beginn des Systemdesigns festgelegt werden kann. Anschließend werden anhand der vorgelegten Systemarchitektur die möglichen Bedrohungen, Schwachstellen und Angriffe gesucht und identifiziert. Dieser wesentliche Schritt wird als intuitiver Prozess angegeben, der durch die Entwickler durchgeführt werden kann. Es ist allerdings zum Einen fraglich, ob ein Entwickler ohne Sicherheitskenntnisse dafür geeignet ist. Zum Anderen treten durch intuitive Vorgehen die bereits erwähnten Probleme auf, wie etwa die geringe Nachvollziehbarkeit und die geringe Chance möglichst nahe an eine vollständige Liste der Bedrohungen zu gelangen.

Im folgenden Schritt werden die Risiken der einzelnen Komponenten diskutiert und abgeschätzt. Fällt das Gesamtrisiko unter einen festgelegten Wert, so geht das System in die nächste, in diesem Vorgehensmodell nicht näher festgelegten Stufe der Entwicklung, wie beispielsweise dem Testen des Systems. Ansonsten werden die potentiellen Gefährdungen priorisiert und für die wichtigen davon Gegenmaßnahmen festgelegt. Diese Maßnahmen werden anschließend im erneuten Durchlauf des ersten Schritts, der Festlegung bzw. der Erweiterung der Systemarchitektur integriert. Der dadurch entstehende Zyklus im Vorgehen kann beliebig oft wiederholt werden, was im Wesentlichen durch ein Gleichgewicht aus den Kosten und der Komplexität des Systems begrenzt wird.

Da das Vorgehen auf die Entwicklung eines neues Systems beschränkt ist, können keine neuartigen Angriffe verhindert werden. Es ist nur im Bereich der präventiven Maßnahmen wirksam und wird in Tabelle 2.3 klassifiziert. Für den Bereich der erkennenden und der reaktiven Maßnahmen bietet diese Methode keine Hilfe und es ist nicht zur Abhärtung bestehender Systeme entwickelt worden.

2.4.1.3 Pattern-orientierter Ansatz

Oft wird eine allgemeine Beschreibung bei der Konstruktion einzelner Gegenmaßnahmen derart vorgeschlagen, dass mit dieser Beschreibung die Maßnahme später in ähnlichen Situationen wiederverwendet werden kann. Die Schwierigkeit ist dabei, die Beschreibung so darzustellen, dass andere Entwickler die Gemeinsamkeiten verstehen können, ohne den Mechanismus der Maßnahme genau zu verstehen. Im Teilbereich der Kryptografie wird dieses Vorgehen bereits intuitiv angewandt, da es für den Entwickler meist viel zu aufwändig wäre, die genauen mathematischen Hintergründe der Kryptoverfahren zu verstehen. Er kennt lediglich Gruppen verschiedener Verfahren und dem jeweiligen Einsatz und wendet sie an. Der Pattern-orientierte Ansatz verfolgt das Ziel, diese Wiederverwendbarkeit auch für andere Verfahren zum Schutz der Systeme, außer der Kryptografie zu erreichen. Dadurch kann auch eine Aufteilung der Aufgaben der Entwicklung der eigentlichen Funktionalität und der Entwicklung von Sicherheitsteilsystemen und Komponenten erreicht werden. Die hauptsächliche Schwierigkeit dabei ist allerdings, eine Sprache zur Beschreibung der Pattern zu finden, die von allen Entwicklern verstanden wird und eingesetzt werden kann.

Eigenschaft	Ausprägung bei *Prävention*
LOC_T	Einsatzumgebung des Systems nicht bekannt
LOC_A	Interner Angreifer oder bekannter externer Angreifer
SUB_A	System, nicht die Einsatzumgebung
SUB_T	Subjekte des Systems oder aus der Umgebung bekannt
OBJ_A	System, nicht die Einsatzumgebung
OBJ_T	Objekte des Systems oder aus der Umgebung bekannt
CON_A	Schutz vor vorhersehbaren Angriffen
TIM_A	Gesamte Entwicklungszeit
INT_A	Unbekannt
VUL_T	Entwickler kann Angreifer sein

Tabelle 2.3: Prävention durch allgemeines Vorgehensmodell

Die UML, wie sie beispielsweise in [JJ02] als Sprache vorgeschlagen wird, eignet sich nur bedingt dafür. Neben der grafischen Darstellung muss auch eine textuelle Beschreibung der Pattern möglich sein, wofür die UML ungeeignet scheint, wie auch in [RJB98] nachzulesen ist.

Die Verwendung dieser Mechanismen-Pattern bei der Entwicklung sicherer Systeme führt zu einer ähnlichen Tabelle, wie sie auch für das allgemeine Vorgehensmodell entstanden ist. Ein interessanter und neuer Ansatz stellt dagegen der Einsatz der Pattern zur Laufzeit von Systemen dar. Es ist vorstellbar, durch Pattern die Mechanismen in Systeme zur Laufzeit hinzuzufügen und so auf die Entstehung neuer Angriffe zu reagieren, die während der Entwicklung unbekannt geblieben sind. Diese Anwendung führt zu Tabelle 2.4, in der das Ergebnis der Klassifikation dargestellt ist. Security-Pattern Ansätze sind jedoch ein anderer Ansatz wie der hier vorgestellte und sollen deshalb nicht näher erläutert werden.

2.4.1.4 Taxonomien

Neben der Beschreibung der Gegenmaßnahmen spielen ebenso die allgemeinen Beschreibungen der Angriffe eine wichtige Rolle. Da die Vielfalt der Angriffsmöglichkeiten nur unvollständig aufzählbar ist, existieren zur Klassifikationen der Angriffe bereits mehrere Taxonomien. Diese kategorisieren die Attacken aufgrund deren bekannter Phasen, die im Folgenden aufgezählt werden.

Eigenschaft	Ausprägung bei *Erkennung*
LOC_T	Beliebig
LOC_A	Nicht innerhalb des Systems, externer Angreifer
SUB_A	Keine Subjekte des Systems
SUB_T und OBJ_T	Beliebig
OBJ_A	Keine Systemobjekte
CON_A, TIM_A und INT_A	Beliebig
VUL_T	Alle Arten

Tabelle 2.4: Erkennung bei Pattern-orientierten Ansätzen

1. Angreifer

   Beschreibung als externer oder interner, aktiver oder passiver Angreifer.

2. Werkzeuge

   Angabe der Werkzeuge, die ein Angreifer für die erfolgreiche Durchführung des Angriffs benötigt. Dazu zählen ebenso physikalische Güter, wie Rechner oder Netzbandbreiten, aber auch Subjekte und Objekte innerhalb der Softwaresysteme.

3. Schwachstelle

   Beschreibung der Schwachstellen, die für den Angriff ausgenutzt werden.

4. Aktion

   Ungefährer Ablauf des Angriffs, zeitliche Reihenfolge der einzelnen Schritte und wichtige Ereignisse.

5. Ziel

   Beschreibung des Ziel und der Teilziele, die der Angreifer zu erreichen sucht. Wichtig ist unter anderem die Angabe, ob Teilziele voneinander abhängig sind, so dass die Verhinderung eines Teilziels zur Vermeidung des gesamten Angriffs führt.

6. Resultat

   Welches Ergebnis ergibt sich für das angegriffene System im Falle eines erfolgreichen Angriffs? Das Resultat ist für die richtige Bewertung bei der Risikoanalyse entscheidend.

7. Absicht

   Angabe der Absicht des Angreifers. Die Absicht kann vom angestrebten Ziel unterschiedlich sein. Beispielsweise kann das Ziel das Ausspionieren persönlicher Daten sein, die Absicht dahinter die Verwendung der Daten um der Person zu schaden.

In dieser Arbeit wird die Klassifizierung von Angriffen durch das Schema in 2.3 durchgeführt, da damit im Gegensatz zu den bekannten Taxonomien auch auf Angriffe reagiert werden kann, die unbekannt sind. Taxonomien werden als präventive Maßnahme eingesetzt, deren Klassifikation lässt sich tabellarisch wie in 2.5 darstellen.

Eigenschaft	Ausprägung bei *Prävention*
LOC_T	Beliebig
LOC_A	Interner oder externer Angreifer
SUB_A	Beschreibbar
SUB_T	Subjekte
OBJ_A	System, nicht die Einsatzumgebung
OBJ_T	Objekte des Systems oder aus der Umgebung bekannt
CON_A	Angriffe müssen vorher bekannt sein
TIM_A	Entwicklungszeit und darüber hinaus nutzbar
INT_A	Ziel und evtl. Absicht beschreibbar
VUL_T	Beliebig

Tabelle 2.5: Prävention durch Verwendung von Taxonomien

2.4.1.5 AVDL — Application Vulnerability Description Language

Auch wenn deren Entwicklung noch nicht weit genug ist, um hier eine tabellarische Klassifikation durchzuführen, soll die Entwicklung von „AVDL“ durch die „Organization for the Advancement of Structured Information Standards“ (OASIS) erwähnt werden, die im April 2003 begonnen hat. „AVDL“ steht für „Application Vulnerability Description Language“, liegt bisher im Wesentlichen als Papierarbeit vor und soll bald zu einer Wissensquelle über bekannte Schwachstellen und Angriffe anwachsen. Für diese Arbeit ist diese Entwicklung deshalb interessant und wichtig, da sie eine einheitliche Sprache zur Darstellung wählt, die durch die Verwendung von XML-Beschreibungstechniken ebenso manuell vom Menschen als auch automatisch durch Maschinen lesbar und auswertbar ist. Dadurch können

erstmals Systeme zur Laufzeit mit Informationen beliefert werden und diese bekommen selbst Gelegenheit, auf mögliche Bedrohungen zu reagieren. Es ist zudem angedacht, die Sprache zu erweitern, um auch Bedrohungen zu beschreiben, die nicht aufgrund expliziter Schwachstellen entstehen. Nähere Informationen finden sich unter [OAS03].

2.4.2 Ratgeber und Kriterienkataloge

Ein wichtiges Ziel der Ratgeber und Kriterienkataloge ist es, die Sicherheitsevaluierung von Systemen nachvollziehbar durchzuführen. Dazu liefern sie einen Vergleichsmaßstab zwischen verschiedenen Sicherheitsarchitekturen, der die Beurteilung aufgrund der eingesetzten Sicherheitsmechanismen und deren Umsetzung erlaubt. Daneben liefern sie Verhaltensregeln für die Entwicklung und die Administration abgesicherter Systeme.

Eine Methode zur Evaluierung von abgesicherten Systemen enthält folgende Eigenschaften.

- Neben der funktionalen Beschreibung ist eine Aufzählung der Sicherheitsanforderungen und Schutzziele durchzuführen, die für das zu untersuchende System und seine Umgebung festgelegt werden.
- Es wird eine Methode zur Verfügung gestellt, die eine Prüfung zur Einhaltung der funktionalen Eigenschaften zulässt.
- Eine Messmethode zur Prüfung der erreichten Sicherheitsstufe, die durch eine Metrik dargestellt wird.

Die im Folgenden angebenen Methoden unterscheiden sich jeweils in diesen drei Punkten.

2.4.2.1 IT-Grundschutzhandbuch

Das IT-Grundschutzhandbuch wird vom Bundesamt für Sicherheit in der Informationstechnik (BSI) entwickelt und stellt einen praktischen Leitfaden im Umgang mit Sicherheit bei IT-Systemen dar. Ziel ist es durch Anwendung von organisatorischen, personellen, infrastrukturellen und technischen Standard-Sicherheitsmaßnahmen die Sicherheit von Systemen zu erhöhen und auf ein für die Einsatzumgebung ausreichendes Niveau zu bringen. Dazu werden einzelne Bausteine des Gesamtsystems betrachtet und analysiert, wie beispielsweise die bauliche Umgebung, die Kommunikationswege, die Arten und Kenntnisse der Benutzer und die beteiligten Netze. Vom Grundschutzhandbuch werden sowohl die Vorgehensweise, wie auch ein Baukasten an Sicherheitsmaßnahmen für jeweilige Anwendungen angegeben. Das Besondere am Grundschutzhandbuch im Gegensatz zu den anderen erwähnten Verfahren ist, dass auch betriebliche Maßnahmen, wie etwa die administrativen Arbeiten beschrieben werden und so analysiert werden können. Dies wird in folgender

Tabelle 2.6 für den Einsatz reaktiver Maßnahmen dargestellt. Das Grundschutzhandbuch bietet klassische Möglichkeiten für den präventiven Schutz, deshalb soll hier gezeigt werden, welche Möglichkeiten auch im reaktiven Bereich geboten werden.

Eigenschaft	Ausprägung bei *Reaktion*
LOC_T	System ist vollständig bekannt und beschrieben
LOC_A	Ort des Angreifers unbekannt
SUB_A	Beliebig
SUB_T	Keine Komponenten, die zur Reaktion nötig sind
OBJ_A	Beliebig
OBJ_T	Keine Komponenten, die zur Reaktion nötig sind
CON_A	Durch administrative Reaktionen kann reagiert werden
TIM_A	Manuelle Reaktionen zeitversetzt
INT_A	Sowohl Angriffe auf Integrität, Vertraulichkeit als auch Verfügbarkeit können erfasst werden
VUL_T	Alle Schwachstellen, deren Ausnutzung erkennbar ist

Tabelle 2.6: Reaktive Maßnahmen durch das Grundschutzhandbuch

2.4.2.2 TCSEC

Eine der ersten, aus dem militärischen Bereich heraus entwickelten Evaluierungsmethoden war die „Trusted Computer Security Evaluation Criteria“ (TCSEC oder Orange Book), die für die untersuchten Systeme die Sicherheitsstufen C1, C2, B1, B2, B3 und A1 zur Verfügung stellt. Grundlage der Überprüfungen ist einerseits das Bell-LaPadula-Modell und andererseits das Referenzmonitor-Konzept. Im Wesentlichen wird das Schutzziel der Vertraulichkeit bei TCSEC geprüft. Durch die Anlehnung an Bell-LaPadula wird auch eine abgeschwächte Prüfung der Datenintegrität durchgeführt, die aber nur die Daten innerhalb der systemweiten Zugriffskontrolle kontrolliert. Daten, die darüberhinaus im System verwaltet werden, können auf Integrität nicht überprüft werden. Das Schutzziel der Vertraulichkeit wird überhaupt nicht berücksichtigt. TCSEC stellt bestimmte Ansprüche an die Sicherheitsfunktionen des Systems, wie etwa Identifizierung und Authentifizierung der Benutzer, einer systemweiten Zugriffskontrolle ab der Sicherheitsstufe B1 oder dem Vorhandensein von Auditmechanismen. Zu den reinen Sicherheitsanforderungen gehören das Durchführen von Tests, eine Dokumentation, die nach genau vorgeschriebenen Inhalten angefertigt werden muss und eine ab hohen Sicherheitsstufen genau einzuhaltende Sy-

stemspezifikation und Verifikation. Der Prozess der Evaluierung ist genauestens festgelegt und wird grob in die drei Phasen Design, Test und einem Review eingeteilt. Dabei spielt die Dokumentation als Schnittstelle zwischen den Entwicklern und den Überprüfern eine wesentliche Rolle, da bei TCSEC keinerlei Quelltextanalyse vorgesehen ist, sondern alles durch exakte Dokumentation belegt sein muss. Da TCSEC zunächst für die Prüfung von Betriebssystemen entwickelt wurde, ist die Anwendung auf andere Bereich schwierig. Prüfungen sind sehr aufwändig und die Entwicklung von TCSEC während der Verwendung führte auch dazu, dass beispielsweise das Erreichen der Stufe C2 nach ein paar Jahren schwieriger wurde, da die Anforderungen weiterentwickelt wurden. Die Klassifikation ist in Tabelle 2.7 dargestellt. Es existieren Papiere, die eine Realisierung von Systemen nach A1 beschreiben, wie etwa in [Wei92].

Eigenschaft	Ausprägung bei *Prävention*
LOC_T	Beliebig
LOC_A	Extern oder authentifiziert interner Angreifer
SUB_A	Ein Angreifer, Kenntnisse des Systems nötig
SUB_T	Beliebige Systeme
OBJ_A	Nicht näher bestimmbar
OBJ_T	Durch Zugriffskontrollen geschützte Objekte
CON_A	Bestimmte Ziele, Angriff auf Vertraulichkeit, konzeptionelle Fehler, keine Fehler in Quelltext oder Laufzeitsystem
TIM_A	Angriffe auf die Vertraulichkeit zur Laufzeit
INT_A	Umgehung von Kontrollen
VUL_T	Nur Vertraulichkeit, keine allgemeine Integrität oder Verfügbarkeit berücksichtigt

Tabelle 2.7: Prävention durch TCSEC

2.4.2.3 ITSEC

Neben TCSEC, das im Wesentlichen im amerikanischen Bereich eingesetzt wurde, gab es auch eine vergleichbare Entwicklung in Europa, die „Information Technology Security Evaluation“, die den europäischen Standard darstellt. Wie bei TCSEC werden dabei 6 Evaluationsstufen festgelegt. Bestimmte Anforderungen an die Sicherheitsarchitekturen, wie etwa Prozessisolation, das „Least Privilege“-Prinzip oder definierte Benutzungsschnittstellen

werden dabei aber nicht definiert. Anders als bei TCSEC werden sowohl Anforderungen an die Compiler und Laufzeitumgebungen gemacht als auch der Quelltext und der Objektcode bei der Evaluierung überprüft. Details dazu finden sich auch in [Cho92]. Grundsätzlich sind die Analysetechniken bei ITSEC näher an der Implementierung angelehnt. Die Schwachstellenanalyse wird nicht am dokumentierten Design, sondern auf Implementierungsebene durchgeführt. Dies ermöglicht auch Schwachstellen zu finden, die nur im Code erkennbar sind, wie Pufferüberläufe oder Implementierungsfehler.

Neben TCSEC und ITSEC existieren noch einige weitere bekannte Methoden, die allerdings für den Maßnahmenkatalog nicht weiter von Bedeutung sind. Demgegenüber stellen die im Folgenden angesprochenen und bewerteten „Common Criteria“ die Fortführung der Entwicklung und die Vereinheitlichung der Methoden dar. Allgemeine Diskussion über Kriterienkataloge finden sich unter anderem in [Ran95].

2.4.2.4 Common Criteria

Den Nachfolger der besprochenen Methoden stellen die Common Criteria (CC) dar. Näheres zur Motivation und Entwicklung findet sich unter [Den93]. Näheres findet sich in [oST99]. Sie sind sowohl international anerkannt (Europa, Amerika und Asien) als auch ISO-standardisiert (ISO 15408). Neben einer allgemeinen Evaluierungsmethode liefert CC auch länderabhängige Methoden. Die Sicherheitsklassen werden in sogenannte Evaluation Assurance Levels (EAL) eingeteilt. In den Protection Profiles (PP) werden die Sicherheitsanforderungen für allgemeine Gruppen von Produkten oder Systemen definiert, die für alle Ausprägungen dieser Klasse anwendbar sind. Beispiele hierfür ist die PP für Chipkarten und deren Betriebssysteme. Die Profile sind in umgangsprachlicher Form verfasst und durch 6 Kapitel strukturiert, wie etwa die Beschreibung der Systemfamilie, die Sicherheitsumgebung und die Sicherheitsanforderungen. Das Security Target (ST) beschreibt ein konkretes Produkt oder System mit seinen Sicherheitsanforderungen als Basis für eine Sicherheitsanalyse. Diese können einerseits direkt mit den Mitteln der Common Criteria neu entwickelt werden, was einen hohen Aufwand darstellt. Andererseits kann das ST auch durch die Verwendung einer passenden Protection Profile entwickelt werden und so viele Elemente daraus übernehmen und verfeinern. Die ST bestehen aus 8 Kapiteln, die denen der PP vergleichbar sind und durch die genaue Beschreibung der Realisierung des Systems und einem Kapitel mit den Zusammenhängen zu verwandten PP erweitert wurden. Die funktionalen Sicherheitsanforderungen werden in 11 wesentliche Klassen zusammengefasst, wie Audit, Kryptografie, Schutz der Kommunikation, Schutz der Benutzerdaten und Privatheitsaspekte. Die Anforderungen an die Sicherheitsmaßnahmen werden in 10 Klassen eingeteilt, wie den Lebenszyklus der Entwicklung, Vorgehen bei Tests, Schwachstellenanalyse und Dokumentationsvorlagen.

Die Common Criteria Maßnahmen, deren Eigenschaften bei der Prävention in Tabelle 2.8 festgehalten sind, liefern zudem auch Richtlinien, wie nach erfolgten Angriffen mit einem System umgegangen werden soll. Diese sind allerdings als administrative Maßnahmen ein-

Eigenschaft	Ausprägung bei *Prävention*
LOC_T	Stationäre als auch mobile Systeme
LOC_A	Interne und externe Angreifer
SUB_A	Statisches Angreifermodell
SUB_T	Nach Systemmodell
OBJ_A	Statisch
OBJ_T	Nach Systemmodell
CON_A	Liefert Maßstab, Metrik, Vergleichsmöglichkeit für Sicherheitsstufe, Schwachstellen auf Implementierungsebene
TIM_A	Beliebig
INT_A	Alle Absichten darstellbar
VUL_T	Bekannte Schwachstellen und Angriffe beschreibbar

Tabelle 2.8: Prävention durch die Common Criteria

zustufen und werden in einem folgenden Abschnitt beschrieben.

2.4.3 Bestehende Mechanismen, Methoden und Modelle

Im folgenden Abschnitt werden bekannte Methoden und Modelle zur Beschreibung sicherer Systeme diskutiert. Es wäre wünschenswert, ein Modell zu finden, das beliebige Systeme darstellen kann und alle möglichen Angriffe und Bedrohungen zeigen kann. Daraus ließen sich dann die notwendigen Gegenmaßnahmen folgern. Die beschriebenen Beispiele, die durchaus in großen Systemen zum Einsatz kommen, zeigen allerdings die Beschränkung der Modelle und Methoden auf spezielle Bedrohungen und sind nicht universell.

2.4.3.1 Formale Modelle und Methoden

Ziel der Methoden und Verfahren, die man zur Absicherung von Systemen einsetzt, sollte es sein, dass die Bedrohungen, denen man damit begegnen will, wirkungsvoll und nachweisbar ausgeschaltet werden. Es ist demnach naheliegend mit formalen Methoden gegen jede bekannte Bedrohung universelle Verfahren zu entwickeln, die das Auftreten verhindern. Die Erfahrung hat allerdings gezeigt, dass dieses Vorgehen nur teilweise zum Erfolg führen kann.

Die Menge der bekannten Bedrohungen ist im Allgemeinen nur ein Teil der möglichen Bedrohungen. Einerseits müsste für das vollständige Erfassen aller Bedrohungen ein vollständiges Modell des Systems und seiner Umgebung zur Analyse vorliegen. Dies wiederum würde zumindest eine vollständige Modellierung der gesamten Welt bedeuten, was aus naheliegenden Gründen schwer möglich ist. Andererseits sind Systeme auch nach der Entwicklung beim Einsatz keine statischen Gebilde, sondern werden weiterentwickelt und verändern sich mit der Umgebung, in der sie eingesetzt werden. Das Modell und die Analyse der Bedrohungen müssten dann permanent aktuell gehalten werden.

Der Einsatz formaler Methoden bei komplexen Systemen ist sehr aufwändig und damit sehr teuer, wie beispielsweise in [Ame80] diskutiert wird. Selbst bei einfachen Projekten ist die Komplexität meistens schon zu hoch für deren Einsatz, so dass auf ingenieurhafte Vorgehen gesetzt wird. Hauptargument für den Sinn dieses Vorgehens ist die Tatsache, dass sowieso nur ein Teil der Bedrohungen abgedeckt werden kann, eine Vollständigkeit und damit sicheres System auch mit exakten Methoden daher nicht erreicht wird.

Dennoch existieren einige Teilbereiche der IT-Sicherheit, in denen formale Methoden sehr wirkungsvoll eingesetzt werden und gegen bestimmte Angriffe schützen, wie auch in [Win98] gezeigt wird. Dies darf aber nicht darüber hinweg täuschen, dass für ein abgesichertes Gesamtsystem erst die Integration der Verfahren aus vielen Bereichen zum Ziel führt. Wirkungsvolle Modelle zur Beschreibung von Authentifizierungsmaßnahmen finden sich etwa in [BAN90], [NS78] und [NS87], Vorgehen bei der Programmierung sicherer Systeme etwa in [HR98]. Beispiele für die Absicherung durch formale Methoden sind die Anwendung der Kryptografie, die auf streng mathematischen Gegebenheiten beruht und die Sicherheitsmodelle, die ihre Exaktheit durch formale Beschreibungen gewinnen. Im Folgenden werden deshalb Sicherheitsmodelle und die Kryptografie klassifiziert.

Bell-LaPadula

Ein bekanntes Modell zur Beschreibung der kontrollierbaren Informationsflüsse in Systemen ist das Modell von Bell und LaPadula, das in [BL73] ausführlich beschrieben wird. Es besteht einerseits aus einem Verfahren zur Darstellung der Subjekte und Objekte mit unterschiedlichen Sicherheitsqualifizierungen und andererseits aus einer mathematischen Beschreibung der kontrollierten Systeme. Wesentlich sind dabei eine Reihe von formalisierten Regeln, die bestimmen, welche Zugriffe auf Objekte durch Subjekte erlaubt sind und welche nicht. Jedes Objekt und jedes Subjekte besitzen eine Sicherheitsstufe, wobei Informationsflüsse nie von einer höheren zu einer niedrigeren Stufe stattfinden dürfen. Die wesentlichen zwei Regeln zur Einhaltung der Vertraulichkeit sind „no read up“ und „no write down“, die das Schreiben von vertraulichen Daten hin zu niedriger klassifizierten und das Lesen von Daten aus höher klassifizierten Daten verbieten. Es sind mehrere Einschränkungen des Modells bekannt, eine wesentliche ist, dass ein weiteres Schutzziel wie die Integrität nur schwer mit dem Modell vereinbar ist. Der sogenannte „blind write“ ist nämlich nach den Regeln von Bell-LaPadula erlaubt und bedeutet nichts anderes, als dass ein Subjekt

niedriger Klassifizierung in ein Objekt höherer Klassifizierung schreiben darf, Daten aber anschließend nicht auslesen darf. Einige Erweiterungen zu Bell-LaPadula existieren, die diese Art der Probleme beheben, so dass das Modell beispielsweise in Betriebssystemen einsetzbar ist. Die Klassifikation ist in Tabelle 2.9 zu sehen, ein Vergleich verschiedener Modelle findet man in [CW87], andere Modelle sind etwa in [TW89] angesprochen. Eine wichtige Anmerkung zum Bell-LaPadula Modell findet sich in [McL85].

Eigenschaft	Ausprägung bei *Prävention*
LOC_T	Systemprozesse
LOC_A	Teil des Systems
SUB_A	Subjekte innerhalb des Systems
SUB_T	Subjekte mit bestimmter Sicherheitsklassifikation
OBJ_A	Objekte innerhalb des Systems
OBJ_T	Objekte mit bestimmter Sicherheitsklassifikation
CON_A	Beliebig
TIM_A	Beliebig
INT_A	Vertrauliche Informationen höherer Klassifikation erhalten
VUL_T	Nur Vertraulichkeit

Tabelle 2.9: Prävention durch Bell-LaPadula

Biba

Der Nachteil des Bell-LaPadula-Modells ist, dass andere Schutzziele als Vertraulichkeit nicht berücksichtigt werden. Mit dem Biba-Modell wird Bell-LaPadula so abgewandelt, dass die Integrität gewährleistet wird. Biba stellt eine Gruppe mehrerer Modelle mit diesem Ziel dar, von denen zwei wichtige kurz besprochen werden, die Klassifikation findet sich in Tabelle 2.10. Das „Mandatory Integrity Model“ kehrt die beiden Hauptregeln von Bell-LaPadula um, so dass eine „no write up“ und eine „no read down“ Regel entsteht. Damit ausgedrückt kann weder von einem niedriger klassifiziertem Subjekt ein höher klassifiziertes beschrieben, noch von einem höher klassifiziertem Objekt Daten in ein niederes Subjekt gelesen werden. Problematisch dabei ist es, dass in Systemen, in denen sowohl Vertraulichkeit als auch Integrität geschützt werden sollen, die Regeln von Biba und Bell-LaPadula gegensätzlich zueinander sind. Die andere Variante ist das „Object Low-Water Mark Model“ und das „Subject Low-Water Mark Model“. Dabei werden die beschriebenen

Objekte (Object Low-Water) und die lesenden Subjekte (Subject Low-Water) jeweils nach der Operation auf die niedrige priviligierte Stufe gesetzt. Bei Object Low-Water wird demnach ein höher klassifiziertes Objekt nach dem Beschreiben durch ein niedrigeres Subjekt auf dessen Stufe zurückgesetzt. Analog wird bei Subject Low-Water das lesende Subjekt von einer höheren Privilegstufe auf die des Objektes gesetzt. Dadurch werden die Folgen des Integritätsverlustes durch Anpassung der Privilegien verringert. Problematisch ist dabei, dass es keine Möglichkeit für Subjekte oder Objekte gibt, eine höhere Privilegstufe zu erreichen.

Eigenschaft	Ausprägung bei *Prävention*
LOC_T	Systemprozesse
LOC_A	Teil des Systems
SUB_A	Subjekte innerhalb des Systems
SUB_T	Subjekte mit bestimmter Sicherheitsklassifikation
OBJ_A	Objekte innerhalb des Systems
OBJ_T	Objekte mit bestimmter Sicherheitsklassifikation
CON_A	Beliebig
TIM_A	Beliebig
INT_A	Integrität höher klassifizierter Daten oder Subjekte beschädigen
VUL_T	Nur Integrität

Tabelle 2.10: Prävention durch Biba

Chinese Wall

Ein an die Probleme in der Praxis bei Interessenskonflikten angepasstes Modell ist das Chinese Wall Modell, wie in [BN89] beschrieben. Dabei werden ähnliche Vertrauensbereiche unterschiedlicher Organisationen klassifiziert, wobei ein Informationsfluss zwischen verwandten Bereichen konkurrierender Organisationen verhindert wird. Neben der Klassifikation sind die Mengen der Subjekte und Organisationen und der schützenswerten Daten festzulegen, wie in Tabelle 2.11 dargestellt ist. Eine Beobachtung der Zugriffshistorie, die für die Systemakteure protokolliert wird, erlaubt die Kontrolle gültiger und ungültiger Zugriffe. In diesem Modell werden damit sowohl die Vertraulichkeit als auch die Integrität der Daten kontrolliert.

Eigenschaft	Ausprägung bei *Prävention*
LOC_T	Systemprozesse und Organisationen
LOC_A	Teil des Systems
SUB_A	Subjekte innerhalb des Systems
SUB_T	Subjekte mit bestimmter Sicherheitsklassifikation, Zugriffshistorie gespeichert
OBJ_A	Objekte innerhalb des Systems
OBJ_T	Objekte mit bestimmter Sicherheitsklassifikation, Zugriffshistorie gespeichert
CON_A	Beliebig
TIM_A	Beliebig, Zugriffshistorie wird gespeichert
INT_A	Integritätsverletzung und Vertraulichkeitsverlust zwischen Organisationen
VUL_T	Vertraulichkeit und Integrität

Tabelle 2.11: Prävention durch Chinese Wall

Erweiterungen des Chinese Wall Modells sind unter anderem in [San92] zu finden. Es existieren weitere Sicherheitsmodelle unterschiedlicher Ausprägungen. Hier wurden nur die wichtigsten angesprochen, die Vertreter der wesentlichen Gegenmechanismen darstellen, weitere finden sich beispielsweise in [DLS+88].

2.4.3.2 Kryptografie

Eine Sonderrolle nehmen die kryptografischen Verfahren ein. Durch die mathematische Durchdringung des Gebietes, im Gegensatz zu den meisten anderen Verfahren, kann über den Sicherheitsgewinn eine klare Aussage getroffen werden. Insbesondere sind deren Schwachstellen im Allgemeinen bekannt, wie etwa die aufzuwendende Rechenleistung zum Brechen von Schlüsseln oder die Wahrscheinlichkeit, Schlüssel zu erraten. Aufgrund der Qualität und Zuverlässikeit von kryptografischen Verfahren ist es stets wünschenswert, in geeigneten Bereichen und Teilsystemen Kryptografie zur Absicherung zu verwenden. Werden Standardverfahren angewendet, bedeutet dies stets hohe, transparente Sicherheit.

Zur Klassifikation wird zwischen der Sicherung der Integrität (Abbildung 2.11) und Absicherung der Vertraulichkeit (Abbildung 2.13) unterschieden. Für die in dieser Arbeit als drittes Schutzziel festgelegte Absicherung der Verfügbarkeit spielen kryptografische Ver-

fahren praktisch keine Rolle.

In der Kryptografie sind jedoch Verfahren zur Absicherung der Authentifikation wichtig. Diese ermöglichen eine vertrauenswürdige, im allgemeinen sichere Antwort auf die Frage: „Wer ist mein gegenüber?“ Dafür existieren eine Reihe von Protokollen und Verfahren, die einen wichtigen Teil für die Kryptografie und für die Konstruktion von sicheren Systemen darstellen. Verfahren zur Authentifikation werden deshalb in Abbildung 2.12 als grobe Übersicht dargestellt.

Die Authentifikation hat in modernen, vernetzten IT-Systemen eine bedeutende Rolle. Deshalb wird der Begriff in die in Abschnitt 2.1 eingeführten Schutzziele eingeordnet.

Authentifikation: *Einordnung in die Schutzziele Vertraulichkeit, Integrität und Verfügbarkeit*
Mit Authentifikation bezeichnet man einen Vorgang, der eine bekannte Identität eines Kommunikationspartners möglichst verlässlich bestimmt. In der Literatur wird im Gegensatz zu dieser Arbeit Authentifikation stets als generelles Schutzziel verstanden, wie die drei Schutzziele Vertraulichkeit, Integrität und Verfügbarkeit. Im Gegensatz dazu stellt die Authentifikation jedoch einen Vorgang dar, der auf diese drei Basisschutzziele zurückgeführt werden kann. Man betrachte dazu ein Kommunikationssystem in Ausführung. Der Vorgang der Authentifikation streckt sich über einen gewissen Zeitraum hin. Als Ergebnis werden Daten im System abgelegt, die den Namen des Kommunikationspartners mit Attributen, seine Identität und seine Rechte abspeichern. Ein Angreifer kann bei einem Angriff auf die Authentifizierung wiederum in drei Phasen wirken: Vorbereitung, Durchführung und Konsequenzen. In der Angriffsvorbereitung können die Integrität und Vertraulichkeit des Systems und des Kommunikationspartners gestört werden. Damit kann bei der eigentlichen Authentifizierung eine falsche Identität vorgegeben werden, die evtl. nicht erkannt werden kann. Wird die Verfügbarkeit des Authentifizierungsmechanismus gestört, so könnte dieser aus praktischen Gründen administrativ abgeschaltet werden. So wird die Verfügbarkeit des Gesamtsystems erhöht, die Sicherheit jedoch stark kompromitiert. Aus Angriffssicht, die in dieser Arbeit im Vordergrund steht, lässt sich demnach ein Angriff auf den Vorgang der Authentifikation auf Angriffe der drei Basis-Schutzziele zurückführen. In der Kryptografie stellt die Authentifikation jedoch einen wichtigen Teil der Mechanismen dar, so dass diese hier speziell betrachtet werden, siehe dazu Abbildung 2.12.

Die Abbildungen sollen nur die Vielseitigkeit und ungefähren Einsatzmöglichkeiten von kryptografischen Verfahren darstellen und daher nicht vollständig sein.

Da die Kryptografie in Hinblick auf deren Wirksamkeit eine Sonderrolle einnimmt und

deren Einsatz sehr flexibel ist, wird auf eine Klassifikation nach dem üblichen Schema verzichtet.

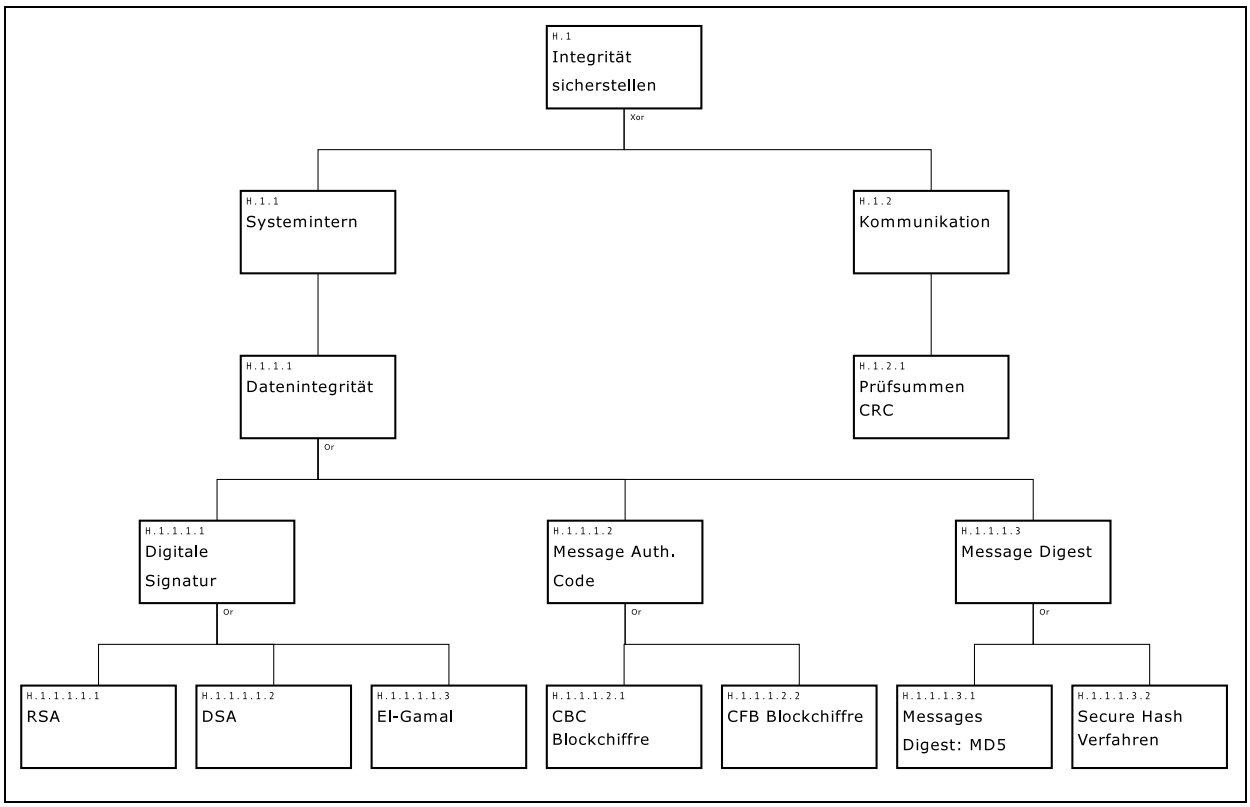

Abbildung 2.11: Hierarchie der Mechanismen zur Absicherung der Integrität

Trotz der Mächtigkeit dieser Verfahren darf nicht vergessen werden, dass zur Verwendung kryptografischer Verfahren auch Strukturen und Verwaltungsinformationen gehören, wie beispielsweise die symmetrischen und privaten Schlüssel. Diese müssen abgesichert verwaltet werden, was schließlich durch andere, zusätzliche Verfahren geleistet wird. Kryptografie ist demnach ein wichtiger Baustein für die Konstruktion abgesicherter Systeme, kann das Ziel im Allgemeinen aber nie alleine erreichen. Klassisches Beispiel dafür ist die erreichte Sicherheit des Passwortsystems unter Unix, das auch in [FK89] und [MT79] besprochen wird. Obwohl das Crypt-Verfahren als sicher eingestuft wird, ist es leicht, Passwörter zu erraten, da die Verwaltung mangelhaft durchgeführt wird. Für kryptografische Verfahren in der Kommunikation sei auf die Texte [Mer78], [RS01], [RSA78] und [Smi98] verwiesen. Implementierungen verschiedener Kryptoverfahren sind in [Sch97] und [MvOV96] zusammengefasst, wobei der selbe Author in [Sch00] beschreibt, was Kryptografie für sichere Systeme tatsächlich leistet.

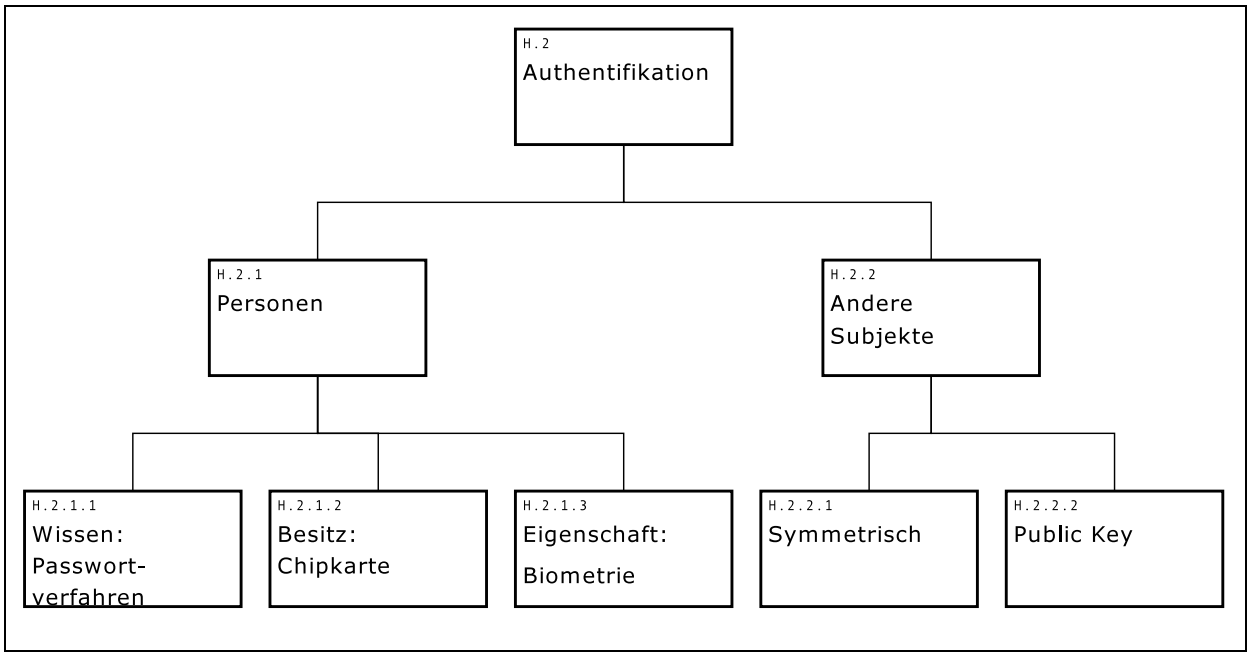

Abbildung 2.12: Hierarchie der Mechanismen zur Authentifikation

2.4.3.3 Schutz für Pufferüberlauf auf dem Keller

Häufigste Schwachstelle in Systemen ist heute einen möglicher Überlauf auf dem Keller erzeugen zu können. Gegen das Vorgehen dabei, das in Abschnitt 2.2.2.2 besprochen wurde, existieren einige Gegenmaßnahmen, die den Überlauf nicht prinzipiell verhindern, aber erkennen können. Eine mögliche Art den Überlauf zu erkennen ist einen beliebigen Wert zwischen die abzugrenzenden Kellerbereiche zu schreiben, der nach Rückkehr aus der Funktion geprüft wird. Eine Veränderung, die auf einen Überlauf schließen lässt, führt zum Prozessabbruch. Theoretisch könnte ein Angreifer aber den Wert erraten und beim Überschreiben den richtigen Wert ablegen, so dass der Überlauf nicht entdeckt werden kann. Trotz der möglichen Umgehung durch Angreifer wird diese Gegenmaßnahme wegen der einfachen Realisierung eingesetzt.

Das Ausnutzen des Pufferüberlaufs auf dem Keller ist nur deshalb möglich, weil der Speicheraufbau des Prozesses auch bei wiederholtem Laden immer identisch ist. Dies ist im Wesentlichen ein Verdienst der virtuellen Speicherverwaltung, die es für einen einfacheren und schnelleren Binde- und Ladevorgang ermöglicht, dass jeder Prozess immer an die selbe virtuelle Speicheradresse geladen wird[4]. Ein Angreifer kann deshalb die Rücksprungadresse einer Funktion schon kennen, bevor der Prozess überhaupt gestartet wurde. Allerdings ist die Adressbildung sehr von der verwendeten Version des Compilers und der Laufzeitumge-

[4]Dies ist möglich, da jeder Prozess virtuell einen leeren, privaten Speicherbereich von 2^{32} Bytes (bei 32-Bit Systemen) erhält.

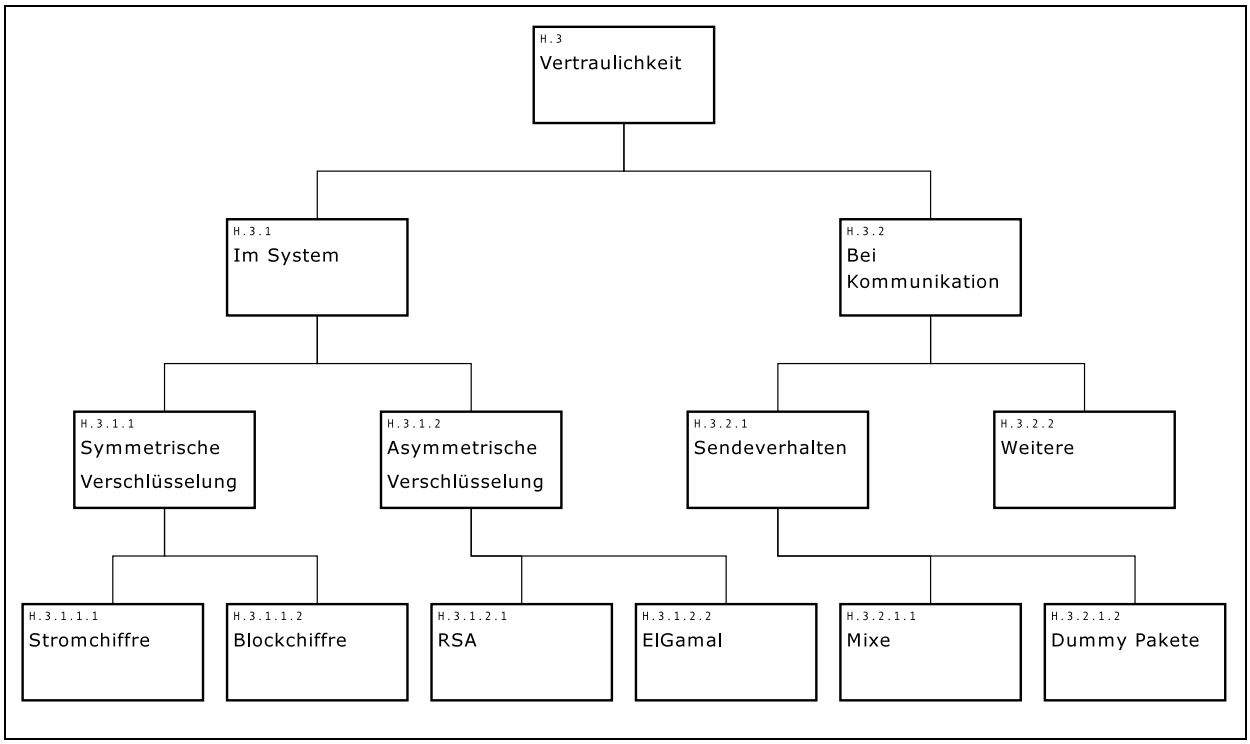

Abbildung 2.13: Hierarchie der Mechanismen zur Vertraulichkeit

bung abhängig. In einigen Systemen wird deshalb versucht, die Ladeadresse des Prozesses bei jedem Laden zu verändern und beim Übersetzen Funktionen um einige Speicherstellen zu verschieben. Damit ist es unwahrscheinlich, dass der Angreifer die richtige Adresse beim Überlauf verwenden kann.

Eine andere Art der Gegenmaßnahmen, die keine Hintertür für einen Angreifer offen lässt, ist der Schutz durch Einfügen von ungültigen Speicherseiten. Dabei werden die einzelnen Kellerbereiche von je einer Speicherseite eingerahmt. Diese zusätzliche Seite ist in der virtuellen Speicherverwaltung als ungültig markiert. Das heisst, dass jeder Zugriff des Prozesses auf eine Adresse dieser Seite zu einem Seitenfehler und einem ununterbrechbaren Einsprung in der Kern führt. Der Kern kann daraufhin überprüfen, ob es sich bei der Seite um eine spezielle Schutzseite auf dem Keller handelt, daraufhin den Prozess beenden und den möglichen Angriffsversuch melden. Die zusätzlich eingeblendeten Seiten verbrauchen keine physikalische Seitenkachel, sie belegen nur jeweils eine Seite im virtuellen Adressraum. Diese Gegenmaßnahme ist die Grundlage für die Bewertung in folgender Tabelle, obwohl der Mechanismus noch nicht eingesetzt wird, weil er größere Änderungen an der Speicherverwaltung im Betriebssystemkern bedeutet und einen großen Aufwand darstellt. Andere Schutzmechanismen existieren und werden beispielsweise in [KZB+90] dargestellt. Die Klassifikation der Verfahren findet sich in Tabelle 2.12.

Eigenschaft	Ausprägung bei *Prävention*
LOC_T	Ein arbeitender Prozess in einem System
LOC_A	Intern und Extern, nicht Teil des Prozesses
SUB_A	Beliebig
SUB_T	Prozess und Prozessteile (Threads)
OBJ_A	Bestimmte Daten für Kellerinhalt nötig
OBJ_T	Kellerinhalte und Daten
CON_A	Funktion mit Schwachstelle muss ausgeführt werden
TIM_A	Angriff zur Laufzeit oder Startzeit
INT_A	Code im Kontext des Prozesses ausführen
VUL_T	Schnittstelle, die Daten ungeprüft auf den Keller kopiert

Tabelle 2.12: Prävention durch Pufferschutz

2.4.3.4 Vertrauenswürdige Hardware

Vertrauenswürdige Hardwareplattformen erweitern Systeme um eine abgesicherte Komponente auf unterster Ebene, der Hardware. Im Regelfall bietet diese Komponente einen Speicher an, der nur für autorisierte Subjekte über definierte Schnittstellen auslesbar ist. Zudem sind Kryptofunktionen integriert, damit bestimmtes Schlüsselmaterial den vertrauenswürdigen Bereich nie verlassen muss. Im Allgemeinen kann man diese abgesicherte Komponente als integrierte, fest verankerte Chipkarte betrachten, die in den Chipsatz der Hardware mit eingearbeitet[5] sein wird. Das Prinzip vertrauenswürdiger Hardware wurde bereits 1986 beispielsweise in [Dob86] dargestellt. Aktuelles Beispiel ist das TPM (Trusted Platform Module) der TCG (Trusted Computing Group, vormals TCPA für Trusted Computing Platform Alliance), das sowohl einen abgesicherten Speicher als auch Kryptofunktionen mit einem Zufallszahlengenerator und Managementfunktionen realisiert. Die Bedenken gegen den Einsatz dieser Architektur, wie etwa aufgrund einer möglichen eindeutigen Identifizierbarkeit, sind weitgehend unbegründet. Zwar werden Schlüssel zur Kommunikation erzeugt, diese sind aber nur für eine Sitzung gültig. Der Hauptschlüssel (Root-key) verlässt das TPM nie, die öffentlichen Schlüssel werden mit jeder neuen Sitzung getauscht. Es ist klar, dass mit diesen Architekturen die Integrität und Authentizität von Daten, sofern entsprechend vorgesehen, sichergestellt werden kann und so ein präventiver und erkennender Schutz geleistet wird. Selbstverständlich sind trotzdem Angriffe gegen diese Art von Systemen möglich. Gerade die heutigen Angriffe von Viren und Würmern, die Schwachstellen wie

[5] Die Integration wird im Wesentlichen aus Kostengründen und höherer Geschwindigkeit durchgeführt.

Pufferüberläufe der Implementierungen ausnutzen sind trotzdem genauso möglich. Details zu den angesprochenen Architekturen finden sich in [BDGL04] und [SS03]. In der folgenden Tabelle 2.13 werden die Schutzcharakteristiken der vertrauenswürdigen Hardware im präventiven Bereich zusammengefasst.

Eigenschaft	Ausprägung bei *Prävention*
LOC_T	Überhalb vertrauenswürdiger Hardware
LOC_A	Intern und Extern
SUB_A	Beliebig
SUB_T	Geschützt durch TPM
OBJ_A	Beliebig
OBJ_T	Geschützt durch TPM
CON_A	Beliebig
TIM_A	Integritätsprüfungen finden meist nur beim Start oder dem Nachladen bekannter Komponenten statt, „Complete mediation“ Prinzip verletzt
INT_A	Datenintegrität bei Speicher und Kommunikation gefährden
VUL_T	Integrität und Vertraulichkeit

Tabelle 2.13: Prävention durch den Einsatz vertrauenswürdiger Hardware

2.4.4 Einsetzbare Systeme

Neben den beschriebenen Methoden existieren Sicherheitskomponenten, die diese Methoden umsetzen und sowohl innerhalb der Rechner als auch in der Kommunikationsinfrastruktur dazwischen anzutreffen sind.

Die Software von Rechnern ist stets in verschiedene Schichten strukturierbar, die Schichten werden dabei in Anlehnung an das ISO-OSI-Modell aus dem Netzwerkbereich eingeteilt. In der untersten Ebene befindet sich die Firmware, die für die Basis-Ein-Ausgabeoperationen und den Systemstart zuständig ist. Sie dient auch der Aufzählung und Initialisierung der Hardware und ist der erste Softwareteil, der nach dem Starten abläuft. Darauf bauen die Betriebssysteme auf, deren Hauptaufgaben die Veredelung der Hardware und die Diensteerbringung für die weiter oben liegenden Anwendungen darstellt. Zwischen den Applikationen und dem Betriebssystem liegen Softwareteile, die weder reine Betriebssystemkomponenten

darstellen, weil sie zu speziell sind, noch Anwendungen im eigentlichen Sinn sind. Diese Middleware bildet einen wichtigen Teil moderner Rechnersysteme und stellt auch für die Absicherung Dienste zur Verfügung.

Sowohl die einsetzbaren Sicherheitskomponenten innerhalb von Rechnern als auch in der Kommunikationsinfrastruktur werden im Folgenden überblicksweise klassifiziert. Dabei werden nur einige wesentliche Systeme angesprochen, selbstverständlich existieren noch wesentlich mehr. Die Betriebssysteme werden in einem eigenen Kapitel behandelt, da sie einen zentralen Punkt dieser Arbeit darstellen.

2.4.4.1 Microsoft .NET

Neben der bekannten Java Architektur, die Sicherheit im Wesentlichen durch Spracheigenschaften und Interpretation erhöht, ist in modernerer Form die .NET-Umgebung entwickelt worden. Bei der Entwicklung für diese Architektur werden beliebige Hochsprachen in eine gemeinsame Zwischensprache übersetzt, die Intermediate Language (IL). Diese ist standardisiert, offen und stellt einen abstrakten Assembler dar, wie es auch bei Java Bytecode für nur eine Hochsprache, der Fall ist. Sowohl Anwendungen als auch statische und dynamische Bibliotheken werden in IL-Code übersetzt, so dass zur Laufzeit kein Unterschied mehr darin besteht, in welcher Hochsprache entwickelt wurde. Durch Signaturen wird die Integrität aller IL-Komponenten geschützt, was ein unbemerktes Verändern schwer macht und während des Binde- und Ladevorgangs, nicht mehr bei der Ausführung, geprüft wird. Neben signierten IL-Komponenten können auch unsignierte Programme entwickelt und übersetzt werden. Die Prozessumgebung dieser Programme wird dabei aber stets als „untrusted“ markiert und kann bestimmte, dynamisch konfigurierbare, Operationen nicht mehr durchführen. Sprachbasierte Sicherheitslösungen, wie Java oder .NET, sind sehr interessant, weil sie präventiv gegen bestimmte Angriffe wirken (z.B. Pufferüberläufe auf dem Stack, Fehler durch Typumwandlung), ohne dass der Programmierer sich damit auseinandersetzen muss. Eine Einordnung in das Klassifikationsschema findet sich in Tabelle 2.14.

2.4.4.2 Kryptografie-Bibliotheken

Neben den zahlreichen und wirkungsvollen Anwendungen der Kryptografie stellt sich bei deren Einsatz immer die Frage, wo in einem geschichteten System Kryptofunktionen realisiert werden. Diese stecken meist in sogenannten Kryptografie-Bibliotheken und beinhalten sowohl symmetrische als auch asymmetrische Verfahren. Deren korrekte Realisierung ist sehr aufwändig, so dass es sich anbietet, nur eine Implementierung in einem System zu verwenden. Eine Diskussion darüber findet sich in [BGL03a]. Heutige Bibliotheken sind meist als Anwendung bzw. Middleware realisiert, so dass eine Nutzung vom Kernadressraum wegen der Kontextwechsel sowohl schwierig als auch zeitaufwändig ist. Kryptofunktionen im Kern existieren vereinzelt, sie werden aber wohl kaum von den Anwendungen genutzt. Für

Eigenschaft	Ausprägung bei *Erkennung*
LOC_T	Prozess und Middleware
LOC_A	Interner Angreifer (in Prozessumgebung)
SUB_A	Prozess oder Systemteil
SUB_T	.NET Prozesse, Threads und Fibers
OBJ_A	Anderer Prozess mit Daten, Kommunikationskanal
OBJ_T	Daten der Applikation
CON_A	Managed Code in IL wird geprüft
TIM_A	Angriff nach der Ladezeit des Systems
INT_A	Schwachstellen ausnutzen
VUL_T	Auch sprachbasierte Schwachstellen

Tabelle 2.14: Erkennung durch .NET

LucaOS, dessen Sicherheitsarchitektur in Kapitel 8 beschrieben wird, wurde ein Kompromiss gewählt. Für die Kryptofunktionen existiert nur eine Codebasis, die je nach Bedarf für den Benutzeradressraum oder den Kernkontext übersetzt wird, wobei hier ein gewisser Aufwand für die bedingte Übersetzung berücksichtigt werden muss. Architekturen mit Krypto-Bibliotheken werden in Tabelle 2.15 eingeordnet.

2.4.4.3 Firewall und Paketfilter

Standardkomponenten zur Erhöhung der Sicherheit in Netzwerken sind Firewalls. Deren Standardverhalten bietet präventiven Schutz durch das grundsätzliche Sperren bestimmter Verbindungen. In IP-basierten Netzen wird dies im Wesentlichen auf Schicht 3 durch das Blockieren von bestimmten Portnummern und damit Diensten realisiert. Man spricht bei dem Netzwerkbereich, der damit vor Angriffen geschützt werden soll, von einer abgesicherten oder demilitarisierten Zone. Dieser Schutz bewirkt einerseits, dass kein Angriff direkt in diese Zone von Außen Wirkung zeigen kann und andererseits, dass der potentielle Angreifer bei der Angriffsvorbereitung keine vollständige Sicht auf das interne Netzwerk bekommen kann. Sinnvollerweise werden meistens nicht gewisse Ports gesperrt, sondern nur einige wesentliche freigegeben, ganz nach dem oben geannnten *Fail-safe defaults* Prinzip.

Die Schwachstellen, die trotzdem auftreten können, entstehen auf zweierlei Arten. Entweder wird die Firewall falsch konfiguriert und administriert, was vom Angreifer vor allem durch Testen ausgenutzt werden kann, oder es handelt sich um einen internen Angrei-

Eigenschaft	Ausprägung bei *Prävention*
LOC_T	Kern-, Middleware- und Benutzeradressraum
LOC_A	Middleware- und Benutzeradressraum, je nach BS auch Kernel
SUB_A	Subjekte in unterer oder derselben Ebene
SUB_T	Subjekte in höherer Ebene
OBJ_A	Objekte in unterer oder derselben Ebene
OBJ_T	Objekte in höherer Ebene
CON_A	Wissen über (Krypto-)verfahren
TIM_A	Beliebig
INT_A	Integrität, Vertraulichkeit
VUL_T	Protokolle, verschlüsselter persistenter Speicher

Tabelle 2.15: Erkennung durch Krypto-Bibliotheken

fer. Man geht deshalb heute dazu über, neben der Firewall für einen großen Bereich, wie etwa eine Firma, Abteilung oder eine Fakultät, auch den Schutz soweit zurückzuziehen, dass aus dem internen Angreifer wieder ein externer wird. Dazu werden auf den einzelnen Systemen je eigene Firewalls installiert, die oft auch als „Personal Firewall“ bezeichnet werden. Es entstehen wiederum Bedrohungen durch den stark erhöhten Administrationsaufwand. Neben den präventiven Maßnahmen haben Firewalls auch erkennende Wirkung, da sie den Verkehr überwachen und damit bestimmte Auffälligkeiten melden können. Eine Einordnung ist in Tabelle 2.16 dargestellt.

2.4.4.4 Integritätsprüfung

Die immer wiederkehrende Prüfung bestimmter Daten auf ungewollte Veränderungen bezeichnet man als Integritätsprüfung. Sie wird zum Erkennen verschiedenster Angriffe eingesetzt, kann diese aber nicht präventiv verhindern, sondern erfolgreiche Angriffe auf die Integrität nur nachträglich erkennen. Wie groß der zeitliche Abstand zwischen dem Angriff selbst, dem Erkennen und der Möglichkeit zur Reaktion ist, hängt sowohl von der Art der Integritätsprüfung als auch von der eigentlichen Anwendung ab. Beispiele hierfür sind Viren- und Wurmscanner, die ausführbaren Code nach Schädlingssignaturen durchsuchen. Sie können aber immer erst dann wirkungsvoll einen Angriff melden, wenn er bereits durchgeführt wurde. Trotz dieser harten Einschränkung, wünschenswert wäre natürlich den

Eigenschaft	Ausprägung bei *Erkennung*
LOC_T	Innerhalb der abgesicherten Zone
LOC_A	Außerhalb der abgesicherten Zone
SUB_A	Netzwerknutzung möglich
SUB_T	Dienst an einer Netzwerkverbindung
OBJ_A	Zugang zum Netzwerk
OBJ_T	Netzwerkverbindung
CON_A	Angriffsmuster bekannt und erkennbar
TIM_A	Auch Timing-basierte Angriffe
INT_A	Zugang zur Netzwerkverbindung innerhalb der abgesicherten Zone
VUL_T	Auch Angriffe auf Verfügbarkeit

Tabelle 2.16: Erkennung durch Firewalls

Angriff sofort zu erkennen, werden diese Integritätsprüfungs-Verfahren heute sehr häufig eingesetzt, da sie das Mittel gegen Viren und Würmer darstellen. Ein anderes Beispiel wäre die Prüfung von zentralen Datenbeständen, die beispielsweise sicherheitskritische Konfigurationsdaten beinhalten können. Die Leistung von Integritätsprüfungen wird in Tabelle 2.17 dargestellt.

2.4.4.5 Einbruchserkenner

Im Rahmen der automatischen Erkennung von Angriffen während der Laufzeit eines Systems existieren heute im Wesentlichen Techniken der „Instrusion Detection Systeme“ (IDS), beziehungsweise Einbruchserkenner wie auch in [Den86] und [Sun96] besprochen wird. Die Techniken zur Erkennung von Angriffen reichen vom einfachen Mustervergleich auf Basis bekannter Angriffe bis hin zu verteilten Sensoren, die durch komplizierte Logiken ungewolltes Verhalten erkennen und so Angriffe melden sollen. Schwierigkeiten ergeben sich im Betrieb der IDS-Systeme vor allem mit der großen Datenmenge, die bei der Beobachtung der Systemaktivitäten anfallen. Die Klassifikation der Einbruchserkenner ist in Tabelle 2.18 wiedergegeben.

Eigenschaft	Ausprägung bei *Erkennung*
LOC_T	Nur die integritätsgeprüften statischen Daten
LOC_A	I.A. außerhalb des Systems
SUB_A	Aktiver Angriff
SUB_T	Subjekte, die Daten verändern können
OBJ_A	Verbindung zum System
OBJ_T	Datenobjekte, evtl. mit Schnittstelle zum Netzwerk
CON_A	Zugriff auf das System (über Netzwerk) notwendig
TIM_A	Die Wirksamkeit des Angriffs hängt hauptsächlich davon ab, wie schnell er aufgefunden wird
INT_A	Einschleusen von Schwachstellen oder Code, Änderung von sicherheitskritischen Parametern
VUL_T	Angriffe auf Integrität

Tabelle 2.17: Erkennung durch Integritätsprüfung

2.4.5 Organisatorische Sicherheit

Als organisatorische Sicherheit bezeichnet man alle Vorgänge, die die Sicherheit eines Systems durch administrative Maßnahmen erhöhen. Darunter fallen unter anderem Sicherungsstrategien, bauliche Maßnahmen oder Richtlinien im Umgang mit den Systemen. Obwohl diese Dinge keine besonderen Verfahren oder Methoden im Sinne der Informatik darstellen, sind sie für die Sicherheit heute von entscheidender Bedeutung, da die meisten erkennenden Maßnahmen immer noch durch den Menschen erfolgen. Er nimmt dabei beispielsweise die Rolle des Administrators ein, der etwa Angriffe von Viren und Würmern oder auftauchende Probleme der Verfügbarkeit des Systems erkennen muss. Zudem werden die reaktiven Maßnahmen heute praktisch ausschließlich durch den Menschen manuell, der nur teilweise durch Systeme unterstützt wird, durchgeführt. Organisatorische Sicherheit wird selbstverständlich auch präventiv eingesetzt. Neben vielen anderen wirkungsvollen informatischen Methoden spielt sie dabei aber nur eine kleine Rolle. Eine Kategorisierung dieser wichtigen Maßnahmen bei der Erkennung erfolgt daher in der Tabelle 2.19, für Reaktion in der Tabelle 2.20.

Eigenschaft	Ausprägung bei *Erkennung*
LOC_T	Überwachtes System, nicht Umgebung
LOC_A	Beliebig, intern oder extern
SUB_A	Alle, außer überwachte Subjekte
SUB_T	Nur überwachte Subjekte
OBJ_A	Alle, außer überwachte Objekte
OBJ_T	Nur überwachte Objekte
CON_A	Angriff muss einem Muster entsprechen
TIM_A	Sehr langandauernde Angriffe können unerkannt bleiben
INT_A	System kann auch Mittel für anderen Angriff sein
VUL_T	Alle, explizit auch Verfügbarkeit

Tabelle 2.18: Einbruchserkenner

2.5 Klassifikation der Gegenmaßnahmen

Je nach Anwendung ist eine differenzierte Klassifizierung der möglichen Gegenmaßnahmen sinnvoll und nötig. Im vorigen Abschnitt wurden einige wesentliche bezüglich ihrer Realisierungseigenschaften eingeteilt und beschrieben, wie etwa der verwendeten Subjekte und Objekte, dem zeitlichen Verhalten und dem nötigen Kontext. Daneben ist es sinnvoll, die Maßnahmen in die drei festgelegten Phasen der Verteidigung einzuordnen, da damit eine Auswahl schneller getroffen werden kann. Diese Aufteilung fällt relativ einfach, allerdings ist stets zu beachten, dass einige Mechanismen in verschiedenen Phasen eingesetzt werden und wirksam sein können. Kryptografische Maßnahmen können beispielsweise sowohl Angriffe wie das Erspähen von vertraulicher Information auf einem Datenkanal verhindern als auch das Erkennen von Angriffen durch Integritätsprüfung ermöglichen. Sie werden deshalb in verschiedene Bereiche eingestuft.

2.5.1 Vermeidung und Verhinderung

Wünschenswert sind grundsätzliche, präventive Abwehrmaßnahmen gegen auftretende Angriffe, so dass man idealerweise vor bestimmten Angriffen prinzipiell geschützt ist. Tatsächlich existieren präventive Mechanismen, die in einem bestimmten Kontext einen gewissen Angriff unwirksam machen können. Es existieren allerdings auch präventive Mechanismen, die Angriffe zwar sehr gut abwehren können, dies allerdings nicht vollständig

Eigenschaft	Ausprägung bei *Erkennung*
LOC_T	Systeme und Umgebung des Administrators/Benutzers
LOC_A	Beliebig
SUB_A	Abhängig vom Angriff
SUB_T	Alle dem A/B bekannten
OBJ_A	Abhängig vom Angriff
OBJ_T	Alle dem A/B bekannten
CON_A	Intelligente Schlussfolgerung, A/B muss am Platz sein
TIM_A	Längere Angriffsdurchführung ist leichter zu erkennen
INT_A	Abhängig vom Angriff
VUL_T	Beliebig

Tabelle 2.19: Erkennung durch Organisationsaspekte

zuverlässig können.

In Anlehnung an die Begriffsbildung zur Verklemmungsbehandlung in Betriebssystemen werden die Begriffe „Angriffsvermeidung“ (kurz Vermeidung) und „Angriffsverhinderung“ (kurz Verhinderung) im Kontext der IT-Sicherheit wie folgt definiert.

Definition 2.17 (Angriffsvermeidung)
Die Angriffsvermeidung *bezeichnet alle präventiven Gegenmaßnahmen, die einen bestimmten Angriff prinzipiell unwirksam machen.*

□

Als Beispiel sei für die Angriffsvermeidung die Realisierung des virtuellen Speichers und des damit verbundenen Speicherschutzes in gängigen Rechnerarchitekturen genannt. Nicht privilegierte Subjekte können dabei nur virtuelle Adressen (auf Seitenbasis) benutzen, die vor dem eigentlichen Speicherzugriff auf den physikalischen Speicher von einer Einheit geprüft werden und in die entsprechende physikalische Adresse (auf Basis von Seitenkacheln) umgerechnet werden. Bei der Umrechnung, die im Allgemeinen von einem eigenen Hardwarebaustein auf Basis von geschützten Tabellen des Betriebssystems durchgeführt wird, wird auch die Gültigkeit des Zugriffs geprüft und gegebenenfalls verboten. Fremder Code, der im Prozesskontext ausgeführt wird, kann deshalb seinen privaten Adressraum nie verlassen, jeder Zugriff auf Speicher außerhalb seiner zugeteilten Seitenkacheln kann verhindert werden. Der Prozessschutz in modernen Betriebssystemen basiert im Wesentlichen auf dem Konzept des virtuellen Speichers und der korrekten Umsetzung in Software und

Eigenschaft	Ausprägung bei *Reaktion*
LOC_T	Systeme und Umgebung des Administrators/Benutzers
LOC_A	Beliebig
CON_A	Intelligente Reaktion, Risikoanalyse sofort intuitiv
TIM_A	Reaktion verteilbar, wird priorisiert durchgeführt
INT_A	Abhängig vom Angriff
VUL_T	Beliebig

Tabelle 2.20: Reaktion auf Basis von Organisationsaspekte

Hardware. Würde eine Schwachstelle entdeckt werden, so könnten Prozesse sich gegenseitig kompromitieren und, schlimmer noch, auch den Speicher des Betriebssystems angreifen und verändern, der im privilegierten Modus genutzt wird. Für moderne, dynamische Sprachen, die zur Ausführung eine virtuelle Maschine benötigen, werden andere, meist sprachbasierte Schutzkonzepte eingesetzt. Da die virtuelle Maschine jedoch als Prozess in einem herkömmlichen Betriebssystem realisiert ist, bestehen hier die selben Bedrohungen für den Fall, dass von der Realisierung des virtuellen Speichers Schwachstellen bekannt werden. Da bis heute keine derartigen Schwachstellen bekannt sind und die Umsetzung der virtuellen Speicherkonzepte zum Großteil in geprüfter, unveränderlicher Hardware stattfinden, spricht man von Angriffsvermeidung.

Definition 2.18 (Angriffsverhinderung)
Unter Angriffsverhinderung *versteht man alle präventiven Maßnahmen, die einem Angriff wirkungsvolle Mechanismen entgegenstellen, diesen aber nicht prinzipiell unwirksam machen können.*

□

Definition 2.19 (Präventive Gegenmaßnahmen)
Mit präventiven Gegenmaßnahmen *beschreibt man die Summe aller vermeidenden und verhindernden Maßnahmen.*

□

Mit der Verschlüsselung von Passwörtern in einer öffentlich lesbaren Datei, wie es lange Zeit unter Unix in der Datei `passwd` üblich war, kann ein Angriff durch das Auslesen des Passwortes verhindert werden. Im normalen Kontext ist es für einen Angreifer nicht möglich, aus dem verschlüsselten String das Passwort zu erraten. Allerdings ist die Funktion bekannt, die das Klartextpasswort in ein verschlüsseltes umwandelt. Sie kann deshalb beliebig oft von einem Angreifer angewendet werden. Dieser kann alle möglichen (und

wahrscheinlichen) Zeichenfolgen damit verschlüsseln und mit dem Passwort aus der Datei vergleichen. Da ein Angreifer mit genügend Rechenkapazität ein 8 Zeichen langes Passwort praktisch automatisch erraten kann, spricht man hier von Angriffsverhinderung.

Es ist keine Regel bekannt, nachdem man aufgrund des Mechanismus oder der Strukturierung im System eine Gegenmaßnahme als vermeidend oder verhindernd einstufen kann. Je nach Anwendung und Kontext ändert sich die Einteilung. Oft spricht es jedoch für Angriffsvermeidung, wenn eine Maßnahme in einem in Schichten strukturierten System in einer Schicht unterhalb des zu schützenden Subjekts oder Objekts angesiedelt ist. Sicherheitsmechanismen, die das Betriebssystem den Anwendungen zur Verfügung stellt, sind häufig verhindernd.

Maßnahmen, die während der Entwicklung von abgesicherten Systemen eingesetzt werden, zählen zu den präventiven, vermeidenden oder verhindernden, Maßnahmen. Dazu zählen alle Vorgehensweisen beim Security Engineering, durchgeführte Schwachstellen-, Bedrohungs- und Risikoanalysen und administrative und betriebliche Maßnahmen. Nähere Informationen dazu finden sich unter anderem in [And01].

2.5.2 Erkennung

Im Unterschied zu verhindernden und vermeidenden Maßnahmen beruhen die erkennenden darauf, dass die Aktion eines Angreifers sichtbar bzw. erkennbar ist. Das System muss dazu vorbereitet sein, wenn die Erkennung durch Software beziehungsweise Hardware unterstützt werden soll und nicht durch Personen manuell erfolgt. Die Erkennung findet immer mit einem gewissen Zeitversatz zwischen der Aktion des Angreifers und dem zweifelsfreien Erkennen statt. In vielen Fällen muss diese Zeitdifferenz so gering gehalten werden, dass entsprechend reagiert werden kann, was zu folgender Definition führt.

Definition 2.20 (Angriffserkennung)
Unter Angriffserkennung *versteht man alle Maßnahmen, die eine Aktion des Angreifers erkennen, die zu einer Schutzzielverletzung führen kann.*

□

Wichtig ist, dass die präventiven Maßnahmen und die zur Erkennung notwendigen nicht streng getrennt voneinander betrachtet werden, da auch präventive Maßnahmen Daten gewinnen können, die für Erkennung wichtig sind. Der Pförtner eines Betriebssystems kann beispielsweise häufige Fehlversuche beim Anmelden eines Benutzer an das Erkennungssystem weiterleiten und dient sowohl als präventive, als auch als erkennende Maßnahme.

