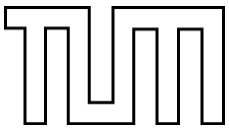
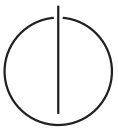


TECHNISCHEN UNIVERSITÄT MÜNCHEN
Fakultät für Informatik
Lehrstuhl für Angewandte Softwaretechnik



Gefahrenerkennung in Konfigurationen verteilter Systeme

Martin Otto Werner Wagner

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. Michael Bader
Prüfer der Dissertation:	1.	Univ.-Prof. Bernd Brügge, Ph. D.
	2.	Univ.-Prof. Dr.-Ing. Georg Carle

Die Dissertation wurde am 4. April 2016 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 19. August 2016 angenommen.

Danksagung

Ich möchte mich bei meinem Betreuer Prof. Bernd Brügge, Ph.D., dafür bedanken, dass er mir die Möglichkeit gab, an seinem Lehrstuhl zu forschen und zu promovieren. Die Gespräche mit ihm waren immer inspirierend und führten zu neuen Erkenntnissen, die ich wahrscheinlich sonst nicht gehabt hätte. Seine Herzlichkeit und Offenheit machte das Arbeiten am Lehrstuhl zu einer angenehmen Erfahrung. Seine engen Kontakte zur Industrie ermöglichten mir, weitere Erfahrungen mit realen Projekten zu sammeln; dafür bin ich ihm sehr verbunden.

Ebenfalls danke ich auch den Mitarbeitern des Lehrstuhls für ihre freundschaftliche Unterstützung. Insbesondere schätzte ich die technische Unterstützung von Monika Markl und Helma Schneider und die guten Ratschläge von Stephan Krusche.

Kurzfassung

IT-Infrastrukturen sind ständig existierenden oder neuen, durch Veränderungen hervorgerufenen Gefahren ausgesetzt. Deshalb muss kontinuierlich auf neue Gefahren bekannter Gefahrenklassen überprüft und auf bisher unbekannte Gefahrenklassen analysiert werden. Diese Dissertation beruht auf der Hypothese, dass die Sicherheitsbewertung eines Systems ein fortwährender Prozess ist. Dieser Prozess der Gefahren- und Gefahrenklassenerkennung wird von CUSTODIAN, einem erweiterbaren Framework zur Sicherheitsbewertung verteilter Systeme unterstützt. Für die Sicherheitsbewertung kombiniert CUSTODIAN einen architektonischen und einen epistemologischen Ansatz.

Der architektonische Ansatz beruht auf dem Blackboard-Muster und erlaubt die dynamische Erweiterung um Wissensquellen zur Erkennung und Visualisierung von Gefahren. Dadurch können Gefahren bekannter Gefahrenklassen in verteilten Systemen erkannt werden. Der epistemologische Ansatz definiert einen Prozess, der auf einem Metamodell zur Klassifizierung von Gefahren aufsetzt. Dieser Prozess erlaubt die bekannten Gefahrenklassen des Metamodells bei der Gefahrenerkennung eines verteilten Systems zu nutzen und das Metamodells um neu erkannte Gefahrenklassen zu erweitern.

Ein weiterer Beitrag dieser Dissertation ist die empirische Validierung von CUSTODIAN in drei Fallstudien, die mithilfe einer Referenzimplementierung durchgeführt wurden. In einer Fallstudie bei einem universitären Netzwerkdienstleister für Lehrstühle und andere Institutionen wurden in Firewalls etliche gefährliche Regeln entdeckt. In einer zweiten Fallstudie bei einem Onlinedienst konnte bei den Kunden die Erschleichung von Leistungen identifiziert und unautorisierte Administratoren eliminiert werden. In einem Hochsicherheitsrechenzentrum entdeckte CUSTODIAN allein in einer Abteilung über neuntausend Gefahren. Mithilfe der Visualisierungsmechanismen in CUSTODIAN wurde genug sozialer Druck aufgebaut, sodass die Gefahren binnen drei Wochen auf vier Gefahren reduziert wurden.

Abstract

IT-infrastructures are constantly exposed to existing threats and new threats which are caused by changes. Hence a continuous analysis has to be performed to detect new threats of known threat classes and unknown threat classes. This dissertation is based on the hypothesis that the security evaluation of a system is an ongoing process. This process of threat and threat class detection is supported by CUSTODIAN, an extensible framework for the evaluation of the security of distributed systems. CUSTODIAN combines an architectural and an epistemological approach to achieve the security evaluation.

The architectural approach is based on the blackboard pattern and supports the dynamic extension of knowledge sources for the detection and visualization of threats. This allows the detection of known threat classes in distributed systems. The epistemological approach defines a process which is based on a meta model for the classification of threats. This process enables the use of the known threat classes of the meta model for detecting threats in a distributed system und to extend the meta model with newly detected threats.

Another contribution of this dissertation is the empirical validation of CUSTODIAN in three case studies which where performed with the aid of a reference implementation. In the first case study on a university network provider for research groups and other institutions, several hazardous firewall rules where discovered. In the second case study with an online service provider, CUSTODIAN identified subreption of benefits and unauthorised administrators where eliminated within. In the third case study in a high security data center, CUSTODIAN could detect more that nine thousand threats in a single department. With the aid of visualisation mechanisms CUSTODIAN build up social pressure with the result that these threats where reduced to four within three weeks.

Inhalt

Danksagung **ii**

Kurzfassung **iii**

Abstract **iv**

1 Einleitung **1**

2 Terminologie **9**
2.1 Systeme 10
2.1.1 Verteilte Systeme 11
2.1.2 Komplexität verteilter Systeme 12
2.2 Komponenten 12
2.3 Konfiguration 14
2.4 Fehlkonfiguration, Gefahr und Schadensfall 16
2.4.1 Gefahrenklasse 19
2.4.2 Sicherheitstest 20
2.4.3 Testobjekt, Testmenge und Testdaten 23
2.4.4 Konfiguration Unter Test 25
2.4.5 Transformation von Konfigurationen zur Analyse 26
2.4.6 Synchronisation zwischen Komponenten 27
2.5 Gefahrenwissen 29
2.6 Wissensgebiet 30
2.7 Kapazitäten 31

3 Die Erkennung gefährlicher Konfigurationen **33**
3.1 Gefahren für Informationssysteme 34
3.2 Modellierung von Gefahren 36
3.3 Die Gefahrenpyramide 37
3.4 Wissen und Metawissen über Gefahren 39

3.5 Methoden der Gefahrenerkennung 40
3.6 Gefahrenexternalisierung 43
3.7 Visualisierung von Gefahren 44
3.7.1 Visual Analytics 44
3.7.2 Softwarekartographie 46

4 Forschungsmethodik 49
4.1 Grundlegende Forschungshypothesen 49
4.2 Sicherheitsframework 54
4.3 Vorgehensmodell 54
4.4 Fallstudien 55

5 Die Meta Threat Facility 57

6 Sicherheitsframework 63
6.1 Anforderungen 64
6.2 Analyse 68
6.2.1 Depot 71
6.2.2 Sicherheitsexperten 78
6.2.3 Steuereinheit 83
6.2.4 Das Datenmodell 86
6.2.5 Die Wahl des Architekturmusters 87
6.3 Systementwurf 91
6.4 Referenzimplementierung 94
6.4.1 Motion-Charts 97
6.4.2 Korrelationsexperte 98
6.4.3 Wissensbinder 100
6.4.4 Wissensbrowser 101
6.4.5 Gefahrenanalytiker 104

7 Das Vorgehensmodell HARVEST 107
7.1 Systemsicherheitsanalyse 108
7.1.1 Komponentenanalyse 108
7.1.2 Berechtigungsanalyse 115
7.1.3 Gefahrenanalyse 117
7.2 Sicherheitsentwurf 119
7.3 Sicherheitstest 122
7.4 Das HARVEST-Ereignismodell 124

8 Evaluation **127**
8.1 Fallstudie Hochsicherheitsrechenzentrum 129
8.1.1 Komponentenanalyse 130
8.1.2 Berechtigungsanalyse 131
8.1.3 Gefahrenanalyse 132
8.2 Fallstudie: RBG-Netzwerk 171
8.2.1 Komponenten- und Berechtigungsanalyse 172
8.2.2 Gefahrenanalyse 173
8.3 Fallstudie: AdWorks2go 181
8.3.1 Komponentenanalyse 182
8.3.2 Berechtigungsanalyse 184
8.3.3 Gefahrenanalyse 185
8.4 Auswertung 194
8.4.1 Fragebogen 195
8.4.2 Evolution der MTF 199
8.4.3 Die entdeckten Gefahrenklassen und Gefahren 202

9 Fazit und Ausblick **205**
9.1 Meta Threat Facility 206
9.2 CUSTODIAN 207
9.2.1 Evaluierung der Einsetzbarkeit 207
9.2.2 Erkennung von Beta-Fehlern durch Fehlerinjektion . . . 207
9.2.3 Einbeziehung von Akteuren in die Konfiguration 209
9.2.4 Automatisierte Erkennung der Gefahrenklassen 209

A Appendix **211**
A.1 IT-Sicherheit 211
A.1.1 Intrusion Detection System 211
A.1.2 Autorisierung und Authentifizierung 211
A.1.3 Firewalls 213
A.1.4 Data Center Infrastructure Management 214
A.2 Komponentenbeschreibungen der verteilten Systeme 217
A.2.1 Fallstudie Hochsicherheitsrechenzentrum 217
A.2.2 Fallstudie RGB-Netzwerk 229
A.2.3 Fallstudie AdWorks2go 231

Abkürzungen **235**

Abbildungsverzeichnis **242**

Tabellenverzeichnis **243**

1 Einleitung

Die herausragende Bedeutung der Sicherheitsfragen bei Informationssystemen bedarf heute wohl kaum noch einer weiteren Begründung. Aber leider: So nachvollziehbar und wichtig der Wunsch nach Systemsicherheit ist, so schwierig ist seine Realisierung. Fehlervermeidung, Fehlertoleranz und Fehlerentdeckung sind die drei wesentliche Ansätze, mit deren Hilfe man Systeme vor Gefahren schützen und dadurch deren Sicherheit verbessern kann. Die Techniken der Fehlervermeidung basieren auf einem geschlossenen Weltbild (closed world assumption) und basieren auf der Hypothese, dass alle Gefahren bekannt sind. Die Fehlervermeidung erfordert, dass durch den Entwurf des Systems alle Gefahren ausgeschlossen werden können. Der Ansatz erfordert bei der Adaption bestehender Systeme einen revolutionäre Vorgehensweise, da das System neu entworfen werden muss. Im Gegensatz zur Fehlervermeidung basieren die Ansätze der Fehlertoleranz und der Fehlerentdeckung auf einem offenen Weltbild (open world assumption). Diese Ansätze gehen davon aus, dass nicht alle möglichen Arten von Fehlern bekannt sind. Fehlervermeidung ist ein vorbeugender Ansatz, der während des Systementwurfs aber auch evolutionär während des Betriebs anwendbar ist. Die Fehlerentdeckung ist ein evolutionär Ansatz der permanent darum bemüht ist, die Hypothese, dass ein System sicher ist, zu invalidieren.

Die Ideen der Fehlervermeidungsstrategie finden eine gewisse Analogie in den Bemühungen der Mathematik um eine Grundlegung im Rahmen des *Formalismusprogrammes* im ersten Drittel des 20. Jahrhunderts. Für jede mathematische Theorie sollten alle korrekten Aussagen aus einem widerspruchsfreien Axiomensystem abgeleitet werden können. David Hilbert war der führende Kopf dieser Denkrichtung, und auch die Bemühungen von Alfred Whitehead und Betrand Russell mit der *Principia Mathematica* [WR27] sind in diesem Kontext zu sehen. Erst durch Kurt Gödels *Unvollständigkeitssatz* wurde 1931 die Undurchführbarkeit des Formalismusprogrammes nachgewiesen.

Die Informatik verfolgte zunächst ebenfalls einen axiomatischen Ansatz zur Verifikation von Programmen. Gesucht war ein Algorithmus, der für jedes Programm feststellt ob es das Gewünschte leistet. Alan Turing bewies mit der Unentscheidbarkeit des Halteproblems [Tur36], dass algorithmisch nicht einmal festgestellt werden kann, ob ein Programm überhaupt ein Ergebnis liefert. Die Fehlerhypothese für Software, also die Annahme, dass jede Software Fehler enthält, ist in der Praxis weit verbreitet. Maurice Wilkes stellte bereits im Juni 1949 ernüchtert fest [Wil85]:

> *... a good part of the remainder of my life was going to be spent in finding errors in my own programs ...*
>
> Maurice Wilkes

Techniken der Fehlervermeidung werden beim Gewähren von Sicherheit in Informationssystemen trotzdem erfolgreich eingesetzt. Das Data Center Infrastructure Management (DCIM) bietet beispielsweise für Rechenzentren verschiedene Ansätze um bereits im Systementwurf die Sicherheit zu steigern [HG15, DTM10]. Durch die strikte Trennung von Sicherheitsregeln und deren Implementierung schafft das Policy-Based Management (PBM) die Möglichkeit, Sicherheitsregeln unabhängig von konkreten Komponenten zu definieren [Str03]. Beim PBM werden zwei gegensätzliche Ansätze verfolgt: Ein autonomer Ansatz erlaubt es, alle sicherheitsrelevanten Konfigurationen auf der Komponente selbst zu hinterlegen [KNMC06]. Das ermöglicht eine geschlossene Sicherheitskonfiguration der Komponente ohne Abhängigkeiten zu anderen Komponenten. Beim zentralisierten Ansatz sind die sicherheitsrelevanten Konfigurationen aller Komponenten in einer sogenannten Policy Information Base (PIB) zusammengeführt [SHC+01a]. Die Sicherheitseinstellungen eines verteilten Systems werden in einer Datenbank vorgehalten, und das System kann als ganzes konfiguriert und auf Sicherheit evaluiert werden. Diese vereinheitlichte Konfiguration eines verteilten Systems erlaubt beispielsweise ein systemweites Konzept für die Umsetzung von Role-Based Access Control (RBAC) [SJN05]. PBM erfordert jedoch die Kompatibilität aller Komponenten mit einem definierten PBM-Standard. Bei existierenden verteilten Systemen mit Komponenten, die PBM nicht unterstützen, ist die Migration zum PBM-Ansatz jedoch nicht ohne das Austauschen dieser Komponenten möglich.

Die Grenzen der Fehlervermeidung liegen allerdings darin, dass IT-Systeme ständig neuen Gefahren ausgesetzt sind, die erst *nach* dem Entwurf des

Systems bekannt werden. Allein beim Betriebssystem Windows werden monatlich mehrere Sicherheitslücken entdeckt [GLM12]. Eine steigende Anzahl von Sicherheitslücken legt die in Abbildung 1.1 dargestellte Entwicklung nahe. Und durch jede Veränderung der Konfiguration kann eine neue Gefahr für das IT-System entstehen [Whi03, WM10]. Diese Gefahren können IT-Systeme über Jahre hinweg gefährden. Beispielsweise war die als sehr gefährlich eingestufte Sicherheitslücke Shellshock über 20 Jahre hinweg in Unix-basierten Systemen enthalten [WY15]. Und der Programmfehler Heartbleed kompromittierte aufgrund eines nicht abgefangenen Buffer-Overflows das als sicher geltende Verschlüsselungsprotokoll TLS über zweieinhalb Jahre lang [DKA+14]. Diese Vorfälle zeigen exemplarisch auf, dass selbst Verfahren, deren Sicherheit mathematisch bewiesen ist, durch fehlerhafte Implementierung oder Nutzung unsicher sein können.

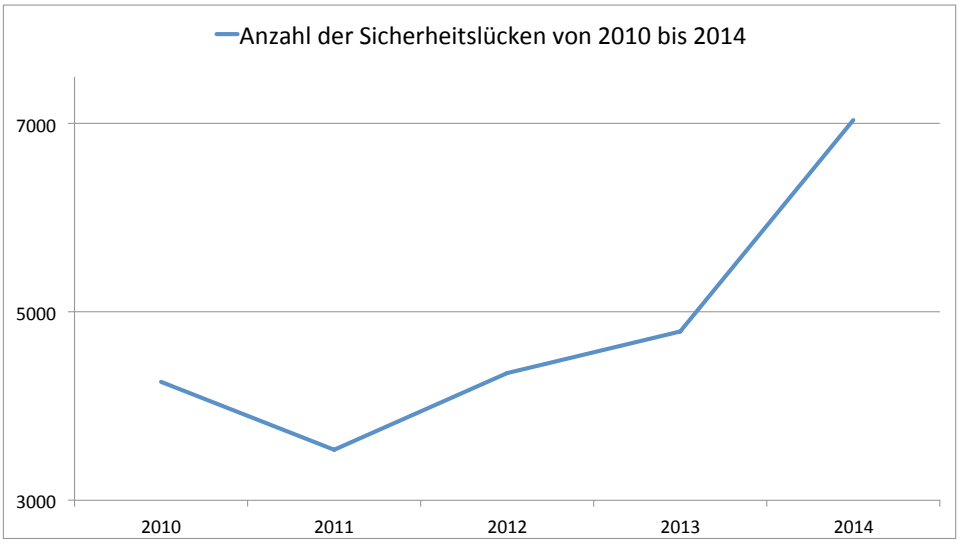


Abb. 1.1: *Zwischen 2010 und 2014 stieg die Anzahl der bei der NVD gemeldeten Sicherheitslücken um 65 Prozent*

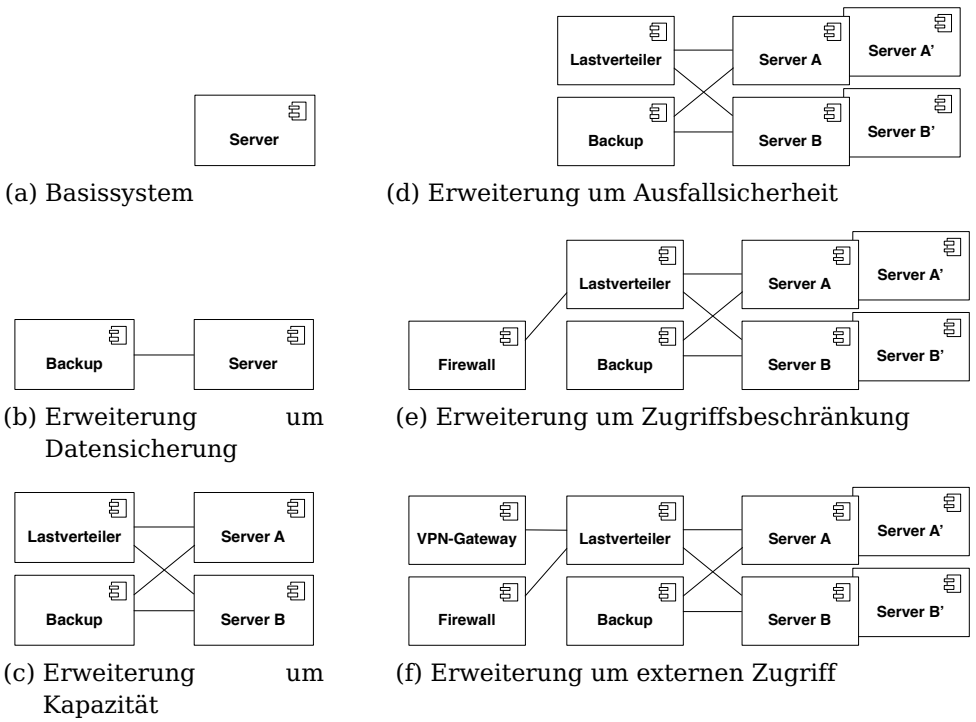
Diese Dissertation geht davon aus, dass man vollständige Sicherheit nicht durch Entwurf garantieren kann. Aufbauend auf den Erkenntnissen von Karl Popper [Pop63, PV74] verfolgt sie einen evolutionären Ansatz der auf Falsifikation beruht. Ein Beitrag dieser Dissertation ist das Vorgehensmodell HARVEST, welches darauf abzielt, die Sicherheit eines Systems zu falsifizieren. Ziel dieses Prozesses ist das kontinuierliche Erkennen neuer Gefahren im System. Die Eigenschaften neuer Gefahren sind unvorhersehbar, müssen aber bei Sicherheitsbewertung des Systems in die Analyse einbezogen werden. Ein weiterer Beitrag dieser Dissertation ist das Sicherheitsframework

CUSTODIAN, das auf einer erweiterbaren Blackboard-Architektur fußt Diese erlaubt es die Sicherheit eines Systems zu evaluieren. Mit Hilfe von HARVEST werden drei Fallstudien durchgeführt um CUSTODIAN zu evaluieren.

Interzeption (Spionage), Ausfall, Modifikation und Fälschung kategorisieren die Gefahren, die auf IT-Infrastrukturen einwirken [PP06]. Diesen Gefahren stehen unterschiedliche Sicherheitsansätze zum Schutz von IT-Infrastrukturen entgegen: Verschlüsslung, Authentifizierung, Autorisierung und Auditierung [Ste03]. Datensicherung und Redundanz schützen vor dem Verlust von Daten und dem Ausfall von Diensten. Die Einführung dieser Sicherheitsansätze erfordert die Erweiterung der IT-Infrastrukturen um zusätzliche Komponenten. Nach jeder Erweiterung muss die Sicherheit der IT-Infrastruktur erneut evaluiert werden. Abbildung 1.2 stellt die Evolution eines einfachen Systems hin zu einem verteilten System am Beispiel eines Startups dar. Das Basissystem stellt mehrere Dienste zur Verfügung (Abbildung 1.2a), ohne die Anforderungen Datensicherheit, Skalierbarkeit, Ausfallsicherheit, Zugriffsbeschränkung und externer Zugriff zu erfüllen. Eine Erweiterung um ein Backupsystem bietet die Möglichkeit der Datensicherung (Abbildung 1.2b). Ein zweiter Server skaliert die Kapazität des Dienstes (Abbildung 1.2c). Weitere Server erhöhen die Ausfallsicherheit, indem sie die bestehenden Server spiegeln (Abbildung 1.2d) Eine Firewall schützt die Dienste vor unerlaubten Zugriffen (Abbildung 1.2e). Und das VPN-Gateway ermöglicht den Zugriff auf die Dienste von außen (Abbildung 1.2f) [Ven01].

Durch die Realisierung dieser zusätzlichen Anforderungen ist die IT-Infrastruktur von einer Komponente auf acht Komponenten mit fünf verschiedenen Komponententypen sowie zusätzlichen Abhängigkeiten zwischen den Komponenten [KC99] angewachsen. Die Komplexität des verteilten Systems ist aufgrund der größeren Anzahl von Komponenten und interkomponentalen Abhängigkeiten gestiegen [Hof98]. Die Sicherheitsevaluation wird mit steigender Komplexität des verteilten Systems aufwendiger. Diese aufwendigere Sicherheitsevaluation muss bestenfalls nach Erweiterung der Komponenten und Veränderung der Konfiguration des verteilte System durchgeführt.

IT-Infrastrukturen sind in der Realität teilweise weitaus komplexer als im Abbildung 1.2f aufgezeigten Beispiel. Sie können viele Dienste mit unterschiedlichen funktionalen und nichtfunktionalen Anforderungen beherbergen. Jede Komponente eines verteilten Systems besitzt eine Konfiguration [Tic15, KM85], welche die Arbeitsweise der Komponente definiert. Am Bei-



(a) Basissystem

(b) Erweiterung um Datensicherung

(c) Erweiterung um Kapazität

(d) Erweiterung um Ausfallsicherheit

(e) Erweiterung um Zugriffsbeschränkung

(f) Erweiterung um externen Zugriff

Abb. 1.2: *Exemplarische Evolution einer IT-Infrastruktur*

spiel der Abbildung 1.2 ist die Konfiguration des verteilten Systems auf alle acht Komponenten verteilt. Auf den Servern ist konfiguriert wer sich dort anmelden darf, auf dem Backupsystem ist konfiguriert, welche Daten wie gesichert werden sollen, auf dem Lastverteiler ist konfiguriert, wie die Server zusammenhängen. Die Firewall konfiguriert welche Netzwerkverbindungen aufgebaut werden dürfen und auf dem VPN-Gateway ist Konfiguriert wer eine VPN-Verbindung aufbauen darf. Der Zusammenschluss mehrerer Komponenten zu einem verteilten System zieht eine Verteilung der Konfiguration auf mehrere Komponenten nach sich.

Die Konfigurationen der Komponenten sind außerdem oft heterogen. Diese Heterogenität kann sowohl durch unterschiedliche Datenformate als auch durch unterschiedliche Konfigurationsschnittstellen der Komponenten des verteiltes Systems bedingt sein. Firewallkonfigurationen werden beispielsweise oft als Liste dargestellt, in Testdateien gespeichert und per SSH administriert. Eine VPN-Gateways-Konfiguration besitzt oft eine hierarchische Gruppenstruktur zur Verwaltung der Benutzer, wird in HTML repräsentiert

und per Webbrowser administriert. Diese unterschiedlichen Merkmale der Konfigurationen der einzelnen Komponenten führen zu einer heterogenen Gesamtkonfiguration des verteilten Systems.