2.5.3 Reaktion

Enger Zusammenhang besteht zwischen der Erkennung und der Reaktion auf Angriffe, da die Erkennung nur dann sinnvoll ist, wenn auch entsprechend automatisch reagiert wird. Hier existiert auch ein Zeitversatz zwischen dem Zeitpunkt des Erkennens, der Auswahl und der Durchführung der geeigneten Reaktion. Dieser ist entscheidend für die Wirksamkeit der Gegenmaßnahmen.

Definition 2.21 (Reaktion auf Angriffe)
Die Reaktion auf Angriffe *fasst alle Maßnahmen zusammen, die die Wirkung einer erkannten Aktion des Angreifers rückgängig oder unwirksam machen.*

□

Auffallend ist die Unterbesetztheit der Methoden zur Reaktion auf Angriffe. Ist der Sinn von präventiven Maßnahmen unbestreitbar, so muss trotzdem über den Sinn der erkennenden Verfahren, wie etwa Einbruchserkennungssysteme, nachgedacht werden, die nicht selbständig auf erkannte Gefahren reagieren. Die Hilfe eines Menschen, zum Beispiel des Administrators, ist dann unverzichtbar für einen wirkungsvollen Schutz. Allerdings kann deren Verhalten und Reaktion sowohl fehlerhaft als auch in wiederkehrenden Fällen unterschiedlich und im schlimmsten Fall nicht nachvollziehbar werden. Entwicklungen in diesem Bereich sind wünschenswert und der steigende Aufwand der Administration der Systeme werden sie fördern.

2.5.4 Komposition

Wie gezeigt kann jeder Angriff in die drei Phasen Vorbereitung, Durchführung und Nutzen strukturiert werden. Die beiden letzten Phasen bauen jeweils auf dem Ergebnis der vorherigen Phase auf und können nur dann starten und durchgeführt werden, wenn diese erfolgreich war. Ist der Angriff in eine Phase eingetreten, kann er von dort aus oft wiederholt werden, so dass eine erfolgreiche Durchführung einer Angriffsphase von Gegenmaßnahmen nur schwer wieder rückgängig gemacht werden kann. Fasst man die gewonnenen Kenntnisse über die allgemeinen drei Phasen des Angriffs und der Verteidigung zusammen, so ist es naheliegend, dass die präventiven Gegenmaßnahmen die entsprechende Angriffsphase beeinträchtigen sollen. Die präventiven Maßnahmen sollen der Vorbereitungsphase entgegenstehen, die Erkennung denen der Durchführung und die Reaktion beeinflusst den Nutzen des Angriffs. Bei genauer Betrachtung wird jedoch klar, dass dieser direkte Zusammenhang ungenügend ist, da beispielsweis auch in der Vorbereitungsphase des Angriffs erkennende Maßnahmen ein Rolle spielen können. Die Betrachtung der Phasen des Angriffs und der Verteidigung wird deshalb wie folgt betrachtet.

In jeder der drei Phasen des Angriffs können alle drei Phasen der Gegenmaßnahmen wirken. Die Angriffsvorbereitung kann sowohl durch präventive Maßnahmen verhindert als auch

erkannt und darauf reagiert werden. Gleiches gilt für die Angriffsdurchführung und auch die Nutzenphase, in der allerdings die Reaktion auf eine erfolgreiche Benutzung der erbeuteten Daten eigentlich keine Anwendung technischer Mittel bedeuten kann, eher juristische Schritte. Dieser Zusammenhang wird in Abbildung 2.14 dargestellt.

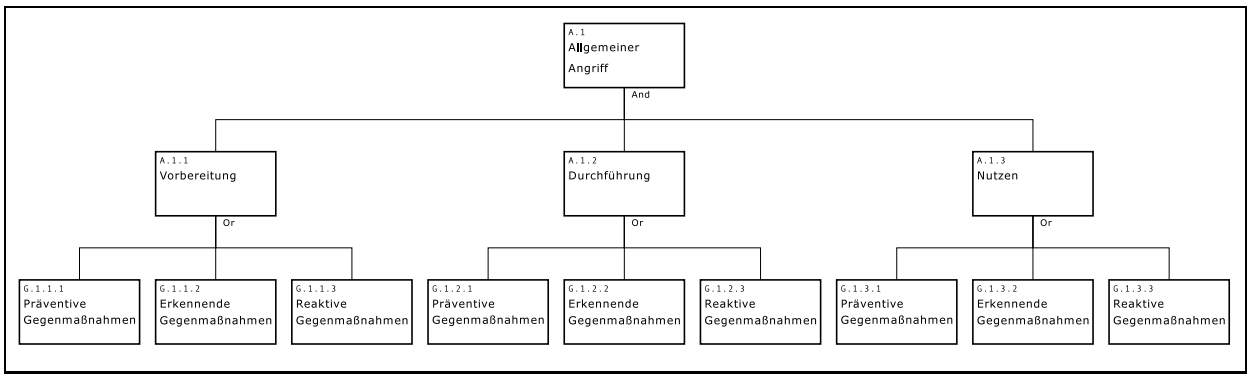

Abbildung 2.14: Einteilung der verschiedenen Kategorien von Gegenmaßnahmen

Auf Basis der hier vorgestellten Klassifikation der verfügbaren Gegenmaßnahmen wird in Kapitel 4 eine Beschreibungstechnik entwickelt, die die einzelnen Methoden des Katalogs auf dieselbe Art beschreiben, wie Angriffe beschrieben werden. Dadurch lassen sich zu erkannten möglichen Angriffen einfach wirkungsvolle Gegenmaßnahmen finden.

Kapitel 3

Abgesicherte Betriebssysteme

Ziel dieser Arbeit ist die methodische Entwicklung eines abgesicherten Betriebssystems. In ersten Teil dieses Kapitels werden daher zunächst Betriebssysteme für allgemeine Rechnerarchitekturen vorgestellt, die bestimmte Sicherheitsanforderungen abdecken, wie sie beispielsweise auch in [Irv97] gefordert werden. Die Betriebssysteme sind durch unterschiedliche Maßnahmen gegen bestimmte Angriffe abgehärtet. Im zweiten Teil wird Betriebssoftware vorgestellt, die für den Bereich der mobilen verteilten Systeme entwickelt werden und diskutiert, welche wirkungsvollen Sicherheitsmechanismen darin enthalten sind. Unter Betriebssoftware wird dabei die Gesamtheit der Systemsoftware verstanden, die sowohl die Aufgaben eines klassischen Betriebssystems wahrnimmt, als auch den Aspekt der Mobilität im Betrieb unterstützt. Neben den üblichen Betriebssystemaufgaben fallen darunter auch Kommunikations- und Verbindungsmanagement. Die Ergebnisse der Analysen werden als Klassifizierungen in tabellarischer Form festgehalten.

3.1 Klassische Betriebssysteme

Betriebssysteme lassen sich nach verschiedenen Kriterien unterteilen, wie etwa nach der Komplexität, der Größe, der internen Architektur oder der eigentlichen Aufgabe, beispielsweise in einem speziellen Anwendungsgebiet. Unabhängig davon existiert folgende Definition des Begriffs nach DIN 44300.

Definition 3.1 (Betriebssystem)
Die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen nennt man Betriebssysteme.

□

Da nach der Definition Betriebssysteme die Grundlage für Rechensysteme bilden und sowohl für die Steuerung, als auch deren Überwachung zuständig sind, liegt es nahe, Sicherheitsmechanismen darin zu verankern.

3.1.1 Überblick

Grundlegend für die Funktion von modernen Rechensystemen sind die Betriebssysteme. Diese bilden die Schnittstelle zwischen der Hardware und den eigentlichen Anwendungen des Rechensystems. Betriebssysteme abstrahieren bzw. veredeln die zugrundeliegende Rechnerhardware und liefern zusätzliche Funktionalität, sowohl für die Anwendungen, als auch für die Hardware.

Einerseits wird der Zugriff der Anwendungen auf die Hardware geregelt, wodurch es möglich wird, dass mehrere Applikationen praktisch gleichzeitig kontrolliert auf die Geräte zugreifen können (auch Hardware-Multiplexing genannt). Wichtiges Beispiel hierfür ist die Kontrolle gemeinsam benutzen Speichers, wie Festplatte oder der Hauptspeicher. Andererseits werden die Zugriffe zwischen Anwendungen verwaltet und kontrolliert, etwa durch eine effiziente Prozessverwaltung und Speicherverwaltung. Einen Überblick über die allgemeinen Schutzmaßnahmen für Betriebssysteme findet sich in [SSd75].

Daneben werden vom Betriebssystem Mechanismen zur Synchronisation bereitgestellt. Diese kontrollieren den Zugriff mehrerer Prozesse auf gemeinsame Ressourcen derart, dass Verklemmungen (engl. Deadlocks) und Probleme des wechselseitigen Ausschlusses weitgehend vermieden[1] werden können.

Die Komplexität von Betriebssystemen ist hoch. Neben den bekannten Funktionen, die im Folgenden kurz erläutert werden, besitzen sie noch viele weitere, die von Anwendungen auch direkt verwendet werden können, wie beispielsweise Dateisysteme. Betriebssysteme werden üblicherweise in funktionale Schichten strukturiert, wodurch die einzelnen Teile überschaubar bleiben. Jede einzelne Schicht erfüllt spezielle Aufgaben und ist idealerweise über definierte Schnittstellen mit den angrenzenden Schichten verbunden.

Die Kenntnis über die grobe Entwicklung der Betriebssystemgenerationen ist für die Architektur der Betriebssysteme für moderne, mobile verteilte Systeme sehr hilfreich. Die ersten mit den heutigen Rechnern vergleichbaren Computer, die Mainframes, wurden zunächst in Assembler programmiert, besaßen keinerlei Möglichkeiten zur gleichzeitigen Ausführung von Programmen und ebenso keine Schutzmechanismen zwischen verschiedenen Prozessen. Später kamen virtueller Speicher, Multitasking und persistenter Speicher, wie Festplatten dazu. Bei den Betriebssystemen wurden der virtuelle Speicher, Interprozesskommunikationsmechanismen und Mehrbenutzermodus später entwickelt [Tan02], wie in Abbildung 3.1

[1] Ein echtes prinzipielles Ausschließen von Verklemmungen kann nur durch hohen Aufwand und starke Einschränkungen durchgeführt werden, weshalb es in gängigen Workstation-Betriebssystemen nicht realisiert ist.

dargestellt ist.

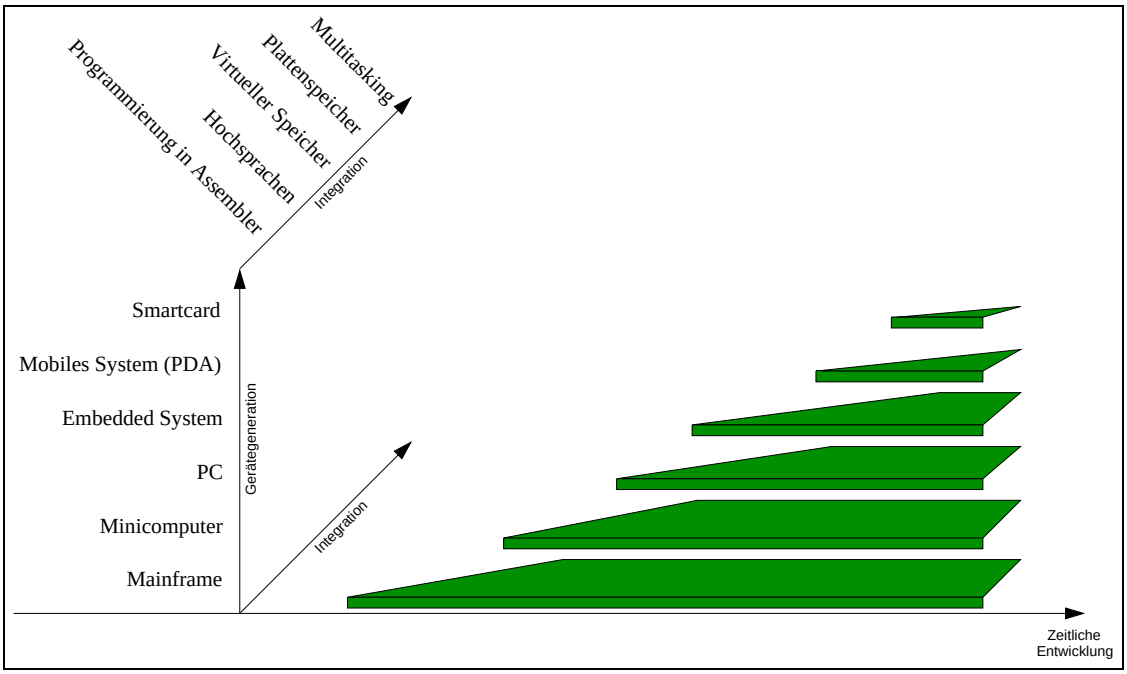

Abbildung 3.1: Entwicklung der Betriebssysteme

Nach den Mainframes wurden die Minicomputer, die PCs und die Rechner für eingebettete Systeme entwickelt. Diese machten alle die selben Entwicklungsstufen durch wie die Mainframes Jahre vorher.

Diese Zyklen kann man auch bei der Entwicklung von mobilen Endgeräten erkennen. Konzepte wie virtueller Speicher oder Multitasking sind sowohl bei den PDAs vorhanden, als auch mittlerweile mit Smardcards möglich geworden, siehe dazu auch [RE02]. Prinzipiell sind bei den mobilen persönlichen Geräten auch Festplatten vorhanden, diese allerdings nicht in Form von mechanischen Scheiben, sondern in Form von Flash-Speicher.

Insgesamt war die Entwicklung der Betriebssysteme und die der übrigen Software aber getrieben bzw. gebremst durch die zur Verfügung stehende Technologie. Deshalb kann man davon ausgehen, dass die Entwicklung der Betriebssysteme für mobile Endgeräte mit der Entwicklung der Hardware und der Technologie weitergehen wird. Für die Ergebnisse dieser Arbeit ist festzustellen, dass sie demzufolge nicht auf die Betriebssysteme für mobile Geräte eingeschränkt zu sehen sind, sondern ebenso für die stationären Systeme anwendbar sind.

Unabhängig von der Evolutionsstufe der Betriebssysteme stellt die Realisierung der Sicherheit für diese Systeme eine große Herausforderung dar. Ein Grund für den hohen Aufwand ist deren Komplexität, die wegen der unterschiedlichen Dienste für Anwendungen und Hardware sehr hoch ist. Die sich daraus ergebenden zahlreichen Schnittstellen bieten An-

greifern mögliche Angriffspunkte und deren vollständige Überprüfung und Absicherung ist aufwändig. Ein weiterer Grund ist, dass die Umgebung, in der Betriebssysteme eingesetzt werden, vielseitig ist. Es ist deshalb nur schwer möglich, bestimmte Angriffe von Anfang an auszuschließen. Sicherheitsmechanismen in Betriebssystemen müssen daher für eine hohe Bandbreite von Anwendungen wirksam sein, was der Vergleich der Betriebssysteme in den folgenden Abschnitten zeigt.

3.1.2 Wesentliche Mechanismen und Konzepte

Zur Absicherung von Betriebssystemen sind verschiedenste Ansätze möglich und realisiert worden. Für Sicherheit gegen externe Angreifer könnten Gruppen von Benutzern mit gleichen Interessen und Vertrauensbereich jeweils eine eigene physikalische Maschine zur Verfügung gestellt werden. Die Kommunikation nach außen wird unter anderem durch kryptografische Protokolle in Bezug auf Abhören und Verfälschen gesichert. Allerdings muss die Kommunikation innerhalb des Systems kontrolliert und gesichert sein. Diese inneren Kontrollen benötigen eine andere Lösung. Diese Kontrollen zu realisieren ist so schwierig, dass es im Wesentlichen bisher nicht ausreichend gelungen ist. Nur für Teile lassen sich solche Kontrollen durchsetzen, aber nicht für das Gesamtsystem. Dafür sprechen nach [AGS83] folgende drei Gründe:

1. Betriebssysteme sind im Allgemeinen große, komplexe Softwaresysteme,
2. die zu erreichende Sicherheit durch die Kontrollen wurden nicht exakt festgelegt und
3. die Sicherheitskontrollen und Mechanismen wurden nur wenig auf Korrektheit hin geprüft.

Für eine allgemeine, integrierte Sicherheitsarchitektur ist ein Modell des Betriebssystemes wünschenswert, in das die Architektur eingebaut werden soll. Dadurch können konzeptionelle Sicherheitslücken vermieden werden, die aufgrund der hohen Komplexität entstehen, die Kontrollen sind für Anwender nachvollziehbar, exakt festgelegt und die Sicherheitsmechanismen können anhand des Modell geprüft werden.

3.1.2.1 Allgemeines Modell für Betriebssysteme

Da Betriebssysteme komplexe Systeme sind, liegt es nahe, dass man ein Modell eines möglichst allgemeinen Betriebssystems als Grundlage für die Entwicklung oder Erweiterung benutzt. Das Modell sollte zunächst die allgemeinen Funktionalitäten und Anforderungen abbilden können und auch für den speziellen Zweck anpassbar sein. Die Freiheitsgrade bei der Entwicklung eines solchen Modells sind prinzipiell sehr hoch, weshalb fraglich ist, ob man im Allgemeinen eine genügende Beschreibung finden kann. Wenn nur Betriebssysteme

mit allgemeinsten Anforderung betrachtet werden, sind die Freiheitsgrade so hoch, dass es sehr allgemein ist. Für den Zweck der Sicherheitsanalyse und -Realisierung wäre dieses Modell nicht angepasst genug, da viele Angriffe nur in sehr detailierter Modelldarstellung erkennbar werden.

Aber dadurch, dass die Anforderungen an diese Systeme gestiegen sind und sowohl funktionale, als auch nicht funktionale Standardanforderungen berücksichtigt werden müssen, werden die Möglichkeiten für die Wahl des Modells stark eingeschränkt. Die Anforderung beispielsweise, dass mehrere Anwendungen gleichzeitig im Speicher verweilen und quasiparallel ausgeführt werden, ohne dass sie sich gegenseitig beeinflussen, legt fest, dass ein Speicherschutzkonzept benutzt werden muss. Die Verwendung persistenten Speichers legt fest, dass irgendeine Art von Dateisystem integriert werden muss. Nicht funktionale Anforderungen, wie etwa die Gewährleistung der Vertraulichkeit der gespeicherten Daten legen wiederum fest, dass Daten klassifiziert und durch Zugriffskontrollen geschützt werden müssen.

Der Vorteil eines entwickelten Betriebssystemmodells wäre für diese Arbeit, dass man Sicherheitskonzepte in das Modell integrieren könnte und aus dem erweiterten Modell das Betriebssystem generieren kann. Zudem können aus Änderungen der Anforderungen die Modelle geändert werden und, möglichst automatisch, neue Instanzen des Betriebssystems erzeugt werden. Für Teilbereiche der Betriebssysteme existieren passende Modelle. Die Prozessverwaltung kann durch Prozesssemantiken und Prozesssysteme beschrieben werden. Für die Darstellung von Synchronisationsproblemen existieren zahlreiche Beschreibungsarten, wie etwa Petri-Netze. Die Beschreibungstechniken und damit die entwickelten Modelle beschränken sich aber lediglich auf einen Teilaufgabenbereich der Systeme, sie sind nicht Teilsystem-übergreifend anwendbar. Es liegt nahe, andere Bereiche der Informatikanwendungen zu betrachten und Modelle und Ideen daraus zu verwenden.

Dem Compilerbau liegt die Sprachentheorie zugrunde. Damit hat man ein theoretisches Modell und eine Metrik für Sprachen, die man mit Übersetzern weiterverarbeiten kann. Der Vorgang des Übersetzens kann derart berechnet werden, dass die Erzeugung des ausführbaren Codes korrekt ist. Die Phasen des Compilers können exakt beschrieben werden, weshalb man sich bei der Erstellung von Übersetzern auf diese Modelle abstützen kann. Die Verwandtschaft zu den Betriebssystemen ist dadurch unterbrochen, dass die Eingabedaten, der Quelltext für die Übersetzung, statisch ist und der eigentliche Übersetzungsvorgang nie durch nicht deterministische Benutzereingaben beeinflusst wird. Es existieren mächtige Modelle für den Übersetzerbau, Modelle aus dem Bereich der Datenbanken sind für die Anwendung im Betriebssystembereich eher geeignet.

Im Bereich der relationalen Datenbanken ist die Algebra festgelegt und exakt beschrieben, die dem Datenmodell zugrundeliegt. Datenbanken sind für die Verwaltung großer Datenmengen, die Synchronisation des Zugriffs auf die Daten durch Transaktionen und die Sicherheitskontrolle der Zugriffe zuständig und haben damit sehr ähnliche Aufgaben wie Betriebssysteme. Tatsächlich existieren einige interessante Zugriffskontrollmodelle, die im

Datenbankbereich eingesetzt werden und als Vorbild für Kontrollen bei den Betriebssystemen dienen können. Wichtiger Vertreter davon sind die Rollenbasierten Zugriffskontrollmodelle, wie sie beispielsweise in [AS00], [OSM00] und [San98] behandelt werden. Durch den großen Unterschied aber, dass sich Datenbanken auf ein definiertes Datenmodell, beispielsweise das Relationenmodell in relationalen Datenbanken, verlassen können, ist es nicht möglich, ein Modell für allgemeine Betriebssysteme daraus zu entwickeln.

Betriebssysteme haben das Ziel der kontrollierten und gesicherten Verwaltung und Vermittlung zwischen Hardware und Anwendungen, wobei die Einheiten Anwendung, Middleware, Daten und Kommunikation wegen universeller Nutzung nur generisch festgelegt sind, ganz im Gegensatz beispielsweise zu Datenbanken. Zudem sind die verwalteten Strukturen dynamisch und zur Laufzeit veränderlich. Das Modell muss diese Dynamik darstellen können, ohne dass man bei der Entwicklung alle Möglichkeiten dynamischer Einflüsse kennt. Nur Teile der Dynamik können aus den statischen Strukturen entwickelt werden, wie beispielsweise aus den zur Verfügung stehenden Quelltexten. Andere Dinge, wie etwa Benutzereingaben, sind unbekannt oder erst zur Laufzeit bekannt. Ein allgemeines Modell für Betriebssysteme zu entwickeln wäre für diese Arbeit zu aufwändig. Für die Entwicklungen der funktionalen Eigenschaften von Betriebssystemen existieren derartige Modelle, die allerdings auf den Einsatz zur Sicherheitsanalyse und der Prüfung nicht funktionaler Eigenschaften nicht vorbereitet sind. Stattdessen werden die Möglichkeiten des Systems auf mobile verteilte Systeme mit einer vorgegebenen Systemcharakteristik (Prozessmodell, Anwendungsbereich und zur Verfügung stehende Hardware) beschränkt.

3.1.2.2 Kern-Architekturprinzipien

Bei der Entwicklung aller Betriebssysteme und speziell der Kernbereiche ist es eine wichtige Festlegung, welche Funktionalitäten in welchen Teilen des Systems angesiedelt werden und wie diese Bereiche voneinander getrennt werden. Eine Diskussion darüber findet sich etwa in [CD95]. Bekannt sind die Mikrokern- und die Monolithischen-Architekturen, die entweder einen möglichst geringen Funktionsumfang innerhalb des am höchsten priviligierten Modus besitzen oder andererseits alles in einem vor Angriffen geschützten Adressraum ausführen. Es existieren noch eine Vielzahl anderer Architekturen, die die Aufteilung der Funktionalität weiter treiben, wie etwa Nano- oder Exo-Kerne. Die Frage, die bei der Entwicklung dieser Architekturkonzepte im Hintergrund steht ist, wieviel der Strukturierungen aus der Entwicklungszeit und dem Quelltext sich in das übersetzte und ausgeführte System übernehmen lässt. Bei Monolithen geht die Strukturierung fast vollständig verloren, im Bereich der Mikrokerne bleibt sie an den geteilten und getrennten Adressräumen für die einzelnen, nicht priviligierten Server sichtbar. Oberflächlich betrachtet scheinen die Systeme, die ihre Strukturen auch zur Laufzeit nicht verlieren, große Vorteile zu haben, wie etwa die Übersichtlichkeit oder die Austauschbarkeit der Komponenten zur Laufzeit, ohne den gesamten Kern neu zu starten. In der Praxis sind diese gedachten Vorteile meist nicht mehr erkennbar, da es zur Laufzeit eher um Stabilität und Ausführungsgeschwin-

digkeit geht, als um eine Austauschbarkeit von Kernkomponenten, die meist keine Vorteil bieten kann. Hier haben die Mikrokerne sogar den entscheidenden Nachteil, dass sie bei der Ausführung und Nutzung von Diensten je nach Aufbau wesentlich mehr Kontextwechsel benötigen, die die Ausführungszeit deutlich erhöhen. Der einzige Vorteil, den man für die strukturierten Mikrokerne nennen kann, ist der modulare Aufbau der einzelnen Komponenten. Dies spielt für funktionale Aspekte allerdings weniger eine Rolle, als für die Sicherheit. Der große Fortschritt ist, dass die kleinen, einzelnen und mit wohldefinierten Schnittstellen versehenen Komponenten leichter überprüfbar, validierbar und verifizierbar sein können, als ein großer monolithischer Kern. Interessanterweise ist aber genau diese Eigenschaft bei der Konstruktion bis heute nicht ausgenutzt worden. Obwohl für die Überprüfbarkeit und Beweisbarkeit sicherer Systeme kleinere Kernarchitekturen offensichtlich besser geeignet scheinen, ist die Betriebssystem-Architektur, die in dieser Arbeit entwickelt und erweitert wird, eine modulare monolithische Architektur. Dies hat im Wesentlichen zwei Gründe. Einerseits wäre die Neuentwicklung eines Systems zu aufwändig. Andererseits ist das Ziel der Arbeit der Analyseteil, die Umsetzung zeigt lediglich die Realisierbarkeit und die Verfikation gehört nicht Teil dieser Arbeit. Zudem ist das entwickelte Vorgehen mit anderen Kernarchitekturen vereinbar.

3.1.2.3 Speicherschutz

Zur Sicherstellung von unterschiedlichen Vertrauensbereichen und dem Schutz der einzelnen Prozesse gegeneinander ist ein Speicherschutzverfahren nötig. Im Allgemeinen existieren mehrere Möglichkeiten zur Realisierung, wie virtueller Speicher, Objektschutzverfahren oder strenge Typprüfung durch die verwendete Programmiersprache. Wie auch in [And72] diskutiert wird, will man sich aufgrund der Flexibilität nicht auf eine Programmiersprache festlegen, womit Ansätze dieser Art im Allgemeinen nicht in Frage kommen. Andere Ansätze durch Software Isolation werden beispielsweise in [WLAG93] und [ZZNM02] angesprochen. Solange Programmiersprachen wie C oder C++ im System verwendet werden, die nicht typbehaftete Pointerarithmetik zulassen, sind Speicherschutzverfahren nötig, die auf Betriebssystemebene jeden Speicherzugriff kontrollieren und verbieten können. Der reine Objektspeicherschutz ist nicht für systemnahe Teile wirksam, so dass im Wesentlichen virtuelle Speicherschutzkonzepte eingesetzt werden müssen. Dies ist auch deshalb von großem Vorteil, weil das Konzept in großen Teilen durch Hardware durchgesetzt wird, die von Software aus nicht zu kompromitieren ist. Die nötige Hardware ist in praktisch allen modernen Systemarchitekturen vorhanden, mittlerweile hält sie auch Einzug in mobile Endgeräte und Chipkarten. Die Klassifikation des virtuellen Speicherkonzeptes zum Zwecke der präventiven Sicherstellung von Vertrauensbereichen wird in der Tabelle 3.1 dargestellt. Es existieren zudem andere Mechanismen, die Speicherkontrollen durchsetzen, wie etwa Architekturen für Einadressräume. Diese werden in zunehmendem Maße mit der Einführung von 64 Bit Architekturen interessant und werden beispielsweise in [KCE92] beschrieben. Die Abhängigkeit der Betriebssystem-Sicherheitsarchitekturen von der Hardware sind beispielsweise in [SPL95] und [Sch83] zu lesen.

Eigenschaft	Ausprägung bei *Präventiv*
LOC_T	Im niedriger priviligierten Speicherbereich als die Verwaltung
LOC_A	Interner Angreifer außerhalb des Speicherbereichs
SUB_A	Beliebig aber keine priviligierten Subjekte
SUB_T	Prozesse und Threads im niedriger priviligierten Speicherbereich
OBJ_A	Beliebig, aber keine priviligierten Objekte
OBJ_T	Daten (Keller oder Halde) im niedriger priviligierten Speicherbereich
CON_A	Prozesskontext ist statisch
TIM_A	Beliebig
INT_A	Nutzen durch Zerstörung des Vertrauensbereichs
VUL_T	Fehler in virtueller Speicher-Implementierung

Tabelle 3.1: Präventive Sicherstellung von Vertrauensbereichen

3.1.2.4 Trusted Computing Base

Ein Ansatz, um die Komplexität des abgesicherten Betriebssystems in den Griff zu bekommen, ist die Verkleinerung der Teile des Systems, die die Sicherheit kontrollieren und durchsetzen. Wesentlicher Bestandteil dieses Systemaufbaus ist der Referenz Monitor.

Definition 3.2 (Referenz Monitor)
Ein Referenz Monitor *ist ein Konzept zur Regelung der Zugriffskontrolle von Subjekten auf Objekte.*

□

Definition 3.3 (Security Kernel)
Ein Security Kernel *ist die Kombination aus Hardware und Software, die das Konzept eines Referenz Monitors umsetzen.*

□

Diese Sicherheitskerne waren die ersten Vertreter von Architekturen, die die Sicherheitsmechanismen zur Zugriffskontrolle von den übrigen Systemkomponenten getrennt haben, wie auch in [AGS83] zu lesen ist. Im Laufe der Zeit wurden zusätzliche Sicherheitsmechanismen

in die Sicherheitskerne integriert, wie beispielsweise Kryptografie oder Logging-Funktionen.

Definition 3.4 (Trusted Computing Base (TCB))
Die Trusted Computing Base *besteht aus allen Hardware- und Software-Komponenten, die für die Durchsetzung der Sicherheitspolitiken zuständig sind.*

□

Aus den oben genannten Gründen, leichtere Überprüfbarkeit und damit erhöhtes Vertrauen, ist das Zusammenfassen der Sicherheitsmechanismen in einen Sicherheitkern wünschenswert, wie auch in [AGS83] beschrieben wird. Ziel bleibt es dabei, durch definierte Schnittstellen eine klare Trennung zu den übrigen Mechanismen zu erreichen und die TCB klein zu halten. Es ist jedoch bei der Realisierung der Systeme oft schwierig, eine strikte Trennung zu erreichen, da Anforderungen wie die Ausführungsgeschwindigkeit und die Erweiterbarkeit eine große Rolle spielen und dem aus naheliegenden Gründen entgegenstehen. Zur Klassifizierung lässt sich im Wesentlichen sagen, dass die TCB ein Gerüst für die einzusetzenden Sicherheitsmechanismen darstellen und sowohl durch präventive, als auch durch erkennende Wirkung eine große Rolle spielen.

3.1.3 Beispiele abgesicherter Betriebssysteme

Es existieren zahlreiche Entwicklung im Bereich der sicheren Betriebssysteme. Einige davon, die das wesentliche Spektrum der Entwicklungen darstellen, werden im Folgenden dargestellt. Dabei stehen die Sicherheitskonzepte und Mechanismen im Vordergrund, spezielle Funktionalitäten werden nicht betrachtet. Zur Einordnung wird jedes untersuchte System in einer Tabelle klassifiziert, in der die wesentlichen Konzepte zusammengefasst sind.

Entwicklungsmethode HDM

Als allgemeine Entwicklungs- und Implementierungsmethode wurde HDM (Hierarchical Development Methodology) nach [CGHM81] entwickelt. Das Ziel dabei ist den gesamten Entwicklungsprozess zu automatisieren und formalisieren. Wie beispielsweise in [Bis03] beschrieben wird, war das Ziel, dadurch zuverlässige, verifizierbare und leicht wartbare Software zu entwickeln. Das generelle Konzept bei HDM ist die stetige Verfeinerung und Überprüfung der Spezifikation. Diese beschreibt zunächst die Anforderungen als Requirements, welche dann in ein Modell transformiert werden. Das Modell und alle weiteren Verfeinerungsschritte, die in der eigenen Sprache HSL (Hierarchy Specification Language) dargestellt werden, können automatisch auf Konsistens überprüft werden.

Innerhalb einer Ebene werden die Systeme als sogenannte Abstrakte Maschinen beschrieben, die durch Module aufgeteilt sind. Diese Module vereinen einerseits jeweils mehrere

zusammengehörige Funktionen und können andererseits wiederverwendet werden. Dafür nutzt man die Beschreibungssprache SPECIAL (Specification and Assertion Language), die auch in [Dis81] näher beschrieben wird. Die zugrundeliegende Semantik in SPECIAL wird durch Zustandsautomaten ausgedrückt, die aus einer Menge von Zuständen und Zustandsübergängen bestehen. Innerhalb einer Abstraktionsebene wird mindestens eine Abstrakte Maschine mit jeweils mindestens einer Modulspezifikation definiert. Die Beziehung zwischen den einzelnen Abstraktionsebenen und Abstrakten Maschinen wird durch sogenannte Mapping Spezifikationen ebenfalls in der Sprache SPECIAL definiert. Das Elegante ist die Softwareunterstützung bei der Entwicklung. Es existieren dazu einige Programme für die Verwendung in HDM, die alle Spezifikationen in SPECIAL sowohl syntaktisch, als auch auf Konsistenz hin automatisch prüfen können.

Die Schnittstellenspezifikationen der einzelnen Module werden durch kleine Programme realisiert, die nach Übersetzung als Bedingungen zur Verifikation dienen. Das Model zur Multilevel-Security ist ein Bell-LaPadula-Modell mit einigen Änderungen, die aufgrund der Toolunterstützung notwendig sind. Ein detailierter Vergleich zwischen Bell-LaPadula und dem SRI-Modell findet sich in [Tay84].

Die Entwicklung von Betriebssystemen mit formalen Methoden, wie beispielsweise HDM, ist wünschenswert, wenn ein System vollständig neu entwickelt wird und bestimmte Aspekte exakt nachgewiesen werden müssen. Neben der Nachweisbarkeit von Eigenschaften besitzen solche Methoden auch den großen Vorteil, dass während der Entwicklung exakte Spezifikationen zur Software entstehen und die Entwicklung nachvollziehbar bleibt. Allerdings entstehen dadurch auch Nachteile, die in ihrer Gesamtheit schließlich dazu führten, dass heutige Standard-Betriebssysteme nicht mit derartigen Methoden entwickelt wurden. Hauptkritikpunkt ist die hohe Komplexität bei der Entwicklung und das Erlernen der Methode durch die Entwickler. In Ausnahmebereichen, wie etwa Hochsicherheitsbereichen, im militärischen Bereich oder in der Luft- und Raumfahrt mag dieses Kostenargument nicht entscheidend sein. Bei Betriebssystemen für den normalen Endbenutzer sind die Kosten zu hoch. Zudem geht bei diesen Standardsystemen die Entwicklung stetig weiter, es sollen stets aktuelle Hardware und neueste Software eingesetzt werden können. Dies verursacht aber einen hohen Aufwand, da sämtliche HDM-Abstraktionsebenen wiederholt erweitert, geprüft oder neu entwickelt werden müssen. Außerdem sind bei der Entwicklung abgesicherter Betriebssysteme niemals alle Angriffe und Bedrohungen auf das System bekannt. Sollten durch den konsequenten Einsatz formaler, exakter Methoden tatsächlich jedes Auftreten neuer Angriffe komplett ausschließen, würde dies eine Modellierung der gesamten Welt bei der Entwicklung[2] bedeuten, was verständlicherweise nicht durchführbar sein kann. Durch die Weiterentwicklung von HDM zur Enhanced HDM (EHDM) wurden einige Nachteile behoben, da jedoch kein Betriebssystem bekannt ist, welches mit EHDM entwickelt wird, wird diese Methodik nicht näher beschrieben. Weitere Diskussionen zu der Entwicklungsmethodik HDM findet sich in [FN79] und [Ano78].

[2]da beispielsweise auch Angriffe wie etwa Krankheiten der Benutzer und Verwalter einbezogen werden müssten

PSOS

Das Betriebssystem PSOS (Provably Secure Operating System) wurde in den 70er Jahren ohne Festlegung auf eine reale Hardware entwickelt. Das Schutzkonzept besteht aus einer kompromisslosen Verwendung von Capabilities, die genau aus zwei Elementen bestehen: einem eindeutigen Identifizierer (UID) und einer Menge von Zugriffsrechten in Form eines boolschen Arrays. Sobald eine Capability erzeugt ist, kann sie niemals verändert werden. Es existieren genau zwei Operationen, die Capabilities erzeugen können. Erstens die Funktion `create_capability()`, die eine neue UID mit allen Rechten des Aufrufers erzeugt, zweitens die Funktion `restrict_access(Capability` c`, RightMask` $rmask$`)`, die eine neue Capability mit der selben UID von c und mit den Rechten von c ohne die angegebenen Rechte $rmask$ erzeugt. Mit diesem Operationen ist es möglich, neue Capabilities zu erzeugen, die höchstens dieselben Rechte wie der Aufrufer besitzt, aber niemals mehr besitzen kann. Zur Verwendung der Capabilities innerhalb des Systems werden die einzelnen Ressourcen wie folgt in eine Hierarchie eingeteilt, wie sie auch in [FN79] beschrieben wird. An der priviligiertesten Ebene (PSOS Level 0) stehen die Capabilities selbst. Sie können von keinem weniger priviligiertem Subjekt verändert oder zerstört werden. In den Stufen darunter befinden sich die physikalischen und virtuellen Ressourcen, der Objektmanager bis zu den Benutzerobjekten. Da nicht alle Elemente einer Hierarchieebene vorhanden sein müssen, steht PSOS für eine Familie von Betriebssystemen und ist als Capability-Konzept zu verstehen. Speziell angepasste Systeme können die Hierarchieebenen unterschiedliche realisieren, lediglich die unterste Ebene mit den Capability-Mechanismen ist festgelegt.

Wie beispielsweise in [NF03] besprochen wird, sind die eigentlichen Sicherheitsmaßnahmen, die auf dem Capability-Konzept aufbauen, von PSOS nicht festgelegt. Diese werden auf höherer Ebene realisiert und können verschiedenste Policies durchsetzen. Die beiden Funktionen zur Erzeugung der Capabilities können von priviligierten Systemfunktionen genauso wie von einfachen Anwendungen genutzt werden.

Eigenschaft	Ausprägung
Spezielles Sicherheitsmodell	allg. Subjekte-/Objektemodell
Security Kernel	Privilegstufen
Speicherschutzkonzept (Prozessschutz)	nicht festgelegt
Schwachstellen-/Bedrohungsanalyse	keine
Entwicklungsvorgehen für sichere Systeme	HDM
Kryptografische Verfahren	keine

Tabelle 3.2: Klassifikation von PSOS

KSOS

Das Betriebssystem KSOS (Kernelized Secure Operating System) wurde in den 70er Jahren für das amerikanische Verteidigungsministerium DARPA entwickelt. Neben den Sicherheitsanforderungen, die im Wesentlichen aus Multilevel-Security in Form des Bell-LaPadula-Modells bestand, war die Kompatibilität zu dem UNIX-Derivat der Firma Western Electric Company, welches kompatibel zu UNIX von Bell Telephone Laboratories war. KSOS wurde für eine Hardwarearchitektur vom Typ DEC PDP-11/70 entwickelt.

Die Hardware der PDP-11 stellt für KSOS einen dreistufigen Schutzmechanismus zur Verfügung, wie auch in [Ano78] beschrieben wird. In der priviligiertesten Schicht, dem Kernel-Modus, wird nur der Kern `KSOS.K` mit den wichtigsten Sicherheitsmechanismen ausgeführt. Im weniger priviligierten Supervisor-Modus wird ein UNIX-Emulator `KSOS.E` ausgeführt, der die Nutzung von standard UNIX-Anwendungen erlaubt. Daneben wird in dieser Schicht der Teil der Sicherheitsmechanismen für das Durchsetzen der Multilevel-Security realisiert, der vertrauenswürdig sein muss. In der unpriviligierten Ebene wird neben den Anwendungen auch der Teil der Sicherheitsmechanismen ausgeführt, der nicht priviligierte Sicherheitsdienste für die Anwendungen zur Verfügung stellt, wie etwa das Drucker-Spooling oder ein Dienst für abgesicherte Mail.

Um die Sicherheitsmechanismen für Multilevel-Security effektiv durchzusetzen, wird in KSOS das Konzept eines Referenzmonitors umgesetzt. Dieser kennt zur Laufzeit stets alle Objektreferenzen aller Subjekte, seine Regelmechanismen und Regeldaten müssen vor unauthorisierten Zugriffen geschützt sein. Der Mechanismus, der dieses Konzept verwirklicht, ist der Security-Kernel, der sowohl die Hardware, als auch die Software zur Realisierung der Sicherheitsanforderungen beinhaltet.

In [Sil83] wird diskutiert, welchen Erfolg man durch die formale Entwicklungsmethode HDM bei der Realisierung von KSOS-6, einer speziellen Version von KSOS, erzielt wurden. Mit den sehr aufwändingen Mitteln der formalen Verifikation innerhalb der HDM-Methode wurden 33 Sicherheitsprobleme gefunden. Davon wurden 29 auch durch einfachere Code-Analyse oder anderer Verfahren aufgespürt. Für den Aufwand des formalen Vorgehens existieren ungefähre Zahlen. Zur Entwicklung und Validierung der 3300 Zeilen SPECIAL-Code, die KSOS-6 darstellten, wurde etwa ein Mannjahr benötigt, als etwa 10 Zeilen pro Tag. Die Umsetzung des Kerns benötigte etwa 9 Mannjahre und bestand am Ende aus etwa 10000 Zeilen UCLA Pascal Quelltext. Obwohl das Ergebnis der aufwändigen Entwicklung als technischer Erfolg für den Einsatz von formalen Methoden angesehen wird, ist das Ergebnis im Verhältnis zum Aufwand der Verifikation und Entwicklung der Tools nach [Sil83] relativ gering. Auf die manuelle Code-Analyse kann nicht verzichtet werden, die auch für eine korrekte Spezifikation durchgeführt werden muss.

Eigenschaft	Ausprägung
Spezielles Sicherheitsmodell	basiert auf Bell-LaPadula
Security Kernel	Referenzmonitorkonzept
Speicherschutzkonzept (Prozessschutz)	dreistufiger Speicherschutz
Schwachstellen-/Bedrohungsanalyse	keine
Entwicklungsvorgehen für sichere Systeme	HDM
Kryptografische Verfahren	keine

Tabelle 3.3: Klassifikation von KSOS

CAP

Die Architektur von `Capability Architecture` (CAP) besteht aus einer Logik, die in Hardware realisiert ist und einem Softwareteil, in Form eines Mikroprogramms. Hauptziel der Architektur ist ein feingranularer Speicherschutz, wie er auch in [NW77] beschrieben ist. Der Speicher wird in Segmente eingeteilt, die jeweils durch eine Basisadresse und ein Limit festgelegt sind. Der Zugriff auf ein Speichersegment wird durch Capabilities kontrolliert. Diese bestehen aus fünf Bits, drei für den Datenzugriff (Read, Write und Execute) und zwei für den Zugriff auf die Capabilities selbst (Read und Write Capability). Capabilities werden in 16 Segmenten gespeichert, die bei der Adressierung von Daten durch Prozesse neben der Position der Daten und der Position der entsprechenden Capability angegeben werden.

Zu jedem Prozess existiert eine Prozess-Ressource-Liste (PRL), die jederzeit die Anzahl aller Ressourcen angibt, die von einem Prozess unabhängig des Ausführungspfades verwendet werden können. Die Capabilities aus den Segmenten besitzen jeweils einen Zeiger auf einen Eintrag in der PRL. Capabilities werden als Argumente beim Aufruf von geschützten Funktionen innerhalb eines Prozesskontextes übergeben. Dadurch können die Rechte und die Ressourcen für jede Funktion eingeschränkt werden. Eine Übergabe von Capabilities zwischen verschiedenen Prozesskontexten ist nicht möglich. Die Verwaltung der Capabilities wird durch die Ressourcen und Prozessverwaltung konsequent durchgeführt.

Die strenge Prozesshierarchie unterscheidet zwischen einem Master-Koordinator in Ebene 1, der für die Verwaltung der Prozesse in Ebene 2 direkt verantwortlich ist. Prozesse in Ebene zwei liegen in Datensegmenten der übergeordneten Ebene und sind streng voneinander getrennt. Diese Prozesse können wiederum neue Prozesse in Ebene 3 erzeugen, für deren Verwaltung sie zuständig sind. Bei der Erzeugung und dem Aufruf von Kindprozessen werden Capabilities und ein Zeiger auf eine Menge von Ressourcen übergeben, die der Prozess dann nutzen kann. Der Eintrag in der RPL des Master-Koordinators besitzt phy-

sikalische Adresseinträge für die Verwaltung seiner Segmente. Jeder Kindprozess besitzt wiederum Einträge in der RPL, die jedoch als Adressen relativ zu dem Adressbereich des Vaters interpretiert werden. Dadurch kann ein erzeugter Prozess niemals mehr Adressbereiche verwenden, wie vom Vater benutzt werden. Für die Interpretation der Adressen ist das Mikroprogramm zuständig. Für die Realisierung der Capability-Verwaltung beim Einsprung in eine Funktion und beim Aussprung ist der Programmierer der Anwendung selbst verantwortlich. Werden diese Mechanismen nicht korrekt realisiert, können Rechte an nicht authorisierte Prozesse ungewollt weitergegeben werden.

Neben einer durchdachten Capability-Verwaltung und einem hierarchischen Speicher- und Prozessverwaltungssystem existieren keine weiteren Angaben zu möglichen Angriffen oder Bedrohungen auf das System. Es wurden keine speziellen Angriffe betrachtet, die die Architektur von CAP begründen. Vielmehr stand die Entwicklung eines Capability-orientierten Speicherschutzsystems im Vordergrund. Diese wird in einigen Projekten verfeinert, wie etwa unter SCAP (Secure Capability Architecture), bei dem ein zusätzlicher ACL-Mechanismus eingesetzt wurde.

Eigenschaft	Ausprägung
Spezielles Sicherheitsmodell	keines
Security Kernel	Hardware und Mikroprogramm
Speicherschutzkonzept (Prozessschutz)	Capability-basiert
Schwachstellen-/Bedrohungsanalyse	keine
Entwicklungsvorgehen für sichere Systeme	keines
Kryptografische Verfahren	keine

Tabelle 3.4: Klassifikation von CAP

KeyKOS

KeyKOS, was etwa in [Har85] näher beschrieben wird, ist direkter Nachfolger von GNOSIS ([Fra79]). KeyKOS tellt einen Capability-orientierten Ansatz dar. Basis ist ein Mikrokern, dessen Aufgaben exakt festgelegt sind. Neben einem einfachen Scheduler unterstützt er die Verwaltung der wesentlichen Objektprimitiven, wie die Capabilities, die an die Prozesse und deren Ausführungsdomänen gekoppelt sind. Zudem werden Nachrichten vom Mikrokern verwaltet, die die einzige Kommunikationsmöglichkeit zwischen Prozessen und Ausführungsdomänen darstellen. Die Multiprogrammierung wird durch die Möglichkeit unterstützt, dass eigene Scheduler für eine Menge von Domänen installiert werden können, die durch den einzigen Scheduler im Mikrokern gesteuert werden. Der Kern stellt Speicher für die Domänen zur Verfügung, wobei diese prinzipiell keinen Unterschied zwischen

Hauptspeicher und Sekundärspeicher (Festplattenspeicher) unterscheiden können. Wichtige Daten im Speicher werden redundant abgelegt. Im Mikrokern werden die Segmente für die einzelnen Domänen innerhalb des virtuellen Speichers festgelegt, nur er kann Speicherzuordnungen verändern. Der Kerncode besitzt kein eigenes Segment, sondern arbeitet stets auf physikalischen Adresse ohne Umrechnung. Aus Sicherheitsgründen ist es deshalb nicht möglich, den Kern durch eigenen Programmcode zu erweitern. Sowohl der Zugriff auf Geräte durch Ein- und Ausgabeports, als auch der Nachrichtenaustausch zwischen Domänen werden durch sogenannte Gate Keys kontrolliert. Diese sind wie Capabilities den Subjekten zugeordnet und werden ausschließlich vom Kern verwaltet. Eine Domänenübergreifende Sicherheits-Policy bestimmt, wer welche Capabilities besitzen darf. Der Mikrokern erzeugt in periodischen Abständen systemweite Checkpoints bei der Ausführung, die bei auftretenden Fehlern den letzten gültigen Arbeitspunkt bestimmen. Fehler können sowohl Ausfälle, als auch Sicherheitsverletzungen sein. Beim Zurücksetzen auf einen vergangenen Checkpoint werden Prozesse beendet, die nicht exakt den Zustand rekonstruieren lassen. Grundprinzip des KeyKOS-Konzeptes ist das Least Privilege Prinzip, das durch die strikte Verwendung von Capabilities durchgesetzt wird. Als modernes System werden in KeyKOS alle Subjekte als Objekte im objektorientierten Sinn verstanden. Durch die Kapselung kann leicht festgestellt werden, welche Nachrichten an andere Domänen bzw. Prozesse kontrolliert und vermittelt werden müssen. Eine Klassifikation nach dem für diese Arbeit entwickelten Schema findet sich in Tabelle 3.5.

Eine moderner Ableger von KeyKOS stellt das Betriebssystem EROS dar, das ausschließlich für Intel-Architekturen entwickelt wird. In EROS werden einige Betriebssystemkonzepte von KeyKOS verbessert, die Grundprinzipien und die Sicherheitsarchitektur bleiben jedoch im Wesentlichen gleich. Nähere Informationen zu EROS findet man in [SSF99].

Eigenschaft	Ausprägung
Spezielles Sicherheitsmodell	keines
Security Kernel	Mikrokernarchitektur
Speicherschutzkonzept	Capability-basiert
Schwachstellen-/Bedrohungsanalyse	keine
Entwicklungsvorgehen für sichere Systeme	keines
Kryptografische Verfahren	keine

Tabelle 3.5: Klassifikation von KeyKOS

L4

L4 beschreibt Konzepte für einen modernen Mikrokern. Die Vorgänger wie L3 und die bereits in der Entwicklung befindlichen neuen Versionen werden hier nicht gesondert betrachtet, da die Unterschiede für den Sicherheitsteil gering sind. Der Fokus bei der Entwicklung liegt besonders auf hoher Performanz des Kerns und der zugehörigen Mikrokernserver. Grundprinzip der Konzepte ist die Minimalität. Nur die wesentlichen Mechanismen werden im μ-Kern realisiert. Er verwaltet die sogenannten Kernobjekte. Einerseits sind das die Threads als Aktivitäten, die in den Prozessadressräumen ablaufen. Andererseits sind das die virtuellen Seiten, die in ihrer Größe nicht festgelegt sind und deshalb Flexpages genannt werden. Ein virtueller Adressraum einer Aktivität wird durch eine endliche Menge dieser Flexpages beschrieben. Diese Seiten sind auch die Basis der Interprozesskommunikation, bei der einzelne Seiten in andere Prozessadressräume eingeblendet und weitergereicht werden können. Für die Verwaltung der Adressräume existieren zwei Sicherheitsdomänen. Tasks bestehen aus den ihnen zugeordneten Threads und einem virtuellen Adressraum. Nur die Threads sind in der Lage, den Adressraum zu verwalten. Die andere Sicherheitsdomäne wird durch ein besonderes IPC-Konzept verwirklicht, das Clans und Chiefs Konzept genannt wird. Ein Clan stellt dabei eine Menge von Threads dar, der einen Adressraum zur Verfügung hat. Jedem Clan ist genau ein Chief voranstehend, der die Kommunikation über die Clan-Grenze hinweg regelt. Nur der Chief ist in der Lage, Kommunikation über die Clangrenze hinaus zu führen. Clanmitglieder (Threads) können beliebig miteinander Daten austauschen. Clans dürfen auch ineinander geschachtelt sein, die Kummunikation über die Clangrenze wird aber stets über den Chief durchgeführt. Es bleibt dem jeweiligen Einsatzgebiet überlassen, wie die Zuordnung zwischen Clans und den vorhandenen Subjekten durchgeführt wird. Für gängige Einsatzgebiete würde es sich beispielsweise anbieten, pro Prozess einen Clan zu verwalten. Damit würde ein gegenseitiger Schutz der Prozesse trotz sehr hoher IPC-Geschwindigkeit realisiert werden. Vor allem auf die Anforderungen der Tasks genau zugeschnittene Sicherheitspolitiken lassen sich so leicht im Benutzeradressraum realisieren. Kontrolliert werden können alle Nachrichten, die an die im Clan enthaltenen Threads gesendet oder von diesen empfangen werden. Die Kontrolle selbst kann durch beliebige Algorithmen durchgeführt werden. Grundlagen des Clans and Chief Konzepts sind beispielsweise in [Lie93] nachzulesen. Aufgrund der gewünschten hohen Performanz wird Prozesskommunikation synchron durchgeführt. Es existieren keine Kanäle oder Ports dafür, Threads kommunizieren direkt miteinander. Der sendende Thread wird deshalb solange blockiert, bis der Empfänger die Nachricht angenommen hat. Es existieren einige Realisierungen von L4, die auf unterschiedlicher Hardware ausführbar sind. Diese demonstrieren die hohe Geschwindigkeit der Interprozesskommunikation und der Realisierung verschiedener Server. Nähere Informationen zu L4 gibt [HHLS97], die Klassifizierung der Sicherheitseigenschaften sind in Tabelle 3.6 zu sehen.

Eigenschaft	Ausprägung
Spezielles Sicherheitsmodell	keines
Security Kernel	Mikrokernarchitektur
Speicherschutzkonzept	Domänenbildung, absicherbare IPC, Clans und Chiefs
Schwachstellen-/Bedrohungsanalyse	keine
Entwicklungsvorgehen für sichere Systeme	keines
Kryptografische Verfahren	keine

Tabelle 3.6: Klassifikation des L3/L4 μ-Kerns

Multics

Multics (Multiplexed Information and Computing Service) nach [BL75] gilt in vielen funktionalen Bereichen der Betriebssysteme als der Vorreiter. Die Entwicklung, die 1965 begonnen wurde, hat schließlich zur Entwicklung von Unix (Uniplexed Information and Computing Service) geführt. Sicherheitsaspekte von Unix werden auch in [GM84] und [MR92] diskutiert, die Verifikation eines Unix Kern in [WKP80] besprochen. Die Speicherverwaltung wird sowohl durch Segmentierung, als auch durch virtuellen Speicher realisiert. Die Segmente sind 1 MByte groß und bilden auch das erste hierarchische Dateisystem im Speicher ab, Dateien und Verzeichnisse werden durch Speicherreferenzen benutzt. Der virtuelle Speicher nach [BCD72] und [Sal74] wird durch eine Hardwarekomponente realisiert, die die Umrechnung von virtuellen in physikalische Adressen umsetzt, prinzipiell einen (wenn auch einfachen) Vorläufer der Memory Management Units (MMUs) heutiger Systeme. Die Verwaltung mehrerer Prozessoren, die alle gleichwertig sind, werden durch Shared Memory unterstützt. Als eines der ersten Betriebssysteme wurde Multics in einer Hochsprache entwickelt, in PL/I. Neben dieser Sprache werden auf Middleware und Anwendungsebene auch andere Sprachen unterstützt und eingesetzt, wie etwa Fortran, Lisp, C, Cobol oder Basic. Funktionen die in einer Sprache realisiert sind, können Funktionen einer anderen Sprache aufrufen, was wegen der damals neu entwickelten Technik der dynamischen Bindung ermöglicht wird. Diese erwähnten Konzepte und das systematische Entwickeln, was ein großer Beitrag zum Software-Engineering darstellt, sind heute noch prägend für die eingesetzten Konzepte in modernen Betriebssystemen. Die Sicherheit bei der Entwicklung wird sowohl durch die Systematik, als auch dadurch erhöht, dass man eine Hochsprache eingesetzt hat, die beispielsweise Schwachstellen durch Pufferüberläufe auf dem Keller vermeidet. Die Durchsetzung der Prozesshierarchie ermöglicht die Weitergabe von Berechtigungen zwischen Eltern- und Kindprozessen. Die Hierarchie im Dateisystem ermöglicht die Konsequente Rechteverwaltung und für die Benutzer eine überschaubare Möglichkeit,

eigenen Daten zu schützen. Der Multics-Speicherbereich wird aus 8 Schutzzonen realisiert, die wegen der Darstellung in Form von ineinander geschachtelten Kreisen Ringe genannt werden. Die Ring 0 Zone, die die prvilegierteste Zone darstellt, führt Code im privilegiertem (Master Mode) und nicht privilegiertem (Slave Mode) Modus aus. Bestimmte, sehr hardwarenahe und kritische Befehle können so nur im privilegiertem Ring 0 Bereich ausgeführt werden. Ring 1-3 ist für weniger privilegierten Systemcode vorgesehen, gängige Anwendungen werden im Ring 4 ausgeführt. Ring 5 ist für Anwendungen vorgesehen, die speziell überwacht werden können und Kommunikation mit anderen Rechnern durchführen. In Ring 6 und 7 können Anwendungen ablaufen, die keine Systemaufrufe durchführen dürfen, sie können lediglich Anwendungen in Ring 4 und 5 aufrufen. Eine grundsätzliche Diskussion über Konzepte für Schutzzonen findet sich in [SS72].

Eigenschaft	Ausprägung
Spezielles Sicherheitsmodell	erstmalig virtueller Speicher
Security Kernel	Sicherheitskern
Speicherschutzkonzept	8 Schutzringe
Schwachstellen-/Bedrohungsanalyse	keine
Entwicklungsvorgehen für sichere Systeme	erstmalig Hochsprache
Kryptografische Verfahren	keine

Tabelle 3.7: Klassifikation von Multics

BirliX

BirliX ist ein verteiltes Betriebssystem mit einer dreischichtigen Sicherheitsarchitektur. Die Hardware wird beim Hochfahren des Systems von einem kryptografisch abgesicherten Bootvorgang auf Vertrauenswürdigkeit geprüft. Der Sicherheitskern stellt die Verwaltung und die Kommunikation zwischen den Objekten der höheren Schicht zur Verfügung. Darauf setzen die Politiken in Form von Objekten auf, die prinzipiell frei beschreibbar sind. BirliX kennt folgende fünf Formen von Subjekten.

1. Benutzer, eindeutig identifizierbar
2. Gruppe, eindeutig identifizierbar und beinhaltet eine Menge von Benutzern
3. Verwalter zur Durchsetzung der Regeln
4. Objekt

5. Typ, abstrakter Datentyp als verallgemeinerte Form von Objekten

Die Zugriffskontrollmechanismen bestehen aus einer detailierten Beschreibung der Rechte zwischen den Subjekten und den Objekten. Die Rechte selbst können vom eingesetzten Verwalter bestimmt werden und sind nicht fest vorgegeben. Neben den ACLs werden Subjektrestriktionslisten eingeführt, die für ein Subjekt die Rechte auf bestimmte Objekte einschränken. Dies ist zusätzlich zu den Zugriffskontrolllisten notwendig, weil in verteilten Umgebungen die Konsistenz und Gültigkeit der einzelnen an die Objekte gebundenen ACLs nicht garantiert werden kann, trotzdem Subjekt an Zugriffen gehondert werden müssen.

In BirliX wird die Kommunikation zwischen den einzelnen Subjekten und Objekten über RPC (Remote Procedure Call) realisiert. Grundlage für die Realisierung der Objekte sind abstrakte Datentypen. Diese beschreiben Gemeinsamkeiten konkreter Instanzen, wie beispielsweise Dateien, Prozesse oder Synchronisationsobjekte. Zu jeder Instanz existiert stets ein Prozess zur Verwaltung, wie die Ausführung der Methoden auf diese Objekte.

Durch die Verwendung sowohl von Zugriffskontrolllisten, als auch daneben von Subjektrestriktionslisten können die Rechte zwischen Subjekten und Objekten sehr flexibel ausgedrückt werden. Die Kontrolle der Rechte geschieht jeweils beim Zugriff durch erweiterbare Verwalterobjekte, wodurch eine Vielzahl an Rechteprüfungen möglich wird. Im Vordergrund der BirliX-Sicherheitsarchitektur steht also eine flexible Rechtekontrolle auf Basis von definierten Systemeinheiten. Besonderen Aufwand bei der Entwicklung, bestehende Bedrohungen in die Entwicklung und das Design einzubeziehen, sind nicht bekannt.

Eigenschaft	Ausprägung
Spezielles Sicherheitsmodell	Subjektrestriktionslisten
Security Kernel	keiner
Speicherschutzkonzept	Objektschutz
Schwachstellen-/Bedrohungsanalyse	keine
Entwicklungsvorgehen für sichere Systeme	
Kryptografische Verfahren	Integritätsprüfung beim Start

Tabelle 3.8: Klassifikation von BirliX

3.1.4 Fazit

Die vorgestellten Betriebssysteme stellen den Teil der bekannten Systeme dar, der bestimmte Sicherheitskonzepte eingeführt und realisiert hat. Es existieren viele weitere Betriebssysteme für stationären Einsatz, die zwar ebenso bekannt und verbreitet sind, allerdings

keine neuen Konzepte für die Sicherheit des Gesamtsystems zur Verfügung stellen. Daher bleibt als Fazit der Zusammenstellung, dass sowohl für die Entwicklung abgesicherter Systeme, als auch präventive Maßnahmen für deren Betrieb Mechanismen und Vorgehensweisen existieren. Diese sind allerdings nur getrennt voneinander zu sehen, wie sie auch in den einzelnen Projekten nicht integriert sind. Die naheliegende Lösung, alle Sicherheitskonzepte in ein System zu integrieren führt aber nicht zwangsweise zum Ziel, da viele Maßnahmen gegensätzlich zueinander stehen und teilweise auch nicht nebeneinander eingesetzt werden können. Deshalb ist es wichtig, die vorgestellten Konzepte, Vorgehensweisen und Mechanismen zu kennen und durch Integration in ein System zusammenzuführen.

3.2 Mobile verteilte Systeme

Neben den funktionalen Eigenschaften werden für Betriebssysteme vor allem in speziellen Einsatzgebieten auch nicht-funktionale Eigenschaften immer wichtiger. Zwei dieser Aspekte, die zusätzliche Qualitäten des Systems beschreiben, sind Mobilität und Sicherheit. Sie werden in den folgenden beiden Abschnitten mit den Betriebssystemen in Zusammenhang gestellt. Der Begriff der mobilen verteilten Systeme ist heute als solcher noch nicht präzise und eindeutig definiert. Mit verteilten Systemen sind im Allgemeinen jedoch Rechensysteme gemeint, die räumlich verteilt sind und über ein Netzwerk miteinander kommunizieren können. Der Begriff der Mobilität bedeutet zunächst, dass sich irgendetwas dieses Systems bewegen kann.

Definition 3.5 (Mobilität)
Die Mobilität *eines Systems bedeutet allgemein, dass sich eine Systemkomponente über die Rechnergrenzen hinweg bewegt.*

□

Die Allgemeinheit dieser Definition ist für den Begriff der Mobilität notwendig, da ohne Kenntnis des Kontextes oder Einsatzgebietes des Systems keine genauere Aussage getroffen werden kann, was sich bewegt. Oft wird der Begriff als Synonym für eine dynamische Systemstruktur verstanden, bei der sich die Kommunikationsbeziehungen abwechseln und verändern. Für viele der heutigen Systeme im mobilen Umfeld trifft diese Sicht zu. Allerdings sind heute beispielsweise schon Architekturen in Diskussion, die sich mit mobilen Diensten beschäftigen. Dabei steht nicht mehr die Kommunikationsbeziehung zwischen den Systemen im Vordergrund, die sich ortsabhängig verändert, sondern veränderbare Dienste des Systems. Prinzipiell können aber auch alle anderen Elemente eines mobilen verteilten Systems mobil sein.

Dienste für die Kommunikation und die Verteiltheit werden innerhalb eines Rechensystems in der Schichtendarstellung möglichst weit unten angesiedelt, wie auch in [LABW92] und [RR83] nachgelesen werden kann, worin Mechanismen zur Authentifizierung in ver-

teilten Systemen umgesetzt werden. Dadurch können dieselben Dienste sowohl vom Betriebssystem selbst, als auch von den Anwendungen genutzt werden. Somit werden sowohl die Kommunikation, als auch das Management der Mobilität allgemein als Dienste des Betriebssystems betrachtet. Kenntnisse über mobile verteilte Systeme setzen damit auch Kenntnisse über Betriebssysteme voraus.

Die Entwicklung hin zu den mobilen verteilten Systemen muss als Folge der gesamten IT-Entwicklung gesehen werden, wobei mehrere parallel verlaufende Entwicklungen als Gründe zu sehen sind.

Ein Grund ist die zunehmende Vernetzung. Zunächst wurden einzelne, stationäre und nicht vernetzte Systeme genutzt, bei denen der Datenaustausch durch den Transport von Datenspeicher durchgeführt wurde. Durch die zunehmende Vernetzung wurde die Kommunikation komfortabler und schneller. Es wurden Systeme entwickelt, deren Hauptaufgabe die Netzwerk-Kommunikation war. Ein anderer Grund ist die Miniaturisierung der Systeme, bei gleichzeitiger wachsender Funktionalität. Beide Ursachen zusammengenommen haben bewirkt, dass kleine, mobile Geräte eingesetzt werden, die eine große Funktionalität mit ausgeprägten Kommunikationseigenschaften besitzen. Diese werden heute in unterschiedlichsten Formaten und Ausprägungen im täglichen Leben eingesetzt, beispielsweise in Form von PDAs oder Mobiltelefonen.

Im Gegensatz zu den stationären Rechensystemen, bei denen der Ort des gesamten Systems statisch ist, ist die Eigenschaft der Mobilität der Systeme als starke erweiternde Eigenschaft zu verstehen. Weil bei mobilen Systemen oft an kleine, handliche PDAs gedacht wird, werden sie oft als eingeschränkt angesehen. Tatsächlich aber bietet diese Geräteklasse der mobilen Geräte eine zusätzliche Eigenschaft, die die folgende Generation auf die der statisch vernetzten Systeme beschreibt. Im Sicherheitskontext bedeutet dies, dass die Systemumgebung nicht als bekannt und vertrauenswürdig vorausgesetzt werden darf. Ebenso wie die Prozesse innerhalb, ändert sich auch die Umgebung dynamisch. Zudem können sich die Vertrauensverhältnisse zwischen Kommunikationspartnern ändern.

3.2.1 Vergleich zu klassischen Betriebssystemen

Um die Eigenschaft der Mobilität im Zusammenhang mit den Dimensionen Raum und Zeit zu beschreiben, wird folgendes gedankliche Modell entwickelt. Ein System besteht aus Komponenten. Diese werden jeweils durch ein Attribut erweitert, das ihren Ort beschreibt und Raumattribut genannt wird. In klassischen Systemen wird sich dieses Attribut im Laufe der Ausführung in keiner Komponente ändern. Die Ausführungsorte, beziehungsweise die Raumattribute sind hier im Allgemeinen über die Laufzeit hinweg konstant, Ausnahmen sind dabei im Wesentlichen administrative Eingriffe oder Systemausfälle. In mobilen Systemen, wobei hiermit jede erdenkliche Art der Mobilität angesprochen sei, ist das Raumattribut abhängig von der Ausführungszeit. Dadurch sind mobile Systeme eindeutig charakterisiert und von den klassischen Systemen abgrenzbar.

IT-Sicherheit bezüglich mobiler verteilter Systeme

Der Aspekt der IT-Sicherheit für die Betriebssysteme in mobilen verteilten Systemen beinhaltet zunächst alle Erkenntnisse, die schon im vorherigen Abschnitt über die klassischen Systeme diskutiert wurden. Zudem lassen sich natürlich die erwähnten Betriebssysteme, zumindest theoretisch, auf mobilen Endgeräten nutzen und deren Absicherungen realisieren. Sicherheit in verteilten Systemen wird unter anderem in [GGKL89] modelliert. Für mobile Systeme, die eine Erweiterung der klassischen Systeme darstellen, genügen diese Architekturen jedoch im Allgemeinen nicht, wie beispielsweise in [Eck03] diskutiert wird. Der Aspekt der Mobilität muss im Zusammenhang mit der Realisierung einer integrierten Sicherheitsarchitektur speziell behandelt werden. Dafür sprechen mehrere Gründe, wie auch in [BEG00] besprochen wird.

Das Bedrohungsmodell, das allgemein zwischen internen und externen Angreifern unterscheidet, muss für den Mobilitätsaspekt angepasst werden. Angreifer klassicher Systeme waren entweder intern, also innerhalb des betrachteten Systems angesiedelt, oder extern, außerhalb der Sichtbarkeit des Systems. Dadurch, dass sich in mobilen Systemen der Ort einzelner Komponenten während der Laufzeit verändert, kann sich ebenso der Ort des Angreifers relativ zum System ändern. Umgekehrt kann das mobile System selbst den Angreifer darstellen und durch seine Ortsveränderungen sowohl interner Angreifer, als auch ein externer sein. Daneben sind die Möglichkeiten von passiven Angriffen gesteigert, Informationen aus der Beobachtung des Ortes und Ortsinformationen des Systems und des Benutzers zu gewinnen.

Das Vertrauensverhältnis, das sich zwischen zwei Kommunikationspartnern, beispielsweise einem stationären und einem mobilen System ergibt, hängt stark vom Kontext der Systeme ab. Zu diesem Kontext gehören auch die Aktivitäten in der Vergangenheit, als auch die der Zukunft. Ein Beispiel hierfür ist die Kommunikation zwischen einem System einer Firma und einem mobilen Endgerät innerhalb des Firmennetzes. Das Vertrauensverhältnis muss sich ändern, wenn bekannt ist, dass sich das mobile Endgerät ebenso in einem anderen Firmennetz eines Konkurrenten der Firma bewegt und dort kommuniziert. Dabei sei auf das Chinese Wall Modell aus dem vorigen Kapitel verwiesen, das für solche Szenarien entwickelt wurde. An diesem konkreten Beispiel lässt sich sehen, dass Mobilität neue Ansprüche an die Sicherheitspolitiken stellt und dass die bisherigen statischen Sicherheitsmechanismen nicht ausreichen.

Die Erweiterbarkeit der mobilen Endgeräte, beispielsweise durch das Hinzufügen von speziellen ortsbezogenen Protokollen oder Diensten, muss abgesichert durchgeführt werden können. Dies betrifft einerseits die Sicherheit der Umgebung, die durch einen Angriff auf die Dienste kompromitiert werden kann und andererseits die Sicherheit des mobilen Systems, das durch das Ausführen mobilen Codes kompromitiert werden kann. Beispiel für die abgesicherte Ausführung von Code kann man in [MRR98] finden.

3.2.2 Beispiele moderner Varianten

Die Ansprüche an Betriebssysteme für mobile Endgeräte sind hoch und die Hardwareleistung, die zur Verfügung steht, ist meistens stark eingeschränkt. Dies betrifft sowohl die Prozessorleistung, den zur Verfügung stehenden Speicherplatz, als auch die zur Verfügung stehende Peripherie, wie etwa persistenter Speicher und Ausbaufähigkeiten. Für die Rechen- und Speicherfähigkeiten dieser Geräte gilt damit, dass sie gegenüber denen der aktuellen stationären Gerätegeneration einige Jahre zurückliegen. Eine Ausnahme ist die Kommunikationsfähigkeit, die stets die neuesten Kommunikationstechnologien beinhaltet. Wesentlich erweitert sind meistens die Energieverwaltungsfunktionen der mobilen Endgeräte, die aufgrund der begrenzten zur Verfügung stehenden Energie nötig sind.

Da die Sicherheitsanforderungen hoch sind, die möglichen Angriffe auf und durch mobile Systeme aber ebenfalls hoch sind, ist zu erwarten, dass moderne Betriebssysteme für mobile Endgeräte auf die wesentlichen Bedrohungen dieser Geräteklasse eingehen und besondere Sicherheitsmechanismen verwenden und zur Verfügung stellen. Wie gut mobile Endgeräte als vertrauenswürdige persönliche Begleiter eingesetzt werden können, wird in [BGL03b] diskutiert. Im Folgenden werden deshalb die bekannten Betriebssysteme analysiert und deren Sicherheitsmechanismen beschrieben.

Windows CE, Pocket PC

Die Windows CE Familie ist für alle Arten von Geräten vorgesehen, die sowohl persönliches mobiles Gerät eingesetzt werden, als auch für den Embedded-Bereich, beispielsweise in Fahrzeugen. CE stellt dabei eine anpassbare Produktfamilie dar, die durch die entsprechenden Hilfsprogramme aus einer Menge von Bibliotheken sowohl einen Betriebssystemkern, als auch die notwendigen Betriebssysteme- und Middleware-Bibliotheken zusammenbinden. Dadurch sind viele Hardwareplattformen und Peripheriegeräte durch CE unterstützt oder können leicht hinzugefügt werden. Wesentliche Eigenschaft der CE Familie ist es, dass sich die Schnittstelle zu den Anwendungen hin derart präsentiert, dass eine möglichst leichte Portierung von Win32 Anwendungen auf Windows CE durchgeführt werden kann. Sowohl Teile der Win32 Schnittstelle, als auch ein Teil der .NET Architektur wurden nach CE portiert.

Der monolithische Kern hat die Aufgaben der Prozess- und Threadverwaltung, des Speichermanagements, des Dateisystems, der grafischen Ein- und Ausgabe mit der Ereignisverwaltung und der Verwaltung der Kommunikationsmechanismen. Da Programme normalerweise im Read-only Speicher abgelegt sind, können sie sowohl zur Ausführung in den Arbeitsspeicher kopiert werden oder wahlweise auch vom Nurlesespeicher aus ausgeführt werden. Das Scheduling der maximal 32 Prozesse im System ist ein prioritätenbasiertes Round-Robin-Verfahren mit 25 ms Zeitquantum, die Prioritäten liegen je nach Version entweder zwischen 0 und 7 oder zwischen 0 und 255. Das Speichermodell legt feste Speicherbereiche für jeden Prozess, die gemeinsam benutzten Bibliotheken und den Kern

fest, die Slots genannt werden. Jeder Slot hat eine feste maximale Größe von 64 MByte zur Verfügung, Prozesse sind damit auf diese Größe begrenzt. Slot 0, der an den Beginn des virtuellen Speicherraums gelegt wird, beinhaltet stets den aktuellen, aktiven Prozess. Die Umschaltung zwischen den Prozessen beinhaltet demnach auch ein Einblenden des aktuellen Prozesses in den Slot 0. Slot 1-4 sind reserviert für die dynamischen Bibliotheken, den Kern, das Dateisystem und die grafische Benutzungsbibliothek. Die Abarbeitung der Unterbrechungen geschieht zweistufig durch eine Interrupt Service Routine, die die wichtigsten Arbeiten erledigt und durch die anschließende Arbeit des Interrupt Service Threads. Die Unterbrechungsbehandlung lässt keine verschachtelten Unterbrechungen zu.

Die Sicherheitsarchitektur in Windows CE .NET, der aktuellsten Windows CE Version, besteht aus einer Crypto API zum eleganten Umgang mit symmetrischen und asymmetrischen Kryptoverfahren, der Schnittstelle zur Verwendung von SmartCards und verschiedenen Sicherheitsprotokollen zur Kommunikation, wie etwa SSL. Es existiert ein sicherer Schutz der einzelnen Prozessadressräume bzw. Slots gegeneinander, der durch die virtuelle Speicherverwaltung realisiert wird. Teile des Betriebssystems, insbesondere das Kernabbild wird mit einer Signatur versehen und die Rechtmäßigkeit von Veränderungen kann überprüft werden. Die bekannten Sicherheitsmechanismen liefern allerdings keinerlei speziell auf mobile Systeme zugeschnittenen Verfahren, wie der Schutz vor fremdem auszuführenden Code oder wechselnder Kommunikations- und Vertrauensumgebungen. Bekannte Bedrohungen mobiler Geräte, wie etwa der Diebstahl des Gerätes und damit der mögliche Verlust der gespeicherten Daten werden auch nicht speziell berücksichtigt. Eine Einordnung von Windows CE in moderner Form ist in 3.9 zu sehen.

Eigenschaft	Ausprägung
Spezielles Sicherheitsmodell	keines
Security Kernel	keiner
Speicherschutzkonzept	virtueller Speicher, Slots größenbeschränkt
Schwachstellen-/Bedrohungsanalyse	keine
Entwicklungsvorgehen für sichere Systeme	keines
Kryptografische Verfahren	Kryptoprovider, erweiterbar

Tabelle 3.9: Klassifikation von Windows CE

Palm OS

Das Betriebssystem Palm OS muss für diese Diskussion in zwei Versionen betrachtet werden. Zunächst wurde Palm OS für eine Hardwarearchitektur entwickelt, eine spezielle low-

cost Version des Motorola 68k, der über keine MMU und damit auch nicht über die Möglichkeit der virtuellen Speicherverwaltung verfügt. Das Betriebssystem und die Anwendungen arbeiten dabei auf den linearen, physikalischen Adressen, Speicherschutz existiert nicht. Diese Einfachheit des Betriebssystems ist zwar für die Ausführungsgeschwindigkeit und Bedienbarkeit der Anwendungen und des Systems nützlich, abgesicherte Systeme, also die Abhärtung des Betriebssystems vor internen und externen Angreifern, lassen sich jedoch mit dieser Basis nicht erreichen. Neueste Versionen basieren auf moderner Hardware, wodurch Speicherschutz mit virtuellem Speicher eingesetzt wird. Die Sicherheitsarchitektur besteht aus einem typischen Kryptoprovider im Benutzeradressraum, der die gängigen symmetrischen und asymmetrischen Verfahren anbietet. Neben der Möglichkeit, Code und Daten zu signieren findet sich auch ein Authentifizierungsmanager, der verschiedenste Mechanismen, auch benutzerdefinierte, zur Authentifizierung gegenüber Anwendungen und dem System integriert. Verglichen mit Windows CE ist die Flexibilität der Systemarchitektur geringer, ist sie doch im Wesentlichen für den Gebrauch in persönlichen digitalen Assistenten gedacht. Der Kern kann nur komplett als solcher verwendet werden, ein eigenes, angepasstes Erzeugen eines Kerns ist nicht vorgesehen. Die Sicherheitsarchitektur, die für diese Arbeit im Vordergrund steht, unterscheidet sich nur in der Realisierung, die gebotenen Konzepte sind sehr ähnlich. Für eine Wahl zwischen den beiden Systemen spielen Funktionalitäten auf Anwendungsebene eine Rolle, die hier nicht näher betrachtet werden. Die Klassifikation von Palm OS ist in 3.10 dargestellt.

Eigenschaft	Ausprägung
Spezielles Sicherheitsmodell	keines
Security Kernel	keiner
Speicherschutzkonzept	virtueller Speicher
Schwachstellen-/Bedrohungsanalyse	keine
Entwicklungsvorgehen für sichere Systeme	keines
Kryptografische Verfahren	Kryptoprovider, erweiterbar

Tabelle 3.10: Klassifikation von Palm OS ab 4

Symbian OS

Ähnlich wie Palm OS wurde Symbian OS für eine spezielle Anwendung entwickelt, als Betriebsystem für persönliche Endgeräte, im wesentlichen für Mobiltelefone. Eine Besonderheit dabei ist, dass die Systemschnittstellen und Bibliotheken zwar in C++ entwickelt, aber sowohl von C/C++, als auch von Java aus vollständig angesprochen werden können. Der Kern verwaltet virtuellen Speicher, das Scheduling, die Interprozesskommunikation,

das Energiemanagement und kann durch dynamisches Binden zur Laufzeit erweitert werden. Ebenso wie Windows CE und Palm OS werden in der Sicherheitsarchitektur Kryptoprovider, Signaturmechanismen für Daten und Code und ein Authentifizierungsmanager angeboten. In der Spezifikation wird explizit erwähnt, dass eine Integration von Digital Rights Management Systemen (DRM) vorgesehen ist. Die freie und für diese Arbeit zur Verfügung stehende Dokumentation beschreibt diese Mechanismen allerdings nicht näher. Die Klassifikation der Sicherheitskonzepte zu Symbian OS ist in 3.11 zu sehen.

Zu den Systemen Windows CE, Palm OS und Symbian OS bleibt zusammenfassend zu sagen, dass deren Sicherheitsarchitekturen aus bekannten, einzelnen und nicht integrierten Mechanismen bestehen. Diese können gegen bestimmte Angriffe zwar wirkungsvoll sein, ein Gesamtkonzept stellt das jedoch nicht dar. Wünschenswert wäre zumindest eine klare Definition, welchen Schutz man mit den Mechanismen und dem gegebenen Betriebssystem erreichen kann und welche Angriffe abgewehrt werden. Da die zur Verfügung stehende Dokumentation spärlich und kein Zugriff auf den vollständen Quellcode möglich ist, sind diese Systeme für die systematische Sicherheitsanalyse und anschließende Realisierung wichtiger Mechanismen nur sehr bedingt geeignet.

Eigenschaft	Ausprägung
Spezielles Sicherheitsmodell	keines
Security Kernel	keiner
Speicherschutzkonzept	virtueller Speicher
Schwachstellen-/Bedrohungsanalyse	keine
Entwicklungsvorgehen für sichere Systeme	keines
Kryptografische Verfahren	Kryptoprovider, erweiterbar

Tabelle 3.11: Klassifikation von Symbian OS

Linux

Linux stellt eigentlich ein Betriebssystem für stationäre Systeme dar, da es ein Unix-artiges System ist. Es bietet jedoch derartige Vorteile, dass es sogar mit den Betriebssystemen vergleichbar ist, die speziell für mobile Systeme entwickelt wurden.

Die Architektur von Linux ist flexibel und plattformunabhängig, so dass es heute auf etwa 40 völlig unterschiedlichen Hardware-Archtiekturen übersetz- und ausführbar ist. Der Portierungsaufwand auf neue Rechnerarchitekturen ist im Allgemeinen gering. Die Struktur des Kerns ist monolithisch, kann aber durch den sogenannten Modullademechanismus zur Laufzeit verändert werden. Die veränderten Teile des Kern sind allerdings nicht vom

übrigen Kernsystem getrennt, sondern arbeiten im selben Adressraum ungeschützt voneinander. Die Prioritäten der Prozesse für das Scheduling können in drei Bereichen vergeben werden, die das Schedulingverfahren bestimmen. Es kann nach dem FIFO-Prinzip, einem prioritätsbasiertem Round-Robin Verfahren oder einem nicht präemptiven Real-Time Verfahren durchgeführt werden. Die Speicherverwaltung für den virtuellen Speicher setzt Speicherschutz durch und realisiert schnelle Interprozesskommunikation durch gemeinsam verwendeten Speicher. Neben der üblichen Rechteverwaltung unter Unix wie ACLs für die Dateisystemverwaltung, Hierarchie der Prozesse und Benutzer- und Gruppenverwaltung werden im Linux-Kern auch Capabilities für die Prozesse verwaltet. Dieses rudimentäre Konzept, das momentan nur wesentliche Berechtigungen kontrolliert, wie den Aufruf von Systemdiensten oder die Verwendung von privilegierten Befehlen, kann leicht erweitert und dem Einsatzgebiet angepasst werden. Daneben existieren zahlreiche Projekte, die den Linux-Kern um wesentliche Sicherheitsmechanismen erweitern. Zu erwähnen sind hier die abgesicherten Dateisysteme, verschiedene Zugriffskontrollmodelle und Kryptobibliotheken im Benutzer- und auch im Kernadressraum. Zudem existiert eine Mandatory Access Control Implementierung, die in [LS01a], [LS01b] und [SEL] beschrieben wird. Einen Überblick über moderne Sicherheitsmechanismen unter Linux findet sich in [BBEG02] und [ZH02]. Nicht nur die Sicherheitsmechanismen sind mit denen der explizit für mobile Endgeräte entwickelten Betriebssysteme vergleichbar. Auch die anderen wesentlichen Funktionen, wie Protokolle und Treiber für drahtlose Geräte, Unterstützung der Hardware aktueller mobiler Endgeräte und Energiemanagement werden von Linux angeboten. Wie beispielsweise in [Tox01] und [MM00] gezeigt wird, existieren verschiedenste Open Source Sicherheits-Tools. Andere Diskussionen über Unix Hilfsprogramme finden sich in [MFS90], Ergebnis der vorgenommenen Klassifikation zu den Sicherheitskonzepten von Linux ist in 3.12 zu sehen.

Eigenschaft	Ausprägung
Spezielles Sicherheitsmodell	verschiedene integrierbar
Security Kernel	keiner
Speicherschutzkonzept	virtueller Speicher
Schwachstellen-/Bedrohungsanalyse	keine
Entwicklungsvorgehen für sichere Systeme	Open Source (?)
Kryptografische Verfahren	Kryptoprovider im Kern und Benutzeradressraum

Tabelle 3.12: Klassifikation von Linux

3.2.3 Fazit

Die erwähnten Repräsentanten der Betriebssysteme für mobile Geräte zeigen, dass heutige Systeme nur wenige Sicherheitskonzepte umsetzen. Die Unterschiede zu den Systemen für den stationären Betrieb liegen an den beschränkten Ressourcen, die in den mobilen Geräten zur Verfügung stehen. Neue Konzepte, auch was die Absicherung der Systeme gegenüber Bedrohungen und Angriffen angeht, sind nicht vorhanden. Die Sicherheitsprobleme, die beispielsweise durch Kontextwechsel oder Verlust des Gerätes entstehen können und durch die Mobilität ausgelöst werden, werden von den Betreibssystemen überhaupt nicht betrachtet. Ein Ziel bei der Entwicklung der Analysemethode in dieser Arbeit muss demnach sein, dass auch die Sicherheitskonzepte der lange existierenden Betriebssysteme für die mobilen Systeme eingesetzt werden können. Die erwähnten Betriebssysteme stellen nur Repräsentanten dar, die ein spezielles Konzept oder Verfahren umgesetzt haben. Die Aufzählung ist nicht vollständig, zeigt aber die wesentlichen Sicherheitskonzepte auf, die bei der Entwicklung in dieser Arbeit berücksichtigt werden müssen.

Kapitel 4

Modellbildung

Für die Arbeit mit verschiedenen Analysetechniken, die in Kapitel 5 dargestellt und diskutiert werden, sind passende Modelle ein wesentlicher Bestandteil. In sicheren Rechensystemen werden Modelle oft zur Beschreibung der Sicherheitsmaßnahmen eingesetzt, Beispiele hierfür sind in [Lan81] und [Lam73] zu finden.

Mit dem Begriff Sicherheitsanalyse wird der Vorgang bezeichnet, der auf dem Wissen über ein System Aussagen über dessen Sicherheit machen kann. Neben den funktionalen Eigenschaften über das System selbst sind Sicherheitsanforderungen, Sicherheits-Policies und Wissen über den Einsatz des Systems nötig. Zur Analyse wird das System in Form eines passenden Modells dargestellt.

Da die Qualität der Sicherheitsanalyse wesentlich von der Güte der zugrundeliegenden Modelle abhängt, sollen im Folgenden zunächst die Anforderungen an die Modelliermethode beschrieben werden. Anschließend werden bekannte, für diese Arbeit erwähnenswerte Methoden diskutiert und anhand der Anforderungen qualifiziert.

Für jede Analyse eines Rechensystems ist eine möglichst exakte Beschreibung des Systems notwendig. Dabei sind sowohl die genaue Beschreibung aller im System vorkommenden Subjekte wie Objekte notwendig, als auch die dynamische Ablaufstruktur des Rechensystems. Zur Beschreibung, beispielsweise für eine folgende Sicherheitsanalyse, werden Modelle des Systems herangezogen.

Definition 4.1 (Modell eines Rechensystems)
Ein Modell *eines Rechensystems ist die möglichst exakte Darstellung sowohl der internen statischen Strukturen und der dynamischen Prozesse.*

□

Analysen über die Sicherheit eines Systems sind immer im Kontext der Umgebung, des Einsatzgebietes und der Eingaben in das System gültig. Änderungen dieser Eingabe- und Umgebungsgrößen können den Kontext des System so verändern, dass die Sicherheitsana-

lyse erneut durchgeführt werden muss. Beispielsweise wird der Einsatz einer Firewall im angeschlossenen Netzwerk eines Systems im Systemmodell nicht erfasst. Das Entfernen der Firewall würde allerdings die Risiken bestimmter Angriffe deutlich erhöhen beziehungsweise neue Angriffe ermöglichen. Im Folgenden wird der Modellbegriff deshalb so erweitert, dass die Umgebung des Systems zusätzlich in die Darstellung einbezogen wird. Die Modellierung der Umwelt kann nur in gewissen, nicht exakt definierbaren Grenzen geschehen. Letztlich könnte jede beliebige Einflussgröße der Umwelt Einfluss auf die Sicherheit haben, so dass unrealistischerweise die gesamte Welt beschrieben werden müsste.

Ein wichtiges Ziel bei der Entwicklung passender Modelle ist es, das Wesentliche komplexer Zusammenhänge eines Systems schnell und möglichst exakt zu vermitteln, wie etwa in [And75] nachzulesen ist. Ohne Betrachtung der Dimension „Zeit“ bestehen diese Zusammenhänge aus einzelnen Teilen des Gesamtsystems, die jeweils Aufgaben und Funktionalitäten besitzen, und den Beziehungen zwischen diesen Teilen. Dieses statische Abbild des Systems grenzt funktionale Einheiten, die verarbeiteten Daten und andere Komponenten voneinander ab.

Zudem muss für die Betrachtung unterschiedlicher Aspekte auch die Umgebung des Systems modelliert werden. Werden beispielsweise Sicherheitsuntersuchungen am Modell vorgenommen, so spielt der Benutzer und andere Teile der Umgebung eine wichtige Rolle. Viele Angriffe können nur unter Einbeziehung dieser Umgebung verstanden und korrekt dargestellt werden.

Rechensysteme sind dynamische Systeme. Dies betrifft einerseits die Hardware, die während des Betriebs Zustandsänderungen erfährt. Andererseits werden zur Laufzeit software-seitig Prozesse erzeugt und beendet, die Strukturen der Prozesse und der Prozessräume können sich verändern. Diese Zusammenhänge können nur unter Betrachtung der Dimension „Zeit“ in Modellen repräsentiert werden, so dass neben den statischen Abbilder auch dynamische Darstellungen nötig werden. Mögliche Darstellungsformen für statische und für dynamische Zusammenhänge werden in diesem Kapitel diskutiert. Neben den funktionalen Eigenschaften können auch die Angriffe und die Gegenmaßnahmen dynamisch beschrieben werden. Sicherheitsmodelle mit dynamischen Policies werden beispielsweise in [GM82] genau beschrieben.

Ein mit dem Modellbegriff eng verwandter Begriff ist die Abstraktion. Bei der abstrakten Modellierung werden die wichtigen Dinge von den unwichtigen unterschieden. Je nach Grad der Abstraktion werden die Modelle immer einfacher, da nur das Wesentliche dargestellt wird. Neben der Abstraktion sind sowohl die Dokumentation als auch die Berechnung wichtige Eigenschaften der Modellierung. Die Dokumentation ist der Bereich der Modellbildung, der den Fokus auf die Verständlichkeit und Übersichtlichkeit der Beschreibung legt. Bei der Berechnung ist es wesentlich, dass das Modell als Eingabe für einen bestimmten Algorithmus dienen kann. Denkbar sind dabei die Berechnung von Metriken des Systems aus dem Modell als auch die Transformation des Modells.

Oft werden mit Modellen abstrahierte bzw. vereinfachte Beschreibungen von Systemen

dargestellt, um damit arbeiten zu können. Dabei ist stets zu beachten, dass durch die Abstraktion Informationen über das System verloren gehen, die für das Erkennen von Bedrohungen und Schwachstellen notwendig sind. Für die Sicherheitsanalyse muss deshalb ein geeigneter Grad der Abstraktion gewählt werden. In der Praxis zeigt sich dies daran, dass die prinzipielle Funktionsweise von Sicherheitsmaßnahmen korrekt ist und keine logischen Fehler enthält. Trotzdem sind derartige Systeme leicht angreifbar, weil viele Fehler auf Realisierungsebene vorhanden sind. Implementierungsfehler wie überlaufende Puffer oder Tippfehler, wie sie in Kapitel 2 beschrieben wurden, können nur mit realisierungsnahen Modellen gefunden werden.

4.1 Anforderungen

Prinzipiell können Modelle auf zwei unterschiedliche Arten ein System beschreiben.

Die *eigenschaftsorientierte Modellierung* oder auch *deklarativer Modellierung* beschreibt ein System aufgrund seiner Eigenschaften. Diese werden dabei oft als einzelne Regeln oder Axiome formuliert und decken im Laufe der Entwicklung alle wesentlichen geforderten Eigenschaften ab. Diese mathematische Art Systeme zu beschreiben hat einige Vorteile. Die entwickelten Eigenschaften lassen sich relativ leicht formalisieren, um so ein Regelsystem zu beschreiben. Mit diesen Regeln können dann weitergehende Eigenschaften formal eingebracht und nachgewiesen werden. Für formale Arbeiten ist diese Beschreibung daher gut geeignet, da sich die entwickelten Modelle aufgrund ihrer Darstellung leicht mit mathematischen Methoden einsetzen lassen. Genauer wird diese Art des Umgangs mit Modellen auch in [Bro98] und [BS03b] beschrieben.

Die Art und Details die bei der Systemkonstruktion und der Realisierung eine Rolle spielen, insbesondere was die Programmiersprache oder das eingesetzte Betriebssystem betrifft, werden bei der eigenschaftsorientierten Modellierung im Allgemeinen nicht beschrieben, wie etwa in [BAN90] anhand der Authentifizierung beschrieben wird. Diese Informationen sind aber bei Sicherheitsanalysen wichtig. Angriffe entstehen zwar einerseits durch fehlerhaft festgelegte Eigenschaften, andererseits aber ebenso durch fehlerhafte Implementierungen. Zudem müssen modellbasierte Analysen sowohl das System selbst als Grundlage kennen, als auch deren Umgebung. Die eigenschaftsorientierte Modellierung würde sich demnach nur dann zur Sicherheitsanalyse eignen, wenn auch die Umgebung beschrieben ist. Das jedoch korrekt und vollständig zu beschreiben, ist sehr aufwändig, womit man eher die folgende Methode einsetzt.

Bei der *modellorientierten Modellierung* oder auch *operationaler Modellierung* werden die Strukturen und Beziehungen zwischen einzelnen Elementen eines Systems beschrieben. Eigenschaften werden dabei als Attribute angefügt. Wesentliche Vorteile dieser Modellierungstechnik sind die Integration der Beschreibungen sowohl des Systems, als auch der systemnahen Umgebung, die für die Sicherheitsanalyse wichtig ist. Meist sind modellorien-

tierte Darstellungen leichter verständlich, als eigenschaftsbeschreibende. Deshalb können darin auch Änderungen in den Modellen leichter durchgeführt werden und die Beschreibung kann leichter auf Änderungen des Systems reagieren. Nachteilig ist allerdings, dass die modellorientierten Beschreibungen, nicht direkt und einfach in mathematischen Verfahren nutzbar sind, wie es bei Eigenschaftsbeschreibungen möglich ist.

Zur Auswahl der geeigneten Modellierungssprache werden im folgenden Kriterien diskutiert, die die Verwendbarkeit vor allem bezüglich einer Sicherheitsanalyse beschreiben.

Darstellbarkeit und Freiheit des Abstraktionsgrades

Wichtig bei der Analyse ist die Darstellbarkeit unterschiedlicher Aspekte eines Systems. Bei der Beschreibung des Systems können sowohl die zu schützenden Daten bzw. Objekte als auch die Komponenten bzw. Subjekte im Vordergrund stehen. Wie beispielsweise in [JK90] beschrieben ist, stehen bei Informationsflussanalysen die schützenswerten Daten im Vordergrund der Modellierung. Für die Methodik in dieser Arbeit werden die Schnittstellen zwischen einzelnen Komponenten betrachtet. Der Abstraktionsgrad der Systemkomponenten bei der Darstellung ist veränderbar zwischen der konkreten Darstellung bis hin zum Quelltext und der abstrakten, schematischen Darstellung des Gesamtsystems. Die entsprechenden Datenflüsse werden darin beschrieben, im Vordergrund der Modellierung stehen aber die Subjekte des Systems.

Grundsätzlich muss bei der Modellierung zwischen der statischen und dynamischen Darstellung unterschieden werden. Die statische Darstellungsweise beschreibt Systemkomponenten, deren Schnittstellen und statische Beziehungen. Die Systeme, die in der Informatik betrachtet werden, ändern sich mit fortschreitender Zeit. Diese Änderungen umfassen die Struktur der Komponenten und auch die Beziehungen dazwischen. Sowohl die Dimension Zeit als auch zeitliche Veränderungen, die in der Informatik immer in diskreten Schritten geschehen, sind in einer statischen Ansicht nicht darstellbar. Dies ist nur in dynamischen Ansichten möglich, die Veränderungen meist weniger Komponenten über die Zeit beschreiben. Daneben ist für die Sicherheitsanalyse auch die Beschreibung der Umgebung des Systems wesentlich. Idealerweise fordert man eine Modellierungsmethode, die die Beschreibung aller drei Ansichten, die statische, die dynamische und die Umgebung zulässt.

Exaktheit und Güte

Ein abstraktes Modell vergröbert die Sicht auf das System, weshalb Informationen verloren gehen. Bleibt der Grad der Abstraktion jedoch auf Realisierungs- bzw. Quelltextebene, so ist es eine Frage der Exaktheit der Modellierungsmethode, in wieweit das Modell vom System abweicht. Die Güte des Modells ist ein Maß für diese Abwechung. Für die Sicherheitsanalyse ist Exaktheit und hohe Güte bzw. niedrige Abweichung des Modells vom System erforderlich. Einerseits können nur so die Schwachstellen und Bedrohungen erkannt wer-

den. Andererseits können wirksame Gegenmaßnahmen nur am Modell ausgewählt werden, wenn das Modell exakt ist.

Formalisierbarkeit und Werkzeugunterstützung

Viele Modelle können in einer Sprache dargestellt werden und so als Eingabe für Berechnungen dienen. Diese Berechnungen können sowohl manuell als auch automatisch durchgeführt werden. Diese Formalisierbarkeit ist vor allem dann wichtig, wenn die Berechnungsmethode durch Werkzeuge unterstützt werden soll. Beispiele hierfür sind die automatische Testgenerierung aus dem Modell heraus oder die automatische Codegenerierung. Bei der Sicherheitsanalyse wäre eine automatische Suche nach bestimmten Schwachstellen wünschenswert, wie sie von einigen einfachen Werkzeugen bereits durchgeführt wird, siehe dazu beispielsweise [VBKM02].

Erweiterbarkeit und Anpassbarkeit

Die Programmsysteme, die durch die Modelle ausgedrückt werden, unterliegen ständigen Veränderungen durch Weiterentwicklung oder Fehlerkorrektur. Dadurch ist es wichtig, dass Modell und System gegenseitig aufeinander angepasst werden können. Ergänzt wird diese Erweiterbarkeit durch die Anpassbarkeit, die Teile von Modellen wiederverwendbar machen kann.

4.2 Beschreibungssprachen für Systemmodelle

Im Folgenden werden einige Modellierungsmethoden angesprochen, die im Rahmen dieser Arbeit betrachtet wurden. Daneben existieren noch viele weitere Methoden, die nicht näher angesehen wurden, trotzdem für die Sicherheitsanalyse geeignet sein können. Die entwickelte Analysetechnik kann in weiteren Arbeiten auf diese Modelle bei Bedarf angepasst werden. Die Auswahl der folgenden Methoden erfolgte aufgrund einiger Randbedingungen. Zum einen ist es aufgrund der Komplexität unmöglich, einen kompletten Überblick über die zur Verfügung stehenden Modellierungsmöglichkeiten zu geben. Zum anderen genügt es, eine Methode zu finden, die ungefähr den oben genannten Anforderungen genügt, um als Basisbeschreibung für die Sicherheitsanalyse zu dienen. Die folgende Auswahl besteht deshalb aus den dem Autor bekannten Methoden, die überhaupt für die Analyse sinnvoll erscheinen. Sie decken den großen Bereich zwischen den informellen und einem Teil der formalen Modelle ab.

4.2.1 Informelles Modellieren

Für bestimmte Anforderungen bei der Beschreibung und Dokumentation von Softwaresystemen kann die informelle Modellierung ausreichend sein. Meist wird dabei ohne festgelegte Syntax ein Modell hoher Abstraktionsstufe gezeichnet und durch entsprechenden Text ergänzt. Den Zusammenhang zwischen Text und Zeichnungselementen bilden passende Namen für die einzelnen Elemente. Obwohl dies nicht als exaktes Vorgehen in der Informatik zählt, können damit in kurzer Zeit wesentliche Zusammenhänge vermittelt werden. Grundvoraussetzung für die Nutzbarkeit dieser Art der Modellierung ist eine gute Wahl der Namen der Elemente, da durch das Verstehen der Namen bereits Wissen über das System assoziiert werden kann. Andererseits können verwirrende Bezeichner die Bedeutung des konkreten Modells verdecken. Selbst für Sicherheitsanalysen wird das informelle Modellieren heute genutzt, wie im Kapitel 5 bei der STRIDE-Methode beschrieben wird. Für einfache Modelle, die nicht für Berechnungen oder Weiterverarbeitungsschritte vorgesehen sind, kann das informelle Modellieren geeignet sein. Oft wird die Syntax aus exakt festgelegten Modellierungsmethoden auch für das informelle Vorgehen verwendet. Für einen Leser der Modells führt dies jedoch häufig zu Missverständnissen.

Entity-Relationship Diagramme

Zur datenorientierten Beschreibung werden oft E-R Diagramme eingesetzt. Diese stellen verschiedene Beziehungstypen zwischen den Entitäten, wie etwa Objekten und Subjekten, dar. Die ausdrückbaren Beziehungen sind im Wesentlichen Vererbung (ist-ein) und Assozation (hat-ein) in Anlehnung an die objektorientierte Sicht. Zudem wird für eine Beziehung der Beziehungsgrad festgelegt, wie $1:1$, $1:n$ oder $m:n$. Für die Beschreibung von Sicherheitsattributen wie Access Control Lists oder Datenmarkierungstechniken ist die E-R Beschreibungstechnik geeignet, für die Darstellung eines komplexen Systems mit Ablaufstrukturen aber nicht.

UML

Die Unified Modeling Language (UML) stellt die Vereinigung verschiedener Objektorientierter Modellierungssprachen dar. Sie besteht aus einer Menge von Diagrammtypen deren Elemente durch die UML festgelegt werden. Zwischen den Elementen findet Kommunikation statt, sowohl synchrone, als auch asynchrone Kommunikation kann beschrieben werden. Die exakte Bedeutung der einzelnen Elemente wird bewusst nur sehr allgemein beschrieben. Die Semantik ist deshalb nicht so streng definiert wie in anderen formalen Techniken, da die UML als universelle Sprache für die Beschreibung von Systemen, auch außerhalb der Informatik, dienen soll.

Die Diagrammtypen umfassen im Wesentlichen drei Typen von Darstellungsarten, statische und dynamische Systemsichten und Diagramme zur Darstellung des Kontextes. Diese

werden beispielsweise in [RJB98] näher beschrieben.

Eines der ersten Projekte zur Verflechtung von Sicherheitseigenschaften in UML Beschreibungen ist UMLsec, das in [JJ02] genauer beschrieben wird. UMLsec ist eine Erweiterung, die UML-Modelle um die Beschreibung von Sicherheitseigenschaften erweitert.

Erweiterungen in UML werden durch Profile beschrieben. Diese bestehen generell aus Stereotypen, Tagged Values, die verschiedene Namen-Werte-Paare beschreiben und sogenannten Constraints zur Angabe von definierten Randbedingungen. Diese drei Profilelemente werden in den Diagrammen an die entsprechenden Elemente angefügt, deren Inhalt aber nicht näher beschrieben. UML bietet für verschiedene Diagrammtypen die Möglichkeit, sogenannte Stereotypen an Elemente anzufügen. In diesen Attributen werden zusätzliche Eigenschaften beschrieben, die durch UML nicht genauer definiert werden. Dadurch können Aspekte jenseits der UML-Spezifikation sowohl informell, als auch durch eine formale Sprache beschrieben werden. UMLsec definiert ein formales Modell für verschiedene Diagrammtypen, basierend auf der allgemeinen UML Spezifikation. Es werden eine Reihe von Stereotypen festgelegt, die einen bekannten Angriff und dessen Kontext beschreiben. Durch Analysemethoden kann die Wirksamkeit der Angriffe beziehungsweise die Verwundbarkeit des Systeme dagegen erkannt werden.

UML bietet eine gute Grundlage für die Systemmodellierung zur späteren Sicherheitsanalyse. Durch zahlreiche Werkzeuge für UML ist sowohl der Schritt vom Modell zum Code, als auch der umgekehrte Schritt der Modellbildung (teil-) automatisch durchführbar. Die Diagramme sind erweiterbar und mit den unterschiedlichen Darstellungen durch die angebotenen Diagrammtypen können statsiche Systemkomponenten, als auch dynamisches Systemverhalten beschrieben werden. Durch die grafische Darstellung der Sprache sind UML Beschreibungen einfach verständlich und im Allgemeinen übersichtlich und erweiterbar. Nur schwer darstellbar sind jedoch Modelle, deren Abstraktionsstufe sehr nah am realisierten Programmcode liegt, da Implementierungsdetails in der grafischen Darstellung stets verborgen bleiben.

4.2.2 Formale Modellierungen

Im Bereich der formalen Modellierungsmittel sind unterschiedliche Ansätze vorhanden, wie algebraische Spezifikationssprachen und Prozessalgebren, Logikkalküle oder sprachliche Ansätze. Im Folgenden werden drei bekannte Methoden vorgestellt, die zur Erstellung eines Systemmodells geeignet sein können. Die Liste ist nicht vollständig. Es können noch sehr viele andere Methodiken aufgezählt werden, was aber nicht im Rahmen dieser Arbeit durchgeführt wird.

Specification and Description Language

Basis für die SDL-Methode (Specification and Description Language) sind erweiterte Zustandsautomaten. Die Methode wurde ursprünglich für die Spezifikation von Software für Vermittlungsstellen in der Telekommunikation von der CCITT entwickelt. Das Systemverhalten wird durch erweiterte endliche Zustandsautomaten beschrieben, das System wird durch die aufeinander aufbauenden Komponenten Prozeduren, Prozesse, Blöcke und System beschrieben. Die Kommunikation erfolgt asynchron über Kanäle, die einen unendlichen Puffer besitzen und abstrakte Datenstrukturen zur Datenmodellierung. Die Spezifikation kann sowohl grafisch, als auch in Textform durchgeführt werden. Beide Varianten sind ineinander ohne Verlust transformierbar. Für beide Varianten existieren zahlreiche kommerzielle Werkzeuge, die nicht näher betrachtet wurden. Der Fokus bei der Entwicklung von SDL waren verteilte, reaktive Systeme, die zur Kommunikationsvermittlung eingesetzt werden. Mit verwendeten Modell des erweiterten endlichen Zustandsautomaten lassen sich daher Protokoll relativ einfach abbilden. Für die Beschreibung algemeiner Systeme ist SDL deshalb nur bedingt geeignet. Darstellungen auf verschiedenen Abstraktionsniveaus, insbesondere nahe der Realisierungsstrukturen, ist nicht vorgesehen.

FOCUS

Die Methode FOCUS ermöglicht die schrittweise Verfeinerung von der abstrakten Spezifikation bis hin zum ausführbaren System. Ein System ist eine abgegrenzte Einheit, die festgelegte Schnittstellen und Verhalten besitzt. Die Kommunikation zwischen Systemen erfolgt über Kanäle, deren send- und empfangbare Nachrichten festgelegt sind. Systeme selbst bestehen aus Komponenten, die wiederum wie die Systeme über Kanäle miteinander kommunizieren können. Das zeitliche Verhalten der Systemteile wird durch Kommunikationsströme beschrieben. Die mathematisch beschreibbaren Ströme bestehen aus einer Menge der festgelegten send- und empfangbare Nachrichten. Das dynamische Verhalten des Gesamtsystems wird durch eine Relation der Kommunikationsströme beschrieben. Transitionssysteme beschreiben das Verhalten des Systems durch eine Art abstrakte Implementierungsbeschreibung. Diese Systeme sind zustandsorientiert und lassen sich deshalb auch als gerichtete Graphen darstellen. Prädikate, die an den Kanten dargestellt werden, beschreiben den Zustandsübergang mit Vor- und Nachbedingung. Nachdem die Beziehung der Systemkomponenten zwischeneinander durch Nachrichtenaustausch beschrieben wird und keine weiteren Nebeneffekte zwischen den Komponenten möglich ist, können Systemteile komponiert und die Darstellung des Systems dadurch vereinfacht werden. Das Werkzeug AutoFocus unterstützt die Entwicklung durch Projektmanagement und Teilautomatisierung der Methode. Focus stellt eine mächtige, formal fundierte Methode zur Spezifikation sowohl der Komponenten, als auch des Verhaltens dar und unterstützt die Systementwicklung. Sicherheitseigenschaften und Schutzziele, wie sie in dieser Arbeit betrachtet werden, können nicht direkt in den Modellen beschrieben werden. Es wäre aber möglich, die Beschreibung der Kommunikationskanäle und Nachrichten um Sicherheitsattribute zu erwei-

tern um so mögliche Angriffe zu finden. Auf abstrakter Ebene ist Focus dafür gut geeignet. Dennoch lassen sich Realisierungsdetails, die zur Aufdeckung von Pufferüberläufen oder anderen Angriffen dieser Art nötig sind, nur sehr schwer darstellen.

4.2.3 Abstract State Machines

ASM (Abstract State Machines), früher als Dynamic Algebras bezeichnet, stellen keinen Entwicklungsprozess, sondern eine Methode zur exakten Dokumentation dar. Unterstützt wird die Beschreibung eines Softwaresystems in den verschiedenen Entwicklungsstadien bis hin zur Realisierungsebene. Ursprünglich wurde ASM als Mittel zur Beschreibung und Formalisierung von Algorithmen entwickelt. Explizit nicht im Fokus standen Ziele wie Komplexitätsbetrachtungen oder Fragen den Berechenbarkeit. Mittlerweile sind zahlreiche Erweiterungen entwickelt worden, so dass diese Einschränkungen nicht mehr gegeben sind, auch wenn die Methode nicht speziell dafür entwickelt wurde. Allgemeine Anforderungen an die Methode sind Präzision und Verständlichkeit der Beschreibungen, Skalierbarkeit im Einsatz an unterschiedlich komplexen Systemen und die mögliche Ausführbarkeit der Modelle.

Grundlage der Abstract State Machines sind drei Postulate. Werden alle drei von einem System erfüllt, so ist es durch eine entsprechende ASM ausdrückbar. Erstens sind die Abstract States, also die beschreibbaren Zustände, als Strukturen erster Ordnung darstellbar. Alle Zustände besitzen die selbe Menge an Signaturen. Zweitens wird durch Bounded Exploration beschrieben, dass Ausführungen stets atomar und eindeutig sind. Drittens beschreibt das Sequential Time Postulat, dass alle Ausführungen durch eine Sequenz eines Anfangszustands und der gültigen Zustände ausgedrückt werden. Die Terminierung ist durch die Definitionen zunächst nicht beschrieben, kann aber durch eine einfache Erweiterung ausgedrückt werden.

Die Beschreibung wird in programmiersprachlichen Notation über abstrakten Daten vorgenommen. Die Sprache besteht nur aus wenigen definierten Schlüsselwörter, die in der Tabelle 4.1 beschrieben sind.

In der Informatik sind die Systeme dynamisch. Diese Zustände entwickeln sich durch Berechnungen, die in zeitlich aufeinanderfolgenden Schritten durchgeführt werden. Der Wechsel von einem Zustand in den nächsten wird in ASM Beschreibungen durch das Ändern der Interpretation der (bzw. einiger) Funktionen durchgeführt. In der Syntax von ASM wird dies durch die Update Regel dargestellt. Updates sind eine endliche Menge von Zuweisungen folgender Form:

$f(s_1, \ldots, s_n) := t$

Die Werte $(s_1, \ldots, s_n)$ und t werden ausgewertet und anschließend $eval(t)$ an die Werte $eval(s_1, \ldots, s_n)$ zugewiesen. Dieser Vorgang beschreibt einen Zustandswechel und kann gedanklich als Änderung der Speicherstellen $s_1, \ldots, s_n$ verstanden werden. Partielle Funk-

Regel	Bedeutung	Beispiel
SkipRule	Keine Operation	**skip**
UpdateRule	Wert von f wird t	$f(s_1, \ldots, s_n) := t$
BlockRule	P und parallel Q	P **par** Q
ConditionalRule	Wenn φ wahr, dann P, sonst Q	**if** φ **the** P **else** Q
LetRule	Zuweisung von t nach x, dann P	**let** $x = t$ **in** P
ForallRule	P parallel für alle x mit φ	**forall** x **with** φ **do** P
ChooseRule	Wähle x mit φ, dann P	**choose** x **with** φ **do** P
SequenceRule	P und dann Q	P **seq** Q
CallRule	Regelaufruf mit Parametern	$r(t_1, \ldots, t_n)$

Tabelle 4.1: Syntax der Abstract State Machines durch Transitionsregeln

tionen werden durch die Einführung des Wertes *undef* total. Für undefinierte $f(x)$ wird das Ergebnis $f(x) = undef$ durch den festen Wert *undef* interpretiert. Bei der Ausführung werden alle gültigen Updates des aktuellen Zustands gleichzeitig ausgeführt. Sind diese konsistent und ausführbar, so gelangt das System in den nächsten Zustand. Nicht konsistente Updates werden immer als Fehler interpretiert und insgesamt nicht ausgeführt. Die Darstellung der Einzelschritte, die alle parallel ausgeführt werden, bringt Vorteile bei der Modellierung und der Abstraktion mit sich. Hervorzuheben ist hier vor allem die einfache Abstraktion oder Konkretisierung von Beschreibungen in ASM. Bei Verfeinerungen wird beispielsweise die Liste der parallel auszuführenden Anweisungen einfach um zusätzliche Aspekte ergänzt. Das Problem, dass durch die Hinzunahme neuer Aktionen Seiteneffekte auftreten, wird dadurch auf die Interpretation der ASM Beschreibung verschoben. Bei der Sicherheitsanalyse übernimmt diese Interpretation der Mensch, der aufgrund des Wissens über das System Seiteneffekte berücksichtigen kann. Die Beschreibung wird dadurch vereinfacht.

Eine wesentliche Erweiterung der Abstract State Machines Modellierungsart ist in [BS00] nachzulesen. Darin wird unter anderem ein Operator für sequentielle Beschreibung und Kompositionierung entwickelt. Dieser wird für die in dieser Arbeit entwickelten Methodik eingesetzt, insbesondere bei der Darstellung realisierungsnaher Codeteile.

Auf eine detailierte Beschreibung der Sprache wird hier verzichtet. Eine mathematisch

exakte Definition und verschiedene, teilweise auch industriell eingesetzte Projekte, finden sich beispielsweise in [BS03a]. Ein Tutorial ist in [HW] zu finden. Zur Ausführung von ASMs existiert das Projekt ASML nach [BG03], das als ausführbare Spezifikationssprache auf der Basis der ASM Theorie entwickelt wurde und zur Spezifikation komplexer Software eingesetzt wird.

4.3 Auswahl der Methode

Die aufgezeigten Modellierungsmethoden verfolgen jeweils unterschiedliche Ziele und sind für unterschiedliche Einsatzgebiete entwickelt worden. Sie sind in bestimmten Anwendungsbereichen sinnvoll einsetzbar. Das Einsatzgebiet wird meist bei deren Entwicklung festgelegt. Viele Methoden können aber darüber hinaus verwendet werden. Für den Einsatz als Beschreibungsbasis zur Bedrohungsanalyse eignen sich die untersuchten Methoden unterschiedlich gut, da keine speziell dafür entwickelt wurde. Die aufgezeigten Modellierungsmethoden stellen nur einen kleinen Ausschnitt aus der Vielzahl der verfügbaren Methoden dar und geben hier nur einen Überblick. Es wurde bei der Auswahl versucht, die Bandbreite der charakteristischen Modellierungsmethoden abzudecken. Erwähnt wurden dann die geläufigen Vertreter der jeweilen Klasse.

Im Kontext dieser Arbeit wird eine Methode ausgewählt, die sowohl eine verständliche Darstellung bietet, als auch verschiedene Grade der Abstraktion und Konkretisierung bietet. Idealerweise können sowohl konkrete programmiersprachliche Konstrukte, als auch abstrakte Systembeschreibungen mit der selben Methode dargestellt werden. Diese hohe Bandbreite der Anforderungen fordert daher einen gewissen Kompromiss zwischen exakten Methoden und einer informellen Darstellungsweise für bessere Handhabbarkeit. Da die Modelle bei der Sicherheitsanalyse aber nicht für exakte Berechnungen eingesetzt werden, sondern für die Analyse als Leitfaden dienen, kann ein Kompromiss gewählt werden.

Folgende Eigenschaften sind bei der Auswahl einer passenden Methode mit der entsprechenden Gewichtung ausschlaggebend:

- Darstellbarkeit und Freiheit des Abstraktionsgrades
- Exaktheit und Güte
- Statische Ansicht
- Dynamische Ansicht
- Umgebungsdarstellung und Kontext
- Formalisierbarkeit und Werkzeugunterstützung
- Erweiterbarkeit und Anpassbarkeit

Für Abstract State Machines spricht die Darstellung von Programmabläufen und Schnittstellen, die sowohl realisierungsnah, als auch auf abstrakter Ebene sein kann. Obwohl der Methode ein fundiertes mathematisches Modell zugrunde liegt, ist die Syntax leicht verständlich. Die Darstellung von Algorithmen, wofür ASMs ursprünglich entwickelt wurden, kann auf hohem Abstraktionsniveau erfolgen und bis zu einem Modell konkretisiert werden, das der Realisierung in einer Programmiersprache gleichwertig ist und ineinander übersetzt werden kann. Durch Erweiterungen der ursprünglichen ASM Methode können ebenso alle anderen Vorgänge in IT-Systemen beschrieben werden. Neben der statischen Beschreibung des Systems (Algortihmen, Datenstrukturen) durch die ASM Syntax werden auch die dynamischen Eigenschaften der Systeme beschrieben. Dazu werden ASM Funktionen als Zustandsübergänge interpretiert. Fehler werden dadurch beschrieben, dass kein konsistenter Übergang in den nächsten Zustand möglich ist. Zur Werkzeugunterstützung eignen sich die ASM Modelle direkt als Eingabe, da deren Syntax eine leicht interpretierbare Sprache für Tools darstellt. Nicht ohne weiteres möglich ist die Darstellung der Umgebung des Systems mittels ASM Syntax. Es ist für die Sicherheitsanalyse jedoch von wesentlicher Bedeutung, dass die Umgebung modelliert wird. Dies lässt sich jedoch dadurch lösen, dass in das Systemmodell alle Stellen markiert werden, an denen Einflüsse der Umgebung Wirkung auf das Systemverhalten zeigen können. Eigenschaften der Umgebung, beziehungsweise Eigenschaften der Umgebung, die nicht auftreten sollen, werden dann an diesen Stellen beschrieben. Das Vorgehen bei der Konstruktion der Systemmodelle zur Sicherheitsanalyse wird in Kapitel 6 beschrieben.

ASM werden in vielen Bereichen der Informatik eingesetzt, auch bei der konkreten Implementierung von Systemen. Wie beispielsweise in [CGHS00] beschrieben wird, existieren einige Werkzeuge, die mit Beschreibungen in ASM umgehen können. Wegen der Vielzahl der Methoden und deren Kategorien, von denen in dieser Arbeit nur einige untersucht wurden, bieten Abstract State Machines sinnvolle Eigenschaften für das Verfahren. Eine Untersuchung anderer Modellierungsmethoden für die Sicherheitsanalyse ist aber durchaus sinnvoll und wünschenswert, das Analyseverfahren kann mit vernünftigem Aufwand auf andere Modelle übertragen werden, wie etwa die UML.

Als Ergebnis ist festzuhalten, dass Abstract State Machines als Basis zur Sicherheitsanalyse eine passende Modellierungsmethode darstellen. Auch informelle Methoden würden gewisse Vorteile für die Analyse bieten, die aber wegen deren Ungenauigkeiten beispielsweise für Werkzeugunterstützung nicht gut geeignet sind. Wegen ihres mathematischen Modells, das ASM zugrundeliegt, und der damit gegebenen Exaktheit eignen sie sich zur Modellierung unterschiedlicher Abstraktionsniveaus hier[1]. Auch wenn deren mathematisches Modell für diese Arbeit nur im Hintergrund steht und nicht näher diskutiert oder verändert wird, stellen Abstract State Machines die Basis für das in den nächsten Kapiteln beschriebene Analyseverfahren dar.

[1] Ungeachtet dessen existieren noch viele weitere, hier nicht betrachtete Beschreibungsmethoden, die sich ebenso zur Darstellung des Systemmodells eignen können.

Kapitel 5

Sicherheitsanalyse bei der Konstruktion

Die softwareseitige Konstruktion von IT-Systemen ist sehr komplex. Alleine das Erfüllen der gewünschten funktionalen Anforderungen ist schwierig, meist können nicht alle Eigenschaften eingehalten werden. Für die Konstruktion sind bereits viele Entwicklungsmodelle und Prozesse ausgedacht und eingesetzt worden. Sie sind komplex, oft sind formale Methoden integriert. Diese verlangen vom Softwaredesigner viel Wissen und zusätzliche Arbeit. Das führt dazu, dass die Methoden zwar vorhanden, aber tatsächlich nicht immer korrekt verwendet werden.

Werden zudem nicht funktionale Eigenschaften gefordert, wie etwa die Sicherheit eines Systems, dann wird der Entwicklungsprozess weiter verkompliziert. Es müssen sowohl die Eigenschaften zur Einhaltung der Funktionen, als auch zusätzliche Eigenschaften eingehalten werden. Außerdem stehen einige der nicht funktionalen Eigenschaften, insbesondere die meisten Sicherheitseigenschaften, den funktionalen Eigenschaften gegenüber. Dies äußert sich meistens schon darin, dass Sicherheitsprüfungen einen zusätzlichen Aufwand bedeuten und die Performanz des Systems beeinflussen. Andere Eigenschaften, wie beispielsweise der Komfort für den Benutzer, werden häufig durch Sicherheitsmechanismen eingeschränkt.

Denial-of-Service-Angriffe beispielsweise werden heute meist durch zeitliche Kontrollen der Benutzungszugriffe verhindert. Diese Kontrollen schränken allerdings oft den regulären Betrieb eines Dienstes, beziehungsweise seinen Durchsatz, ein. Dabei gilt es einerseits, dem Benutzer transparent zu machen, warum der Dienst eingeschränkt wird, andererseits müssen die festgelegten Schranken auf ein vernünftiges Niveau gesetzt werden. Liegen diese zu niedrig, sind weiterhin Angriffe möglich, sind sie zu hoch, werden die Benutzer zu sehr beeinträchtigt.

Nachträglich die Sicherheitsarchitektur von Softwaresystemen zu verändern gelingt nur schlecht. Ein wesentlicher Grund hierfür ist der enge Zusammenhang zwischen der Architektur zur Durchsetzung der funktionalen Eigenschaften und dem Design der Sicherheits-

mechanismen. Dies ist an einem bekannten Beispiel aus [Amo94] leicht einzusehen.

Darin wird exemplarisch das nachträgliche Erweitern eines Betriebssystems um Sicherheitsfunktionalitäten diskutiert. Das Absichern und damit Verändern der Systemdiensteschnittstelle beispielsweise hätte meist unabsehbare Folgen für die Anwendungen. Entweder diese werden daraufhin verändert, oder man entscheidet sich dafür, die Änderungen im Betriebssystem nicht direkt an der Aufrufschnittstelle zu den Anwendungen hin zu machen, sondern an Funktionen innerhalb des Systems. Unabhängig von der Entscheidung werden die Dienste, die das Betriebssystem nutzen, von der Änderung beeinflusst. Betroffen sind entweder Anwendungen im klassischen Sinn oder Dienste, die die Funktionalität des Betriebssystems erweitern. Als Fazit bleibt, dass Architekturentscheidungen eines Basisdienstes, wie etwa eines Betriebssystems, letztlich auch die Anwendungen beeinflussen. Für ihre Wirkung müssen die Sicherheitsdienste aber möglichst in den untersten Schichten realisiert werden, da ihr Schutz lediglich auf alle höheren Schichten wirken kann. Ein späteres Ändern wäre im Allgemeinen fatal.

In diesem Kapitel werden zunächst die Begriffe definiert, die im Rahmen der Entwicklungsprozesse für abgesicherte Systeme wesentlich sind. Anschließend werden die bekannten Methoden zur Analyse, wie beispielsweise zur Bedrohungsanalyse, in diesen Prozessen beschrieben und anhand beschriebener Eigenschaften miteinander verglichen. Schließlich werden auf dieser Basis die Anforderungen für die in dieser Arbeit entwickelten Methode angegeben.

5.1 Die systematische Sicherheitsanalyse

Im Wesentlichen werden Sicherheitseigenschaften von entwickelten Systemen bei der Konstruktion festgelegt. Alle Entwicklungsprozesse für abgesicherte Systeme, die schon in Kapitel 2 beschrieben wurden, führen zu Beginn eine Analyse der Bedrohungen und möglichen Angriffe auf das System durch. Dabei wird stets das Wissen über die möglichen Angriffe und Bedrohungen auf ein System vorausgesetzt.

Allerdings ist es nicht unwesentlich, dass es momentan nur wenige Verfahren gibt, die eine systematische Suche nach möglichen, wirkungsvollen Angriffen ermöglichen. Zu unterscheiden ist bei einem gegebenen System die Suche nach Schwachstellen, Angriffen und Bedrohungen. Dazu sei auf die entsprechenden Definitionen in Abschnitt 2.1 dieser Arbeit verwiesen.

Der Zusammenhang dieser Begriffe wird im Folgenden kurz erklärt. Schwachstellen sind mit hoher Wahrscheinlichkeit in jedem (nicht trivialen) System enthalten. Sie beschreiben alle Eigenschaften des Systems, die zu einer unerlaubter Umgehung der festgelegten Schutzziele führen können.

Aus den Schwachstellen können Angriffe folgen, die sich diese zu Nutze machen. Dem-

gegenüber existieren allerdings auch Angriffe, die keine Schwachstelle ausnutzen, sondern das System auf andere Weise treffen. Hierfür seien beispielsweise verschiedene Formen von Brute-Force oder Denial-of-Service Attacken genannt, die keinerlei bekannte Schwachstelle ausnutzen, sondern das System durch spezielles Benutzen angreifen. Es ist Frage der Definitionen, ob ein solcher erfolgreicher Angriff nach Bekanntwerden als Schwachstelle des Systems eingeordnet wird. Da dies jedoch erst im eigentlichen Betrieb des Systems geschieht und hier das Thema Systementwicklung im Vordergrund steht, wird dieser Angriff zunächst nicht als Schwachstelle aufgefasst.

Bisher ist jedoch noch nichts über die Wirksamkeit der Angriffe ausgesagt worden. Erst durch bestimmte, ungewollte Folgen und Auswirkungen kann aus einem Angriff eine Bedrohung werden. Es existieren allerdings sowohl Angriffe, die keine Bedrohung für das System darstellen, als auch Bedrohungen, die nicht durch mögliche Angriffe entstehen. In der Praxis ist es deshalb nötig, dass sowohl Schwachstellen, als auch Angriffe und Bedrohungen durch jeweils angepasste Verfahren gesucht werden. Nur eine Kombination daraus kann eine wirkungsvolle Sicherheitsanalyse darstellen. Die Begriffe dazu werden im Folgenden kurz definiert.

5.1.1 Schwachstellenanalyse

Schwachstellen können grundsätzlich durch Unkenntnis und Fahrlässigkeiten entstehen. Sie können aber auch vorsätzlich in ein System eingebracht werden. Diese Unterscheidung ist für bestimmte Teilbereiche notwendig, beispielsweise für die Analyse von Ursachen und rechtlichen Folgen von Sabotage, wie in der Computerforensik zur technischen Aufklärung von Straftaten. Für die in dieser Arbeit betrachtete Sicherheitsanalyse ist dies aber nur nebensächlich, da zum Zeitpunkt der Analyse weder entschieden werden kann, ob vorsätzlich gehandelt wurde, noch dadurch eine spezielle Auswahl der Gegenmaßnahmen getroffen werden könnte. Wichtiger ist die Entdeckung der Schwachstelle. Die Schwachstellenanalyse kann sowohl auf Basis des Quelltextes als auch auf der Basis eines abstrakten Modells durchgeführt werden. Beide Arten zielen auf unterschiedliche Typen von Schwachstellen. Analyse am Quelltext dient zum Entdecken von Realisierungsmängeln und logischen Schwachstellen, abstrakte Modelle helfen nur bei der Suche logischer Schwachstellen. Fehler, die durch nicht korrekt arbeitende Übersetzer in ein System eingebracht werden, können nur dann entdeckt werden, wenn der Übersetzer selbst als Bestandteil des Systems betrachtet und einer Sicherheitsanalyse unterzogen wird.

Die quelltextbasierte Schwachstellenanalyse wird auch Programmanalyse oder Codeanalyse genannt. Sie dient hauptsächlich dazu, Implementierungsfehler und typische Fehler bei der Verwendung der eingesetzten Programmiersprache zu finden. Diese Codeanalyse kann bei gegebenem Quelltext durchgeführt werden. Als klassisches Beispiel ist hier die Verwendung der `strcpy`-Funktion in der Programmiersprache C genannt.

Fallbeispiel: *Gefahr durch die* `strcpy`*-Funktion*
Es existieren eine Menge von Bibliotheksfunktionen für die Programmiersprache C in unterschiedlichen Betriebssystemen. Einige dieser Funktionen können bei unachtsamer Benutzung zu unvorhersehbarem Verhalten führen und Schwachstellen erzeugen. Bekanntes Beispiel hierfür ist die `strcpy`-Funktion mit folgender Signatur:

```
char* strcpy(char* Ziel, char* Quelle);
```

Problematisch wird die Verwendung der Funktion dadurch, dass in einigen Fällen das Verhalten dieser Funktion undefiniert ist. Die beiden Argumente müssen Null-terminiert sein, ansonsten ist das Ergebnis nicht definiert. Ebenso unbestimmt ist das Verhalten der Funktion, wenn einer der beiden Argumente einen Null-Zeiger enthält bzw. nicht initialisiert wurde. Die populärste Schwachstelle entsteht aber dadurch, dass der Quellstring länger als der Speicher für das Ziel ist. Genau dann entsteht ein Pufferüberlauf, der von Angreifer auf vielfältige Weise ausgenutzt werden kann. Abhängig davon, wo sich die Adresse von `char* Ziel` befindet, kann damit ein Überlauf auf den Keller oder auf der Halde erzeugt werden.
Die gängiste Abwehrmaßnahme dagegen sind genaue und teilweise aufwändige Prüfungen der Bereiche vor der Verwendung einer solchen Funktion.

Es existieren noch viele weitere Funktionen, die für derartige Schwachstellen bekannt sind. Mehr Informationen dazu kann man in [JV02] und [HL03] finden. Ist der Quelltext für eine Analyse zu komplex, so wird durch Abstraktion ein Modell des Systems analysiert. Dies wird modellbasierte Schwachstellenanalyse genannt. Falls der Quelltext nicht zur Verfügung steht, kann alternativ dazu der Maschinencode analysiert werden. Wesentliche Unterschiede zur Quelltextanalyse sind dabei die maschinennahe Struktur des Programms und die meistens verlorenen Bezeichner für Programmelemente (Funktions- und Variablennamen usw.). Am Beispiel der Intermediate Language des .NET-Frameworks wurde dies innerhalb dieser Arbeit durchgeführt. Das dafür entwickelte Werkzeug wird in Kapitel 7 beschrieben.

Das Entdecken von Schwachstellen in Systemen nennt man Schwachstellenanalyse. Diese kann sowohl aus der quelltextbasierten und modellbasierten Schwachstellenanalyse bestehen und ist wie folgt definiert.

Definition 5.1 (Schwachstellenanalyse)
Die Schwachstellenanalyse *ist der Vorgang, der die Eigenschaften eines Systems findet, die zur Umgehung der Policy und damit zur Verletzung der Schutzziele ausgenutzt werden können.*

□

5.1.2 Angriffsanalyse

Stehen bei der Analyse die Angriffe im Vordergrund, so wird versucht auf Basis bekannter Angriffsmuster das System auf diese Möglichkeiten hin zu untersuchen. Nötig ist dazu zunächst der Aufbau eines Angriffskatalogs beziehungsweise eine Informationssammlung über bekannte Angriffe. Eine Klassifizierung der Angriffe ist durch die Unterscheidung von internen und externen Orten oder durch den Unterschied von aktiven und passiven Aktionen des Angreifers möglich. Ist das Finden konkreter Angriffe effizient und erfolgreich, so müssen diese später nach ihrer Eintrittswahrscheinlichkeit bzw. ihrem Risiko bewertet werden.

Definition 5.2 (Angriffsanalyse)
Die Angriffsanalyse *dient zur Bestimmung der möglichen Angreifer und aller internen und externen, aktiven und passiven Angriffe. Dazu wird konkretes Wissen über bekannte Angriffe als Wissensbasis genutzt.*

□

5.1.3 Bedrohungsanalyse

Neben dem Auffinden von Schwachstellen und dem Erkennen von möglichen Angriffen ist die systematische Erfassung aller relevanten Bedrohungen auf ein System wichtig. Schließlich sind es die Bedrohungen, die man mit geeigneten Gegenmaßnahmen verhindern möchte. Ein Möglichkeit, Bedrohungen zu ermitteln, kann durch die Bewertung des Risikos bekannter Angriffe auf das System geschehen. Übersteigt das Risiko einen festgelegten Wert, so ist die Angriffsmethode als Bedrohung auf das System einzustufen. Allerdings würde dabei fälschlicherweise angenommen, dass alle Angriffe bekannt sind und Bedrohungen nur aus Angriffen heraus entstehen können.

Eine systematische und mit vertretbarem Aufwand durchführbare Bedrohungsanalyse ist jedoch sehr schwierig. Zudem ist sie nicht vollständig automatisierbar sondern nur durch Werkzeuge unterstützbar. Trotzdem ist es ein wesentliches Ziel bei der Absicherung von Systemen, die Bedrohungen geeignet zu erfassen. Dadurch können entsprechende Gegenmaßnahmen ausgewählt und eingesetzt werden. Die Bedrohungen werden typischerweise in Bedrohungsmatrizen oder Bedrohungsbäumen dargestellt.

Definition 5.3 (Bedrohungsanalyse)
Die Bedrohungsanalyse *ist der Vorgang, der möglichst alle wahrscheinlichen Verletzungen der IT-Sicherheit eines Systems liefert.*

□

5.1.4 Sicherheitsanalyse

Für die Sicherheitsanalyse wird ein möglichst exaktes Modell des zu untersuchenden Systems benötigt. Die Qualität bzw. Vollständigkeit der Analyse wird im Wesentlichen durch zwei Faktoren beeinflusst. Zum einen können durch Fehler im Modell bestimmte Angriffe nicht erkannt werden. Zum anderen bedeuten abstrakte Modelle auf hoher Ebene, dass realisierungsnahe Schwachstellen nicht entdeckt werden können.

Die Sicherheitsanalyse kann zum einen auf ein in der Entwicklung befindliches System angewendet werden kann. Zum anderen kann sie auch zur Abhärtung eines bestehenden Systems eingesetzt werden.

Definition 5.4 (Sicherheitsanalyse)
Die Sicherheitsanalyse *ist der Vorgang, der zu einem gegebenen (Teil-) System möglichst alle erkennbaren Schwachstellen, Angriffe und Bedrohungen liefert. Sie stellt die für den jeweiligen Kontext angepasste Kombination aus Schwachstellen-, Angriffs- und Bedrohungsanalyse dar.*

□

Demgegenüber wird die nachträgliche Bewertung eines abgesicherten Systems als Sicherheitsevaluierung bezeichnet. Obwohl dies nicht direkt Thema im Rahmen dieser Arbeit ist, wird sie an dieser Stelle kurz erwähnt. Zur Sicherheitsevaluierung können die selben Verfahren wie bei der Analyse eingesetzt werden. Darauf aufbauend werden beispielsweise Metriken für die Bewertung der Sicherheit von Systemen eingeführt, was zu folgender Definition führt.

Definition 5.5 (Sicherheitsevaluierung)
Der Vorgang der Sicherheitsevaluierung *bewertet nach einem gegebenen Schema die Qualität und Wirksamkeit der Sicherheitsmechanismen gegenüber den Anforderungen.*

□

Im Folgenden werden deshalb Kriterien eingeführt, die wesentliche Merkmale für eine Sicherheitsanalyse darstellen und nach denen die später beschriebenen Methoden bewertet werden. Die vorgestellten Merkmale gelten prinzipiel auch für die Sicherheitsevaluierung, die aber nicht näher betrachtet wird.

5.1.5 Kriterien zur systematischen Sicherheitsanalyse

Für die Sicherheitsanalyse wären alle der folgenden Kriterien zu erfüllen. Sie sind durch die Anwendung bestehender Methoden zur Bedrohungs-, Angriffs- und Schwachstellenanalysen entstanden, die später in diesem Kapitel diskutiert werden. Diese Kriterien sind für alle drei

Bereiche der idealisierten Sicherheitssanalyse wünschenswert. Allerdings sind die Kriterien in der Praxis im Allgemeinen nur schwer komplett zu erreichen.

1. *Vollständigkeit:* Als Ergebnis der Sicherheitssanalyse werden alle Bedrohungen, Schwachstellen und Angriffe erfasst, die in dem zugrundeliegenden System zum Zeitpunkt der Analyse erkennbar sind. Idealerweise können keinerlei mögliche Bedrohungen vergessen werden oder durch Fehler unerkannt bleiben.

2. *Reproduzierbarkeit:* Eine wiederholte Sicherheitssanalyse erkennt mindestens die Menge der bereits bekannten Bedrohungen, Schwachstellen und Angriffe. Dies gilt auch und insbesondere dann, wenn unterschiedliche Personen die Analyse durchführen. Für die optimale Analyse sollte (die nicht einzuhaltende Annahme) zutreffen, dass unabhängig vom subjektiven Einfluss der analysierenden Person stets dasselbe Ergebnis entsteht.

3. *Zeittransparenz:* Die Analyse muss das korrekte Ergebnis unabhängig vom Zeitpunkt der Analyse ergeben. Dies ist im Wesentlichen bei der Entwicklung eines Systems von Bedeutung, wenn die Analyse iterativ zu den Entwicklungsschritten durchgeführt wird. Sie sollte während der üblichen Entwicklungsphasen permanent eingesetzt werden können.

4. *Strukturierung des Analyseprozesses:* Der Analyseprozess muss strukturiert durchgeführt werden. Dadurch werden die Ergebnisse nachvollziehbar und das Verfahren universell anwendbar. Der Aufwand zur Einarbeitung sinkt und es besteht die Möglichkeit, dass auch Personen Teile der Analyse durchführen können, die keine Sicherheitsexperten sind. Idealerweise kann dann entwicklungsbegleitend Wissen über neue Bedrohungen zu einer bestehenden (Teilsystem-) Analyse hinzugefügt werden, ohne dass die Analyse von vorne begonnen werden muss. Strukturierung muss dabei nicht unbedingt eine Reduktion der Komplexitäten bedeuten, wesentlicher ist die Nachvollziehbarkeit auch bei aufwändiger werdenden Prozessen.

5. *Strukturierung des Ergebnisses:* Die Ergebnisse der Analyse müssen in klaren, verständlichen Strukturen aufgebaut werden. Dadurch sind die Ergebnisse des Sicherheitsteams für die Entwickler zugänglich und verständlich. Sie können so als Basis der Abwehrmaßnahmen auch von Entwicklern eingesetzt werden, die bei der Analyse nicht beteiligt waren.

6. *Wiederverwendbarkeit:* Teile des Ergebnisses der Analyse können später wiederverwendet werden. Das kann ebenso neue Sicherheitssanalysen unterstützen als auch andere Vorgänge bei der Softwareentwicklung betreffen, wie etwa das Testen von Software. Es ist denkbar, dass man Teile des Analyseergebnisses als Muster oder „Pattern“ einsetzt und so eine Bibliothek über Bedrohungs-, Angriffs- und Schwachstellenwissen aufbauen kann. Zukünftige Analysen könnten sich darauf abstützen. Zudem kann der Softwaretest durch die gewonnenen Ergebnisse unterstützt werden.

Wenn beispielsweise eine bestimmte Bedrohung in einem Kontext erkannt wurde, könnte nach ähnlichen Kontexten gesucht werden, die dann auf dieselbe Bedrohung hin getestet werden können.

7. *Angepasster Aufwand der Analyse:* Der Aufwand der Sicherheitsanalyse als Summe der Bedrohungs-, Angriffs- und Schwachstellenanalysen muss in einem vernünftigen Verhältnis zum Wert des Systems und seiner Informationen stehen. Dieses sehr subjektive Kriterium kann nur im Einzelfall konkretisiert werden. Es muss aber deutlich werden, dass Analysemethoden je nach untersuchtem System sehr aufwändig sein können und nur bei vollständiger Anwendung sinnvoll sind.

8. *Werkzeugunterstützung:* Jede einzelne der drei Teilanalyse ist komplex und zeitaufwändig. Software-Werkzeuge für die Analysen helfen Fehler zu vermeiden, Ergebnisse einfach wiederverwenden zu können und den Zeitaufwand der Analysen gering zu halten und abschätzbar zu machen. Deshalb sollte ein Sicherheitsanalyseverfahren die Möglichkeit zur Werkzeugunterstützung bieten. Aufgrund der prinzipiellen Sicherheitsprobleme, wie sie schon zu Beginn dieser Arbeit beschrieben wurden, kann der Analysevorgang nie automatisch durchgeführt werden. Die erforderlichen Werkzeuge können den Analyseprozess also nur unterstützen.

Es existieren bereits einige Methoden zur Durchführung dieser Analysen. Sie unterscheiden sich allerdings erheblich untereinander und bei der Bewertung durch die oben genannten Kriterien. In den folgenden Abschnitten werden die bekannten Methoden und Techniken vorgestellt und mittels der genannten Kriterien bewertet. Diese Bewertung erfolgt auf einer Skala von 1 (++) bis 5 (−−), die zum Aufzeigen der Defizite der Methoden genügt. Anschließend wird eine neue Methode entwickelt.

5.2 Schwachstellenorientierte Analysetechniken

Die schwachstellenorientierte Analysetechniken können erst eingesetzt werden, wenn die Umsetzung begonnen hat und Systemteile realisiert werden, da bekannte Analysemethoden zum Auffinden von Schwachstellen stets auf Basis des Quelltextes arbeiten. Mit Hilfe von Werkzeugen wird dazu der Programmcode verarbeitet und auf Funktionen durchsucht, die für bestimmte Gefährungen bekannt sind. Überblick über die bestehenden Werkzeugen finden sich in [VBKM02] und [JV02]. Diese Werkzeuge führen die ersten Phasen eines Übersetzungsvorgangs durch, die syntaktische Analyse des Quelltextes. Der gesamte Quelltext wird dabei als eine Einheit analysiert. Mit den Werkzeugen ist eine Teilanalyse von interessanten Codeabschnitten nicht möglich. Aus den erkannten Token werden diejenigen Funktionen markiert, die in einer Liste mit bekannten Schwachstellen übereinstimmen. Letztlich entscheiden kann aber nur der Entwickler, ob eine markierte Codestelle tatsächlich eine Schwachstelle für das System darstellt. Die Werkzeuge liefern dafür nur

Hinweise und hängen in großem Maße von dem zur Verfügung stehenden Wissen über bekannte Bedrohungen ab. Eine Bewertung dieser Analysetechnik findet sich in der Tabelle 5.1. Für die Suche bestimmter Sicherheitsprobleme ist diese Technik geeignet, macht aber für eine vollständige Sicherheitsanalyse eines Systems nur im Zusammenhang mit der Analyse von Bedrohungen Sinn, da Schwachstellen nur eine mögliche Voraussetzung für Bedrohungen darstellen.

Kriterium	Bewertung
Vollständigkeit	abhängig vom Wissen über Schwachstellen (−)
Reproduzierbarkeit	reproduzierbar, Tools wiederverwendbar (++)
Zeittransparenz	Analyse immer durchführbar (++)
Strukturierung des Analyseprozesses	keine Struktur vorhanden, Werkzeuge als Strukturmaßnahme nicht geeignet (−−)
Strukturierung des Ergebnisses	keine Struktur der Ergebnisse (−−)
Wiederverwendbarkeit	Werkzeuge, Wissen muss aufgebaut werden (+)
Angepasster Aufwand der Analyse	Nur komplette Quellen analysieren (−−)
Werkzeugunterstützung	Werkzeuge als Hilfen verfügbar (++)

Tabelle 5.1: Bewertung der schwachstellenorientierten Analyse auf Quelltextbasis

5.3 Angriffsorientierte Analysetechniken

Mit Hilfe der Sammlungen bekannter Angriffe in der Literatur oder im Internet kann man versuchen, während der Entwicklung oder im Betrieb mögliche Angriffe auf ein System zu finden. Es existieren aber nur wenige bekannte, derartige Verfahren, die im Folgenden kurz beschrieben werden.

5.3.1 Ad-hoc Durchführung

Die naheliegendste Methode, mögliche Angriffe auf ein System zu erkennen, ist eine Ad-hoc Durchführung, die mehr oder weniger nach dem Fachwissen der analysierenden Person eine Menge von durchführbaren Angriffen liefert. Für das genauere Verständnis der Anwendung

kann ein Brainstorming mit den Entwicklern nützlich sein. Zur Strukturierung könnte man die Unterteilung zwischen internen und externen, aktiven und passiven Angreifern nutzen.

Diese Methode ist deswegen erwähnenswert, weil sie heute meist die einzige ist, die bei vielen Entwicklungen durchgeführt wird. Gefundene Angriffe ergeben sich meistens aus Kenntnissen bekannter Angriffe und vergleichbaren Systemarchitekturen.

Bewertung

Dieses einfache Analyseverfahren kann leicht in Entwicklungsprozesse integriert werden. Das breit vorhandene Wissen über konkrete Angriffe kann dazu verwendet werden. Allerdings kann weder die Vollständigkeit der Angriffe, noch die Reproduzierbarkeit der Analyse erreicht werden. Meist werden wichtige Angriffsklassen vergessen und das Ergebnis kann nur schwer bei anderen Analysen wiederverwendet werden. Werkzeuge existieren dafür nicht, es sind höchstens Datenbanken denkbar, die alle bekannten Angriffe gewisser Angriffsklassen enthalten können. Eine Bewertung dazu ist in Tabelle 5.2 dargestellt.

Kriterium	Bewertung
Vollständigkeit	nicht erreichbar (−−)
Reproduzierbarkeit	nicht möglich (−−)
Zeittransparenz	nicht erreichbar (−−)
Strukturierung des Analyseprozesses	keine Struktur vorhanden (−−)
Strukturierung des Ergebnisses	keine Struktur der Ergebnisse (−−)
Wiederverwendbarkeit	nur sehr eingeschränkt möglich (−)
Angepasster Aufwand der Analyse	(+)
Werkzeugunterstützung	(−−)

Tabelle 5.2: Bewertung der Ad-hoc Angriffsanalyse

5.3.2 Attack Graphs

Ein bekannte, modellorientierte Analysetechnik für Angriffe auf Netzwerke ist die Entwicklung sogenannter Attack Graphs. Diese werden auch in [SJW02] und [JSW02] beschrieben. Zur Erstellung nutzt man das vorhandene Wissen über bekannte Angriffe und über die Profile der Angreifer. Diese Informationen müssen von vornherein bekannt sein, können aber nachträglich erweitert werden, wenn beispielsweise neue Angriffe bekannt werden. Die

Strukturen der zu untersuchenden Netzwerke werden als Modell in Form von endlichen Automaten beschrieben. Im ursprünglichen Ansatz zur Entwicklung von Attack Graphs, der in [PS98] beschrieben wird, werden die folgenden drei Eingaben zur Analyse gefordert.

1. *Attack Templates:* beschreiben generische Angriffsmuster in Form von Graphen. Jeder Knoten des Graphen enthält Informationen über den Zustand im jeweiligen Angriffsschritt. Diese Informationen sind wie folgt gegliedert:

   - Benötigte Benutzerrechte
   - Beteiligte Systeme
   - Schwachstellen, die durch den Angreifer ausgenutzt werden
   - Aktueller Zustand
   - Zustandseigenschaften

   Die Kanten beschreiben die einzelnen Aktionen des Angreifers oder andere Informationen über relevante Ereignisse.

2. *Attacker Profiles:* beinhalten die Mittel und Verfahren, deren sich der potentielle Angreifer bedienen kann. Darunter sind beispielsweise Hilfsprogramme zur verstehen, die Angriffe automatisch durchführen oder die Möglichkeit, hohe Rechenleistung zum Brechen von kryptografischen Schlüsseln einzusetzen. Zudem sind wichtige Möglichkeiten beschrieben, etwa physikalischer Zugang zum Netzwerk oder zu den Rechnern und wieviel Geld und Zeit der Angreifer zur Verfügung hat.