Die Gesamtkonfigurationen verteilter Systeme können somit komplex, verteilt und heterogen sein. Hamed und Al-Shaer beschreiben in ihrer Taxonomie, wie kollidierende heterogene Anforderungen Auswirkungen auf die Sicherheit, insbesondere auf die Gefährdung eines Systems haben [HAS06].

Die Gefahrenanalyse in komplexen, verteilten und heterogen verteilten Systemen steht deshalb im Fokus dieser Dissertation. Eine Gefahr liegt genau dann vor, wenn die Konfiguration des verteilten Systems ein unerwünschtes Verhalten zulässt. Eine Gefahrenklasse fasst die Menge der gleichartigen Gefahren zusammen. Die Gefahrenklasse „Unerlaubte externe Erreichbarkeit“ beschreibt beispielsweise jede Möglichkeit, eine rein interne Komponente von außerhalb des Systems zu erreichen. Jede Firewallregel, welche Externen den Zugriff auf eine interne Komponente gewährt, ist eine Gefahr dieser Gefahrenklasse.

Das Ziel dieser Dissertation ist es, ein Vorgehensmodell für die Gefahrenanalyse zu definieren, mit dem man neue Gefahrenklassen erkennt, und diesen Prozess mit einer Architektur zu kombinieren, die es erlaubt, Gefahren bekannter Gefahrenklassen nach einer Systemänderung automatisch zu erkennen.

Die Dissertation ist folgendermaßen aufgebaut. In Kapitel 2 werden erforderliche Grundlagen zu verteilten Systemen beschrieben. Der aktuelle Forschungsstand zu den Themen Fehler, Gefahren und Gefahrenerkennung wird in Kapitel 3 dargelegt. Darüberhinaus geht das Kapitel auf existierende Modelle ein, welche es ermöglichen, Wissen zu modellieren oder Gefahren zu beschreiben. Kapitel 4 stellt die Forschungshypothese dieser Dissertation vor, erläutert die sich daraus ableitenden Bedingungen und gibt einen Überblick über die Methoden zur Stärkung der genannten Forschungshypothese.

Die Meta Threat Facility (MTF) ist ein erweiterbares Modell für die Definition von Gefahrenklassen. Kapitel 5 stellt MTF vor und zeigt auf, wie diese das Auffinden von Gefahren in verteilten Systeme und bei der Modellierung von Gefahren in UML unterstützt. Neben der MTF ist das Sicherheitsframework CUSTODIAN der zweite Kern dieser Dissertation.

Kapitel 6 beschreibt die Sicherheitsarchitektur von CUSTODIAN, insbesondere die zugrundeliegenden Anforderungen, die Analyse, der System- und der Detailentwurf. Außerdem behandelt das Kapitel 6 eine Referenzimplementierung des Sicherheitsframeworks CUSTODIAN namens KUSTOS.

Kapitel 7 beschreibt das Vorgehensmodell HARVEST, welches speziell für die Gefahrenerkennung in verteilten Systeme unter Einsatz von CUSTODIAN definiert ist. HARVEST beschreibt nicht nur das Vorgehensmodell für den Einsatz von CUSTODIAN, sondern auch den Prozess der durchgeführten Fallstudien. Das Kapitel beschreibt auch das UML-Profil „Sicherheitsdiagramm“, ein weiterer Beitrag dieser Dissertation. Sicherheitsdiagramme erlauben die kompakte Modellierung eines verteilten Systems, indem sie Konzepte von Klassendiagrammen und Komponentendiagrammen vereinen.

Kapitel 8 beschreibt die Einsatzfähigkeit des Vorgehensmodells HARVEST und die Adaptionsfähigkeit der Architektur CUSTODIAN anhand von drei Fallstudien. CUSTODIAN half während der Fallstudien die verteilten Systeme auf die Gefahren der insgesamt zehn erkannten Gefahrenklassen zu analysieren. Dabei entdeckte CUSTODIAN bis zu 19.000 Gefahren und half beim Eliminieren dieser Gefahren. Ein Fragebogen diente der Evaluierung der Resultate der Fallstudien, in dem die Sicherheitsverantwortlichen den Einsatz von CUSTODIAN bewerteten. Kapitel 9 fasst die Beiträge dieser Dissertation zusammen und gibt einen Ausblick über weiterführende Forschungsideen.

2 Terminologie

Dieses Kapitel befasst sich mit den grundlegenden Begriffen im Bereich der Gefahrenerkennung in verteilten Systemen. Einige dieser Begriffe sind mit einer abweichenden oder engeren Bedeutung als im allgemeinen Sprachgebrauch versehen. Wie beispielsweise der Terminus Kapazität, der in dieser Arbeit eine Abstraktion der Begriffe Spezialist und Experte definiert. Weder die Deutsche Industrie-Norm (DIN) noch die International Organization for Standardization (ISO) definieren die benötigten Begriffe noch grenzen sie die für diese Arbeit entscheidenden Unterschiede hinreichend voneinander ab.

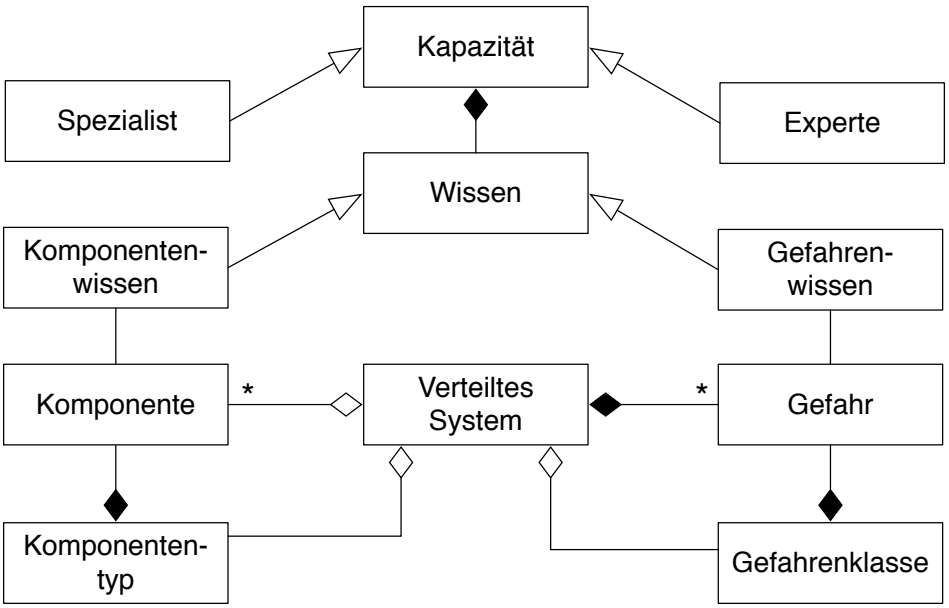


Abb. 2.1: *Termini der Gefahrenerkennung in verteilten Systemen*

Das Klassendiagramm in Abbildung 2.1 stellt die zentralen Begriffe dieser Dissertation in Relation. Verteilte Systeme und deren Abgrenzung zu Systemen sind in Abschnitt 2.1.1 beschrieben. Abschnitt 2.1.2 definiert, bei welchen verteilten Systemen die Gefahrenerkennung problematisch ist.

Abschnitt 2.2 beschreibt den Zusammenhang zwischen verteilten Systemen, Komponenten, Komponententypen und Konfigurationen. Abschnitt 2.3 geht auf die Bedeutung der Konfiguration ein. Die Gefahren, die durch den in dieser Dissertation vorgestellten aufgedeckt werden sollen, entstehen durch fehlerhafte Konfigurationen. Sowohl fehlerhaften Konfigurationen aber auch Gefahren, Gefahrenklassen, Sicherheitstests und Schadensfälle werden in Abschnitt 2.4 erläutert.

Das für die Erkennung von Gefahren benötigte Wissen wird in dieser Dissertation Gefahrenwissen genannt. Diese Dissertation definiert einen Sicherheitstest als Kombination von Gefahrenwissen verschiedener Wissensklassen, welche die Erkennung von Gefahren erlaubt (siehe Abschnitt 2.4.2). In Abschnitt 2.5 ist der Begriff des Gefahrenwissens im Allgemeinen beschrieben. Die Träger von Gefahrenwissen heißen Kapazitäten (siehe Abschnitt 2.7) und können entweder durch einen Experten oder einen Spezialisten verkörpert sein.

2.1 Systeme

System ist ein Lehnwort des altgriechischen Wortes Sỳstēma (altgr. σὺστεμα) und bedeutet soviel wie „aus mehreren Einzelteilen zusammengesetztes Ganzes“ [Wik15]. Mit der Norm DIN 19226 [DIN94] des Deutschen Instituts für Normung (DIN) existiert ein industrieller Standard, der sich mit den Begrifflichkeiten eines Systems befasst. Klostermann [Klo07] fasst die Systembeschreibung der Norm wie folgt zusammen:

Im Kontext eines verteilten Systems stellt sich die Frage der Einordnung des Begriffs System. Hierfür paraphrasieren wir „verteiltes System“ im Vokabular der DIN 19226 als „Anordnung von Elementen“. Sowohl bei der Anordnung als auch bei den Elementen es sich um Systeme handeln. Da laut DIN beides möglich ist, wird im Rahmen dieser Arbeit bei einer Anordnung von einem verteilten Systeme gesprochen und bei den einzelnen Elementen von Komponenten. Die Komponenten können selbst wieder eigenständige Systeme sein. Wir gehen deshalb von folgender Definition aus:

Ein System besteht aus einer gegebenen Anordnung von Elementen, die miteinander in Beziehung stehen. Das System wird aufgrund bestimmter

Vorgaben von seiner Umgebung abgegrenzt. Die Elemente können ihrerseits wieder Systeme sein.

2.1.1 Verteilte Systeme

Der Begriff der verteilten Systeme tritt seit etwa 1980 in Erscheinung. Bengler legt dar, wie deren Verbreitung durch die erhöhte Leistungsfähigkeit von Prozessoren, dem Aufkommen lokaler Netzwerke und das Internet begünstigt wurde [Ben13]. Die Autoren Coulouris, Dollimore und Kindberg definieren in ihrem Buch [CDK02] verteilte Systeme folgendermaßen: „Als verteiltes System wird ein System bezeichnet, bei dem sich die Hardware- und Softwarekomponenten auf vernetzten Rechnern befinden und nur über den Austausch von Nachrichten kommunizieren und ihre Aktionen koordinieren.“ Coulouris et. al. folgern aus dieser Definition drei Charakteristiken: Die Komponenten des verteilten Systems sind nebenläufig, haben keinen globalen Zeitgeber und können unabhängig voneinander ausfallen [CDK05].

Tanenbaum legt bei der Beschreibung verteilter Systeme den Fokus besonders auf die Art und Weise, in der sie sich dem Benutzer gegenüber offenbaren: „A distributed operating system is one that looks to its users like an ordinary centralized operating system but runs on multiple, independent central processing units (CPUs)“ [TVR85] und „A distributed system is a collection of independent computers that appears to its users as a single coherent system“ [TVS07]. Er setzt bei verteilten Systemen voraus, dass sie sich dem Benutzer als ein zentral operierendes System darstellen. Ein cyber-physikalisches System kann sich dem Benutzer als mehrere, scheinbar unabhängige Systeme darstellen und wird von Tannenbaums Definition ausgeschlossen. Im Rahmen dieser Dissertation werden jedoch auch cyber-physikalische Systeme betrachtet.

Wellings sieht die Verfolgung eines gemeinsamen Ziels als entscheidenden Faktor: „A distributed computer system is defined to be a system of multiple autonomous processing elements cooperating in a common purpose or to achieve a common goal“ [Wel04]. Wellings Voraussetzung des gemeinsamen Ziels kann beispielsweise in einem Rechenzentrum, welches viele heterogene Dienste anbietet, nur mittels sehr abstrakter Zieldefinitionen erreicht werden: „Die Komponenten des Rechenzentrums dienen der Erfüllung der Anforderungen der Nutzer“. Legt man dem Ziel die verschiedenen funktio-

nalen Anforderungen der bereitgestellten Dienste zu Grunde, lässt sich oft keine gemeinsame Zieldefinition finden.

Im Rahmen dieser Dissertation spielt die Konfigurierbarkeit mehrerer Komponenten eines verteilten Systems die zentrale Bedeutung. Als grundlegende Definition zur Charakterisierung verteilter Systeme verwenden wir folgende Definition. Ein verteiltes System besteht aus mehreren individuell konfigurierten, interagierenden Komponenten. Die Konfiguration passt eine allgemein ausgelegte Komponente auf die individuellen Anforderungen ihres Einsatzkontexts und die Interaktion mit andere Komponenten an.

2.1.2 Komplexität verteilter Systeme

Verteilte Systeme unterscheiden sich in ihrer Komplexität. Die Komplexität eines verteilten Systems ist ebenfalls nicht allgemein definiert. Verschiedene Einflussfaktoren bestimmen die Komplexität verteilter Systeme. Haeberlen et. al. [HKD07] beschreiben die Anzahl der Komponenten als einen dieser Faktoren. Die Anzahl der Komponententypen, die Komplexitäten der bereitgestellten Dienste und die Anzahl der für den Betrieb notwendigen Experten haben ebenfalls Einfluss auf die Komplexität des verteilten Systems. Für den Zweck dieser Dissertation sind verteilte Systeme komplex, wenn Gefahrenwissen mehrerer Spezialisten bei der Gefahrenerkennung notwendig ist.

2.2 Komponenten

Unter einer Komponente versteht man einen Bestandteil eines Systems. Der UML-Standard definiert Komponenten wie folgt:

> *A component can always be considered an autonomous unit within a system or subsystem. It has one or more provided and/or required interfaces (potentially exposed via ports), and its internals are hidden and inaccessible other than as provided by its interfaces.*
>
> – UML-Spezifikation, Version 2.4.1 [BRJ11]

Diese Dissertation definiert eine Komponente als einen Bestandteil eines verteilten Systems, der aus der Sicht des Betriebs monolithisch ist und deren innere Struktur nicht von Bedeutung ist. Wenn beispielsweise eine Firewall aus einer Hauptplatine, einem Netzteil und einem Speichermodul besteht, nimmt der Betreiber sie als nicht untrennbare Einheit wahr. Somit ist eine Komponente bei den Betrachtungen dieser Dissertation ein Monolith, der aus der Sicht des Betriebs keine Subsysteme besitzt. Ein verteiltes System ist eine Aggregation von Systemen (siehe Abbildung 2.2), indem die Komponenten die Blätter und die untergliederbaren Systeme die Komposita darstellen. Jede Komponente gehört einem Komponententyp an.

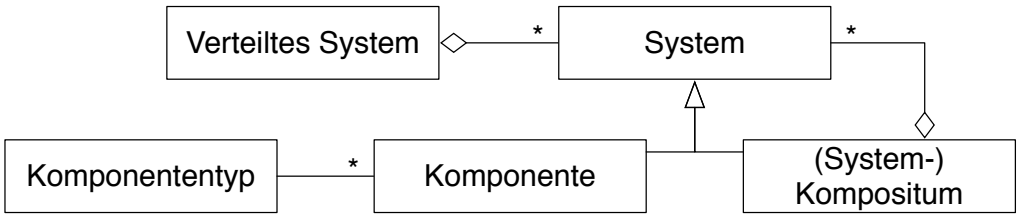


Abb. 2.2: *Komponenten als Teil eines verteilten Systems*

Komponenten haben eine spezifizierte Funktionsweise, eine beobachtete Funktionsweise und eine Konfiguration (siehe Abbildung 2.3). Die spezifizierte Funktionsweise beschreibt das vom Hersteller beabsichtigte Verhalten einer Komponente und wird durch ihren Komponententyp definiert. Die beobachtete Funktionsweise beschreibt das tatsächliche Verhalten einer Komponente. Die spezifizierte Funktionsweise und die beobachtete Funktionsweise können sich voneinander unterscheiden. Ein solcher Unterschied kann durch Konstruktionsfehler des Herstellers, fehlerhafte Bauteile der Komponente oder Hardwareschaden hervorgerufen werden. Verändert ein Update der Firmware das Verhalten einer Komponente, hat diese danach eine andere spezifizierte Funktionsweise und gehört einem anderen Komponententyp an.

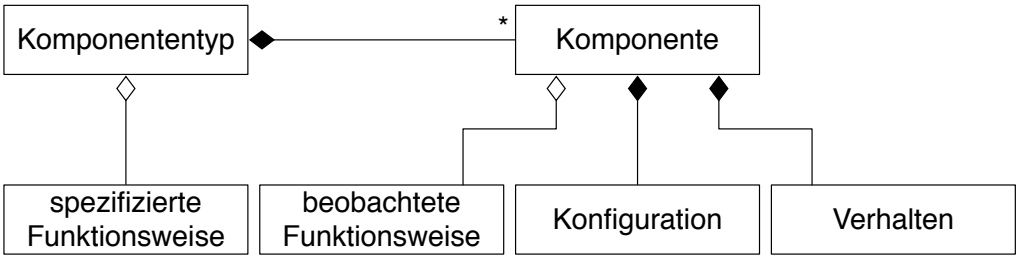


Abb. 2.3: *Spezifizierte Funktionsweise, beobachtete Funktionsweise, Konfiguration und Verhalten einer Komponente*

Mit der Konfiguration passt der Betreiber die Komponente für ihren Einsatz an. Die Konfiguration ist hinsichtlich der Funktionsweise ein externer Faktor, der auf die Komponente einwirkt. Die Funktionsweise ist somit das tatsächliche, konfigurationsunabhängige Verhalten einer Komponente.

Beispiel 1
In einem verteilten System ist eine Firewall vom Komponententyp „PA-500 Version 2.37“ installiert. Der Hersteller bringt die „Version 2.38“ als Update zur Behebung einer Sicherheitslücke heraus und ein Administrator spielt dieses ein. Durch die Behebung einer Sicherheitslücke hat sich die beobachtete Funktionsweise der Firewall geändert. Deshalb hat sich die Version der Firewall auf „PA-500 Version 2.38“ geändert.

2.3 Konfiguration

Die Konfiguration eines verteilten Systems ist auf dessen Komponenten verteilt. Diese verteilte Konfiguration ist somit eine Komposition der Konfigurationen aller Komponenten des verteilten Systems. Das Modell in Abbildung 2.4 definiert die Zusammenhänge zwischen dem verteilten System, dessen Komponenten und der Konfiguration. Das Konfigurationsaggregat setzt sich aus den Konfigurationen der einzelnen Komponenten zusammen und definiert die Berechtigungen. Die Berechtigungen definieren, welche Rechte für die Systeme des verteilten Systems gelten. Die Komposition der Konfigurationen aller Komponenten eines verteilten Systems bezeichnen wir als Konfigurationsaggregat oder als Konfiguration des verteilten Systems. Da die Subsysteme voneinander abhängig sind, müssen die Einzelkonfigurationen des Konfigurationsaggregats auch gemeinsam evaluiert werden, um Fehler zu erkennen.

Der Begriff Konfiguration unterscheidet sich in dieser Dissertation in seiner Bedeutung von der Bedeutung im Konfigurationsmanagement. Das Konfigurationsmanagement als Teildisziplin des Softwareengineerings beschäftigt sich mit der „... Identifizierung von Konfigurationselementen, um ein System als eine Menge von sich entwickelnden Komponenten zu modellieren ...“ [BD04]. Im Konfigurationsmanagement definiert eine Konfiguration die Menge der Komponenten, aus denen ein System besteht. In dieser Dissertation definiert eine Konfiguration die individuelle Anpassung einer Komponente für ihren Einsatzkontext.

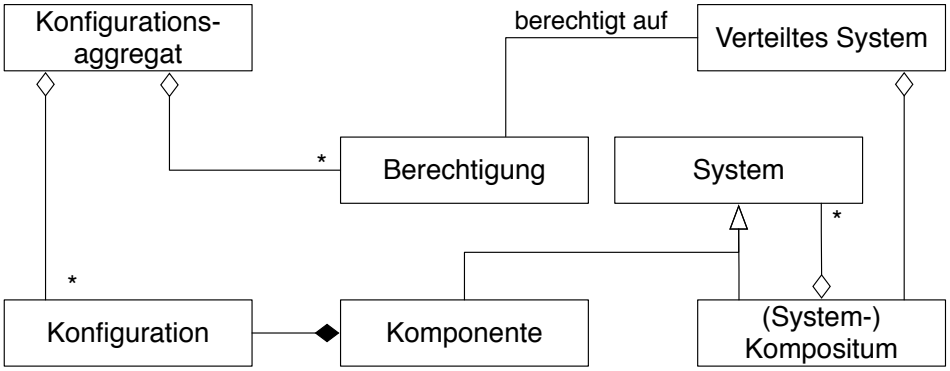


Abb. 2.4: *Die Konfiguration eines verteilten Systems*

An Komponenten kann man Einstellungen vornehmen. Firewalls erlauben beispielsweise die Definition von Regeln. Die Konfiguration einer Komponente ist die Menge aller vom Betreiber vorgenommenen Einstellungen dieser Komponente um diese individuell für ihre Verwendung anzupassen. Die Konfiguration einer Komponente kann sich unabhängig von der Funktionsweise der Komponente verändern. Das Zusammenwirken der beobachteten Funktionsweise und der Konfiguration einer Komponente bestimmt deren Verhalten (siehe Abbildung 2.3).

Unterschiedlich spezifizierte aber gleich konfigurierte Komponenten können daher ein unterschiedliches Verhalten aufweisen. Das kann man in der Praxis bei sogenannten Firmware-Updates beobachten. Durch ein solches Update ändert sich die Spezifikation der Komponente. Das Verhalten kann sich trotz gleichbleibender Konfiguration ändern. Eine Konfiguration kann aus mehreren Konfigurationsklassen bestehen. So sind beispielsweise in einem LDAP [How95] Gruppen und Personen konfiguriert (siehe Abbildung 2.5). Jede Gruppe besteht aus einer Menge von Personen. Die Konfigurationsklassen des LDAPs sind `Gruppe` und `Person`, die Assoziationsklasse ist `Mitgliedschaft`. Eine Konfigurationsklasse definiert eine Klasse von Informationen die in der Komponente konfigurierte werden kann.

Abb. 2.5: *Die Konfigurationsklassen eines LDAP-Verzeichnis*

2.4 Fehlkonfiguration, Gefahr und Schadensfall

Das Reliability Engineering unterscheidet zwischen Defekten[1] (fault), fehlerhaften Zuständen (erroneous state) und Störfällen (failure). „Ein Defekt ist ein Entwurfs- oder Codierungsfehler, der regelwidriges Komponentenverhalten verursachen kann. Ein fehlerhafter Zustand ist ein unerwünschter Systemzustand, der durch einen oder mehrere Fehler verursacht wird und zu einem Störfall führen kann. Ein Störfall ist eine Abweichung zwischen der Spezifikation und dem tatsächlichen Verhalten eines Systems. Nicht alle fehlerhaften Zustände lösen einen Störfall aus.“ [BD04]

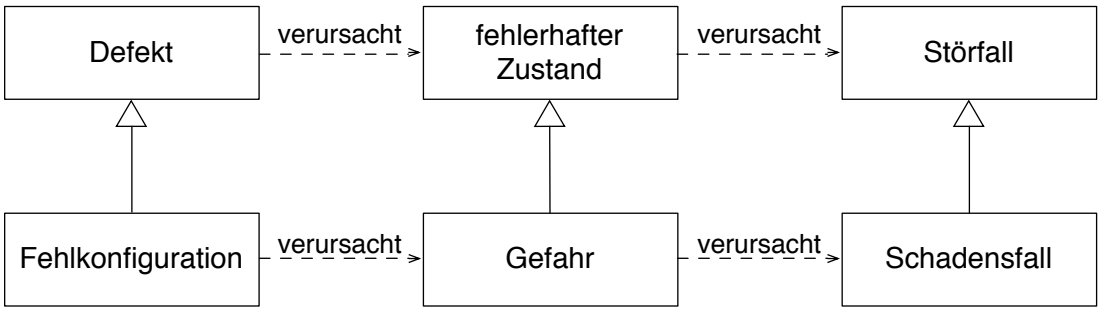


Abb. 2.6: *Herleitung und Zusammenhang von Fehlkonfigurationen, Gefahren und Schadensfällen*

Für die Beurteilung von Konfigurationen verteilter Systeme gelten im Rahmen dieser Dissertation die Fachbegriffe Fehlkonfiguration, Gefahr und Schadensfall. In Abbildung 2.6 sind die Abhängigkeiten der Begriffe Fehlkonfiguration für Defekt, Gefahr für fehlerhafter Zustand und Schadensfall für Störfall und deren Herleitungen aus dem Reliability Engineering aufgezeigt. Eine Konfiguration gilt als Fehlkonfigurationen bzw. fehlerhafte Konfigurationen, wenn durch sie ein regelwidriges Verhalten[2] des verteilten Systems verursacht werden kann. Ein Fehlkonfigurationen kann durch bewusstes oder unbewusstes Handeln entstehen. Ist der Konfigurierende sich dem vollen Ausmaß seines Handelns nicht bewusst, kann das zu einer unbeabsichtigten herbeigeführten Fehlkonfiguration führen. Eine gezielt von einem Angreifer manipulierte Konfiguration führt zu einer beabsichtigt herbeigeführten Fehlkonfiguration.

[1] Fehler ist ein häufig verwendetes Synonym für Defekt. Der Begriff Fehler ist in der deutschen Sprache mehrdeutig.

[2] Ein regelwidriges Verhalten ist ein vom Betreiber nicht beabsichtigtes Verhalten.

Diese Dissertation definiert eine Gefahr als einen unerwünschter Zustand eines verteilten Systems, der durch ein oder mehrere Fehlkonfigurationen verursacht wird und zu einem Schadensfall führen kann. Hiermit unterscheidet sich das Verständnis von Gefahren zu der der DIN-Norm[3]. Fehlkonfiguration können, müssen aber nicht zu einer Gefahr führen. Denn fehlertolerante Systeme können Fehlkonfigurationen bis zu einem gewissen Maß abfangen. Eine Fehlkonfiguration, die bei einer bestimmen Konstellation nicht zu einer Gefahr führt, kann nach der Veränderung des verteilten Systems zu einer Gefahr führen. Dies ist der Fall, wenn erst durch eine zweite Fehlkonfiguration eine Gefahr vorliegt. Andere Fehlkonfigurationen (wie beispielsweise ein Rechtschreibfehler in einer Benutzeroberfläche) führen zu keiner Gefahr. Doch auch fehlerhafte Konfigurationen, die ohne akute Auswirkung auf das verteilte System sind, können eine Gefahr darstellen (siehe Beispiel 2).

Beispiel 2

Ein Administrator soll zwei externen Personen die notwendigen Rechte gewähren. Dazu fügt er die Externen der Gruppe ihrer assoziierten Abteilung zu. Jedoch ist diese Gruppe nur für interne Mitarbeiter vorgesehen und später sollen ihr explizit interne Rechte zugewiesen werden. Weil diese Konfiguration unbeabsichtigt Personen der falschen Gruppe zuweist, ist sie fehlerhaft. Und weil sie bei einer späteren korrekten Zuweisung eines internen Rechts an die Gruppe schädlich sein kann, ist die Konfiguration eine Gefahr.

Ein Schadensfall ist eine durch ein oder mehrere Gefahren verursachte schadhafte Auswirkung auf das verteilte System. Im Schadensfall entsteht dem Betreiber oder einem Benutzer des verteilten Systems ein Schaden. Verschiedene Bereiche können von einem solchen Schaden betroffen sein. Das verteilte System selbst wird kompromittiert, wenn vertraulichen Informationen offengelegt bzw. manipuliert werden oder wenn ein angebotener Dienst beeinflusst wird. Der Schaden kann aber auch außerhalb des verteilten Systems auftreten, indem Dinge beschädigt oder gar Menschen verletzt werden.

[3] Gefahrendefinition der DIN-Norm 31000: „eine Sachlage, bei der das Risiko größer als das Grenzrisiko ist.“ [DIN87]

Beispiel 3

Eine Fehlkonfiguration liegt beispielsweise dann vor, wenn eine Person regelwidrig einer Benutzergruppe zugeordnet ist. Ist diese Person aufgrund dieser Zuordnung in der Lage, auf für sie nicht bestimmte Informationen zuzugreifen, liegt eine Gefahr vor. Wenn diese Person auf die für sie nicht bestimmte Informationen tatsächlich zugreift, liegt ein Schadensfall vor.

In Abbildung 2.7 ist eine Taxonomie von Konfigurationen dargestellt. Eine Konfigurationen kann eine Fehlkonfiguration oder eine korrekte Konfiguration sein. Von einer Fehlkonfiguration kann wiederum eine Gefahr oder keine Gefahr ausgehen. Eine Gefahr kann zu einem Zeitpunkt noch zu keinem Schaden geführt haben oder bereits einen Schadensfall ausgelöst haben.

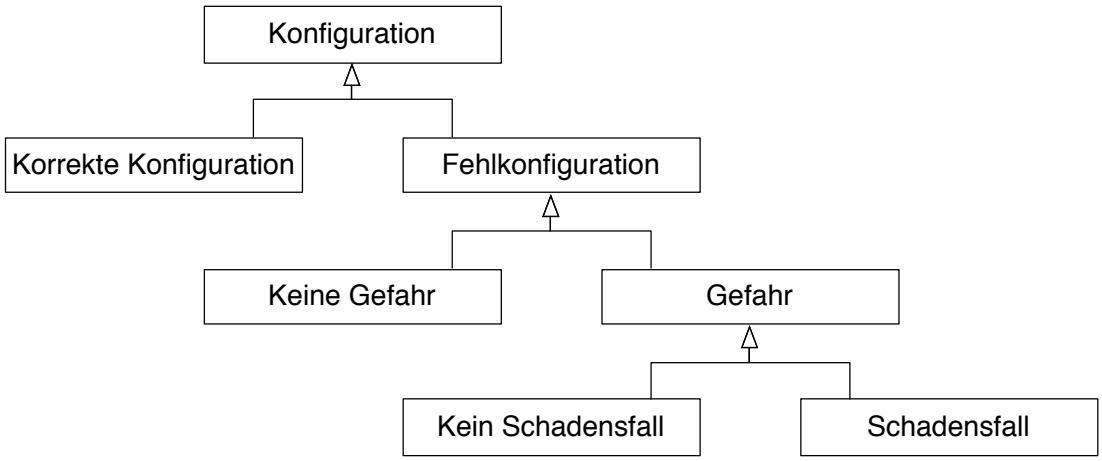


Abb. 2.7: *Taxonomie von Konfigurationen*

Fehlkonfigurationen können durch Konfigurationsveränderungen, aber auch durch weitere Einflüsse entstehen. Eine gleichbleibende Konfiguration kann nach einer Regeländerung ein regelwidriges Verhalten des verteilten Systems ermöglichen. Auch das Hinzufügen und Entfernen von Komponenten kann zu Fehlkonfigurationen führen. Bleiben beispielsweise bei der Abschaltung eines Systems die Firewallregeln bestehen, kann bei der Wiedervergabe der Netzwerkadressen eine Gefahr entstehen. Oder verlässt eine Person das Unternehmen, werden zuvor unbedenkliche Rechte ebenfalls zur Gefahr. Auch die Zeit kann sich degradierend auf den Zustand der Konfiguration auswirken. Wenn beispielsweise Berechtigungen in bestimmten Zeitin-

tervallen bestätigt werden müssen, stellt ein Recht außerhalb des Intervalls einen unbeabsichtigten Zustand dar.

2.4.1 Gefahrenklasse

Eine Gefahrenklasse ist eine Gruppe von Gefahren, die sich durch ihr Zustandekommen oder ihre Auswirkung von anderen Gefahren abgrenzt. Die Gefahrenklasse „Rollenkonflikt“ beschreibt beispielsweise die Zusammenfassung aller Berechtigungen bei denen der Berechtigende und der Berechtigte identisch sind. So ist eine Berechtigung auf das System X bei der Person A sowohl berechtigt hat als auch berechtigt ist eine Instanz dieser Gefahrenklasse.

Für jedes verteilte System existiert eine Menge von Gefahrenklassen. Da Gefahrenklassen unterschiedlich abstrakt definiert werden können, kann ein und die selbe Gefahr mehreren Gefahrenklassen unterschiedlicher Abstraktionsebenen zugeordnet werden. Durch eine hierarchische Zuordnung der Gefahrenklassen nach ihrer Abstraktionsebene entsteht eine Taxonomie wie sie exemplarisch in Abbildung 2.8 beispielhaft ist.

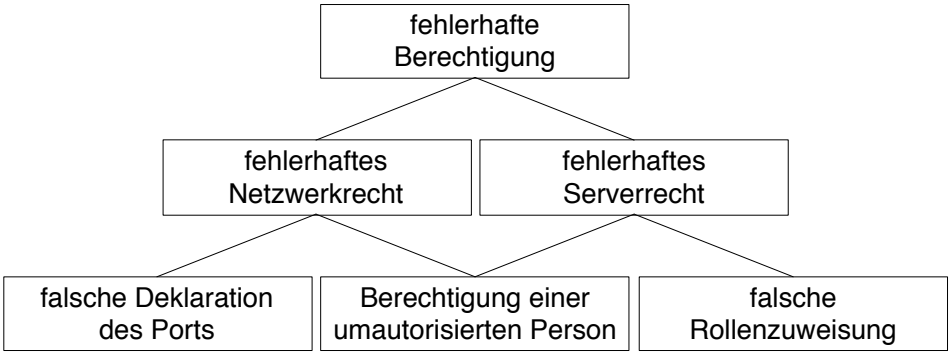


Abb. 2.8: *Exemplarische Taxonomie der Gefahrenklassen eines verteilten Systems*

Beispiel 4

Eine für Rechenzentren relevante Gefahrenklasse sind falsche Berechtigungen von Personen auf Systemen. Ein Beispiel dieser Gefahrenklasse ist: Max Mustermann besitzt auf dem Personalsystem PS1 irrtümlich Administrationsrechte.
Diese Gefahrenklasse ist für verteilte Systeme ohne personenbezogenen Berechtigungsmöglichkeiten irrelevant.

Für die Einordnung von Gefahrenklassen bestehen bereits abstrakte Taxonomien, deren Taxa die Gefahrenklassen von verteilten Systemen zugeordnet werden können. Li et. al. schlagen zur Klassifizierung von Gefahren das STRIDE-Modell [JCDZ11] vor, wobei STRIDE als Akronym für die sechs Gefahrenquellen „Spoofing, Tampering, Repudiation, Message Disclosure, Denial of Service and Elevation of Privilege“ steht. Jede Gefahr kann im STRIDE-Modell einer oder mehreren Gefahrenquellen zugeordnet werden. Durch die konkrete Zuordnung von Gefahren nach dem STRIDE-Modell kann für jedes System eine Gefahrentaxonomie entwickelt werden. Babar et. al. haben eine Taxonomie, welche die Gefahren im Umfeld des Internet of Things (IoT) klassifiziert [BMS+10]. Sie definieren als schutzbedürftigen Güter Sicherheit, Datenschutz und Vertrauen. Alle Gefährdungen dieser Güter seien auf Schwachstellen in den sechs Gefahrenklassen Identification, Communication, Physical threat, Embedded Security, Storage management, Dynamic Binding zurückzuführen. Diese abstrakten Definitionen erlauben eine taxonomische Einordnung von Gefahren.

2.4.2 Sicherheitstest

Im Softwareengineering ist der Begriff Testfall folgendermaßen definiert: „Ein Testfall besteht aus einer Reihe von Eingabedaten und erwarteten Ergebnissen, mit denen eine Komponente geprüft wird, um einen Ausfall auszulösen und Mängel aufzudecken“ [BD04]. Analog zur Verwendung der Terminologie der Testfälle im Softwareengineering verwenden wir bei der Gefahrenerkennung die Terminologie der Testfälle, die in dieser Dissertation Sicherheitstests heißen. Dieser Begriff lässt sich auf das Testen von Konfigurationen übertragen: „Ein Sicherheitstest besteht aus einer Reihe von Konfigurationsartefakten und erwarteten Ergebnissen, mit denen eine Konfiguration geprüft wird, um eine Fehlkonfiguration aufzudecken“

Sicherheitstests dienen der Erkennung von Gefahren einer Gefahrenklasse in einem verteilten System. Die alleinige Kenntnis einer Gefahrenklasse ist nicht hinreichend, um deren Gefahren identifizieren zu können. Die Identifikation setzt eine definierte Methode voraus, die Mittels geeigneter Identifikationsmerkmale die Gefahren einer Gefahrenklasse erkennen kann. Sicherheitstests dürfen auf heuristischen Ansätzen basieren [Bin96]. Eine Heuristik muss nicht jede Gefahr einer Gefahrenklasse erkennen und kann auch Gefahren erkennen, die tatsächlich keine sind. Fehldiagnosen resultie-

ren entweder aus α- oder aus β-Fehler und sind in Abschnitt 7.3 detailliert beschrieben.

Für ein und die selbe Gefahrenklasse können unterschiedliche Sicherheitstests existieren. Diese können zu verschiedenen Ergebnissen führen. Beispiel 5 veranschaulicht das anhand zweier für die gleiche Gefahrenklasse definierten Sicherheitstests.

Beispiel 5

Sei `faule Äpfel` eine Gefahrenklasse. Mit dem Muster `brauner Fleck` kann man `faule Äpfel` auf der Basis einer optischen Analyse (zumindest einen Teil) erkennen. Ebenso können `faule Äpfel` auch mit dem Muster `weiche Stelle`, basierend auf einer haptischen Analyse, erkannt werden.

Die Gefahren, die auf ein System einwirken, sind unterschiedlich. Solange man nicht alle Gefahren kennt, ist es auch nicht möglich, eine vollständige Taxonomie über alle denkbaren Gefahren zu definieren. Dennoch ist es denkbar eine Struktur für Taxonomien zu Gefahren eines beliebigen Systems vorzugeben. Die hier vorgestellte Struktur ist eine Taxonomie über die verschiedenen Abstraktionsebenen zur Gruppierung von Gefahren, also ein Metamodell für Gefahren (siehe Tabelle 2.9).

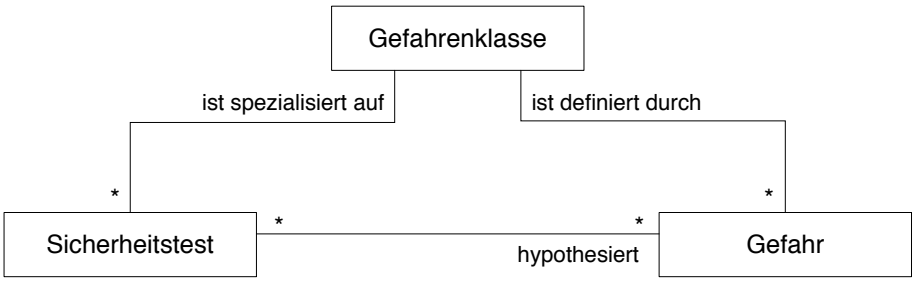
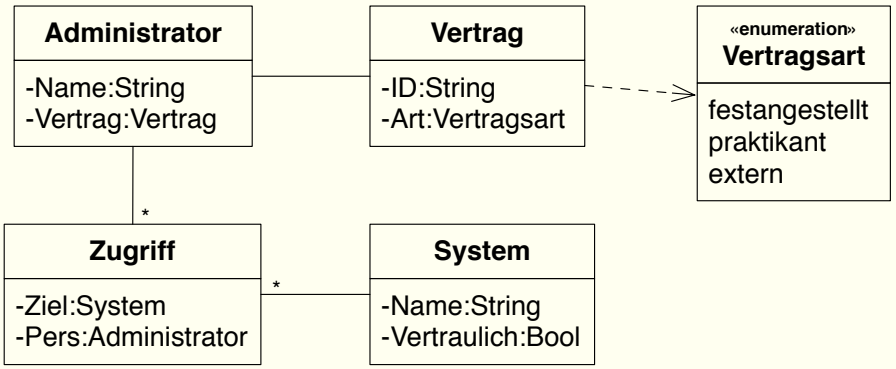


Abb. 2.9: *Der Zusammenhang zwischen Gefahrenklasse, Sicherheitstest und Gefahr*

Diese Abstraktionsebenen sind wichtige Elemente der Terminologie dieser Dissertation. Die Menge aller Gefahren kann in Gefahrenklassen unterteilt werden. Einige Bespiele von Gefahrenklassen sind in Tabelle 3.3 beschrieben. Informationen zu Ursprung, Verursacher und Intention sind für die Beschreibung einer Gefahrenklasse im Gegensatz zur Konsequenz nicht essentiell. Bei einem Sicherheitstest werden, im Gegensatz zur Gefahrenklasse, konkrete, auf das System bezogene, Einschränkungen getroffen (siehe Beispiel 6).

Beispiel 6

Die Gefahrenklasse `GK:NAA` „Ein nicht akkreditierter Administrator hat Zugriff zu einem sensiblen System.“ kann mit mehreren, sich ergänzenden Sicherheitstests geprüft werden. Die Sicherheitstests „Ein Externer kann auf ein sensibles System zugreifen.“ und „Ein Praktikant kann auf ein sensibles System zugreifen.“ entdecken verschiedene Gefahren der Gefahrenklasse.



2.4.3 Testobjekt, Testmenge und Testdaten

Ein Sicherheitstest testet eine Konfiguration Unter Test (KUT) (siehe Abschnitt 2.4.4) auf Gefahren einer Gefahrenklasse. Hierfür untersucht ein Sicherheitstest eine Menge von Objekten, die Testmenge, auf das Vorliegen von Gefahren der Gefahrenklasse. Jedes Objekt der Testmenge ist ein Testobjekt (siehe Abbildung 2.10). Alle Konfigurationsdaten, die für die Analyse benötigt, werden sind die Testdaten.

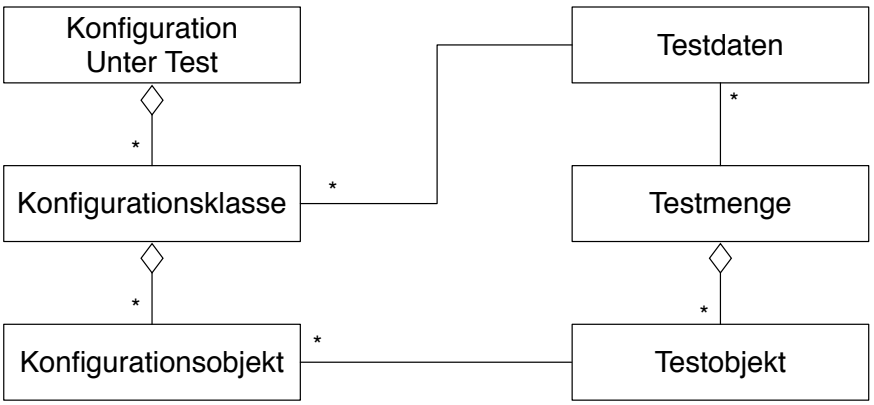


Abb. 2.10: *Die Erhebung von Testmengen in verteilten Systemen*

Bei der Gefahrenklasse `GK:NAA` des Beispiels 6 werden alle Zugriffe getestet. Jeder Zugriff ist ein Testobjekt und die Menge der Zugriffe ergibt die Testmenge. Die Testdaten sind alle Daten, die für die Analyse benötigt werden. Im Fall des Beispiels 6 bestehen die Testdaten aus den Konfigurationsklassen (siehe Abschnitt 2.3) System, Zugriff, Administrator, Vertrag und Vertragsart.

Sicherheitstests testen eine Konfiguration (in Beispiel 7 die Firewallregeln) auf Gefahren einer bestimmten Gefahrenklasse. Für diese Tests können mehrere Bereiche der Konfiguration Unter Test (KUT) benötigt werden (siehe Beispiel 7). Die Gefahrenerkennung einer konkreten Gefahrenklasse benötigt daher nicht zwangsläufig die gesamte Konfiguration unter Test.

Beispiel 7

Eine KUT bestehe aus den Konfigurationsklassen Firewallregel, Benutzerstammdaten und Zugriffsrecht. Sei die Gefahrenklasse „Illegale Firewallregel“ so definiert dass eine Person aufgrund einer Firewallregel über das Netzwerk auf einen Rechner zugreifen kann, auf dem sie kein Zugriffsrecht hat. Um alle illegalen Firewallregeln zu bestimmen, sind zwei Bestandteile der KUT notwendig: die Firewallregeln und die Zugriffsrechte. Die Menge der Firewallregeln wird auf ihre Korrektheit überprüft und ist somit die Testmenge. Die Testobjekte sind somit vom Typ Firewallregel. Für jede Firewallregel wird in der Menge der Zugriffsrechte ein äquivalentes Zugriffsrecht gesucht. Wenn es kein äquivalentes Zugriffsrecht unter der Menge der Zugriffsrechte gibt liegt keine Gefahr im Sinne der Gefahrenklasse „Illegale Firewallregel“ vor.

Im Bezug auf die Testmenge bestehen bei der Gefahrenerkennung drei Aufgaben. Die Testmenge muss zuerst erhoben und die gefährlichen Testobjekte, also die Gefahren, identifiziert werden. Daraufhin müssen die Gefahren in einer geeigneten Form kommuniziert und visualisiert werden.

2.4.4 Konfiguration Unter Test

Im Bereich der Softwaretests hat sich der Begriff System Under Test (SUT) etabliert. SUT beschreibt das System, dessen Verhalten mithilfe von Tests auf die Erfüllung der Anforderungen getestet wird [GEMT06]. Ein verteiltes System, das auf die Erfüllung der Sicherheitsanforderungen überprüft wird, ist ein SUT. Abbildung 2.11 zeigt ein verteiltes System als eine Komposition von Komponenten. Jede Komponente setzt sich aus einer Hardware, einer Software und einer Konfiguration zusammen.

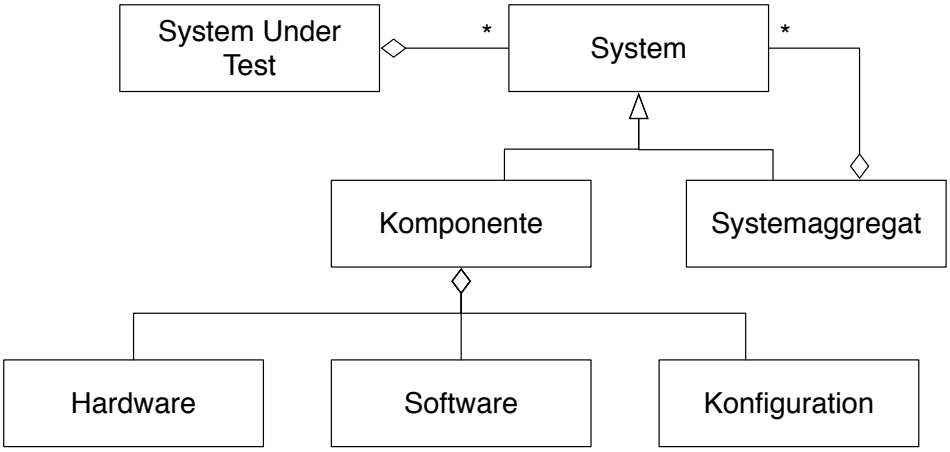


Abb. 2.11: *Komposition eines verteilten Systems als System Under Test*

Die Konfiguration eines verteilten Systems bezeichnen wir als KUT. Sie steht im Fokus der Betrachtungen dieser Dissertation. Eine KUT ist das Aggregat der Konfigurationen aller Komponenten eines verteilten Systems (siehe Abbildung 2.12). Denn die Konfiguration Under Test wird mithilfe von Tests auf die Erfüllung der Sicherheitsanforderungen überprüft.

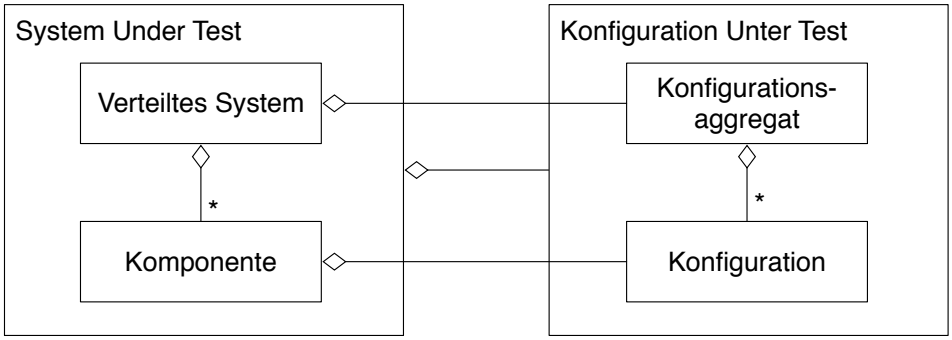


Abb. 2.12: *Die Konfiguration Unter Test als Teil des System Under Test*

2.4.5 Transformation von Konfigurationen zur Analyse

Jede Konfiguration ist eine Aggregation von Informationen, die in einer bestimmten Struktur vorliegt. Die Struktur, in der man die Konfiguration einer Komponente exportieren kann, ist durch die Komponente festgelegt. Konfigurationsstrukturen sind häufig für bestimmte Anforderungen wie z.B. Performance der Komponente oder Wartbarkeit (siehe Beispiel 8) optimiert.

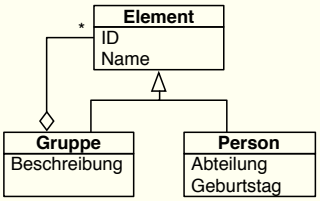
Beispiel 8

Firewalls müssen für Anfragen schnell Entscheidungen treffen können. Hierfür müssen sie die Firewallregeln auswerten, die durch die Konfiguration vorgegeben sind. Die Konfigurationsstrukturen mancher Firewalls ist speziell auf die **schnelle Beantwortung von Anfragen optimiert**. Die Regeln werden in flachen Strukturen abgelegt und können dadurch mit einem einzelnen Zugriff vollständig abgefragt werden. Benutzerverzeichnisse müssen viele Gruppenmitgliedschaften verwalten. Zur **Vereinfachung der Wartbarkeit** können Gruppen in einer hierarchischen Struktur angeordnet werden. Dadurch wird die Zuweisung einer bestimmten Menge an Mitgliedschaften für alle Inhaber einer bestimmten Rolle ermöglicht.