3. *Configuration Files:* beschreiben die wesentlichen Informationen über die vorhandenen Betriebssysteme, die Netztopologie, die Art und Konfiguration der Netzwerke und die Konfigurationen der aktiven Netzwerkgeräte. Insbesondere müssen für jedes Netzwerkgerät folgende Informationen bereitgestellt werden:

   - Art des Gerätes (Workstation, Drucker, usw.)
   - Hardwaretyp (Intel-PC)
   - Betriebssystem (Typ, Version, Patch-Level)
   - Benutzer und Gruppen
   - Konfiguration der Netzwerkschnittstelle (Dienste, Ports, usw.)
   - Netzwerkanschluss (Eigenschaften von Schicht 1 und 2)

Dazu werden oft sehr große Diagramme von sogenannten „Red Teams“ entwickelt und diese dienen dann zur (meist informellen) Analyse von Angriffsabläufen auf die untersuchten Netzwerke.

Das übliche manuelle Entwickeln ist allerdings sowohl fehleranfällig, als auch für größere Netzwerke (ab einigen hundert Knoten) praktisch nicht mehr durchführbar. Eine Weiterentwicklung für Attack Graphs wird in [SJW02] und [JSW02] diskutiert. Dabei werden durch Model Checking auf einem Modell des Netzes Attack Graphs automatisch generiert. Hinter dem Vorgehen steht ein durchdachtes formales Modell. Es werden neben Angriffen auch andere fehlerhafte Ereignisse beschreibbar, wie etwa Verbindungsfehler oder Fehler durch den Benutzer.

Das Vorgehen dabei ist wie folgt vorgegeben:

1. *Modellierung des Netzwerks:* Das Netzwerkmodell wird als endlicher Automat beschrieben. Die Zustandsübergänge korrespondieren mit den atomaren Aktionen eines Angreifers, die Knoten beschreiben einen bestimmten Zustandskontext. Genauere Informationen über das Erzeugen des Netzwerkmodells werden nicht gegeben. Zusätzlich wird eine geforderte Sicherheitseigenschaft spezifiziert. Diese wird dann vom simulierten Angreifer zu umgehen oder verletzen versucht.

2. *Erzeugen des Attack Graphs:* Mit einer modifizierten Version des NuSMV Model-Checker Werkzeugs werden aus dem erstellten Netzwerkmodell die Attack Graphs generiert. Die Sicherheitseigenschaft wird dazu in Computation Tree Logic (CTL) Form beschrieben und vom Model-Checker ausgewertet. Das Werkzeug versucht jeden denkbaren Attack Graphen als Gegenbeispiel zu erzeugen, der genau diese Sicherheitseigenschaft verletzt.

3. *Analyse des Attack Graphs:* Nachdem die erzeugten Attack Graphs (die Gegenbeispiele) genau auf einem Abstraktionsniveau dargestellt werden, sind die ungeeignet für weitere Analysen, die über das Modelchecking auf bekannte Angriffe hin hinausgehen. Die Graphen werden mit Zusatzinformationen versehen, was nach [SJW02] ebenfalls automatisch geschehen kann. In den Zusatzinformationen werden Hinweise auf den Angriff abgelegt, wie beispielsweise das notwendige Wissen für den Angriff. Denkbar für die Analyse sind auch Anwendungen wie die kleinste Menge an Angriffsschritten herauszufinden, die verhindert werden müssen, um die Sicherheitseigenschaft zu gewährleisten.

Bewertung

Dieses Verfahren beschreibt eine Technik, die zur Einhaltung von Sicherheitseigenschaften in Netzen vorgesehen ist. Prinzipiell sind ganz ähliche Verfahren jedoch auch für die Angriffsanalyse allgemeiner Systeme, auch Betriebssysteme geeignet. Attack Graphs lassen nicht direkt dafür einsetzen, da sie keine Schichten analysieren können, die bei Betriebssystemen nötig sind, wie Hardware, Kern, Treiber und Middleware.

Nach einer Modellierung der zu untersuchenden Netzwerke kann mit Hilfe von Werkzeugen die Angriffssanalyse bezüglich einer festgelegten Sicherheitseigenschaft automatisch

durchgeführt werden. Die Vollständigkeit des Ergebnisses hängt im Wesentlichen von zwei Faktoren ab. Je nach Genauigkeit des Netzwerkmodells können Angriffe unerkannt bleiben, da sie im Modell nicht ausgedrückt und dargestellt werden können. Zudem können Modelle derart komplex werden, dass sie für Model Checking Werkzeuge allgemein nicht in endlicher Zeit durchgerechnet werden können. Über die Komplexitätsgrenzen des Verfahrens schweigen sich die Autoren jedoch aus. Wichtige Eigenschaften dieses Verfahrens sind aber auch die Reproduzierbarkeit der Analyse und die mögliche Wiederverwendbarkeit der entwickelten Netzwerkmodelle oder Teile davon. Eine Bewertung dazu ist in Tabelle 5.3 angegeben.

Kriterium	Bewertung
Vollständigkeit	abhängig vom Modell und seiner Komplexität (+)
Reproduzierbarkeit	möglich (+)
Zeittransparenz	immer (++)
Strukturierung des Analyseprozesses	bis auf Erstellung des Modells strukturiert (+)
Strukturierung des Ergebnisses	Struktur der Ergebnisse durch Attack Graphs (++)
Wiederverwendbarkeit	nur eingeschränkt Teile des Modells (−)
Angepasster Aufwand der Analyse	teilautomatisch (+)
Werkzeugunterstützung	Model Checking unterstützt (++)

Tabelle 5.3: Bewertung der Attack Graphs Methode

5.4 Bedrohungsorientierte Analyse

Wesentliches Ziel bei der Abhärtung eines Systems ist die Eliminierung aller Bedrohungen auf das System. Es existieren einige Methoden zum entwicklungsbegleitenden Umgang mit Bedrohungen. Diese werden im folgenden Abschnitt vorgestellt.

5.4.1 Ad-hoc Durchführung

Wie bei der Ad-hoc Angriffsanalyse hängt die Ad-hoc Bedrohungsanalyse von der Kenntnis der durchführenden Person ab. Allerdings existieren für die Bedrohungsanalyse auch

einfache Strukturierungsmaßnahmen, die einen gewissen Leitfaden bei der Anwendung bieten. Ein Ansatz ist die Erstellung einer Bedrohungsmatrix bei der Analyse. Dabei werden in den Zeilen Angreiferkategorien festgelegt, wie etwa externer und interner Angreifer. In den Spalten werden die möglichen Angriffsziele beschrieben, so dass Elemente in der Matrix jeweils mögliche Aktionen der Angreifer für das beschriebene Ziel enthalten. Es existieren andere Arten der Dokumentation der analysierten Bedrohungen, wie etwa Bedrohungsbäume. Diese werden aufgrund ihrer Bekanntheit in einem folgenden Abschnitt beschrieben und beispielsweise in [Eck02] und [Bis03] genauer erklärt.

Die Ergebnisse der Bedrohungsanalyse sind dann eine Liste von erkannten Bedrohungen. Diese Ergebnisse können an Entwickler weitergegeben werden, die sich um die Umsetzung wirkungsvoller Gegenmaßnahmen kümmern. Diese Analyseart ist auch für bestehende Systeme anwendbar und nützlich um Bedrohungen aufzuzeigen.

Bewertung

Positiv an diesem leicht zu verstehenden Analyseverfahren ist der geringe Aufwand, die leichte Umsetzung und einfache Integration in Entwicklungsprozesse. Ohne genaue Betrachtung der Vorteile anderer Verfahren sind dies wichtige Punkte für eine leichte Anwendbarkeit, die für einen Einsatz in der Praxis der Systementwicklung sprechen.

Allerdings sind die hier untersuchten Kriterien durchweg nicht einzuhalten. Vollständigkeit der Angriffe kann nicht annähernd, theoretisch höchstens zufällig erreicht werden. Im Allgemeinen sollte aber davon ausgegangen werden, dass wichtige Angriffsklassen vergessen werden und so kein Gesamtansatz eines sicheren Systems entstehen kann. Ebenso schlecht sind die Ergebnisse reproduzierbar und für spätere Einsätze wiederverwendbar. Es existieren weder Strukturen, um die Ergebnisse darzustellen und zu dokumentieren, noch um die Vorgehensweise des Analyseprozesses zu protokollieren. Personen, die eine Ad-hoc Analyse durchführen, müssen sowohl viel Wissen über das System, als auch über IT-Sicherheit besitzen. Sicherheitsexperten ohne Kenntnisse über das zu untersuchende System können diese Methode nicht alleine verwenden. Werkzeuge werden dabei höchstens zur Dokumentation eingesetzt, daher ist die Wiederverwendbarkeit der Ergebnisse nur schlecht möglich, wie auch in Tabelle 5.4 gezeigt wird.

Trotzdem ist diese Art der Analyseverfahren wichtig, da mit ihnen sämtliche denkbaren Bedrohungen beschrieben werden können. Zudem ist der Aufwand der Analyse anpassbar an die vorhandenen Ressourcen, und die Ergebnisse sind verständlich.

5.4.2 Bedrohungsbäume

Neben Bedrohungsmatrizen wurden zur Beschreibung und Darstellung von Bedrohungen auch Bedrohungsbäume entwickelt, im Englischen Threat Trees genannt. Bäume sind für

Kriterium	Bewertung
Vollständigkeit	nicht erreichbar (−−)
Reproduzierbarkeit	nicht möglich (−−)
Zeittransparenz	nicht erreichbar (−−)
Strukturierung des Analyseprozesses	keine Struktur vorhanden (−−)
Strukturierung des Ergebnisses	Struktur der Ergebnisse durch Darstellungen möglich (Bäume, Matrix) (+)
Wiederverwendbarkeit	nur sehr eingeschränkt möglich (−)
Angepasster Aufwand der Analyse	(+)
Werkzeugunterstützung	(−−)

Tabelle 5.4: Bewertung der Ad-hoc Bedrohungsanalye

die Darstellung von Bedrohungen eine geeignete Datenstruktur, da sie durch den Pfad vom Blatt bis zur Wurzel alle Eigenschaften und Aktionen beschreiben, die für die Wirksamkeit der Bedrohung nötig sind. Diese beschreibbaren Eigenschaften und Aktionen sind notwendige Voraussetzungen für die Bedrohung und werden im Folgenden Bedrohungsfaktoren genannt. Die Bedrohung, beispielsweise die Verletzung eines konkreten Schutzziels, wird dabei in der Wurzel beschrieben. Sämtliche Bedrohungsfaktoren müssen unterhalb dieser Wurzel auftauchen.

Die Verfeinerung von Faktoren wird durch einen Ast ausgedrückt. Dieser beschreibt vom Blatt bis zur Wurzel verschiedene Aspekte des jeweiligen Faktors. Es ist nicht festgelegt, welche Kriterien für die Wahl der Aspekte zugrunde liegen. Möglich wäre beispielsweise die Unterscheidung zwischen Aktionen von außen und innerhalb des Systems oder eine Unterscheidung der benötigten Ressourcen des jeweiligen Schritts.

Das Füllen der Bäume geschieht im Allgemeinen aber manuell. Es existieren grobe Leitfäden zur Erstellung der Bedrohungsbäume, allerdings keine detailierten Konzepte oder Methoden. Quellen dazu sind etwa [Eck02] und [Amo94]. Eine Folge davon ist, dass sich die entstehenden Baumstrukturen bei Betrachtung des selben Problems durch unterschiedliche Personen unterscheiden können.

5.4.2.1 Fehlerbäume

Das Konzept, Bedrohungen bzw. Fehler in einer Hierarchie anzuordnen, um die einzelnen atomaren Schritte für deren Entstehung zu beschreiben, ist nicht neu. Der Ursprung liegt

in der Konstruktion von Fehlerbäumen, die aus dem Bereich der Zuverlässigkeit stammen, im Allgemeinen auch bekannt als „Safety“-Bereich. Die dort schon länger bekannten Fehlerbäume, im Englischen als „Fault-Tree“ bezeichnet, dienen der systematischen Beschreibung von Fehlerzuständen und deren Ursachen. Sie unterstützen die Fehleranalyse beim Design von Systemen. Eine formale Definition der Fehlerbäume findet sich in [VGRH81]. Es scheint deshalb sinnvoll zu sein, die Herkunft dieser Fehlerbäume näher zu untersuchen. Daher werden im Folgenden die wesentlichen Konstruktionselemente dargestellt.

Konstruktionselemente

Syntaktisch werden verschiedene Elemente zur Konstruktion definiert. Der Fehlerbaum beschreibt an seiner Wurzel den eigentlichen Fehler, der durch einen Vorgang entsteht, dessen einzelne Ereignisse in den einzelnen Blättern hin zur Wurzel beschrieben sind.

Symbole der Basis- und Zwischenereignisse

Die Elemente zur Beschreibung von Ereignissen werden „Ereignissymbole“ (engl. event symbols) genannt und können entweder unbedingt auftreten, oder mit internen oder externen Ereignissen verknüft sein. In Tabelle 5.5 werden alle möglichen Ereignistypen erläutert. Im Gegensatz dazu werden mit Zwischenereignissen Events beschrieben, die aus der logischen Zusammensetzung anderer Ereignisse entstehen, wie in Tabelle 5.6 dargestellt ist.

Symbol	Bezeichner	Logische Bedeutung
○	Basisereignis	Ein elementarer, auftretender Fehler, der nicht verfeinert werden muss.
⬭	Abhängiges Ereignis	Spezielle zusätzliche Bedingungen, die einem logischen Gatter zugeordnet werden; hauptsächlich werden diese in Zusammenhang mit dem „priorisierten Und“ ⏢ und der „Sperre“ ⬡ verwendet.
◇	Nicht betrachtetes Ereignis	Ein Ereignis, das aufgrund geringer zu erwartender Konsequenzen oder zu wenig Informationen nicht näher betrachtet wird.
⬠	Externes Ereignis	Damit werden Ereignisse beschrieben, die keine Besonderheit darstellen, etwa die Eingabe eines Benutzers.

Tabelle 5.5: Symbole für Basisereignisse

Symbol	Bezeichner	Logische Bedeutung
	Folgeereignis	Ein Ereignis, das durch die Verknüpfung mit logischen Gattern aus anderen Ereignissen entsteht. Dieses Symbol wird für die Baumknoten verwendet, die keine Blätter sind.

Tabelle 5.6: Zwischenereignisse

Symbole der logischen Gatter

Die Verbindung zwischen einem Elternknoten und seinem Kind wird durch ein logisch verknüpfendes Element dargestellt. Diese werden „Gatter“ (engl. gates) genannt und mit den klassischen Symbolen aus der Schaltungstechnik beschrieben. Für die logischen Gatter sind wahlweise folgende Möglichkeiten zur Beschreibung vorhanden, wie sie in Tabelle 5.7 dargestellt sind.

Transfersymbole

Da die Fehlerbäume in der Praxis sehr groß werden können, wird eine Möglichkeit benötigt, Bäume in Teilbäume aufzuspalten. Dazu existieren die Symbole der Eingangs- und Ausgangsreferenz, wie in Tabelle 5.8 beschrieben wird.

Beachtenswert ist hierbei, dass kein Ausdruck für ein logisches „Nicht“ definiert wird, welches in Erweiterungen der Fehlerbäume meist durch das $\triangle$ Symbol dargestellt wird, wie etwa in [CM03]. Durch Negierung der informellen Beschreibung kann ohne logisches „Nicht“ ausgekommen werden, für exakte Aussagen ist diese Erweiterung allerdings sehr hilfreich.

Fehlerbäume sind der Ursprung von Bedrohungsbäumen, die hier in diesem Abschnitt diskutiert werden. Eine grundlegende Beschreibung für Fehlerbäume ist durch die beschriebene Syntax ausreichend erbracht. Es wäre anzunehmen, dass die Syntax von Threat-Trees der von Fault-Trees sehr ähnlich ist, schließlich ist der Zweck sehr ähnlich, wenn auch aus unterschiedlichen Bereichen kommend. Tatsächlich verwenden Threat-Trees nur eine kleine Menge der Fault-Tree Syntax, obwohl viele der nicht verwendeten Elemente auch für Bedrohungsbäume sinnvoll und nützlich sind.

5.4.2.2 Syntax von Bedrohungsbäumen

Bedrohungsbäume werden beispielsweise in [Amo94] oder [Eck02] beschrieben. Dabei gilt immer, dass die in der Wurzel beschriebene Bedrohung nur dann wirksam sein kann, wenn

Symbol	Bezeichner	Logische Bedeutung
	Und	Wenn alle n Eingänge ($n \geq 2$) „Fehler“ enthalten, ist die Ausgabe „Fehler“.
	Oder	Wenn mindestens ein Eingang „Fehler“ enthält, ist die Ausgabe „Fehler“.
	Exklusiv Oder	Wenn genau ein Eingang „Fehler“ enthält, ist die Ausgabe „Fehler“
	Priorisiertes Und	Wenn alle n Eingänge ($n \geq 2$) „Fehler“ enthalten und diese genau in definierter Reihenfolge auftreten, ist die Ausgabe „Fehler“. Die Reihenfolge wird meistens durch ein „abhängiges Ereignis“ ⬭ rechts des ⏢ Symbols dargestellt.
	Sperre	Der Ausgang einer Sperre liefert „Fehler“ genau dann, wenn der einzige Eingang „Fehler“ enthält und eine zusätzliche Bedingung ebenfalls zutrifft. Diese zusätzliche Bedingung wird als „abhängiges Ereignis“ ⬭ rechts des ⬡ Symbols dargestellt.

Tabelle 5.7: Logische Gatter

alle Faktoren von den Blättern bis zur Wurzel gültig sind.

Es existieren zwei Arten, wie Faktoren voneinander abhängig sein können. Durch eine `UND`-Verknüfung wird ausgedrückt, dass beide Faktoren gemeinsam gültig sein müssen. Eine `ODER`- Verknüpfung beschreibt, dass einer der gegebenen Faktoren der Unterbäume für die Existenz der Bedrohung nötig ist. Das Attribut für die Konjunktion (ein logisches Und) und die Disjunktion (ein logisches Oder) wird dazu am jeweiligen Knoten gespeichert, für dessen Unterbäume die Verknüpfung gelten soll.

Durch die Darstellung in einer Hierarchie gewinnen Bedrohungsbäume prinzipiell einen Vorteil gegenüber anderen Darstellungen, wie etwa den Bedrohungsmatrizen. Die Pfade bis zur Wurzel stellen zusätzlich den Ablauf bzw. die Voraussetzungen für die Wirksamkeit der Bedrohung dar. Wegen dieser zusätzlichen Informationen sind Bedrohungsbäume geeignet für Berechnungen von Risiken und Abwehrmaßnahmen.

Symbol	Bezeichner	Logische Bedeutung
△	Eingangsreferenz	Es wird gekennzeichnet, dass an dieser Stelle ein Teilbaum aus einer anderen Beschreibung eingefügt wird. Das kann einerseits die Darstellung komplexer Baumgraphen durch Aufteilung auf mehrere Seiten bedeuten, oder aber die Wiederverwendung gleichbleibender Teilbäume erleichtern.
△	Ausgangsreferenz	Der dargestellte Teilbaum kann an dieser Stelle als Ganzes an einen anderen Fehlerbaum angefügt werden.

Tabelle 5.8: Transfersymbole

Berechnung von Risiken

Auf Basis entwickelter Bedrohungsbäume können die Risiken für eine spezielle Bedrohung berechnet werden. Das Risiko, dass ein konkreter Angriff aufgrund der Bedrohung durchgeführt wird, wird durch die Teilrisiken der Unterbäume berechnet. Für alle Blätter muss dafür das Risiko des Eintretens des beschriebenen Ereignisses angegeben werden. Die Berechnung kann dann beispielsweise rekursiv beginnend von der Wurzel aus durchgeführt werden.

Ein einfaches Verfahren zur Berechnung des Gesamtrisikos wird hier kurz beschrieben. Für jeden Knoten gilt dabei, dass sich das Gesamtrisiko aus den Teilrisiken der Unterbäume berechnet. Bei einer Disjunktion der Unterbäume wird das höchste Risiko der Unterbäume berücksichtigt, bei einer Konjunktion das Minimum der Risiken der Unterbäume. Das Verfahren funktioniert zum einen durch automatische Berechnung und zum anderen kann es jederzeit an veränderte Umgebungen angepasst werden, in denen sich Teilrisiken verändern.

Auf ähnliche Weise lässt sich ebenso die Eintrittswahrscheinlichkeit für Angriffe auf Basis der modellierten Bedrohungen berechnen oder die möglichen entstehenden Kosten bei der erfolgreichen Durchführung von Angriffen.

Berechnung von Abwehrmaßnahmen

Die Berechnung von Abwehrmaßnahmen anhand der entwickelten Bedrohungsbäume basiert darauf, dass alle notwendigen Voraussetzungen für das Entstehen der Bedrohung in dem Baum beschrieben sind. Aufgrund der Konjunktion oder Disjunktion der einzelnen Bedrohungsfaktoren kann man die Faktoren finden, die man durch die Gegenmaßnahmen

ungültig setzen muss, um die Gesamtbedrohung zu verhindern. Bei Disjunktionen müssen alle Faktoren der Unterbäume verhindert werden, bei Konjunktionen nur ein Faktor. Bei der Suche nach Gegenmaßnahmen wird man deshalb im Baum bewerten, wie aufwändig das jeweilige Verhindern bestimmter Faktoren ist. Schließlich wird man genau die Faktoren verhindern, bei denen die Aufwandssumme möglichst gering und das verhinderte Risiko möglichst hoch ist. Eine genaue Berechnung dieser Auswahl hängt immer vom konkreten betrachteten Fall ab und kann unter Umständen nicht eindeutig bestimmt werden.

Bedrohungsbäume werden für exakte Aussagen so beschrieben, wie folgende Grammatik in Tabelle 5.9 zeigt.

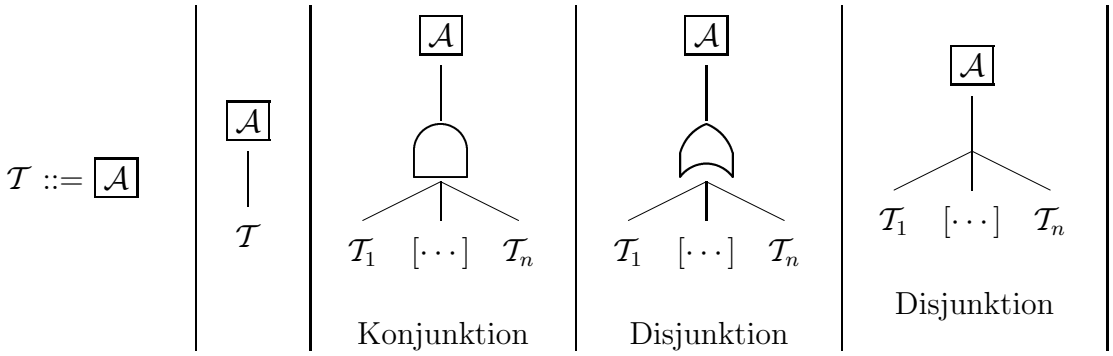

Tabelle 5.9: Herkömmliche Erzeugung von Bedrohungsbäumen

Werden keine Symbole bei der Pfadteilung abgegeben, so handelt es sich üblicherweise um eine Disjunktion. Die Syntax wird häufig so angewendet, wie sie in Tabelle 5.9 angegeben ist. Die Disjunktion wird häufig ohne Symbol gekennzeichnet, da sie im Allgemeinen häufiger vorkommt. Oft können die Werkzeuge die logischen Symbole aus der Schaltungstechnik nicht darstellen, so dass die entsprechende Verknüpfung mit dem Begriff `UND` und `ODER` bezeichnet wird. Dadurch wird eine intuitive Darstellung mit den besprochenen Symbolen behindert.

Sehr beachtenswert ist die Tatsache, dass die meisten Symbole der Fehlerbäume nicht vorhanden sind. Es ist aber denkbar, dass Symbole wie ⏢ (Exklusiv Oder) und ein ⏢ (Priorisiertes Und) auch bei Bedrohungsbäumen Anwendung finden. Durch ein ⏢ lässt sich darstellen, dass genau ein Faktor aller Unterfaktoren eine Bedrohung wirksam macht, falls diese Faktoren sich gegenseitig ausschließen können. Mit einem ⏢ lassen sich zeitliche Abfolgen von Faktoren beschreiben, falls diese voneinander abhängig sind. Ebenso könnten Bedrohungsbäume, die in der Praxis oft unhandlich groß werden, durch die Symbole △ und △ in einzelne Teilbäume aufteilen. Es ist jedoch keine Arbeit bekannt, die die Symbolik der Fehlerbäume für Bedrohungsbäume verwendet. Die Werkzeuge, die im Rahmen dieser Arbeit entstanden sind und in Kapitel 7 vorgestellt und in Kapitel 8 angewendet werden, sind dazu in der Lage. Der semantische Nutzen dieser Erweiterungen liegt vor allem bei der Be-

rechnung innerhalb der Bedrohungsbäume. Dazu zählen die Berechnung von vorhandenen Risiken und damit die Entwicklung von Abwehrmaßnahmen gegen wichtige Bedrohungen.

Bewertung

Bedrohungsbäume spielen heute in der Praxis eine große Rolle. Entscheidender Vorteil ist die leichte Verständlichkeit der erzeugten Beschreibung über die existierenden Bedrohungen. Das bedeutet einerseits, dass Entwickler mit vernünftigem Aufwand die entsprechenden Gegenmaßnahmen einsetzen können. Andererseits kann ein Benutzer die Bedrohungsbäume lesen und daraus Vertrauen in die Sicherheitsarchitektur gewinnen. Allerdings hängt die Qualität der Bedrohungsbäume entscheidend von deren Entwicklungsstrategie ab. Es ist nur sehr schwer möglich, die Qualität eines Bedrohungsbaums zu bestimmen, insbesondere wie vollständig er ist bzw. welche Bedrohungen nicht berücksichtigt wurden. Werkzeugunterstützung zur Erzeugung und Verwaltung von Bedrohungsbäumen existiert nicht. Teilbäume sind wiederverwendbar, es kommt bei der Anwendung auch vor, dass identische Teilbäume an verschiedenen Stellen des Bedrohungsbaums auftauchen. Damit ergibt sich für eine Bewertung nach Tabelle 5.10. Durch Verbesserungen der Vollständigkeit der Analyseergebnisse und einer nachvollziehbaren Strukturierung des Prozesses wären Bedrohungsbäume ein sehr wirkungsvolles Verfahren zur Sicherheitsanalyse.

Kriterium	Bewertung
Vollständigkeit	nicht überprüfbar (−−)
Reproduzierbarkeit	möglich, Darstellung verständlich (++)
Zeittransparenz	möglich, Bäume erweiterbar (+)
Strukturierung des Analyseprozesses	keine Struktur für den Prozess (−)
Strukturierung des Ergebnisses	Struktur der Ergebnisse durch Darstellung (+)
Wiederverwendbarkeit	möglich, auch für Teilbäume (++)
Angepasster Aufwand der Analyse	Analyse von Teilsystemen möglich (+)
Werkzeugunterstützung	Verwaltung und Erstellung möglich, keine Tools bisher vorhanden (+)

Tabelle 5.10: Bewertung der Bedrohungsbäume

5.4.3 Octave

Stellvertretend für alle Methoden, die nicht nur die technische Sicherheit der Systeme, sondern zusätzlich auch die organisatorische Sicherheit betrachten, wird an dieser Stelle Octave (Operationally Critical Threat, Asset, and Vulnerability Evaluation) vorgestellt. Nähere Informationen sind dazu auch in [Oct04] zu finden. Octave stellt einen risikoorientierten Gesamtansatz dar, dessen Vorgehen durch dokumentierte Teilprozesse beschrieben ist. Der Einsatz für Octave ist vorzugsweise im Bereich mittlerer bis großer Unternehmen gedacht. Ziel der Methode ist es, ein Gesamtbild über die möglichen Bedrohungen aller Werte des Unternehmens zu finden, die in Zusammenhang mit Informationen stehen. Da Zusammenhänge zwischen der Bewertung komplexer, großer Unternehmen und komplexen Rechensystemen bestehen, wird Octave hier kurz angesprochen.

Mit Werten werden alle informationsbezogenen Güter bezeichnet, wie etwa:

- *Informationen:* Dokumente in elektronischer oder Papierform, die unternehmenswichtig sind.
- *Systeme:* zur Verwaltung von Informationen; diese bestehen im Octave-Kontext aus einer Kombination aus Software, Hardware und Informationen.
- *Software:* Anwendungen und Dienste, die mit Informationen in Wechselwirkung treten.
- *Hardware:* Geräte aus dem informationstechnischen Bereich.
- *Personen:* die Fähigkeiten, Wissen und Erfahrung besitzen, die nicht leicht ersetzbar sind.

Dazu werden die drei Aspekte betriebliche Risiken, Informationssicherheit und die zur Verfügung stehende Technologie betrachtet. Als prinzipielle Gefährdungen werden der Verlust der Vertraulichkeit, Verfügbarkeit und Integrität angesehen. Im Unterschied zu anderen Evaluationsmethoden wird bei Octave nicht ein einzelnes System, sondern eine gesamte Organisation bewertet, wie etwa eine Firma. Der Fokus liegt dabei nicht auf der möglichen Technologie, sondern im Einsatz bestehender Verfahren.

Das Vorgehen bei Octave besteht aus drei Phasen, die im Folgenden kurz mit den geforderten Prozessen erläutert werden. Jede Phase besitzt einige Prozesse, die dabei kurz erwähnt werden.

1. *Bedrohungsprofile entwickeln: (Prozesse 1-4)* In diesem Schritt werden alle wesentlichen Güter dokumentiert und beschrieben, was zu deren Schutz bereits getan wird. Für die wichtigsten informationsbezogenen Güter werden die Schutzanforderungen aufgestellt und ein Bedrohungsprofil dafür entwickelt. Prozesse 1-3 identifizieren das

vorhandene Wissen aus verschiedenen Geschäftsbereichen. In Prozess 4 werden etwa fünf wichtige Güter aus jedem Wissensbereich ausgewählt und Bedrohungsprofile erstellt. Wie die Bedrohungen gefunden werden, ist nicht festgelegt.

2. *Schwachstellen der Infrastrukturen beschreiben: (Prozesse 5 und 6)* Dabei werden wichtige Angriffswege ermittelt, wobei Rechnernetze dabei im Vordergrund stehen. Daneben werden die Netzwerkkomponenten berücksichtigt und eine Liste aller möglichen Bedrohungen erstellt. In den Prozessen 5 und 6 werden die an den Bedrohungen beteiligten Komponenten identifiziert und bewertet.

3. *Sicherheitsstrategien entwickeln: (Prozesse 7 und 8)* Auf Basis der beschriebenen Bedrohungen werden in Prozess 7 Kriterien für die Risikobewertung entwickelt. Diese Kriterien werden durch die festen Stufen hohes, mittleres und geringes Risiko bewertet. Prozess 8 beschreibt die Arbeit eines Teams bei der Suche nach den entsprechenden Sicherheitsstrategien. Darunter fallen nicht nur technische Lösungen, sondern ebenso organisatorische Maßnahmen.

Bewertung

Die einzelnen drei Phasen und die 8 Prozesse werden ausführlich beschrieben. Dennoch sind bestimmte Vorgehensweise nicht erklärt, wie etwa das Finden der Bedrohungen auf bestimmte Güter. Octave ist trotzdem ein interessanter Gesamtansatz zur Verringerung der Gefährdungen von Unternehmen. Die Prozesse sind nicht für die Analyse von einzelnen Systemen ausgelegt, das risikoorientierte strukturierte Vorgehen ermöglicht jedoch das Nachvollziehen der Ergebnisse. Die einzelnen Bewertungen sind in Tabelle 5.11 zusammengefasst.

5.4.4 STRIDE-Methode

Die Firma Microsoft hat im Jahr 2002 eine große Sicherheit-Offensive gestartet, die als „Security Push“ bezeichnet wird. Neben Vorgängen wie einem veränderten Patch-Management und Fehlerverwaltung wurde auch die Software-Entwicklung für abgesicherte Systeme verbessert.

Wie in [HL03] nachzulesen ist, wird bei der Software-Entwicklung eine strukturierte Methode zur Bedrohungsmodellierung (engl. Threat Modeling) durchgeführt. Neben dem eigentlichen Ziel werden noch indirekte vorteilhafte Ergebnisse genannt, die sich bei der Bedrohungsmodellierung ergeben.

Durch das Erstellen der Bedrohungsmodelle wird die Anwendung besser verstanden, denn bei der Modellierung wird die Anwendung, bzw. der Quelltext, unter dem Sicherheitsaspekt betrachtet und analysiert. Durch diese unterschiedliche Betrachtungsweise werden funktionale Eigenschaften oft deutlicher. Daneben werden bei der Analyse des Codes oft

Kriterium	Bewertung
Vollständigkeit	nur Teile der IT-Sicherheit werden betrachtet (−−)
Reproduzierbarkeit	möglich, aber sehr aufwändig (+)
Zeittransparenz	erreichbar (+)
Strukturierung des Analyseprozesses	aufwändiger Analyseprozess vorgegeben (++)
Strukturierung des Ergebnisses	Struktur der Ergebnisse durch spezielle Darstellungen (+)
Wiederverwendbarkeit	eingeschränkt möglich (+)
Angepasster Aufwand der Analyse	stets sehr hoher Aufwand (−)
Werkzeugunterstützung	nur wenige, kommerzielle Werkzeuge vorhanden (−)

Tabelle 5.11: Bewertung von OCTAVE

auch Fehler entdeckt, die nicht unbedingt sicherheitsrelevant sind. Dazu zählen nicht nur Implementierungsfehler, sondern ebenso logische Fehler, die auf hohem Abstraktionsniveau gemacht wurden. Schließlich sind Bedrohungsmodelle nach der STRIDE-Methode ebenso für den Softwaretest nutzbar und hilfreich. Die Tester können nach [HL03] diese Modelle nutzen um etwa neue Testtools zu erstellen.

Die groben Schritte für das Verfahren sind folgendermaßen angegeben:

1. Zusammenstellung eines Bedrohungsanalyse-Teams
2. Zerlegung und Strukturierung der Anwendung
3. Bestimmung der wesentlichen Bedrohungen auf das System
4. Einordnung der Bedrohungen durch eine Risikoanalyse
5. Auswahl der entsprechenden Reaktion auf Bedrohungen
6. Verfahren zur Entschärfung der Bedrohungen diskutieren
7. Auswahl der wirksamen Mechanismen zur Umsetzung der gewählten Methoden

In den folgenden Abschnitten werden diese einzelnen Phasen der Methode genauer beschrieben.

5.4.4.1 Zusammenstellung eines Bedrohungsanalyse-Teams:

Wesentlich für eine erfolgreiche Analyse sind Teammitglieder, die sowohl das Systemdesign auf hoher Ebene als auch die Implementierung kennen und Wissen über bekannte Bedrohungen mitbringen und verwenden können. Aus den einzelnen Disziplinen des Software-Engineerings, wie Designentwicklung, Implementation, Programmtest und Dokumentation, soll jeweils ein Mitglied teilnehmen.

5.4.4.2 Zerlegung und Strukturierung der Anwendung:

Die Zerlegung und Strukturierung der zu untersuchenden Anwendung wird mit Hilfe von Datenflussdiagrammen (engl. Data Flow Diagram oder DFD) durchgeführt. Sie stellen einen formalen Rahmen für das zugrundeliegende Modell der STRIDE-Methode dar.

Datenflussdiagramm — DFD

In der Abbildung 5.1 werden die wichtigsten syntaktischen Elemente von Datenflussdiagrammen dargestellt.

Neben Datenflussdiagrammen existieren noch weitere Diagramme, die für eine derartige Beschreibung nützlich erscheinen. Datenflussdiagramme stellen nur eine Möglichkeit dar, das zu analysierende System darzustellen. Es ist auch denkbar, andere Diagramme einzusetzen und wird von den STRIDE-Entwicklern auch explizit so vorgeschlagen. Erwähnenswert ist vor allem eine Diagrammart aus dem UML-Katalog, die Activity Diagramme. Diese beschreiben nicht den Datenfluss, sondern den Kontrollfluss innerhalb eines Systems und liefern demnach ein ähnliches, nicht aber identisches Konzept. Eine Diskussion darüber folgt in der Bewertung zur STRIDE-Methode.

In der ersten Phase wird zunächst ein sehr allgemeines Diagramm der Datenflüsse gezeichnet. Im wesentlichen sollen dabei die Schnittstellen zwischen den vertraulichen und nicht vertraulichen Elementen auftauchen, aber keine Datenspeicher. Von diesem Diagramm ausgehend werden dann Verfeinerungen der einzelnen Elemente erstellt. Nach dem methodischen Vorgehen sind keine exakten Abläufe für den Schritt der Verfeinerung beschrieben, es wird lediglich empfohlen, dass höchstens drei bis vier Tiefen dargestellt werden. Eine größere Genauigkeit wäre zu aufwändig und der Nutzen zu gering.

5.4.4.3 Bestimmung der wesentlichen Bedrohungen auf das System:

Die einzelnen Elemente der entwickelten Diagramme werden jetzt schrittweise auf bestimmte Bedrohungen hin untersucht. Dazu wurden Bedrohungsklassen definiert, die den Methodennamen STRIDE bestimmen und in einem folgenden Abschnitt beschrieben werden.

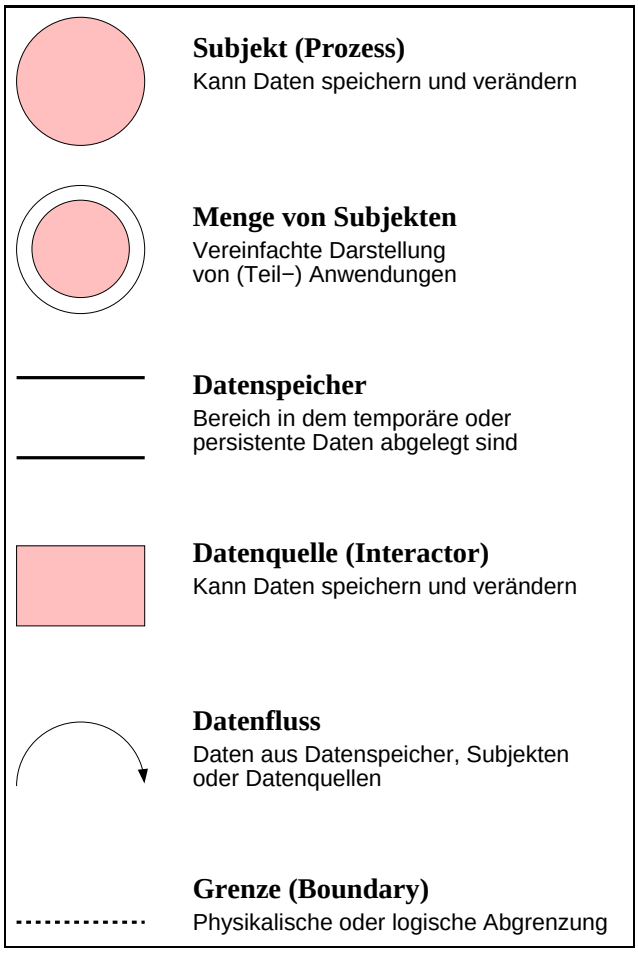

Abbildung 5.1: Syntax der Datenflussdiagramme (DFD)

Im Wesentlichen sind Fragen von Interesse, ob unauthorisierte Subjekte vertrauliche Daten lesen oder verändern können, ob irgendjemand Dienste für andere Benutzer behindern kann und ob jemand auf spezielle Art höhere Rechte als die zugestandenen erlangen kann.

Weil diese Liste der Fragen weder vollständig, noch für jeden Anwendungsfall geeignet ist, wurden zur Unterstützung der Analyse folgende Bedrohungsklassen entwickelt, deren Anfangsbuchstaben den Namen der STRIDE-Methode beschreibt.

Spoofing identity
„Spoofing identity“ sind Arten der Bedrohungen, die das fälschliche Vorgeben einer anderen Identität beschreiben, beispielsweise während des Vorgangs der Authentifizierung.

Tampering with data
Die „Tampering“-Bedrohung bedeutet das unerlaubte Verändern von schützenswerten Daten.

Repudiation
Mit „Repudiation“ werden die Bedrohungen bezeichnet, die ein Abstreiten von durchgeführten Aktionen darstellen.

Information disclosure
„Information disclosure“ beschreibt die unerlaubte Weitergabe von vertrauenswürdigen Daten an Dritte, die zur Nutzung der Daten nicht berechtigt sind.

Denial of service
Die Behinderung eines Dienstes für einen rechtmäßigen Benutzer wird mit „Denial of service“ bezeichnet.

Elevation of privilege
Kann ein Angreifer unerlaubt priviligierte Rechte erhalten und somit ausreichend Einfluss auf das laufende System bekommen, um es auszunutzen oder zu zerstören, so wird das „Elevation of privilege“ genannt.

Diese einzelnen STRIDE-Bedrohungsklassen werden auf die einzelnen entwickelten Modelle angewendet. Dazu wird jeder Datenfluss auf jede der sechs Bedrohungsarten hin untersucht. Die Beschreibung der Methode gibt keinerlei nährere Hinweise für eine detailiertere und strukturiertere Vorgehensweise, so dass es wesentlich auf die Qualität der Datenflussdiagramme sowie die Qualifikation der untersuchenden Personen ankommt. Insbesondere sind somit keine reproduzierbaren Ergebnisse zu erwarten, wie im Bewertungsabschnitt weiter unten diskutiert wird.

Die einzelnen STRIDE-Bedrohungsklassen sind allerdings nicht unabhängig voneinander. Deshalb werden zur Repräsentation der erkannten Bedrohungen Bedrohungsbäume eingesetzt, die sowohl grafisch, als auch listenartig verwendet werden. Im Gegensatz zu der Syntax, wie sie von [Amo94] eingeführt wurde, werden die Bedrohungsbäume hier in einer erweiterten Form eingesetzt, die auch in der Abbildung 5.2 dargestellt wird.

Eine Erweiterung der Baumdarstellung ist die Verwendung von zwei Verbindungsarten zwischen einzelnen Knoten. Neben der gängigen Verbindung zweier Knoten durch einen einfachen Strich werden auch gestrichelte Linien für Angriffspfade verwendet, deren Auftreten von vorneherein unwahrscheinlich erscheint.

Zudem wird bei den Blättern, deren Pfade zur Wurzel gestrichelte Linien enthalten, eine Beschreibung in Form eines Kreises dazugefügt. Der Text beschreibt jeweils den Grund, warum dieser Angriffspfad als weniger wahrscheinlich eingestuft wird.

Allerdings sollen diese Erweiterungen, vor allem die Erweiterungen durch Risikoanalyse (Kreise), erst später geschehen, nicht zur Bedrohungsanalyse. Genauer wird jedoch das Vorgehen dazu nicht beschrieben.

Neben dem Namen der gefundenen Bedrohung und einem beschreibenden Bedrohungsbaum sollen zudem noch zusätzliche Fakten aufgeschrieben werden, die in der folgenden Tabelle

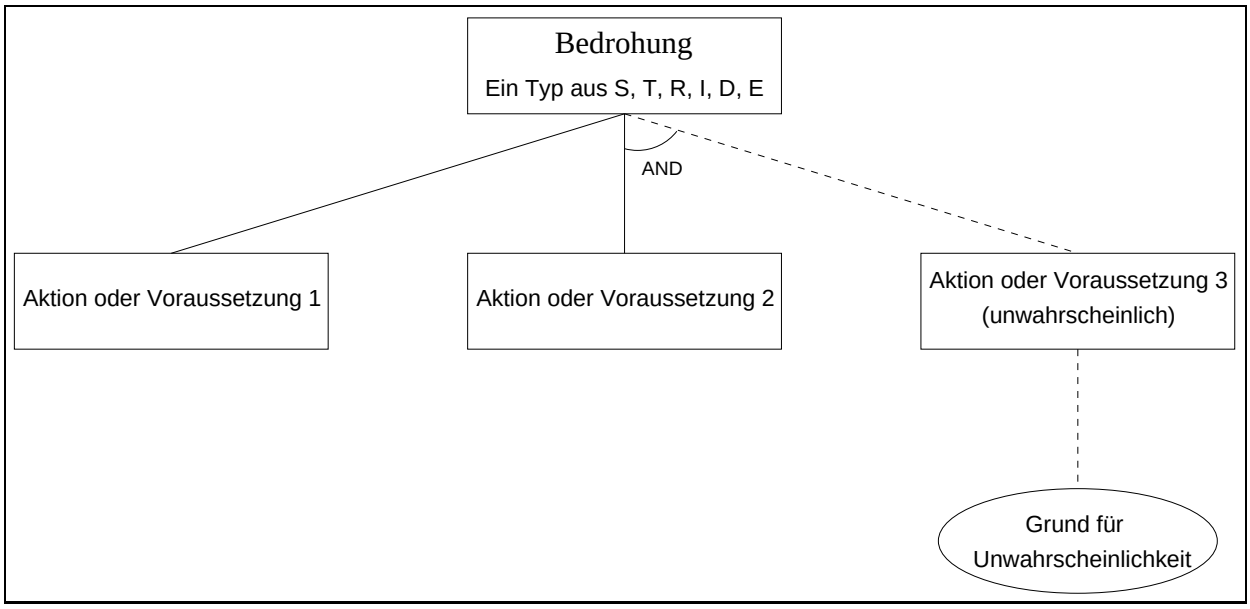

Abbildung 5.2: Syntax der erweiterten Bedrohungsbäume

beschrieben werden.

5.4.4.4 Einordnung der Bedrohungen durch eine Risikoanalyse:

Nachdem die Liste der Bedrohungen erstellt wurde, müssen sie durch Risikoanalyse bewertet werden. Wichtig ist dabei, dass die Analyse homogen und reproduzierbar ist, welche der zahlreichen Methoden tatsächlich eingesetzt wird, ist bei STRIDE nicht festgelegt.

Eine sehr einfache Methode ist die Anwendung der allgemeinen Formel zur Risikoberechnung:

$Risk_{CO} = Criticality * Likelihood\ of\ Occurence$

Die Höhe des Risikos ist proportional zu $Risk_{CO}$. Meist werden für die beiden Größen „Criticality“ (Gefährdungsmaß) und „Likelihood of Occurence“ (Eintrittswahrscheinlichkeit) ein Wertebereich von $[1 \ldots 10]$ festgelegt. Das höchste Risiko wäre demnach durch die Bewertung mit $Risk_{CO} = 100$ gegeben.

Eine von den STRIDE-Entwicklern angegebene Methode zur Risikobewertung ist die $Risk_{DREAD}$-Methode. `DREAD` steht dabei als Abkürzung für die ersten Buchstaben folgender Gefährdungsklassen. Jede Klasse wird mit einem Wert aus dem Intervall $[1 \ldots 10]$ bewertet, wobei 10 die höchste Gefährdung darstellt.

Bedrohungsattribut	Beschreibung
Name	Einfacher, beschreibender Name für die Bedrohung
Ziel der Bedrohung	Welcher Teil der Anwendung wird bedroht? Werden mehrere Teile bedroht?
Bedrohungsklasse(n)	In welche der sechs STRIDE-Klassen fällt diese Bedrohung? Werden evtl. mehrere Klassen angesprochen?
Risiko des Auftretens	Ergebnisse einer bestimmten Methode zur Risikoanalyse. Die STRIDE-Methode ist nicht auf eine Risikoanalyse-Methode festgelegt.
Bedrohungsbaum	Überschaubare Bäume, die den Weg oder die Wege des Angriffs erläutern können.
Verfahren zur Abwehr (optional)	Wie könnte man die Bedrohung abwehren? Welche Mechanismen oder Methoden könnten eingesetzt werden? Wie groß ist der Aufwand zur Abwehr der Bedrohung?
Aktuelle Gefahrensituation	Wurde bereits etwas gegen die Bedrohung unternommen oder werden schon andere Verfahren eingesetzt, die gegen die Bedrohung wirksam sein könnten?
Fehlernummer (Bug-Tracking)	Wenn Mechanismen zur Fehlerbearbeitung eingesetzt werden, wie beispielsweise Systeme zur Fehlerbearbeitung, sollen Referenzen dazu dokumentiert werden.

Tabelle 5.12: Bewertung der STRIDE-Methode

Damage potential
Das stellt ein Maß für den möglichen zu erwartenden Schaden dar.

Reproducibility
Damit wird bewertet, wie leicht dieser Angriff, auch von einer anderen Person als dem ursprünglichen Angreifer, erfolgreich wiederholt werden kann.

Exploitability
Damit wird bewertet, wie einfach ein Angriff durchgeführt werden kann und welcher Aufwand, auch der finanzielle, dafür notwendig wäre.

Affected user
Dies stellt die ungefähre Anzahl an Systembenutzern dar, die durch einen durchgeführten Angriff betroffen wären.

Discoverability
Hierbei wird bewertet, wie wahrscheinlich ein Bekanntwerden der Bedrohung und des entsprechenden Angriffs ist. Dabei wird in der Literatur angegeben, dass das wohl sehr schwer zu bewerten sein und die Authoren geben an, dass sie selbst diesen Wert stets mit 10 Punkten bewerten und bei der Risikoanalyse den anderen Werte mehr Bedeutung zuteilen.

Selbstverständlich ist das nur ein möglicher Ansatz, der im Rahmen der STRIDE-Methode zur Risikoanalyse genannt wird. Es existieren weit genauere, systematischere und formalere Ansätze dazu, etwa aus den Bereichen der Finanz- und Wirtschaftmathematik. Risikoanalyse wird in dieser Arbeit allerdings nicht näher betrachtet, da der Schwerpunkt auf der Analyse der Bedrohungen liegt.

5.4.4.5 Auswahl der entsprechenden Reaktion auf Bedrohungen:

Es existieren vier wichtige Möglichkeiten, wie auf erkannte Bedrohungen reagiert werden kann.

Naheliegendste Möglichkeit ist auf die Bedrohung zunächst überhaupt nicht zu reagieren. Dadurch wird die entsprechende Angriffsmöglichkeit evtl. auch nicht bekannt und das Problem kann in eine der folgenden Systemversionen behoben werden. Der Benutzer wird davon im Idealfall nichts bemerken und kein Angriff wird aus der Bedrohung hervorgehen. Normalerweise ist diese Reaktionsart jedoch die Schlechteste, so dass eine der drei folgenden Arten gewählt wird.

Eine anderer Weg ist der, den Benutzer bei der Verwendung der entsprechenden Systemteile vor der Bedrohung zu warnen. Obwohl diese Möglichkeit bei der STRIDE-Methode vorgeschlagen wird, gibt es doch erhebliche Probleme damit. Erstens muss die Bedrohung auf ein evtl. entfernbares Teilsystem abgrenzbar sein. Nur dann macht die Warnung für den Benutzer soweit Sinn, dass er das Teilsystem abschalten kann. Zudem muss der Benutzer entsprechendes Wissen besitzen, um über die Konsequenzen der Bedrohung nachdenken zu können.

Wenn der Systemteil abgegrenzt werden kann, der die Ursache für die Bedrohung darstellt, kann vor dem Betrieb des Systems auch entschieden werden, dass der entsprechende Systemteil entfernt wird.

Letzte Möglichkeit ist die Verhinderung und Vermeidung der Bedrohung durch entsprechende Gegenmaßnahmen. Der Aufwand für deren Realisierung kann hoch sein. Wird die Bedrohung erst zum Ende der Entwicklung entdeckt oder beim Systembetrieb, so sind die wichtige Entwicklungsschritte, wie etwa der Softwaretest, erneut durchzuführen. Auch wenn man mit den ersten genannten drei Möglichkeiten diesen Aufwand verhindern kann, wird man spätestens bei der Weiterentwicklung über die Realisierung entsprechender Gegenmaßnahmen kümmern müssen.

5.4.4.6 Verfahren zur Entschärfung der Bedrohungen diskutieren:

In der Dokumentation zur STRIDE-Methode werden für alle 6 Bedrohungsklassen einige Gegenmaßnahmen aufgelistet. Es wird empfohlen, diese gegen erkannte Bedrohungen zu verwenden. Diese sehr eingeschränkte Aufzählung bringt im Vergleich zu Kapitel 2 keine neuen Erkenntnisse und wird deshalb hier nicht näher erläutert.

5.4.4.7 Auswahl der wirksamen Mechanismen zur Umsetzung der gewählten Methoden:

Ebenso wie die Diskussion über die anwendbaren Verfahren im vorigen Abschnitt findet die Auswahl auf Basis der beschriebenen Liste der Gegenmaßnahmen statt. Dabei wird nicht näher auf die Integration der einzelnen Verfahren eingegangen.

5.4.4.8 Bewertung

Im Fokus der Entwickler der STRIDE-Methode stehen die Daten, nicht die Anwendung. Vorteilhaft bei der datenorientierten Sicht ist, dass Schutzziele wie etwa Vertraulichkeit und Integrität direkt am Datenfluss geprüft werden können.

Bestimmte Angriffe, etwa Angriffe auf die Verfügbarkeit, sind allerdings dabei nur schlecht zu modellieren. Bei einem Denial-of-Service-Angriff etwa müsste zumindest der Ablauf und die Zeit im Diagramm beschrieben sein, um ihn darzustellen.

Insgesamt liefert die Methode eine gute Möglichkeit, Bedrohungen zu suchen und zu finden, die an Datenflüssen sichtbar sind. Es hängt starkt vom entwickelten Datenflussmodell ab, welche Bedrohungen übersehen werden können. Das Vorgehen bei der Bedrohungsanalyse ist grob vorgegeben. Die Modellierungsart und die eigentliche Suche der Bedrohungen geschehen jedoch im Wesentlichen durch die Kenntnis der Entwickler. Das Dokumentieren der gefundenen Bedrohungen in Bedrohungsbäumen ermöglicht eine Strukturierung der Ergebnisses und damit sowohl Verständlichkeit, als auch die Möglichkeit zur Wiederverwendung von Teilbäumen. Es existieren noch keine bekannten Werkzeuge für die STRIDE-Methode. Die Bewertung findet sich in Tabelle 5.13.

Kriterium	Bewertung
Vollständigkeit	nur schwer erreichbar (−)
Reproduzierbarkeit	je nach DFD-Modell und Bedrohungsbäumen möglich (+)
Zeittransparenz	erreichbar (+)
Strukturierung des Analyseprozesses	Struktur vorhanden, nur Bedrohungen werden untersucht (+)
Strukturierung des Ergebnisses	Struktur der Ergebnisse durch Bedrohungsbäume (++)
Wiederverwendbarkeit	eingeschränkt möglich für Teilbäume (+)
Angepasster Aufwand der Analyse	je nach Modell (+)
Werkzeugunterstützung	existiert nicht (−−)

Tabelle 5.13: Bewertung der STRIDE-Methode

5.5 Fazit der Bewertung bestehender Methoden

Alle vorgestellten Verfahren zielen nur auf eine Teilanalyse der gesamten Sicherheitsanalyse ab. Ein integriertes Verfahren zur Schwachstellen-, Angriffs- und Bedrohungsanalyse existiert nicht. Die Verfahren arbeiten stets auf Modellen, deren Entwicklung manuell und fehlerbehaftet geschieht. Es werden keine Prozesse zur Untersuchung der Modelle angegeben, die Analyse des Modells geschieht ohne Leitfaden. Darstellungen, wie die Bedrohungsbäume zur Dokumentation aller erkannten Bedrohungen, sind sehr hilfreich. Allerdings ist deren Erstellung nur schlecht nachvollziehbar, da die Struktur der Bäume jedesmal von Neuem entwickelt wird. Die diskutierten Verfahren dienen als Ausgangspunkt für die im folgenden Kapitel entwickelte Analysemethode. Folglich muss eine integrierte Sicherheitsanalyse aus den Teilen Schwachstellen-, Angriffs- und Bedrohungsanalyse bestehen und den Analyseprozess genau beschreiben. Eine Methode dazu wird im folgenden Teil dieser Arbeit entwickelt.

Teil II

Synthese

Kapitel 6

Integrierte Sicherheitsanalyse

In den vorherigen Kapiteln wurden zunächst die Grundlagen über Bedrohungen, Angriffe und Schwachstellen im Bereich der Sicherheit diskutiert und Definitionen und Zusammenhänge dargestellt. Neben einem Überblick über bekannte Gegenmaßnahmen wurden alle nennenswerten Ansätze zur Sicherheitsanalyse beschrieben. In diesem Kapitel wird das entwickelte Verfahren zur Konstruktion von Bedrohungsbäumen behandelt. Dieses Verfahren wird bei der Entwicklung der Betriebssystemteile in Kapitel 8 angewendet. Das angegebene Verfahren nutzt die Bewertungen der bestehenden Ansätze zur Sicherheitsanalyse und fasst deren positiven Eigenschaften in einer integrierten Sicherheitsanalyse zusammen.

Zunächst werden das gegebene Wissen bei einer Sicherheitsanalyse und die Voraussetzungen bei der Entwicklung einer integrierten Sicherheitsanalyse dargelegt. Gesucht ist ein Verfahren zur integrierten Sicherheitsanalyse, die die Ergebnisse strukturiert darstellt. Der zentrale Ansatz der vorgestellten Lösung ist die Darstellung aller Bedrohungen, Angriffe und Schwachstellen in einer Struktur. Diese speziellen Bedrohungsbäume werden anschließend genau definiert und stellen eine Erweiterung der bekannten Bedrohungsbäume dar.

6.1 Ziel der integrierten Sicherheitsanalyse

Das Ziel dieses Abschnittes ist es, einen Zusammenhang zwischen Bedrohungen, Angriffswegen und Schwachstellen herzustellen. Dazu wird skizziert, welches Wissen über das System bei einer Sicherheitsanalyse im Allgemeinen vorhanden ist. Dieses Wissen wird als Basis zur integrierten Sicherheitsanalyse genutzt, die aus einer Kombination der Schwachstellen-, Angriffs- und Bedrohungsanalyse besteht.

6.1.1 Grundlagen

Bei der Entwicklung der Sicherheitsanalyse waren gewisse Voraussetzungen gegeben. Diese werden dargestellt und anschließend wird kurz beschrieben, welches Ziel die Methode verfolgt.

6.1.1.1 Voraussetzungen

Es wird zunächst davon ausgegangen, dass die geforderten funktionalen Eigenschaften des Systems bezogen auf die Anforderungen eingehalten werden. Vor allem das Schutzziel der Verfügbarkeit könnte sonst durch Programmfehler leicht verletzt werden. Diese Überprüfung ist nicht Gegenstand dieser Arbeit. Auf funktionalen Fehlern basierende Bedrohungen können aber dennoch durch das vorgestellte Verfahren dargestellt werden. Es existieren vielfältige Arbeiten zur Einhaltung funktionaler Eigenschaften in Programmen, die aber aus einem anderen Teil der Informatik stammen und deshalb nicht näher betrachtet werden. An dieser Stellen soll aber auf den dabei sichtbar werdenden Zusammenhang der beiden Sicherheitsbereiche der Informatik aufmerksam gemacht werden. Auch wenn sich beide Bereiche, die Safety und die Security teils völlig anderer Methoden bedienen, scheinen doch beide bei der Entwicklung abgesicherter Systeme notwendig zu sein.

Die Annahme bedeutet insbesondere, dass bei der Sicherheitsanalyse eines realisierten oder zum Teil realisierten Systems vorhandene Sicherheitsmechanismen zunächst als korrekt funktionierend angesehen werden. Wenn beispielsweise ein Authentifikationsmechanismus in einem System integriert ist, so wird bei der Sicherheitsanalyse des Gesamtsystems davon ausgegangen, dass er funktional korrekt arbeitet. Der Vorgang der Authentifikation liefert dann bei spezifizierten Eingaben stets die richtige Antwort auf die Frage nach der Identität des Kommunikationspartners.

Die folgenden zwei Möglichkeiten helfen jedoch, mit dieser Einschränkung zurechtzukommen.

1. Es ist möglich, bekannte Bedrohungen des Authentifikationssystems im Ergebnis der Sicherheitsanalyse darzustellen. Im Bedrohungsbaum kann dazu ein entsprechender Pfad eingefügt werden, was später noch beschrieben wird. Dazu zählen auch Bedrohungen, die aufgrund funktionaler Mängel vorhanden sind.

2. Bei der Schwachstellensuche, die gegebenenfalls auch automatisch auf Basis des Quelltextes durchgeführt werden kann, können funktionale Fehler in der Implementierung gefunden werden. Das stellt aber kein fundiertes Verfahren zur Verifikation des Systems dar, sondern basiert auf Wissen bekannter Schwachstellen und Wissen der analysierenden Personen.

6.1.1.2 Gegeben

Die Kenntnisse, die man zur Durchführung einer Sicherheitsanalyse braucht, können in unterschiedlicher Form gegeben sein. Die Verfügbarkeit der Quelltextes ist die konkreteste Informationsform. Damit lassen sich sowohl Schwachstellen im Code finden, als auch die Funktion und Wirkung bekannter Angriffe nachvollziehen. Ebenso kann das Wissen durch die Benutzung des Systems vorhanden sein. Auch diese Informationen können zum Finden von Bedrohungen ausreichend sein. Allerdings kann keine Schwachstellenanalyse am Quelltext durchgeführt werden, so dass auf viele Schwachstellen und mögliche Angriffe nur durch Ausprobieren getesten werden kann. Die schwächste Form der Informationen über ein System kann durch allgemeine Dokumentation erhalten werden. Dazu zählen Architekturbeschreibungen ebenso wie Bedienungsanleitungen. Die Schwachstellen eines Systems werden meist durch fehlerhafte Implementierungen erzeugt und sind deshalb in Dokumentationen nicht festgehalten.

Das Wissen über das System wird durch Kenntnisse seines dynamischen Verhaltens zur Laufzeit erweitert. Dieses kann beispielsweise aus der statischen Beschreibung, wie etwa dem Quelltext, gewonnen werden. Dazu zählen Systemverhalten wie die Antwortzeiten bei der Bedienung oder die Dauer von Berechnungen. Es sind Bedrohungen denkbar, die beispielsweise durch Auslastung des zugeteilten Prozessors oder dem Belegen von Speicher entstehen. Daneben können Angreifer durch reine Beobachtung des Zeitverhaltens beispielsweise bei Berechnungen oder bei der Kommunikation auf die verwendeten Verfahren schließen. Es existieren sogar Angriffe, die durch diese Beobachtungen auf das verwendete, private Schlüsselmaterial bei kryptografischen Berechnungen schließen können. Die dynamische Veränderung des Systems wird hier derart verstanden, dass deren Entwicklung aus dem Quelltext ablesbar ist und die Systemkenntnisse deshalb auch die dynamischen Systementwicklungsaspekte beinhalten.

Ebenso kann Wissen über das System aufgrund von Kenntnissen über das Einsatzgebiet und die Art der Benutzung des Systems entstehen. Bei der Bewertung bestimmter Bedrohungen spielen Einsatzgebiet und Art der Benutzung eine wesentliche Rolle. Wird ein System beispielsweise in einem abgesicherten Raum betrieben, so sind Risiken durch Diebstahl oder Ausspionieren von Passwörtern bei der Eingabe sehr gering.

Meist sind konkrete Kenntnisse über Schwachstellen, mögliche Angriffe und bekannte Bedrohungen vorhanden. Diese stehen zunächst nicht im Zusammenhang mit dem zu untersuchenden System. Dieses Wissen kann durch Literatur oder Erfahrungen durch Sicherheitsanalysen vorhanden sein und wird bei der Analyse des Systems mit eingebracht.

Notwendig ist zudem, dass das Schutzbedürfnis bekannt ist. Dazu gehört die Festlegung der Schutzziele für Systemteile und verarbeitete Daten. Dies kann durch Kommentare im Quelltext geschehen oder durch eine Architekturbeschreibung, die die Schutzziele enthält. Im folgender Tabelle 6.1 werden nochmals die beschriebenen Wissensteile bei der Sicherheitsanalyse im Überblick angegeben.

Wissen	Beispiel
Systemkenntnisse (statisch):	Quelltext vorhanden
	Dokumentation vorhanden
	Betriebliche Erfahrungen vorhanden
Systemkenntnisse (dynamisch):	Berechnungs- und Antwortzeiten
	Ressourcenauslastung
Wissen über Ausführungskontext:	Einsatzgebiet
	Art der Benutzung (evtl. Use Case)
Kenntnisse bzgl. Sicherheit:	bekannte Schwachstellen
	konkrete Angriffe
	Bedrohungen
Schutzbedürfnis:	Schutzziele für Systemteile
	Schutzziele für Daten

Tabelle 6.1: Vorhandenes Wissen bei der Sicherheitsanalyse

6.1.1.3 Gesucht

Wie besprochen sind verschiedene Informationen über das Systemdesign, Systemverhalten und Wissen über bekannte Schwachstellen und mögliche konkrete Angriffe gegeben. Die Hauptaufgabe der Sicherheitsanalyse lässt sich folgendermaßen beschreiben:

> Hauptaufgaben der *Sicherheitsanalyse:*
> Bei der Sicherheitsanalyse werden alle Strategien, Mechanismen und Vorgehensweisen eines Systems auf potentielle Verletzungen der Sicherheitspolicy untersucht.

6.1.2 Bedrohungsorientierung

Die Sicherheitsanalyse ist ein komplexer Vorgang, der aus der Schwachstellen-, Angriffs- und Bedrohungsanalyse besteht. Durch Anwendung der in Kapitel 5 beschriebenen Verfahren können diese einzeln durchgeführt werden. Für jeden Teil existieren bereits Methoden. Nachdem im Folgenden der Ansatz zur integrierten Sicherheitsanalyse vorgestellt wird,

muss der Zusammenhang der drei einzelnen Analysen und deren Ergebnisse diskutiert werden.

Das Ziel der Sicherheitsanalyse ist das Auffinden aller relevanten Bedrohungen gegen das System und die enthaltenen Daten. Die Menge aller Bedrohungen ist demnach das zentrale Ergebnis der Analyse. Da sich Bedrohungen durch Bedrohungsbäume gut strukturieren und darstellen lassen, werden diese Datenstrukturen auch im Folgenden eingesetzt. Bedrohungen stellen nach Definition aus 2 eine mögliche Verletzung der Sicherheit dar. Die obersten Hierarchien der Bedrohungsbäume kategorisieren die bekannten Bedrohungen nach bestimmten Kriterien. Dazu zählen beispielsweise die Unterscheidung zwischen Client- und Serverbezogenen Bedrohungen und Bedrohungen, die auf die Schutzziele Vertraulichkeit, Integrität und Verfügbarkeit abzielen. Die Blätter der Bedrohungsbäume enthalten dann eine bestimmte Bedrohung, die bewertet werden kann und dann als Basis zur Integration von Gegenmaßnahmen dient. Allerdings bleiben die Beschreibungen der Bedrohungen in den Blättern stets abstrakt. Für die Umsetzung müssen die angegebenen Bedrohungen auf das System konkret übertragen werden. Diese Übertragung wird intuitiv durchgeführt und ist demnach sowohl fehleranfällig als auch schlecht nachvollziehbar. Ein Beispiel wäre die Bedrohungsbeschreibung „Verletzung der Integrität der Daten auf dem Server“. Es kann sehr viele Möglichkeiten für Angreifer geben, einen Angriff auf Basis dieser Bedrohung durchzuführen. In den Bedrohungsbäumen werden die Angriffe allerdings nicht näher beschrieben, obwohl Wissen darüber beispielsweise durch Angriffs- und Schwachstellenanalysen oder Kenntnisse von bekannten Angriffen vorhanden sind.

Angriffe bestehen aus einer Menge von Aktionen, die zur Vorbereitung, bei der eigentlichen Durchführung und beim Ausnutzen der Ergebnisse des Angriffs ausgeführt werden. Diese Aktionen werden in Angriffsbäumen dargestellt. Die Struktur ist der von Bedrohungsbäumen ähnlich. Der Angriff wird in der Wurzel beschrieben, die einzelnen Aktionen werden als Pfad von den Blättern bis zur Wurzel angegeben. Der in der Wurzel beschriebene Angriff nutzt eine bestimmte vorhandene Bedrohung aus.

Dieser Zusammenhang ermöglicht es, dass an den Blättern der Bedrohungsbäume die Wurzeln der Angriffsbäume eingehängt werden, die diese Bedrohung ausnutzen können. Dadurch kann in einer Datenstruktur sowohl das Wissen über die Bedrohungen, als auch Konkretes über die möglichen Angriffe dargestellt werden.

Schwachstellen sind Eigenschaften des Systems, die zur Umgehung der Sicherheitsregeln führen können. Sie sind für die erfolgreiche Durchführung vieler Angriffe nötig und werden deshalb als Knoten in den Angriffsbäumen beschrieben. Damit werden in einer Datenstruktur sowohl Bedrohungen, als auch Angriffe und Schwachstellen dargestellt. Ein Pfad führt so von der Wurzel mit der abstrakten Beschreibung der Bedrohung hinunter zu den möglichen Angriffen und einzelnen Schwachstellen.

Die Bewertungsberechnungen für Risiken und Schäden, die in Bedrohungsbäumen durchgeführt werden können, lassen sich durch die Integration in eine Hierarchie auf die Angriffe und Schwachstellen ausweiten. Wesentlicher Vorteil ist dabei, dass die Entfernung

von bestimmten Schwachstellen im System oder die Verhinderung von Angriffen durch die Darstellung in einem Baum sofort zur Veränderung der Bewertung der entsprechenden Bedrohung führt.

6.2 Schwachstellenanalyse

Mit der Schwachstellenanalyse wird das Ziel verfolgt, alle Eigenschaften des Systems zu ermitteln, die zur Umgehung von Policies führen können. Neben den quelltextbasierten Verfahren, die in Kapitel 5 vorgestellt wurden, existieren auch Versuche zur systematischen Analyse von Schwachstellen. Beispielsweise wurde in [Bis99] die Entwicklung eines Werkzeugs zur Unterstützung bei der Suche von Schwachstellen begonnen. Konkrete Ergebnisse sind bisher allerdings nicht bekannt geworden, da die zugrundeliegende Klassifikation ungeeignet ist, wie im Papier diskutiert wird. Im Folgenden werden deshalb zunächst bekannte Arten von Schwachstellen beschrieben und anschließend Verfahren zu deren Entdeckung entwickelt.

6.2.1 Arten von Schwachstellen

Schwachstellen in Systemen können in beliebiger Form existieren. Es wäre wünschenswert, alle möglichen Schwachstellen aufzuzählen, und das System dann auf diese hin zu testen. Eine vollständige Aufzählung ist im Allgemeinen nicht möglich. Deshalb ist es sinnvoll, die Orte autretender Schwachstellen zu beschreiben und so Gruppen zu bilden. Da ein System stets neben der Software und Hardware auch aus einer physikalischen Umgebung besteht, bietet sich folgende Gruppierung an:

- Software
- Hardware
- Benutzer
- Administrator
- Sonstige Umgebung

Mit den Mitteln aus der Informatik lassen sich Schwachstellen aus der Software-Gruppe beschreiben. Die Verfügbarkeit des Quelltextes beispielsweise stellt eine genaue Beschreibung des Systems dar. Schwachstellen aus den anderen Gruppen sind nur schlecht beschreibbar. Trotzdem dürfen sie bei einer Sicherheitsanalyse nicht vergessen werden. Schwachstellen, die durch die Umgebung entstehen, beispielsweise die unabsichtliche Weitergabe

eines Passworts durch Benutzer, sind schwerwiegend und wesentlich bei einer vollständigen Sicherheitsanalyse.

Ein praktisches Beispiel hierfür betrifft den abgesicherten Speicher, der in Kapitel 8 vorgestellt und analysiert wird. Dieses Konzept dient zur Verschlüsselung und persistenten Speicherung wichtiger Daten. Die Einheiten, in denen Daten gespeichert werden, sind Dateien. Da auf einem persönlichen mobilen Endgerät meistens sowieso nur eine Person mit den Daten arbeitet, liegt es nahe, ein symmetrisches Verfahren mit einem Schlüssel für die Verschlüsselung zu verwenden. Die Einfachheit der Architektur und die Geschwindigkeit beim Ver- und Entschlüsseln sprechen dafür. Würde jedoch der Schlüssel bekannt werden, so wäre die Vertraulichkeit aller Daten des abgesicherten Speichers auf einmal gefährdet. Deshalb wird ein mehrstufiges Verfahren eingesetzt, das aus einer Kombination aus symmetrischem und asymmetrischem Kryptoverfahren besteht. Diese Architekturentscheidung erfolgt aber nur dann, wenn auch die Schwachstellen der Umgebung bekannt sind. Der Verlust des symmetrischen Schlüssel an irgendjemanden in der Umgebung ist ein gutes Beispiel dafür.

6.2.1.1 Benutzerbezogene Schwachstellen

Alle Schwachstellen, die Aufgrund des Verhaltens der Benutzer vorhanden sind, werden benutzerbezogene Schwachstellen genannt. Viele dieser Schwachstellen entstehen durch Fehler bei der Benutzung, die sich aber niemals vollständig vermeiden lassen. Es ist trotzdem sinnvoll, sie zu dokumentieren und wenn möglich vorbeugende Mechanismen in das System zu integrieren, wie etwa Warnhinweise für den Benutzer bei bekannten Fehlbedienungen.

Aufgrund bekannter benutzerbezogener Schwachstellen lassen sich drei Klassen bilden. Eine falsche Bedienung beziehungsweise Fehlverhalten bei der Benutzung lassen Schwachstellen entstehen. Dazu zählt etwa das versehentliche Löschen von Daten und somit die Zerstörung deren Verfügbarkeit oder das Ausschalten einer Firewall aus Komfortgründen. Die zweite Klasse betrifft das Wissen des Benutzers über bestimmte Sicherheitsmaßnahmen. Dazu kann beispielsweise zählen, dass ein Benutzer bekannt gibt, wo Passwortlisten abgelegt sind und welche Daten auf welchen Rechnern abgelegt werden. Eine dritte Klasse betrifft alles Wissen über persönliche Daten. Dazu können Daten dritter gehören, die datenschutzrechtlich relevant sind. Ebenso zählt hier die direkte oder indirekte Weitergabe von Passwörtern an Unberechtigte.

6.2.1.2 Administrative Schwachstellen

Schwachstellen können auch durch administrative Aktionen entstehen. Einfachste Einteilung dieser administrativen Schwachstellen geschieht nach Phasen der administrativen Anwendung. Sowohl bei der Systeminstallation, als auch bei der Systemwartung und dem Softwareupdate können Schwachstellen im System entstehen. Solange Menschen derartige

Aufgaben manuell durchführen müssen, ist die Entstehung dieser Schwachstellen nicht zu vermeiden.

6.2.1.3 Umgebende Schwachstellen

Alle Schwachstellen, die nicht durch die oben genannten Gruppen abgedeckt werden und nicht der Systemsoftware zugeordnet werden können, fallen in den Bereich der umgebenden Schwachstellen. Dazu zählen im Wesentlichen Schwachstellen, die durch die eingesetzte Hardware entstehen und so nicht Personen wie Benutzer oder die Administration betreffen. Diese Gruppe wurde bisher nicht näher eingeteilt, da in den Anwendungen nur sehr wenige derartige Schwachstellen zu finden sind. Ein Beispiel wäre die Verwendung von Speicherkarten in mobilen Systemen, die nach dem Entfernen immer noch persönliche Daten beinhalten, so dass der Besitz dieser Speichergeräte die Vertraulichkeit der Daten beschädigen kann.

6.2.1.4 Systemintegrierte Schwachstellen

Die interessanteste Gruppe der Schwachstellen ist die der systemintegrierten Schwachstellen, da es dafür Möglichkeiten gibt, diese teilweise automatisch zu erkennen. Mit Verfahren zur Erkennung befassen sich die nächsten Abschnitte. Zunächst werden die Bereiche zur Einteilung beschrieben.

Die systemintegrierten Schwachstellen lassen sich nach der Systemschichtung einteilen. Allgemein wäre dies die Hardware, Treiber und Betriebssystem, Middleware sowie Binde- und Lademechanismen, Anwendungen und die Benutzerverwaltung. Diese Einteilung kann allerdings je nach betrachtetem System variieren. Bei der Analyse eines Betriebssystems beispielsweise wird dieses in weitere Schichten einteilbar sein, bei der Analyse einer Middleware werden keine Anwendungen mitbetrachtet.

Eine andere Art der Einteilung geschieht nach Abstraktionsebenen. Die unterste Ebene ist die Ebene der Realisierung. Dort sind alle systemintegrierten Schwachstellen zu sehen. Allerdings führt man die Analyse aus Komplexitätsgründen oft auf höheren Abstraktionsstufen durch, die in Form von Modellen dargestellt werden. Mit der Höhe der Abstraktion gehen Details verloren und damit auch Schwachstellen, die dann nicht mehr sichtbar sind.

In der Praxis ist es sinnvoll, zunächst die wichtigen Systemschichten zu bestimmen und anschließend für jede Schicht die Abstraktionsgrade festzulegen. Dieses Vorgehen wird in den nächsten Abschnitten genauer beschrieben. Zusammenfassend lassen sich die Schwachstellenklassen wie in Tabelle 6.2 beschreiben.

Klasse	Beispiel
Benutzerbezogene Schwachstellen:	Verlust über Systemwissen
	Wissensverlust persönlicher Daten, Schlüssel
	Fehlverhalten und falsche Bedienung
Administrative Schwachstellen:	Während Installation
	Durch Softwareupdate
	Durch Softwarewartung
	Durch Sicherung
Umgebende Schwachstellen:	Systemumgebung
Systemintegrierte Schwachstellen:	Nach Schichten geordnet
	Nach Abstraktionsebene geordnet

Tabelle 6.2: Klassen bekannter Schwachstellen

6.2.2 Modellebene

Im Folgenden werden nur noch die systemintegrierten Schwachstellen genauer betrachtet, da im Normalfall nur für das System eine Beschreibung oder ein Modell vorliegt. Beschreibungen zu den anderen Gruppen sind im Allgemeinen nicht vorhanden. Es wird aufgezeigt, wie Schwachstellen gefunden werden können.

Liegt ein Modell höherer Abstraktionsebene des Systems vor, so kann das Modell manuell nach Schwachstellen durchsucht werden. Das Ergebnis hängt dabei vom Wissen der untersuchenden Personen ab. Dieser Nachteil lässt sich prinzipiell nicht beseitigen, man muss aber versuchen, den Analysevorgang durch systematische Maßnahmen zu leiten. Eine vielversprechende Möglichkeit dafür, die aus dem Bereich der Safety-Analysis während dieser Arbeit in den Security Bereich angepasst wurde, ist Hazard and Operability Analysis (HAZOP), wie etwa in [Lev95] beschrieben. Dieses Verfahren wird allgemein dazu eingesetzt, alle Fehlerzustände eines Systems systematisch zu erfassen. Da es nicht in einen Analyseteil dieser Arbeit aus den vorherigen Kapiteln passt, wird es hier angesprochen.

HAZOP Analyse besteht aus zwei Teilen. Zunächst werden sogenannte Guide-Words als Leitfaden zum Auffinden von Fehlern benutzt, die dann anschließend vermieden werden sollen. Die Menge der Guidewords $\mathcal{M}_{guidewords}$ ist wie folgt gegeben:

$$\begin{aligned}\mathcal{M}_{guidewords} = \{&„NO, NOT, NONE“,\\ &„MORE“,\\ &„LESS“,\\ &„AS\ WELL\ AS“,\\ &„PART\ OF“,\\ &„REVERSE“,\\ &„OTHER\ THAN“\}\end{aligned}$$

Ein Ergebnis dieser Arbeit stellt die Erweiterung und Anwendung der HAZOP Analyse im Security-Bereich dar. Dabei wird durch schrittweise Prüfung der Variablen mit Hilfe aller Guidewords Fehler, bzw. potentielle Schutzzielverletzungen gesucht. Aus den angegebenen Worten werden Fragen gebildet, deren positive Beantwortung auf einen Fehler schließen lassen. Eine Frage könnte beispielsweise für das Wort „OTHER THAN“ sein: „Wird die Variable x mit einem anderen Wert als y beschrieben?“ Im zweiten Teil der Analyse wird eine Vorgehensweise für die Bewertung der eventuell gefundenen Fehler beschrieben. Diese ist allerdings für die Sicherheitsanalyse nicht einsetzbar. Die Guidewords dagegen können ebenso auf Systemmodelle angewendet werden um Schwachstellen zu finden. Dazu wird die Menge der ursprünglichen Wörter um zeitliche Fragen erweitert. Diese werden in Tabelle 6.3 vorgestellt, die Erweiterungen sind im unteren Teil der Tabelle beschrieben.

Schwachstellen sind für den Angreifer entweder direkt oder indirekt durch Funktionsaufrufe ausnutzbar. Das Modell muss derart beschrieben sein, dass alle relevanten Datenflüsse und Funktionsaufrufe mit diesen generischen Fragen überprüft werden können. Dafür sind nur Personen mit guten Sicherheitskenntnissen und hohem Verständnis für das System geeignet. Könnte eine Frage positiv beantwortet werden, so ist die Ursache dafür eine potentielle Schwachstelle im System. Durch die Systematik werden relevante Schwachstellen gefunden, wie auch später bei der Umsetzung in Kapitel 8 erkennbar ist. Die so entstehende Liste wird dann später bei der Erzeugung der Bedrohungsbäume eingesetzt.

6.2.3 Quelltextbasis

Neben der Form bestimmter Modelle kann das System auch durch den Quelltext beschrieben werden. Schwachstellen auf Basis der Systemquellen zu finden kann teilweise automatisch durchgeführt werden. Dazu wurde ein Werkzeug entwickelt, dessen Grundlagen im Folgenden beschrieben werden.

Es sind im Wesentlichen zwei Fragen, die man bei der quelltextbasierten Schwachstellenanalyse beantworten kann.

1. Welche Stellen im Quelltext sind sicherheitsrelevant und müssen auf Schwachstellen

Frage	Bedeutung
No	Nichts geschieht
More	Es ereignet sich zuviel,
Less	oder zu wenig
As well as	Daneben noch etwas Zusätzliches
Part of	Nur ein Teil des Gewünschten
Reverse	Genau das Gegenteil wird erreicht
Other than	Etwas völlig anderes geschieht
Early	Etwas ereignet sich zu früh (bzgl. Zeit)
Late	Etwas ereignet sich zu spät (bzgl. Zeit)
Before	Etwas ereignet sich zu früh (bzgl. Ereignis)
After	Etwas ereignet sich zu spät (bzgl. Ereignis)

Tabelle 6.3: Guidewords zur modellorientierten Schwachstellensuche

hin getestet werden?

2. Welche Schwachstellen, die durch Programmierfehler entstanden sind, sind in dem Quelltext vorhanden?

Die Antworten auf beide Fragen können durch ein Werkzeug unterstützt beantwortet werden. Das Werkzeug und dessen Arbeitsmodell wird in Kapitel 7 vorgestellt.

ad 1. Aus Erfahrungswerten heraus sind die Stellen gesucht, die einer Prüfung durch die oben beschriebenen Guidewords unterzogen werden sollen. Die Antwort kann in die beiden folgenden Teile zerlegt werden.

   a. Zunächst zählen alle Variablen und Speicher dazu, die vertrauliche, wichtige Daten beinhalten. Das Werkzeug kann alle Variablen ausgeben, aber nicht entscheiden, welche davon sicherheitsrelevant sind und welche nicht. Es ist vorstellbar, dass die Programmierer diese Stellen im Quelltext markieren. Dadurch könnte das Werkzeug die entscheidenden Variablen bestimmen. Dies wurde in dieser Arbeit jedoch nicht weiter verfolgt, da auch Quelltexte zu analysieren waren, die nicht derart markiert sind. Einfacher ist es, dass aus der Liste aller Variablen die relevanten manuell bestimmt werden.

Die gefundenen Variablen der Menge $\mathcal{M}_{secvar}$ werden dann durch systematisches Abfragen mit den Guidewords $\mathcal{M}_{guidewords}$ getestet. Es wird dabei nach dem möglichen Zugriff auf die Variable gefragt. Für das Wort „MORE“ und eine Variable „i“ wird dann beispielsweise die Frage gestellt: „Welcher Umstand kann eintreten, damit der Zugriff auf die Variable „i“ `mehr` bzw. `zu häufig` („MORE“) geschieht?“ Damit wird der Zugriff auf die Variablen aus $\mathcal{M}_{secvar}$ getestet. Die Anzahl der zu stellenden Fragen n ist

$n = |\mathcal{M}_{guidewords}| \cdot |\mathcal{M}_{secvar}|$, mit den gegebenen Guidewords also

$n = 11 \cdot |\mathcal{M}_{secvar}|$,

da alle Fragen aus $\mathcal{M}_{guidewords}$ stets für alle Variablen geprüft werden müssen. Die Anzahl der Fragen kann durchaus groß werden, so dass es wichtig ist, die Menge der zu testenden Variablen sinnvoll einzuschränken. Dies ist allerdings ausschließlich eine Frage, die nur nach Anwendungsfall geklärt und nicht allgemein beantwortet werden kann.

b. Neben Variablen sind auch Methoden und Funktionen durch die Guidewords-Methode zu testen, die von einem Angreifer aus verwendet werden können. Die Funktionen können beispielsweise durch Argumente für einen Angriff ausgenutzt werden, die nicht wie gefordert übergeben werden. Durch die automatische Generierung von Aufrufgraphen aus dem Quelltext heraus können Beziehungen und Abhängigkeiten zwischen Funktionen dargestellt werden. Eine Funktion, die etwa ein Argument des Benutzers entgegennimmt und auswertet, ruft wiederum andere Funktionen auf und verwendet globale Variablen. Dies kann durch das entwickelte Werkzeug automatisch analysiert werden. Problematisch ist die Tiefe der Darstellung der Aufrufgraphen. Diese wird auf einen bestimmbaren numerischen Wert festgelegt. Dadurch wird die Anzahl der zu testenden Funktionen beschränkt.

   Jeder Aufrufgraph, der aus einer Menge $\mathcal{M}_{secfunc}$ von Funktionen besteht, wird anschließend mit der Guidewords Methode abgefragt. Dabei wird nach dem Aufruf der Funktion gefragt. Die Frage durch „BEFORE“ wird demnach für eine Funktion $f(a)$ lauten: „Was passiert, wenn die Funktion $f(a)$ vor einem für die Ausführung notwendigen Ereignis aufgerufen wird?“ Die Frage zielt beispielsweise auf nicht initialisierte Variablen und Synchronisationsprobleme. Wenn dadurch Schwachstellen gefunden werden, müssen diese nicht automatisch sicherheitsrelevant sein. Es ist durchaus möglich, dass ein Angreifer diese nicht oder nur mit zu großem Aufwand ausnutzen kann.

   Die Anzahl der zu stellenden Fragen n ist

   $n = |\mathcal{M}_{guidewords}| \cdot |\mathcal{M}_{secfunc}|$,

   da alle Fragen aus $\mathcal{M}_{guidewords}$ stets für alle Funktionen geprüft werden müssen.

ad 2. Schwachstellen, die durch Programmierfehler entstehen, können aufgrund von Erfahrungen gefunden werden. Praktische Beispiele sind C-Funktionen, die nicht korrekt eingesetzt werden und so etwa anfällig für einen Pufferüberlauf sind. Voraussetzung

dafür ist eine ausreichend große Bibliothek mit bekannten Fehlerstellen und Programmierkonstrukten, die für jede Programmiersprache neu aufgebaut werden muss und teilweise plattformabhängig ist. Das Werkzeug, das später vorgestellt wird, findet die in der Bibliothek spezifizierten Muster und gibt deren Codestelle aus. Allerdings muss nicht jede Ausgabe auch automatisch eine Schwachstelle darstellen. Selbst wenn Implementierungsmängel gefunden werden, müssen diese nicht immer für einen Angreifer ausnutzbar sein. Dies muss allerdings am konkreten Beispiel entschieden werden.

6.2.4 Automatisierung

Wie im vorigen Abschnitt erwähnt, können einige der angegebenen Vorgänge bei der Schwachstellenanalyse automatisch durchgeführt werden. Im Folgenden werden die Grundlagen für die Realisierung des entsprechenden Werkzeugs beschrieben. Das Werkzeug selbst wird im folgenden Kapitel vorgestellt.

Entwickelt wurde ein Werkzeug, das Quelltexte verschiendener Sprachen analysiert und folgende Berechnungen durchführt.

- Bestimmung des Aufrufgraphen

  Gegeben ist der Quelltext, ein Funktionsname f und eine maximale Berechnungstiefe t. Das Werkzeug liefert alle Funktionen f_n, die von der Funktion f aus aufgerufen werden können. Anschließend wird dies für alle Funktionen f_n durchgeführt. t beschreibt die maximale Berechnungstiefe.

- Bestimmung aller Variablen

  Das Werkzeug liefert für einen gegebenen Quelltext alle nicht lokalen Variablen. Diese werden manuell bewertet und helfen bei der Guidewords Methode wie oben beschrieben zur Schwachstellensuche.

- Suche nach Schwachstellen

  Ein Quelltext wird nach bekannten Programmkonstrukten durchsucht. Sowohl einzelne verdächtige Funktionsnamen, als auch komplizierte Konstrukte können erkannt werden.

Allgemein muss das Werkzeug den gegebenen Quelltext ebenso verarbeiten, wie ein Übersetzer in der Analysephase. Es wird ein abstrakter Syntaxbaum aufgebaut, der alle Informationen des Programms enthält und mit einfachen Methoden durchlaufen werden kann. Alle drei eben geforderten Berechnungen lassen sich an diesem Baum durchführen, weshalb ein universelles Werkzeug dafür geeignet ist.

Der abstrakte Syntaxbaum ist allerdings mit allen Informationen gefüllt, die während der Analyse bzw. einer Übersetzung anfallen können. Für verschiedene Aufgabe sind Abstraktionen dieser Datenstruktur hilfreich.

6.2.4.1 γ-Abstraktionen

Diese Anpassungen heißen γ-Abstraktionen und stellen jeweils einen passenden Ausschnitt des abstrakten Syntaxbaums dar. Diese Abstraktionen sind besondere Modelle des Quelltextes, wobei das Modell mit der vollständigen Information des abstrakten Syntaxbaums das sogenannte Basismodell ist. Aus den Basismodellen lassen sich problembezogene Ausschnitte generieren. Ist der Ausschnitt beziehungsweise das Modell genau mit den Informationen gefüllt, die zur Untersuchung der syntaktischen oder semantischen Eigenschaften des Quelltextes genügen, so wird das Modell auch Lösungsmodell genannt und für die Berechnungen eingesetzt. Da es oft nicht einfach möglich ist, aus dem Basismodell das Lösungsmodell in einem Schritt zu erzeugen, werden zunächst Zwischenmodelle generiert. Detailierte Informationen zu diesem Werkzeug werden in Kapitel 7 gegeben.

Zur Darstellung werden zwei einfache Beispiele angeführt.

Darstellung aller globalen Variablen: Zu einem beliebigen Quelltext sollen alle globalen Variablen und Strukturen aufgelistet werden. Aus dem Basismodell wird dazu direkt ein Lösungsmodell generiert, das eine listenartige Struktur besitzt. In dieser Struktur werden alle globalen Variablen gespeichert.

Entwicklung eines Aufrufgraphen: Zu einem beliebigen Quelltext und einer bekannten Funktion sollen ein Aufrufgraph erstellt werden. Das Lösungsmodell wird dazu eine graphenartige Struktur besitzen, wobei die Knoten die einzelnen genutzten Funktionen und die Kanten die Funktionsaufrufe darstellen. Diese Struktur wird aber nicht direkt aus dem Basismodell generiert, sondern durch Zwischenmodelle entwickelt. Dadurch ist die Entwicklung des Lösungsmodells in diesem Beispiel einfacher.

6.2.4.2 Netzwerkschwachstellen

Es existiert noch eine weitere Möglichkeit, ein System auf Schwachstellen hin zu testen. Mit Netzwerkscanner können Systeme auf alle offenen Schnittstellen zur Netzwerkkommunikation hin geprüft werden. Da viele der Dienste, die sich hinter den Kommunikationsendpunkten befinden, bekannte Schwachstellen besitzen, können so wichtige Angriffspunkte erkannt werden. Dies ist deshalb besonders bedrohlich, da ein Angreifer keinen physikalischen Zugang zum System haben muss. Er kann über ein Netzwerk das System auf offene Schwachstellen abklopfen und dann gezielte Angriffe starten. Meistens werden weder die Schwachstellenprüfungen, noch die Angriffe erkannt. Mit Hilfe sogenannter Netzwerk- bzw. Portscanner werden die Prüfungen völlig automatisch durchgeführt. Diese Werkzeuge besitzen meistens sogar Mechanismen zur Verschleierung des Untersuchungsvorgangs. Es sind

die folgenden drei Gründe, weshalb die automatische Untersuchung auf Schwachstellen von Netzwerkdiensten hier nicht näher untersucht wurde.

1. Diese Schwachstellen lassen praktisch vollständig durch einfache Verwendung einer Firewall (netzwerkweit oder pro Station) ausschalten. Eine Firewall blockiert sowohl die Suche nach Schwachstellen, verhindert einen Angriff und kann diesen sogar meistens erkennen und darüber berichten.

2. Sowohl zur Erkennung der Schwachstellen als auch zur Erkennung von Angriffen existieren bereits zahlreiche Werkzeuge.

3. Der Fokus dieser Arbeit liegt auf der Untersuchung von Programmquellen, da für die Umsetzung Teile eines bestehenden Betriebssystems analysiert wurden. Die Analyse von Netzwerkdiensten wäre ein völlig neuer und eigenständiger Bereich.

6.3 Angriffssuche

Neben der Untersuchung eines Systems auf vorhandene Schwachstellen sind auch Kenntnisse über mögliche Angriffe wichtig. Grundsätzlich kann zwischen aktiven und passiven Angriffen unterschieden werden. Dieser Unterschied ist für die Suche von entscheidender Bedeutung. Aktive Angriffe sind Sequenzen von Aktionen des Angreifers, die zur Umgehung von Schutzzielen führen können. Passive Angriffe dagegen bedeuten das reine Beobachten des Systems und das Sammeln von Informationen, die in irgendeiner Form sicherheitsrelevant sein können. Die Suche nach Angriffen muss demnach in zwei Teile aufgespalten werden, der Suche nach passiven und nach aktiven Angriffen. Diese Aufteilung wird bei der Beschreibung zur methodischen Suche deutlich.

6.3.1 Angriffsbasis

Angriffe, sowohl aktive wie passive, dienen einem bestimmten Zweck, den der Angreifer verfolgt. Der Zweck, beziehungsweise das Ziel kann oft auf unterschiedliche Art durch verschiedenste Angriffe erreicht werden. Der Angreifer wird die Attacke starten, die den geringsten Aufwand und die geringste Gefährdung für ihn und sein Ziel bedeutet. Der selbe Angriff kann meistens mehrmals durchgeführt werden, falls eine Wiederholung notwendig ist. Das System kann selbst durch Erkennung des Angriffs nur selten entscheiden, welches eigentliche Ziel der Angreifer verfolgt. Deshalb ist es für den Angreifer leicht, nachdem eine Attacke entdeckt und Gegenmaßnahmen installiert wurden, mit einem völlig anderen Angriff wiederum dasselbe Ziel mit denselben Absichten zu attackieren. Bei der Entdeckung von Angriffen lässt sich daher nur wenig über den Angreifer aussagen. Letztendlich bleibt daher bei der Abwehr von Angriffen nur auf bestehendes Wissen durch bekannte Angriffe

zurückzugreifen. Allerdings kann man versuchen, alle Aktionen, die ein Angreifer in einem System durchführen kann, systematisch auf mögliche Angriffsmöglichkeiten zu untersuchen. Dazu wird ein allgemeines Modell beschrieben, das schon in Kapitel 2 angesprochen wurde. Dieses Modell beschreibt Angriffe auf zwei Arten allgemein. Zum einen durch die Ausführungsphasen aller Angriffe und zum anderen durch die allgemeinen Ziele der Angreifer.

Jeder Angriff besteht aus folgenden drei Phasen.

$\mathcal{A}_{(pre)}$ Angriffsvorbereitung

Hierzu zählen alle Maßnahmen, die der Angreifer durchführt, um den Angriff zu unterstützen und zu ermöglichen. Denkbar ist etwa das Sammeln von Informationen über das System und seine Architektur.

$\mathcal{A}_{(act)}$ Angriffsdurchführung

Die Schritte, in denen der Angreifer durch bestimmte Aktionen ein Schutzziel des Systems verletzen will, sind die Angriffsdurchführung.

$\mathcal{A}_{(post)}$ Wirkung und Nutzen

Nachdem der Angriff erfolgreich durchgeführt wurde, kann der Angreifer damit eine Wirkung erzielen oder einen direkten Nutzen davon haben. Wirkung und Nutzen müssen unterschieden werden, da es viele Angriffe gibt, die zwar Wirkung auf ein System zeigen, der Nutzen für den Angreifer aber nicht erkennbar ist oder überhaupt kein Nutzen vorhanden ist. Beispiele für „Nutzungs-lose“ Angriffe mit Wirkung sind etwa aktuelle Angriffe durch Viren und Würmer, die hohen Schaden anrichten und vermutlich kaum Nutzen für den Angreifer zeigen.

Diese Phasen müssen sequentiell abgearbeitet werden und jede Folgephase kann nur ablaufen, wenn die vorhergehende Phase erfolgreich durchgeführt wurde. Bei der Verteidigung wird versucht, durch geeignete Gegenmaßnahmen $\mathcal{A}_{(pre)}$ zu verhindern. Dies gelingt nur den Fällen, bei denen die Vorbereitungsphase bekannt ist, so dass man zusätzlich $\mathcal{A}_{(act)}$ verhindert. Selbst wenn die Durchführung erfolgreich ist, kann man unter Umständen danach noch die Wirkung und den Nutzen des Angriffs minimieren.

Die zu verteidigenden allgemeinen Schutzziele sind Integrität, Vertraulichkeit und Verfügbarkeit. Demzufolge existieren drei grundsätzliche Arten von Angriffen, die wie folgt beschrieben werden.

$\mathcal{A}_{S\rightarrow A}$ Angriff auf die Vertraulichkeit

Ein Angriff auf das Schutzziel der Vertraulichkeit wird durch das Symbol $\mathcal{A}_{S\rightarrow A}$ beschrieben. Dadurch wird der Informationsfluss $\rightarrow$ bei der Attake eines Angreifers A weg vom System S beschrieben.

$\mathcal{A}_{S\leftarrow A}$ Angriff auf die Integrität

Verlust der Integrität wird durch den symbolischen Informationsfluss $\leftarrow$ vom Angreifer A hin zum System S beschrieben.

$\mathcal{A}_{S\Leftrightarrow A}$ Angriff auf die Verfügbarkeit

Der Angriff auf die Verfügbarkeit wird durch die Kopplung $\Leftrightarrow$ zwischen Angreifer A und dem System S dargestellt.

Jeder bekannte Angriff lässt sich in eine Gruppe aus

$\mathcal{M}_{attacks} = \{\mathcal{A}_{S\rightarrow A}, \mathcal{A}_{S\leftarrow A}, \mathcal{A}_{S\Leftrightarrow A}\}$

beschreiben. Jeder einzelne Angriff aus $\mathcal{M}_{attacks}$ wird wiederum in die drei Ausführungsphasen gegliedert

$\mathcal{M}_{attackphases} = \{\mathcal{A}_{(pre)}, \mathcal{A}_{(act)}, \mathcal{A}_{(post)}\}$.

Alle Angriffe lassen sich durch die Verwendung dieser Phasen und den Strukturen für Bedrohungsbäume als Hierarchie darstellen. Wie in Kapitel 5 beschrieben wird, existieren dafür Angriffsbäume, die aber in bestehenden Verfahren nicht nach diesem Schema aufgebaut werden.

In dieser Arbeit werden Angriffsbäume nach folgenden Kategorien strukturiert. Jeder Angriffsbaum besitzt damit die Struktur, die in Abbildung 6.1 zu sehen ist.

- Aktive und passive Angriffe
- $\mathcal{M}_{attacks}$ beschreibt die möglichen Angriffsziele
- $\mathcal{M}_{attackphases}$ zur Strukturierung der einzelnen Angriffe
- Externe und interne Angriffe, in der Abbildung aus Platzgründen nicht dargestellt, sondern nur durch einen Pfeil $\downarrow$ gekennzeichnet.

Neben diesem generischen Wissen über Angriffe sind jeweils konkrete Schritte bekannt. Diese bestehen aus einzelnen, sequentiellen Aktionen zur Durchführung des Angriffs. An den Blättern des Baumes werden die Aktionsfolgen als Pfad angehängt und der Baum so verfeinert.

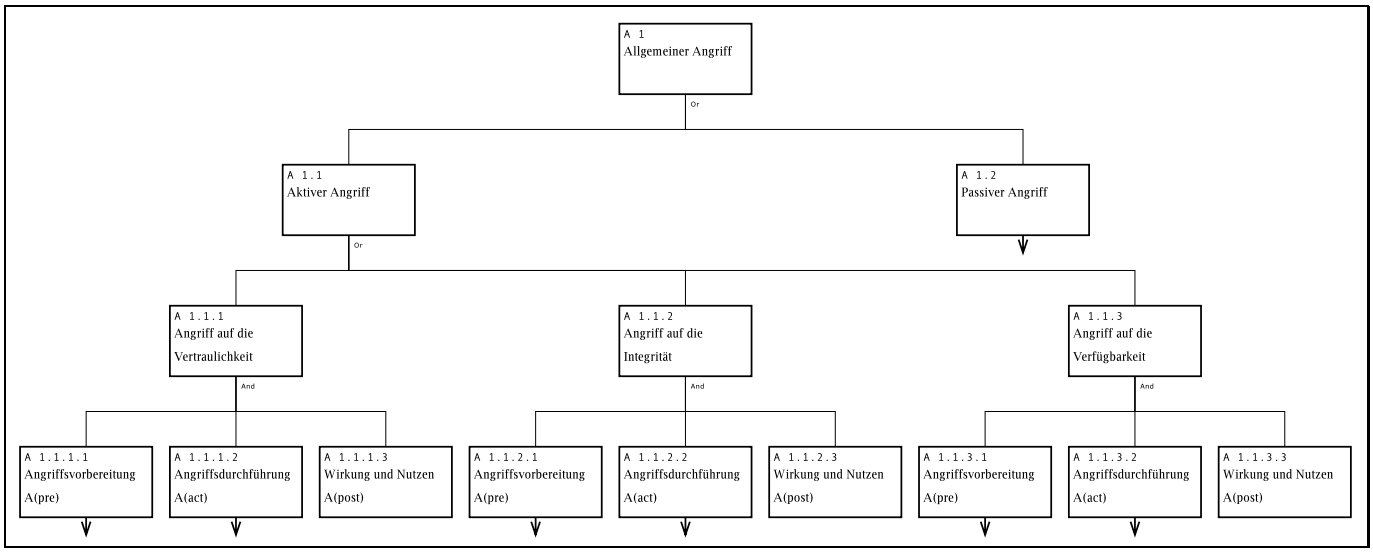

Abbildung 6.1: Allgemeine Struktur eines Angriffsbaums

Beziehung zu Schwachstellen

Die Schwachstellen, die durch die Schwachstellenanalyse erkannt wurden, stellen Grundlagen für die Durchführung der Angriffe dar. Es ist naheliegend, dass sie im Angriffsbaum als Knoten eingesetzt werden. Es gibt zwei Gründe, wann Schwachstellen nicht in die Angriffsbäume integriert werden können. Einerseits kann ein entsprechender Angriff übersehen worden sein, andererseits kann es sich um eine Schwachstelle handeln, die für einen Angriff nicht ausgenutzt werden kann, so dass die Schwachstelle keinen Anker im Bedrohungsbaum besitzen kann.

6.3.2 Modellierung

Grundlage bei der Suche nach Angriffen sind Modelle. Wie bei der Schwachstellensuche ist es sinnvoll, sowohl auf konkreter Quelltextbasis, als auch auf abstrakteren Modellen nach Angriffen zu suchen. Aktive Angriffe sind grundsätzlich durch einzelne Aktionen des Angreifers beschreibbar. Daher ist es bei der Suche nach aktiven Angriffen nötig, alle Schnittstellen zwischen dem System und einem potentiellen internen und externen Angreifer zu suchen. Ausgehend von diesen möglichen Angriffspunkten werden dann Operationen gesucht, die das System kompromitieren können. Die Modelle sind daher derart gestaltet, dass alle Schnittstellen zu Angreifern sichtbar sind. Die Angriffe zielen meist nicht nur direkt auf die Schnittstelle ab, sondern erzeugen weitere Wirkungen im System, die dann zu einer erfolgreichen Attacke führen. Das untersuchte Modell stellt neben den Schnittstellen auch mögliche Wechselwirkungen zu anderen Systemteilen dar. Prinzipiell gilt für

Modelle zur Suche passiver Angriffe das selbe, da sie ebenso die Schnittstellen nach außen, wie auch die internen Systembeziehungen bis zur Schnittstelle darstellen. Die Suche nach potentiellen Angriffen selbst erfolgt dann allerdings auf eine andere Art. Dabei werden die möglichen Daten- bzw. Informationsflüsse vom System nach außen untersucht.

Die Methode zur Suche von Angriffen muss deshalb differentiert danach betrachtet werden, ob nach aktiven oder passiven Angriffen gesucht wird. Im folgenden Abschnitt wird eine Systematik für beide Möglichkeiten beschrieben.

6.3.2.1 Systematik zur Angriffssuche

Die Strukturierung der Systematik ergibt sich aus der oben gebildeten Klassifikation allgemeiner Angriffe.

1. Aktive Angriffe

   Bei der Suche von aktiven Angriffen anhand vorhandener Modelle oder Quelltexte helfen verschiedene Annahmen über allgemeine Angriffe. Aktive Angriffe lassen sich nach internen und externen Angriffen unterscheiden. Dazu kann man für jeden Angriffstyp nach Vorbereitung, Durchführung und dem Nutzen der gewonnenen Daten oder dem veränderten Systemverhalten fragen. Aktive Angriffe sind stets dadurch gekennzeichnet, dass der Angreifer das System durch bestimmte Aktionen aktiv beeinflusst.

   (a) Externe, aktive Angriffe

   Für die externe Angriffsanalyse sind die Methoden zu untersuchen, über die ein potentieller Angreifer das System aktiv beeinflussen kann. Dazu zählen Funktionen, die Daten über das Netzwerk empfangen oder Systemteile, die auf Veränderungen der Systemumgebung reagieren, wie etwa das Anschließen neuer Hardware.

   (b) Interne, aktive Angriffe

   Interne, aktive Angriffe stellen eine Besonderheit dar. Zu untersuchen sind zunächst wie bei externen Angriffen alle Methoden, über die der potentielle Angreifer Einfluss auf das System hat. Der entscheidende Unterschied ist der, dass ein interner Angreifer das System von innen verändern kann, um den Angriff zu ermöglichen. Dies hat zur Folge, dass das Systemmodell unter Umständen nicht mehr dem System entspricht, beispielsweise durch die Veränderung des Systems durch den Angreifer zur Laufzeit. Dies muss bei der Analyse soweit wie möglich berücksichtigt werden.

2. Passive Angriffe

   Passive Angriffe beschränken sich darauf, durch Beobachtung, ohne Einflussnahme auf das anzugreifende System, Informationen zu gewinnen. Es ist sehr schwierig Aussagen zu treffen, ob beobachtbare Daten für den Angreifer nützlich sein können oder nicht. Selbstverständlich gilt dies bei Passworten oder schützenswerten privaten Daten. Der Zusammenhang zwischen öffentlichem Wissen über das System und den schützenswerten Gütern ist allerdings nicht einfach zu erkennen.

   Im Unix Betriebssystem sind beispielsweise immer alle Prozesse für jeden Benutzer sichtbar. Beim Systemdesign geht man davon aus, dass alleine die Kenntnisse über laufende Prozesse anderer Benutzer keine Angriffe zulässt. Das ist nicht richtig. Zwar können direkt keine Daten der Prozesse gewonnen werden oder die Prozesse beeinflusst werden. Jedoch kann zu jedem Prozess der Benutzer, der Startzeitpunkt und die Prozessorauslastung festgestellt werden. Zudem sind die Programmargumente sichtbar, die beim Start übergeben werden. Diese Informationen sind oft schützenswert, da sie mehr Informationen an andere Benutzer oder passive Angreifer weitergeben, als nötig. Als Argumente könnten beispielsweise Passwörter oder URLs angegeben werden, die dann bekannt werden.

   Zur Suche passiver Angriffe ist es notwendig alle Systemteile zu bestimmen, die Daten irgendwelcher Art ausgeben. Dazu zählt sowohl die direkte Ausgabe als auch beobachtbares Verhalten des Systems. Die Suche nach allen informationsliefernden Methoden kann nur im Einzelfall genauer spezifiziert werden. Letztlich stellt dies alles dar, was überwacht und beobachtet werden kann. Der Informationsfluss über Seitenkanäle, die nicht direkt erkennbar sind, kann sehr kompliziert werden. Zudem sind keine Verfahren erkennbar, die diese Suche anleiten können oder vereinfachen. Seitenkanäle werden deshalb in dieser Arbeit nicht speziell betrachtet.

   Der Unterschied zwischen externen und internen passiven Angriffen ist in der Praxis nicht wesentlich ausgeprägt. Das Vorgehen ist in beiden Varianten identisch.

   (a) Externe, passive Angriffe

   Bei dieser Analyse müssen nur informationsliefernde Methoden betrachtet werden, die Effekte über die Systemgrenzen hinaus und damit bis hin zum Angreifer haben.

   (b) Interne, passive Angriffe

   Dabei müssen zusätzlich informationsliefernde Methoden betrachtet werden, die Effekte innerhalb des Systems haben. Bei der Analyse ist es wichtig zu überlegen, wo der interne Angreifer innerhalb des Systems sitzt. Beispielsweise ist es ein grundsätzlicher Unterschied, ob ein interner Angreifer im Kernadressraum oder im eingeschränkten Benutzeradressraum arbeitet.

Sowohl bei der Suche nach aktiven, als auch bei der Suche nach passiven Angriffen werden für jeden zu untersuchenden Systemteil alle drei Phasen des Angriffs betrachtet. In jedem

Schritt wird deshalb analysiert, ob eine Aktion zur Vorbereitung oder Durchführung von Angriffen möglich ist. Zudem wird über den Nutzen nach einem möglichen erfolgten Angriff nachgedacht. Da der letzte Schritt jedoch meist spekulativ durchgeführt wird, kann hierfür nur wenig Anleitung gegeben werden.

6.3.3 Tests

Das Finden der möglichen theoretischen Angriffe ist ein wesentlicher Teil der Analyse. Oft kann jedoch zusätzlich gefragt werden, ob ein bestimmter Angriff auch tatsächlich den gewünschten Erfolg hat. Um das herauszufinden, kann das realisierte System in einer kontrollierten Umgebung dem Angriff ausgesetzt werden. Dies ist nicht Teil dieser Arbeit und wird hier nur kurz angesprochen um diese wichtige Möglichkeit nicht zu vergessen.

Eine Möglichkeit stellt das manuelle Testen dar. Dabei werden die Schritte durchgeführt, die im Angriffsbaum angegeben sind. Treten besondere Umstände auf, können diese im Baum sofort markiert oder eingetragen werden. Nach gesammelter Erfahrung sind dies vor allem vergessene Voraussetzungen oder verändertes Systemverhalten. Beim automatischen Test führen Programme die entsprechenden Schritte aus. Es existieren Werkzeuge für das Simulieren von Benutzereingaben in Bedienoberflächen. Diese erzeugen die entsprechenden Ereignisse von Eingabegeräten wie Tastatur oder Maus und schicken diese an die Oberflächenbibliothek. Bei der Untersuchung der Systemdiensteschnittstelle ist im Rahmen dieser Arbeit ein Werkzeug entstanden, dass Systemdienste mit zufälligen Argumenten aufruft. Alleine durch diesen einfachen Mechanismus können bereits Möglichkeiten gefunden werden, die Systemstabilität und damit die Verfügbarkeit zu beeinflussen. Nähere Informationen wurden in [Gör01] besprochen.

6.4 T_{AV}-Bäume

Für die Dokumentation und Darstellung der Bedrohungen, Angriffe und Schwachstellen sind Baumstrukturen sinnvoll. Diese werden bereits in anderen Verfahren eingesetzt, wie in Kapitel 5 beschrieben wird. Da Bedrohungen (engl. **T**hreats) die grundlegenden Ergebnisse der Sicherheitsanalyse darstellen und Angriffe und Schwachstellen (engl. **A**ttacks and **V**ulnerabilities) die Grundlagen bilden, werden die entwickelten Strukturen T_{AV}-Bäume genannt. Die folgenden Abschnitte beschreiben deren Syntax, den Aufbau und mögliche Prüfungen der Bäume.

6.4.1 Syntax

T_{AV}-Bäume sind Erweiterungen bekannter Bedrohungsbäume, die syntaktische Elemente der Fehlerbäume übernehmen.

6.4.1.1 Bedrohungen

Die Wurzel des Baums beschreibt ein schützenswertes Gut des Systems, was insbesondere verarbeitete Daten betrifft. Von dort existieren stets drei Unterknoten, die die Schutzziele „Vertraulichkeit“, „Integrität“ und „Verfügbarkeit“ beschreiben, so dass die Wurzel eines T_{AV}-Baums stets wie in Abbildung 6.2 dargestellt wird.

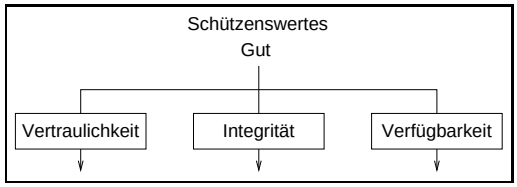

Abbildung 6.2: Allgemeine Wurzel

Aus Platzgründen gekürzte Pfade werden durch einen Pfeil ↓ beendet. Folgeelemente, die durch ▭ gekennzeichnet werden, sind von darunterligenden Elementen abhängig. Diese Abhängigkeit wird durch unterschiedliche logische Symbole beschrieben.

Logische Verknüpfungen

Zur Darstellung einer Konjunktion wird das Symbol ⌓ verwendet, wie in Abbildung 6.3 dargestellt. Es müssen alle Eingänge $e_1 \dots e_n$ eine Voraussetzung für eine Bedrohung darstellen, dann ist auch der Ausgang a eine Bedrohung.

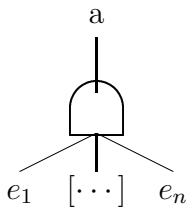

Abbildung 6.3: Logische Konjunktion innerhalb eines T_{AV}-Baums

Die Disjunktion wird analog durch das Symbol ⌓ dargestellt, siehe Abbildung 6.4. Es muss mindestens ein Eingang aus $e_1 \dots e_n$ eine Voraussetzung für eine Bedrohung darstellen, dann ist auch der Ausgang a eine Bedrohung.

Ein Sonderfall ist die vereinfachte Darstellung ohne logisches Verknüpfungssymbol, wie auch in der Abbildung 6.2 gezeigt. Damit wird immer das logische Oder beschrieben, das in der Praxis häufig vorkommt.

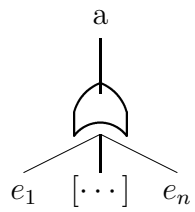

Abbildung 6.4: Logische Disjunktion

Bedrohungsvoraussetzungen

Neben den Voraussetzungen für Bedrohungen, die durch das ▭ Symbol dargestellt werden, wie auch in Abbildung 6.2 zu sehen ist, existiert ein Symbol zur Beschreibung nicht weiter betrachteter Bedrohungen. Dadurch kann ausgedrückt werden, dass die Bedrohung zwar bekannt, aber von vorneherein uninteressant, zu unwahrscheinlich oder nicht ausnutzbar ist. Dafür wird das ◇ Symbol benutzt, das immer als Blatt auftritt, wie in Abbildung 6.4 dargestellt wird.

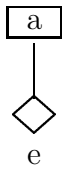

Abbildung 6.5: Nicht weiter betrachtete, aber bekannte Bedrohung

6.4.1.2 Angriffe

Die bereits beschriebenen Angriffsbäume werden an den obersten Knoten der Bedrohungsbäume angehängt, wie in der Abbildung 6.13 skizziert wird. Dadurch werden die meist abstrakten Bedrohungsbeschreibungen durch konkrete Angriffsvorgänge beschrieben. Dazu sind allerdings einige Symbole zur Darstellung sinnvoll, die im Folgenden aufgezählt werden.

Darstellung der Angriffsschritte

Die einzelnen Schritte eines Angriffs werden durch ○ beschrieben, wie in Abbildung 6.6 zu sehen ist.

Wie bei Bedrohungen gibt es auch die Möglichkeit, Verfeinerungen bestimmter Angriffsschritte wegzulassen. Dies wird dann durch ein Blatt mit dem Symbol ◇ nach Abbildung

Abbildung 6.6: Die Angriffsschritte a_1, a_2 und a_3

6.7 dargestellt. Eine Ursache dafür kann sein, dass der Schritt zu aufwändig und daher sehr unwahrscheinlich erscheint.

Abbildung 6.7: Der Angriffsschritt a_1 wird nicht weiter betrachtet

Aus Platzgründen gekürzte Pfade werden wie bei Bedrohungsdarstellungen auch hier durch einen Pfeil $\downarrow$ beendet. Daneben existieren externe Ereignisse, die als Voraussetzung für bestimmte Angriffsschritte nötig sind. Diese werden mit dem $\square$ Symbol dargestellt, Abbildung 6.8.

Logische Verknüpfungen

Die logischen Verknüpfungen für Konjunktion und Disjunktion werden wie die der Bedrohungen verwendet. In Abbildung 6.9 werden Konjunktion und Disjunktion für Angriffsschritte dargestellt.

Zur präziseren Darstellung werden zusätzlich folgende logischen Verknüpfungen definiert, die für Bedrohungen nicht einsetzbar und sinnvoll sind.

Zur Darstellung, dass keiner der Schritte $a_1 \ldots a_n$ möglich sein darf, damit a durchführ-

Abbildung 6.8: Der Angriffsschritt a_1 benötigt das externe Ereignis ev.

Abbildung 6.9: Logische Konjunktion und Disjunktion innerhalb eines T_{AV}-Baums

bar ist, wird das Symbol ⧋ wie in Abbildung 6.10 a) verwendet. Dies ist sinnvoll, wenn Ausschlusskriterien für Angriffe dargestellt werden sollen.

Soll dargestellt werden, dass exakt eine von mehreren möglichen Bedingungen für einen Angriff nötig ist, wird ein exklusives Oder verwendet, das in Abbildung 6.10 b) gezeigt ist. Zudem kann das schrittweise Eintreten von Voraussetzungen in einer bestimmten Reihenfolge zu beschreiben sein, was in Abbildung 6.10 c) dargestellt wird. Dabei müssen a_1 bis a_n in genau dieser Reihenfolge auftreten, damit a gültig werden kann.

Ist ein Angriffsschritt von einem zusätzlichen Ereignis abhängig, so wird das durch das ⬡ Symbol dargestellt. Für einen Angriffsschritt kann beispielsweise eine bestimmte Benutzereingabe nötig sein, die durch ev in wie Abbildung 6.11 beschrieben wird.

6.4.1.3 Schwachstellen

Schwachstellen sind meistens notwendige Voraussetzungen zur Durchführung eines Angriffs bzw. eines Angriffsschrittes. Diese werden durch ein spezielles Symbol ⬭ wie in Abbildung 6.12 dargestellt.

Zusammenfassend werden in Tabelle 6.4 nochmals alle syntaktischen Elemente zur Dar-

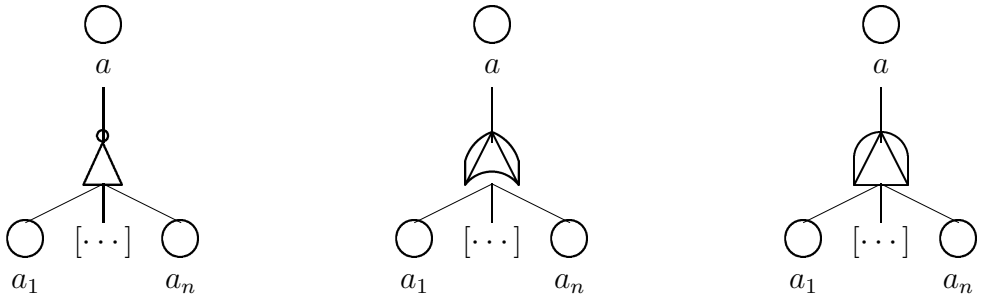

Abbildung 6.10: a) Logisches Nicht, b) Exklusiv-Oder und c) priorisiertes Und

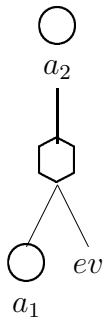

Abbildung 6.11: a_2 kann nur dann erfolgen, wenn a_1 erfolgt und ev zusätzlich gültig ist

stellung von T_{AV}-Bäumen aufgelistet.

6.4.2 Erweiterungen

Die im vorigen Abschnitt beschriebene Erweiterung für Bedrohungsbäume bietet einige neue Möglichkeiten, die in den nächsten Abschnitten ausgeführt werden.

Durch die Erweiterungen lassen sich die Beziehungen zwischen Verfeinerungen der Bedroh-

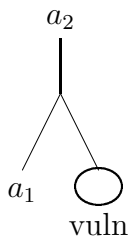

Abbildung 6.12: Darstellung einer Schwachstelle *vuln* im Verbund der Angriffsschritte a_1 und a_2

Symbol im Bedrohungsteil	Bedeutung
	Verfeinerung von Bedrohungen
	Alle Voraussetzungen müssen erfüllt werden
	Mindestens eine der Voraussetzungen muss erfüllt sein
	Bedrohung an dieser Stelle (Blatt) nicht näher betrachtet
↓	Baum aus Platz-/Darstellungsgründen gekürzt
Zusätzliche Symbole im Angriffsteil	**Bedeutung**
	Einzelne Angriffsschritte
	Externe Ereignisse als Voraussetzung für Angriffsschritte
	Keine der Voraussetzungen darf gelten
	Genau eine der Voraussetzungen muss gelten
	Reihenfolge aller Voraussetzungen muss stimmen
	Zusätzliches Ereignis ist nötig
Symbole für Schwachstellen	**Bedeutung**
	Darstellung einer Schwachstelle

Tabelle 6.4: Alle syntaktischen Elemente der T_{AV}-Bäume

ungen einerseits und zwischen mehreren Angriffsschritten andererseits genauer darstellen. Bei den Bedrohungsbeschreibungen ist es für die Übersichtlichkeit wesentlich, dass bestimmte Äste gekürzt werden können. Damit wird dem Betrachter klar, dass zwar Verfeinerungen existieren könnten, diese aber nicht näher interessant sind. Werden diese Teilbäume allerdings für spätere Analysen wiederverwendet, so können diese Teile unter anderen Bedingungen wichtig und an den gekürzten Stellen erweitert werden.

Für die Beschreibung der Angriffsschritte ist es wichtig, dass alle möglichen Voraussetzungen so genau wie möglich beschrieben werden. Schließlich sollen anhand des Baumes Gegenmaßnahmen entwickelt werden. Diese entstehen vor allem dadurch, dass in den Angriffsteilen der Bedrohungsbäume nach Ursachen gesucht wird, die mit einfachen Mitteln verhindert werden können. Dafür ist es nötig, dass die logischen Zusammenhänge zwischen Voraussetzungen und Ursachen klar und so exakt wie möglich angegeben werden. Deshalb

sind oft genauere Angaben als eine Konjunktion und Disjunktion hilfreich. Daneben werden Schwachstellen besonders gekennzeichnet, da sie eine nicht erwünschte Eigenschaft des Systems darstellen. Sie können in der Schwachstellenanalyse teilweise automatisch gesucht werden und müssen bei der Suche von Gegenmaßnahmen besonders betrachtet werden.

6.4.3 Strukturen

Im Hinblick auf die gewünschte Werkzeugunterstützung muss kurz über die Strukturen diskutiert werden, die bei der Konstruktion der T_{AV}-Bäume entstehen. Für den Bedrohungsteil, der aus einer festen Wurzel, den drei Bedrohungen auf die Standardschutzziele und den Verfeinerungen besteht, ist die Baumstruktur klar. Angemerkt muss dabei allerdings werden, dass beispielsweise für eine Bedrohung der Vertraulichkeit eines schützenswerten Gutes auch die Zerstörung der Integrität eines anderen Gutes eine Voraussetzung darstellen kann. Das bedeutet, dass unterhalb der Bedrohungen auf die drei Schutzziele nicht nur über dieses eine Schutzziel diskutiert werden darf, sondern die Dreiteilung immer wieder auftaucht. Anders ausgedrückt kann unter der Kompromitierung eines wesentlichen Systemgutes g_1 ein anderes schützenswertes Gut g_2 als Voraussetzung der Bedrohung gelten. Wiederum können alle drei Schutzziele dieses g_2 bedroht werden, so dass die Dreiteilung auch im Bedrohungsteil weiter unten auftauchen kann. Deshalb können auch Güter, Angriffsschritte und Teilbäume innerhalb des Bedrohungsbaums an verschiedenen Stellen in gleicher Form auftauchen.

Im Angriffsteil spielt die Dreiteilung in die Angriffsphasen eine Rolle. Für jeden möglichen Angriff wird über die Phasen aus $\mathcal{M}_{attackphases}$ diskutiert, wobei in allen drei Stufen des Angriffs wiederum verschiedene Voraussetzungen nötig sein können. Bei der Konstruktion der Angriffsteile findet man häufig Pfade, die bereits für andere Teile des Baums erzeugt wurden. Eigentlich entsteht dadurch eine Verzweigung von einem Teilbaum in einen anderen Teilbaum, möglicherweise sogar ein Zyklus im Graphen. Für den menschlichen Betrachter ist dieser Strukturbruch handhabbar, er wird mit dem Graphen trotzdem arbeiten können. Für das automatische Werkzeug jedoch ist dies mit großem Aufwand verbunden. Deshalb wird der Konflikt derart gelöst, dass ein T_{AV}-Baum immer den bekannten Definitionen einer Baumstruktur genügt. Jeder Knoten hat eine Menge von Kindern und genau einen Vater. Es existiert nur ein Knoten, der keinen Vater besitzt und Wurzel genannt wird. Alle Knoten sind einfach miteinander verbunden. Wenn Teilbäume gefunden werden, die mit anderen Teilbäumen identisch sind, kann das als Attribut an die entsprechenden Teilbaumwurzeln angegeben werden. Eine Verbindung zwischen den Teilbäumen findet dabei aber nicht statt und wird von dem Werkzeug in Kapitel 7 auch nicht unterstützt.

6.4.4 Generierung

Die Schritte zur Erzeugung der T_{AV}-Bäume werden in Abbildung 6.13 nochmals dargestellt. Dadurch wird der Zusammenhang zwischen den Strukturen des Baums und dem vorhandenen Wissen deutlich.

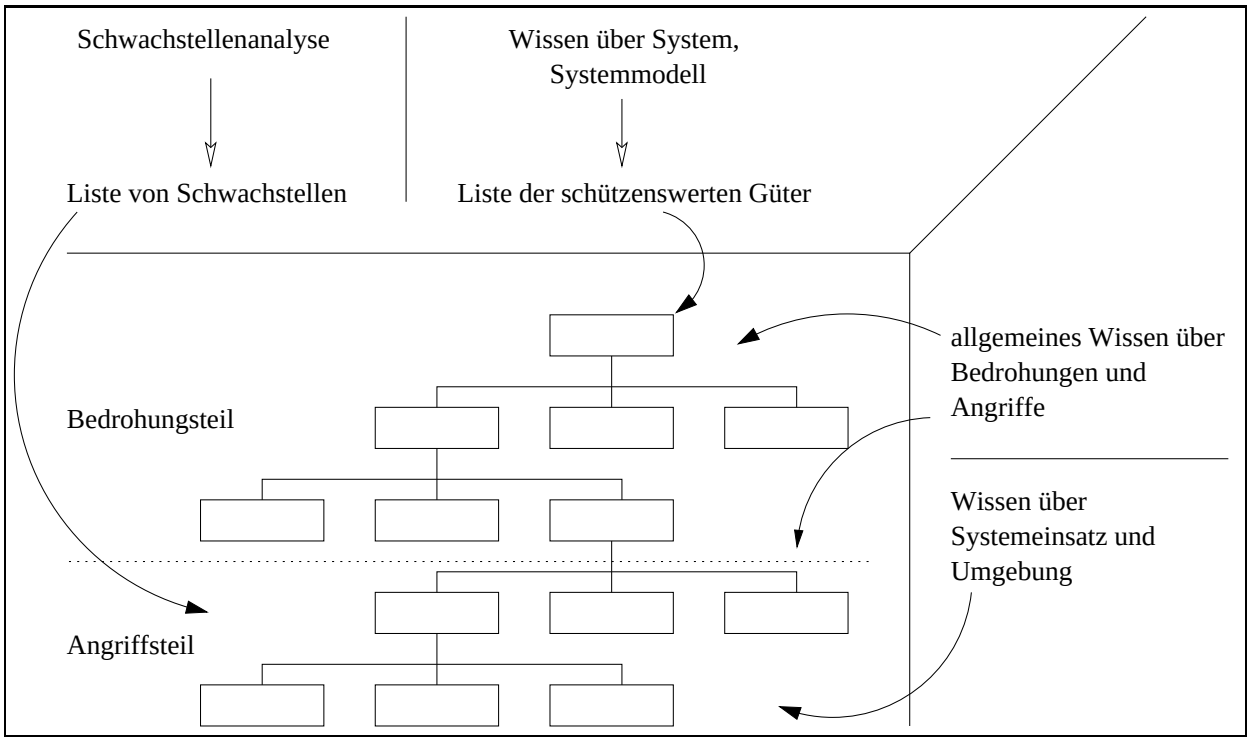

Abbildung 6.13: Schritte zur Erzeugung eines T_{AV}-Baums

Neben der kompletten, meist aufwändigen Neuerzeugung der Bäume können auch Teile anderer Analysen verwendet werden. Sinnvollerweise können Schwachstellenlisten und Wissen über Bedrohungen und Angriffe in Form von Teilbäumen verwendet werden. Das zur Konstruktion der T_{AV}-Bäume entwickelte Werkzeug unterstützt die bibliothekenartige Verwaltung von Teilbäumen. Oft hilft eine Mustersuche zum Finden von gleichartigen Teilbäumen aus der Bibliothek, die auf einem Vergleich der Beschreibungen in den Knoten basiert. Das Auffinden hängt demnach wesentlich von kurzen, prägnanten Beschreibungen ab, die im Allgemeinen durch Erfahrung bei der Konstruktion gefunden werden kann.

Die schrittweise Erzeugung des Baums hängt sehr vom untersuchten System ab. Bei kleineren Analysen kann der Baum top-down entwickelt werden. Dabei werden für die wichtigen schützenswerten Güter einzelne Bäume von den Bedrohungen beginnend entwickelt. Das Vorgehen bei komplizierteren Systemen bedeutet einerseits, dass Angriffsbäume als Teile der T_{AV}-Bäume separat entwickelt werden und dann später integriert werden. Andererseits werden die Bäume durch iteratives Vorgehen verfeinert, wie im folgenden Abschnitt

beschrieben wird.

6.4.5 Iteration

Die Entwicklung der erweiterten Bedrohungsbäume wird zwar unterstützt durch verschiedene festgelegte Strukturen, wie etwa Angriffsphasen oder die drei Bedrohungen. Trotzdem bleibt deren Entwicklung bis zum letzten Schritt nicht automatisierbar und deren Qualität unterliegt immer den Kenntnissen der durchführenden Personen. Dies liegt daran, dass die Möglichkeiten der Angreifer, deren Absichten und deren Position und Angriffsweg stets unbekannt sind. Im Allgemeinen kann ein konstruierter T_{AV}-Baum daher niemals vollständig und abgeschlossen sein. Es ist deshalb notwendig, dass die Bäume, die zur Dokumentation und zur Konstruktion von Gegenmaßnahmen dienen, erweitert werden können. Werden neue schützenswerte Güter bekannt, muss ein neuer Baum erzeugt werden. Die Erweiterung von Bedrohungsarten oder Angriffswegen führt zu keinen strukturellen Problemen. Die Bäume können an jeder beliebigen Stelle verfeinert und mit den syntaktischen Mitteln der T_{AV}-Bäume erweitert werden. Zu beachten sind allerdings zwei wesentliche Dinge.

Erstens können sich Angriffs- und Bedrohungsvoraussetzungen durch die Erweiterungen ändern. Die bereits eingesetzten Gegenmaßnahmen müssen daraufhin angepasst werden. Zweitens können ähnliche Teilbäume an anderen Stellen der T_{AV}-Bäume vorkommen. Diese müssen dann ebenso auf die Erweiterungen hin geprüft werden. Durch entsprechende Hinweise durch das Werkzeug kann auf derartige Probleme aufmerksam gemacht werden.

6.4.6 Prüfung

Unter die Prüfung der entwickelten Bäume fallen zwei Aspekte. Der erste Aspekt ist der, dass bei der Konstruktion der Bäume geprüft wird, ob die wesentlichen Bedrohungen und Angriffe beschrieben sind. Der andere Aspekt behandelt die Prüfung der Bäume von anderen Personen als den eigentlichen Konstrukteuren. Beide Prüfungsarten können durch folgende Fragen unterstützt werden.

1. Sind alle schützenswerten Güter berücksichtigt? Dazu zählen neben den Daten auch die verwendeten Ressourcen und die Systemverfügbarkeit.

2. Wurden alle Schwachstellen berücksichtigt? Diese finden sich prinzipiell im Angriffsteil der T_{AV}-Bäume in Knoten, die durch das Symbol gekennzeichnet sind.

3. Wurden für jeden Angriff alle drei Phasen aus $\mathcal{M}_{attackphases} = \{\mathcal{A}_{(pre)}, \mathcal{A}_{(act)}, \mathcal{A}_{(post)}\}$ berücksichtigt?

4. Wurden stets die Angriffstypen aus $\mathcal{M}_{attacks} = \{\mathcal{A}_{S\rightarrow A}, \mathcal{A}_{S\leftarrow A}, \mathcal{A}_{S\Leftrightarrow A}\}$ berücksichtigt?

Daneben wären Konsistenzprüfungen der verschiedenen Teilbäume, auch durch den Vergleich mit bestehenden Teilbäumen aus einer Bibliothek sinnvoll. Dies wurde in dieser Arbeit jedoch nicht weiter betrachtet und macht vor allem im Zusammenhang mit der Erweiterung der rechnergestützten Werkzeuge Sinn.

6.4.7 Darstellung

Die grafische Darstellung der T_{AV}-Bäume als Baum besitzt gewisse praktische Nachteile. Vor allem aufgrund der Größe der Bäume können sie nur noch bedingt in Dokumenten eingesetzt und abgebildet werden. Möglich ist vor allem die Kürzung der Bäume mit den dafür vorgesehenen syntaktischen Elementen. Oft ist aber der Bezug zwischen Knoten und Teilbäumen und dem Text schwer zu beschreiben, so dass eine textuelle Darstellung der Bäume hilfreich ist. Dazu werden die Knoten des Baums zunächst nummeriert, der Algorithmus dazu arbeitet wie folgt.

Die Wurzel erhält einen Namen, der aus beliebigen Zeichen bestehen kann. Die Kinder der Wurzel werden beginnend mit 1 durchnummeriert und für jedes Kind wird der Name aus Vaternamen plus eigene Nummer generiert. Dann wird rekursiv dasselbe für alle Kinder durchgeführt. In der Tabelle 6.6 wird die textuelle, listenartige Darstellung des T_{AV}-Baums aus Abbildung 6.1 gezeigt. Anwendungsbeispiele für die Verwendung der Darstellung finden sich in Kapitel 8.

6.5 Die Methode TANAT

Im folgenden Abschnitt wird die Methode TANAT („**t**hreat **an**d **a**ttack **t**ree modeling and analysis“) vorgestellt, die einen Vorgehensrahmen für die integrierte Sicherheitsanalyse beschreibt. Bisher wurden die einzelnen Schritte dargestellt, die eine Schwachstellenanalyse, die Bedrohungsanalyse und Syntax und Konstruktion der erweiterten Bedrohungsbäume umfassen. Mit dem durch TANAT beschriebenen Vorgehen werden diese einzelnen Komponenten integriert.

Die Methode wurde für die Sicherheitsanalysen entwickelt, die im Kapitel 8 beschrieben sind. Während dieser Arbeit wurde TANAT bei der Anwendung verfeinert, so dass die hier beschriebene Version die letzte aktuelle einsatzfähige Version darstellt.

Die Schritte bei der Anwendung von TANAT teilen sich in zwei Phasen auf, die Bildung der Einheiten und das eigentliche Vorgehen. Bei der Bildung der Einheiten wird das zu analysierende System für die Anwendung der Methode geeignet dargestellt. Die einzelnen Schritte lassen sich wie folgt aufzählen.

Hierarchie	Type	Name	Risk
A 1	∩	Allgemeiner Angriff	
A 1.1	∩	Aktiver Angriff	
A 1.1.1	∩	Angriff auf die Vertraulichkeit	
A 1.1.1.1		Angriffsvorbereitung A(pre)	
A 1.1.1.2		Angriffsdurchführung A(act)	
A 1.1.1.3		Wirkung und Nutzen A(post)	
A 1.1.2	∩	Angriff auf die Integrität	
A 1.1.2.1		Angriffsvorbereitung A(pre)	
A 1.1.2.2		Angriffsdurchführung A(act)	
A 1.1.2.3		Wirkung und Nutzen A(post)	
A 1.1.3	∩	Angriff auf die Verfügbarkeit	
A 1.1.3.1		Angriffsvorbereitung A(pre)	
A 1.1.3.2		Angriffsdurchführung A(act)	
A 1.1.3.3		Wirkung und Nutzen A(post)	
A 1.2		Passiver Angriff	

Tabelle 6.6: Listenartige Darstellung des T_{AV}-Baums aus Abbildung 6.1

1. Bildung der Einheiten
   (a) Allgemeine Systembeschreibung
   (b) Modellierung der funktionalen Einheiten
   (c) Darstellung der Schnittstellen zwischen den Einheiten
   (d) Beschreibung des Einsatzkontextes
   (e) Darstellung der funktionalen Beziehungen
2. Vorgehen
   (a) Verfeinerungsschritte
   (b) Wahl der Aufruftiefe
   (c) Iterationen

Abbildung 6.14 zeigt die prinzipielle Vorgehensweise bei der Anwendung der Methode. Aus vorhandenem Wissen wird ein Systemmodell erzeugt. Dieses Wissen kann sowohl durch

vorhandenen Quelltext oder durch abstraktere Darstellung und Modelle als auch durch zusätzliche Kenntnisse des Laufzeitverhaltens vorhanden sein. Wichtige Bausteine des Modells, die zur Prüfung bei der integrierten Sicherheitsanalyse herangezogen werden müssen, sind die Schnittstellen zwischen den funktionalen Einheiten. Bei der eigentlichen Analyse wird auf einem bestimmten Verfeinerungsgrad des Modells gearbeitet, der je nach Bedarf iterativ verändert werden kann. Die Erkenntnisse der Analyse werden in den T_{AV}-Bäumen festgehalten. Die genauen Umsetzungen der Schritte sind als Vorschlag zu verstehen. Es ist durchaus möglich und sinnvoll, andere Arten zur Modellbeschreibung oder zusätzliche Schritte bei der Anwendung von TANAT zu verwenden.

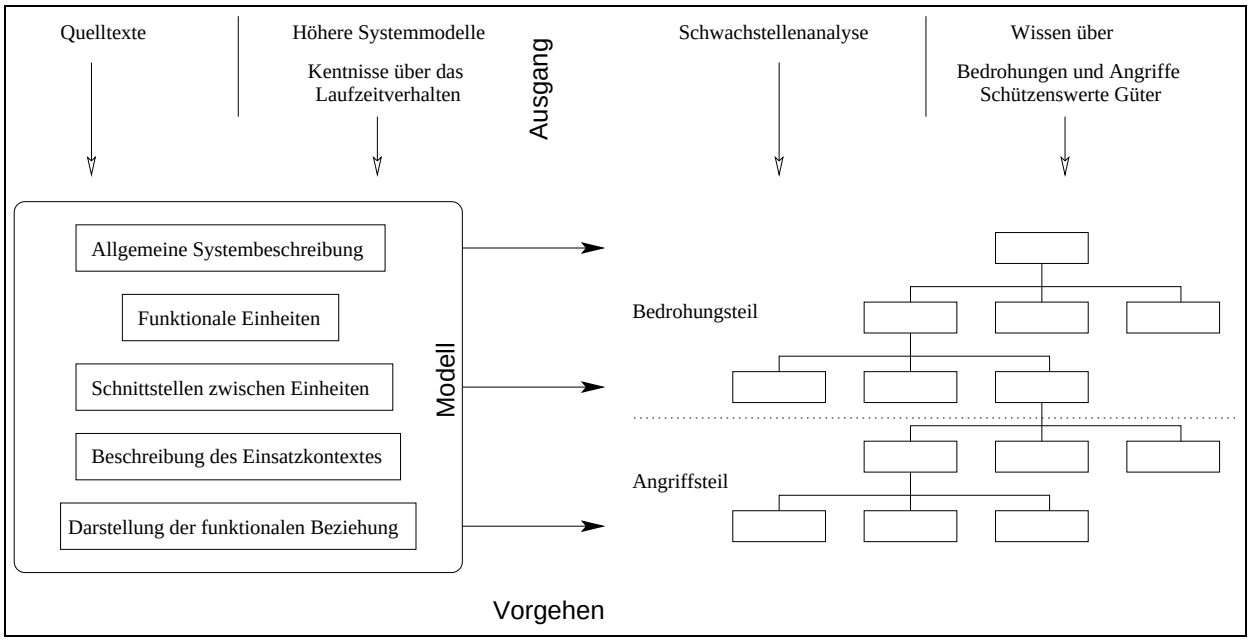

Abbildung 6.14: Allgemeines Vorgehen durch die Methode TANAT

6.5.1 Bildung der Einheiten

In der ersten Phase werden die Einheiten des Systems wie folgt beschrieben.

6.5.1.1 Systembeschreibung

Basis für das Systemverständnis ist eine grobe Beschreibung des Systems. Dies kann auf informellem Wege geschehen, meistens ist ein Fließtext oder Stichpunkte ausreichend. Wichtige Elemente sind Eingabe- und Ausgabewerte und Formate, Einsatz- und Bedienungsstruktur des Systems und die geforderte und umgesetzte Funktion.

6.5.1.2 Einheitenmodellierung

Bei der Analyse werden die Wege zwischen Systemeinheiten auf mögliche Schwachstellen, Angriffe und ausnutzbare Bedrohungen getestet. Unter einer Einheit versteht man in diesem Zusammenhang eine abgrenzbare Struktur, die eine beschreibbare Funktion durchführt und ein Systembestandteil darstellt. Beispiele für derartige Einheiten eines Systems sind der Benutzer, Speicherplatz, Programm- und Betriebssystemfunktionen oder auch Teile der Hardware. Die Vollständigkeit deren Aufzählung beeinflusst wesentlich die Sicherheitsanalyse. Werden wichtige Einheiten weggelassen, so können deren Einflüsse auf das Gesamtsystem nicht systematisch geprüft werden. Es ist nicht einfach festzustellen, welche Einheiten wesentlich sind oder ob etwas vergessen wurde. Jedoch soll der vorige Schritt, die Beschreibung des Systems, dabei helfen. Während der Anwendung von TANAT in dieser Arbeit hat es sich stets bewährt, dass alle Subjekte der Beschreibung als Einheiten beschrieben werden. Als Ergebnis der Einheitenmodellierung liegt eine noch unstrukturierte Liste aller Systemeinheiten vor. Bekanntes Wissen über Schwachstellen und mögliche Angriffe oder die Speicherung vertrauenswürdiger Daten werden der Einheitenbeschreibung angefügt. Zur Anleitung durch den Analyseprozess wurde ein Format für die Einheiten verwendet, das in Tabelle 6.7 dargestellt ist.

Einheit	Datenspeicher	Schwachstellen, Angriffe
Name der Einheit	Welche Daten werden in dieser Einheit abgelegt	Sind Schwachstellen in der Einheit oder mögliche Angriffe bekannt?

Tabelle 6.7: Mögliche Darstellung der Einheiten

6.5.1.3 Schnittstellenbildung

Die gefundenen Systemeinheiten werden nicht isoliert betrachtet, sondern sind über definierte Schnittstellen in der Lage, miteinander zu interagieren. Ein Schritt ist deshalb das Beschreiben aller Schnittstellen, über die ein Angriff stattfinden kann. Auf hoher Abstraktionsebene sind diese leicht an Modellen sichtbar, für die detailierte Darstellung ist der Quelltext des Systems notwendig. Der größte Teil der Beschreibung sind die Funktions- und Methodenaufrufe, die zusammen mit den Parametern und Rückgabewerten angegeben werden. Die Schnittstellen werden für jede Systemeinheit aufgeschrieben, wobei dazu die möglichen Eingaben und Ausgaben notiert werden und eventuell beschrieben wird, ob die Schnittstelle zeitlich abhängig ist. Einige Schnittstellen können nur zu gewissen Zeiten nutzbar sein und nur genau dann für den Angriff genutzt werden.

6.5.1.4 Kontextbeschreibung

Analysiert werden immer Systembeschreibungen, deren Komponenten, wie Systemeinheiten und deren Schnittstellen, sich über die Laufzeit hinweg verändern können. Da die Möglichkeit eines Angriffs jedoch auch von der Umgebung des Systems abhängt, die niemals vollständig beschrieben werden kann, muss der Kontext des Systems mit angegeben werden. Es können Systemteile analysiert werden, deren Umgebung stets gleich bleibt. Bei anderen Systemteilen ändert jedoch der Ausführungsort schon die Umgebung, so dass dies in der Kontextbeschreibung ausgedrückt werden muss. Die Beschreibung selbst kann durch Stichpunkte erfolgen und dient lediglich später zur Nachvollziehbarkeit der Analyse. Ein Nutzer der Analyse kann feststellen, ob die Analyse den entsprechenden Kontext berücksichtigt und ob Teile in einem anderen Kontext wiederverwendet werden können.

6.5.1.5 Funktionale Beziehungen

Nachdem die Einheiten und deren Schnittstellen beschrieben wurden, müssen in diesem Schritt die funktionalen Beziehungen zwischen den Systemeinheiten dargestellt werden. Für Funktionen sind dies alle Funktionsaufrufe. Für andere Schnittstellen, wie etwa die Benutzungsschnittstelle sind die Beziehungen ebenso darstellbar. Der Darstellung der Einheiten liegt prinzipiell die Frage zugrunde, hinter welcher Einheit sich ein Angreifer verbergen könnte. Bei der Analyse wird theoretisch davon ausgegangen, dass von jeder Einheit aus ein Angriff möglich ist. Ebenso kann ein Angriff nicht direkt, sondern als Zwischenschritt über mehrere Einheiten ausgeführt werden. Die funktionalen Beziehungen stellen dabei die Wirkungen zwischen den Systemeinheiten dar.

Um dies zu verdeutlichen werden kurz die beiden Angreifertypen externer und interner Angreifer betrachtet. Ein Angreifer kann externen Ursprungs sein, dann ist keines der modellierten Einheiten gleich dem Angreifer. Trotzdem kann ein Angriff durch diese Einheiten wirken. Wenn die Analyse vom zu schützenden Objekt ausgeht, kann trotzdem der mögliche Angriffsweg ermittelt werden, obwohl der Angreifer durch die Systemeinheiten nicht beschrieben wird. Für interne Angreifer gilt, dass ein oder mehrere beschriebene Einheiten den Angreifer modellieren. Das Prinzip, dass bei der Angriffssuche vom schützenden Gut aus der Angriffsweg gesucht wird, bleibt auch dabei bestehen.

Zur Darstellung der funktionalen Beziehungen werden Abstract State Machines verwendet. Dies hat im Wesentlichen zwei Vorteile.

1. Die operationale Darstellung der Funktionalitäten kann direkt für die beschriebenen Suchverfahren nach Schwachstellen und Angriffen verwendet werden.

2. Die Darstellung der ASMs bietet die Möglichkeit, dass die Schritte zur Verfeinerung durch Erweiterungen der ASM-Beschreibungen durchgeführt werden können.

Funktions- und Methodenaufrufe werden in ASMs ebenfalls als Funktionen dargestellt. Die Zustandsübergänge der ASMs beschreiben dabei stets den Zustandswechsel des Systems und damit einen möglichen Angriffsschritt. Alle anderen wichtigen Modellelemente wie Variablen oder Modulnamen können in ASMs direkt übernommen werden. Da die Semantik von ASMs, wie in Kapitel 4 beschrieben, die parallele Ausführung der Anweisungen darstellt, werden bei Schritten der Verfeinerung die Anweisungsblöcke erweitert. Die Darstellung in ASMs kann bis zu einem dem Quelltext äquivalenten Modell erfolgen. Wie bereits erwähnt können aber auch andere Darstellungsformen außer den Abstract State Machines möglich sein.

6.5.2 Vorgehen

Die zweite Phase von TANAT ist die Durchführungsphase. Dabei wird die Analyse auf den Beschreibungen durchgeführt und die Ergebnisse in T_{AV}-Baumstrukturen festgehalten.

6.5.2.1 Verfeinerung

Die Beschreibungen der Systemeinheiten und Schnittstellen können unterschiedlich detailiert sein. Jeder mögliche Detailierungsgrad ist für verschiedene Anforderungen der Analyse geeignet. Bei den bisherigen Analysen mit TANAT hat sich ergeben, dass maximal drei unterschiedliche Abstraktionsstufen sinnvoll sind. Die Anzahl hängt aber von der Größe und Komplexität der Systeme ab, so dass auch mehrere Stufen möglich wären. Die drei gewählten Stufen werden folgendermaßen genannt.

1. Stufe des Grobentwurfs
2. Stufe des Feinentwurfs
3. Implementierung

Bei der Erstellung der drei Entwürfe wird folgendermaßen vorgegangen. Typischerweise beginnt man mit dem Grobentwurf. Dabei sind die wesentlichen Einheiten dargestellt und die Schnittstellen, die die schützenswerten Güter direkt betreffen. Im Feinentwurf werden alle Einheiten dargestellt und zusätzlich alle Schnittstellen, die Einfluss auf die Güter haben können. In der Implementierungsebene werden alle Einheiten und Schnittstellen beschrieben, wie sie im Quelltext sichtbar sind. Die Verfeinerungsschritte können ebenso nach dem ersten Iterationsschritt der Analyse durchgeführt werden.