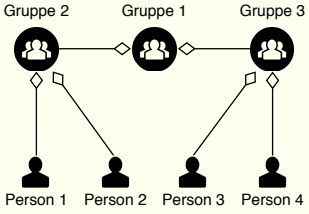
Die Struktur einer Konfiguration ist auch für die Gefahrenanalyse von Bedeutung. So ist es dienlich, wenn die gewünschten gefahrenrelevanten Aspekte direkt erkennbar sind und irrelevante Aspekte ausgeblendet werden (siehe Beispiel 9). Für die Analyse ungeeignete Strukturen können sich während der Analyse behindernd auswirken. Sie können sich negativ auf die Analyselaufzeit oder -ergebnisse auswirken. Deshalb kann es sinnvoll sein Konfigurationen vor der eigentlichen Analyse in ein geeignetes Format zu überführen.

Beispiel 9

In einem LDAP-Verzeichnis können unter anderem Personen, Gruppen und Gruppenmitgliedschaften definiert werden. Es gibt auch transitive Mitgliedschaften (Person 1 ist transitiv in Gruppe 1, da Person 1 in Gruppe 3 ist und Gruppe 3 in Gruppe 1. Gruppenmitglieder sind entweder andere Gruppen oder Personen:



LDAP-Modell

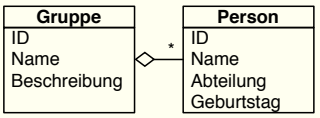


LDAP-Graph

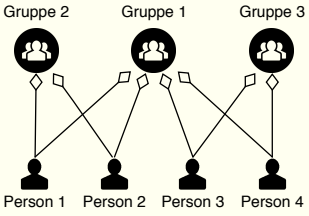
Gruppe	Mitglied
Gruppe 1	Gruppe 2
Gruppe 1	Gruppe 3
Gruppe 2	Person 1
Gruppe 2	Person 2
Gruppe 3	Person 3
Gruppe 3	Person 4

LDAP-Zugriffsliste

Während der Analyse sind nur die (auch transitiven) Mitgliedschaften von Personen in Gruppen wichtig. In der Komponentenkonfiguration ist der relevante Aspekt „Person 1 ist in Gruppe 1“ nicht direkt erkennbar. Hingegen ist der irrelevante Aspekt „Gruppe 2 ist in Gruppe 1“ mit aufgeführt. Es bietet sich an, die gegebenen Struktur des LDAP in folgende Struktur zu transformieren:



Analyse-Modell



Analyse-Graph

Gruppe	Mitglied
Gruppe 1	Person 1
Gruppe 1	Person 2
Gruppe 1	Person 3
Gruppe 1	Person 4
Gruppe 2	Person 1
Gruppe 2	Person 2
Gruppe 3	Person 3
Gruppe 3	Person 4

Analyse-Zugriffsliste

Die Hierarchie wurde rekursiv aufgelöst: Relevante Aspekte werden direkt angezeigt und irrelevante Aspekte sind ausgeblendet.

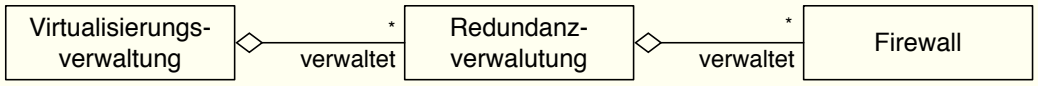
2.4.6 Synchronisation zwischen Komponenten

Manche Komponenten verwalten die für sie notwendigen Konfiguration nicht oder nur teilweise selbst. Anstelle der eigenständigen Verwaltung der

Konfiguration wird dies auf einer weiteren, dafür spezialisierten Komponente vorgenommen. Leblang und Chase beschreiben, wie derartige Abhängigkeiten entstehen [LCJ87]. Eine solche fremdverwaltete Komponente ist in Beispiel 10 beschrieben. Im redundanten Betrieb werden mehrere Instanzen der Firewall im gleichen Zustand in Betrieb genommen. Die Konfiguration wird auf einer externen Komponente (im Beispiel das SmartDashbaord) verwaltet und den einzelnen Instanzen zur Verfügung gestellt. Eine derartige Virtualisierung von Firewalls wird in virtuellen Netzwerken benötigt [CB09]. Der Ansatz, sämtliche Netzwerkkomponenten nur noch virtuell bereitzustellen, führt dazu, dass auch die Firewall und deren Verwaltungssystem virtualisiert sind.

Beispiel 10

Im Falle der CheckPoint Firewall können über die zusätzliche Komponente FireWall 1™ [NT03] ein oder mehrere reelle oder virtuelle Instanzen SmartDashbaord verwaltet werden (siehe Abbildung). Wird eine derartige Firewall redundant virtualisiert so wird die Fremdverwaltung der Firewall fremdverwaltet.



Auch fehlende Funktionalitäten der Konfigurationsschnittstelle können zu einer fremdverwaltete Konfiguration führen. Das war beispielsweise bei einer in einem Rechenzentrum durchgeführten Fallstudie (Abschnitt 8.1) zu beobachten. Die Verwaltung der SSL-VPN Regeln ist dort in ein externes System ausgelagert. So wird es ermöglicht, Workflows zur Beantragung, zur Bestätigung, zur Löschung und zur jährlichen Überprüfung einer Regel vorzugeben. Hierfür werden die Regeln im externen Werkzeug administriert. Bei jeder Änderung der Regeln werden alle Instanzen der SSL-VPN-Gateways mit dem neuen Regelsatz aktualisiert.

Komponenten, die auf externe Konfigurationen zugreifen, sind auf eine Synchronisierung angewiesen. Die Synchronisierung durch einen Push-Mechanismus, einen Pull-Mechanismus oder manuell erfolgen 2.13. Push-Mechanismen werden änderungsbasiert oder zeitbasiert und Pull-Mechanismen werden bedarfsbasiert oder zeitbasiert ausgelöst. Manuelle Synchronisierungen lösen Personen aus.

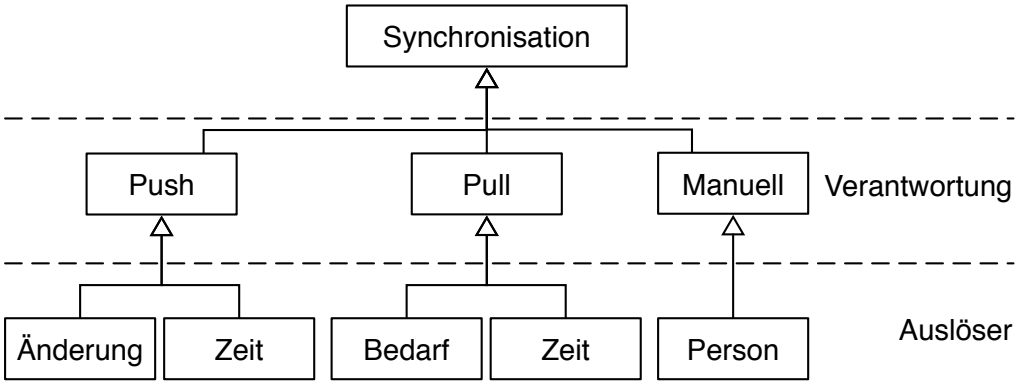


Abb. 2.13: *Eine Taxonomie zur Synchronisation zwischen Komponenten*

Loch et. al. beschreiben, dass menschliche Fehler als die größte Gefahr für Informationssysteme gelten [LCW92]. Whitman kommt in Studien aus den Jahren 2003 [Whi03] und 2010 [WM10] zu dem Schluss, dass menschliche Fehler von Sicherheitsverantwortlichen, nach absichtlich herbeigeführten Attacken von außen und Softwarefehlern, als das drittgrößte Risiko bewertet werden. Daraus erschließt sich, dass bei der manuellen Synchronisation Fehler geschehen, die zu Gefahren führen können. Eine fehlerhafte Synchronisation kann durch den Vergleich der Konfigurationen von Anbieter und Empfänger erkannt werden. Dementsprechend kann auch das Auslesen der Konfiguration des Empfängers sinnvoll sein. Im Falle der bedarfsgesteuerten Auslösung der Synchronisation ist es denkbar, dass der Empfänger die Konfiguration nicht speichert, sondern die Konfigurationen nach dem Empfang und der Verarbeitung wieder verwirft.

2.5 Gefahrenwissen

Wissen, dass Spezialisten bei der Erkennung von Gefahren einsetzen, heißt Gefahrenwissen. Gefahrenwissen ist Wissen über ein verteiltes System, welches zur Erkennung von Gefahren beitragen kann. Gefahrenwissen wird von Kapazitäten bereitgestellt. Es gibt unterschiedliche Gebiete, aus denen Gefahrenwissen entspringt. Abbildung 2.14 beschreibt eine Taxonomie über Gefahrenwissen für die Gefahrenerkennung in verteilten Systemen. Da unterschiedliche verteilte Systeme unterschiedliches Gefahrenwissen für die Gefahrenerkennung voraussetzen, handelt es sich um eine beispielhafte, frei erweiterbare Taxonomie.

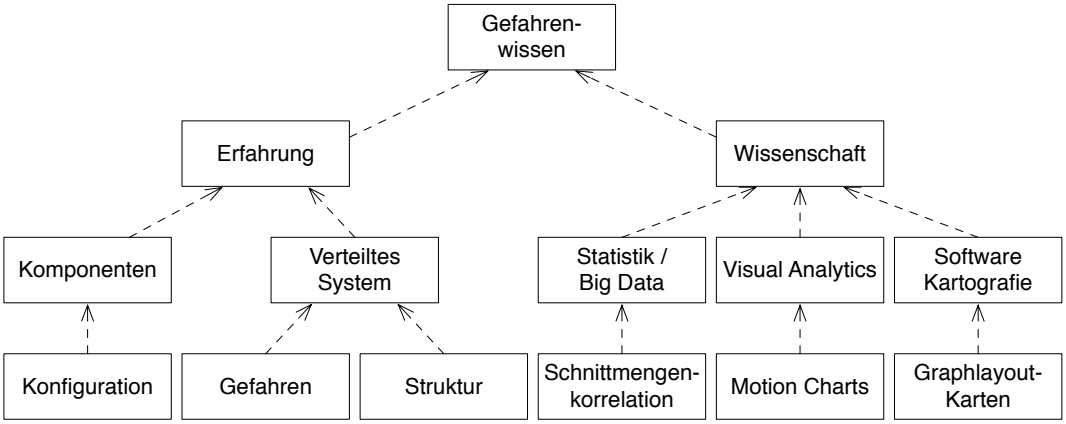


Abb. 2.14: *Erweiterbare Taxonomie über Wissen hinsichtlich der Gefahrenerkennung*

Erfahrungswissen resultiert aus dem praktischen Umgang mit einzelnen Komponenten oder dem verteilten System als Ganzen. Das kann beispielsweise die Fähigkeit sein, die Konfiguration einer Komponente korrekt zu interpretieren oder Wissen zu strukturellen Zusammenhängen und Gefahren zu dem verteilten System sein. Auch die Wissenschaft trägt Erkenntnisse für die Gefahrenerkennung bei. Statistische Methoden spielen bei der Interpretation von Daten eine entscheidende Rolle. Ott und Longnecker widmeten dieser Thematik ein ganzes Buch [OL10]. Diese Methoden sind im wichtigen Forschungsbereich Big Data [CCS12], dem Verstehen von großen, oft unstrukturierten Datenmengen [ZE+11], eine Grundvoraussetzung. Motion-Charts [ZE08], eine Applikation der Visual Analytics [TC+06], helfen bei der visuellen Darstellung von Veränderungen, die sich während eines Zeitraums ergeben haben. Und Graphenlayout-Karten erlauben zusammenhängende Informationen in einer geeigneten Struktur verständlich zu präsentieren [Mat08, KELN10].

2.6 Wissensgebiet

Verteilte Systeme sind, wie bereits in Kapitel 1 beschrieben, oft heterogen. Sie bestehen aus verschiedenartigen Komponenten und sind mehreren unterschiedlichen Gefahren ausgesetzt. Dadurch ergeben sich mehrere Bereiche, in denen Gefahrenwissen über das verteilte System und dessen Gefahren aufgebaut werden kann. Ein Wissensgebiet ist ein Bereich eines verteil-

ten Systems, für den Gefahrenwissen benötigt wird, um das verteilte System auf Gefahren zu prüfen.

Beispiel 11
Ein verteiltes System besteht aus Firewalls und Servern. Für den Betrieb muss es den Anforderungen der Sarbanes-Oxley Act (SOA) genügen. Die Wissensgebiete des verteilten Systems sind in diese Beispiel Firewalls, Server und SOAs.

2.7 Kapazitäten

Der Terminus der Kapazität ist die gemeinsame Abstraktion von Spezialisten und Experten. Eine Kapazität stellt bei der Gefahrenerkennung in verteilten Systemen Gefahrenwissen aktiv zur Verfügung. Ein Spezialist ist eine Person mit Gefahrenwissen. Ein Experte ist ein Programm, welches das Gefahrenwissen automatisiert zur Verfügung stellen kann.

Für die Rolle des Spezialisten kommt, abhängig von dem beigesteuerten Gefahrenwissen, ein heterogener Personenkreis in Frage. Es ist naheliegend, dass die Administratoren und Verantwortlichen des verteilten Systems selbst substantielle Erfahrungen zur Struktur und damit auch zu den Gefahrenklassen des verteilten Systems beitragen können. Für Komponentenwissen kommen Spezialisten, wie zum Beispiel Mitarbeiter des Herstellers in Frage. Nicht nur Wissenschaftler, sondern auch spezialisierte Berater können Gefahrenwissen aus dem wissenschaftlichen Bereich beisteuern. Experten existieren ohne die Einführung des Sicherheitsframeworks im Regelfall nicht. Eine Bedienungsanleitung einer Komponente ist keine Kapazität. Diese kann zwar Gefahrenwissen vermitteln, steuert diese aber nicht aktiv bei der Gefahrenerkennung bei.

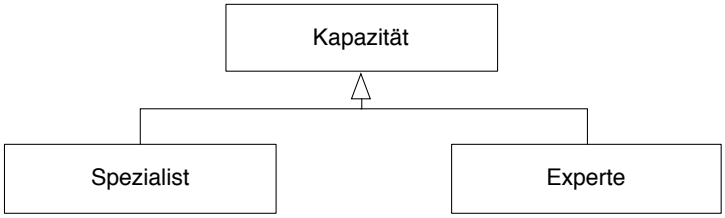


Abb. 2.15: *Taxonomie über Kapazitäten*

Kapazitäten sind somit entweder Spezialisten (Menschen) oder Experten (Programme), die Gefahrenwissen bereitstellen (siehe Abbildung 2.15). Abbildung 2.16 vereinigt die Taxonomie des Gefahrenwissens (siehe Abbildung 2.14) und die der Kapazitäten zu einer Taxonomie mit Mehrfachvererbung. Diese vereinte Taxonomie verdeutlicht, dass Gefahrenwissen verschiedener Wissensgebiete sowohl von Spezialisten als auch von Experten bereitgestellt werden kann.

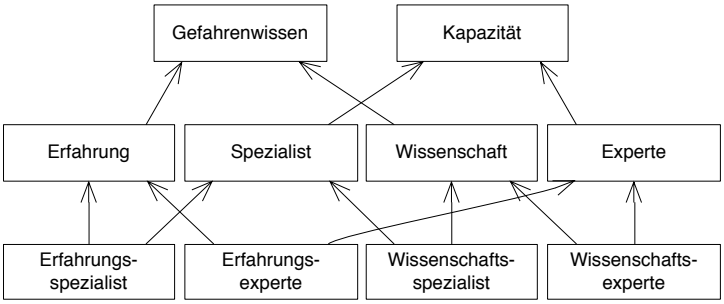


Abb. 2.16: *Eine Taxonomie über Gefahrenwissen und Kapazitäten*

In Beispiel 11 ist ein verteiltes System beschrieben. Die Gefahrenklasse „Illegale Firewallregel“ aus Beispiel 7 existiert in diesem verteilten System. Abbildung 2.17 ist ein Modell, welches die Zusammenhänge zwischen den Komponenten, den Spezialisten und Gefahrenwissen des verteilten Systems darstellt. In dem Modell ist zu erkennen, dass Spezialisten aufeinander aufbauen können. Der Auditor, der sich mit den Wissensgebieten Firewall und Server nicht auskennt, kann trotzdem auf das Gefahrenwissen zugreifen und so die illegalen Firewallregeln ermitteln.

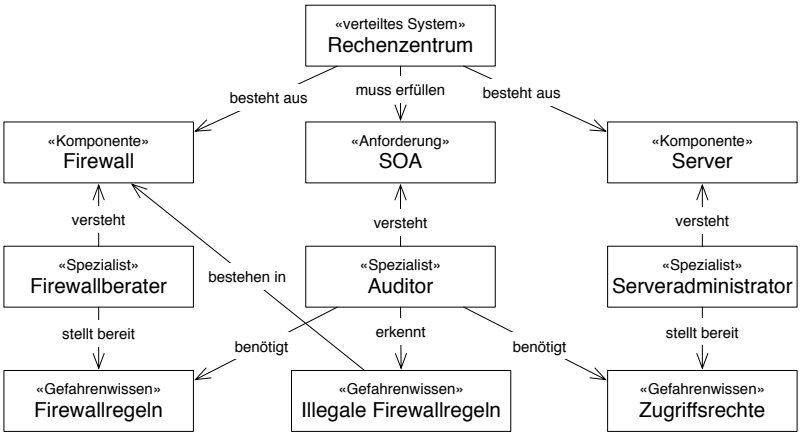


Abb. 2.17: *Ein Modell der Komponenten, Spezialisten und des Gefahrenwissens einer Gefahrenklasse*

3 Die Erkennung gefährlicher Konfigurationen

> *The task of keeping the communication system operable is highly critical due to the configuration complexity and the need for manual administration*

Klenk, Niedermayer, Masekowsky und Carle [KNMC06]

Dieses Kapitel beschreibt, welche Gefahren von fehlerhaften Konfigurationen ausgehen und wie diese erkannt werden können. Die verschiedenen Auswirkungen, die fehlerhafte Konfigurationen auf Systeme im Allgemeinen und verteilte Systeme im Speziellen haben können, werden in Abschnitt 3 beschrieben. Anschließend wird dargelegt, wie Gefahren erkannt werden können. Dies schließt das Wissen über Gefahren in Abschnitt 3.4 sowie die Methoden der Gefahrenerkennung in 3.5 ein.

Mit der Komplexität eines Systems steigt auch dessen Fehleranfälligkeit. Sha vertritt die These, dass die Anzahl der Fehler in einer Software in proportionaler Abhängigkeit zu ihrer Komplexität steht [Sha01]. Eine steigende Komplexität einer Software führt nach Sha zu mehr Fehlern bzw. einer höheren Wahrscheinlichkeit von Fehlern. Zwischen Software und Konfigurationen verteilter Systeme lassen sich einige Parallelen beobachten. Beide beeinflussen das Verhalten eines Systems, werden in einer bestimmten Sprache entwickelt und dann interpretiert, können umfangreich sein und werden von Spezialisten entwickelt. Analog zu Software birgt auch die Konfiguration eines (verteilten) Systems, ab einer gewissen Komplexität, fehlerhafte Konfigurationen [HKD07]. Häberlen et. al. beschreiben das wie folgt:

> *At large scale, it is increasingly likely that some nodes [of the distributed systems] are accidentally misconfigured ...*

Haeberlen, Kouznetsov und Druschel [HKD07]

Gabriel und Northrop et. al. definieren die Begrifflichkeit des Ultra-Large-Scale System (ULSS) [GNSS06, NFG+06]. Ein ULSS ist ein besonders umfangreiches verteiltes System. Sie beschreiben die ULSS's aufgrund von heterogenen, sich ändernden Komponenten, inkonsistente Elemente als fehleranfällig. Fehlerhafte Konfigurationen seien eher die Norm als die Ausnahme, und ein gleichzeitiges korrektes Funktionieren aller Komponenten wäre sehr unwahrscheinlich. Auch Whitman beschreibt die Häufigkeit versehentlicher Fehlkonfigurationen und zeigt, dass Fehlkonfigurationen eine ernstzunehmende Gefährdung für Systeme darstellt [Whi03] (siehe Tabelle 3.1). Die häufigsten Arten von Fehlkonfigurationen ermittelte Wool am Beispiel von Firewalls [Woo04].

3.1 Gefahren für Informationssysteme

Gefahren spielen bei der Verwaltung eines Informationssystems eine wichtige Rolle. Ein Großteil der Verantwortlichen geht davon aus, dass ihre Organisation aufgrund von Fehlern in den Informationssysteme gefährdet ist. Das zeigt eine Studie aus dem Jahre 92 [LCW92] bei der 131 Manager von Informationssystemen befragt wurden. Die Manager wurden um eine Einschätzung der existierenden Gefährdung ihrer Organisation durch Störungen der IT gebeten. Der Durchschnittswert von 3,7 von 7 möglichen Punkten belegt dass derartige Gefahren durchaus ernst genommen wird. Ein bemerkenswertes Ergebnis der Studie ist, dass fast zwei Drittel des Gefahrenpotentials von Organisationsinternen Personen ausgeht (siehe Tabelle 3.1).

Gefahren		Verursacher	
		Mensch	kein Mensch
Quelle	Intern	62,4 %	0 %
	Extern	9,4 %	27,6 %

Tab. 3.1: *Experteneinschätzung von Gefahren nach Quelle und Verursacher*

In den Jahren 2003 und 2010 wurden von Whitman ähnliche aufgebaute Studien [Whi03, WM10] durchgeführt. Sie kommen zu dem Ergebnis, dass die Gefahr für Organisationen durch Störungen der IT-Systeme immer noch als bedeutend gilt. Greg et. al. zeigen in ihrem Report aus 2003 auf, dass der

Eintritt von Gefährdungen Schäden von bis zu 80 Millionen US-Dollar verursachten [GCH03]. Eine Studie im Auftrag des FBI [GLLR05] kommt zu dem Ergebnis, dass im Jahr 2005 etwa 55 Prozent der Firmen unautorisierten Zugriff auf Ihre Systeme zu verzeichnen hatten. Die bis 2004 rückgängigen Zahlen seien 2005 wieder gestiegen. Der unautorisierten Zugriff von außerhalb der Organisation hielt sich mit dem von innerhalb etwa die Waage. 13 Prozent der Firmen waren nicht in der Lage zu bestimmen, ob sie in den letzten 12 Monaten Opfer von unautorisierten Zugriffen wurden.

Eine große Vielfalt unterschiedlicher Gefahren können auf ein Informationssystem einwirken. Loch et. al. beschreiben in ihrer Studie [LCW92] abstrakt die als am bedeutendsten wahrgenommenen Gefahren. Um diese einzuordnen, führen sie vier verschiedene Kriterien, die sogenannten Dimensionen der Sicherheit von Informationssystemen, ein: Ursprung, Verursacher, Intension und Konsequenz (siehe Tabelle 3.2).

Dimension	**Ausprägungen**	
Ursprung (Ur)	intern (I)	extern (E)
Verursacher (Ver)	Mensch (M)	kein Mensch (!m)
Intension (Int)	Absicht (A)	keine Absicht (!a)
Konsequenz (Kon)	Offenlegung (O)	Modifikation (M)
	Zerstörung (Z)	Beschränkung (B)

Tab. 3.2: *Ausprägungen der Sicherheitsdimensionen nach Loch et. al. [LCW92]*

In der Dimension Ursprung wird zwischen internen und externen Gefahren unterschieden. Intern Gefahren gehen von unternehmenszugehörigen und externe Gefahren gehen von unternehmensfremden Akteuren aus. Der Verursacher kann, muss aber kein Mensch sein. Bei der Intension wird unterschieden, ob die Gefahr beabsichtigt oder unbeabsichtigt herbeigeführt wurde. Letztlich unterscheidet die Konsequenz, ob die Daten in einer schädlichen Form offengelegt, modifiziert, zerstört oder zugriffsbeschränkt wurden. Die Tabelle 3.3 veranschaulicht die Kategorisierung in den vier Dimensionen anhand von Beispielen.

Beispiele	**Ur**	**Ver**	**Int**	**Kon**
Ein Computervirus spioniert Daten aus.	E	!m	A	O
Bewusstes Löschen von Daten durch einen Hacker.	E	M	A	Z
Ein Angestellter löscht Daten versehentlich.	I	M	!a	Z
Ein Administrator berechtigt eine falsche Person.	I	M	!a	O
Ein Administrator kategorisiert ein System falsch .	I	M	!a	O
Ein Angestellter ändert Daten bewusst.	I	M	A	M
Ein Sturm beschädigt das Rechenzentrum.	E	!m	!a	B
Eine Fehler führt zur Veröffentlichung interner Daten.	I	!m	!a	O
Unverschlüsselte Kommunikation wird belauscht.	I	!m	!a	O

Tab. 3.3: *Kategorisierung von Gefahren nach den Loch'schen Sicherheitsdimensionen*

3.2 Modellierung von Gefahren

Zwei Philosophien polarisieren die bestehenden Modellierungsansätze von Gefahren. Die Positivansätze basieren auf der Modellierung der erlaubten, ungefährlichen Use-Cases eines Systems. Die Negativansätze hingegen fokussieren auf die Anti-Use-Cases eines Systems, also die Gefahren, die nicht eintreten dürfen.