6.5.2.2 Aufruftiefe

Wie bereits weiter oben erwähnt sollte eine gewisse Aufruftiefe festgelegt werden, wie weit man bei der Analyse die Funktionsaufrufe durch die Einheiten verfolgt.

6.5.2.3 Mustervergleich der Angriffsklassen

Sind die Einheiten, Schnittstellen und Beziehungen dazwischen in einer der drei Abstraktionsstufen dargestellt, so kann mit der Analyse begonnen werden. Die Schritte bei der Suche von Schwachstellen und passiven und aktiven Angriffswegen wurden zu Beginn des Kapitels vorgestellt und werden bei der Analyse für alle zu schützenden Güter durchgeführt. Das Verfahren zum Auffinden von Schwachstellen auf der Basis von Guide-Words betrachtet die Funktionsschnittstellen so, als würde dahinter eine Schwachstelle vorliegen, die vom Angreifer über diese Schnittstelle ausgenutzt werden kann. Der Mustervergleich der Angriffsklassen basiert auf dem Wissen über bekannte Angriffe und sucht für jede Schnittstelle nach möglichen Attacken aus den Mengen $\mathcal{M}_{attackphases}$ und $\mathcal{M}_{attacks}$. Gefundene Angriffe werden dann an den entsprechenden Stellen im T_{AV}-Baum beschrieben. In diesem Schritt sind die Kenntnisse der analysierenden Person entscheidend. Der Mustervergleich kann nicht durch Berechnungen (als weit ausgelegter Begriff) durchgeführt werden, sonder durch intuitives Verständnis der Systemeigenschaften, des Systemverhaltens und der Möglichkeiten eines Angriffs. An dieser Stelle wird sichtbar, warum die Sicherheitsanalyse nicht vollständig automatisch durchgeführt werden kann und die Qualität immer auch vom Wissen der durchführenden Person abhängig ist.

6.5.2.4 Iteration der Bedrohungsanalyse

Die Analyse wird über alle Verfeinerungsschritte hinweg durchgeführt, wie es in Abbildung 6.15 programmatisch dargestellt ist.

6.5.3 Bewertung

Die beschriebene Methode zur integrierten Sicherheitsanalyse TANAT bietet einen Rahmen zur schrittweisen Analyse von Systemen oder Teilen davon. Sie bietet im Gegensatz zu den bestehenden Verfahren eine wesentlich ausgeprägtere Systematik, so dass die Analyse leichter durchführbar und nachvollziehbar wird. Der erfolgreiche Einsatz der Analyse wird im Kapitel 8 an einigen Beispielen besprochen. Tatsächlich lassen sich dadurch theoretisch mögliche Angriffe finden, die auch im ausgeführten System möglich sind und Bedrohungen für das System darstellen.

Die Struktur des Problems führt jedoch dazu, dass auch mit dieser Methode nicht prinzipiell alle Gefährdungsursachen gefunden werden können und dass die Qualität der Analyse

```
Alle Ebenen aus Grob-, Feinstruktur und Implementierung:
{
  Alle modellierten Schnittstellen
  {
    Alle beschriebenen Gueter
    {
      Suche nach Schwachstellen
      {
        Fragenkatalog durch {M_guidewords}
        {
          alle Variablen
          alle Methoden
        }
      }
      Suche nach Angriffen
      {
        Suche nach Angriffen aus {M_attacks}
        {
          Phasen nach {M_attackphases}
          {
            aktive und
            passvie Angriffe
          }
        }
      }
    }
  }
}
```

Abbildung 6.15: Beschreibung der Iteration bei der Analyse

in einigen Schritten auf die Erfahrungen der durchführenden Personen angewiesen ist. Ein folgender wesentlicher Punkt spricht dagegen, dass diese Sicherheitsanalyse irgendwann vollständig und automatisch durchgeführt werden kann. Der Angreifer, mit allen seinen Möglichkeiten und seinem Wissen, ist nicht allgemein vollständig beschreibbar. Daher werden immer neue Angreifer auftauchen können, die einen neuartigen Angriff durchführen, der ein System kompromitieren kann. Genau aus diesem Grund ist eine von Personen durchgeführte Analyse notwendig.

6.6 Gefahrenpotential

Die entwickelten T_{AV}-Bäume stellen mögliche Bedrohungen dar. Daneben lassen sich Abläufe bei der Vorbereitung, Durchführung und der Nutzen eines Angriffs ablesen. Diese Phasen eines Angriffs werden durch die Pfade innerhalb des Baums dargestellt. Neben den eigentlichen Informationen können die Kanten und Knoten durch Attribute/Wertepaare erweitert werden. Erst dadurch werden Berechnungen anhand der T_{AV}-Bäume ermöglicht.

Das wichtigste Ergebnis aus den Berechnungen sind die möglichen Gefahrenpotentiale aufgrund der Bedrohungen. Die Gefahrenpotentiale können mathematisch durch Risikoberechnungen im Rahmen der Risikoanalyse ermittelt werden. Auch wenn die Berechnung der Risiken kein zentraler Bestandteil dieser Arbeit und der Methode ist, wird an dieser Stelle kurz darauf eingegangen. Es wird gezeigt, dass die Strukturen der T_{AV}-Bäume für verschiedene Berechnungen geeignet sind.

6.6.1 Allgemeine Risikoanalyse

Bei der allgemeine Risikoanalyse ist ein Zahlenfaktor gesucht, der zu einem bestimmten Angriff durch seinen numerischen Wert das Risiko dieses Angriffs beschreibt. Da dabei im Allgemeinen Werte ohne absolute Bezugsgröße und Werte ohne Einheiten berechnet werden, stellen sie das Risiko relativ zu den anderen berechneten Risiken dar.

Zur Berechnung des Risikos werden zumindest die beiden Faktoren Eintrittswahrscheinlichkeit des Ereignisses und Wert der zu schützenden Güter berücksichtigt. Das Ereignis stellt in diesem Kontext immer einen Angriff dar, der auf der untersuchten Bedrohung basiert.

Das Risiko für einen konkreten Angriff hängt von allen Faktoren ab, von denen dieser Angriff abhängig ist. In den T_{AV}-Bäumen sind diese Abhängigkeiten durch die unter der Angriffsbeschreibung liegenden Teilbäume dargestellt. Es liegt daher nahe, die Berechnungen des Risikos so durchzuführen, dass die Risiken aller enthaltenen Teilbäume rekursiv zusammengerechnet werden. Die Art dieser Berechnung hängt jeweils von der logischen Verknüpfung (Konjunktion, Disjunktion usw.) ab.

Definition 6.1 (Risiko einer Bedrohung)
Das Risiko $\mathcal{R}$ wird definiert als

$$\mathcal{R} = k * \mathcal{P} * \mathcal{V}$$

wobei $\mathcal{P}$ die Eintrittswahrscheinlichkeit des Auftretens darstellt, $\mathcal{V}$ den Wert der zu schützenden Güter und k eine Proportionalitätskonstante ist.

□

Zur Berechnung muss zunächst ein Wertebereich für $\mathcal{P}$ und $\mathcal{V}$ festgelegt werden. Dieser

Wertebereich bestimmt nach der Formel auch die möglichen Werte für das Risiko. In der Praxis findet man häufig die Gültigkeitbereiche

$$\mathcal{P} \in \mathbb{R} \wedge \mathcal{P} \in [0\,;1]$$

und

$$\mathcal{V} \in \mathbb{N} \wedge (0 \leq \mathcal{V} \leq 100)$$

Damit gilt für den Wertebereich des berechneten Risikos

$$\mathcal{R} \in [0\,;100]$$

Ebenso ist ein Maßstab für die Bestimmung der Werte $\mathcal{P}$ und $\mathcal{V}$ festzulegen. Dieser wird bei der Bewertung der Angriffe verwendet. In vielen Fällen werden die Werte $\mathcal{P}$ und $\mathcal{V}$ jedoch nicht exakt berechnet, weshalb die ermittelten Risiken ebenfalls nur Anhaltspunkte darstellen. Nach der Risikoanalyse dienen sie nur zur Gewichtung der Risiken untereinander. Idealerweise werden anschließend Gegenmaßnahmen eingeführt, die die Angriffe beginnend mit den höchsten Risikobewertungen vermeiden.

6.6.2 Risikobewertung an T_{AV}-Bäumen

Die allgemeine Berechnung des Risikos wird bei der Verwendung von T_{AV}-Bäumen verfeinert. Zunächst gilt, dass die Eintrittswahrscheinlichkeiten als Attribute-Wertepaare an die entsprechenden Knoten und Blätter angetragen werden. Diese beschreiben die Wahrscheinlichkeit, dass das im Knoten beschriebene Ereignis eintritt. Daneben werden auch die Werte der Güter in einem Attribute-Wertepaar angegeben, die bei einem erfolgreichen Angriff verloren gehen können. Man geht bei der Analyse immer davon aus, dass der gesamte Wert eines Gutes entweder erhalten bleibt oder vollständig verloren geht, mit Bruchteilen also nicht zu rechnen ist.

Die Gesamtwahrscheinlichkeit eines Ereignisses, das im Knoten $\mathcal{K}_1$ beschrieben wird, berechnet sich aus der angegebenen Eintrittswahrscheinlichkeit und den Eintrittswahrscheinlichkeiten der Knoten in den darunter liegenden Teilbäumen. Das Risiko eines Ereignisses lässt sich dann aus den gesamten Eintrittswahrscheinlichkeiten und den Wertesummen errechnen. Die Berechnung hängt dabei von der Art der Kanten beziehungsweise der logischen Verknüpfung ab und wird in Abbildung 6.16 skizziert.

```
func: Integer RISK calculate (Vertex V)
{
  if V.CHILDBRANCH = ∩ or ⩛ then {
    forall CHILD in V.CHILDS {
      let RISK = RISK · CHILD.RISK
    }
    return RISK
  }
  if V.CHILDBRANCH = ∩ or ⩛ then
    return max( forall CHILD.RISK in V.CHILDS)
  }
  if V.CHILDBRANCH = △ then {
    forall CHILD in V.CHILDS {
      let RISK = RISK · CHILD.RISK
    }
    return 1 − RISK
  }
  V.RISK = V.P · V.V
  return V.RISK·calculate(V.CHILD)
}
```

Abbildung 6.16: Algorithmus zur Risikoberechnung in T_{AV}-Bäumen

6.6.2.1 Schadensberechnung

Neben der Berechnung des Risikos bestimmter Angriffe und Bedrohungen sind oft auch die Berechnungen der möglichen auftretenden Schäden interessant. Diese können durch die angegebenen Größen der Werte der Güter ausgerechnet werden. Die Berechnung selbst erfolgt nach den selben Prinzipien, wie für die Berechnung der Risiken und möglichen Nutzen. Der zu erwartende Schaden ist aus zwei Gründen interessant. Erstens können Schäden vom Angreifer direkt gewollt sein, da viele Angriffe heute weniger nach ihrem Nutzen, sondern eher nach der Schadenshöhe ausgewählt werden. Zweitens können die möglichen Schäden ebenfalls bei der Auswahl der wesentlichen Bedrohungen helfen.

6.6.2.2 Aufwand und Nutzen

Ein Angreifer hat für die Durchführung des Angriffs einen gewissen Aufwand und oft einen Nutzen durch den Erfolg des Angriffs. Der Gesamtaufwand setzt sich zusammen aus der Summe aller Kosten in den Unterbäumen zu einem Angriff. Der Kostenfaktor in einem Angriffsschritt entsteht aus den anfallenden Kosten für die Vorbereitung (pre) und für die Durchführung des Angriffs (act).

Der Nutzenfaktor in einem Angriffsschritt kann direkt aus dem Attributwert im Nutzen-

zweig (post) abgelesen werden. Der Gesamtnutzen errechnet sich damit aus der Summe aller Nutzenwerte in den Unterbäumen. Werden für die Kosten- und Nutzenwerte monetäre Einheiten festgelegt, beispielsweise in Euro (€), dann sind Kosten- und Nutzenwerte miteinander vergleichbar und der Gesamtnutzen $\mathcal{N}$ kann wie folgt exakter bestimmt werden. k sei die Anzahl der darunterliegenden Teilbäume, $\mathcal{N}_i$ der Gesamtnutzen des Teilbaums i und $\mathcal{K}_i$ die Gesamtkosten des Teilbaums i.

$$\mathcal{N}_{gesamt} = \sum_{i=1}^{k} (\mathcal{N}_i - \mathcal{K}_i)$$

Ist $\mathcal{N}_{gesamt}$ positiv, so ist der Angriff lohnenswert, wird $\mathcal{N}_{gesamt}$ negativ, dann kostet der Angriff mehr, als er dem Angreifer Vorteile bringt.

6.7 Komposition des Gesamtsystems

Die beschriebene Methode zur integrierten Sicherheitsanalyse kann sowohl entwicklungsbegleitend, als auch für bestehende Systeme angewendet werden. Der Analyse muss auch eine Umsetzungsphase folgen, in der die gewonnenen Kenntnisse zur Erhöhung der Sicherheit in das System integriert werden. Diese Komposition des Gesamtsystems erfolgt durch die Auswahl der geeigneten Gegenmaßnahmen aus dem bekannten Maßnahmenkatalog.

6.7.1 Modellhafte Komposition

Die Auswahl von wirkungsvollen Gegenmaßnahmen ist anspruchsvoll und kann nicht automatisch erfolgen. Oft gibt es für eine Bedrohung mehrere unterschiedliche Abwehrmaßnahmen, so dass diese nur durch „subjektive“ Auswahl eingesetzt werden. Diese kreative Arbeit von den Personen, die die Sicherheitsanalyse anwenden, wird aber durch folgende Faktoren beeinflusst.

- Verfügbarkeit der Gegenmaßnahmen in Form von Wissen, Quelltext oder Bibliotheken
- Beeinflussung der Ausführungsgeschwindigkeit und des Komforts des Gesamtsystems
- Benutzerakzeptanz und rechtliche Rahmenbedingungen
- Probleme bei der Komponierbarkeit verschiedener Gegenmaßnahmen

Zum letzten Punkt muss angemerkt werden, dass die Integration mehrerer Gegenmaßnahmen in ein System nicht grundsätzlich problemlos möglich ist. Es ist denkbar, dass

verschiedene Sicherheitsmaßnahmen sowohl funktional, als auch für den Benutzer widersprüchliche Sicherheitspolicies verfolgen und sich so entweder gegenseitig behindern oder sogar den Gewinn an Absicherung reduzieren. Es existieren aber Ansätze zur Komposition verschiedener Sicherheitsmechanismen, die beispielsweise in [Man02] und in [McC87] nachgelesen werden können.

6.7.2 Realisierung

Die Realisierung der Gegenmaßnahmen kann sowohl entwicklungsbegleitend, als auch außerhalb des eigentlichen Systementwicklungsprozesses erfolgen. Da die Sicherheitsmaßnahmen im System integriert sein sollen, ist auch die Integration des Security-Engineering in den Entwicklungsprozess wünschenswert. Dadurch kann eine den Anforderungen entsprechende Zusammenarbeit zwischen den Funktionen und den Sicherheitsmechanismen realisiert werden. In der Realität der Systementwicklung ist es jedoch meistens so, dass der Systementwickler und der Sicherheitsexperte verschiedene Personen sind, die nicht sehr eng zusammenarbeiten. Deshalb wird zwischen der Entwicklung der Gegenmaßnahmen und der eigentlichen Funktionalitäten des Systems eine gewisse Phasenverschiebung auftreten und beide Systemteile zu unterschiedlichen Zeitpunkten realisiert werden.

Bei der Realisierung außerhalb des Entwicklungsprozesses kann man oft auf Bibliotheken mit entsprechenden Schutzmechanismen zurückgreifen, wie es etwa mit den Kryptografiebibliotheken üblich ist. Nicht alle Gegenmaßnahmen lassen sich so einfach integrieren, notwendige Anpassungen können relativ aufwändig werden, wie beispielsweise bei der Integration von Zugriffskontrollen durch ACLs in ein bestehendes System.

6.7.3 Modelle der Gegenmaßnahmen

Eine Gesamtmethode zur Entwicklung abgesicherter Systeme muss aus der Analyse und der Komponier- und Integrationsphase bestehen. Deshalb ist es das Ziel, dass die Beschreibungen der Sicherheitsprobleme, die Bedrohungen nach T_{AV}-Baumstrukturen, ebenso beschrieben werden, wie die verfügbaren Gegenmaßnahmen in einem Maßnahmenkatalog. Damit können die Schutzmechanismen einfach ausgewählt und in das abzuhärtende System integriert werden. In dieser Arbeit werden die Bedrohungen auf Basis von Abstract State Machine Beschreibungen analysiert, so dass es naheliegend ist, die Elemente im Gegemaßnahmenkatalog ebenso durch ASMs zu beschreiben. Durch einfaches bzw. automatisches Vergleichen sollten die Gegenmaßnahmen durch die Analyse der T_{AV}-Bäume gefunden werden.

Dies funktioniert in der Praxis nur teilweise, da die Modelle der Gegenmaßnahmen und der Anforderungen nach den T_{AV}-Bäumen nicht immer vergleichbar sind. Der Vergleich gelingt nur durch den Vorgang des menschlichen Verstehens, das durch Maschinen und mit informatischen Mitteln nicht ausdrückbar ist. Damit bleibt die Auswahl der Gegenmaßnahmen

ein informeller Prozess, der durch den Katalog unterstützt wird.

Ein Ansatz zur Verbesserung dieser Tatsache ist die Beschränkung auf ein spezielles Sicherheitsproblem. Werden etwa nur Informationsflüsse analysiert und für deren Schutz Gegenmaßnahmen gesucht, so bestehen gute Aussichten, dass man den Prozess besser anleiten kann. Allerdings werden als Ziel dieser Arbeit uneingeschränkt alle möglichen Gegenmaßnahmen angeboten. Die Beschränkungen auf Informationsflussmodelle wie in [Den76] und [DD77] decken aber nur den Teil der Sicherheitsbetrachtungen ab, der die gewollten und ungewollten Informationflüsse beschreibt und kontrolliert.

Kapitel 7

Toolunterstützung bei der Analyse

In diesem Kapitel wird ein Überblick über die Werkzeuge gegeben, die für die Sicherheitsanalyse entwickelt wurden. Zunächst werden allgemeine Anforderungen an die Tools diskutiert und anschließend die Werkzeuge zur Analyse von Quelltexten und zur Konstruktion von T_{AV}-Bäumen beschrieben.

Im Wesentlichen wurden drei Werkzeuge in dieser Arbeit umgesetzt. Eines zur Erzeugung und Analyse von Aufrufgraphen bestehender Quelltexte. Damit können die Programmablaufspuren für die Sicherheitsanalyse verfolgt werden. Das zweite Werkzeug dient der Konstruktion und Verwaltung der T_{AV}-Bäume, die innerhalb der Methode TANAT entwickelt werden. Das Dritte stellt ein allgemeines Werkzeug zur Analyse von Quelltexten dar, womit beispielsweise bekannte Schwachstellen in Programmquellen gefunden werden können. Diese Werkzeuge werden im Rahmen dieses Kapitels im Überblick erklärt.

7.1 Anforderungen

Zu Beginn waren einige grundlegenden Anforderungen an die Werkzeuge gegeben, die nun kurz angesprochen werden. Sie resultieren aus den Zielen und den vorhandenen Kernteilen für das Betriebssystem.

- Analyse bestehender Quelltexte

  Da bestehende Quelltexte weiterverwendet werden, ist es wichtig, dass die Werkzeuge mit bestehenden Quellen umgehen können. Zu der korrekten Analyse der Quelltexte gehört damit ein vollständiger Scanner und Parser für die Standardsprache, wobei im Wesentlichen C-Quellen Wiederverwendung fanden. Da systemnahe Quellen auch Anweisungen enthalten, die nur spezielle Compiler verstehen, hier der GCC Version 3, die nicht dem Sprachstandard entsprechen, sondern Spezialanweisungen sind, müssen

diese Erweiterungen bei der Analyse ebenso berücksichtigt werden. Ansonsten würde die Analyse an den entsprechenden erweiterten Stellen abgebrochen werden.

Als Beispiel sei das Programm 7.1 gezeigt, wobei die Anweisungen `__user` sowie `asmlinkage` keine gültigen C-Anweisungen sind. Sie dienen dem Compiler zur Erkennung von C-Funktionen, die von einem Assemblercode aufgerufen werden sollen. Dadurch können auf einigen Architekturen Fehler vermieden und Codeoptimierungen durchgeführt werden. Die Unterstützung dieser Erweiterungen, von denen es im verwendeten Compiler GCC Version 3.1 einige gibt, sind für die fehlerfreie Analyse notwendig. Der Teil der dargestellten Funktion selbst ist der Einsprungpunkt vom Benutzerprozess aus in den Kern, nachdem der Systemdienst für das Neustarten aufgerufen wurde. Zeilen 4–6 zeigen die üblichen Prüfungen auf die erforderlichen Rechte zur Ausführung des Systemdienstes. Damit wird sichtbar, wie sehr die Realisierungsebene von systemnaher Software auch von den eingesetzten Werkzeugen abhängig ist. Dadurch wird einerseits die Portabilität der Anwendungen eingeschränkt, andererseits die Schwierigkeiten für quelltextbasierte Analysewerkzeuge sichtbar.

```
asmlinkage long sys_reboot(int magic1, int magic2,
2     unsigned int cmd, void __user * arg)
{
4   if (current->euid != current->losseceuid &&
        !capable(CAP_SYS_BOOT))
6         return -EPERM;
[...]
```

Abbildung 7.1: Sprachliche Erweiterungen der Compilerwerkzeuge

- Sprachunabhängigkeit

  Abgesehen von den Eigenheiten des verwendeten Compilers sollen die Werkzeuge sprachunabhängig sein. Das bedeutet, dass verschiedene Quellsprachen analysiert werden können, bzw. dass das Werkzeug für andere Sprachen anpassbar ist. Gezeigt wird dies am Ende dieses Kapitels durch die Analyse von .NET Intermediate Language Programmen mit dem selben Tool, mit dem auch C-Quelltexte analysiert werden. Genaue Informationen finden sich dazu im Abschnitt 7.4 über COMA

- Modularität und Erweiterbarkeit

  Die Beschreibung zur Methode TANAT stellt den momentanen Stand der Entwicklung dar. Die Werkzeuge zur Unterstützung der Vorgehensweise sollen bei Verbesserungen und Änderungen der Methode anpassbar sein. Das entwickelte Werkzeug ist beispielsweise bei der Verwaltung von T_{AV}-Bäumen so flexibel, dass jederzeit neue logische Verknüpfungen eingebunden werden können. Zudem sind die Berechnungen der Bäume, etwa die Risikoberechnung in einzelnen Modulen verankert, die durch eigene spezielle Berechnungsmethoden ausgetauscht werden können.

7.2 Konstruktion und Analyse von Aufrufgraphen

Zur Untersuchung der Abläufe in zu analysierenden Quelltexten ist es sehr hilfreich, die Zusammenhänge zwischen einzelnen Funktionen zu verstehen. Im Wesentlichen ist dabei interessant, welche Funktion aufgerufen wird und welche die aufrufende Funktion ist. Ziel ist es, bei der Analyse einer speziellen Funktion einen Aufrufgraphen zu entwickeln, an dem abgelesen werden kann, welche Funktionen von dort aus aufgerufen werden können.

7.2.1 Entwicklung der Aufrufgraphen

In der Phase zur Schwachstellenanalyse werden, wie im letzten Kapitel beschrieben, sowohl Speicherplätze, als auch alle Funktionen durch eine feste Anzahl von Fragen auf Schwachstellen hin abgeprüft. Dabei sind sowohl die Menge der Funktionen interessant, die von außen (vom potentiellen internen oder externen Angreifer, also einem Benutzer oder einer von anderen Prozessen verwendbaren Funktion) aufgerufen werden können, als auch die Funktionen, die von dort aus weiter angesprungen werden.

Bei der beschriebenen Angriffsanalyse werden Funktionen als Angriffspunkte für externe und interne Angreifer angesehen. Dabei ist auch wichtig, dass wiederum alle aufgerufenen Funktionen mit betrachtet werden. Wird ein möglicher Angriff entdeckt, so stellt die Kette der Funktionsaufrufe den Pfad im T_{AV}-Baum dar.

7.2.1.1 Manuelle Erzeugung

Bei der manuellen Analyse von Quelltext wird anhand der Funktionsstruktur des Programms vorgegangen. Beginnend mit der Hauptfunktion, die beim Programmstart ausgeführt wird, werden alle aufgerufenen Funktionen betrachtet. Dabei werden Aufrufgraphen erzeugt, die meist nicht dokumentiert werden. Zudem ist diese Analyse fehleranfällig. Da es für jede Programmiersprache stets möglich ist, die Aufrufgraphen automatisch zu erstellen, wird dazu ein Werkzeug entwickelt.

7.2.1.2 Automatische Generierung

Das Werkzeug basiert auf dem COMA-Framework, das auch im letzten Abschnitt dieses Kapitels vorgestellt wird. COMA bietet unter anderem die Möglichkeit, bei gegebener Grammatik einer Sprache alle Elemente eines analysierten Programms zu finden. Wie in den ersten drei Compilerphasen erzeugt es dazu einen abstrakten Syntaxbaum. Damit ist es möglich, aus einem gegebenen Programm alle Funktionen und darin alle Funktionsaufrufe herauszusuchen. Die funktionalen Beziehungen des Programms können dann für die Schwachstellen- und Angriffsanalyse verwendet werden. Neben den Funktionsnamen

werden alle Elemente, die auch im Quelltext sichtbar sind, bei der Analyse mitgeliefert. So können auch globale und lokale Variablen, alle Quelltextpositionen und sämtliche Programmkonstrukte identifiziert werden.

Im Rahmen dieser Arbeit wurde darauf verzichtet, Grammatiken für mehrere Programmiersprachen in COMA zu entwickeln. Da der Quelltext des hier eingesetzten Betriebssystemkerns in C geschrieben ist, wird die automatische Generierung von Aufrufgraphen nur für die Sprache C exemplarisch gezeigt. Aber es gilt heute auch für die meisten anderen Betriebssystemkerne, dass sie im Wesentlichen aus C und kleinen Teilen Assembler programmiert werden. Dies liegt vor allem daran, dass C sehr maschinennahe Programmierung zulässt und beispielsweise Speicherverwaltungen durch Pointerprogrammierung entwickelt werden kann. In anderen Hochsprachen wie Java oder C++ liegt die Speicherverwaltung dagegen zu einem gewissen Teil in der Aufgabe des Laufzeitsystems und kann nur umständlich selbst beeinflusst werden. Zur Unterstützung anderer Sprachen müsste lediglich die Grammatik für COMA bereitgestellt werden.

Als Beispiel zur Generierung der Aufrufgraphen dient ein einfaches Programm aus Abbildung 7.2. Dieses einfache Beispielprogramm wird im Folgenden mit dem COMA Framework analysiert.

```
typedef int integer;

int function1(integer i) {
  return(function2(i+1));
}

int function2(integer i) {
  return i+2;
}

int main(int argc,char* argv[]) {
  function1(1);
  function2(2);
}
```

Abbildung 7.2: Einfaches Beispiel zur automatischen Erzeugung der Aufrufgraphen

In dem Beispielprogramm werden drei Funktionen definiert, deren Aufrufstruktur direkt ablesbar ist. Für dieses einfache Beispiel wäre die manuelle Analyse leicht durchführbar, somit kann das Ergebnis des Werkzeugs direkt verstanden werden, das in Abbildung 7.3 gezeigt wird. Die Abbildung zeigt die vollständige Darstellung der Analyseausgabe mit allen Ergebnissen. Wichtigstes Ergebnis der Analyse sind die Funktionszusammenhänge in den Zeilen 49–54. Die anderen Ausgaben werden für eine genauere Analyse der Auf-

rufabhängigkeiten eingesetzt. Sie zeigen alle wesentlichen Attribute des abstrakten Syntaxbaums und sind für die Sicherheitsanalyse wichtig. An ihnen können beispielsweise Sichtbarkeits- und Gültigkeitsbereiche von Variablen festgestellt werden, die für die Analyse von Pufferüberläufen nötig sind.

```
Starting preprocessor ... Test_Suite/test.c
2
Modelling
4 ---------
Building base Model ...ok
6 Build IdentifierList Defining ...ok
Build IdentifierList Using ...ok
8 FunctionViewList
==========================
10
Analysis
12 --------

14 0. [[function1|Test_Suite/test.c|3]] DEFINING
  Global Variable
16   NONE
  Local Variable
18   0. [[i|Test_Suite/test.c|3]] DEFINING
       Type: (NAMETYPE | NOQUALIFIER | NOSTORAGE ::
20       [[integer|Test_Suite/test.c|3]] USING)
   Expressions
22   0. ((NAME | [[function2|Test_Suite/test.c|4]] USING) ::
       CALL | List Type : EXPRESSION Number of Elements : 1)
24 1. [[function2|Test_Suite/test.c|7]] DEFINING
  Global Variable
26   NONE
  Local Variable
28   0. [[i|Test_Suite/test.c|7]] DEFINING
       Type: (NAMETYPE | NOQUALIFIER | NOSTORAGE ::
30       [[integer|Test_Suite/test.c|7]] USING)
   Expressions
32   0. ((NAME | [[i|Test_Suite/test.c|8]] USING) |
       (CONSTINT | [1]) :: ADD)
34 2. [[main|Test_Suite/test.c|11]] DEFINING
  Global Variable
36   Local Variable
   0. [[argc|Test_Suite/test.c|11]] DEFINING
38       Type: (INT | NOQUALIFIER | NOSTORAGE)
   1. [[argv|Test_Suite/test.c|11]] DEFINING
40       Type: ATTR (POINTER | NOQUALIFIER | NOSTORAGE ::
       ATTR (POINTER | NOQUALIFIER | NOSTORAGE ::
42       (CHAR | NOQUALIFIER | NOSTORAGE)) | (VOID))
   Expressions
44   0. ((NAME | [[function1|Test_Suite/test.c|12]] USING) ::
       CALL | List Type : EXPRESSION Number of Elements : 1)
46   1. ((NAME | [[function2|Test_Suite/test.c|13]] USING) ::
       CALL | List Type : EXPRESSION Number of Elements : 1)
48
Summary
50 -------
function1:[[4] CALL function2(1)]
52 function2:[]
main:[[12] CALL function1(1)]
54 main:[[13] CALL function2(1)]
```

Abbildung 7.3: Analyse der Aufrufgraphen nach Beispiel 7.2

In der Ausgabe werden in Zeile 11–47 die kompletten Informationen des aufgebauten, abstrakten Syntaxbaums für die Sprachelemente

1. globale Variablen,
2. lokale Variablen und
3. die Anweisungen (Expressions)

für die einzelnen Funktionen ausgegeben. Dazu werden alle Attribute der Sprachelemente mit ausgegeben, wie die Datentypen, Namen, Speicherart oder definierendes oder benutzendes Identifikatorvorkommen. Diese detailierten Ergebnisse werden danach ausgewertet. Die Zusammenfassung der Analyse in Zeile 49–54 reicht für das Erzeugen der Aufrufgraphen aus.

7.2.2 Bewertung der Aufrufgraphen

Aus der gezeigten Zusammenfassung wird daraufhin ein Abstract State Machines Modell erzeugt. Dieses wird für das Beispiel aus Abbildung 7.2 in Abbildung 7.4 gezeigt. Die C-Funktionen werden als Zustände interpretiert, die in der ASM-Spezifikation durch den Zustand in `STATE` angezeigt werden. Der große Vorteil der abstrakten Darstellung im ASM-Modell liegt in der einfachen Verfeinerung der Modelle. Durch diese Darstellung können die Modelle zunächst sehr grob ausgegeben werden und anschließend an definierten Stellen des Codes verfeinert werden. Dadruch reduziert sich der Aufwand der Code-Analyse auf ausgesuchte Stellen.

```
if STATE=function1 then {
  SEQ: {
    STATE=function2(i+1)
    STATE=main
  }
}

if STATE=function2 then {
  STATE=main
}

if STATE=main then {
  SEQ: {
    STATE=function1(1)
    STATE=function2(2)
  }
}
```

Abbildung 7.4: Aufrufgraph des Beispiels als ASM dargestellt

Bei der Analyse anhand der generierten ASM-Modelle werden die Sprüngen zu den verschiedenen Funktionen durchgerechnet. In jeder erreichten Funktion wird dann der entsprechende Teil der Sicherheitsanalyse durchgeführt. Dabei können sehr lange Sprungsequenzen auftreten, die ab einer bestimmten Tiefe aus praktischen Gründen abgebrochen werden müssen. Ein Grund dafür ist der Aufwand bei der Analyse langer Aufruferketten, da die Möglichkeiten der Programmverzweigungen mit der Tiefe zunimmt und nicht alle Verzweigungen durchgesucht werden können. Für die Analyse bedeutet dies allerdings, dass eine mögliche Schwachstelle übersehen werden kann, die sich in weiteren Aufrufen verbirgt. Dabei gilt es jedoch zu beachten, dass auch ein Angreifer, der diese Schwachstelle ausnutzen will, die Analyse durchführen muss. Im Fall der in dieser Arbeit analysierten Pufferüberläufe war die Aufruftiefe nicht größer als drei. Der Programmablauf vom Funktionsaufruf mit dem entsprechenden Argument bis zur Kompromitierung des Systems ging demnach höchstens über zwei Funktionsaufrufe. In weiteren Arbeiten muss bei bekannten Angriffen untersucht werden, bis zu welcher Aufruftiefe der Angreifer das Programm analysieren musste, um den Angriff erfolgreich durchzuführen.

Diese Eindringtiefe in Funktionen sollte zu Beginn der Analyse festgelegt werden und hängt wesentlich von der Komplexität des Systems ab. Es sind keine Ansätze für Verfahren bekannt, die die Eindringtiefe automatisch reduzieren beziehungsweise Hinweise für einen passenden Wert geben können. In späteren Modellen in ASM-Syntax werden der Übersichtlichkeit halber stets nur die wichtigen Teile der Programmstrukturen dargestellt, auch wenn die Darstellungstiefe noch viel feiner bis hin zur Implementierung gehen könnte.

7.3 Synthese der Bedrohungsbäume

Die T_{AV}-Bäume werden bei der Analyse schnell sehr groß und unhandlich. Es ist naheliegend, dass man geeignete Werkzeuge für deren Konstruktion und Verwaltung benötigt. Zudem ist die Syntax der hier vorgestellten Bäume aufwändiger, als die bekannter Bedrohungsbäume, da unter anderem zusätzliche Symbole und spezielle Strukturen eingesetzt werden. Im folgenden Abschnitt werden bekannte Werkzeuge vorgestellt und anschließend einige wichtige Ausschnitte aus dem entwickelten Werkzeug für die Methode TANAT beschrieben.

7.3.1 Verwandte Werkzeuge

Neben der Software zur Entwicklung und Verwaltung von Bedrohungsbäumen, die für diese Arbeit entwickelt und im Folgenden vorgestellt wird, existieren einige andere Tools, die für diesen Zweck in der Praxis eingesetzt werden können. Tatsächlich existieren nur wenige Werkzeuge für diesen Zweck, die im Folgenden kurz vorgestellt werden. Für einen Einsatz der Bedrohungsbäume wird schnell klar, dass ein spezielles Tool notwendig ist, dass im

Anschluss besprochen wird.

7.3.1.1 Allgemeine Tools

In der Praxis werden Bedrohungs- und Angriffsbäume meist mit einfachen Mittel, wie etwa Zeichenprogrammen und Visualisierungstools erstellt. Oft ist den Anwendern nicht klar, dass dafür spezielle Software existiert, naheliegender Weise wird das bereits erlernte Zeichenprogramm für die Visualisierung verwendet. Die dabei entstehenden Nachteile werden im Folgenden kurz erläutert.

Als Beispiele seien hier die verbreiteten Programme `Visio`, `xfig` und `graphviz` genannt. Wegen der sehr eingeschränkten Möglichkeiten, die ein Standardkonstruktionstool für Bäume und deren Attributierung bietet, werden folgende Funktionen im Allgemeinen von allen Vertretern nicht angeboten.

Bedrohungsbaum-Bibliotheken

Einzelne Teilbäume können als Grafik abgespeichert und in andere Zeichnungen eingebunden werden. Eine Neuberechnung der Attribute, die Neupositionierung der einzelnen Knoten und eine automatische Benennung der Knoten innerhalb der Hierarchie des Baums sind allerdings nicht möglich. Die Bibliotheken sollen als Textdateien abgespeichert werden, die auch anderweitig generiert und eingesetzt werden können.

Baumattributierung

Standardtools bieten keine Möglichkeit, einzelne Knoten oder Äste eines Baums mit Attributen zu versehen. Das manuelle Anbringen der Attribute ist bei großen Bäumen sowohl sehr arbeitsaufwändig, als auch fehleranfällig und muss bei Wiederverwendung eines Teilbaums immer wieder komplett durchgeführt werden.

Risikoberechnung

Die Errechnung des Gesamtrisikos aus den Einzelrisiken der angegebenen Unterbäume ist nur manuell möglich und wird durch reine Zeichensoftware nicht unterstützt. Programme zur Visualisierung von Graphen, wie etwa `graphviz` können zwar darauf angepasst werden, die Berechnung selbst müsste aber dafür neu programmiert werden.

Übernahme generierter Bäume

Wünschenswert ist die Übernahme von Bedrohungsbäumen, die bei einer Bedrohungsanalyse beispielsweise durch bekannte Angriffe oder vorherige Analysen automatisch oder teilautomatisch generiert wurden. Hierfür müssten die Bäume in das Programmspezifische Format gewandelt werden, was umständlich, aber sicherlich möglich wäre. Wesentlich aufwändiger ist die Tatsache, dass dieser Konverter auch die Plazierung der einzelnen Knoten und Kanten durchführen muss.

Weiterverwendung generierter Bäume

Neben der Übernahme ist auch der Export der generierten Bedrohungsbäume für die Durchsetzung von Sicherheitsmaßnahmen wichtig. Denkbar ist der Einsatz der Bedrohungsbäume aus der präventiven Analyse heraus im Betrieb eines Systems für angriffserkennende und reaktive Maßnahmen, wie beispielsweise die Konfiguration von Einbruchserkennungssystemen.

Ein Standard-Zeichentool eignet sich zwar für das ausschnittsweise Darstellen von kleinen Bäumen, nicht aber für die Konstruktion, die in einem Analyseprozess gefordert ist. Einziger Vorteil ist die Handhabbarkeit eines bereits bekannten Programms und die Tatsache, dass meist keinerlei Einarbeitungsaufwand geleistet werden muss.

SecurITree

Als eines der wenigen verfügbaren Produkte zur komfortablen Erstellung von Angriffsbäumen existiert das Programm `SecurITree` von Amenaza Technologies, Inc. Das in Java entwickelte, kommerzielle Tool stellt erzeugte Bäume grafisch dar und verwendet ein proprietäres Format zur Speicherung und eventuellen Weiterverwendung der generierten Bäume. Die Bäume können durch numerische Attribute erweitert werden, durch die später die Risiken berechnet werden. Als logische Verknüpfungen der Unterbäume und Blätter werden ausschließlich `UND` und `ODER` Verbindungen angeboten, eine genauere Differenzierung, wie etwa das ausschließende `NOT`, ist nicht möglich.

Die Darstellung der Bäume wird durch Farbe für die jeweiligen Teilbaumarten (`UND`, `ODER`) unterstützt. Die grafische Darstellung, insbesondere bei größeren Bäumen könnte dadurch verbessert werden, dass die Unterbäume horizontal zentriert werden, um weniger Platz zu belegen.

In der Dokumentation und in den verfügbaren Beispielen wird im Wesentlichen nicht zwischen den Begriffen Bedrohung und Angriff unterschieden, SecurITree soll explizit nur zur Entwicklung von Angriffsbäumen eingesetzt werden.

Als Vorgehen wird ein einfacher Prozess vorgeschlagen, der aus den folgenden fünf Schritten

(a)–(e) besteht.

(a) Erstellung eines Angriffsbaums: Die grafische Erstellung der Bäume wird nicht näher beschrieben, sie ist als rein intuitiver Prozess vorstellbar. Es kann zwischen Angriffswegen unterschieden werden, die mehrere Baumpfade als notwendige Voraussetzung für einen Angriff erfordern (`UND`-Pfad), und zwischen alternativen Angriffswegen (`ODER`-Pfad).

Scheint ein Angriffsschritt in einem Blatt als genau genug für die Analyse, so wird er als atomar bezeichnet und nicht weiter verfeinert. Es werden allerdings weder Kriterien für eine sinnvolle, nachvollziehbare Abbruchbedingung angegeben, noch wird eine strukturierte Entwicklung des Baums unterstützt.

(b) Mögliche Angriffe identifizieren: Zur Identifizierung möglicher Angriffe werden einzelne Bewertungsfaktoren hinzugezogen. Diese sind frei wählbar, Beispiele sind etwa die notwendigen Ressourcen, die ein Angreifer für einen erfolgreichen Angriff benötigt, wie Zeit, Geld, technisches Wissen und die Gefahren durch mögliche Strafen. Diese einzelnen Attribute werden mit einem natürlichen Zahl bewertet und geben Anhaltspunkte für die Wahrscheinlichkeit der einzelnen Wege innerhalb des Baums vom Blatt zur Wurzel.

(c) Folgen einzelner Angriffe abschätzen: Das Erkennen der Folgen eines einzelnen Angriffs ist nur durch eine Analyse von Angriffsbäumen in Kombination mit der eigentlichen Anwendung oder den Einsatzszenarien möglich. Softwareunterstützung ist dafür nur in geringem Maße möglich, weil die Szenarien meist nur informell beschrieben werden und die Analyse nur in Spezialfällen durch Algorithmen durchgeführt werden kann.

(d) Risiko einzelner Angriffe abschätzen: Die Risikobewertung erfolgt durch Berechnung der einzelnen Teilrisiken innerhalb des Baums. Sind die Söhne eines Knotens `UND`-verknüpft, dann ist das Gesamtrisiko das kleinste der Teilrisiken allen Unterbäume. Bei `ODER`-verknüpften Teilbäumen errechnet sich das Gesamtrisiko durch die Bestimmung des größten Risikos aller Unterbäume. Die einzelnen Risiken lassen sich so direkt aus dem Baum ablesen.

(e) Angriffserkennung: Dieser angedachte Teil des Vorgehens liegt im Bereich der reaktiven Sicherheitsmaßnahmen und verwendet die erzeugten Angriffsbäume zur automatischen Erkennung von Angriffen auf ein System. Es exitieren keinerlei Informationen, wie die Umsetzung der Angriffsbäume auf konkrete Erkennungs- und Abwehrmechanismen durchzuführen ist. Insbesondere wird nicht diskutiert, welche Arten von Angriffen überhaupt derart dynamisch beschrieben und erkannt werden können.

Fazit

Zur Konstruktion von Angriffsbäumen und der anschließenden Berechnung der resultierenden, möglichen Bedrohungen ist derartige Softwareunterstützung wichtig. Die Nachteile dieses angesprochenen Tools resultieren im Wesentlichen aus der Tatsache, dass die Software nicht für ein eigenes Vorgehen angepasst und erweitert werden kann. So kann weder die mangelhafte Darstellung der entwickelten Bäume verbesert, noch eine eigene Berechnungsmethode angewendet werden. Dadurch, dass die Angriffsbäume in einem speziellen proprietären Format abgelegt werden, ist eine spätere Verwendung für andere Zwecke, etwa zur Dokumentation oder der Konfiguration eines Sicherheitsmanagements, schwierig.

7.3.2 Baumaufbau

Im Folgenden wird das Werkzeug TANAT beschrieben, das die Methode TANAT unterstützt. Die Programmstruktur des Werkzeugs lässt sich wie folgt gliedern.

- Importmodule

  Der Datenimport zum Einlesen von Baumdaten wird in einem Modul realisiert, das sich selbständig um die Verarbeitung kümmert und die Datenstrukturen der Baumverwaltung füllt. Damit können leicht neue Module für andere Baumbeschreibungen integriert werden, beispielsweise für den Import anderer Programmformate. Momentan ist nur ein Importmodul implementiert, das die TANAT-eigene Beschreibungssprache versteht und für das einfache Laden ganzer Bäume gedacht ist.

- Exportmodule

  Momentan sind drei Exportmodule implementiert, die sowohl das TANAT-eigene Format, als auch LaTeX-Format und `.csv`-Dateien (eine Abkürzung für Comma Separated Value) erzeugen können. Es können beliebige Exportmodule realisiert werden, die mit der Datenstruktur der Baumverwaltung zusammenarbeiten. Neben dem Export ganzer Bäume ist ebenso der Export von Teilbäumen oder generierten Bäumen möglich.

- Berechnungsmodule

  Die Kanten und Knoten, die im Baum gespeichert werden, sind generische Objekte, die beliebig erweiterbar sind. Die Berechnungen innerhalb des Baums werden daher von den Methoden innerhalb der spezialisierten Objekte durchgeführt. Da diese Objekte von jedem beliebig erweiterbar sind und bereitgestellt werden können, werden die Methoden zur Berechnungen in eigenen Modulen integriert, die auch für jede beliebige Erweiterung der Kanten und Knoten verwendbar sind.

- Benutzungsschnittstelle

  Die Benutzungsschnittstelle wurde mit der GTK+ Bibliothek erstellt und kann in folgende Oberflächenelemente eingeteilt werden:

  - Menü, Menüleiste mit grafischen Symbolen und Statusbereich
  - Zeichenfunktionen mit Koordinaten- und Zoombereichsanzeige
  - Baum-Bibliotheksfunktionen mit hierarchischer Baumdarstellung
  - Zeichenbereich, Vergrößerung frei einstellbar
  - Attributfenster zur Darstellung der definierten Attribute und Berechnungsmodule
  - Eingabe- und Dialogbereich

- Baumverwaltung

  Die Datenstruktur und die vorhandenen Methoden zur Verwaltung der T_{AV}-Bäume sind ein zentraler Bestandteil des Werkzeugs und werden deshalb später genauer erklärt.

- Bibliotheksverwaltung

  Ausgewählte Teilbäume können als eigenes Bibliothekselement persistent abgespeichert werden. Dabei wird der Teilbaum im selben Format abgelegt, wie komplette T_{AV}-Bäume. Die gespeicherten Elemente der Bibliothek können an beliebiger Stelle in andere Bäume übernommen werden. Mit abgespeichert werden neben Beschreibungen und Attributen auch die jeweiligen Attributauswertungen beziehungsweise Berechnungsmethoden.

Zur Verwaltung der T_{AV}-Bäume wurde eine generische Baumdatenstruktur entwickelt, die für beliebige Erweiterungen nutzbar ist. Der Baumaufbau erfolgt durch eine Reihe festgelegter Operationen, die Methoden zur Berechnung der Attribute können beliebig festgelegt werden.

Datenstruktur

Die Datenstruktur zur Verwaltung der Bäume kennt nur zwei Basisklassen, die Kanten und Knoten repräsentieren. Alle Elemente der T_{AV}-Bäume werden in diesen Klassen verwaltet. In Abbildung 7.5 werden die beiden leicht verkürzten Klassenspezifikationen mit den Signaturen der Methoden dargestellt. Das Verständnis der Datenstruktur ist dann notwendig, wenn eigene Berechnungsmodule für spezielle Attribute oder spezielle Baumtransformationen programmiert werden sollen. Der Zugriff dieser externen Module auf die zentrale Baumdatenstrukturen geschieht durch diese Programmierschnittstelle.

```
class Vertex {
protected:
  std::deque<class Edge> edgeDeque;
public:
  std::deque<class Edge>* getEdgeList (void) throw();
  Vertex* addNewVertex (void) throw();
  void addVertex (Vertex*) throw();
  Edge* addNewEdge (void) throw();
  void addEdge (Edge*) throw();
  Vertex* removeEdge (Edge*) throw();
  void removeVertex (Vertex*) throw();
  Edge* lookupEdge (Edge*) throw();
  Vertex* lookupVertex (Vertex*) throw();
};

class Edge {
protected:
  class Vertex* vertex[2];
public:
  Vertex* getVertex(int) throw();
  void setVertex(int, Vertex*) throw();
  void setVertex(Vertex*) throw();
  Vertex* addNewVertex(void) throw();
  Vertex* addNewVertex(int) throw();
  int removeVertex(int) throw();
  int removeVertex(Vertex*) throw();
  void swapVertex(void) throw();
};
```

Abbildung 7.5: Einfache Darstellung der beiden Klassen zur Verwaltung der Baumstruktur

Jeder Knoten (engl. Vertex) enthält eine Liste aller ihm zugeordneter Kanten (engl. Edge), die in einer Double Ended Queue (deque) abgelegt sind, jede Kante bindet stets an zwei Knoten. Die Operationen zum Erzeugen und Verbinden (Präfix `add...`), Entfernen und Suchen sind realisiert und arbeiten damit auf den erzeugten Strukturen auf der Halde. Die Menge aller Knoten und Kanten werden in einer übergeordneten Klasse `class Tree` gespeichert, dort wird auch ein ausgezeichneter Wurzelknoten definiert. Zyklen sind nach der Klassenstruktur zwar möglich, werden aber bei der Erzeugung dadurch vermieden, dass bei der Kantenzuordnung die bisherige Belegung überprüft wird. Möglich ist aber eine Verbindung zwischen zwei Knoten durch mehrere Kanten. Das Löschen von Kanten ist nur dann möglich, wenn keine unverbundenen Teilbäume entstehen. Die Datenstruktur wird direkt im Zeichenbereich dargestellt.

Attribute

Ausgehend von diesen Basisklassen werden die spezialisierten Knoten und Kanten abgeleitet bzw. vererbt. Die Berechnungsfunktionen sind virtuelle Methoden der Basis, die über die Strukturen der angrenzenden Teilbäume Werte berechnen. Attribute werden in Knoten und Kanten gespeichert. Abhängig von der Funktion der Attribute werden konkrete Werte nur in bestimmten Baumelementen abgelegt. Die Berechnung der Attribute bedeutet dabei, dass die fehlenden Attribute durch Auswertung aller Teilbäume berechnet und in allen Baumelementen gespeichert werden. Beispiel hierfür wäre etwa die Berechnung der Risiken bestimmter Bedrohungen. Es gibt keine Beschränkung für den Einsatz der Attribute. Diese können prinzipiell alles darstellen, was mit Klassen darstellbar ist.

Das Werkzeug stellt eine Bibliothek zur Verwaltung der T_{AV}-Bäume zur Verfügung. Diese kann an eigene Berechnungsmodelle und Anforderungen angepasst werden, was einleitend im Anhang A beschrieben wird.

7.3.3 Aufwand und Einschränkungen

Für die Konstruktion und Verwaltung großer Bedrohungs- bzw. T_{AV}-Bäume ist ein Werkzeug wie das hier vorgestellte notwendig. Der Aufwand, bestehende Bäume in das eingesetzte Format zu konvertieren, hängt vom bisher verwendeten Format ab, kann aber automatisch geschehen. Durch die Möglichkeit, Baumbibliotheken zu verwalten, sinkt der Aufwand für die Erzeugung neuer Bäume durch Wiederverwendung. Einschränkend ist die bisherige Unterstützung von Mustervergleiche, das Auffinden gleicher Teilbäume ist deshalb noch nicht implementiert. Damit würden mindestens zwei Vorteile geschaffen. Erstens könnten Teilbäume auf bereits vollständig entwickelte Bäume in der Bibliothek hin geprüft werden und so Hinweise zur Übernahme bestehender Strukturen gegeben werden, womit die Entwicklungszeit weiter verkürzt werden kann. Zweitens können wiederkehrende Teilbäume innerhalb eines Graphen erkannt werden um redundante Entwicklungsschritte zu vermeiden.

7.4 Code-orientierte Analyse

Eine Hauptaufgabe zur Entwicklung sicherer Software ist die Gewährleistung von angegebenen Bedingungen. Dafür existieren verschiedene Strategien, wie etwa formale Spezifikation und Verifikation oder Modelchecking. Diese Methoden sind allerdings meist auf die Neuentwicklung von Systemen ausgelegt, so dass während dieser Arbeit ein Projekt gestartet wurde, das Prüfungen in bestehendem Quelltext integrieren kann. Dabei wird der Quelltext vollständig analysiert, die formulierten Bedingungen werden möglichst automatisch in den Quelltext eingebaut und nach der Neuübersetzung werden diese Bedingungen zur Laufzeit

erfüllt beziehungsweise sie lösen Fehler aus.

Kernstück dieses Projektes ist ein Werkzeug zur Analyse von Quelltexten, wobei jede kontextfreie Sprache verwendet werden kann. Dieses Tool muss die Quellform in eine im Sinne einer Übersetzung vollständige Repräsentation überführen. Diese Repräsentation kann durch ein zur Verfügung gestelltes Framework analysiert und verändert werden und anschließend bei Bedarf wieder in den Quelltext zurückübersetzt werden. Da verschiedenste komplexe Probleme mit diesem Werkzeug gelöst werden können, ist es erforderlich, dass verschiedene Granularitäten der Repäsentation vorhanden sind.

Das entwickelte COMA Framework (Code orientierte Modellierung und Analyse) bietet die Möglichkeit, ein Modell einer kontextfreien Sprache zu entwickeln. Auf Basis dieses Modells können dann beliebige Quelltexte dieser Sprache untersucht werden. Aus dem Quelltext werden dazu sogenannte Entitäten-Prädikaten-Modelle erzeugt. Diese können in verschiedenen Granularitätsklassen vorliegen. Das Basismodell entspricht dem vollständigen Wissen aus dem Quelltext und ist für die spezielle Problemlösung meistens zu komplex. Sollen beispielsweise alle abhängigen Funktionen einer Variablen x gesucht werden, so ist für das Ergebnis die Realisierung der einzelnen Funktionen unwichtig. Das Lösungsmodell abstrahiert deshalb derart vom Basismodell, dass es für das Problem genau angepasst ist. Sowohl Strukturen, als auch die Inhalte der beiden Modelle können sich deutlich unterscheiden. Da der Schritt zwischen Basis- und Lösungsmodell unter Umständen groß ist, werden sogenannte Zwischenmodelle eingeführt, die die Transformation in beide Richtungen erleichtern. Lösungsmodelle enthalten genau die syntaktischen und semantischen Informationen, die der Benutzer des Frameworks untersuchen und prüfen will. Dient beispielsweise das Lösungsmodell zur Darstellung aller globalen Variablen, so wird es eine listenartige Struktur besitzen. Sollen dagegen Aufrufgraphen analysiert werden, so wird es eher graphenartige Strukturen haben.

Die Modelle werden durch C++-Objektstrukturen im Speicher dargestellt. Das hat zwei wesentliche Vorteile. Ersten sind in den Objekten die möglichen Operationen enthalten und zweitens können die unterschiedlichen Objektklassen voneinander abgeleitet werden. Die inneren Knoten des Basismodells werden durch die Entitäten beschrieben, die äußeren Knoten durch die Prädikate. Im Prinzip sind alle weiteren Elemente von diesen beiden Typen abgeleitet. Aus praktischen Gründen wird eine Oberklasse `Object` definiert, von der neben den beiden Basistypen aus noch die Klasse `Value` abgeleitet wird. `Value` beschreibt alle eingebauten Datentypen der Sprache, wodurch der praktische Umgang mit dem Framework vereinfacht wird. Die weiteren Vererbungsschritte unterteilen sich in sprachunabhängige und sprachabhängige Klassen. Details dazu können sowohl in der Dokumentation zu COMA, als auch in [Sch03] nachgelesen werden. Einen Überblick über die grundlegende Funktionsweise von COMA gibt Abbildung 7.6.

Bei der Implementierung des COMA-Frameworks wurden die Entität-Prädikat-Strukturen zweier Sprachen (der Sprache C und der Intermediat Language der Common Language Infrastructure) spezifiziert, sowie Scanner und Parser dafür implementiert. Damit wurden

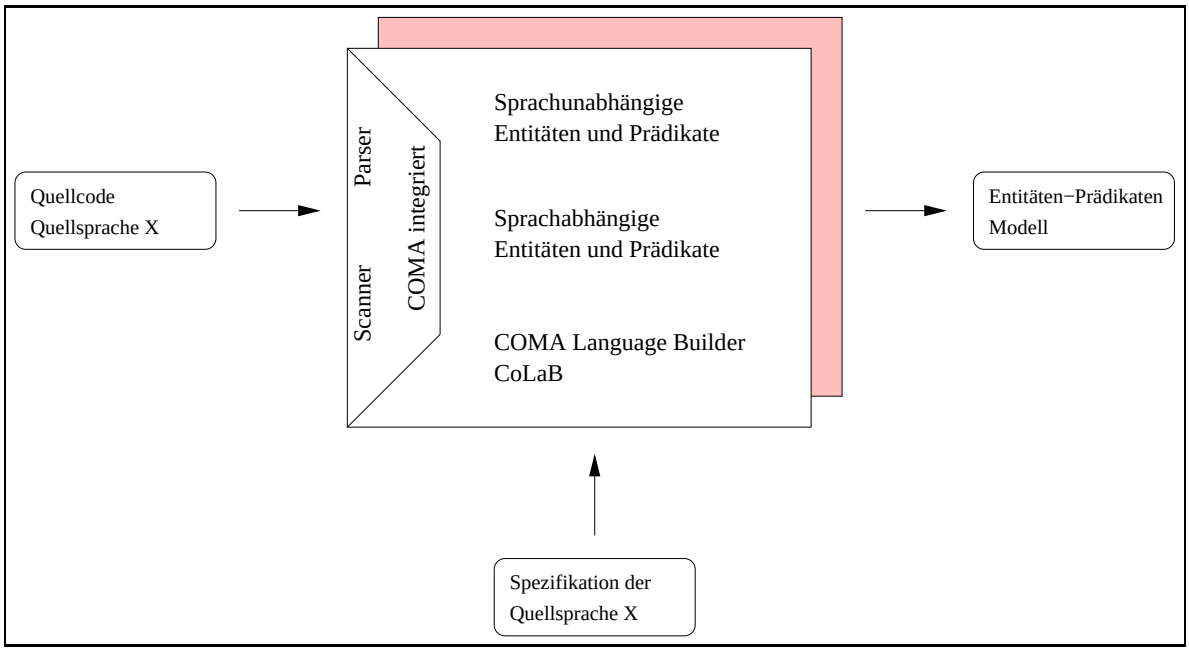

Abbildung 7.6: Überblick über die COMA-Architektur

zwei Anwendungen umgesetzt, die im Folgenden beschrieben werden.

7.4.1 Analyse von C-Quelltext

Für die Sprache C existieren viele verfügbare Grammatikbeschreibungen. Es liegt daher nahe, für die COMA-Sprachspezifikation eine dieser Beschreibungen zu verwenden. Eine Besonderheit der Spezifikation der Entitäten-Prädikaten-Struktur für die Sprache C ist allerdings, dass nicht sämtliche Produktionen der Grammatik in den sprachabhängigen Entitäten auftauchen. Das liegt daran, dass in der Grammatik Variablen-, Funktions- und Typedefinitionen durch eine einzige Produktion mit dem Name „Declaration“ dargestellt werden, um die Modellierung zu vereinfachen. Dadurch entsteht der Nachteil, dass die Implementierung des Parsers etwas komplizierter wird, der Umgang in COMA allerdings leichter und übersichtlicher ist.

Finden von Pufferüberläufen

Die Grundlagen zur Entstehung von Pufferüberläufen wurden bereits in Kapitel 2 dargestellt. Das COMA-Framework eignet sich für das Auffinden von Programmstellen im Quelltext, die potentielle Ursache für Pufferüberläufe sein können. Dazu müssen zwei Fälle unterschieden werden, wobei bisher nur C-Programme untersucht werden.

1. Pufferüberläufe auf dem Keller

   Kellerüberläufe haben als Ursache stets einen begrenzten Puffer, der lokal in einer Funktion gültig ist. Die Stellen, an denen lokale Speicherbereiche auf dem Keller definiert werden, können durch COMA leicht gefunden werden. Der Überlauf kann aber nur dann stattfinden, wenn tatsächlich Daten in den lokalen Puffer geschrieben werden, häufig durch eine Kopierfunktion wie `strcpy`. Diese können ebenfalls gefunden werden und auf die Verwendung des lokalen Puffers hin überprüft werden. Werden beide Bedingungen erfüllt, so kann an dieser Stelle ein Überlauf passieren. In Abbildung 7.8 wird der relevante Teil einer Analyse für das einfache Beispiel aus Abbildung 7.7 abgebildet. Die beiden Warnungen (Zeile 10 und Zeile 15) zeigen die Hinweise auf mögliche Pufferüberläufe. Ob diese Stellen tatsächlich für einen Angriff ausgenutzt werden können, kann nicht automatisch ermittelt werden, da die Benutzereingaben und andere Variableninhalte im Allgemeinen nicht bekannt sind.

```
#include<stdio.h>
2
char *str="Ein Stringplatz";
4
void a(int x) {
6   char lokalerPuffer[800];
   strcpy(ch,str,x);
8   printf("%s\n",ch);
}
10
int main() {
12   a(1);
   return 0;
14 }
```

Abbildung 7.7: Einfaches Beispielprogramm in C mit begrenztem lokalem Puffer

   Nicht ausgewertet wird momentan das Weiterreichen des Puffers an andere Funktionen, die dann ihrerseits den Überlauf erzeugen können. Die Erweiterungen dafür sind aber leicht durchzuführen, es muss lediglich nach der Verwendung des Puffer gesucht werden.

2. Pufferüberläufe auf der Halde

   Zur Erkennung der Haldenüberläufe werden alle dynamisch angelegten Speicherbereiche gesucht. Diese können in C-Programmen stets dadurch gefunden werden, dass ein Systemdienst für deren Speicherzuteilung nötig ist, der meist durch einen `malloc`-Befehl ausgelöst wird. Der eigentliche Überlauf geschieht dann durch eine Verwendung des Puffers ohne Prüfung der Puffergrenzen.

```
[...]
0.    [[a|/tmp/helloworld.c|5]] DEFINING
  Globale Variablen
  Lokale Variablen
  0.    [[x|/tmp/helloworld.c|5]] DEFINING Type:
        (INT | NOQUALIFIER | NOSTORAGE)
  1.    [[lokalerPuffer|/tmp/helloworld.c|6]] DEFINING Type:
        ATTR (CHAR | NOQUALIFIER | NOSTORAGE ::
        (CHAR | NOQUALIFIER | NOSTORAGE) | (CONSTINT | [800]))
Warning: local bounded puffer detected
        [lokalerPuffer|/tmp/helloworld.c|6]
  Expressions
  0.  ((NAME |  [[strcpy|/tmp/helloworld.c|7]] USING) ::
      CALL | List Type : EXPRESSION Number of Elements : 2)
Warning: strcpy uses [lokalerPuffer|/tmp/helloworld.c|6]
  1.  ((NAME |  [[printf|/tmp/helloworld.c|8]] USING) ::
      CALL | List Type : EXPRESSION Number of Elements : 2)
1.    [[main|/tmp/helloworld.c|11]] DEFINING
[...]
```

Abbildung 7.8: Gekürzte Ausgabe der Analyse von Pufferüberläufen

7.4.2 Analyse von Intermediate Language-Code

In einem Teilprojekt dieser Arbeit wurde versucht, Informationsflüsse innerhalb eines Intermediate Language-Programms zu kontrollieren, um den universellen Charakter von COMA zu demonstrieren. Dazu wurde als Beispiel für die Analyse einer anderen Sprache die Intermediate Language aus dem .NET Framework herangezogen. Diese Sprache stellt eine abstrakte Assemblersprache dar, in die sämtliche Hochsprachen übersetzt werden können. Momentan sind dies C/C++, C#, Visual Basic und eine spezielle Skriptsprache. Vergleichbar ist die IL damit mit dem Bytecode, der unter Java als abstrakter Assembler eingesetzt wird. Der Vorteil einer Zwischensprache liegt in der Plattformunabhängigkeit und in einfachen Mechanismen beim Einsatz unterschiedlicher Programmiersprachen in einem Projekt.

Die Grammatik der Intermediate Language des Common Language Infrastructure ist durch eine große Anzahl von Produktionen gekennzeichnet. Dies liegt im Wesentlichen daran, dass die Intermediate Language eine Assemblersprache einer virtuellen Stackmaschine mit objektorientierten Spracheigenschaften ist. Die Grammatik einer anderen Assemblersprache wird dafür um viele Eigenschaften erweitert. Deshalb wird auch die Entitäten-Prädikaten-Struktur entsprechend komplex. Dies macht sich im Wesentlichen an der Ausführungsgeschwindigkeit beziehungsweise dem Speicherverbrauch bemerkbar. Um eine Vorstellung zu vermitteln, wie die Syntax der Intermediate Language aussieht, wurde ein „Hello World“ Programm von C# übersetzt und anschließend mit einem Disassembler analysiert. Die

Ausgabe wird in Abbilung 7.9 gezeigt.

```
.class private auto ansi beforefieldinit 'MainApp'
       extends [mscorlib]System.Object
{

  // method line 1
  .method public hidebysig specialname rtspecialname
         instance default void .ctor () cil managed
  {
       .maxstack 8
       IL_0000: ldarg.0
       IL_0001: call instance void valuetype
                'System.Object'::.ctor()
       IL_0006: ret
  }

  .method private static hidebysig
         default void 'Main' () cil managed
  {
       // Method begins at RVA 0x20f4
       .entrypoint
       // Code size 11 (0xb)
       .maxstack 8
       IL_0000: ldstr "Hello World"
       IL_0005: call void class
                'System.Console'::'WriteLine'(string)
       IL_000a: ret
  }

}
```

Abbildung 7.9: Einfaches Beispiel eines Intermediate Language Programms

Ausgehend von den Möglichkeiten, die COMA mit der IL-Unterstützung bietet, wird im folgenden Abschnitt gezeigt, wie Zugriffskontrolle automatisch in IL-Programme integriert werden kann. Dieses Beispiel stellt die Mächtigkeit der Analyse von .NET-Code dar und zeigt eine Möglichkeit der automatisierten Anreicherung und Änderung von Code.

Zugriffskontrolle für Objekte

Ein Programm *Informationflow* enthält eine Klasse `Person`, welche die geschützten Felder `Name::String` und `PIN::int` besitzt. Außerdem existiert ein Namensraum, in dem die

beiden Klassen `UnsafeChannel` und `SafeChannel` definiert sind. Objekte dieser Klassen simulieren das Übertragen von Informationen.

Von `Person` wird eine Klasse `ExtPerson` abgeleitet, die um ein ungeschütztes Feld `info::int` und um zwei Methoden `computeInfo` und `getInfo` erweitert ist. Die Methode `computeInfo` kopiert `PIN` nach `info` genau dann, wenn `Name` „Alice“ ist. In der `Main`-Methode des Programms wird beim Ablauf ein Objekt der Klasse `ExtPerson` zufällig entweder mit dem Namen „Alice“ oder „Bob“ und einer PIN-Nummer erzeugt. Darauhin erzeugt das Programm eine sicheren oder unsicheren Kommunikationskanal (zufällig ausgewählt entweder durch die Klasse `SafeChannel` oder `UnsafeChannel`) und überträgt einen Wert für eine bestimmte Person. Damit werden zur Laufzeit wahlweise schützenswerte und öffentliche Daten über sichere oder unsichere Kanäle gesendet.

Ziel der Modifikation ist es, zu gewährleisten, dass Daten, die von dem Wert `PIN` abhängen, über einen sicheren Kanal übertragen werden. Dies wird durch Kontrollen zur Laufzeit sichergestellt, die durch COMA in den IL-Code des Programms integriert wurden. Dazu wurde das Programm `cilmodifier` entwickelt, das Konstruktoren, Methoden und Felder innerhalb Klassen durch eindeutige Namen kennzeichnet. Die Zugriffskontrolle erfolgt dadurch, dass die Elemente in Sicherheitsklassen eingeteilt werden. Im Laufe einer Ausführung wird ein globaler Sicherheitszustand erzeugt. Der Wert dieses Sicherheitszustands ist am Anfang nicht festgelegt. Durch den Zugriff auf ein Element wird zunächst überprüft, ob es einen Konflikt zwischen Sicherheitszustand und Sicherheitsklasse gibt. Wenn ein Konflikt vorliegt, so wird dieser unerlaubte Zugriff beendet. Im obigen Beispiel wäre dies eine Übertragung der PIN-Daten über einen unsicheren Kanal. Ist einem Element noch keine Sicherheitsklasse zugeordnet, so erhält es nach dem Zugriff den Wert des aktuellen Sicherheitszustandes.

Grundsätzlich hat man durch die Anreicherung des analysierten Codes Prüfstellen in den Code integriert. Diese Stellen können dann externe Funktionen zur Laufzeit aufrufen, die in dazugeladenen Modulen realisiert sind. Diese kontrollieren dann Zugriffe auf verschiedene Daten zur Laufzeit. Die eigentlichen Regeln für die Zugriffe sind in dem angesprochenen Beispiel recht einfach. Es ist aber leicht möglich, diese Policies zur Laufzeit in den externen Modulen zu konfigurieren und die Zugriffsregeln sogar dynamisch festlegen zu lassen.

7.5 Ergebnisse

Die Beschreibungen der Werkzeuge stellen den aktuellen Stand ihrer Entwicklungsstufen dar. Sie sind in der Praxis einsetzbar und erweiterbar. Für die einzelnen Tools können folgende Erweiterungen sinnvoll sein.

Konstruktion der Aufrufgraphen

Die Konstruktion der Aufrufgraphen kann momentan nur mit C-Quelltexten erfolgen. Die Anpassung an Sprachen wie C++ oder Java kann allerdings mit geringem Aufwand durchgeführt werden. Die Anpassung von COMA zur Code-orientierten Analyse von Intermediate Language-Code hat etwa 3 MT benötigt. Für andere Sprachen ist der Aufwand vergleichbar. Zudem könnte ein grafisches Werkzeug bei der Analyse der Funktionsaufrufe nützlich sein, das die Historie der Aufrufe mit Parametern und die unterschiedlichen Möglichkeiten präsentiert. Eine Integration dieser Analyse und dem Konstruktionswerkzeug für T_{AV}-Bäume ist noch nicht umgesetzt worden, würde aber den Aufwand der Quelltextanalyse verkleinern.

Synthese der Bedrohungsbäume

Momentan existieren nur einfache, grundlegende Beispiele zur Attributberechnung in T_{AV}-Bäumen. Für verschiedene Anwendung können weitere Berechnungsmodule entwickelt werden, wie etwa zur Risikoanalyse und den Kosten- und Nutzenberechnungen von Angriffen. Die Berechnung des Layouts der Bäume ist momentan einfach, da jeder Knoten des Baums eine konstante Größe besitzt. Eleganter wäre die Berechnung jeder Boxengröße und einer abgestimmten Anpassung des Layouts. Das Einlesen von Fremdformaten zur Speicherung von Bäumen ist sinnvoll, um Daten der Bedrohungsbäume in das Werkzeug zu importieren, und bisher nicht realisiert.

Code-orientierte Analyse

Analysierbare Quellsprachen sind bisher C und die Intermediate Language vom .NET-Framework. Eine Unterstützung anderer Sprachen, allen voran C++, wäre sinnvoll. C++ besitzt als objektorientierte Erweiterung von C die selben Schwachstellen und Sicherheitsrisiken wie C. Für die Generierung von Laufzeitprüfungen wäre es sicher auch sinnvoll, Unterstützung für Java zu integrieren.

Die Werkzeuge sind Hilfsmittel bei der komplexen Sicherheitsanalyse und werden im folgenden Kapitel am konkreten Beispiel eingesetzt.

Teil III

Umsetzung

Kapitel 8

Realisierung der LucaOS-Sicherheitsarchitektur

In diesem Kapitel wird die Umsetzung des Betriebssystems LucaOS für mobile Endgeräte beschrieben. Sie stellt eine praktische Anwendung der in der Arbeit entwickelten Methode TANAT dar. Zu Beginn des Kapitels werden die Randbedingungen erläutert, die für das Realisierungsprojekt gegeben waren. Dazu zählen die zur Verfügung stehende Hardware und Softwarebasis, auf der aufgesetzt wurde. Im Anschluss werden die Teilsysteme vorgestellt, die näher analysiert und abgesichert wurden. Dabei werden die charakteristischen Schritte und Ergebnisse vorgestellt. Nach der Beschreibung der einzelnen Analysen und des Konzeptes für ein Sicherheitsmanagement, werden Realisierungsaspekte diskutiert und die Ergebnisse abschließend zusammengefasst.

8.1 Randbedingungen

Im Rahmen der Arbeit waren für die Realisierung einige Randbedingungen gegeben, die im Folgenden besprochen werden. Hauptziel war die Entwicklung eines lauffähigen Prototypen[1], der für bestimmte Anwendungen einsetzbar ist.

8.1.1 Eingesetzte Hardware

Neben der Software wurde auch die Hardware berücksichtigt, da die Angriffe, die auf mobile Endgeräte durchgeführt werden können, denen der stationären Geräte grundsätzlich sehr ähnlich sind und zusätzlich Angriffe betrachtet werden müssen, die den Verlust und Diebstahl des Gerätes voraussetzen. Dabei bringt der Angreifer das Gerät und damit die

[1] Ein Prototyp wurde auf der Cebit 2003 im Rahmen des Schwerpunktprogramms Sicherheit vorgestellt.

Daten in seinen Besitz und hat diese permanent zur Verfügung. Die Angriffe, die dann möglich sind, gehen weit über die typischen Angriffe in Rechnernetzen hinaus. Sowohl ein Öffnen des Gerätes, als auch jede Art der Erweiterung oder Manipulation der Hardware sind dann möglich. Diese Angriffe besitzen eine sehr hohe Eintrittswahrscheinlichkeit und stellen damit ein hohes Risiko dar.

Zur Realisierung existierten nur wenige mobile Hardwareplattformen, erwähnenswert sind vor allem die zur Verfügung stehenden Handheldgeräte mit StrongARM-Prozessor, namentlich Compaq IPAQ 3630 und 3870. Kennzeichnend für diese Geräte ist ein Prozessor nach ARM-Architektur, eine verbreitete RISC-Architektur, die sowohl im Kleingerätebereich als auch bei stationären Geräten Verwendung findet. Das Prozessordesign bietet Unterstützung von virtuellem Speicher mit 32 Bit Adressgröße und der Unterscheidung zwischen priviligiertem und nicht priviligiertem Ausführungsmodus. Eine Fließkommaeinheit ist nicht vorhanden, entsprechende Maschinenbefehle werden vom jeweiligen Betriebssystem abgefangen und durch arithmetische Ganzzahloperationen simuliert. Für Sicherheitsmechanismen ist dies höchstens im Bereich der Kryptografie und Performance-Aspekte von Interesse. Da allerdings die meisten kryptografischen Verfahren auf Ganzzahlarithmetik beruhen, spielt diese Einschränkung in dieser Arbeit keine Rolle. Neben flüchtigem Hauptspeicher ist bei allen Modellen ein nicht flüchtiger Flash-Speicher vorhanden. Die Größe der Speicher liegt in aktuellen Geräten bei 64 MByte und enthält alle Objektdateien und Daten, die zur Ausführung nötig sind. Dieser Speicher ist prinzipiell ungeschützt, nach Diebstahl des Gerätes ist er auslesbar. Das Powermanagement arbeitet mehrstufig. Im Betrieb können die Prozessoren mit unterschiedlichen Frequenzen getaktet werden und so Energie sparen. Bei Nichtbenutzung oder durch Ausschalten wird ein Wartezustand eingenommen, der sämtliche flüchtigen Register speichert und so ein Weiterarbeiten später in exakt dem Zustand ermöglicht.

Als Schnittstellen nach außen sind prinzipiell zwei Klassen von Geräten zu unterscheiden, Benutzungsschnittstellen und Kommunikationsschnittstellen. Schnittstellen für den Benutzer sind das Touchscreen und mehrere Bedienschalter. Für die Sicherheitsanalyse sollten die nach außen geführten Systembusse, die Stromversorgung und der Reset-Taster beachtet werden. Diese können von einem Angreifer, der im Besitz des Gerätes ist, jederzeit und beliebig oft benutzt werden und können in speziellen Systemzuständen die Sicherheit des Systems beeinflussen, wie später im Abschnitt 8.3 noch gezeigt wird. Zur Kommunikation stehen eine Infrarotschnittstelle, möglicherweise ein eingebautes Bluetooth-Modul zur drahtlosen Kommunikation, modulare Netzwerkkarten für drahtlose und drahtgebundene Verbindungen sowie Lautsprecher und Mikrofon zur Verfügung. Details zur verwendeten Hardware werden in Anhang D gegeben.

8.1.2 Verwendete Software

Wie man bei einigen Enwicklungen von Betriebssystemkernen gelernt hat, ist der Aufwand für die Entwicklung abgesicherter Kerne sehr hoch und kann nur schwer abgeschätzt werden. Dies betrifft sowohl monolithische als auch feiner strukturierte Kerne. Obwohl zahlreiche Entwicklungen existieren, werden dadurch meistens nur Teilaspekte abgesichert oder der Aufwand ist unverhältnismäßig hoch. Beispiele hierfür sind die in Kapitel 3 vorgestellten Betriebssysteme wie PSOS und KSOS.

Da ein Ziel die Entwicklung eines funktionsfähigen Prototypen war, musste bei der Entwicklung auf ein bestehendes System aufgesetzt werden. Dadurch konnte der Aufwand zur Entwicklung der eigentlichen Funktionalitäten möglichst gering gehalten werden. Der Nachteil dabei ist aber die Verwendung fremden Programmcodes, der nur teilweise analysiert werden konnte. Das entwickelte Verfahren sowie die einzelnen realisierten Konzepte lassen sich prinzipiell auch in anderen, auch neu entwickelten Betriebssystemen anwenden.

Da für die ausgewählte Hardware im Wesntlichen nur zwei Betriebssyteme existieren, muss zwischen diesen beiden gewählt werden. PocketPC, ein System für PDAs aus der WindowsCE-Familie, hat große Vorteile durch die Verfügbarkeit von Entwicklungswerkzeugen. Diese sind entweder in die Standardwerkzeuge integriert oder diesen nachempfunden. Dies ermöglicht sowohl eine geringe Einarbeitungszeit als auch die Übernahme von bestehendem Quellcode anderer Systeme. Der Quelltext zu PocketPC ist nur in Teilen und unter bestimmten rechtlichen und sehr restriktiven Bedingungen verfügbar. Wichtige Teile des Betriebssystems, die nicht im Quelltext zur Verfügung stehen, sind das vollständige Dateisystem, der Bootlader mit den Teilen des Kerns, die für das Initialisieren zuständig sind und die Teile der Speicherverwaltung, die die Rechtevergabe der einzelnen Slots regeln. Nähere Informationen zur WindowsCE Architektur findet sich in Abschnitt 3.2.2. Die Unterstützung für den Code wird im Wesentlichen durch den Hersteller bereitgestellt, freie Community-Unterstützung existiert nicht. Für die Kernentwicklung, die in dieser Arbeit im Vordergrund stand, ist PocketPC deshalb nur sehr eingeschränkt geeignet, da wesentliche Bestandteile des Kerns weder analysiert werden können, noch veränderbar sind. Vorteil dieser Herstellerabhängigkeit ist allerdings, dass weder unterschiedliche Versionen parallel nebeneinander entwickelt werden, noch dass wesentliche Programmierschnittstellen geändert werden, außer vom Hersteller. Code auf Basis von PocketPC-Schnittstellen ist beständig gegenüber Änderungen des Betriebssystems, was allerdings durch eine geringe Flexibilität bei der Eigenentwicklung innerhalb des Systemkerns erkauft wird. Zudem haben Entwickler außerhalb des Herstellers praktisch keinen Einfluss auf die zukünftige Entwicklung.

Das andere portierte Betriebssystem auf die verwendete Plattform ist ein Linux-Kern mit den entsprechenden Bibliotheken und allen verfügbaren Anwendungen. Das Open Source System Linux ist zwar ursprünglich als Server- und Desktop-Betriebssystem entwickelt worden, mittlerweile existieren aber Portierungen zu allen bekannten Prozessorarchitekturen, auch auf die hier eingesetzte StrongARM Plattform. Für Linux sprechen vor allem

die Verfügbarkeit von Werkzeugen für die Entwicklung, sowohl im Kernbereich als auch für Anwendungen. Unterstützung und Dokumentation für die frei verfügbaren Quelltexte existiert ausschließlich durch die Entwicklergemeinde. Damit sind immer alle Teile des Kernbereichs veränderbar. Einzige Problematik liegt darin, dass auch wichtige Kernkomponenten und Schnittstellen bei neueren Zwischenversionen verändert werden. Dadurch können bei einer solchen offenen, verteilten Entwicklung auch Entwickler Schwachstellen und Hintertüren in ein System einbauen. Dies wird zunächst dadurch verhindert, dass genau ein Entwickler jede Änderung des Kerns analysiert und freigibt. Mit wachsender Größe wurde dies erweitert, so dass nun für jeden Versionszweig des Linux-Kerns eine Person weltweit für die Überprüfung und die Freigabe verantwortlich ist. Ob dies mit der erweiterten Hardwareunterstützung und damit Komplexität der Quelltexte und den steigenden Qualitätsanforderungen in Zukunft noch vereinbar sein wird, muss gut überlegt werden. Es existieren jedenfalls Anzeichen, wie beispielsweise die länger werdende Zeitdauer zwischen Versionsfreigaben, dass dies eines der kommenden Probleme sein kann.

8.1.3 Basis der LucaOS Architektur

Die grundlegende Architekturüberlegung für LucaOS ist die Kontrolle der Kommunikation innerhalb des Kerns und zwischen Kern und den Anwendungen. Unter Kommunikation werden dabei die Kontrollflüsse verstanden, die beispielsweise einen aktiven Angriff darstellen können. Neben den eigentlichen generischen Kontrollfunktionen innerhalb des Betriebssystemkerns muss ein Management zur Verwaltung der Regeln integriert werden. Die Architektur ist in Abbildung 8.1 dargestellt.

In der Bildmitte rechts sind die Kernteile eines modernen Betriebssystems dargestellt, so wie sie sowohl unter Linux oder Windows bekannt sind. Darunter ist mögliche Hardware für den Einsatz am mobilen Endgerät für Kommunikation, persönliche externe Datenspeicher und Positionsbestimmungssysteme aufgezählt. Die Anwendungen können typischerweise nur über Systemdiensteschnittstellen mit dem Kern kommunizieren, ein direkter Datenaustausch zwischen Anwendungen oder zwischen Anwendungen und Kern ist nicht möglich. Darüber finden sich die typischen Anwendungsfälle für den Benutzer, Interaktion mit Anwendungen und gegenseitige Authentifizierung. Im Bereich der mobilen Endgeräte ist die Konfiguration der Sicherheitseigenschaften auch Aufgabe des Endbenutzers. Er muss die eigentliche Rolle des Administrators in stationären Systemen übernehmen und Sicherheitseinstellungen dem aktuellen Kontext und der Umgebung angepasst vornehmen. Das Prinzip der Architektur von LucaOS ist die Kontrolle der Kommunikation zwischen den einzelnen Einheiten, sowohl innerhalb des Kerns als auch zu den Anwendungen hin. Getrennt davon ist auf der linken Seite das zunächst betriebssystemunabhängige LucaOS-Konzept zu sehen, das in ein entsprechendes Betriebssystem integriert werden kann. Die Sicherheitsdatenbank speichert wichtige auftretende Ereignisse und enthält die festzulegende Sicherheits-Policy. Die Kontrollen selbst werden durch den sogenannten Policy-Manager durchgeführt, der Nachrichten der zu überwachenden Kontrollstellen erhält und diese re-

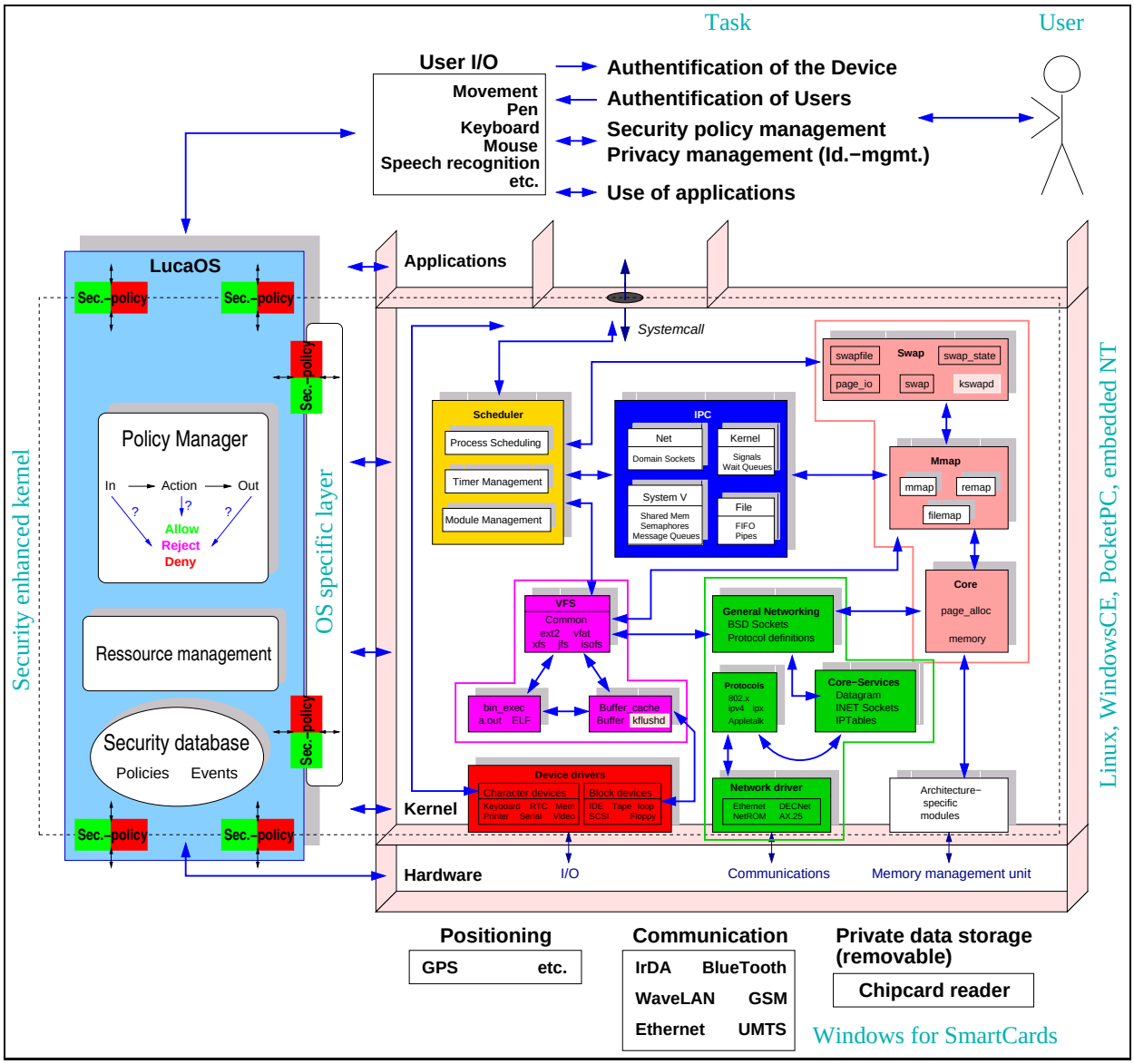

Abbildung 8.1: Überblick über die LucaOS-Architektur

gelt. Dieses grundlegende Design erzeugt nicht automatisch ein abgesichertes System, es stellt vielmehr einen Rahmen für vielseitige Zugriffskontrollen innerhalb des Systems dar. Dazu ist es wesentlich, eine passende Sicherheits-Policy für die jeweilige Anwendung des Systems zu entwickeln. Im mobilen Bereich kommt hinzu, dass die Policy nicht wie in statischen Systemen konstant sein kann, sondern sich beispielsweise in Abhängigkeit der Umgebung beziehungsweise des Ausführungskontextes ändern und anpassen muss. Diese grundlegende Architektur ist für diese Arbeit realisiert worden. Kontrollierbar sind bisher die Systemdiensteschnittstellen und verschiedene Teilsysteme, deren Analyse in den fol-

genden Abschnitten besprochen wird. Andererseits steht bei der Umsetzung die Methodik zur Sicherheitsanalyse im Vordergrund. Es hat sich gezeigt, dass sich bei der Analyse die mobilen Systeme konzeptionell nicht mehr von den stationären, herkömmlichen Systemen unterscheiden. Die Bedrohungen und möglichen Angriffe existieren in beiden Welten, wenn auch in unterschiedlicher Ausprägung und Gewichtung. Die Gewichtung hängt dann aber von der Risikoanalyse ab, die hier nicht genauer betrachtet wird.

Die Konstruktion der entsprechenden grundlegenden Sicherheits-Policies ist ein Ergebnis der im Folgenden vorgestellten Bedrohungsanalyse. Eine spezialisierte Policy für bestimmte Anwendungsfälle und die Realisierung des Sicherheitsmanagements werden aber in einer anderen Arbeit durchgeführt. Dort wird auch über die Entwicklung des Managements diskutiert. Ziel dieser Arbeit ist die erweiterte Sicherheitsanalyse einiger Bausteine dieses Grundsystems.

Als Basis der Realisierung wurden Teile des Linux Kerns 2.4.21 verwendet. Damit sind die üblichen Schutzkonzepte bekannter Betriebssysteme vorhanden, wie etwa virtueller Speicher und ein abgesichertes Prozessmodell, wie sie auch in Kapitel 3 beschrieben wurden. Erweitert durch ein Rahmenwerk für systemweite Zugriffskontrolle (Mandatory Access Control MAC), das aus dem SELinux-Projekt übernommen wurde, sind Kontrollen in allen Teilen des Kerns möglich. Die einzelnen Stellen innerhalb des Kerns, an denen Kontrollen stattfinden können, werden durch sogenannte Aussprungpunkte „Hooks“ fest kodiert. Diese Codestellen sind allerdings nur generische Sprunganweisungen, die je nach festgelegter Sicherheits-Policy kontrolliert werden oder nicht. Die eigentlichen Kontrollen können an diesen Stellen beliebig in einer Hochsprache formuliert und während der Laufzeit verändert werden. Das MAC-Framework stellt ein generisches Regelwerk für systemweite Zugriffskontrollen zur Verfügung, Policies können im Betrieb integriert und verändert werden. Den damit entstandenen Realisierungsrahmen zeigt Abbildung 8.2. Viele Details zur SELinux Implementierung finden sich in [LS01a]. Die in der Abbildung dargestellten Datenbank (DB) ist Teil des noch zu integrierenden Sicherheitsmanagements und nicht Aufgabe dieser Arbeit gewesen.

Auf dieser Basis wurden die Sicherheitsanalysen und Realisierungen durchgeführt, wie sie in den folgenden Abschnitten beschrieben werden. Da aus den genannten Gründen, die für den Einsatz von Linux als Basisplattform sprechen, der Schluss gezogen werden könnte, dass Open Source Projekte sicherer sind als andere, wird in Anhang C kurz diskutiert, ob Open Source für sichere Systeme eine wichtige Grundlage ist.

8.2 Wahl der Teilsysteme

Nachdem ein Überblick über das komplette Betriebssystem bekannt ist, werden die interessanten Teilsysteme für die Sicherheitsanalyse ausgewählt. Dazu wird eine Lebenszeitanalyse des Systems durchgeführt. Dabei werden die wesentlichen Vorgänge im System

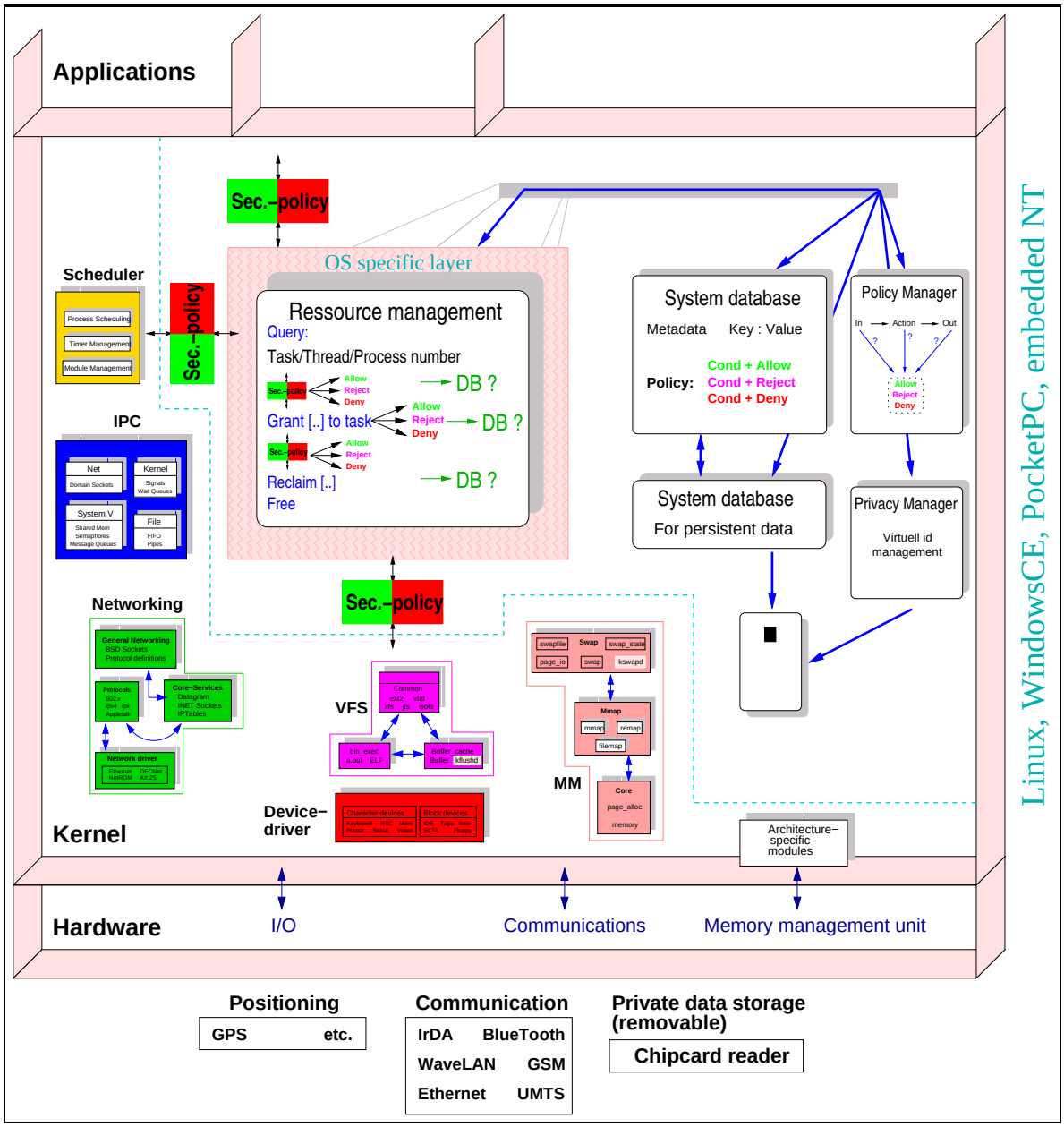

Abbildung 8.2: Die LucaOS-Architektur integriert

betrachtet, die beginnend mit der Installation geschehen und sicherheitsrelevant sind. Für die einzelnen ausgewählten Phasen wurden Absicherungen realisiert, die vor naheliegenden Angriffen schützen und die grundlegenden Sicherheitseigenschaften sichern. Diese einzelnen Realisierungen werden mit der TANAT Methode analysiert und auf weitere Bedrohungen geprüft.

Installationsphase

Die Installation eines Betriebssystems ist der notwendige erste Schritt zur Verwendung des Systems und wird im Allgemeinen nur einmal durchgeführt. Trotzdem ist ein abgesicherter Installationsvorgang wesentlich für die Sicherheit des Systems. Neben dem Festlegen der Passwörter und der angebotenen Dienste wird bei der Installation auch die Integrität der kopierten Programme und Module bestimmt. Eine Kompromitierung während der Installationsphase kann zum Verlust der Gesamtsicherheit des Systems führen.

Bootvorgang

Erster Laufzeitschritt eines Betriebssystems ist das geordnete Starten der Dienste in den einzelnen Schichten. Typischerweise werden sie in der Reihenfolge von den hardwarenahen hin zu den anwendungsnahen Diensten gestartet. Um wirksam gegen mögliche Angriffe zu sein, müssen die Abwehrmaßnahmen einer Schicht aktiv sein, wenn eine darüberliegende Schicht initialisiert wird. Nur so kann eine Vertrauenskette aufgebaut werden, die auf der Sicherheit des Betriebssystemkerns im privilegierten Modus basiert. Wichtig für die Sicherheit des Gesamtsystems ist die Kontrolle der Integrität der einzelnen Komponenten und der zum Start nötigen Daten. Die Dienste arbeiten meist im privilegierten Modus und bieten vertrauenswürdige Services für die Anwendungen an, deren Kompromitierung die Sicherheit aller Anwendungen gefährden würde. Die einfache Realisierung in LucaOS setzt die Integrität des Bootladers vorraus. Dieser enthält eine Prüfsumme des zu ladenden Kerns und verweigert dessen Start, wenn die Prüfsumme oder das obligatorische Bootpasswort falsch ist. Der Bootlader selbst könnte allerdings von einem Angreifer mit entsprechendem Wissen und dem Besitz des Geräts verändert werden. Ohne Hardwareschutz, wie sie etwa ein TPM bietet, kann an dieser Stelle kein besserer Schutz in Software realisiert werden.

Laufzeit des Betriebssystems

Zur Laufzeit des Systems sind viele Vorgänge des Betriebssystems sicherheitsrelevant. Für LucaOS wurden in dieser Arbeit folgende Teilsysteme analysiert.

1. Zuverlässige Authentifizierung
2. Sichere Erweiterbarkeit des Kerns
3. Gesicherte Systemdienste
4. Abgesicherter persistenter Speicher

Diese Systemfunktionalitäten sind für die Sicherheit des Systems äußerst relevante Teile, die bisher nur schlecht abgesichert sind und nicht analysiert wurden.

Ein wirkungsvoller Mechanismus zur sicheren Erweiterbarkeit des Kerns wurde auch umgesetzt. Da die beschriebene Hardwarebasis des StrongARM allerdings dafür unbrauchbar ist, wurde dieser Teil der Arbeit für einen x86-Kern entwickelt, da diese Hardware die nötigen Register und Segmentunterstützung besitzt. Folgende ARM-Architekturen für mobile Systeme werden allerdings sehr wahrscheinlich die notwendigen Unterstützungen für Segmentierung der virtuellen Speichers beinhalten, so dass die Archtektur auch für diese Systeme einsetzbar ist.

Updatevorgang

An einem installierten und weitgehend korrekt arbeitenden Betriebssystem werden bestimmte Änderungen durchgeführt. Diese sind aus mehreren Gründen notwendig. Das Hinzufügen neuer Hardware und die damit nötige Treiberunterstützung bedeuten Änderungen im Betriebssystem. Selbst wenn einfache Gerätetreiber für Block- oder Zeichen-orientierte Geräte im laufenden Betrieb ein- und ausgeladen werden können, ist dies bei Treibern für neue Busssysteme oder der Anpassung des Ressourcenmanagements keinesfalls mehr möglich. Dabei, oder nach Bekanntwerden von Sicherheitsschwachstellen ist die Veränderung des Betriebssystemkerns unumgänglich. Auch hier können für Angreifer Möglichkeiten entstehen, das System bezüglich seiner Sicherheit zu kompromitieren.

Herunterfahren

Eine besondere Phase im Umgang mit priviligierten Diensten ist das Herunterfahren eines laufenden Systems. Dabei werden wichtige Anwendungen und Dienste kontrolliert beendet. Es gibt unterschiedliche Strategien im Umgang mit nicht beendbaren Prozessen, die untersucht werden.

8.3 Sicherheitsanalysen

In diesem Abschnitt werden die Sicherheitsanalysen der identifizierten wichtigen Teilsysteme beschrieben. Sie wurden mit der in Kapitel 6 beschriebenen Methode TANAT und den entsprechenden Werkzeugen durchgeführt. Die Methode wurde durch den Einsatz an diesen Teilsystemen weiterentwickelt. Deshalb liegen den einzelnen Analysen unterschiedliche Detailgrade zugrunde. Eine vollständige Beschreibung aller Sicherheitsanalysen würden den Rahmen dieses Kapitels sprengen, deshalb werden nur die wesentlichen Schritte und Ergebnisse präsentiert.

Im Allgemeinen sind die Analysephasen folgendermaßen gegliedert, wie sie auch in Kapitel 6.5 bei der Beschreibung der Methode TANAT angegeben sind:

1. Allgemeine Systembeschreibung
2. Modellierung der funktionalen Einheiten
3. Schnittstellenbildung
4. Kontextbeschreibung
5. Funktionale Beziehungen
6. Verfeinerung und Wahl der Aufruftiefe
7. Mustervergleich der Angriffsklassen
8. Iteration der Bedrohungssuche

8.3.1 Analyse der Installationsphase

Einen besonderen Teil der Analyse nimmt die Installationsphase des Systems ein, da sie für ein System im Allgemeinen nur einmal ausgeführt wird und mit der eigentlichen Systemanwendung nicht direkt zusammenhängt.

8.3.1.1 Allgemeine Systembeschreibung

In der Installationsphase werden die benötigten Softwarekomponenten von einem Zentralspeicher auf den nichtflüchtigen Systemspeicher übertragen. Die Systemkomponenten werden dabei entpackt und in eine Dateisystemstruktur integriert. Zudem werden Konfigurationseinträge für die einzelnen Komponenten erzeugt, die sowohl aus Skripten, als auch aus Textdateien bestehen können. Neben den Anwendungen werden auch der Binder und Lader, der Betriebssystemkern und die systemnahen Programme installiert. Es ist zu prüfen, welche Bedrohungen durch Manipulation bei diesen Vorgängen entstehen können.

8.3.1.2 Modellierung der funktionalen Einheiten

In der Tabelle 8.1 werden die funktionalen Einheiten beschrieben, die bei der Installationsphase bei der Sicherheitsanalyse berücksichtigt sind. Bei der Übernahme in andere Kontexte der Ergebnisse ist diese Liste anzupassen, da bei anderen Systemen beispielsweise andere Subjekte auftauchen können.

8.3.1.3 Schnittstellenbildung

Im nächsten Schritt werden die Schnittstellen des System beschrieben. Dazu wird für jede Systemeinheit eine Liste augestellt, die in Tabelle 8.2 aufgeführt wird.

Einheit	Beschreibung
Geräteschnittstellen	Menge aller möglichen Einflussgrößen
Persistenter Systemspeicher	Flash ROM
Zentralspeicher	Anderer Rechner mit IP-Netzwerk verbunden
Installationsprozess	Prozesse von `apt`
Betriebssystem	Alle Betriebssystemdienste
Administrator	Person

Tabelle 8.1: Funktionale Einheiten

Einheit	Schnittstellen
Geräteschnittstellen	Display, Tasten und Schalter, Bus
Persistenter Systemspeicher	Speicheroperationen
Zentralspeicher	Leseoperationen
Installationsprozess	Eingaben, Unterbrechungen und Fehler
Betriebssystem	Systemdienste, Netzwerk
Administrator	Bedienung, „verletzbare Person“

Tabelle 8.2: Überblick über die Schnittstellen

Erklärung benötigt der erste Eintrag zu den „Geräteschnittstellen“ in der Tabelle. Damit sind alle Möglichkeiten für Aktionen bezeichnet, die eine Person über Schnittstellen auf das System ausüben kann. Die Person muss dazu im Besitz des Gerätes sein. Für die spezielle Sicherheitsanalyse hier sind damit folgende Schnittstellen der verwendeten Hardware (das mobile Gerät iPAQ) untersucht: berührungssensitives Display, alle Tasten des Gerätes, Schalter (auch Reset) und die vorhandenen Schnittstellen wie RS-232, Bluetooth oder der nach außen geführte Datenbus. Zu diesen externen Schnittstellen existieren noch zahlreiche innere Schnittstellen, wie CPU- und Speicherbenutzung und die (meist priviligierte und kontrollierte) Nutzung der Hardware durch Prozesse. Da dieselbe Hardware auch bei den folgenden Analysen eingesetzt wird, gelten die Menge der Geräteschnittstellen auch dort.

8.3.1.4 Kontextbeschreibung

Es wird angenommen, dass die Installation von jedem Benutzer durchgeführt werden kann. Er kann zwar keine eigenen Installationspakete erzeugen und diese in das System integrieren, er kann aber den Installationsort und den Zeitpunkt bestimmen.

8.3.1.5 Funktionale Beziehungen

Die funktionalen Beziehungen zwischen den identifizierten Einheiten werden durch ASMs dargestellt. Der Grobentwurf ist in Abbildung 8.3 zu sehen. Nicht dargestellt sind dabei zunächst die Beziehungen zum Betriebssystem. Allerdings ist klar, dass alle Kopier- und Ein-/Ausoperationen durch Systemdienste realisiert sind.

Zu sehen ist der Ablauf bei der Installation mit dem Diensteprogramm `apt`. Der Installationsablauf besteht aus der Auswahl der Pakete, dem Laden vom Installationsserver und der eigentlichen Installation.

8.3.1.6 Verfeinerung und Wahl der Aufruftiefe

Für diese Analyse ist nur die Darstellung des Grobentwurfs interessant, da weitere Verfeinerungen keine neuen Bedrohungen liefern. Daher wird hier nur der Grobentwurf beschrieben, bei der Anwendung der Methode wurden aber weiterere Verfeinerungen durchgeführt, die keine weiteren sicherheitsrelevanten Ergebnisse brachten. Die Aufruftiefe muss deshalb für diesen einfachen Fall nicht festgelegt werden.

8.3.1.7 Mustervergleich der Angriffsklassen

Die Suche nach Schwachstellen mit der Guide-Words Methode wird in folgender Reihenfolge durchgeführt, wobei jeweils die Zeilennummer der Darstellung 8.3 verwendet wird.

1: (Late) Der Benutzer kann die Installationsroutine benutzen und während dessen das System herunterfahren bzw. ausschalten.

6: (No) Der Benutzer gibt keine Auswahl an und blockiert damit das Paketverwaltungssystem für alle anderen. Damit ist auch kein automatisches Softwareupdate mehr möglich.

15: (No und Other than) Es werden keine oder gefälschte Paketdaten empfangen. Diese sind zwar signiert, allerdings können eigene Signaturschlüssel erzeugt werden. Die Signatur dient eher einer Integritätsprüfung.

```
Administrator/User: Installationsprozess apt()
2
func: Installationsprozess apt()
4 if STATE=auswahl then {
  SEQ: {
6    dialog(paketewahl) durch Geraeteschnittstellen
     STATE=download
8  }
}
10
if STATE=download then {
12  SEQ: {
      forall paket in paketewahl
14    {
        empfangen(paket) vom Zentralspeicher
16        // persistenter Speicher:
        speichern(paket) in /var/apt/archives/
18    }
      STATE=install
20  }
}
22
if STATE=install then {
24  SEQ: {
      forall paket in paketewahl
26    {
        entpacke paket in /var/tmp
28      preinstallation
        kopieren der Dateien in Systemspeicher
30      erzeugen der Konfigurationen
        postinstallation
32    }
      RETURN
34  }
}
```

Abbildung 8.3: Darstellung der funktionalen Beziehungen

16: (Less) Der Speicherplatz ist voll und es können nur Teile der Pakete gespeichert werden.

26: (Less) Entpacken wird aufgrund Speichermangels unvollständig beendet.

28+29: (Less) Kopieren und Konfiguration erstellen wird unvollständig beendet.

30: (No) Postinstallation wird durch Abschalten abgebrochen.

Angegeben sind dabei nur die Fälle, die tatsächlich auch im existierenden System möglich sind. Beispielsweise wäre es ebenso bedrohlich, wenn der Benutzer das Installationsprogramm öfter (More) aufrufen würde und widersprüchliche Dinge installiert. Dies wird allerdings bereits von vorneherein durch Synchronisationsmaßnahmen verhindert.

Die einzelnen denkbaren Schwachstellen, die nun gefunden wurden, dienen als Ausgangspunkte für die möglichen Angriffe. Sie tauchen daher später in Form von Blättern im T_{AV}-Baum wieder auf.

Angriffe auf die Vertraulichkeit

Die schützenswerten Daten sind die Installationsdateien und Konfigurationen. Ein gefundener Angriff ist das Ausspionieren dieser Daten während der Stufen in Zeilen 26–29. Die temporären Dateien sind für alle Benutzer lesbar, so dass damit sowohl eine Angriffsvorbereitung als auch ein Angriff möglich ist. Ein gefundener passiver Angriff ist der, dass ein interner Angreifer Buch über alle installierten Pakete und Versionsnummer führt und für diese Pakete gezielt nach Schwachstellen sucht.

Angriffe auf die Integrität

Die Integrität der Pakete kann auf dem System nicht verändert werden. Allerdings kann dies auf dem Weg dorthin (Zeile 15) oder auf dem Zentralspeicher geschehen. Zudem könnte ein Angreifer einen Man-in-the-Middle Angriff durchführen oder einen kompromitierten Zentralspeicher zur Verfügung stellen.

Angriffe auf die Verfügbarkeit

Bei der Angriffssuche nach externen Angriffen können drei ausnutzbare Schnittstellen für den Angreifer untersucht werden, die Geräteschnittstellen, der Zentralspeicher und Einflussnahme auf den Administrator. Beeinflussung der Verfügbarkeit des Administrators kann im Wesentlichen der Diebstahl von Passworten und das Vortäuschen einer anderen Identität sein. Angriffe auf die Verfügbarkeit des Zentralspeichers betreffen zunächst das System nicht direkt und können nur durch Kenntnisse über diesen Speicher analysiert werden.

8.3.1.8 Iteration der Bedrohungssuche

Diese Analyse wurde ohne weitere Verfeinerungsschritte durchgeführt. Die möglichen Angriffe, die durch die Analyse gefunden wurden, zeigt der T_{AV}-Baum in Abbildung 8.4.

Hierarchie	Typ	Name	Risk
T 1	⌓	Bedrohungen gegen die Softwareinstallation	
T 1.1	⌓	Angriff auf die Vertraulichkeit	
T 1.1.1	◇	Aktiver Angriff auf die Vertraulichkeit	
T 1.1.2	○	Passiver Angriff auf die Vertraulichkeit	
T 1.1.2.1	○	Auslesen der temporären Dateien in Schritt 26	
T 1.1.2.1.1	⬭	*VUL* /var/tmp/ ist lesbar	
T 1.2	⌓	Angriff auf die Integrität	
T 1.2.1	⌓	Aktiver Angriff auf die Integrität	
T 1.2.1.1	◇	Manipulation des Zentralspeichers (Schritt 15)	
T 1.2.1.2	◇	Man-in-the-Middle bei der Übertragung (Schritt 15)	
T 1.2.2	◇	Passiver Angriff auf die Integrität	
T 1.3	⌓	Angriff auf die Verfügbarkeit	
T 1.3.1	⌓	Aktiver Angriff auf die Verfügbarkeit	
T 1.3.2	◇	Passiver Angriff auf die Verfügbarkeit	

Tabelle 8.4: Bedrohungsbaum für die Installationsphase

Nächster Schritt wäre die Risikobewertung der Analyseergebnisse im T_{AV}-Baum. Dies wurde in dieser Arbeit nicht durchgeführt, es wurden lediglich einfache Gegenmaßnahmen für die offensichtlichen Bedrohungen im Prototypen realisiert. Beispielsweise wurde die Rechtevergabe in den temporären Verzeichnissen auf sehr restriktive Einstellungen geändert.

8.3.2 Abgesicherter Systemstart

Nächste Phase zur Analyse des Systems ist der Systemstart. Problematisch dabei ist grundsätzlich, dass ein Benutzer des mobilen Systems dieses stets neu starten kann und versuchen kann, einen eigenen Betriebssystemkern zu laden. Dieser könnte dann unter Umgehung aller sonst durchgesetzten Sicherheitsmechanismen alle Daten ausspionieren und weiterleiten. Zudem könnten alle Benutzereingaben, auch Passworte, abgefangen und gespeichert werden. Es ist demnach zwingend notwendig, dass der Systemstart nur durch

authorisierte Administratoren oder den Besitzer verändert werden kann. Diese Veränderungen sind allerdings zeitweise notwendig, wenn etwa der Betriebssystemkern und systemnahe Module (Treiber) angepasst werden müssen.

8.3.2.1 Allgemeine Systembeschreibung

Der Systemstart wird durch bestimmte einfache Maßnahmen abgesichert, so dass er nur durch bestimmte Personen abgebrochen und verändert werden kann. Der Startvorgang erfolgt in der angegebenen Reihenfolge:

1. Starten des Bootladers an Adresse `0x0` im Flash-Speicher durch Systemstart.
2. Der Bootlader kopiert sich selbst in den RAM-Speicher, erst danach wird die MMU programmiert. Dadurch kann der Flash-Speicher in andere Speicherbereiche eingeblendet werden und der Bootlader wird schneller ausgeführt. Danach sieht die Speicheraufteilung im RAM aus, wie im Anhang D in Tabelle D.1 abgebildet.
3. Der Bootlader lädt dann den Standardkern vom Flash-Speicher an eine Adresse im RAM. Dort wird der Kern entpackt und die Kontrolle an ihn übergeben.
4. Zu Administrationszwecken kann der Bootlader durch die Eingabe eines Codeworts angehalten werden. Der Bootvorgang wird abgebrochen, wenn die Eingabe mit einer gespeicherten Zeichenfolge übereinstimmt. Anschließend kann auf alle Speicherbereiche im Flash und RAM beliebig geschrieben werden. Dadurch kann auch das Betriebssystem und der Kern ersetzt werden.

Nachteil der bestehenden Implementierung ist die komplizierte Änderung des Codewortes. Dieses wird beim Übersetzen des Bootladers in die Objektdatei fest einkompiliert und kann nur durch Neuübersetzung und Überschreiben des Flash-Bereichs übernommen werden. Das ist für den Prototypen ausreichend, müsste aber derart verändert werden, dass das Codewort durch ein Linkerskript an einer festen physikalischen Speicheradresse abgelegt und dort änderbar ist. Diese Verbesserung dient der einfachen Änderung des Bootlader-Passworts, das nur von autorisierten Personen oder Prozessen durchgeführt werden kann.

8.3.2.2 Modellierung der funktionalen Einheiten

Für die Analyse wurden die Einheiten berücksichtigt, die in Abbildung 8.5 angegeben sind.

8.3.2.3 Schnittstellenbildung

Die Schnittstellen ergeben sich aus den funktionalen Einheiten wie in Abbildung 8.6 gezeigt.

Einheit	Beschreibung
Geräteschnittstellen	Menge aller möglichen Einflussgrößen
Persistenter Systemspeicher	Flash ROM, Systemquelle beim Start
Administrator	Person durch PIN-Eingabe autorisiert

Tabelle 8.5: Funktionale Einheiten

Einheit	Schnittstellen
Geräteschnittstellen	Display, Tasten und Schalter, Bus
Persistenter Systemspeicher	Speicheroperationen
Administrator	typ. Person

Tabelle 8.6: Schnittstellen des Systemstartmoduls

8.3.2.4 Kontextbeschreibung

Der Systemstart kann prinzipiell immer durchgeführt werden, weil das Zurücksetzen des Systems nie unterdrückt werden kann. Damit ist der Neustart für jede Person möglich, die im Besitz des Gerätes ist.

8.3.2.5 Funktionale Beziehungen

Die Funktionalität des Systemstartprogramms wird in Abbildung 8.4 gezeigt. Die Programmierung der Memory Management Unit (MMU) und das Einlagern des Betriebssystemkerns kann nicht verändert oder beeinflusst werden.

8.3.2.6 Verfeinerung und Wahl der Aufruftiefe

Die Verfeinerungen der oben gezeigten Grobstruktur zur Feinstruktur bringen keine neuen Erkenntnisse bei der Analyse. Die Betrachtung der Implementierung jedoch lässt eine schwerwiegende Schwachstelle erkennen, die in Abbildung 8.5 gezeigt wird. Siehe dazu auch die Analyse in Abschnitt 8.3.2.7. Für die untersuchten Versionen des Bootladers wird durch eine spezielle Eingabe beim Systemstart aus Komfortgründen ein externer Kern geladen und gestartet. Jede Person, die das Gerät benutzt, kann durch einfachen Neustart dadurch das System kompromitieren.

```
Administrator/User: Bootprozess startup()

func: Bootprozess startup()
if STATE=copyToRAM then {
  SEQ: {
    // BootldrBaseAdr und BootldrSize beim Compilieren
    // festgelegt
    memcopy(0x10000, BootldrBaseAdr, BootldrSize)
    // Programmierung der MMU unveraenderbar
    progMMU
    // Aufruf des Bootladers im RAM
    call 0x14000
    STATE=waitForUserinput
  }
}

if STATE=waitForUserinput then {
  <2 Sekunden Timer starten>
  {
    process getInput()
    // gekuerzt: Eingabe pruefen
    // alternativen Kern laden
  }
  STATE=loadKernel
}

if STATE=loadKernel then {
  SEQ: {
    KernelBaseAdr = getKernelBaseAdr
    KernelSize = getKernelSize
    memcopy(0xc000000, KernelBaseAdr, KernelSize)
    call 0xc0004000 // Einsprungpunkt durch Compiler
                    // festgelegt
    // Anweisung wird nie erreicht
  }
}
```

Abbildung 8.4: Darstellung der funktionalen Beziehungen

8.3.2.7 Mustervergleich der Angriffsklassen

Zunächst wird nach Schwachstellen im Grobentwurf mit der Guide-Words Methode gesucht:

```
Function UpdateOS
SER:
if instVersion<0.31 then return(-ENOVER);
ioperm(0x03f);
if keyPress=IP_KEYUP then loadExternKernel();
foreach UTASK
{
  PAR:
   kill UTASK.NUM;
   rc.end(NUM);
}
unsetFlashProtect();
[..]
memcpyall(&ROOTMEM400,&FLASHMEM,imgSize);
call(0xfffffe00);
```

Abbildung 8.5: Ausschnitt aus der fehlerhaften Implementierung

8 und 10: (Part of) Die Ausführungen können zwar durch ein Rücksetzen des Systems unterbrochen werden, jedoch wird dann auf jeden Fall die Anweisung vollständig von Beginn an ausgeführt. Dadurch sind keine Schäden zu erreichen.

20: (More/Less/Other) Es muss näher untersucht werden (Feinentwurf und Implementierung), ob der Benutzer bestimmte Eingaben als Angriff ausnutzen kann.

27-30: Auch hier kann kein Einfluss genommen werden.

Der Implementierungsentwurf nach Abbildung 8.5 ergibt:

5: (Other than) Eine fehlerhafte Funktion erlaubt es, dass ein externer Kern von einem Flashspeicher geladen werden kann, wenn die `KEYUP`-Taste gedrückt wird.

8-10: (Late) Das Erneuern des Flash-Bereichs wird durch eine Beendigung der laufenden Prozesse und Prozessgruppen vorbereitet, da der Flash-Speicher vom schreibgeschützten in den Lese-/Schreib-Modus geschaltet wird (Zeile 12). Allerdings wird nicht auf die Beendigung der Prozesse gewartet, so dass ein bösartiger Prozess den Flash-Speicher ändern kann.

Angriffe auf die Vertraulichkeit

Ein möglicher Angriff auf die Vertraulichkeit wäre die vorbereitende Maßnahme $\mathcal{M}_{attackphases} = \mathcal{M}_{(pre)}$, dass ein fremder Kern beim Systemstart geladen wird, der praktisch jede beliebige spätere Angriffsdurchführung $\mathcal{M}_{attackphases} = \mathcal{M}_{(act)}$ ermöglichen kann.

Außerdem könnte ein feindlicher Prozess seine Beendigung verhindern und bei einem Systemupdate nach dem Zulassen der Schreiboperationen auf den Flash-Bereich diesen verändern. Dazu muss allerdings ein Prozess durch den Angreifer installiert oder derartig verändert worden sein, was im Wesentlichen durch fehlerhafte Anwendungen möglich ist. Diese müssen separat analysiert werden.

Angriffe auf die Integrität

Die Angriffe auf die Integrität entsprechen denen auf die Vertraulichkeit.

Angriffe auf die Verfügbarkeit

Neben dem bewussten Einführen eines geänderten Kerns kann auch die Verfügbarkeit des Systems zerstört werden. Dies kann sehr einfach durch das Überschreiben des Systemstartbereichs im Flash-Speicher geschehen. Das System kann dann ohne Reparatur nicht mehr gestartet werden.

8.3.2.8 Iteration der Bedrohungssuche

Dieses Analysebeispiel zeigt die Notwendigkeit der Analyse auf Implementierungsebene. Die beiden Schwachstellen sind in aktuellen Programmversionen bereits entfernt worden. Trotzdem können sie einerseits für spätere Arbeiten helfen und andererseits in nächsten Versionen versehentlich wieder eingebaut werden.

8.3.3 Zuverlässige Authentifizierung

Wichtig für die Sicherheit mobiler Geräte ist das Sperren der Bedienelemente des Systems bei Nichtbenutzung. Dazu gehört neben den Elementen der grafischen Benutzungsoberfläche auch alle Hardware-Bedienelemente wie Schalter und Tasten. Bei der verwendeten Hardware lassen sich bis auf den Rücksetzschalter alle Tasten durch die Software kontrollieren. Kann das Sperren noch zeitgesteuert und manuell durch den Benutzer ohne Komfortverlust erfolgen, so ist das Entsperren etwas aufwändiger.

8.3.3.1 Allgemeine Systembeschreibung

Für das beschriebene System wurde eine komfortable mehrstufige Benutzerauthentifikation gewählt. Dabei existieren mehrere Zustände des gesperrten Systems, die jeweils unterschiedlich starke Authentifizierungen und Anmeldungen verlangen. Die schwächste Stufe stellt eine Reihe grafischer Symbole dar, von denen die Auswahl eines Symbols das Gerät

entsperrt. Falscheingabe versetzt das Gerät immer in die nächst stärkere Sperrung des Systems. Die Anzahl der Stufen und die einzelnen Mechanismen sind frei definierbar. Im Prototypen, der hier analysiert wird, waren nur 2 Stufen integriert. Alle nicht privilegierten Prozesse erhalten im zweiten gesperrten Modus keine Rechenzeit mehr, da sie für den Scheduler als gestoppt markiert werden.

8.3.3.2 Modellierung der funktionalen Einheiten

Mit dem System werden Personen, bzw. der Benutzer, gegenüber dem System authentifiziert. Daher muss zusätzlich zu den bekannten Einheiten auch der Benutzer mit aufgenommen werden, wie Abbildung 8.7 zeigt.

Einheit	Beschreibung
Geräteschnittstellen	Menge aller möglichen Einflussgrößen
Persistenter Systemspeicher	Flash ROM, Systemquelle beim Start
Betriebssystem	Systemdienste, Netzwerk
Prozesse	Gestartete Anwendungen
Benutzer	Person

Tabelle 8.7: Funktionale Einheiten

8.3.3.3 Schnittstellenbildung

Die wichtigen Schnittstellen dabei sind alle Geräteschnittstellen, wie sie bei bisherigen Analysen bekannt waren.

8.3.3.4 Kontextbeschreibung

Nach dem durchgeführten Systemstart und einer erfolgreichen Anmeldung kann das System in einen gesperrten Zustand (lockedState) versetzt werden. Dies geschieht entweder durch eine Systemuhr nach einer Minute oder durch eine Benutzereingabe.

8.3.3.5 Funktionale Beziehungen

Die Funktionalität der gesicherten Authentifizierung wird in Abbildung 8.6 gezeigt.

```
Administrator/User: Auth lock()
Administrator/User: Auth unlock()
Timer 1 Minute: Auth lock()

func: Auth lock()
if STATE=unlocked then {
  SEQ: {
    clearScreen
    mesgToKernelModule
    STATE=lockedState1
  }
}

func: Auth unlock()
if STATE=lockedState1 then {
  SEQ: {
    <1 Minute Timer starten>
    SEQ: {
      in=getInput
      if in=SYMB1 STATE=unlocked
    }
    forall p in processlist.unpriv
      p.setState(STOPPED)
    STATE=lockedState2
  }
}

if STATE=lockedState2 then {
  SEQ: {
    in=getInput
    if in=SYMB2 STATE=unlocked
  }
}
```

Abbildung 8.6: Darstellung der funktionalen Beziehungen

8.3.3.6 Verfeinerung und Wahl der Aufruftiefe

Der Feinentwurf bringt bei dieser Analyse keine neuen Bedrohungen hervor.

Implementierung

Die Implementierungssicht ermöglicht die Analyse der Funktionen `clearScreen` und `mesgToKernelModule`, die aber von internen und externen Angreifern nicht ausnutzbar sind.

8.3.3.7 Mustervergleich der Angriffsklassen

Zunächst wird nach Schwachstellen im Grobentwurf mit der Guide-Words Methode gesucht:

9+10: (Late) Die Ausführungen der beiden Folgefunktionen von `clearScreen` können durch starke CPU-Auslastung verzögert werden. Dadurch könnte ein Angreifer dem Benutzer beim Sperren des Systems vortäuschen, dass es gesperrt ist, obwohl nur der Bildschirminhalt gelöscht wurde, die eigentlichen Sperrfunktionen aber noch nicht aufgerufen wurden. Es ist vorstellbar, dass diese dann nicht ausgeführt werden und das System ungesperrt bleibt.

19: (Other/Less/More) Es sind keine möglichen Eingaben erkennbar, die eine Schwachstelle darstellen (auch 30:).

Angriffe auf die Verfügbarkeit

Einziger erkannter Angriffspunkt ist der Angriff auf die Verfügbarkeit des Systems. Dies stellt eine angriffsvorbereitende Maßnahme dar, die durch hohe Systemauslastung erreicht werden kann. Zur Umgehung dieser Bedrohung wurde der Abschnitt in den Zeilen 8-10 synchronisiert.

8.3.3.8 Iteration der Bedrohungssuche

Die Iteration der Suche macht keine weiteren Bedrohungen sichtbar.

8.3.4 Sichere Erweiterbarkeit des Kerns

Eine Aufgabe eines Betriebssystems für mobile Endgeräte ist die Erweiterbarkeit des Kerns zur Laufzeit um fremden Code. Dieser kann von einer nicht vertrauenswürdigen Umgebung empfangen werden und einen Dienst zur Verfügung stellen. Das bedeutet, dass der Betriebsystemkern um neue Funktionalitäten erweitert wird und möglicherweise auch unsicherer Programmcode im priviligierten Kernkontext ausgeführt wird. Deshalb sind gewisse Einschränkungen gegenüber dem fremden Code zu treffen, die die Möglichkeiten eines Angriffs

auf den Kern unterbinden. Dazu zählt vor allem eine begrenzte Anzahl an nutzbaren Systemdiensten. Grundsätzlich muss aber ein Speicherschutzkonzept eingesetzt werden, dass innerhalb des Kernkontextes Speicherbereiche voneinander trennt. Dadurch wird erreicht, dass eingeladene Module nicht auf fremde Kernbereiche ungeschützt zugreifen dürfen. Dadurch entsteht ein Schutz, der eine Erweiterbarkeit des Kerns erlaubt, ohne dass dafür Archtiekturen wie Mikrokerne eingesetzt werden müssten. Das vorliegende Konzept basiert aber nicht wie andere Projekte auf einer formalen Beweisführung des fremden Codes, sondern liefert Schutzmechanismen zwischen dem Code und dem Betriebssystem.

8.3.4.1 Allgemeine Systembeschreibung

Dazu werden mehrere Protection Domains, also unterschiedlich priviligierte Schutzbereiche eingeführt. Einen Bereich benutzt das Betriebssystem, einen anderen der fremde Code. Der Wechsel zwischen den beiden Bereichen wird durch ein vertrauenswürdiges, vom Betriebssystem bereitgestelltes Modul `interpd` kontrolliert. Dieses Modul kontrolliert jeden Sprung zwischen den Protection Domains, die vorher bei einem zweiten vertrauenswürdigen Modul, dem `management`, registriert sein müssen. Beide Module müssen daher als funktionale Einheit bei der Sicherheitsanalyse auftreten. Die genauen Details über den Mechanismus und Implementierungsdetails sind in [Für03] nachzulesen. Die Implementierung ist auf die x86 Architektur abgestimmt. Sie eignet sich für den verwendeten Prozessor StrongARM im mobilen System nicht, da dieser keine derartige Segmentverwaltung besitzt. In den zukünftigen Architekturen der ARM-Familien soll diese Unterstützung aber integriert werden. Andere Ansätze zur abgesicherten Erweiterbarkeit finden sich in [Nec97] und [NL96].

8.3.4.2 Modellierung der funktionalen Einheiten

Zu den funktionalen Einheiten, die für die Analyse der sicheren Erweiterbarkeit des Kerns nötig sind, zählen auch die Hardwareressourcen wie Rechenzeit und Speicherverbrauch. Da das Ziel der Analyse eine Kernkomponente ist, können diese Ressourcen weitgehend unkontrolliert verbraucht werden. Eine einfache Ausnutzung wäre das Lahmlegen des Systems durch ständiges Belegen der Ressourcen, weitergehende Angriffe könnten durch verdeckte Kanäle auftreten. Die entsprechende Liste findet sich in Abbildung 8.8.

8.3.4.3 Schnittstellenbildung

Wesentlich für die Analyse sind die Schnittstellen zwischen dem fremden Code (bzw. dem potentiellen internen Angreifer) und dem betriebssystemspezifischen `interpd`-Code.

Einheit	Beschreibung
Persistenter Systemspeicher	Flash ROM, Systemquelle beim Start
Flüchtiger Systemspeicher	RAM
Betriebssystem	Systemdienste, Unterbrechungen, interpd und management Dienst
Prozesse	Gestartete Anwendungen
Hardwareressourcen	Rechenzeit- und Speicherplatzverbrauch

Tabelle 8.8: Funktionale Einheiten der abgesicherten Funktionalität zur dynamischen Kernerweiterung

8.3.4.4 Kontextbeschreibung

Ein Prozess kann durch den `interpd`-Dienst einen Stub aufrufen, der in die Kern-PD wechselt und dort priviligierte Dienste ausführt.

8.3.4.5 Funktionale Beziehungen

Für die Nutzung des Aufrufmechanismus wurden in dieser Arbeit zusätzliche Callgates eingerichtet, die einen Wechsel zwischen verschiedenen Privilegstufen ermöglichen. Sie stellen einen ähnlichen Mechanismus dar wie Softwareunterbrechungen und wechseln nach Aufruf an eine definierte Stelle im priviligierten Bereich. Der Trampolincode dazu wird in Abbildung 8.7 dargestellt. Wird von einem Programm aus der erweiterte Kern aufgerufen, so werden zunächst entsprechende Vorbereitungen zur Rückkehr durchgeführt, wie etwa die Bereitstellung der Rücksprungadresse auf dem Stack. Anschließend folgt der Befehl zur Ausführung des Call-Gates (Zeile 13), der durch die Hardware in den priviligierten Systemkontext wechselt und eine definierte, unveränderbare Stelle im Kern bzw. hier den `interpd`-Dienst aufruft.

In Zeile 13 wird das Callgate für den Wechsel zum `interpd`-Dienst ausgeführt. Der Grobentwurf der Architektur ist sehr einfach und wird aus Platzgründen nicht näher beschrieben.

8.3.4.6 Mustervergleich der Angriffsklassen

Die Schwachstellenanalyse des Grob- und Feinentwurfs bringt keine wichtigen Erkenntnisse, da die Aufrufe über die Callgates für jeden Prozess möglich sind. Ein möglicher Missbrauch ist dabei aber nicht erkennbar.

```
Process: usertrampoline()
2
func: Process usertrampoline()
4 if STATE=unprivUserlevel then {
  SEQ: {
6     [...] // gekuerzt
      rdtsc
8     movl %%eax, prof_userreturn
      movl %%edx, 4+prof_userreturn
10    popl %%edx
      popl %%eax
12    // den folgenden Befehl darf jeder Prozess ausfuehren!
      lcall 0x08,0
14    STATE=privKernelStubEntry
  }
16 }
```

Abbildung 8.7: Darstellung der Implementierung des Trampolincodes

Bei der Analyse der Implementierung (Abbildung 8.7) wird jedoch klar, dass jeder Prozess den Mechanismus des Trampolin nachbilden kann und das Callgate aufrufen darf. Die Aufrufberechtigung für ein Callgate ergibt sich schließlich nur aus der Privilegstufe des aufrufenden Segments und diese ist sowohl bei den Erweiterungen, als auch den Benutzerprozessen Ring 3. Ein Benutzerprozess kann damit unkontrolliert in den Kern springen.

Angriffe auf die Vertraulichkeit, Integrität und Verfügbarkeit

Die Vertraulichkeit, Integrität und Verfügbarkeit können von einem Angreifer gleichermaßen angegriffen und zerstört werden, wenn er den Mechanismus der Callgates ausnutzt. Er kann durch Erreichen des priviligierten Systemmodus jeden Speicherbereich des System lesen und beschreiben.

8.3.4.7 Iteration der Bedrohungssuche

In dieser Arbeit wurden keine weiteren Schritte für diesen Systemteil durchgeführt, da die Realisierung einer Schutzmaßnahme des erkannten Angriffs bereits sehr kompliziert ist und größere Änderungen an Bibliotheken und Anwendungen nach sich zieht.

8.3.5 Gesicherte Systemdienste

Ein Angreifer, der im Besitz des mobilen Endgerätes ist, kann Anwendungen im Benutzeradressraum starten und verwenden. Die Anwendungen werden als Black Box betrachet. Da es beliebig viele davon gibt und diese nicht alle auf Sicherheitslücken kontrolliert werden, ist die Schnittstelle zwischen Anwendungen und Betriebssystem aus Kernsicht zu kontrollieren.

8.3.5.1 Allgemeine Systembeschreibung

Diese Schnittstelle besteht aus einer kleinen Zahl (die aktuelle Anzahl hängt sehr von der verwendeten Betriebssystemversion ab und beträgt bei dem hier verwendeten System 253) von Systemdiensten, die bekannt sind, und somit kann jeder Aufruf kontrolliert werden. Basis für die Kontrollen sind Sicherheitsregeln, die dynamisch festlegbar sind. Diese Regeln können sowohl die Argumente, als auch die Gültigkeit des Aufrufers und die Menge der Aufrufe prüfen. Die Kontrolle findet durch den Kern bei der Durchführung des Kontextwechsels statt, bevor der Systemdienst ausgeführt wird.

Die Notwendigkeit der abgesicherten Systemdiensteschnittstelle wird in [Gör01] beschrieben. Es existieren auch andere Projekte, wie in [BGM00] gezeigt wird. Diese begrenzen den mechanischen Schutz jedoch auf die Abwehr gegen bekannte Angriffe und führen keinerlei Analyse gegen auffindbare Angriffe und Schwachstellen durch. Zudem ist es damit später praktisch nicht möglich, auf neu gefundene Angriffe zu reagieren, die nicht vom Rahmen des Abwehrmechanismus erkannt werden können.

8.3.5.2 Modellierung der funktionalen Einheiten

Die funktionalen Einheiten sind die selben, wie bei der dynamischen Kernelerweiterung, wie sie auch in Tabelle 8.8 angegeben sind.

8.3.5.3 Schnittstellenbildung

Als Schnittstellen werden hier nur die zwischen Prozess und Systemdiensteschnittstelle betrachtet. Diese Menge besteht aus allen Systemdiensten, die der Betriebssystemkern zur Verfügung stellt. Abstraktionen dieser Systemdienste, wie sie etwa Bibliotheken wie `libc` bieten, ändern nichts an der prinzipiellen Ausnutzbarkeit für Angriffe und werden deshalb hier nicht analysiert.

8.3.5.4 Kontextbeschreibung

Jeder Prozess in ausführbarem Zustand kann im System jeden Systemdienst aufrufen.

8.3.5.5 Funktionale Beziehungen

Bei der Darstellung des Grobentwurfs werden aufrufender Prozess und aufgerufener Systemdienst in Abbildung 8.8 dargestellt. Der Rücksprung zum aufrufenden Prozess geschieht nicht explizit, sondern durch den Aufruf des Schedulers, der die Rechenzeit für die Prozesse verteilt und so den nächsten rechenbereiten Prozess auswählt. Die Schedulingfunktionalität wird hier jedoch nicht näher untersucht.

```
// Zwischen syscall0-6 wird nicht unterschieden
Process: syscall()

func: Process syscall()
if STATE=unprivUserlevel then {
  SEQ: {
    eax=Systemdienstnummer (0-258)
    call Interrupt 0x80
    STATE=privKernellevel
  }
}

func: Kernel enterSyscall()
if STATE=privKernellevel then {
  SEQ: {
    SaveRegisters
    call syscall:eax
    RestoreRegisters
    // Zurueck zum Prozess
    Schedule
  }
}
```

Abbildung 8.8: Darstellung der funktionalen Beziehungen

In weiteren Interationsschritten wird der Assemblercode der Implementierung analysiert, wie er in Abbildung 8.9 dargestellt ist. Er führt die nötigen Kontrollen beim Einsprung in den Kern durch. Hier kann nur ein kleiner Ausschnitt aufgelistet werden, da die Teile des Kerns für die Verwaltung der Systemdienste recht komplex sind. Der Einsprungpunkt vom Benutzeradressraum aus nach der Ausführung der Softwareunterbrechung durch `int 0x80` wird in Zeile 4 angezeigt. Nach der Systemüberwachung (Zeilen 8 und 9) wird aus

einer entsprechenden Systemdienstetabelle die gewählte im Register `eax` übergebene Systemdienstenummer aufgerufen. Weitere Prüfungen finden nicht statt.

```
func: Kernel enterSyscall()
if STATE=privKernellevel then {
  SEQ: {
    pushl %eax
    SAVE_ALL
    GET_THREAD_INFO(%ebp)
    cmpl $(nr_syscalls), %eax
    jae syscall_badsys
    testb $(_TIF_SYSCALL_TRACE|_TIF_SYSCALL_AUDIT),
            TI_flags(%ebp)
    jnz syscall_trace_entry
    call *sys_call_table(,%eax,4)

    // gekuerzt
  }
}
```

Abbildung 8.9: Darstellung der funktionalen Beziehungen

8.3.5.6 Mustervergleich der Angriffsklassen

Nach der Anwendung der Guide-Words ergibt sich:

8: (More) Eine bedeutende Schwachstelle kann dadurch ausgenutzt werden, dass bestimmte Systemdienste zu oft aufgerufen werden. Bekanntes Beispiel ist der Systemdienst `sys_fork`, der bei zu häufigem Aufruf die Verfügbarkeit des gesamten Systems zerstören kann. Ursache hierfür ist die geringe Prüfung beim Einstieg in die Systemdienste.

Zudem wurden einige Systemdienste analysiert, die für diese Arbeit in den Kern integriert wurden. Dazu zählen die Systemdienste für den abgesicherten persistenten Speicher, Dienste zur Änderung der Bootlader Passwörter und Dienste zur zuverlässigen Authentifizierung. Dabei wurden allerdings keine wesentlichen Schwachstellen entdeckt.

Angriffe auf die Verfügbarkeit

Als Angriff konnte die häufige Benutzung einiger Systemdienste erkannt werden. Ein interner Angreifer kann dadurch das System durch extreme Ausnutzung der Ressourcen

Rechenzeit und Speicherplatz in unkontrollierbare Zustände bringen, die bis zum Absturz des Systems reproduziert werden können. Allerdings ist bisher kein Angriff erkennbar, der diese Ressourcenlast für gezielte Angriffe auf die Integrität oder Vertraulichkeit ausnutzen kann.

8.3.5.7 Iteration der Bedrohungssuche

Im nächsten Schritt werden aus dem Grobentwurf, der in Abbildung 8.8 dargestellt wird, die einzelnen Funktionen verfeinert. Die Verfeinerung der Funktion `func: Kernel enterSyscall()` wird in Abbildung 8.9 gezeigt.

8.3.6 Abgesicherter persistenter Speicher

Für schützenswertes Datenmaterial, wie etwa private Schlüssel oder persönliche Konfigurationsinformationen wird ein abgesicherter persistenter Speicher realisiert. Dieser hat die Aufgabe, nur den Prozess bzw. Benutzer Daten schreiben oder lesen zu lassen, für die das jeweilige Subjekt berechtigt ist. Der Speicher muss möglichst robust gegen jede Art von Vertraulichkeits- und Integritätsverlust sein. Zudem genügt es nicht, dass nur ein abgesicherter Bereich vorhanden ist, der beispielsweise mit nur einem Schlüssel gesichert ist. Es ist notwendig, dass für jeden Vertrauensbereich des mobilen Systems und für jeden Benutzer eigene Bereiche durch eigene Geheimnisse verschlüsselt werden. Theoretisch könnten benutzerbezogene Chipkarten die wichtigsten Daten sicher aufbewahren, sie haben jedoch folgende Nachteile:

1. *Niedrige Geschwindigkeiten*: Sowohl die Übertragungs- als auch die Verarbeitungsgeschwindigkeiten von Chipkarten sind wesentlich geringer als die des mobilen Systems. Das Übertragen und Entschlüsseln des Datenmaterials wäre sehr zeitaufwändig.

2. *Geringe Speicherkapazitäten*: Die Speicherkapazitäten des Chipkartenspeichers ist begrenzt. Darauf kann prinzipiell auf zwei Arten reagiert werden. Erstens werden tatsächlich nur die wesentlichen Daten auf dem Chipkartenspeicher abgelegt, wie persönliche Schlüssel. Zweitens werden die Daten verschlüsselt im Netzwerk abgelegt. Die Chipkarte ist die einzige Instanz, die diese Daten entschlüsseln kann, womit das Speicherproblem auf die Kommunikation abgewälzt wird. Dadurch wird der Benutzer aber abhängig von der Verfügbarkeit eines entsprechenden Netzwerks.

3. *Eigene Chipkarte pro Benutzer*: Für jeden Benutzer muss eine Chipkarten vorhanden sein. Gegen den Missbrauch bei Verlust der Chipkarte müssen entsprechende Maßnahmen eingeführt werden.

Chipkarten bedeuten einen höheren Hardware-technischen Aufwand als Softwarelösungen. Hier wird daher die implementierte reine Softwarelösung analysiert, die im folgenden Abschnitt kurz vorgestellt wird.

8.3.6.1 Allgemeine Systembeschreibung

Grundlegende Idee ist die Entwicklung eines abgesicherten Dateisystems, das Gruppenverschlüsselung unterstützt. Die Basisentwicklung war nicht Bestandteil dieser Arbeit, sondern wurde hier nur verfeinert und auf die StrongARM-Plattform umgesetzt. Genaue Details zur Dateisystemimplementierung finden sich auch in [EEG00] und [TO03].

Die Grundlage zur Verschlüsselung ist die Kryptofunktionalität des Kerns, die in [Her00] beschrieben wird und verschiedene symmetrische Verschlüsselungsverfahren bereitstellt. Die Verschlüsselung einer Datei geschieht aus Geschwindigkeitsgründen durch ein symmetrisches Verfahren mit dem Schlüssel $\mathcal{F}ilekey_{sym}$. Dieser Schlüssel wird nicht bekannt und nie im Klartext abgespeichert, sondern im Dateisystem mit allen öffentlichen Schlüsseln der Subjekte und Gruppen verschlüsselt und im sogenannten Repository abgelegt. Dadurch existiert eine Liste von verschlüsselten $\mathcal{F}ilekey_{sym}$ pro Datei. Vereinfacht ausgedrückt werden in dieser Liste auch die Zugriffskontrollrechte abgelegt. Beim Lesen einer Datei muss das Subjekt den $\mathcal{F}ilekey_{sym}$ durch die Anwendung seines privaten Schlüssels entschlüsseln, sonst bekommt es keinen Zugriff. Aus Gründen einfacher Verwaltung werden die Dateien und die Metadaten (Schlüssellisten, Dateinamen und Attribute usw.) nicht in einem eigenen Dateisystem gehalten, sondern in einer transparenten Zwischenschicht. Daher kann jeder Benutzer auch ohne Schlüssel die Dateien auflisten, ohne das der Dateiinhalt lesbar ist. Die gesamte Dateisystemstruktur wird durch das betriebssystemeigene VFS, einer Systemabstraktionsschicht für Dateisysteme, verwaltet. Dadurch müssen keine Operationen für verschiedene Blockgeräte oder sonstige Hardware bereitgestellt werden. Da sowohl Dateiinhalt, als auch Metadaten verschlüsselt werden, kann ein Angreifer typischerweise nichts mit diesen Dateien anfangen. Nach der erfolgreichen Annahmen eines Dateischlüssels kann der Benutzer, der vom Kern durch seine systemeindeutige Benutzer-ID identifiziert wird, solange die verschlüsselte Datei benutzen, bis er sich vom System abmeldet. Die Prüfung innerhalb des Kerns beruht nur auf der Prüfung der 32 Bit langen Benutzer-ID. Jeder Prozess, der unter dieser ID arbeitet, hat dann Zugriff auf die Datei. Das Konzept bietet den Vorteil, dass eine Schlüsselvergabe nicht nur für einzelne Benutzer, sondern auch für ganze Benutzergruppen erfolgen kann. Dazu werden Gruppenschlüsselpaare analog zur den Benutzerschlüsseln erzeugt, die dann ebenso für die Verschlüsselung von $\mathcal{F}ilekey_{sym}$ herangezogen werden können. Weiterer großer Vorteil des Konzeptes ist die Ablage der gesicherten Daten im Dateisystem. Der Zugriff darauf funktioniert daher wie üblich transparent über die vorhandenen Lese- und Schreiboperationen. Die Anwendungen müssen nicht weiter angepasst werden.

Ein anderer Ansatz zur Realisierung eines kryptografischen Dateisystems wird in [Bla93] verfolgt. Es werden jedoch keine Benutzergruppen unterstützt und eignet sich deshalb nicht

als abgesicherter persistenter Speicher.

8.3.6.2 Modellierung der funktionalen Einheiten

Zu den funktionalen Einheiten zählen bei dieser Analyse auch die Benutzer des Systems, da sie durch Passworte oder Chipkarten die privaten Schlüssel verwalten. Diese Einheiten werden in Abbildung 8.9 dargestellt.

Einheit	Beschreibung
Persistenter Systemspeicher	Flash ROM, Repository
Flüchtiger Systemspeicher	RAM
Betriebssystem	Systemdienste, Unterbrechungen, VFS-Schnittstelle
Prozesse	Dienstprogramme (Login, Schlüsselverwaltung)
Benutzer	Person

Tabelle 8.9: Funktionale Einheiten des abgesicherten persistenten Speichers

8.3.6.3 Schnittstellenbildung

Die Schnittstellen vom Benutzeradressraum aus beschränken sich auf die Schlüsselverwaltungsroutinen, den Login-Prozess und die VFS-Schnittstelle. Alle Operationen des abgesicherten persistenten Speichers lassen sich über diese Schnittstellen bedienen. Kann ein Angreifer den Betriebssystemkern kompromitieren und privilegierten Code ausführen, hat er auch die Kontrolle über den sicheren Speicher. Die Dateiinhalte sind zwar verschlüsselt, während eines Zugriffs eines autorisierten Benutzers befindet sich der Inhalt aber im virtuellen Speicher unverschlüsselt und könnte dann vom Angreifer ausgelesen werden. In einer monolithischen Betriebssystemarchitektur, wie der hier entwickelten, können Betriebssystemdienste niemals vor derartigen Angriffen geschützt werden, da nur eine privilegierte Schutzdomäne vorhanden ist. Diese internen Angriffe stellen demnach einen sehr ernsthaften Angriff dar. Besser als monolithische Architekturen wären deshalb beispielsweise Mikrokernarchitekturen, die aber nur wenige verfügbar und Neuentwicklungen zu aufwändig sind.

8.3.6.4 Kontextbeschreibung

Das Speichersystem lässt sich nur in hochgefahrenem Zustand benutzen, davor stehen die Systemoperationen nicht zur Verfügung. Die Authentifizierung erfolgt über die Prozess-

Identifikatoren (PIDs).

8.3.6.5 Funktionale Beziehungen

Die Darstellung des Grobentwurfs für die Administrationsschnittstelle folgt in Abbildung 8.10.

```
Administrator/User: SichererSpeicher admin()
Prozess/User: SichererSpeicher accessfile()

func: SichererSpeicher admin()
if STATE=addUser then {
  SEQ: {
    forall u in Userlist
      u != uid
    {
      generateKeys(uid)
      addUserInRepository
    }
    STATE=cont
  }
}

if STATE=delUser then {
  SEQ: {
    Userlist.uid.delete
    deleteUserInRepository
    deleteKeys(uid)
    STATE=cont
  }
}

// Weitere Operationen zur Schluesselverwaltung usw.
```

Abbildung 8.10: Darstellung der funktionalen Beziehungen

Die einzelnen Subsysteme wurden detailierter analysiert, wobei in Abbildung 8.11 die Schnittstelle von den Prozessen aus dargestellt ist. Diese ist für die Analyse der Prozessschnittstelle wichtig und zeigt den Ausschnitt der Lese- und Schreiboperationen.

```
Prozess/User: SichererSpeicher accessfile()

func: SichererSpeicher accessfile(file)
if STATE=Open then {
  SEQ: {
     file.open
     if !file.exists return NOFILE
     if file.uid != file.acl return NOACCESS
     STATE=cont
  }
}

if STATE=read then {
  SEQ: {
     if file.uid != file.acl return NOACCESS
     // Kernfunktion
     do_read
     STATE=cont
  }
}

if STATE=write then {
  SEQ: {
     if file.uid != file.acl return NOACCESS
     // Kernfunktion
     do_write
     STATE=cont
  }
}

// Weitere VFS-Operationen
```

Abbildung 8.11: Darstellung der Schnittstellen vom Prozessadressraum zum abgesicherten persistenten Speicher

8.3.6.6 Verfeinerung und Wahl der Aufruftiefe

Da das abgesicherte Speichersystem sowohl einen Teil im Benutzeradressraum als auch im Kernadressraum ausführt, ist die Analyse der Schnittstelle zwischen den beiden Privilegstufen wichtig. In den dargestellten Ausschnitten können diese bereits in den Grobentwürfen analysiert werden. Genaue Rechtekontrollen müssen aber in den Fein- und Implementierungsdarstellungen betrachtet werden.