Lodderstedt et. al beschreiben mit SecureUML einen Positivansatz [LBD02]. SecureUML fußt auf dem RBAC Ansatz [SCFY96] (mehr zu RBAC in Abschnitt A.1.2 des Appendix) und erlaubt es, mit Unified Modeling Language (UML) und der Erweiterung Object Constraint Language (OCL) sichere Systeme zu modellieren. Diese Modelle können daraufhin in Code überführt werden, und die mit Hilfe von OCL definierten Sicherheitsregeln werden mit Hilfe von Enterprise JavaBeans [BMH06] generiert. Dieser Ansatz ähnelt dem der Aspektorientieren Programmierung [KLM$^+$97], bei dem der Aspekt Sicherheit isoliert entwickelt wird.

Ponder ist eine, eigens für die Definition von Sicherheit geschaffene, objektorientierte Programmiersprache für verteilte Systeme [SL02]. Der Positivansatz von Ponder ermöglicht es das Autorisierungskonzept eines verteilten Systems über dessen Komponentengrenzen hinaus zu deklarieren.

Jürjens stellt mit UMLSec einen weiteren Positivansatz von [Jür02]. UMLSec ist eine Erweiterung von UML und erlaubt es ebenfalls sichere Systeme zu

modellieren. Hierfür führt Jürjens eine Menge von Stereotypen ein, mit deren Hilfe sicherheitsrelevante Aspekte in UML modelliert werden können.

Ein weiterer Positivansatz ist das Access Controll Class Diagram (ACCD), ebenfalls eine Erweiterung von UML [Wag14]. Es erweitert die UML um die Möglichkeit, die Autorisierung auf ein Objekt zu definieren. ACCD modelliert, wie sich diese Autorisierung auf andere, von diesen Objekt abhängige Objekte transitiv auswirkt.

Sindre und Opdahl beschreiben wie Anti-Use-Cases („misuse cases“), für die Erhebung von Sicherheitsanforderungen eingesetzt werden können [SO05]. Sommestad stellt mit CySeMoL einen Negativansatz vor, der ein Metamodell für IT-Infrastrukturen von Organisationen bereitstellt [SEH13]. CySeMoL ermöglicht die Analyse, welche Auswirkungen die Schwachstellen eines Systems auf andere Systeme haben können. Mit der Hilfe von CySeMoL zeigt Svensson die Zusammenhänge historischer Angriffe auf IT-Infrastrukturen auf [Sve15].

Hussein und Zulkernine stellen mit den UML-Profil UMLintr einen weiteren Negativansatz zur Modellierung von Gefahren vor [HZ06]. UMLintr fokussiert auf die Spezifikation verschiedener Arten von Angriffen und stellt damit eine Grundlage für Intrusion Detection Systems (IDS’s) dar [Sma88].

Dem in dieser Dissertation vorgestellte System zur Gefahrenerkennung liegt der Negativansatz zugrunde. Während das Sicherheitsframework CUSTODIAN (siehe Kapitel 6) für die Erkennung von Gefahren bestimmt ist, fokussiert das UML-Profil MTF (siehe Abschnitt 5) auf die Definition von Gefahrenklassen.

3.3 Die Gefahrenpyramide

Die Konfiguration eines verteilten Systems kann in der Form von Exporten der einzelnen Komponenten extrahiert werden. Zwischen diesen Konfigurationsexporten und den Gefahren besteht ein Zusammenhang, wie er bereits abstrakt in existierenden Modellen beschrieben wird. Verschiedenen Ansätze modellieren Daten, Informationen und Wissen in Form einer Wissenspyramide. Die Daten bilden hierbei die Grundlage für die Informationen, welche ihrerseits die Grundlage für das Wissen bilden. Abhängig von der Betrach-

tungsweise kursieren verschiedene Erweiterungen dieses „Basismodells“. Nürnberger und Wenzel [NW11] erweitern das Modell in ihrem Ansatz um eine neue Qualität, die Weisheit, welche sich aus dem Wissen bildet. Mit dem Modell von Bodendorf [Bod06] lässt sich der Zusammenhang zwischen Konfigurationsexporten und den Gefahren gut veranschaulichen.

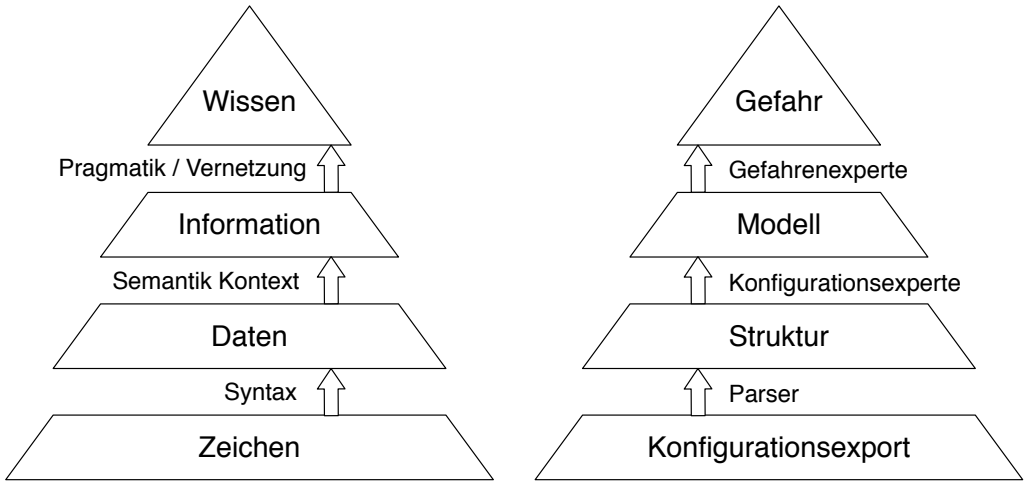


Abb. 3.1: *Äquivalenzen der Bodenhofschen Wissenspyramide [Bod06] und der Gefahrenpyramide*

Bodendorf beschreibt drei Prozesse durch, die Transformationen zwischen den verschiedenen Ebenen der Pyramide ermöglicht werden. In jedem dieser Prozesse wird mit Hilfe von Metawissen die Transformation einer Ebene in die nächsthöhere Ebene erreicht. Zeichen können mit dem entsprechenden syntaktischen Metawissen in Daten transformiert werden, Daten mit semantischen Metawissen in Informationen und Informationen mit Vernetzungsmetawissen und Pragmatik in Wissen (siehe Abbildung 3.1). Das Modell hat in Anwendung auf Konfigurationsexporte folgenden Bezug: Konfigurationsexporte werden durch Parser in Strukturen überführt. Das können beispielsweise Listen oder XML- beziehungsweise JSON-Bäume sein. Strukturen werden durch Gefahrenwissen zu Komponenten in ein Modell überführt, welches die Konfiguration des verteilten Systems beschreibt. In diesem Modell können mit Hilfe von Gefahrenwissen zu Gefahrenklassen existierende Gefahren im verteilten System identifiziert werden. Die Begriffe des Spezialisten, des Experten und des Gefahrenwissens wurden in Abschnitt 2.7 definiert. Bei der Erkennung von gefährlicher Konfigurationen in verteilten Systemen spielen die Konfigurationen selbst aber auch Gefahrenwissen zu den Komponenten und den Gefahrenklassen eine Rolle.

3.4 Wissen und Metawissen über Gefahren

„ Es gibt vier Arten von Männern:
- Diejenigen, die wissen und wissen, dass sie wissen ...
Sein Pferd der Weisheit wird den Himmel erreichen.
- Diejenigen, die wissen, aber nicht wissen, dass sie wissen ...
Er schläft schnell ein, so dass man ihn aufwecken muss.
- Diejenigen, die nicht wissen, aber wissen, dass sie nichts wissen ...
Sein lahmendes Maultier wird ihn schließlich nach Hause bringen.
- Diejenigen, die nicht wissen, und nicht wissen, dass sie nichts wissen ...
Er wird auf ewig in seiner Vergessenheit verloren sein. “

Ibn Yamin, Poet des 13. Jahrhunderts

Die Kenntnis von Gefahren entspricht Gefahrenwissen, und dieses Gefahrenwissen ist eine wichtige Grundlage zum Eliminieren der Gefahr. Um bisher unbekannte Gefahren zu erkennen, muss man abstraktes Wissen über Gefahren im Allgemeinen besitzen. Dieses abstrakte Wissen heißt Metawissen. Das zur Erkennung von Gefahren notwendige Metawissen ergibt sich aus deren Gefahrenklasse. Gefahren, die mit dem gleichen Metawissen erkannt werden können, gehören der gleichen Gefahrenklasse an.

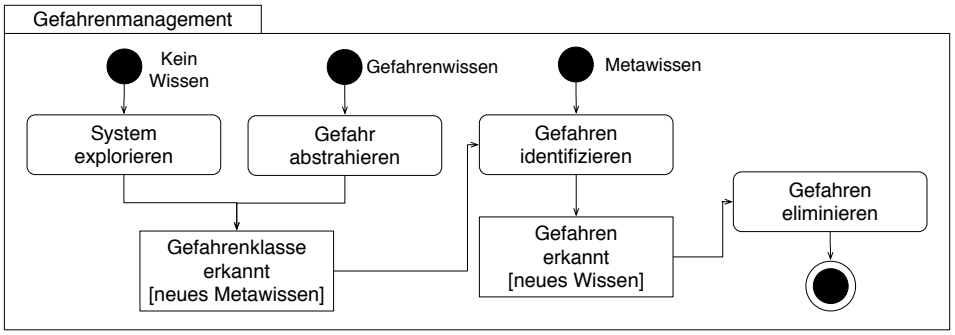


Abb. 3.2: *Gefahrenmanagement unter der Annahme existierender Gefahren*

Gefahren können unterschiedliches Metawissen zur Erkennung voraussetzen. Man muss dementsprechend die Eigenschaften kennen, die eine Begebenheit zur Gefahr werden lässt. Folglich ist Metawissen auch über nicht vorhandene Gefahrenwissen notwendig.

Beispiel 12

Gefahrenwissen und Metawissen können am Beispiel der Guillaume-Affäre beschrieben werden: Während die Erkenntnis, dass Günter Guillaume ein Spion ist, Gefahrenwissen ist, ist die Erkenntnis, dass die DDR versucht Spione auf höchster Ebene einzuschleusen, Metawissen.

Gefahrenwissen	**Metawissen**	**Mögliche Maßnahmen**
existent	existent	- Guillaume verhaften - Sicherheitsregeln einführen
existent	nonexistent	- Guillaume verhaften
nonexistent	existent	- Sicherheitsregeln einführen
nonexistent	nonexistent	- keine

Anhand dieses Beispiels wird deutlich, dass man in manchen Fällen anhand von Gefahrenwissen Rückschlüsse auf Metawissen ziehen kann. Denn durch die Bekanntheit von Guillaume wurde dieser Typ der Spionage aufgedeckt.

3.5 Methoden der Gefahrenerkennung

Prinzipiell gibt es zwei Möglichkeiten, Systeme auf Fehlkonfigurationen zu untersuchen. Bei der White-Box-Analyse wird die Konfiguration selbst untersucht, um Fehlkonfigurationen zu entdecken. Die White-Box-Analyse erlaubt auch das Entdecken von Fehlkonfigurationen, die aktuelle keine Auswirkungen haben [NN09]. Alfaro et. al. stellen einen White-Box-basierten Ansatz vor, mit dem Sicherheitstests für eine Black-Box-Analyse erstellt werden [ABCC08]. So kann der Ist- mit dem Sollzustand automatisch überprüft werden. Alternativ wird ein Expertensystem von Eronen und Zitting vorgestellt, mit dem Firewallregeln analysiert werden können [EZ01].

Bei der Black-Box-Analyse wird die Verhaltensweise eines konfigurierten Systems getestet. Dabei wird das konfigurierte System unter realen oder realitätsähnlichen Bedingungen observiert. Auf Basis dieses Verhaltens werden Rückschlüsse auf die Konfiguration des observierten Systems gezogen. Das erlaubt es, auch Fehler in Konfigurationen zu finden, wenn diese nicht vorliegen. Grabowski stellt mit TTCN-3 eine Sprache vor, mit der Test für

verteilte Systeme spezifiziert werden können [Gra00]. Häberlein et. al. stellen PeerReview vor, ein System, welches die Nachrichten zwischen den Komponenten eines verteilten Systems observiert [HKD07]. Damit ist PeerReview nicht nur in der Lage zu erkennen, ob ein verteiltes System Fehlkonfigurationen enthält, sondern auch auf welche Komponente sie sich befinden.

Mit dem White-Box-Verfahren können Fehlkonfigurationen auch dann erkannt werden, wenn sie ohne weitere Fehler noch keine Auswirkungen haben. Die White-Box-Analyse erlaubt Tests, bei denen weder Testumgebungen aufgebaut werden müssen noch Tests am produktiven System durchgeführt werden müssen. Die Black-Box-Analyse erlaubt hingegen das Testen ohne vorliegende Konfiguration. Denn die Black-Box-Analyse testet nicht nur die Korrektheit der Konfiguration sondern, auch die des observierten Systems.

Zwischen der White-Box- und er Black-Box-Analyse gibt es hybride Ansätze, welche Grey-Box-Analyse genannt werden [EGPQ06]. Der in dieser Dissertation vorgestellte Ansatz verfolgt primär das Prinzip der White-Box-Analyse. Es wird jedoch auch, wenn die Konfiguration einer Komponente nicht vorliegt, die Black-Box-Analyse unterstützt.

Um verteilte Systeme mit einer White-Box-Analyse auf Fehlerkonfigurationen zu überprüfen, muss deren Konfiguration vorliegen. Es gibt zwei konträre Ansätze, um die gesamte Konfiguration eines verteilten Systems zu erlangen. Der Top Down Ansatz basiert auf einer zentral verwalteten Konfiguration aller Komponenten eines verteilten Systems. Alle Änderungen werden an der zentralen Konfiguration vorgenommen und von dort aus auf die einzelnen Komponenten gespielt. Dieser Top-Down-Ansatz wird beispielsweise beim Software-Defined Networking (SDN) verfolgt [Kir13]. Beim Ansatz des SDN ist es ein Ziel, nicht mehr die Komponenten zu konfigurieren, sondern nur noch abstrakt Services [MRF+13] und Grundsätze [Slo94] zu definieren. SDNs kommen in IaaS[1] Umgebungen zu Einsatz, einem Ansatz bei dem sämtliche Leistungen einer Infrastruktur als Service angeboten werden [LW14]. Greenberg et. al. [GHM+05] schlagen zur Reduzierung der Komplexität das 4D-Modell vor, die eine Architektur in die vier Ebenen Decision, Dissemination, Discovery und Data unterteilt. SDN gilt als Methode, die nur sehr aufwändig in existierende Architekturen zu integrieren ist [KF13]. Ein bekanntes Beispiel für den Einsatz von IaaS sind die Amazon Web Services (AWS). Der Top Down Ansatz erfordert Application Programming In-

[1] Infrastructure as a Service (IaaS)

terfaces (API's) die das Konfigurieren durch externe Werkzeuge zulassen. Der Einsatz von Komponenten ohne eine derartige API ist nicht möglich. Die Frage, wie existierende Systeme in ein SDN überführt werden können, ist bisher noch nicht beantwortet.

Im Gegensatz zum Top-Down-Ansatz erlaubt der Bottom-Up-Ansatz weiterhin eine dezentrale, eigenständige Konfiguration auf den einzelnen Komponenten. Das ist eine wesentliche Eigenschaft der in dieser Dissertation untersuchten verteilten Systeme. Reiter et. al. zeigen, wie in einem Institutional Management Information System (IMIS) eine Sicherheitsarchitektur mittels einem speziellen Zugriffskontrollschema integriert werden kann [RBG92]. Stiemerling et. al. stellen die EVOLVE-Architektur vor, die es Komponenten erlaubt, Schnittstellen für ihre Konfiguration bereitzustellen [SHC99]. Chandy und Lamport stellen einen Algorithmus vor, der den globalen Status verteilter Systeme ermittelt und es erlaubt Fehler besser aufzuspüren [CL85]. Über die Schnittstellen kann nicht nur die aktuelle Konfiguration ausgelesen werden, sie bestimmt auch wie die Konfiguration verändert werden darf und wie dies visualisiert werden soll. Als Erweiterung stellen Stiemerling et. al. ein 3D-Interface für EVOLVE vor [SHC01b]. Auch wenn EVOLVE eine individuelle Konfiguration der einzelnen Komponenten unterstützt, sind nur jene Komponenten mit EVOLVE kompatibel, welche die durch EVOLVE definierten Schnittstellen bereitstellen.

Sowohl die Top-Down-Analyse als auch die Bottom-Up-Analyse erlaubt die Trennung von Regelwerk und Komponente. Eine solche Trennung wird vom PBM propagiert welches in Abschnitt A.1.4.1 des Appendix genauer definiert ist.

In dieser Dissertation wird ein Bottom-Up-Ansatz verfolgt der eine Bedingung an die im verteilten Systems eingesetzten Komponenten stellt: Die Konfiguration muss exportiert sein, eine Bedingung die nahezu alle in den Fallstudien untersuchten Komponenten erfüllten. Jene Komponenten, die keinen Export der Konfiguration unterstützen, können dennoch mit einer Black-Box-Analyse auf ihr Verhalten untersucht werden. So kann zwar nicht festgestellt werden, ob die Konfiguration des restlichen verteilten Systems zu der Konfiguration der Komponente passt.

3.6 Gefahrenexternalisierung

Gefahrenexternalisierung beschreibt den Transfer von Gefahrenwissen von Gefahrenspezialisten zu Gefahrenexperten. Hierfür müssen die Spezialisten ihr Gefahrenwissen in eine ausführbares Programme überführen. Derartige Programme heißen in dieser Dissertation Experten. Unter der Externalisierung ist der Gefahrenwissenstransfer von Spezialisten zu Experten zu verstehen (siehe Abschnitt 2.7). Dieser erlaubt es, Gefahrenwissen ohne Beanspruchung von menschlichen Ressourcen, den Spezialisten, einzusetzen. Beim Vorgang der Externalisierung müssen ein oder mehrere Spezialisten Gefahrenwissen bereitstellen (siehe Abbildung 3.3).

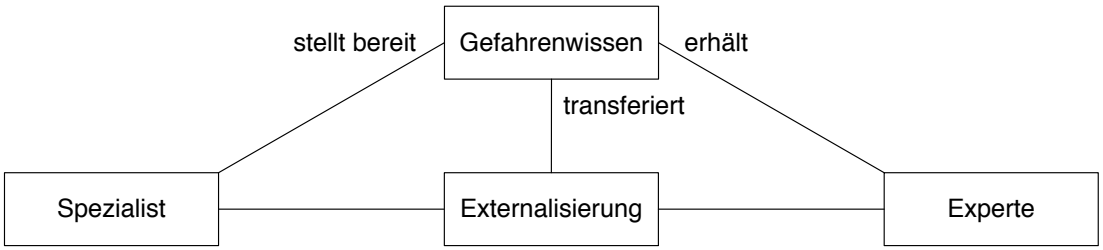


Abb. 3.3: *Gefahrenwissenstransfer von Spezialisten zu Experten*

Das bereitgestellte Gefahrenwissen wird einem Experten zugeführt. Dieses Zuführen findet, in Abhängigkeit vom Gefahrenwissen, unterschiedlich statt. Bei Wissen über die Konfiguration von Komponenten kann beispielsweise ein neuer Experte erstellt werden, der in der Lage ist, die Konfiguration auszulesen und bereitzustellen. Dieser Prozess erfordert Programmierkenntnisse und die Spezialisten können hierbei von Software-Entwicklern unterstützt werden. Wissen über die Erkennung von Fehlkonfigurationen kann mittels eines neuen Regelsatzes an einen bestehenden Experten übergeben werden. Dieser Experte entspräche dann einem regelbasierten System, welches auf der Basis der Konfigurationen des verteiltes Systems und dem Regelsatz Hypothesen zu Fehlkonfigurationen erstellt. Solche regelbasierten Systeme kommen bereits bei Angrifferkennungssystemen (engl. IDS) zu Einsatz [IKP+95].

3.7 Visualisierung von Gefahren

Die Informatik gilt als die Grundlagentechnologie für die Applikation von Visualisierungen in mehreren Fachgebieten [GE97]. In der Modellierung beispielsweise, beschäftigt sich die UML mit der Visualisierung von Spezifikation, Konstruktion und Dokumentation von Software oder anderen Systemen [Uni10]. In der Medizin gewinnt die Visualisierung zunehmend an Bedeutung: Durch Bildanalyse können Operateure während der Operation wichtige Informationen erhalten [FLR+98]. Bei ein Krebserkennung können Geschwüre identifiziert und aufgezeigt werden [WHS+10]. Und in der Pränataldiagnostik können mit Hilfe von Visualisierungen frühzeitig Krankheiten erkannt werden [NAB+92]. Die folgenden Abschnitten stellen einige Visualisierungstechniken vor, mit deren Hilfe Gefahren visualisiert werden können.

3.7.1 Visual Analytics

Miller beschreibt mit seinem 7 ± 2 – Gesetz, dass Menschen nur eine begrenzte Anzahl von Informationen gleichzeitig erfassen können [Mil56]. Ebenso können manche Informationen besonders gut im Kontext anderer Informationen verstanden werden. Andrienko et. al. stellen in ihren Publikationen verschiedene Techniken der Visual Analytics vor [AAB+13, AAW07]. Die Herausforderungen, die Visual Analytics an ihre Applikationen stellt, werden von Keim et. al. beschrieben [KAF+08]. Eine verbreitete Technik ist Repräsentation von Koordinaten auf einer Karte.

Die Visualisierung historischer Entwicklungen ist ein weiterer Bereich aus der Visual Analytics. Denn bei historisiert vorliegenden Informationen fällt bei der gleichen Menge der beschriebenen Elemente eine vielfache Menge an Informationen an. Historisierte Daten lassen sich mit Motion-Charts in geeigneten Aggregationen vereinfacht darstellen [ZE08]. Motion-Charts sind Blasendiagramme, welche um eine dritte Dimension – die Zeit – erweitert wurden. Die beiden ersten Dimensionen werden durch die X- und Y-Achse des zugrundeliegenden Blasendiagramms dargestellt. Zwei weitere Dimensionen können durch Größe und Farbe der Blasen visualisiert werden.

Rosling verwendet Motion-Charts um die Korrelation zwischen dem Durchschnittseinkommen und der durchschnittlichen Lebenserwartung aufzuzei-

gen [RRR05]. Jede in Abbildung 3.4[2] dargestellten Blase repräsentiert ein Land. Die Y-Achse stellt die durchschnittliche Lebenserwartung und die X-Achse stellt das durchschnittliche Einkommen der Länder dar. Die Farben der Blasen definieren die Herkunftsregionen der Länder und die Größe der Blasen verkörpert die Einwohneranzahl. Der horizontale Schieber am unteren Rand ermöglicht dem Betrachter die Navigation zwischen den Jahren 1800 bis 2014.

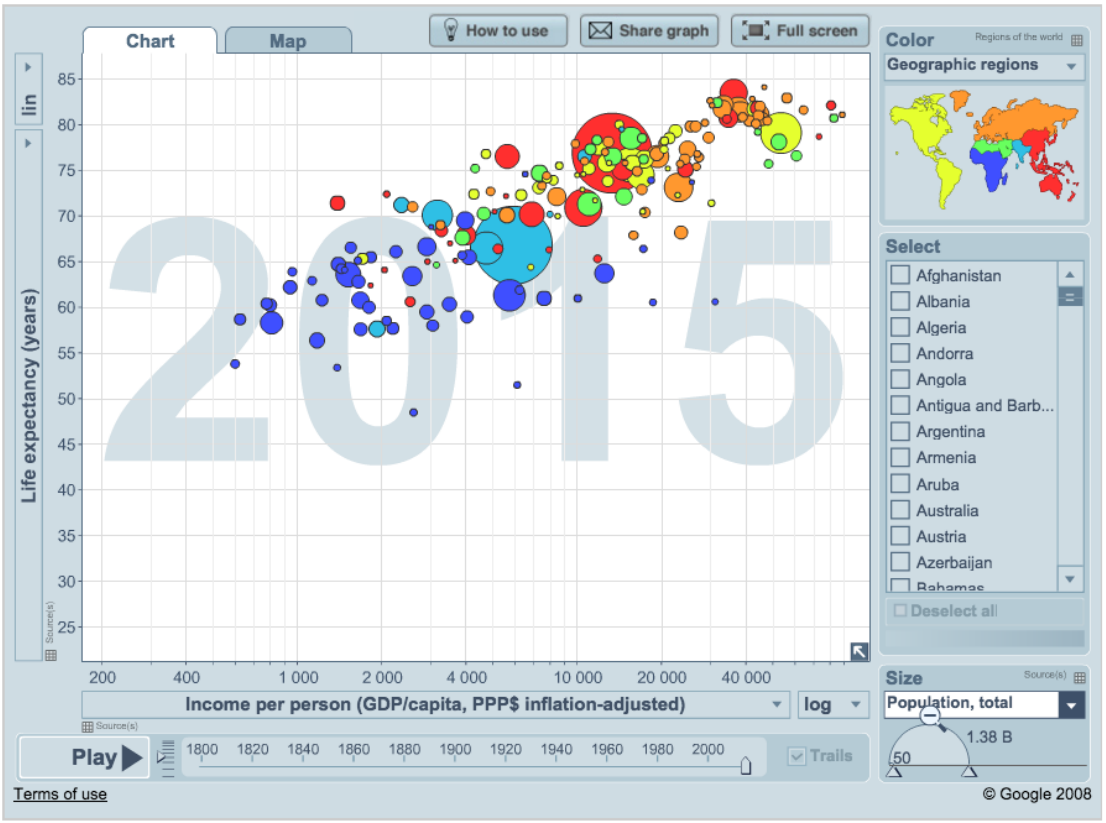


Abb. 3.4: *Motion-Chart zur Visualisierung von Abhängigkeiten zwischen Lebenserwartung, Einkommen, Bevölkerungsanzahl und Staaten [RRR05]*

[2] gapminder.org: `www.gapminder.org/`
Motion-Chart: `www.bit.ly/11rv4NV`

3.7.2 Softwarekartographie

> *„Ziel der Softwarekartographie ist es, unter Rückgriff auf Erkenntnisse und Methoden der Kartographie, komplexe Anwendungslandschaften in Unternehmen systematisch darzustellen und damit die Beschreibung, Bewertung und Gestaltung von Anwendungslandschaften zu verbessern.“*
>
> – Josef Lankes, Florian Matthes [LMW05]

Die Softwarekartographie gehört der Visual Analytics an und befasst sich mit der Darstellung von Systemen als Karte. Diese Karten können dabei helfen das System zu verstehen, eine Voraussetzung zum Erkennen und Verstehen von Gefahren. Für die Erstellung derartiger deskriptiver Visualisierungen für Software gibt es den manuellen und den automatisierten Ansatz. Der manuelle Ansatz basiert auf einer Analyse von Spezialisten, welche aufgrund ihres Wissens und ihrer Erkenntnisse manuell eine Visualisierung erstellen. Der automatisierte Ansatz basiert auf auf einer durch Rechner durchgeführte Analyse, auf deren Basis Visualisierungen algorithmisch erstellt werden.

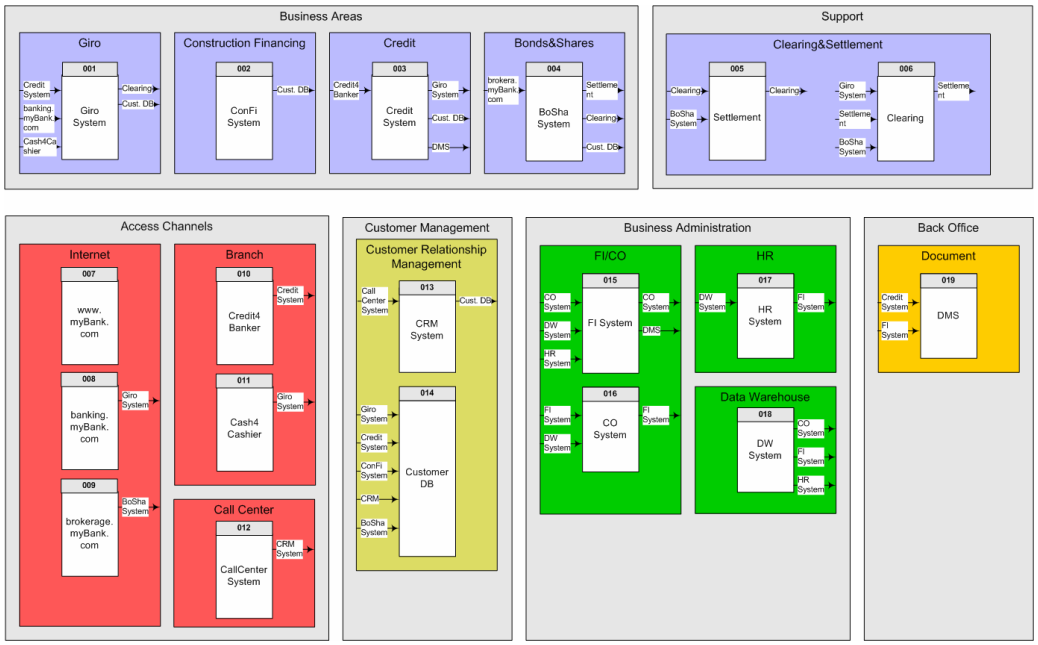


Abb. 3.5: *Clusterkarte einer Anwendungslandschaft mit logischen Domänen [Mat08]*

Clusterkarten (siehe Abbildung 3.5) untergliedern ein verteiltes System in sogenannte Domänen. Matthes beschreibt, wie Anwendungslandschaften – eine Unterklasse von verteilten Systemen – mit Clusterkarten, kartesische Karten und Graphlayout-Karten beschrieben werden können [Mat08]. In diesen Domänen werden die einzelnen Komponenten mit ihren zugreifenden und benötigten Komponenten modelliert. Bei kartesischen Karten gewinnt die Position einer Komponente Bedeutung. Beiden Achsen werden hierfür in Intervalle aufgeteilt, denen die Komponenten zugeordnet werden. Somit wird jede Komponente auf der kartesischen Karte in zwei Kategorisierungen eingeordnet. Beispiele für kartesische Karte sind Prozessunterstützungskarten und Zeitintervallkarten.

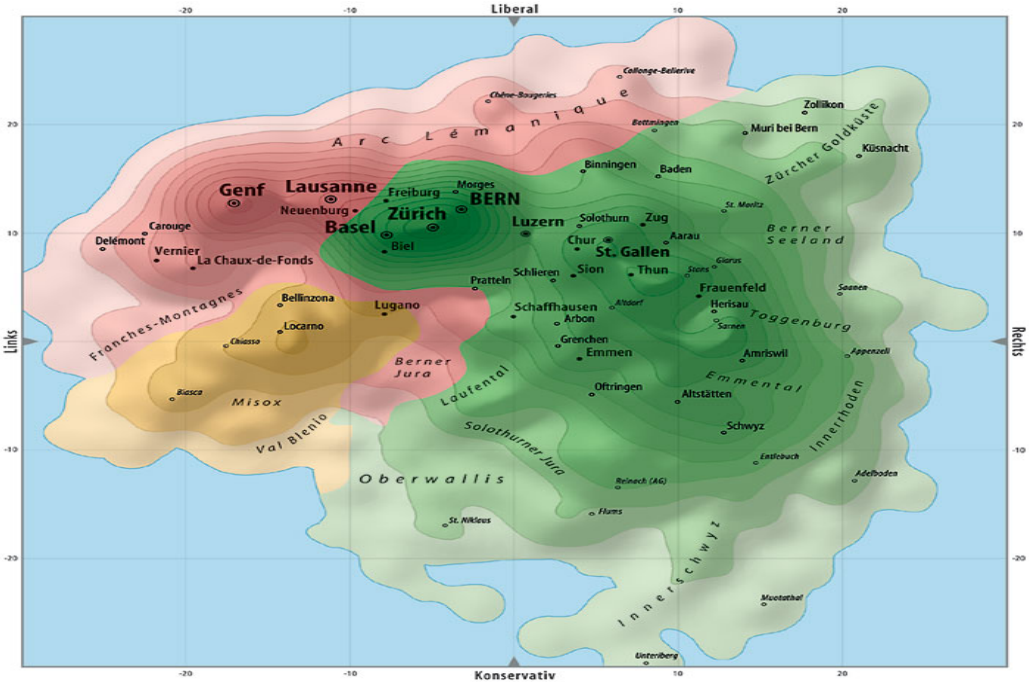


Abb. 3.6: *Politische Landschaft: Eine Anordnung Schweizer Gemeinden abhängig von deren Wahlergebnissen. Übernommen aus [Lor11]*

Bei automatisch erstellten Kartographien verlieren individuelle Meinungen und Einschätzungen an Bedeutung. Solche Kartographien basieren auf einen deterministischen, nachvollziehbaren Algorithmus [KELN10]. Loretan stellt in einem Beispiel „Thematische Softwarekarten“ vor, und wenden dieses, zur allgemeinen Verständlichkeit, auf die vergangen Schweizerischen Wahlergebnisse an (siehe Abbildung 3.6). Die auf der Y-Achse werden die Gemeinden auf einer Skala von Liberal zu Konservativ eingeteilt, auf der X-

Achse werden sie auf einer Skala von politisch Links zu politisch Rechts eingeteilt. Als Ergebnis erhält er eine Karte, die einer topographischen Landkarte ähnelt. Dadurch ist er in der Lage, auch die Größe und die Häufungen der Gemeinden zu visualisieren. Mittels Färbung gelingt es ihm einen vierten Aspekt auf der Karte darzustellen.

Als Grundlage, sowohl für die Visual Analytics im Allgemeinen als auch bei der Softwarekartographie im Speziellen, dienen Methoden aus der Statistik und der Verarbeitung von Big Data [LLS+13]. Ott und Longnecker beschreiben in ihren Buch viele wichtige Grundlagen, wie Korrelationsanalysen, Lineare Regression und Kovarianzanalyse [OL10].

4 Forschungsmethodik

Diese Kapitel definiert die bearbeiteten Forschungshypothesen dieser Dissertation und die wissenschaftlichen Methoden zur Bestätigung dieser Hypothesen. Abschnitt 4.1 erläutert zunächst die offenen Forschungshypothesen und legt die Zusammenhänge zwischen den Forschungshypothesen dar [ESSD08]. Anschließend beschreibt es die gewählten Forschungsansätze, also die angewandten wissenschaftlichen Methoden. Die Grundlagen für das Sicherheitsframework und dessen Referenzimplementierung CUSTODIAN werden in Abschnitt 4.2 vorgestellt. Abschnitt 4.3 geht auf das Vorgehensmodell HARVEST ein, welches für das Sicherheitsframework optimiert ist. Die Evaluationsmethoden, anhand derer CUSTODIAN als Sicherheitsframework validiert wird, sind in Abschnitt 4.4 erläutert.

4.1 Grundlegende Forschungshypothesen

Die klassische Erkennung von gefährlichen Konfigurationen in verteilten Systemen, wie sie in Kapitel 3 beschrieben ist, bedarf Gefahrenwissen mehrerer Spezialisten und ist fehleranfällig, teuer und zeitaufwändig. Deshalb stellt sich die Frage, ob dieser Prozess mit den Mitteln der Softwaretechnik unterstützt werden kann. Diese Dissertation untersucht die Einsetzbarkeit des Blackboard-Architekturmusters zur Erkennung von gefährlichen Konfigurationen in verteilten Systemen anhand der folgenden Ausgangshypothese:

> *„Mit der Hilfe einer Blackboard-Architektur können bisher unerkannte Gefahren in den Konfigurationen verteilter Systeme erkannt werden.“*
>
> Ausgangshypothese H_0

Aus dieser Ausgangshypothese lassen sich mehrere Voraussetzungen ableiten. Diese dürfen zur Aufrechterhaltung der Ausgangshypothese nicht widerlegbar sein. Die Voraussetzungen sind ebenfalls Hypothesen, die im Rahmen dieser Dissertation untermauert werden sollen. Derartige zugrundeliegenden Hypothesen können wiederum auf Bedingungen, also weiteren Hypothese, basieren. Dadurch entsteht ein Hypothesenbaum mit der Ausgangshypothese als Wurzel. Alle unter dieser Wurzel stehenden Knoten sind Hypothesen und direkte oder indirekte Voraussetzungen der Ausgangshypothese. Die Falsifikation einer beliebigen im Baum befindlichen Hypothese widerlegt transitiv die Ausgangshypothese. Ebenso transitiv wirkt sich die Stärkung einer Voraussetzung stärkend auf Ausgangshypothese aus. Der aus der Ausgangshypothese abgeleitete Hypothesenbaum ist in Abbildung 4.1 dargestellt.

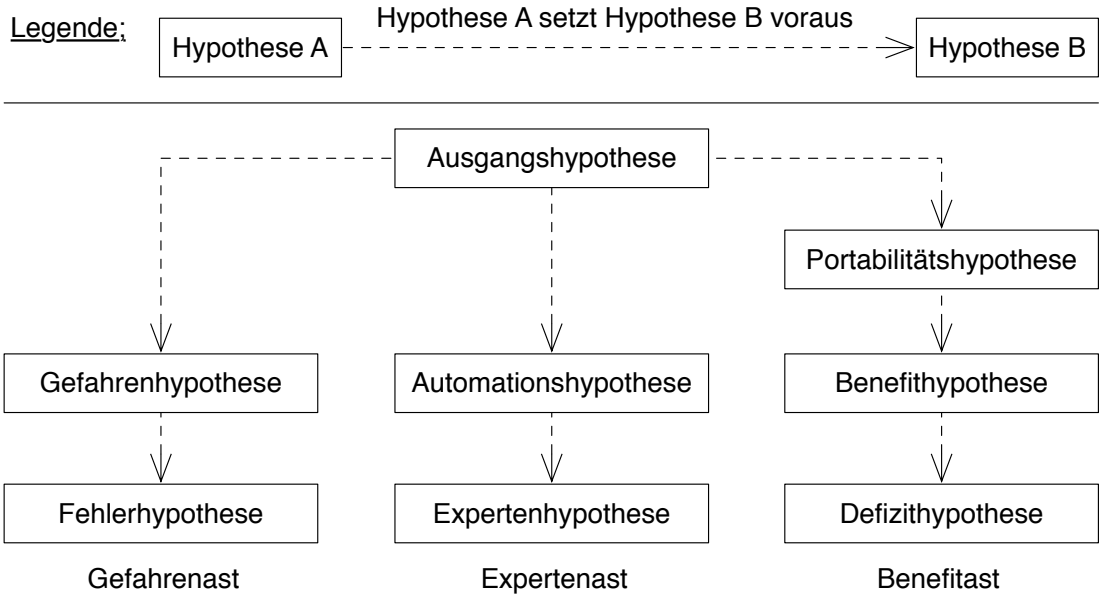


Abb. 4.1: *Hypothesenbaum zur Validierung der Ausgangshypothese*

Die Hypothesen dieses Hypothesenbaums sind für den Rahmen dieser Arbeit zur leichteren Referenzierung eindeutige benennt. Im Folgenden werden alle von der Ausgangshypothese abgeleiteten Hypothesen im Einzelnen definiert. Diese Definition beinhaltet auch die Methoden, anhand derer die Ausgangshypothese in dieser Dissertation untermauert wird.

H_{1a} **Fehlerhypothese:** „Verteilte Systeme bergen fehlerhafte Konfigurationen.“ Die Fehlerhypothese gilt für als komplex geltende verteilte Sys-

teme (siehe Abschnitt 2.1.2). Die Fehlerhypothese wird sowohl durch Beobachtungen anderer Forscher (siehe Abschnitt 3) als auch durch die Beobachtungen in den Fallstudien dieser Disseration gestärkt.

H_{1b} **Gefahrenhypothese:** „Fehlerhafte Konfigurationen verteilter Systeme können diese gefährden.“ Unter einer fehlerhaften Konfiguration verstehen wir eine Konfiguration, welche sich entweder in ihrer Gestalt oder in ihren Einfluss auf das verteilte System von der Intention des verantwortlichen Konfigurierenden unterscheidet. Sowohl wissenschaftlichen Veröffentlichungen (siehe Kapitel 3) als auch die Beobachtungen in den Fallstudien sprechen für die Gefahrenhypothese.

H_{2a} **Spezialistenhypothese:** „Die Erkennung vor Gefahren in verteilten Systemen setzt Gefahrenwissen zu Komponenten und den Gefahren voraus.“ Dieses Gefahrenwissen kann von entsprechenden Spezialisten (siehe Abschnitt 2.5) eingeholt werden. Über verschiedenartiges Wissen kann Gefahrenwissen bestehen: zu Gefahren (siehe Abschnitt 2.4) oder zur Funktionsweise einer Komponente (siehe Abschnitt 2.2). Die Spezialistenhypothese wird wie die Automationshypothese mittels der Fallstudien in Kapitel 8 validiert.

H_{2b} **Automationshypothese:** „Gefahrenwissen zur Erkennung von Gefahren in verteilten Systeme kann externalisiert und in ein spezialisiertes System überführt werden.“ Solche spezialisierten Systeme werden nach Abschnitt 2.7 dieser Dissertation Experten genannt. Mit einem Experten kann ein Nutzer auf Gefahrenwissen von Spezialisten zugreifen, ohne dass diesen ein konkreter Aufwand entsteht. Nicht nur Nutzer, sondern auch andere Experten können auf Experten zugreifen und das Gefahrenwissen des ursprünglichen Spezialisten verwenden. Die Automationshypothese wird im Rahmen der Fallstudien in Kapitel 8 validiert.

H_{3a} **Defizithypothese:** „In verteilten Systemen existieren unerkannte Fehlkonfigurationen.“ Die Defizithypothese beschreibt das Defizit bei der Erkennung von Fehlern in verteilten Systemen. Manche fehlerhafte Konfigurationen können nicht oder nicht mit vertretbarem Aufwand erkannt, werden. Andere Fehlkonfigurationen werden nicht erkannt weil sie einer unbekannten Gefahrenklasse angehören. Diese Hypothese gilt für verteilte Systeme, welche laut Abschnitt 2.1.2 als komplex gelten. Die Defizithypothese wird in den Fallstudien gestärkt.

H_{3b} **Benefithypothese:** „Der Einsatz der Blackboard-Architektur unterstützt die Erkennung fehlerhafter Konfigurationen eines verteilten

Systems.“ Die Erkennung bisher verborgener fehlerhafter Konfigurationen definiert den Benefit. Anhand der Fallstudien wird dargelegt, dass bisher verborgene fehlerhafte Konfigurationen mit Hilfe der Blackboard-Architektur erkannt werden.

H_{3c} **Portabilitätshypothese:** „Eine Lösung auf Basis der Blackboard-Architektur kann bei mehreren verschiedenartigen verteilten Systemen zur Erkennung von fehlerhaften Konfigurationen eingesetzt werden.“ Die Benefithypothese definiert, dass auf der Basis der Blackboard-Architektur Systeme zur Erkennung von fehlerhaften Konfigurationen Benefit erzeugen können. Nach der Portabilitätshypothese ist eine für verteilte Systeme universell einsetzbare Lösung auf Basis der Blackboard-Architektur möglich. Mit dieser allgemeinen Lösung können bei mehreren, verschiedenartigen verteilten Systemen fehlerhafte Konfigurationen erkannt und somit ein Benefit erzeugt werden. Die Portabilitätshypothese erfordert daher mindestens zwei Fallstudien, bei denen verschiedenartige verteilte Systeme mit der gleichen Lösung hinsichtlich der Benefithypothese überprüft werden. Im Gegensatz zur vorliegenden, schwach formulierten Portabilitätshypothese lautet die starke Formulierung: „Eine Lösung auf Basis der Blackboard-Architektur kann bei allen verteilten Systemen zur Erkennung von fehlerhaften Konfigurationen eingesetzt werden.“ Die starke Portabilitätshypothese wird im Rahmen dieser Dissertation jedoch nicht belegt. Das Kapitel 9 gibt aber einen Ausblick, wie die starke Portabilitätshypothese validiert werden könnte.

Der vorliegende Hypothesenbaum teilt sich in drei Äste auf, den Gefahrenast, den Expertenast und den Benefitast. Der Gefahrenast stellt Hypothesen zur zugrundeliegenden Problematik der gefährlichen Konfigurationen in verteilten Systemen auf. Die Annahmen, auf denen der in dieser Arbeit vorgeschlagenen Lösungsweg basiert, sind im Expertenast als Hypothesen formuliert. Dieser Lösungsweg ist mit der Definition des Sicherheitsframeworks (Kapitel 6) aufgezeigt. Die Referenzimplementierung KUSTOS ist eine Instanz dieses Sicherheitsframeworks CUSTODIAN. KUSTOS ermöglicht es, die aufgestellten Hypothesen anhand von Fallstudien zu validieren. Das Vorgehensmodells HARVEST ist ein Benutzerhandbuch für CUSTODIAN und dessen Referenzimplementierung KUSTOS. Der Benefitast hypothesiert den Mehrwert für existierende verteilte Systeme. Es wurden drei Fallstudien durchgeführt, welche zur Validierung des Benefitasts dienten. Dennoch stärken die Ergebnisse der Fallstudien auch die Hypothesen des Experten-

und des Gefahrenasts. Wie die einzelnen Forschungshypothesen in den einzelnen Kapiteln belegt sind ist aus der folgenden Übersicht ersichtlich:

H_{1a} **Fehlerhypothese**

Kapitel 3 Verweis auf existierende Literatur, welche darauf hinweist, dass verteilte Systeme fehlerhafte Konfigurationen besitzen.

Kapitel 8 In Fallstudien werden Fehler identifiziert. Die Antworten auf Frage 1 des Fragebogens stärken die Hypothese.

H_{1b} **Gefahrenhypothese**

Kapitel 3 Verweis auf existierende Literatur, welche darauf hinweist, dass Fehler zu Gefahren führen.

Kapitel 8 Die in den Fallstudien identifizierten Konfigurationsfehler sind Gefahren. Die Antworten auf Frage 2 des Fragebogens stärken die Hypothese.

H_{2a} **Spezialistenhypothese**

Kapitel 8 In allen Fallstudien waren jeweils mehrere Spezialisten vonnöten, um die Gefahren zu identifizieren. Die Antworten auf Frage 3 des Fragebogens stärken die Hypothese.

H_{2b} **Automationshypothese**

Kapitel 8 Die benötigten Spezialisten konnten in Experten überführt werden. Die Antworten auf Frage 4 des Fragebogens stärken die Hypothese.

H_{3a} **Defizithypothese**

Kapitel 8 In den Fallstudien existierten unbekannte Gefahrenklassen und Gefahren. Die Antworten auf Frage 5 des Fragebogens stärken die Hypothese.

H_{3b} **Benefithypothese**

Kapitel 8 In den Fallstudien wurden neue Gefahrenklassen und unbekannte Gefahren identifiziert. Die Antworten auf Frage 6 des Fragebogens stärken die Hypothese.

H_{3c} **Portabilitätshypothese**

Kapitel 8 CUSTODIAN wurde für die Fallstudie in einem Hochsicherheitsrechenzentrum entwickelt. Die Tatsache, dass CUSTODIAN in ähnlichen verteilten System der RBG und auch im sehr unterschiedlichen verteilten System von AdWorks2go erfolgreich eingesetzt wurde, stärkt die Hypothese.

4.2 Sicherheitsframework

Die in Abschnitt 4.1 definierte Defizithypothese unterstellt, dass die aktuell eingesetzten Mitteln nicht für die Erkennung mancher fehlerhaften Konfigurationen geeignet ist. Entweder sind die Sicherheitsverantwortlichen nicht in der Lage, die Gefahren zu entdecken, oder ihnen stehen nicht die notwendigen Ressourcen zur Verfügung. Daraus entwickelte sich die Überlegung, den Sicherheitsverantwortlichen bei der Gefahrenerkennung ein technisches Hilfsmittel, also ein zur Gefahrenerkennung spezialisiertes System, zur Seite zu stellen. Jedoch ist mit der Erkennung von gefährlichen Konfigurationen in einem konkreten verteilten System noch nicht das allgemeine Problem der gefährlichen Konfigurationen aller verteilten Systemen gelöst. Dementsprechend besteht der Anspruch ein System zu entwickeln, welches allgemein für die Gefahrenerkennung in verteilten Systemen eingesetzt werden kann. Darunter fallen auch verteilte Systeme, die heute noch nicht bekannt sind. Um auch bisher unbekannte verteilte Systeme auf Gefahren analysieren zu können, ist eine generische Lösung gefragt. Frameworks bieten eine generische Lösung für eine bestimmte Problemdomäne [Pre94]. Im Sicherheitsframework sind sowohl Gefahren als auch die Komponenten von verteilten Systemen abstrahiert. Diese Abstraktion erlaubt es, Gefahren in unterschiedlichen verteilten Systemen zu erkennen.

Zur Klassifizierung von Gefahren führt diese Dissertation die Meta Threat Facility (MTF) ein (siehe Abschnitt 5). Gefahrenklassen werden in die Meta Threat Facility aufgenommen. Bei der Analyse verteilter Systeme unterstützt die MTF, bereits bekannte Gefahren zu erkennen. Modellierern erlaubt die MTF Gefahren auf UML-Basis zu modellieren.