Stufe des Grobentwurfs

Die prinzipielle Funktionsweise der einzelnen Systemteile wird im Grobentwurf sichtbar. Zudem existiert eine formale Beschreibung der Architektur, die zur Verfeinerung verwendet werden kann. Näheres findet sich auch in [Man02].

Implementierung

Eine genaue Beschreibung der Implementierungsdetails findet sich ausführlich in [EEG00]. Eine Ausarbeitung zur Versionenverwaltung des abgesicherten Speichersystems findet man in [TO03]. Zudem ist der Quellcode verfügbar.

8.3.6.7 Mustervergleich der Angriffsklassen

Die Sicherheitsanalyse des Systems ergibt im Wesentlichen einige logische Schwachstellen, die im Folgenden gezeigt werden. Die Suche nach Schwachstellen durch die Guide-Words Methode für die Administrationsschnittstelle ergibt:

7: (More) Synchronisationsbedingungen müssen geprüft werden, es kann bei mehreren Zugriffen auf `addUser` zu falschen Ergebnissen bis zur Zerstörung der `Userlist`-Struktur kommen.

11: (No) Es ist möglich (aber unwahrscheinlich), dass die Benutzerschlüssel angelegt werden (Zeile 10), dann aber der Benutzer nicht im Repository registriert wird. Ein Angreifer könnte dies etwa durch ein Zurücksetzen des Systems im richtigen Moment erreichen.

19: (More) Wird der letzte Benutzer einer Datei aus dem Repository gelöscht, so geht auch der letzte Schlüssel der Datei verloren. Sie kann dann nie wieder entschlüsselt werden. Deshalb wurde eine Regel eingeführt, dass jede Datei einen Administrator als Benutzer besitzen muss.

Für die Prozessschnittstelle ergibt sich:

8: (More/Early) Ein Benutzer ohne Zugriffsrechte auf eine Datei kann deren Existenz prinzipiell feststellen. Dazu öffnet er sie und erhält entweder `NOFILE` oder `NOACCES` als Ergebnis. Diese Schwachstelle kann für Angriffe auf die Vertraulichkeit ausgenutzt werden.

13-29: Für die Lese- und Schreiboperationen gilt allgemein, dass keine Schwachstellen gefunden wurden, aber die Synchronisationsmaßnahmen geprüft werden müssen. Es

könnte zu verschiedenen Nebenläufigkeitseffekten kommen, deren Ausnutzung durch einen Angreifer aber sehr unwahrscheinlich erscheint und dafür kein Beispiel gefunden wurde.

Angriffe auf die Vertraulichkeit

Wie beschrieben kann die Vertraulichkeit durch eine Schwachstelle bei der Rechteprüfung und Fehlerrückgabe angegriffen werden. Angriffe auf die Integrität konnten nicht nachvollzogen werden.

Angriffe auf die Verfügbarkeit

Die Verfügbarkeit kann durch das Löschen des letzten eingetragenen Benutzers einer Datei zerstört werden. Dadurch sind alle symmetrischen Schlüssel der Datei verloren und die Datei kann nicht mehr entschlüsselt werden. Es existieren einige Stellen im Quellcode, bei denen nicht synchronisiert wird. Diese könnten als Schwachstellen gegen die Verfügbarkeit dienen, Angriffe konnten aber keine gefunden werden.

8.3.6.8 Iteration der Bedrohungssuche

Der T_{AV}-Baum dafür wird in der Tabelle 8.11 dargestellt.

8.4 Integration

Die Analyse hat einige Schwachstellen und damit mögliche Angriffe hervorgebracht. Es gilt nun zwei Dinge zu beachten. Erstens müssen die entsprechenden Abwehrmaßnahmen in den Kern möglichst fehlerfrei integriert werden. Zweitens muss nach erfolgter Integration eine erneute Analyse durchgeführt werden. Diese Iteration muss bei jeder Änderung des Kern nachgezogen werden. Es hängt im Wesentlichen von der Komplexität des Systems ab, wie man mit diesen Iterationen umgeht und wann sie abgebrochen werden.

Nach erfolgter Sicherheitsanalyse müssen die Ergebnisse konkret umgesetzt werden. Dazu ist zunächst eine Bewertung der verschiedenen Bedrohungen notwendig um die Notwendigkeiten für einzelne Gegenmaßnahmen festzustellen. Typischerweise wird dazu eine Risikoanalyse durchgeführt, die allerdings für die gezeigten Analysen nicht vollständig gemacht wurde. Hier wurden nur die offensichtlichen Bedrohungen behandelt, da das Thema der Risikoanalyse ein eigenes großes Gebiet darstellt und den Rahmen dieser Arbeit gesprengt hätte. Im Folgenden werden deshalb nur die wichtigsten Punkte bei der Integration der Gegenmaßnahmen aufgeführt.

Hierarchie	Typ	Name	Risk
T 1	⌓	Bedrohungen gegen die Softwareinstallation	
T 1.1	⌓	Angriff auf die Vertraulichkeit	
T 1.1.1	○	Aktiver Angriff auf die Vertraulichkeit	
T 1.1.1.1	○	Gezieltes Öffnen von Dateien	
T 1.1.1.1.1	⬭	*VUL* `open` liefert bestimmten Fehler	
T 1.1.2	◇	Passiver Angriff auf die Vertraulichkeit	
T 1.2	◇	Angriff auf die Integrität	
T 1.3	⌓	Angriff auf die Verfügbarkeit	
T 1.3.1	⌓	Aktiver Angriff auf die Verfügbarkeit	
T 1.3.1.1	○	Löschen von Benutzern	
T 1.3.1.1.1	○	Löschen des letzten Benutzers, der letzten Schlüssel besitzt	
T 1.3.2	◇	Passiver Angriff auf die Verfügbarkeit	

Tabelle 8.11: Bedrohungsbaum der Analyse des abgesicherten Speichers

8.4.1 Realisierung in ein bestehendes System

Einen guten Überblick über die konkreten Realisierungsprobleme abgesicherter Systeme findet sich in [JV02]. Zur Realisierung algorithmischer Sicherheitsmaßnahmen, wie die Umsetzung von Zugriffskontrollmodellen, sei auf [Knu97] und [Knu74] verwiesen. Eine allgemeine Diskussion über den möglichen Schutz in Systemen wird in [Lam71] durchgeführt. Die Realisierung wird in [Luc99] beschrieben, die Implementierungen der Systemteile sind dort ebenfalls verfügbar.

8.4.2 Analyse und Nutzung von vorhandenen Mechanismen

Neben der Umsetzung neuer Sicherheitsmechanismen können auch oft Gegenmaßnahmen verwendet werden, die bereits im System vorhanden sind. Typisches Beispiel ist die Anwendung kryptografischer Methoden, die meist sowieso schon integriert sind und für weitere Mechanismen eingesetzt werden können. Dabei wird aber davon ausgegangen, dass die Mechanismen fehlerfrei arbeiten, soweit sie nicht einer weiteren Analyse unterzogen wurden.

8.4.3 Einschränkungen

Um nicht nur die leicht erkennbaren Bedrohungen zu erkennen und mit entsprechenden Gegenmaßnahmen zu reagieren, ist eine Risikoanalyse nötig. Diese bewertet letztlich den Grad der Gefährdung und des Schadens, ist aber nicht Bestandteil dieser Arbeit. Der große Vorteil des hier entwickelten Analyseverfahrens, der ohne durchgeführte Risikoanalyse aber nicht voll ausgenutzt wird, ist durch die hierarchische Darstellung gegeben. Dadurch können bei der Risikoanalyse sowohl die Gefährdungen als auch die Schäden berechnet werden. Weiterhin muss die Sicherheitsanalyse nach erfolgter Integration erneut durchgeführt werden. Dies muss nicht das gesamte System betreffen, sondern vor allem die geänderten Systemteile. Bestehende Vorgehensweisen zur Risikoanalyse sind mit dem hier gezeigten Sicherheitsanalyseverfahren vereinbar und können direkt eingesetzt werden.

8.5 Sicherheitsmanagement

Das Sicherheitsmanagement stellt die Kombination aus den Sicherheitsmechanismen und deren statischen und dynamischen Regeln dar. Es muss in den Betriebssystemschichten sehr weit unten realisiert werden, damit es einerseits gut geschützt für das gesamte System verfügbar und andererseits schon früh beim Systemstart für andere Dienste vorhanden ist. Da der Begriff sehr weitreichend ist, muss man zwischen statischem und dynamischem Management unterscheiden. Unter statischem Sicherheitsmanagement versteht man einen Dienst, der alle Policies zu einem bestimmten Zeitpunkt vor oder während der Laufzeit des Systems erhält. Die Regeln können von außen durch Benutzer oder andere Prozesse erzeugt werden oder auch vom Management selbst generiert werden. Mit dynamischem Sicherheitsmanagement bezeichnet man den Sicherheitsdienst, der kontinuierlich (zu diskreten Zeitpunkten) die Policies ändert und anpasst. Für die Entscheidungen des Sicherheitsmanagements sind die hier durchgeführten Analysen notwendig. Außerdem ist es vorstellbar, dass die Ergebnisse der Analysen in verarbeitbarer Form an das Management übergeben werden und die Entscheidungen dynamisch darauf angepasst werden. Für mobile Systeme kann das etwa bedeuten, dass die Umgebung des Systems Analyseergebnisse bereithält und diese dem jeweiligen Management eingibt. Dadurch kann das Sicherheitsmanagement die Policies kontextbezogen erzeugen und umsetzen. Dafür fehlt allerdings noch die Integration eines derartigen Managements in das gezeigte System, was nicht Gegenstand dieser Arbeit ist.

8.6 Ergebnisse

Ziel der Umsetzung der Sicherheitsanalyse ist es, ein Betriebssystem abzusichern. Die Beschränkung auf mobile Systeme ist für diese allgemeine Analysetechnik nicht notwendig,

auch wenn das in dieser Arbeit und hier gezeigte Zielsystem für mobile Geräte vorgesehen ist. Die entstandene Lösung ist gleichermaßen für die Analyse von stationären, wie auch von mobilen Systemen anwendbar. Dabei stehen keine speziellen Sicherheitsaspekte im Vordergrund, sondern es soll ein umfassender Schutz gegen die vorstellbaren Bedrohungen erzeugt werden. Die grundsätzlichen Überlegungen zur Architektur des Systems können von Grund auf neu oder als Integration in ein bestehendes System realisiert werden. Wichtig dabei ist jedoch immer, dass man durch Analysen des Gesamtsystems die erreichte Absicherung prüft. Diese Analysen wurden für bestimmte ausgewählte Systemteile durchgeführt. Die Ergebnisse wurden teilweise zur Verbesserung und Absicherung des System verwendet. Mit der entwickelten Analysemethode TANAT lassen sich Bedrohungen finden, die nicht naheliegend sind und somit unter Umständen nicht gefunden werden. Der Aufwand der Analyse kann abhängig vom Detailgrad groß werden, insbesondere hängt der Aufwand immer direkt von der Größe des Systems und der Anzahl der modellierten Einheiten ab.

Kapitel 9

Schlussbetrachtung

In diesem Kapitel werden die wichtigsten Ergebnisse der Arbeit zusammengefasst. Es werden offene, wissenschaftliche Fragestellungen erörtert, die in weiterführenden Arbeiten behandelt werden können. Abschließ end wird ein Ausblick darüber gegeben, welchen Gewinn man durch die Ergebisse bekommt und welche Verbesserungen wünschenwert sind.

9.1 Zusammenfassung und Fazit

Ziel dieser Arbeit ist die Konstruktion eines abgesicherten Betriebssystems für mobile verteilte Systeme. In den letzten Jahrzehnten wurden zahlreiche stationäre Betriebssysteme entwickelt, die besondere „Security“-Eigenschaften besitzen und deshalb gegen spezielle Angriffe und Bedrohungen resistent sein sollen. Zu Beginn der 70er Jahre waren die Entwicklungen sehr aufwändig. Man konzentrierte sich auf den Einsatz formaler Entwicklungsmethoden, denen meist einen Top-Down Ansatz zugrunde lag und mathematisch beweisbar sichere Systeme liefern sollten. Einige Systeme wurden mit hohem Aufwand derart entwickelt und in besonders sicherheitskritischen Bereichen (Militär, Regierung usw.) eingesetzt. Doch sowohl der Aufwand der Neuentwicklung, als auch die Wartung dieser Systeme war so teuer, dass sie sich für den Großteil der heutigen Systeme kaum eignen und nur an wenigen, besonders kritischen Stellen eingesetzt werden. Bei der Konstruktion funktional korrekter Systeme sind derartige Entwicklungsmethoden erfolgreich. Bei der Absicherung von Systemen gegen beliebige, meistens bei der Entwicklung unbekannte Angreifer und Bedrohungen scheitern diese jedoch in Bezug auf eine umfassende Anwehr. Dies liegt hauptsächlich daran, dass die Angreifer und die möglichen Bedrohungen während der Entwicklung nicht bekannt sind. Die Angreifer versuchen zur Laufzeit des Systems mit beliebigen Mitteln das System zu kompromitieren. Da bei der Auswahl dieser Mittel dem Angreifer keine Grenzen gesetzt sind, ist es sehr wahrscheinlich, dass ein Angriffsversuch zum Erfolg führen wird. Neben der unbekannten Art und Weise, wie ein Angreifer vorgeht, ist zudem nicht bekannt, welche Ziele er verfolgt. Insbesondere gibt es

kaum Zusammenhänge zwischen der Höhe des möglichen Schadens durch einen Angriff und dem Aufwand für den Angreifer. Dies führt zu der merkwürdigen Situation, dass sogenannte „Script-Kiddies“ durch sehr geringen Aufwand Angriffe produzieren können, die Miliardenschäden verursachen können. Außerdem unterscheiden sich die Angriffe auf IT-Systeme von Angriffen im täglichen Leben in einigen Punkte sehr stark. Hervorzuheben ist dabei, dass ein Angreifer den Angriff automatisieren kann, von fast beliebigen Stellen eines Netzwerks aus zuschlagen und die Angriffe technisch gesehen beliebig oft wiederholen kann. Die Ergebnisse eines Angriffs können anschließend kopiert und vervielfältigt werden. Gegenmaßnahmen aus dem täglichen Leben können daher nur in sehr beschränktem Umfang in die IT-Sicherheit übertragen werden und das Sicherheitsbewusstsein und Vertrauen eines Benutzers müsste auf diese Systeme eingestellt werden.

Diese Arbeit beschäftigt sich mit der Entwicklung einer Methode zur Analyse und Konstruktion abgesicherter Softwaresysteme. Dabei werden neben den systemeigenen Funktionalitäten auch die Eigenschaften und Einflüsse der Systemumgebungen berücksichtigt. Diese Analysemethode ermöglicht es, entwicklungsbegleitend anhand von Systemmodellen nach möglichen Schwachstellen, Angriffen und Bedrohungen zu suchen. Als Ergebnis entsteht eine hierarchisch strukturierte Menge aller dieser Sicherheitsprobleme, die neue, nicht naheliegende Schwachstellen, Angriffen und Bedrohungen liefert. Dieses Ergebnis wird anschließend zur Risikobewertung genutzt und stellt die Grundlage zur Auswahl entsprechender Gegenmaßnahmen dar.

Diese Arbeit liefert zu Beginn eine Einführung anhand konkreter Beispiele in die IT-Sicherheit und stellt dabei die charakteristische Problemstellung dar. Anschließend werden für die wesentlichen Begriffe Definitionen angegeben, die sich von den bekannten Begriffsdefinitionen aus der Literatur durch die Berücksichtigung des Internetstandards unterscheiden. Anschließend werden konkrete Bedrohungen und Schwachstellen gezeigt und daraus Bedrohungen und Angriffe nach verschiedenen Kriterien klassifiziert. Die Vorstellung allgemeiner, bekannter Sicherheitsmaßnahmen und deren Einteilung führt zum Vergleich bestehender abgesicherter Betriebssysteme. Die für den stationären Betrieb entwickelten Betriebssysteme bieten unterschiedliche Schutzmechanismen und wurden mit verschiedenen Entwicklungsmethoden konstruiert. Die nur wenig verfügbaren betrachteten Betriebssysteme für den mobilen Bereich sind meist kommerziell und bieten demgegenüber nur wenig neues. Teilweise sind diese sogar schlechter abgesichert, als ihre älteren stationären Vertreter, was hauptsächlich an den begrenzten Ressourcen liegt. Keines der betrachteten Systeme analysiert die möglichen Schwachstellen, Angriffen und Bedrohungen, so dass sich die Gegenmaßnahmen daher nur auf bei der Entwicklung bekannter Probleme beziehen. Eine strukturierte Sicherheitsanalyse findet nicht statt. Da für die Analysemethode, die in dieser Arbeit entwickelt wird, Modellbeschreibungen nötig sind, werden danach verschiedene Vertreter von Modellierungsverfahren verglichen. Dieser Vergleich kann aber nur eine kleine Menge der verfügbaren Methoden abdecken, so dass die ausgewählte Methode ersetzbar bleibt. Am Ende des Analyseteils dieser Arbeit wird das Konzept der systematischen Sicherheitsanalyse vorgestellt. Dabei werden auch die Teilverfahren diskutiert, die in

diesem Bereich verfügbar sind, die aber immer nur einen Teil der Schwachstellen-, Angriffs und Bedrohungsanalyse abdecken.

Im Syntheseteil wird zunächst das Ziel der integrierten Sicherheitsanalyse vorgestellt, wobei einzelne Methoden aus Schwachstellen-, Angriffs und Bedrohungsanalyse zu einem systematischen Gesamtansatz kombiniert werden. Anschließend werden die unterschiedlichen Aspekte bei der Schwachstellenanalyse und der Angriffssuche aufgezeigt. Zur strukturierten Darstellung der Analyseergebnisse werden im Folgenden sogenannte T_{AV}-Bäume eingeführt, die durch die beschriebene Methode TANAT entwickelt werden. Die Entwicklung der integrierten Sicherheitsanalyse schließt mit einem Ausblick auf die folgende Phase der Risikoanalyse oder einem Überblick über die allgemeine Komposition des Gesamtsystems. Für verschiedene Teile des Analyseprozesses sind Werkzeuge sinnvoll und hilfreich. Die im Rahmen dieser Arbeit entwickelten werden ausführlich beschrieben und an sinnvollen Stellen beschrieben, wie die Schnittstelle zur Nutzung und Erweiterung der Tools später genutzt werden kann.

Im letzten Teil, der Umsetzung, wird beschrieben, wie die entwickelte Methode bei der Konstruktion eines abgesicherten Betriebssystems eingesetzt wurde. Als Ausgangspunkt steht dabei ein bestehender Betriebssystemkern und die verwendete Hardwarearchitektur. Darauf aufbauend wurden verschiedene Basissicherheitsmechanismen entwickelt, die dann das Ziel der integrierten Sicherheitsanalyse darstellen. Die einzelnen Analysen dieser Systemteile werden in den Schritten des Prozesses dargestellt. Da sowohl die konzeptionellen Aspekte des Sicherheitsmechanismus, als auch die Realisierungsdetails analysiert werden, können diese Abschnitte aus Platzgründen nur knapp gehalten werden. Die wesentlichen Ergebnisse der durchgeführten Analysen und die prinzipielle Vorgehensweise werden aber klar dargestellt.

Die Integration der verschiedenen Analysemethoden zu einer Sicherheitsanalyse und die Entwicklung eines Bedrohungsbaums daraus hilft bei der Absicherung von Systemen sehr. Sowohl die Auswahl der Gegenmaßnahmen, als auch die Dokumentation der möglichen Schwachstellen, Angriffe und Bedrohungen sind von großem Wert für die Entwicklung, Wartung und das Vertrauen der Benutzer. Die eingesetzte Modellorientierung als Basis zur Analyse bewirkt, dass sowohl realisierungsnahe Systembeschreibungen bis hin zum Quelltext, und auch abstrakte, vereinfachte Schemata des Systems analysiert werden und Bedrohungen gefunden werden können. Die entstandenen Werkzeuge sind für einen weiteren Einsatz nützlich und können für verschiedene Einsatzgebiete angepasst werden. Zur Risikoanalyse und der Auswahl der Gegenmaßnahmen sind sie äußerst nützlich. Die analysierten Systemteile und die gefundenen Bedrohungen bestätigen die Einsetzbarkeit der Methode.

9.2 Ausblick

Neben den Analysen von verschiedenen Betriebssystemteilen, wie es hier in dieser Arbeit vorgeführt wurde, können auch beliebige andere Software-Architekturen mit dieser Methode auf Sicherheitsaspekte hin untersucht werden. Dazu zählen nicht nur systemnahe Softwaremodule, sondern ebenso Middleware-Architekturen oder Anwendungen. Das grundlegende Vorgehen unterscheidet sich dabei nicht von der vorgestellten Art. Das Wissen über bekannte Schwachstellen und Angriffe kann jedoch je nach Anwendungsdomäne verschieden sein. Anknüpfungspunkte der systematischen Bedrohungsanalyse finden sich ebenso zu bestehenden Entwicklungsmethoden als auch zu vorhandenen Verfahren zur Risikoanalyse. Diese beiden Entwicklungsphasen wurden in dieser Arbeit betrachtet um Zusammenhänge darzustellen.

Da diese Zusammenhänge allerdings nicht vollständig untersucht wurden, ergeben sich mögliche weitere Arbeitsschritte. Der Einsatz der Methode in Kombinationen mit speziellen Entwicklungsmethoden kann für die einzelnen Methoden untersucht werden. Insbesondere ist jeweils zu klären, in welcher Phase des Entwicklungsprozesses die integrierte Sicherheitsanalyse eingesetzt werden soll und wie deren Ergebnisse iterativ den Entwicklungsprozess begleiten können. Ebenso ist eine vollständige, nachvollziehbare Risikoanalyse und die möglichst fehlerfreie Umsetzung der Sicherheitsarchitektur nötig.

Neben der Untersuchung des Einsatzkontextes sind auch direkte Verbesserungen der entwickelten Methode denkbar. Die Schwachstellenanalyse kann durch Werkzeugunterstützung für weitere Quellsprachen erweitert werden. Außerdem kann auch die modellhafte Untersuchung von Netzwerkarchitekturen auf Schwachstellen ein weiteres Arbeitsfeld werden, dessen Ergebnisse in die Konstruktion der T_{AV}-Bäume einfließen können. Die Werkzeuge können sowohl für andere Modellierungsarten, als auch zusätzliche Bewertungsverfahren für die Gegenmaßnahmen erweitert werden.

Die entstandenen Verbesserungen durch das integrierte, systematische Verfahren sind ein strukturiertes Vorgehen und strukturierte Ergebnisse bei der Bedrohungsanalyse. Auch wenn nicht bewiesen werden kann, dass dadurch alle Schwachstellen, Angriffe und Bedrohungen gefunden werden, so liefert diese Methode doch neue Angriffsmöglichkeiten und potentielle Schwachstellen für ein gegebenes (Teil-) System.

Anhang A

TANAT Programmierschnittstelle

Einsatz

Im Folgenden werden einige wichtige Punkte für die Erweiterung des Werkzeugs TANAT beschrieben. Besonders wichtig ist dies für mögliche Erweiterungen, die TANAT modular hinzugefügt werden können.

Benutzungsoberfläche

Die Benutzungsoberfläche stellt eine Möglichkeit dar, der die wichtigen Fähigkeiten von TANAT einfach benutzbar macht. Da die Bedienung relativ intuitiv ist und durch kleine Hilfen wie ein Toolbar- und Hilfesystem unterstützt wird, wird sie hier nicht näher beschrieben. Es muss aber darauf hingewiesen werden, dass TANAT auch als Bibliothek vollständig ohne die Bedienoberfläche verwendet werden könnte. Dies ist dann notwendig, wenn das Werkzeug in ein anderes bestehendes System integriert wird. Dazu wäre lediglich die Hauptfunktion in der Datei `tanat.c++` anzupassen und alle Klassen mit dem Namespräfix `Main...` müssen dann nicht mehr verwendet werden.

Dateiformat

Das Dateiformat wurde so entwickelt, dass es neben der automatischen Verarbeitung auch einfach manuell zu erstellen und zu verändern ist. Im Wesentlichen muss dabei eine Hierarchie in eine flache textuell dargestellte Struktur überführt werden. Dazu werden die einzelnen Knoten und Kanten durchnummeriert. Diese Bezeichner bestimmen dann sowohl Hierarchieebene als auch die Verbindung zum Vaterknoten. Neben einer Bezeichnung werden auch die Attribute abgespeichert. Momentan ist die Grammatik nur auf eine be-

stimmte Menge von bekannten Attributen ausgelegt. Diese sind für die gezeigte Methode TANAT und eine anschließende Risikoanalyse und Bewertung ausreichend. Eine mögliche Erweiterung wäre es jedoch, die Daten in einer XML-Struktur abzulegen, so dass beliebige Attribute abgespeichert werden können. Die Schlüsselworte und die Grammatik für das einfache Dateiformat sind im Anhang B dargestellt.

Berechnungsschnittstelle

Eine Basisfunktionalität zum Umgang mit den T_{AV}-Bäumen ist die Berechnung und Auswertung der Attribute. Das Werkzeug wurde so strukturiert, dass verschiedenste Berechnungsmethoden als Modul hinzugefügt werden können. Wichtig dafür ist aber die Programmierschnittstelle, die den Zugriff auf die Baumdatenstrukturen mit den Attributen ermöglicht. Der Vorgang dabei ist stets der, dass ausgehend von einem Knoten, beispielsweise von der Wurzel aus, ein oder mehrere Attribute berechnet werden sollen. Dazu sind in Kanten und Knoten Werte gegeben, aus denen die Attribute berechnet werden. Meistens werden aus Werten, die in Blättern gespeichert sind, die Attribute bis zu einer Teilbaumwurzel berechnet, wie es etwa bei der Risikoberechnung und der Schadensberechnung gemacht wird. Das Programmkonstrukt für die Berechnung von Blättern zu einer Wurzel wird in Abbildung A.1 gezeigt.

```
void Tree::gotoTreeVertexParents (TreeVertex* tv,
                                  TreeVertex* root)
{
  class TreeVertex* parentVertex;
  parentVertex = tv->getParentVertex ();
  if (parentVertex == NOVALIDVERTEX ||
                tv == root) return;
  // BERECHNUNG
  [...]
  calcTreeVertexChilds (parentVertex, root);

}
```

Abbildung A.1: Programmkonstrukt zur Berechnung der Attribute

Ebenso ist es nötig, dass bei einem gegebenen Knoten als Wurzel eines Teilbaums die Attribute zu den Blättern hin berechnet werden. Dazu ist das Durchlaufen bis zu allen Blättern nötig, dabei werden schrittweise die Attributberechnungen durchgeführt. Das Programmgerüst zur Attributberechnung ist in Abbildung A.2 gezeigt.

```
void Tree::gotoTreeVertexChilds (TreeVertex* tv)
{
  std::deque<class TreeVertex*> childVertex;
  childVertex = tv->getChildVertexList ();
  for (int i=0; i < childVertex.size(); i++)
  {
    // BERECHNUNG
    [...]
    gotoTreeVertexChilds (childVertex[i]);
  }
}
```

Abbildung A.2: Programmkonstrukt zur Berechnung der Attribute

Anhang B

TANAT Dateiformat

Schlüsselworte

And	Or	Xor	Not
Name	Description	Knowledge	Risk
Probability	Damage	Note	Property
Detail			

Tabelle B.1: Schlüsselworte des Dateiformats

Grammatik

subtrees	::=	subtrees subtree
subtree	::=	numdescription textdescription
		|| numdescription **And** textdescription
		|| numdescription **OR** textdescription
		|| numdescription **XOR** textdescription
		|| numdescription **NOT** textdescription
numdescription	::=	< numorpointlist >
numorpointlist	::=	[0-9]*
		|| numorpointlist . [0-9]*
textdescription	::=	{ attributeslist }
text	::=	"[a-zA-Z0-9]*"
attributeslist	::=	attributes attributeslist
attributes	::=	**Name** text ;
		|| **Description** text ;
		|| **Knowledge** text ;
		|| **Risk** text ;
		|| **Probability** text ;
		|| **Damage** text ;
		|| **Note** text ;
		|| **Property** text ;
		|| **Detail** text ;

Tabelle B.2: Grammatik zum Dateiformat

Anhang C

Sicherheit durch Open Source

Der Begriff Open Source ist seit 1997 definiert und legt Regeln für eine Softwarelizenz fest. Open Source bedeutet daher nicht „lizenzfrei“. Diese 10 Regeln lauten wie folgt.

1. „Free Redistribution“, die Weiterverbreitung der Software ist jederzeit möglich.
2. „Source Code“, die Quelltexte der Software sind verfügbar.
3. „Derived Works“, verwendete und abgeänderte Software anderer Urheber sind als solche zu erwähnen.
4. „Integrity of The Author's Source Code“, Urheberschaft muss erkennbar bleiben
5. „No Discrimination Against Persons or Groups“ und
6. „No Discrimination Against Fields of Endeavor“, keine Diskriminierungen
7. „Distribution of License“, Weitergabe der Lizenz mit der Software.
8. „License Must Not Be Specific to a Product“, Lizenz soll nicht nicht produktspezifisch sein und
9. „The License Must Not Restrict Other Software“ darf keine andere Software beeinflussen.
10. „The License must be technology-neutral“, die Lizenz soll technologieneutral sein.

Es existieren zahlreiche unterschiedliche Lizenzen, die dem Open Source Rahmen entsprechen, allen voran die GPL. Diese Lizenz ist auch für den Linux-Kern verwendet. Allen Lizenzen ist gemein, dass sie die Sicherheit nicht erwähnen, aber vor allem von der Quelltexteinsicht leben. Diese Freigabe der Quellen täuscht Sicherheit des Systems nur vor, beabsichtigt ist eine Absicherung damit zunächst nicht. Schließlich ist es immer fraglich, wer eine potentielle Schwachstelle zuerst findet, ein Angreifer oder jemand, der das System untersucht. Zudem darf man sich nicht darauf verlassen, dass alleine die Verfügbarkeit der Quellen automatisch eine Sicherheitsanalyse folgen lässt. Umgekehrt kann auch die Aussage

sinnvoll sein, dass die Quelltexte geheim bleiben sollen, damit kein Angreifer Schwachstellenanalyse betreiben kann. Dieses Vorgehen liefert natürlich auch keine sichere Software, allerdings gewinnt der Softwarehersteller dadurch eventuell Zeit bei der Bekämpfung von Schwachstellen gegenüber Angreifern. Dagegen ist zu sagen, dass Quelltextanalyse die einfachste und naheliegendste Analysetechnik ist. Die Übersetzung in die Zielsprache alleine ist keine Analyseverhinderung und kann teilweise sogar rückgängig gemacht werden. An einem kleinen Beispielprogramm in Java C.1 wird dies deutlich.

```
1 class Hallo
2 {
3   public static void main (String[]args)
4   {
5     String Test="HalloTest";
6     System.out.println ("Hallo Welt!");
7   }
8 }
```

Abbildung C.1: Java Quelltext

Der Quelltext wird mit einem Standard Java-Übersetzer in den Bytecode transformiert und dann rückübersetzt. Das durchaus lesbare und mit dem Ausgangsquelltext vergleichbare Produkt ist in Listing C.2 zu sehen.

Bei der Rückübersetzung aus einer Maschinensprache in eine Hochsprache gehen meistens die Bezeichner verloren. Das angegebene Beispiel in Java behält viele Bezeichner auch nach einer Rückübersetzung, was eine Analyse sehr einfach macht. Andere Sprachen bzw. deren Compiler entfernen viele Bezeichner aus den Maschinenprogrammen, so dass nach einer Rückübersetzung im Wesentlichen automatisch generierte Namen für Funktionen, Variablen usw. vergeben werden. Ein Großteil des Codes, auch in einer Hochsprache, kann dadurch nur schwer gelesen werden. Trotzdem bleibt die Rückübersetzung von Code auch dann ein wichtiges Mittel zur Analyse, die durch Übersetzung nicht verhindert werden kann.

Daneben existieren verschiedene Verfahren zur Codeverwürfelung, die dazu führen, dass neben dem Verschwinden der Bezeichner auch die Strukturen des Programmcodes verloren gehen. Dadurch können Programmhersteller die Möglichkeiten von Rückübersetzung weitgehend verzögern, vermeiden lässt sich dies nie. Beispiele für Codeverwürfelung findet sich etwa in [Hoh98].

Wie bei Kryptoverfahren gilt bei Quelltexten offensichtlich ebenso, dass die Sicherheit durch Geheimhaltung der Quellen eine vorgetäuschte Sicherheit darstellt. Vernünftig ist es, gute Verfahren auch für jeden überprüfbar zu machen und die Sicherheit einzig vom Wissen oder Besitz des Benutzers (PIN, Chipkarten usw.) abhängig zu machen. Eine Diskussion

```
> javap -c Hallo

Compiled from Hallo.java
class Hallo extends java.lang.Object {
    Hallo();
    public static void main(java.lang.String[]);
}

Method Hallo()
   0 aload_0
   1 invokespecial #1 <Method java.lang.Object()>
   4 return

Method void main(java.lang.String[])
   0 ldc #2 <String "HalloTest">
   2 astore_1
   3 getstatic #3 <Field java.io.PrintStream out>
   6 ldc #4 <String "Hallo Welt!">
   8 invokevirtual #5 <Method void println(java.lang.String)>
  11 return
```

Abbildung C.2: Java Bytecode rückübersetzt

zu diesem Thema fand auch in [Gör03] statt.

Anhang D

Hardwarebasis der Realisierung

Da Kenntnisse über die Hardware bei der Realisierung der Teile im Kern und des Bootladers und deren Sicherheitsanalyse nötig sind, werden im Folgenden die wesentlichen Fakten der eingesetzten Hardware beschrieben.

Speicheraufteilung nach dem Start des Bootladers

Start	Länge	Beschreibung
`0xd00000`	`0x200000`	Halde
`0xf00000`	`0x40000`	Bootlader RAM Bereich
`0xf40000`	`0x4000`	Seitentabelle
`0xf44000`	`0x3c000`	Keller
`0xf80000`	`0x80000`	Puffer für Kernabbild

Tabelle D.1: Speicherbereiche während der Bootlader aktiv ist

Komponenten für Rechenfähigkeit:

- Intel StrongARM SA-1110 32-bit Prozessor
- Klasse der ARM V4 Architektur
- 206 MHz maximal Takt

- 100 MHz Takt für den Speicherbus
- 16 kByte Instruktions-Cache
- 8 kByte write-back Daten-Cache
- Zusätzlich ein 512-byte Mini-Daten Cache
- Zwei Oszillatoren (PLL) mit 3,6864 MHz und 32,768 MHz

Komponenten für Speicherfähigkeit:

- 32-fache MMU
- 32-facher Assoziativspeicher
- SDRAM und SMROM (Synchronous Mask ROM) sind möglich
- Speicher Controller integriert

Komponenten für Kommunikationsfähigkeit:

- 28 Ein-/Ausgabeports zur freien Verfügung, interruptgesteuert
- Echtzeituhr
- Watchdog
- Intervall Timer
- Power Management Controller
- Interrupt und Reset Controller
- 6-Kanal DMA Controller
- LCD Controller
- SDLC Controller
- PCMCIA/PCCard Controller, 2 Slots
- 16550 UART für serielle Schnittstellen, bis zu 230 kBit/s
- IrDA kompatibler Controller, fast und slow IrDA

- SSP (Synchronous Serial Port), UCB1100, UCB1200, SPI, TI, Wire
- USB Endpoint Interface, 12 MBit/s
- Codec für Audio

Power Management

Der StrongARM arbeitet mit folgenden Frequenzen:

- bei 150 MHz: 150 Dhrystone, 2,1 MIPS
- bei 206 MHz: 235 Dhrystone, 2,1 MIPS

Power Management Modi

Folgende Power Management Modi kennt der SA-1110:

- Normal mode (full-on)
- Idle mode (power-down)
- Sleep mode (power-down)

Stromverbrauch

Stromverbrauch im Normal Mode:

- $<$ 240 mW bei 1,55 Volt Core-Spannung bei 133 MHz, etwa 155 mA
- $<$ 400 mW bei 1,75 Volt Core-Spannung bei 206 MHz, etwa 230 mA

Durch Software einstellbare Taktfrequenzen

Die Frequenzen der Clock-Generatoren können über I/O-Ports verstellt werden.

CCF [4...0]	Prozessortakt [MHz],
00000	59,0
00001	73,7
00010	88,5
00011	103,2
00100	118,0
00101	132,7
00110	147,5
00111	162,2
01000	176,2
01001	191,7
01010	206,4
01011	221,2

Tabelle D.2: Programmierung des Prozessortaktes

Tabelle der programmierbaren Frequenzen

Programmieren der Frequenz

Nach dem Einschalten arbeitet der SA-1110 Prozessor mit der niedrigsten Frequenz und die Software (Bootlader oder Betriebssystem) muss sich um die Einstellung kümmern.

Dazu kann das Register CCF (Core Clock Configuration Field) im PPCR (Power Management PLL Configuration Register) geändert werden, wie Tabelle D.2 darstellt. Das Betriebssystem kann dieses Register auch im normalen Betrieb ändern um beispielsweise bei unbenutzten Zyklen die CPU zu bremsen und dadurch Strom zu sparen. Es dauert allerdings etwa 150 Mikrosekunden, bis der neue Wert erreicht ist und in dieser Zeit kann der Prozessor nicht auf externe Ereignisse reagieren.

Das Speicherlayout

Das Speicherlayout des virtuellen Speichers wird in Tabelle D.3 dargestellt. Wesentlich bei der Realisierung waren die Bereich mit dem nicht flüchtigen Speicher (0x00000000

- 0x07FFFFFF) und Controller- und Ein-/Ausgabebereiche zur Ansteuerung der Geräte durch Speicherzugriffe (Memory Mapped I/O).

Ein-/Ausgabe (I/O)

In dem Adressbereich von 0x80000000 bis 0xBFFFFFFF werden I/O-Port auf den Speicher abgebildet. Folgende Hardwarekomponenten können dadurch mittels einfachem Speicherzugriff bedient werden:

- SDRAM
- Flash Speicher
- PCMCIA Ports und aktive Karten
- Serielle Schnittstellen
- LCD-Controller
- GPIO (General Purpose I/O-Controller)
- Power Management Register

Es existieren drei General Purpose I/O-Controller (GPIO). Der im Prozesor integrierte GPIO, der erweiterte E-GPIO und der Mikrocontroller GPIO-P.

Der integrierte GPIO

Der integrierte GPIO kann folgende Komponenten abfragen und steuern:

- Die Tasten des Gerätes
- LCD-Display
- PCMCIA-Controller
- Audio-Codec-Controller
- UART für die RS-232C Schnittstelle

Der erweiterte GPIO

Der erweiterte GPIO kann diese Komponenten abfragen und steuern:

- Programmiererlaubnis für das FlashROM
- Reset-Signale für alle Komponenten
- Stromversorgungen für einzelne Komponenten (LCD, Audio, RS-232C, ...)

Die seriellen Schnittstellen

RS-232C

Die integrierte RS-232C Schnittstelle unterstützt folgende Signale: RXD, TXD, RTS, CTS, DCD, DTR und DSR. Das Interface wird am kleineren Stecker herausgeführt und belegt den seriellen Port 3.

IrDA

Der iPAQ arbeitet momentan nach dem IrDA 1.2 Standard und kann sowohl den Fast (FIR) als auch den Slow Ir-Modus (SIR). Der IrDA Kanal belegt für Senden als auch Empfangen den seriellen Port 2 am Prozessor. Der iPAQ unterstützt aber nicht den Consumer IR-Standard. Der IrDA Port ist für eine Übertragungsentfernung von bis zu 30 cm ausgelegt.

USB

Der iPAQ besitzt standardmäßig die Hardware eines USB-Clients. Das zugehörige Kabel wird jedoch nicht mitgeliefert. Die maximale Übertragungsgeschwindigkeit beträgt 12 MBit/s und der USB-Client belegt den seriellen Port 0. Der USB-Port wird am kleineren externen Stecker zusätzlich herausgeführt.

Das integrierte Audiosystem

Das Audiosystem auf Basis des Philips UDA1341 belegt den seriellen Port 4. Die Sample-Raten 8, 11.025, 22.05, 44.1 und 48 kHz werden unterstützt. Als Eingang kann nur das

integrierte Mikrophon verwendet werden. Es ist nicht möglich ein anderes Mikrophon anzuschliessen. Es existieren drei Ausgänge: ein integrierter Mono-Lautsprecher, ein Klinkenstrecker für einen Kopfhörer und das Stereo-Audiosignal am Expansion-Port.

Das Touchpanel Interface

Keine Informationen über dessen Auflösungsvermögen.

Die Schnittstelle für die Tastatur

Es existieren ein 5-Knopf Joystick, 4 Tasten für Anwendungen, ein Aufnahmeknopf, ein Ein-/Ausschaltknopf, ein Reset-Taster und ein Schalter für das Trennen der Batterie (dadurch kann man den RAM-Speicher komplett löschen).

Diese Tasten können nur dann korrekt abgefragt werden, wenn sie einzeln gedrückt werden, da ein Widerstandsnetzwerk zur Abfrage verwendet wird und das Drücken jeder Taste eine bestimmte Spannung erzeugt.

Der LCD-Controller

Der LCD-Controller bekommt die Daten über einen Framebuffer, der über DMA angesprochen wird. Die Auflösung beträgt 320x240 Bildpunkte auf einem TFT-Display mit 70 Hz Bildwiederholrate. Dadurch ergibt sich eine Videobandbreite von:

$((320x240x16)/8) * 70 = 10.752MHz$

Der TFT-Controller verwendet nur die obersten 4 Bit der Farbinformation, für alle drei Farben ergibt sich also eine maximale Farbtiefe von 4096 Farben.

Die Ansteuerung der LED

Die LED ist zweifarbig (amber und grün) und kann von der Software im priviligierten Modus durch I/O-Befehle angesteuert werden. Beide LEDs können gleichzeitig betrieben werden.

Der Cradle-Anschluss

Der 15-polige Anschluss für die mitgelieferte Dockingstation führt die RS-232C, die USB Schnittstelle und den Ladeanschluss der externen Stromquelle nach aussen.

Der Expansions-Port

Der 100-polige Anschluss für Erweiterungen führt alle notwendige Datenleitungen der internen Komponenten heraus. Allerdings wird nicht der Anschluss für das Mikrophon und auch nicht der komplette Prozessor-Bus herausgeleitet. Es ist aber möglich, gleichzeitig beispielsweise bis zu zwei PCMCIA/CompactFlash Adapter zu verwenden.

Speicherbereich	Größe	Benutzt für
0x00000000 - 0x07FFFFFF	128 MByte	Static Bank 0 - iPAQ Onboard FlashROM
0x08000000 - 0x0FFFFFFF	128 MByte	Static Bank 1 - Reserviert
0x10000000 - 0x17FFFFFF	128 MByte	Static Bank 2 - Expansion Pack
0x18000000 - 0x1FFFFFFF	128 MByte	Static Bank 3 - Expansion Pack
0x20000000 - 0x2FFFFFFF	256 MByte	PCMCIA Socket 0 - Expansion Pack PCMCIA/CF Slot 0
0x30000000 - 0x3FFFFFFF	256 MByte	PCMCIA Socket 1 - Expansion Pack PCMCIA/CF Slot 1
0x40000000 - 0x47FFFFFF	128 MByte	Static Bank 4 - Statischer Speicherbereich und MMAP I/O
0x48000000 - 0x4FFFFFFF	128 MByte	Static Bank 5 - Erweitertes I/O und PPSH
0x50000000 - 0x7FFFFFFF	768 MByte	Reserviert - Nicht benutzt
0x80000000 - 0x8FFFFFFF	256 MByte	Peripheral Control Module Register
0x90000000 - 0x9FFFFFFF	256 MByte	System Control Module Register
0xA0000000 - 0xAFFFFFFF	256 MByte	Memory und Expansion Pack Register
0xB0000000 - 0xBFFFFFFF	256 MByte	LCD und DMA Register
0xC0000000 - 0xC7FFFFFF	128 MByte	RAM Bank 0 - iPAQ nutzt nur diesen Speicherbereich
0xC8000000 - 0xCFFFFFFF	128 MByte	RAM Bank 1 - Nicht benutzt
0xD0000000 - 0xD7FFFFFF	128 MByte	RAM Bank 2 - Nicht benutzt
0xD8000000 - 0xDFFFFFFF	128 MByte	RAM Bank 3 - Nicht benutzt
0xE0000000 - 0xE7FFFFFF	128 MByte	Bank immer mit Nullen gefüllt, Cache flush replacement
0xE8000000 - 0xFFFFFFFF	384 MByte	Nicht benutzt

Tabelle D.3: Speicheraufteilung

Definitionsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[AGS83] Ames, S., M. Gasser und R. Schell: *Security Kernel Design and Implementation*. Computer, 16(7):14–22, 1983.

[Ame80] Ames, Stan: *Panel Discussion: Verification of Secure Software Systems*. In: *IEEE Symposium on Security and Privacy*, Seiten 157–166, 1980.

[Amo94] Amoroso, Edward: *Fundamentals of Computer Security Technology*. Prentice Hall, April 1994.

[And72] Anderson, J. P.: *Computer Security Technology Planning Study*. Technischer Bericht ESD-TR-73-51, ESD/AFSC, Hanscom AFB, Bedford, MA, 1972.

[And75] Andrews, Gregory R.: *Partitions and principles for secure operating systems*. In: *Proceedings of the 1975 annual conference*, Seiten 177–180. ACM Press, 1975.

[And01] Anderson, Ross: *Security Engineering*. Wiley Computer Publishing. John Wiley & Sons, 2001.

[Ano78] Anon: *KSOS Executive Summary*, 1978. Ford Aerospace and Communications Corp.

[AS00] Ahn, Gail-Joon und Ravi Sandhu: *Role-based authorization constraints specification*. ACM Transactions on Information and System Security (TISSEC), 3(4):207–226, 2000.

[BAN90] Burrows, Michael, Martin Abadi und Roger Needham: *A logic of authentication*. ACM Transactions on Computer Systems (TOCS), 8(1):18–36, 1990.

[BBEG02] Baumgarten, U., A. Buchmann, C. Eckert und H. Görl: *Ein Vergleich moderner Linux-Sicherheitsmechanismen*. In: *eBusiness Processes 2002 (EBP 2002 – Schwerpunkt IT-Sicherheit)*, Rot, September 2002.

[BCD72] Bensoussan, A., C. T. Clingen und R. C. Daley: *The Multics virtual memory: concepts and design.* Communications of the ACM, 15(5):308–318, 1972.

[BDGL04] Buchmann, A., F. Dötzer, H. Görl und S. Lachmund: *Lösen TCPA und Palladium die Sicherheitsprobleme von heute?* In: *D-A-CH Security – Bestandsaufnahme und Perspektiven*, Basel, März 2004.

[BEG00] Baumgarten, Uwe, Claudia Eckert und Harald Görl: *Trust and Confidence in Open Systems: Does Security harmonize with Mobility?* In: *9th ACM SIGOPS European Workshop „Beyond the PC: New Challenges for the Operating System“*. ACM, September 2000.

[Bel89] Bellovin, S. M.: *Security problems in the TCP/IP protocol suite.* Computer Communications Review, 19:2:32–48,http://www.research.att.com/∼smb/papers/ipext.pdf, 1989.

[BG03] Blass, Andreas und Yuri Gurevich: *Abstract state machines capture parallel algorithms.* ACM Trans. Comput. Logic, 4(4):578–651, 2003.

[BGL03a] Buchmann, A., H. Görl und S. Lachmund: *Placement of Cryptographic Mechanisms in Operating Systems.* In: *The 2003 International Conference on Security and Management (SAM'03)*, Las Vegas, Juni 2003. submitted paper.

[BGL03b] Buchmann, A., H. Görl und S. Lachmund: *Sind mobile Endgeräte als Personal Trusted Devices geeignet?* In: *D-A-CH Security – Bestandsaufnahme und Perspektiven*, Erfurt, März 2003.

[BGM00] Bernaschi, Massimo, Emanuele Gabrielli und Luigi V. Mancini: *Operating system enhancements to prevent the misuse of system calls.* In: *Proceedings of the 7th ACM conference on Computer and communications security*, Seiten 174–183. ACM Press, 2000.

[Bis99] Bishop, M.: *Vulnerabilities Analysis.* In: *Proceedings of the 2nd International Workshop on Recent Advances in Intrusion Detection (RAID'99)*, 1999.

[Bis03] Bishop, Matt: *Computer Security, Art and Science.* Addison Wesley Longman, Reading, Massachusetts, 2003.

[BL73] Bell, D. und L. LaPadula: *Secure Computer Systems: Mathematical Foundations and Model.* Technischer Bericht M74-244, MITRE Corporation, Bedford, MA, Nov 1973.

[BL75] Bell, David E. und Leonard LaPadula: *Secure Computer Systems: Unified Exposition and Multics Interpretation.* Technischer Bericht ESD-TR-75-306, ESD/AFSC, Hanscom AFB, Bedford, MA, 1975.

[Bla93] Blaze, Matt: *A cryptographic file system for UNIX.* In: *Proceedings of the 1st ACM conference on Computer and communications security*, Seiten 9–16. ACM Press, 1993.

[BN89] Brewer, David F. C. und Michael J. Nash: *The Chinese Wall Security Policy.* In: *1989 Symposium on Security and Privacy*, Seiten 206–214. IEEE Computer Society Press, 1989.

[Bro98] Broy, Manfred: *Informatik - Eine grundlegende Einführung.* Springer-Verlag, Berlin, zweite Auflage, 1998.

[BS00] Börger, E. und J. Schmid: *Composition and Submachine Concepts for Sequential ASMs*, 2000.

[BS03a] Börger, E. und R. F. Stärk: *Abstract State Machines—A Method for High-Level System Design and Analysis.* Springer-Verlag, 2003.

[BS03b] Broy, Manfred und Ralf Steinbrüggen: *Modellbildung in der Informatik.* Springer-Verlag, 2003.

[CD95] Cheriton, David R. und Kenneth J. Duda: *A caching model of operating system kernel functionality.* ACM SIGOPS Operating Systems Review, 29(1):83–86, 1995.

[CGHM81] Cheheyl, Maureen Harris, Morrie Gasser, George A. Huff und Jonathan K. Millen: *Verifying Security.* ACM Computing Surveys (CSUR), 13(3):279–339, 1981.

[CGHS00] Compton, K., Y. Gurevich, J. Huggins und W. Shen: *An Automatic Verification Tool for UML.* Technischer Bericht CSE-TR-423-00, CSE, 2000.

[Cho92] Chokhani, Santosh: *Trusted products evaluation.* Communications of the ACM, 35(7):64–76, 1992.

[CM03] Cervesato, I. und C. Meadows: *A fault-tree representation of NPATRL security requirements*, 2003.

[Cor91] Corbato, Fernando J.: *On building systems that will fail.* Communications of the ACM, 34(9):72–81, 1991.

[CW87] Clark, D. D. und D. R. Wilson: *A Comparison of Commercial and Military Computer Security Policies.* In: *Proceedings of the Symposium on Security and Privacy 1987*, Seiten 184–193. IEEE Press, 1987.

[DD77] Denning, Dorothy E. und Peter J. Denning: *Certification of programs for secure information flow.* Communications of the ACM, 20(7):504–513, 1977.

[Den76] Denning, Dorothy E.: *A lattice model of secure information flow.* Communications of the ACM, 19(5):236–243, 1976.

[Den86] Denning, D.: *An Intrusion Detection Model.* In: *Proceedings of the Symposium on Security and Privacy 1986*, Seiten 118–131. IEEE Press, 1986.

[Den93] Denning, Dorothy E.: *A new paradigm for trusted systems.* In: *Proceedings on the 1992-1993 workshop on New security paradigms*, Seiten 36–41. ACM Press, 1993.

[Dis81] Discepolo, Anne-Marie G.: *Towards a practical specification language.* In: *Proceedings of the ACM '81 conference*, Seiten 144–153. ACM Press, 1981.

[DLS+88] Denning, Dorothy E., Teresa F. Lunt, Roger R. Schell, William R. Shockley und Mark Heckman: *The SeaView Security Model.* In: *1988 Symposium on Security and Privacy*, Seiten 218–233. IEEE Computer Society Press, 1988.

[Dob86] Dobson, J.E. ; Randell, B.: *Building reliable secure computing systems out of unreliable insecure components.* In: *1986 Symposium on Security and Privacy*, Seiten 187–93. IEEE Computer Society Press, 1986.

[Eck02] Eckert, Claudia: *IT-Sicherheit - Konzepte, Verfahren und Protokolle.* Oldenbourg-Verlag, München, Dezember 2002.

[Eck03] Eckert, C.: *Mobil, aber sicher!* In: *Total vernetzt.* Springer, 2003. to appear.

[EEG00] Eckert, Claudia, Florian Erhard und Johannes Geiger: *GSFS – A New Group-Aware Cryptographic File System.* In: *Proceedings of the World Computer Congress*, Beijing, August 2000. SEC2000.

[Els03] Elser, Dennis: *A decompilation of the Lovesan/MSBLAST Worm.* http://www.astalavista.com/code/assembly/A-decompilation-of-the-Lovesan-MSBLAST-Worm.txt, abgerufen August 2003.

[FK89] Feldmeier, David C. und Philip R. Karn: *UNIX Password Security - Ten Years Later.* In: *CRYPTO*, Seiten 44–63, 1989.

[FN79] Feiertag, R. und P. Neumann: *The Foundations of a Provably Secure Operating System (PSOS).* In: *Proceedings of the National Computer Conference*, Seiten 329–334, 1979.

[Fra79] Frantz, Bill: *GNOSIS - A Prototype Operating System for the 1990's.* In: *Proceedings of SHARE 52 I*, Seiten 3–17. SHARE Inc, 1979.

[Für03] Fürst, Armin: *Untersuchung effizienter, sicherer Erweiterungskonzepte für Betriebssysteme.* Diplomarbeit, TU München, 2003.

[GGKL89] Gasser, M., A. Goldstein, C. Kaufman und B. Lampson: *The Digital Distributed System Security Architecture*. In: *Proc. 12th NIST-NCSC National Computer Security Conference*, Seiten 305–319, 1989.

[Gli83] Gligor, Virgil D.: *A Note on the Denial-of-Service Problem*. In: *Proceedings of the Symposium on Security and Privacy 1983*, Seiten 139–149. IEEE Press, 1983.

[GM82] Goguen, J. A. und J. Meseguer: *Security Policies and Security Models*. In: *1982 Symposium on Security and Privacy*, Seiten 11–20. IEEE Computer Society Press, 1982.

[GM84] Grampp, F. T. und R. H. Morris: *UNIX Operating System Security*. AT&T Bell Laboratories Technical Journal, 63(8):1649–1672, 1984.

[Gol99] Gollmann, Dieter: *Computer Security*. John Wiley & Sons, West Sussex, England, August 1999.

[Gör01] Görl, Harald: *Realisierung einer überwachten Systemdiensteschnittstelle*. In: *Fachgruppentreffen für Betriebssysteme*, TU Ilmenau, November 2001. GI.

[Gör03] Görl, Harald: *Entwicklung sicherer Software*. In: *Sicherheit mit Opensource*, Darmstadt, März 2003. CAST-Forum.

[Har85] Hardy, Norman: *KeyKOS architecture*. SIGOPS Oper. Syst. Rev., 19(4):8–25, 1985.

[Her00] Herbert Valerio Riedel: *The GNU/Linux CryptoAPI site*. http://www.kerneli.org/, 2000.

[HHLS97] Härtig, Hermann, Michael Hohmuth, Jochen Liedtke und Sebastian Schönberg: *The performance of µ-kernel-based systems*. In: *Proceedings of the sixteenth ACM symposium on Operating systems principles*, Seiten 66–77. ACM Press, 1997.

[HHP03] Hussain, Alefiya, John Heidemann und Christos Papadopoulos: *A framework for classifying denial of service attacks*. In: *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, Seiten 99–110. ACM Press, 2003.

[HL03] Howard, Michael und David LeBlanc: *Writing Secure Code*. Microsoft Press, Redmond, WA, USA, 2nd Auflage, 2003.

[Hoh98] Hohl, Fritz: *A Model of Attacks of Malicious Hosts Against Mobile Agents*. Technischer Bericht, Universität Stuttgart, Stuttgart, Germany, 1998.

[HR98] Heintze, Nevin und Jon G. Riecke: *The SLam calculus: programming with secrecy and integrity.* In: *Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, Seiten 365–377. ACM Press, 1998.

[HRU76] Harrison, Michael A., Walter L. Ruzzo und Jeffrey D. Ullman: *Protection in operating systems.* Communications of the ACM, 19(8):461–471, 1976.

[HW] Huggins, James K. und Charles Wallace: *An Abstract State Machine Primer.*

[Irv97] Irvine, Cynthia E.: *Security in innovative new operating systems.* In: *1997 Symposium on Security and Privacy*, Seiten 202–203. IEEE Computer Society Press, 1997.

[JJ02] Jürjens Jan: *UMLsec: Extending UML for Secure Systems Development.* In: *LNCS*, Dresden, September 2002. Springer.

[JK90] Jajodia, Sushil und Boris Kogan: *Integrating an Object-Oriented Data Model with Multilevel Security.* In: *1990 Symposium on Security and Privacy*, Seiten 76–85. IEEE Computer Society Press, 1990.

[JSW02] Jha, Somesh, Oleg Sheyner und Jeannette M. Wing: *Minimization and Reliability Analyses of Attack Graphs.* Technischer Bericht CMU-CS-02-109, CMU, February 2002.

[JV02] John Viega, Gary McGraw: *Building Secure Software.* Addison-Wesley, 2002.

[KCE92] Koldinger, Eric J., Jeffrey S. Chase und Susan J. Eggers: *Architecture support for single address space operating systems.* In: *Proceedings of the fifth international conference on Architectural support for programming languages and operating systems*, Seiten 175–186. ACM Press, 1992.

[Knu74] Knuth, Donald E.: *Computer programming as an art.* Communications of the ACM, 17(12):667–673, 1974.

[Knu97] Knuth, Donald E.: *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms.* Addison-Wesley Longman Publishing Co., Inc., 1997.

[KZB+90] Karger, Paul A., Mary Ellen Zurko, Douglas W. Bonin, Andrew H. Mason und Clifford E. Kahn: *A VMM Securtiy Kernel for the VAX Architecture.* In: *1990 Symposium on Security and Privacy*, Seiten 2–19. IEEE Computer Society Press, 1990.

[LABW92] Lampson, Butler, Martin Abadi, Michael Burrows und Edward Wobber: *Authentication in distributed systems: theory and practice.* ACM Transactions on Computer Systems (TOCS), 10(4):265–310, 1992.

[Lam71] Lampson, Butler: *Protection.* In: *Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems*, Seiten 437–443, Princeton University, 1971.

[Lam73] Lampson, Butler W.: *A note on the confinement problem.* Communications of the ACM, 16(10):613–615, 1973.

[Lan81] Landwehr, Carl E.: *Formal Models for Computer Security.* ACM Computing Surveys (CSUR), 13(3):247–278, 1981.

[Lan83] Landwehr, Carl E.: *The Best Available Technologies for Computer Security.* Computer, 16(7):86–100, 1983.

[LBMC94] Landwehr, Carl E., Alan R. Bull, John P. McDermott und William S. Choi: *A taxonomy of computer program security flaws.* ACM Comput. Surv., 26(3):211–254, 1994.

[Lee88] Lee, Theodore M.P.: *Using Mandatory Integrity to enforce „Commercial" Security.* In: *1988 Symposium on Security and Privacy*, Seiten 140–146. IEEE Computer Society Press, 1988.

[Lev95] Leveson, N. G.: *Safeware: system safety and computers.* Addison-Wesley, 1995. ISBN 0201119722.

[Lie93] Liedtke, Jochen: *Improving IPC by kernel design.* In: *Proceedings of the fourteenth ACM symposium on Operating systems principles*, Seiten 175–188. ACM Press, 1993.

[LS01a] Loscocco, P. und S. Smalley: *Integrating Flexible Support for Security Policies into the Linux Operating System.* In: *2001 USENIX Annual Technical Conference (FREENIX '01)*, 2001.

[LS01b] Loscocco, P. und S. Smalley: *Meeting Critical Security Objectives with Security-Enhanced Linux.* In: *2001 Ottawa Linux Symposium*, 2001.

[Luc99] *LucaOS-Projekthomepage.* http://www13.in.tum.de/SPP/, 1999.

[Man02] Mantel, Heiko: *On the Composition of Secure Systems.* In: *Proceedings of the IEEE Symposium on Security and Privacy*, Seiten 88–101, Oakland, CA, USA, May 12–15 2002. IEEE Computer Society.

[McC87] McCullough, Daryl: *Specifications for Multi-Level Security and a Hook-Up Property.* In: *1987 Symposium on Security and Privacy*, Seiten 161–166. IEEE Computer Society Press, 1987.

[McL85] McLean, John: *A Comment on the 'Basic Security Theorem' of Bell and LaPadula*. Information Processing Letters, 20(2):67–70, 1985.

[McL87] McLean, John: *Reasoning about Security Models*. In: *1987 Symposium on Security and Privacy*, Seiten 123–131. IEEE Computer Society Press, 1987.

[McL90a] McLean, John: *Security Models and Information Flow*. In: *IEEE Symposium on Security and Privacy*, Seiten 180–189, 1990.

[McL90b] McLean, John: *The Specification and Modeling of Computer Security*. Computer, 23(1):9–16, 1990.

[McL94] McLean, John: *Security Models*. In: Marciniak, J. (Herausgeber): *Encyclopedia of Software Engineering*. John Wiley & Sons, 1994.

[MEL01] Moore, A., R. Ellison und R. Linger: *Attack modeling for information security and survivability*, 2001.

[Mer78] Merkle, Ralph C.: *Secure communications over insecure channels*. Communications of the ACM, 21(4):294–299, 1978.

[MFS90] Miller, Barton P., Louis Fredriksen und Bryan So: *An empirical study of the reliability of UNIX utilities*. Communications of the ACM, 33(12):32–44, 1990.

[Mil76] Millen, Jonathan K.: *Security Kernel validation in practice*. Communications of the ACM, 19(5):243–250, 1976.

[ML00] Myers, Andrew C. und Barbara Liskov: *Protecting privacy using the decentralized label model*. ACM Transactions on Software Engineering and Methodology (TOSEM), 9(4):410–442, 2000.

[MM00] Mann, Scott und Ellen L. Mitchell: *Linux System Security: The Administrator's Guide to Open Source Security Tools*. Prentice Hall, Upper Saddle River, NJ, 2000.

[Moo90] Moore, A. P.: *The Specification and Verified Decomposition of System Requirements Using CSP*. IEEE Transactions on Software Engineering, 16(9):932–948, 1990.

[MR92] McIlroy, M. D. und J. A. Reeds: *Multilevel security in the UNIX tradition*. Software–Practice & Experience, 22(8):673–694, 1992.

[MRR98] Malkhi, Dahlia, Michael K. Reiter und Aviel D. Rubin: *Secure Execution of Java Applets using a remote Playground*. In: *1998 Symposium on Security and Privacy*, Seiten 40–51. IEEE Computer Society Press, 1998.

[MSB03] *MSBlast complete recode / analysis.* http://www.security-focus.com/archive/82/ 333313/2003-08-09/2003-08-15/0, abgerufen August 2003.

[MT79] Morris, Robert und Ken Thompson: *Password security: a case history.* Communications of the ACM, 22(11):594–597, 1979.

[MvOV96] Menezes, Alfred, Paul van Oorschot und Scott Vanstone: *Handbook of Applied Cryptography.* CRC Press, 1996.

[Nec97] Necula, George C.: *Proof-Carrying Code.* In: *Conference Record of POPL '97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Seiten 106–119, Paris, France, jan 1997.

[Neu73] Neuhold, E. J.: *Towards the formal description of operating systems.* In: *Proceeding of ACM SIGPLAN - SIGOPS interface meeting on Programming languages - operating systems*, Seiten 123–126. ACM Press, 1973.

[NF03] Neumann, P. G. und R. J. Feiertag: *PSOS Revisted.* In: *Proceedings of the 19th Annual Computer Security Applications Conference (ASAC 2003)*, Seiten 208–216, December 2003. Available under http://www.acsac.org/.

[NL96] Necula, George C. und Peter Lee: *Safe Kernel Extensions Without Run-Time Checking.* In: USENIX (Herausgeber): *2nd Symposium on Operating Systems Design and Implementation (OSDI '96), October 28–31, 1996. Seattle, WA*, Seiten 229–243, Berkeley, CA, USA, 1996. USENIX.

[NS78] Needham, Roger M. und Michael D. Schroeder: *Using encryption for authentication in large networks of computers.* Communications of the ACM, 21(12):993–999, 1978.

[NS87] Needham, R. M. und M. D. Schroeder: *Authentication revisited.* ACM SIGOPS Operating Systems Review, 21(1):7–7, 1987.

[NW77] Needham, R. M. und R. D. H. Walker: *The Cambridge CAP computer and its protection system.* In: *Proceedings of the sixth symposium on Operating systems principles*, Seiten 1–10. ACM Press, 1977.

[OAS03] OASIS, Organization for the Advancement of Structured Information Standards: *Application Vulnerability Description Language.* http://www.avdl.org/, 2003.

[Oct04] *Operationally Critical Threat, Asset and Vulnerability Evaluation.* http://www.cert.org/octave/, abgerufen April 2004.

[OSM00] Osborn, Sylvia, Ravi Sandhu und Qamar Munawer: *Configuring role-based access control to enforce mandatory and discretionary access control policies.* ACM Transactions on Information and System Security (TISSEC), 3(2):85–106, 2000.

[oST99] Standards, National Institute of und Technology: *Common Criteria Version 2.1, Common Language to Express Common Needs*, August 1999.

[PF78] Popek, Gerald J. und David A. Farber: *A model for verification of data security in operating systems.* Communications of the ACM, 21(9):737–749, 1978.

[Pfl97] Pfleeger, C. P.: *Security in Computing.* Prentice Hall, Upper Saddle River, NJ, 2nd Auflage, 1997.

[PS98] Phillips, Cynthia und Laura Painton Swiler: *A graph-based system for network-vulnerability analysis.* In: *Proceedings of the 1998 workshop on New security paradigms*, Seiten 71–79. ACM Press, 1998.

[Ran95] Rannenberg, Kai: *Evaluationskriterien zur IT-Sicherheit - Entwicklungen und Perspektiven in der Normung und außerhalb.* In: *GI-Fachtagung*, Seiten 45–68, Verläßliche IT-Systeme, April 1995. GI.

[RE02] Rankl, Wolfgang und Wolfgang Effing: *Handbuch der Chipkarten.* Hanser Verlag, München, 2002.

[RJB98] Rumbaugh, J., I. Jacobson und G. Booch: *The Unified Modeling Language Reference Manual.* Addison Wesley Longman, Reading, Massachusetts, December 1998.

[RR83] Rushby, John und Brian Randell: *A Distributed Secure System.* Computer, 16(7):55–67, 1983.

[RS01] Ryan, Peter und Steve Schneider: *The Modelling and Analysis of Security Protocols: the CSP Approach.* Addison Wesley Longman, Reading, Massachusetts, 2001.

[RSA78] Rivest, R. L., A. Shamir und L. Adleman: *A method for obtaining digital signatures and public-key cryptosystems.* Communications of the ACM, 21(2):120–126, 1978.

[Rus81] Rushby, J. M.: *Design and verification of secure systems.* In: *Proceedings of the eighth symposium on Operating systems principles*, Seiten 12–21. ACM Press, 1981.

[Sal74] Saltzer, Jerome H.: *Protection and the control of information sharing in multics.* Communications of the ACM, 17(7):388–402, 1974.

[San92] Sandhu, Ravi: *Lattice-Based Enforcement of Chinese Walls*, 1992.

[San98] Sandhu, Ravi: *Role activation hierarchies*. In: *Proceedings of the third ACM workshop on Role-based access control*, Seiten 33–40. ACM Press, 1998.

[Sch83] Schell, Roger R.: *A Security Kernel for a Multiprocessor Microcomputer*. Computer, 16(7):47–53, 1983.

[Sch97] Schneier, Bruce: *Angewandte Kryptographie*. Addison Wesley (Deutschland) GmbH, Bonn, 1997.

[Sch00] Schneier, Bruce: *Secrets and Lies*. John Wiley & Sons, New York, 2000.

[Sch03] Schnabel, Ingo: *COMA — Implementierung eines Tools zur Analyse von Quelltexten*. Technischer Bericht, TU München, 2003.

[SEL] *Security Enhanced Linux*. http://www.nsa.gov/selinux/.

[Sha79] Shamir, Adi: *How to share a secret*. Communications of the ACM, 22(11):612–613, 1979.

[Shi00] Shirey, R.: *RFC 2828: Internet Security Glossary*, Mai 2000.

[Sil83] Silverman, Jonathan M.: *Reflections on the verification of the security of an operating system kernel*. In: *Proceedings of the ninth ACM symposium on Operating systems principles*, Seiten 143–154. ACM Press, 1983.

[SJW02] Sheyner, Oleg, Somesh Jha und Jeannette M. Wing: *Automated Generation and Analysis of Attack Graphs*. In: *Proceedings of the IEEE Symposium on Security and Privacy*, May 2002.

[Smi93] Smith, H. Jeff: *Privacy policies and practices: inside the organizational maze*. Communications of the ACM, 36(12):104–122, 1993.

[Smi98] Smith, Richard E.: *Internet-Kryptographie*. Addison Wesley Longman Verlag GmbH, Bonn, 1998.

[Spa89a] Spafford, E. H.: *Crisis and aftermath*. Communications of the ACM, 32(6):678–687, 1989.

[Spa89b] Spafford, E. H.: *The Internet Worm Incident*. In: Ghezzi, C. und J. A. McDermid (Herausgeber): *ESEC'89 2nd European Software Engineering Conference*, University of Warwick, Coventry, United Kingdom, 1989. Springer.

[SPL95] Sibert, Olin, Phillip A. Porras und Robert Lindell: *The Intel 80x86 Processor Architecture: Pitfalls for Secure Systems*. In: *1995 Symposium on Security and Privacy*, Seiten 211–222. IEEE Computer Society Press, 1995.

[SS72] Schroeder, Michael D. und Jerome H. Saltzer: *A hardware architecture for implementing protection rings.* Communications of the ACM, 15(3):157–170, 1972.

[SS03] Sadeghi, Ahmad-Reza und Christian Stüble: *Bridging the Gap between TCPA/Palladium and Personal Security.* Technischer Bericht, Saarland University, 2003.

[SSd75] Saltzer, Jerome H. und Michael D. Schroeder: *The Protection of Information in Computer Systems.* Proceedings of the IEEE, 63(9):1278–1308, September 1975.

[SSF99] Shapiro, Jonathan S., Jonathan M. Smith und David J. Farber: *EROS: a fast capability system.* In: *Proceedings of the seventeenth ACM symposium on Operating systems principles*, Seiten 170–185. ACM Press, 1999.

[Ste91] Sterne, Daniel F.: *On the Buzzword „Security Policy“.* In: *1991 Symposium on Security and Privacy*, Seiten 219–230. IEEE Computer Society Press, 1991.

[Sto88] Stoll, Clifford: *Stalking the wily hacker.* Communications of the ACM, 31(5):484–497, 1988.

[Sto96] Storey, Neil: *Safety-Critical Computer Systems.* Addison-Wesley, 1996.

[SUN] SUN Microsystems: *Java Cryptography Extension (JCE).* http://java.sun.com/products/jce/.

[Sun96] Sundaram, Aurobindo: *An introduction to intrusion detection.* Crossroads, 2(4):3–7, 1996.

[Tan02] Tanenbaum, Andrew S.: *Moderne Betriebssysteme.* Pearson Studium, München, zweite Auflage, 2002.

[Tay84] Taylor, T.: *Comparison Paper Between the Bell and LaPadula Model and the SRI Model.* In: *Proceedings of the IEEE Symposium on Security and Privacy*, Seiten 195–292, CA, USA, Apr 1984. IEEE Computer Society.

[Tho84] Thompson, Ken: *Reflections on trusting trust.* Communications of the ACM, 27(8):761–763, 1984.

[TO03] Ochs Tilman: *Anpassung eines verschlüsselnden Dateisystems an verschiedene Versionen des Linux-Kerns.* Technischer Bericht, TU München, 2003.

[Tox01] Toxen, Bob: *Linux System Security: The Administrator's Guide to Open Source Security Tools.* Prentice Hall, Upper Saddle River, NJ, 2001.

[TW89] Terry, Phil und Simon Wiseman: *A 'new' security model*. In: *Proceedings of the IEEE Symposium on Security and Privacy*, Seiten 215–228, Oakland, CA, USA, May 1989. IEEE Computer Society.

[VBC01] Viega, John, J.T. Bloch und Pravir Chandra: *Applying Aspect-Oriented Programming to Security*. Cutter IT Journal, February 2001.

[VBKM02] Viega, John, J. T. Bloch, Tadayoshi Kohno und Gary McGraw: *Token-based scanning of source code for security problems*. ACM Transactions on Information and System Security, 5(3):238–261, August 2002.

[VGRH81] Vesely, W. E., F. F. Goldberg, N. H. Roberts und D. F. Haasl: *Fault Tree Handbook*. U. S. Nuclear Regulatory Commission, NUREG-0492, Washington DC, January 1981.

[Wei92] Weissman, Clark: *BLACKER: Security for the DDN, Examples of A1 Security Engineering Trades*. In: *1992 Symposium on Security and Privacy*, Seiten 286–292. IEEE Computer Society Press, 1992.

[Win98] Wing, J.: *A symbiotic relationship between formal methods and security*, 1998.

[WKP80] Walker, Bruce J., Richard A. Kemmerer und Gerald J. Popek: *Specification and verification of the UCLA Unix security kernel*. Communications of the ACM, 23(2):118–131, 1980.

[WLAG93] Wahbe, Robert, Steven Lucco, Thomas E. Anderson und Susan L. Graham: *Efficient software-based fault isolation*. In: *Proceedings of the fourteenth ACM symposium on Operating systems principles*, Seiten 203–216. ACM Press, 1993.

[WSO+75] Walter, K. G., S. I. Schaen, W. F. Ogden, W. C. Rounds, D. G. Shumway, D. D. Schaeffer, K. J. Biba, F. T. Bradshaw, S. R. Ames und J. M. Gilligan: *Structured specification of a Security Kernel*. In: *Proceedings of the international conference on Reliable software*, Seiten 285–293, 1975.

[ZH02] Zakrezewski, Miroslaw und Ibrahim Haddad: *Kernel korner: Linux distributed security module*. Linux Journal, 2002(102):6, 2002.

[ZZNM02] Zdancewic, Steve, Lantian Zheng, Nathaniel Nystrom und Andrew C. Myers: *Secure program partitioning*. ACM Transactions on Computer Systems (TOCS), 20(3):283–328, 2002.