4.3 Vorgehensmodell

Das Sicherheitsframework in der Gestalt der Referenzimplementierung CUSTODIAN bietet eine technische Unterstützung zur Erkennung von Gefahren in verteilten Systemen zu erkennen. CUSTODIAN soll ein in der Praxis anwendbares Werkzeug sein. Jedoch hilft die alleinige Existenz eines Werkzeugs nicht zwingend weiter. Deshalb beschreibt diese Dissertation auch ein auf das Sicherheitsframework abgestimmtes Vorgehensmodell. Dieses

Vorgehensmodell soll als methodische Bedienungsanleitung für CUSTODIAN verstanden sein. Es umfasst Grundlagen für die Nutzung des Sicherheitsframeworks im Allgemeinen und von CUSTODIAN im Speziellen. In diese Grundlagen flossen im Wesentlichen die intentionalen und reflexiven Aspekte der Entwicklung des Sicherheitsframeworks ein. Die intentionalen Aspekte ergeben sich aus der Absicht, mit der das Sicherheitsframework entwickelt wurde und Entwurfsentscheidungen getroffen wurden. Die reflexiven Aspekte ergeben sich aus dem reflektierten Beobachten des Einsatzes der Referenzimplementierung von CUSTODIAN. Das resultierende Vorgehensmodell heißt HARVEST und wird in Kapitel 7 beschrieben. HARVEST beschreibt sogleich den Prozess, der bei den Fallstudien zum Einsatz kam.

4.4 Fallstudien

CUSTODIAN bieten zwei Konzepte der Erweiterbarkeit. Die MTF ist um die Definition von Gefahrenklassen erweiterbar. Neu erkannte Gefahrenklassen können somit in beschrieben werden. Und CUSTODIAN erlaubt die Erweiterungen um neue Gefahrenexperten. Die Gefahren der neu erkannten Gefahrenklassen können somit automatisch erkannt werden. Wir wollen durch die Fallstudien die evolutionäre Erweiterung der MTF um neue Gefahrenklassen und von CUSTODIAN um neue Gefahrenexperten herbeiführen. Ebenso sollen in den Fallstudien Gefahren bekannter Gefahrenklassen in verteilten Systemen erkannt werden. Wir verfolgen somit eine evolutionäre Forschungsmethodik.

5 Die Meta Threat Facility

> *Your assumptions are your windows on the world. Scrub them off every once in a while, or the light won't come in.*
>
> – Isaac Asimov, Biochemiker und Schriftsteller

Die grundlegende Hypothese der Dissertation ist, dass wir weder alle Gefahren noch alle Gefahrenklassen kennen und unser Gefahrenwissen unvollständig ist. Folglich gilt für uns kein verteiltes System als sicher, auch dann wenn uns keine Gefahren für dieses System bekannt sind. Dieser Auffassung liegt die Open World Assumption (OWA) zugrunde, die es verbietet, Rückschlüsse aus nicht vorhandenen Wissen zu ziehen [Kee13].

Im semantischen Web wird die OWA ebenfalls getroffen [BLHL+01]. Sprachen wie die Web Ontology Language (OWL) bauen auf der OWA auf [AH09]. Das semantischen Web bietet damit ein Konzept, welches unvoreingenommen mit unbekanntem Wissen umgeht: Einerseits liefert das semantische Web keine Aussagen über unbekanntes Wissen, andererseits erlaubt es, unbekanntes Wissen bei Bekanntwerden zu integrieren. Auch das Sicherheitsframework CUSTODIAN soll offen mit unbekanntem Wissen umgehen: Es soll bisher unbekanntes Wissen über Gefahren und Komponenten in CUSTODIAN zum Einsatz kommen können.

Aus diesem Grund ist die Erweiterbarkeit eine wichtige nichtfunktionale Anforderung an das CUSTODIAN. Dieser Abschnitt stellt die Meta Threat Facility (MTF), eine erweiterbare Klassifikation von Gefahren, vor. MTF ist ein erweiterbares UML-Profil [Uni10] und dient als Ausgangspunkt eines evolutionären Metamodells für Gefahren. Bei jedem Auffinden einer neuartigen Gefahr wird CUSTODIAN um dieses Gefahrenwissen und MTF um eine Gefahrenklasse erweitert. Die MTF ist, wie für UML-Profile vorgesehen, ein Modell auf der Meta-Ebene M2 (siehe Abbildung 5.1).

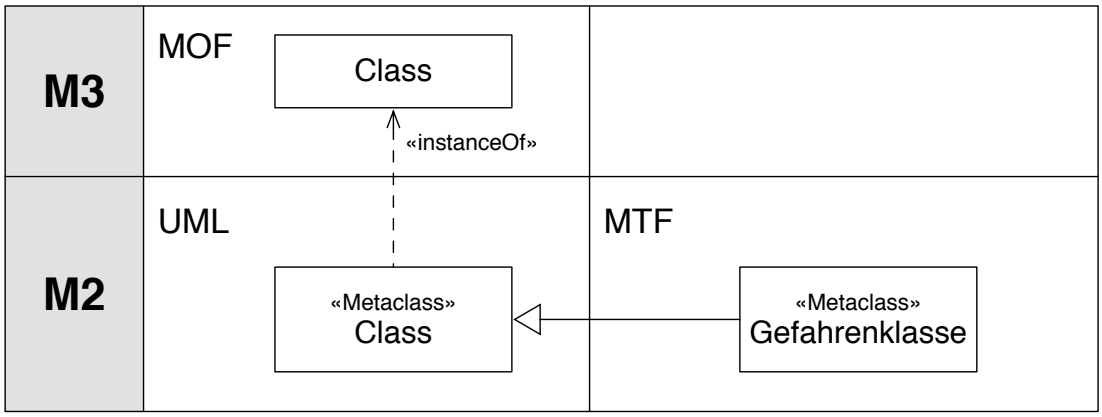


Abb. 5.1: *Der Kern der Meta Threat Facility (MTF) als UML-Profil*

MTF definiert `Gefahrenklasse` als Metaklasse, welche die Superklasse aller später definierten konkreten Gefahrenklassen ist. `Gefahrenklasse` ist der Kern der MTF und kann flexibel um Gefahrenklassen erweitert werden. Das entspricht dem Muster des Mikrokernels [Bus98]. `Gefahrenklasse` ist vom UML-Objekt `Class` abgeleitet und die wiederum ist eine Instanz des MOF-Objektes `Class` [Met15].

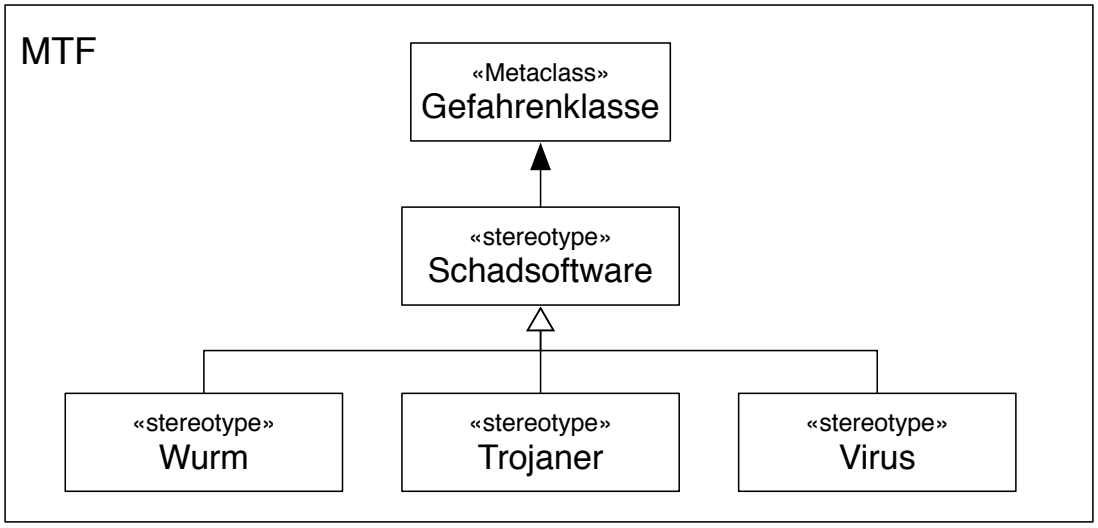


Abb. 5.2: *Beispielhafte Taxonomie in MTF über Schadsoftwarearten (Ebene M2)*

Die konkreten Gefahrenklassen werden als Stereotypen definiert und leiten sich entweder von `Gefahrenklasse` oder einer anderen konkreten Gefahrenklasse ab. Dadurch ergibt sich eine Taxonomie der Gefahrenklassen.

Gefahrenklassen sind auch ontologisch mit den Ansätzen des semantischen Webs beschreibbar. In dieser Dissertation ist jedoch ein taxonomischer Ansatz auf Basis der UML gewählt. Abbildung 5.2 zeigt beispielhaft eine Taxonomie in MTF über die Gefahrenklasse `Schadsoftware`, die sich in die Gefahrenklassen `Wurm`, `Trojaner` und `Virus` [Coh87] untergliedert.

Drei Aktivitäten erlauben bei der Analyse verteilter Systeme das Auffinden neuer Gefahrenklassen:

- **Replikation** Das Überprüfen des verteilten Systems auf bereits in MTF definierte Gefahrenklassen. In anderen verteilten Systemen gewonnenes Gefahrenwissen kommt somit zum Einsatz.
- **Exploration** Das Explorieren des verteilten Systems mit Hilfe des Sicherheitsframeworks CUSTODIAN. Durch die Zusammenführung von Konfigurationen erhalten die Analysten einen Überblick über die Gesamtkonfiguration des verteilten Systems.
- **Akquise** Das Einbinden von Spezialisten, in die Analyse. Spezialisten die sich beispielsweise mit Komponenten oder Sicherheitsanforderungen des verteilten Systems befassen. Diese Spezialisten können aufgrund ihrer Sachkenntnis auf neue Gefahrenklassen hinweisen.

Bei der Gefahrenanalyse eines verteilten Systems können alle drei Erkennungsmethoden gleichzeitig eingesetzt werden. Abbildung 5.3 stellt den Prozess dar, in dem für jede der drei Erkennungsmethoden ein eigener Workflow durchlaufen wird. Wenn man eine neue Gefahrenklassen in einem verteilten System erkennt muss man bestimmen, ob diese bereits in der Taxonomie der MTF existiert. Ist das nicht der Fall, liegt eine neuartige Gefahrenklassen vor, die in die MTF aufgenommen wird. In jedem Fall wird die Sicherheitsarchitektur CUSTODIAN um Gefahrenwissen erweitert, welches es erlaubt, das verteilte System in Bezug auf die neue Gefahrenklasse zu analysieren. Dieser Prozess kann einmalig aber auch kontinuierlich durchlaufen werden. Als kontinuierlicher Prozess kann auch neues Gefahrenwissen in die Analyse einbezogen werden. Das einmalige Durchlaufen ist hingegen diskret und geht auf die Evolution des verteilten Systems und des Gefahrenwissens nicht ein.

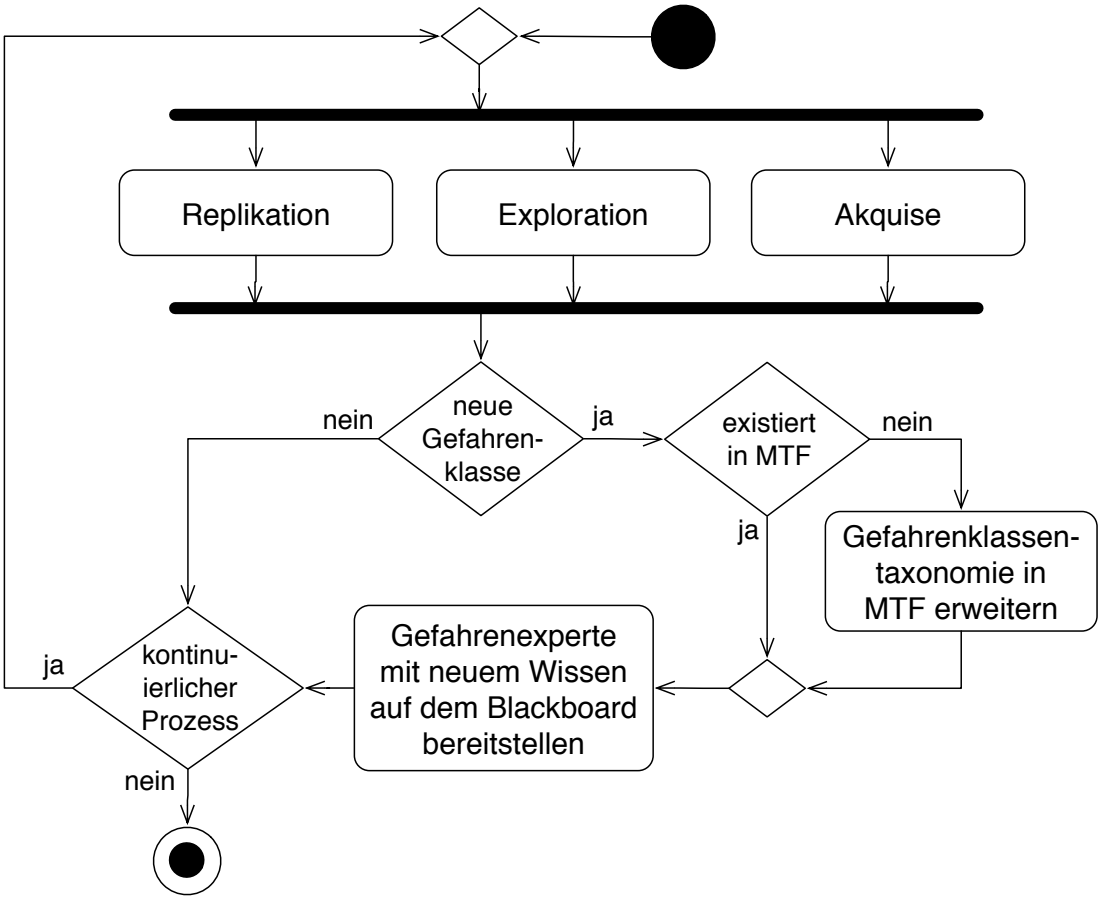


Abb. 5.3: *Prozess zur Gefahrenerkennung in verteilten Systemen und der Gefahrenklassenerkennung für MTF*

Bei jeder Gefahrenanalyse wird der in Abbildung 5.3 vorgestellte Gefahrenerkennungsprozess initiiert. In Abbildung 5.4 ist der Einsatz von CUSTODIAN und der MTF am Beispiel des fiktiven verteilten Systems „VS-Bsp“ als Lebenszyklusmodell dargestellt.

Drei Ereignisse haben in diesem Lebenszyklusmodell Einfluss auf die MTF und auf CUSTODIAN. Das Auffinden einer neuen Gefahrenklasse im analysierten verteilten System ist mit dem Symbol ✚ gekennzeichnet. Das Symbol ▼ zeigt die Erweiterung der MTF um eine neue Gefahrenklasse. Das Symbol ◆ zeigt, die Erweiterung von CUSTODIAN um eine Gefahrenklassen, indem ein entsprechender Gefahrenspezialist auf dem Sicherheitsblackboard von CUSTODIAN hinzugefügt wird.

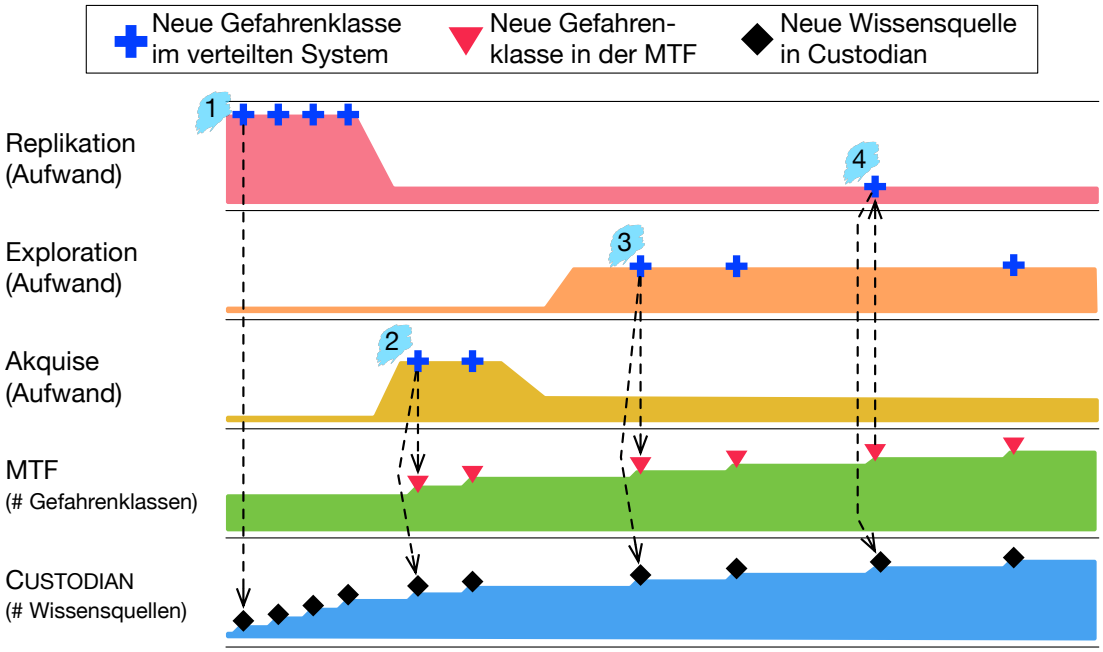


Abb. 5.4: *Beispielhafter Einsatz von CUSTODIAN in einem verteilten System. Der Arbeitsaufwand der drei Aktivitäten Replikation, Exploration und Akquise sind in den oberen drei Bahnen der Abbildung modelliert. Die Anzahl der in MTF definierten Gefahrenklassen ist in der vierten Bahn modelliert. Und die Anzahl der Gefahrenexperten auf dem Sicherheitsblackboard von CUSTODIAN ist in der letzten Bahn modelliert.*

Abbildung 5.4 veranschaulicht, wie während des Einsatzes von CUSTODIAN die MTF um weitere Wissensquellen ergänzt und das Sicherheitsblackboard um Gefahrenklassen bereichert wird. Ebenfalls zeigt die Abbildung, wie die Ereignisse ✚, ◆ und ▼ einander auslösen können. Im Folgenden sind die im Lebenszyklusmodell mit hellblau hinterlegten Nummern markierten Ereignisfolgen beschrieben.

[1]: Zu Beginn des Einsatzes von CUSTODIAN wird analysiert, ob die in MTF definierten Gefahrenklassen in VS-Bsp replizierbar sind. CUSTODIAN wird um die replizierbaren Gefahrenklassen erweitert.

[2]: Spezialisten des verteilten Systems VS-Bsp identifizieren mit Hilfe von neuem Gefahrenwissen eine neue Gefahrenklasse. Mit einem Gefahrenexperten erweitern die Spezialisten CUSTODIAN für die Gefahrenerkennung und MTF erweitern sie um die Definition der neuen Gefahrenklasse.

[3]: CUSTODIAN erlaubt das explorieren des verteilten Systems VS-Bsp. Während dieses Prozesses fällt den Analysten eine

VS-Bsp wird mit Hilfe von CUSTODIAN exploriert und eine neuen Gefahrenklasse wird identifiziert. Die Gefahrenklasse erweitert sowohl die MTF als auch CUSTODIAN.

[4]: Das Identifizieren einer neuen Gefahrenklasse eines anderen verteilten Systems erweitert die MTF. Diese Gefahrenklasse ist in VS-Bsp replizierbar erweitert CUSTODIAN.

Die Taxonomie der MTF kann flexibel und kontextübergreifend erweitert werden. Auch Gefahrenklassen, die für verteilten Systeme irrelevant sind, können die die Taxonomie aufgenommen werden. Abbildung 5.2 zeigt eine solche Erweiterung in Anlehnung an Beispiel 12. Die MTF aus Abbildung 5.2 wird um Gefahrenklasse `Spionagemethode` erweitert. Die Gefahrenklassen `Abhöranlage`, `Sabotage` und `Spion` sind Spezialisierungen der Gefahrenklasse `Spionagemethode`.

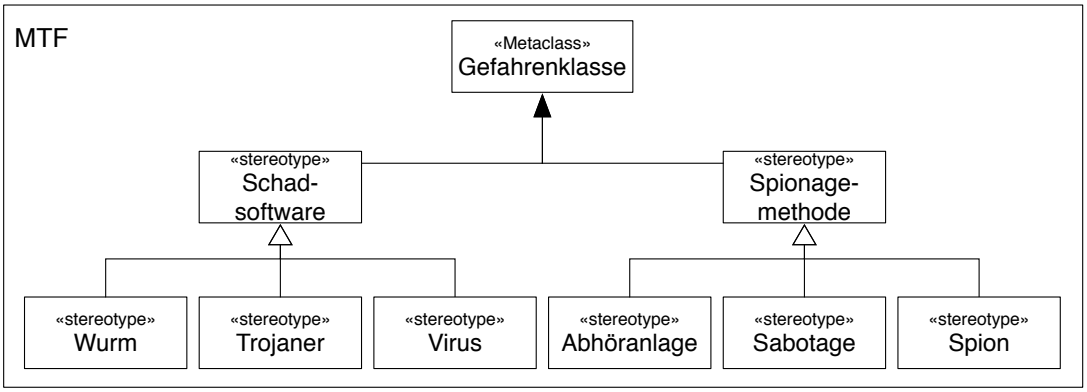


Abb. 5.5: *Die Erweiterung von MTF um Spionagemethoden (Ebene M2)*

6 Sicherheitsframework

„ Die einzige Konstante im Universum ist die Veränderung “

– Heraklit von Ephesos

Dieses Kapitel stellt das Sicherheitsframework CUSTODIAN[1] vor. CUSTODIAN erlaubt dem Benutzer, Gefahrenklassen zu identifizieren und Gefahren zu erkennen (siehe Abbildung 6.1). Über das UML-Profil MTF sind Benutzer von CUSTODIAN in der Lage, auf bestehende Gefahrenklassen zurückzugreifen und neu identifizierte Gefahrenklassen zu definieren. Zusätzlich kann das Sicherheitsframework Gefahren automatisiert erkennen und visualisieren.

Für das Sicherheitsframework CUSTODIAN wurde im Rahmen dieser Dissertation die Referenzimplementierung KUSTOS (siehe Kapitel 6.4) erstellt. KUSTOS realisiert somit die durch CUSTODIAN vorgegebene Architektur. Diese Referenzimplementierung kam bei drei Fallstudien zum Einsatz, die in Kapitel 8 beschrieben sind.

Diese Kapitel ist wie folgt aufgebaut: Abschnitt 6.1 beschreibt die bei der Realisierung des Sicherheitsframeworks zu erfüllenden Anforderungen. Abschnitt 6.2 legt die Ergebnisse der Analyse dar, und Abschnitt 6.3 stellt den Systementwurf vor.

[1] Custos ist ein Beiname Jupiters, welcher als Beschützer der Städte gilt [Vol36]. Wie Jupiter Städten Schutz bietet, soll auch CUSTODIAN Schutz für verteilte Systeme bieten.

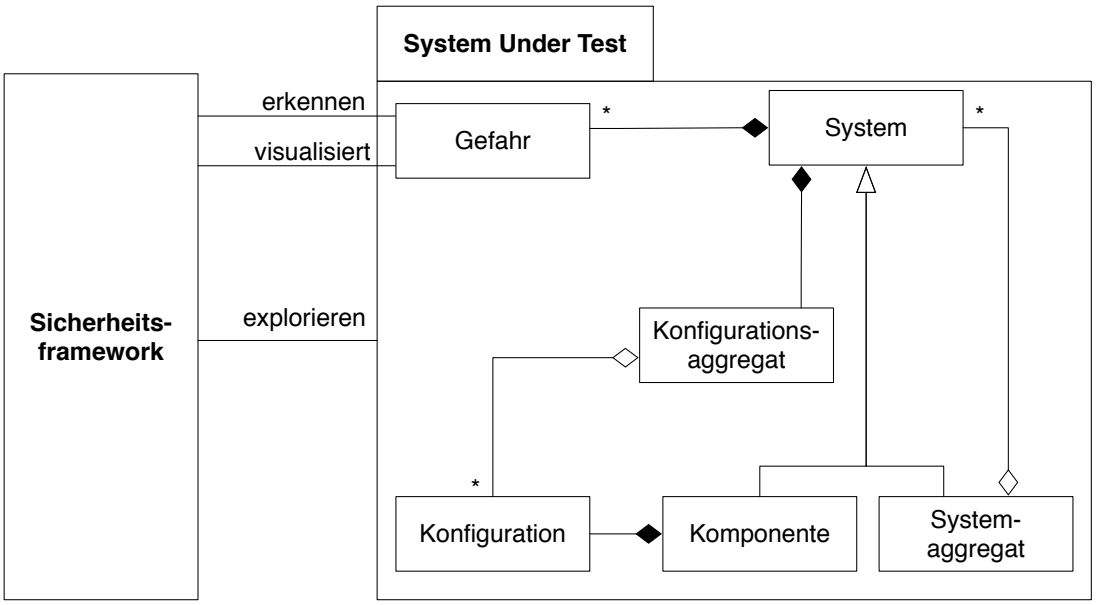


Abb. 6.1: *Erkennung von Gefahren in verteilten Systemen mit CUSTODIAN*

6.1 Anforderungen

Dieser Abschnitt beschreibt die an das Sicherheitsframework gestellten Anforderungen. Es wird zwischen funktionalen Anforderung (**FA#**) und nichtfunktionalen Anforderung (**NFA#**) unterschieden.

FA1: Eine Sicherheitsüberprüfung ist die Ausführung der Menge aller Sicherheitstests. Das Sicherheitsframework soll Sicherheitsüberprüfungen auf der Basis von Veränderungen, Zeit und neuen Erkenntnissen starten können. Bisher konnten die in der Taxonomie der Abbildung 6.2 modellierten Ereignisse als mögliche Auslöser für eine Sicherheitsüberprüfung ausgemacht werden. Prinzipiell sind diese Ereignisse in drei verschiedene Kategorien zu unterteilen. Veränderungen am verteilten System, das Erreichen eines Zeitpunkts und das Gewinnen neuer Erkenntnisse. Das verteilte System kann sich durch eine neue Zusammensetzung der Komponenten oder durch die Veränderung der Konfiguration einer Komponente, welche eine veränderte KUT zur Folge hat, ändern. Ein Erkenntnisgewinn kann durch die neue Gefahrenklassen in der MTF oder neue Erkenntnisse von Spezialis-

ten eintreten. Das Sicherheitsframework soll auf diese Ereignisse reagieren und eine Sicherheitsüberprüfung auslösen können. Sicherheitsverantwortliche werden dadurch in der Lage versetzt, zeitnah auf neue Gefahren zu reagieren.

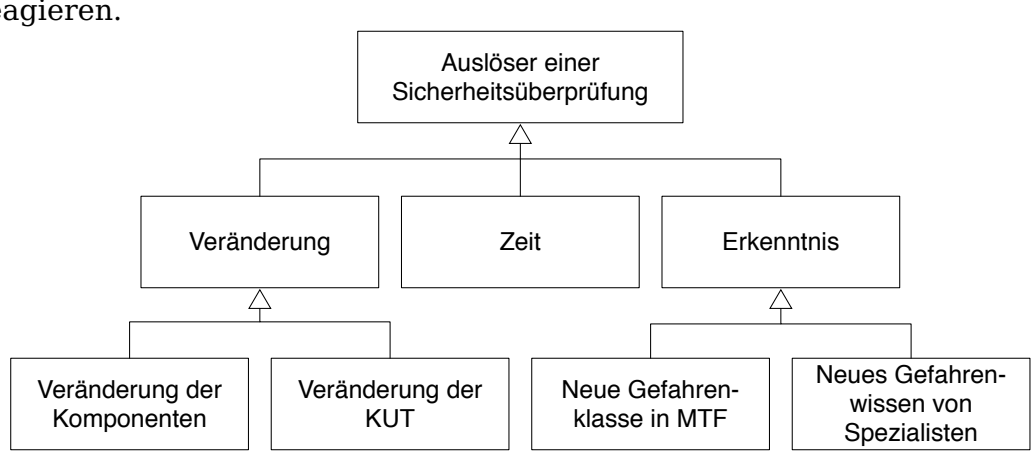


Abb. 6.2: *Taxonomie der Auslöser einer Sicherheitsüberprüfungen*

NFA1: Die in der Abbildung 6.2 aufgezeigten Ereignisse umfasst wohlmöglich nicht alle denkbare Auslöser für Sicherheitsüberprüfungen. Auch hier soll, basierend auf der Open World Assumption, auf neue Ereignisse reagiert werden können. Deshalb ist eine Erweiterbarkeit um zusätzliche Auslöser der Sicherheitsüberprüfung gewünscht.

FA2: Das Sicherheitsframework soll die Konfigurationen der Komponenten eines verteilten Systems einlesen können. Wie in Abschnitt 3.2 dargelegt, verfolgt das Sicherheitsframework eine White-Box-Analyse. Für die White-Box-Analyse der Konfiguration einer Komponente muss diese dem interpretierenden System zur Verfügung stehen. Das erfordert das Einlesen der Konfigurationen der Komponenten.

NFA2: Das Sicherheitsframework soll es ermöglichen, Konfigurationen in ein spezielles Analysemodell zu überführen. Manche Komponenten weisen eine für die Sicherheitsbeurteilung ungeeignete Struktur auf (siehe Abschnitt 2.4.5). Das Sicherheitsframework soll es ermöglichen, ungeeignet strukturierte Konfigurationen in geeignete Strukturen zu überführen.

FA3: Veränderungen der Konfiguration sollen bei der Sicherheitsbeurteilung nachvollziehbar sein. Das Sicherheitsframework soll alle historischen Zustände der KUT speichern und zur Exploration bereitstellen.

NFA3: Die Komposition der Komponenten verteilter Systeme verändern sich. Das Sicherheitsframework soll auf neue, aus dem System entfernte und ausgetauschte Komponenten reagieren können.

NFA4: Das Sicherheitsframework soll um neue Gefahrenklassen erweiterbar sein. Aufgrund der Open World Assumption muss stets auf neue Gefahrenklassen reagiert werden können.

FA4: Das Sicherheitsframework soll es ermöglichen, Gefahren einer Gefahrenklasse individuell zu visualisieren. So soll eine individuell angepasste Kommunikation von Gefahren ermöglicht werden.

FA5: Das Sicherheitsframework soll Gefahren bekannter Gefahrenklassen automatisiert erkennen. Die Erkennung soll mit Hilfe der durch Spezialisten erstellten Gefahrenexperten durchgeführt werden. Die Sicherheitsexperten sollen nicht regelmäßig mit der Gefahrenerkennung der gleichen Gefahrenklasse beschäftigt werden. Deshalb soll das Sicherheitsframework Gefahren bekannter Gefahrenklassen ohne Hilfe der Sicherheitsexperten erkennen können.

FA6: Das Sicherheitsframework soll es Sicherheitsexperten ermöglichen, Sicherheitstests zur Erkennung von Gefahren zu erstellen. In Abschnitt 2.4.2 ist beschrieben, wie Sicherheitstests bei der Gefahrenerkennung eingesetzt werden. Das Sicherheitsframework soll eine graphische Oberfläche bereitstellen, mit der neue Sicherheitstests definiert werden können.

FA7: Das Sicherheitsframework soll Sicherheitsexperten automatisiert über neue Gefahren informieren. Die Verantwortlichen sollen auf neu erkannte Gefahren unverzüglich reagieren können.

NFA5: Das Wissen über Gefahren ermöglicht es Angreifern, Schwachpunkte des verteilten Systems auszunutzen. Darum muss das CUSTODIAN Daten verschlüsselt versenden können.

FA8: Das Sicherheitsframework soll die Navigation zwischen zusammengehörigen Konfigurationen ermöglichen. Im semantischen Web werden Informationen verknüpft, um diese verständlich zu machen [BLHL+01]. Verknüpfte Informationen sind dann nicht nur für Maschinen, sondern auch

für Menschen besser verständlich [MWC04]. Das Sicherheitsframework soll diesen Effekt ermöglicht, indem es einen Browser zum Navigieren in der KUT bereitstellt.

NFA6: Das Sicherheitsframework soll eine große Anzahl von Gefahren durch geeigneten Aggregationen verständlich visualisieren. Bei den Arbeiten in den Fallstudien gab es Gefahrenklassen die über 19.000 Gefahren in einem verteilten System auswiesen. Es zeigte sich, dass eine reine Auflistung der Gefahren nicht ausreichend ist. Eine aggregierte Darstellung der Gefahren soll die Verständlichkeit verbessern.

6.2 Analyse

Eine zentraler Anspruch an die Architektur des Sicherheitsframeworks ist die Gefahrenerkennung, die mehrere funktionalen Anforderungen adressiert (siehe Abschnitt 6.1). Eine Hypothese dieser Arbeit ist die Open World Assumption (OWA), deshalb muss das Framework erweiterbar sein. Das Anwendungsfalldiagramm der Abbildung 6.3 zeigt die für die Exploration der Konfiguration Unter Test (KUT) notwendigen Anwendungsfälle.

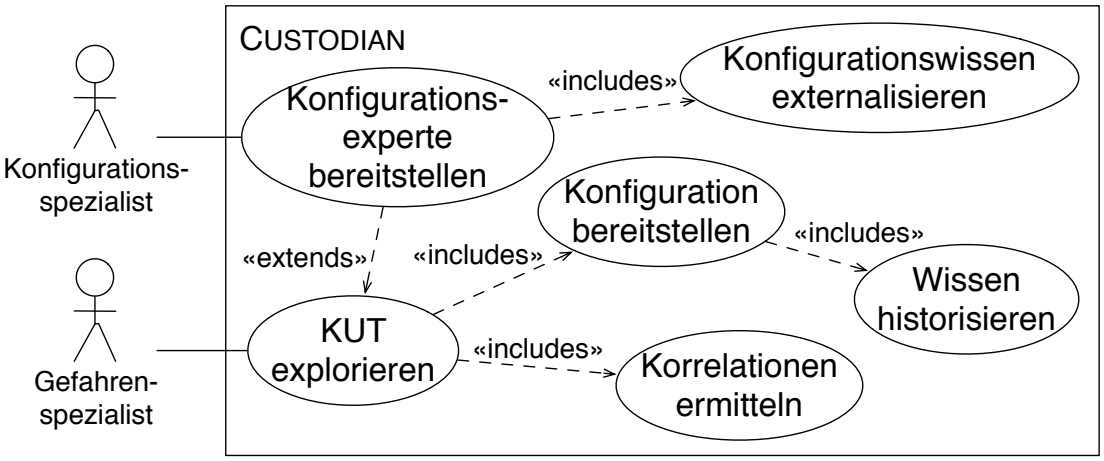


Abb. 6.3: *Anwendungsfälle für das Explorieren der Konfiguration Unter Test*

Ein Gefahrenspezialist kann nur jene KUT explorieren, die Sicherheitsexperten dem Sicherheitsframework bereits bekanntgegeben haben. Der Anwendungsfall **Konfigurationsexperte bereitstellen** erweitert genau dann den Anwendungsfall **KUT explorieren** wenn eine benötigte Konfiguration dem Sicherheitsframework noch nicht vorliegt. Spezialisten erstellen einen Konfigurationsexperten, indem sie ihr Konfigurationswissen (Wissen über die Konfiguration einer Komponente) in Form eines Programmes externalisieren. Nach der Bereitstellung eines Konfigurationsexperten kann dieser dem Sicherheitsframework daraufhin immer die aktuelle **Konfiguration bereitstellen**. Damit die Evolution der bereitgestellten Konfiguration explorierbar ist, muss das Sicherheitsframework das bereitgestellte **Wissen historisieren**. Um den Analysten beim Nachvollziehen von Zusammenhängen in der KUT zu unterstützen, muss das Sicherheitsframework für das bereitstehende Wissen die **Korrelationen ermitteln**.

Ein Gefahrenspezialist muss die **KUT explorieren**, um eine **Gefahrenklasse definieren** zu können (siehe Abbildung 6.4). Hierfür muss der Gefahrenspezialist das für die Gefahrenklasse relevante **Gefahrenwissen verbinden**. Auf der Basis des verbundenen Gefahrenwissens kann der Gefahrenspezialist einen **Gefahrenexperten bereitstellen**. Ein Gefahrenexperte ist ein Programm, welches in der Lage ist, die KUT auf Gefahren der Gefahrenklasse zu analysieren und diese Gefahren dem Sicherheitsframework bekanntzugeben.

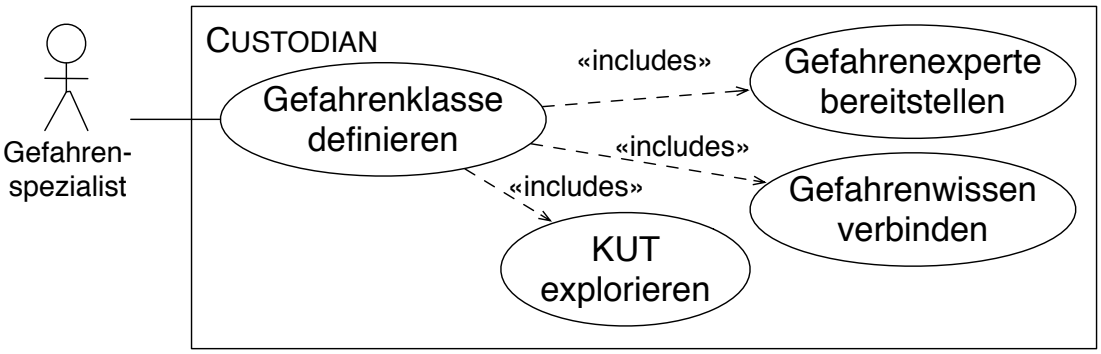


Abb. 6.4: *Anwendungsfälle für das Explorieren von Gefahren*

CUSTODIAN kann mit Hilfe der **bereitgestellten Konfiguration** die Gefahren einer **definierten Gefahrenklasse** erkennen (siehe Abbildung 6.5). Um die **Gefahren zu erkennen**, muss CUSTODIAN die Sicherheitsanalyse auslösen, die den Gefahrenexperten anstößt. Benutzer können daraufhin die **Gefahren explorieren**, die CUSTODIAN mit der Hilfe des Gefahrenexperten erkannte.

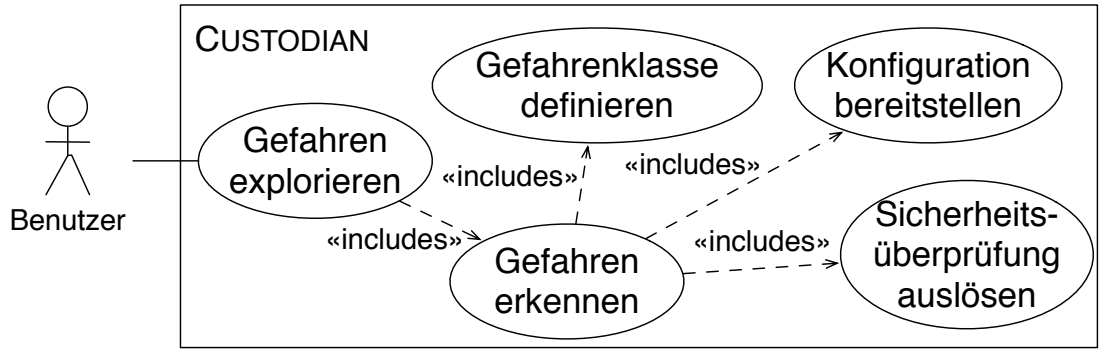


Abb. 6.5: *Anwendungsfälle zur Erkennung von Gefahren*

CUSTODIAN kann für bereits **erkannte Gefahren** Visualisierungen erstellen (siehe Abbildung 6.6). Den Benutzern von CUSTODIAN helfen diese Visualisierungen beim Verstehen der Gefahren. Um Visualisierungen für Gefahren erstellen zu können, muss ein Visualisierungsspezialist für die Gefahrenklassen einen geeigneten **Visualisierungsexperten bereitstellen**.

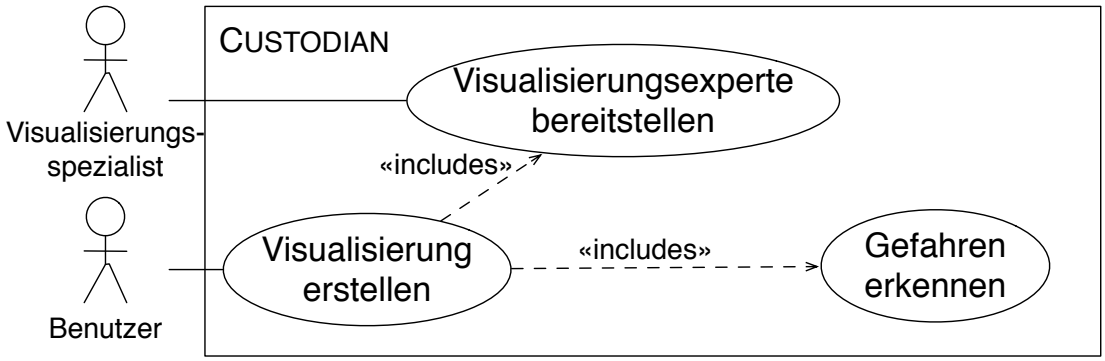


Abb. 6.6: *Anwendungsfälle für die Visualisierung von Gefahren*

CUSTODIAN hilft mit von Spezialisten externalisiertem Wissen bei der Gefahrenerkennung. Hierfür können verschiedene Spezialisten verschiedenes Gefahrenwissen bereitstellen (siehe Abbildung 6.7). Dadurch ist CUSTODIAN um Wissen zu Konfigurationen, zu Visualisierung, zur Gefahrenerkennung und Ereignissen erweiterbar.

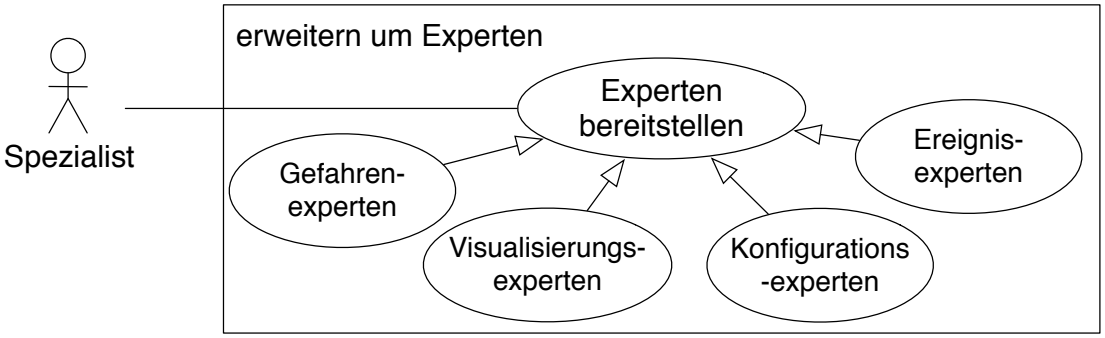


Abb. 6.7: *Die Spezialfälle des Anwendungsfalls „erweitern um Experten“*

6.2.1 Depot

Das Sicherheitsframework basiert auf dem Entwurfsmuster des Depots und kapselt seine Funktionalität durch eine Fassade. Sowohl Sicherheitsexperten als auch die Steuereinheit greifen ausschließlich über diese Fassade auf das Depot zu. Diese Fassade ist in Abbildung 6.8 modelliert. Über sie können neue Wissensklassen registriert (siehe Abschnitt 6.2.1.2), Wissen bereitgestellt (Abschnitt 6.2.1.3) und Wissen abgefragt (Abschnitt 6.2.1.4) werden. Die Fassade des Depots bietet Zugang zu komplexen Funktionalitäten über einfache Methoden an. Dadurch wird beispielsweise die komplexe Historisierung (Abschnitt 6.2.1.3) für den Zugreifer abstrahiert.

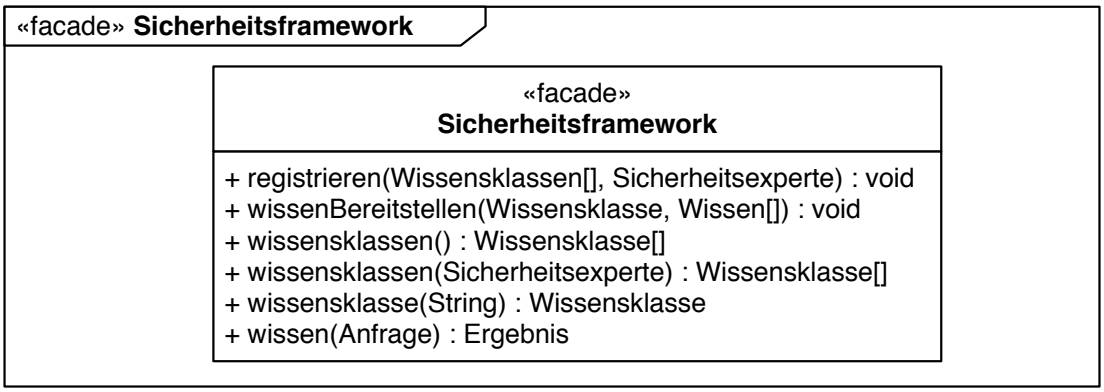


Abb. 6.8: *Definition der Fassade des Depots*

6.2.1.1 Fassade

Die Fassade des Depots ist von der konkreten Implementierung des Depots unabhängig. Diese Unabhängigkeit wird über die Anwendung des Strategiemusters erreicht wie in Abbildung 6.9 modelliert. Die Klasse Depotfassade ist eine Adapterklasse, welche über das einzusetzende Depot entscheidet. Als Adapterklasse bietet die Depotfassade bequemen Zugriff, indem all ihre Methoden statisch implementiert und somit ohne Erzeugung eines Objekts erreichbar sind.

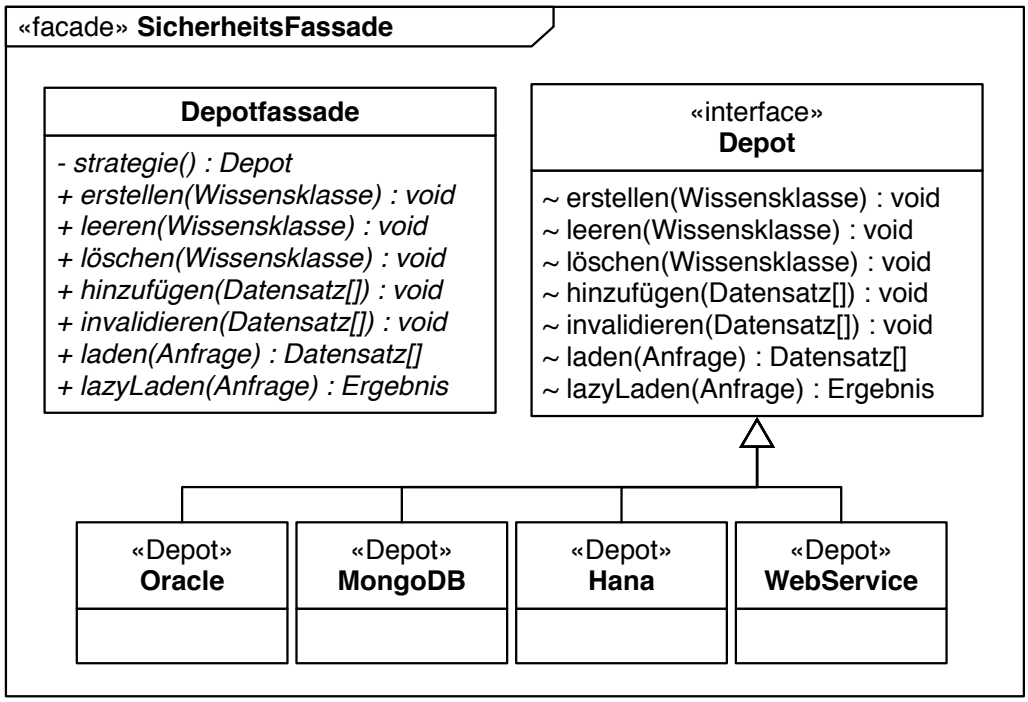


Abb. 6.9: *Die Schnittstelle Depot mit vier möglichen Implementierungen und deren Zugriffsfassade Depotfassade*

Das Sicherheitsframework ist in der Lage, das Depot mit unterschiedlichen Depot-Implementierungen zu betreiben. Für eine konkrete Instanz des Sicherheitsframeworks kommt eine konkrete Implementierung eines Depots zum Einsatz. Die Depotfassade stellt den Kontext im Sinne des Strategiemusters dar. Sie entscheidet, welche Strategie, also welche konkrete Implementierung eines Depots verwendet wird. Jede Implementierung muss die Schnittstelle Depot erfüllen.

Die Schnittstelle Depot definiert Methoden zum Erstellen, Leeren und Löschen von Wissensklassen. Diese Methoden entsprechen der Data Definition Language (DDL) der Structured Query Language (SQL). Die weiteren Methoden zum Hinzufügen, Invalidieren und Laden von Wissen entsprechen hingegen der Data Manupulation Language (DML). Klassisches synchrones Laden erlaubt die Abfrage aller Datensätze. Für den effizienten Umgang mit großen Datenmengen kommt das Entwurfsmuster des Lazy Loadings, wie er in Abschnitt 6.2.1.4 genauer erklärt ist, zum Einsatz.

Das Depot verwaltet die registrierten Wissensklassen (siehe Abbildung 6.10). Jede Wissensklasse hat eine eindeutige `id` und ein `name`, der die Wis-

sensklasse beschreibt. Die Wissensklassen enthalten die Datensätze, die von Sicherheitsexperten in das Depot eingestellt wurden.

Das Depot ist für die Historisierung der Datensätze verantwortlich. Jeder Datensatz beinhaltet in einem Feld die Nutzdaten, die durch die zugehörige Wissensklasse definiert werden (siehe Abbildung 6.10). Zusätzlich hält jeder Datensatz die Attribute `id`, `hash`, `gültigVon` und `gültigBis`, mit denen das Depot die Historisierung steuert.

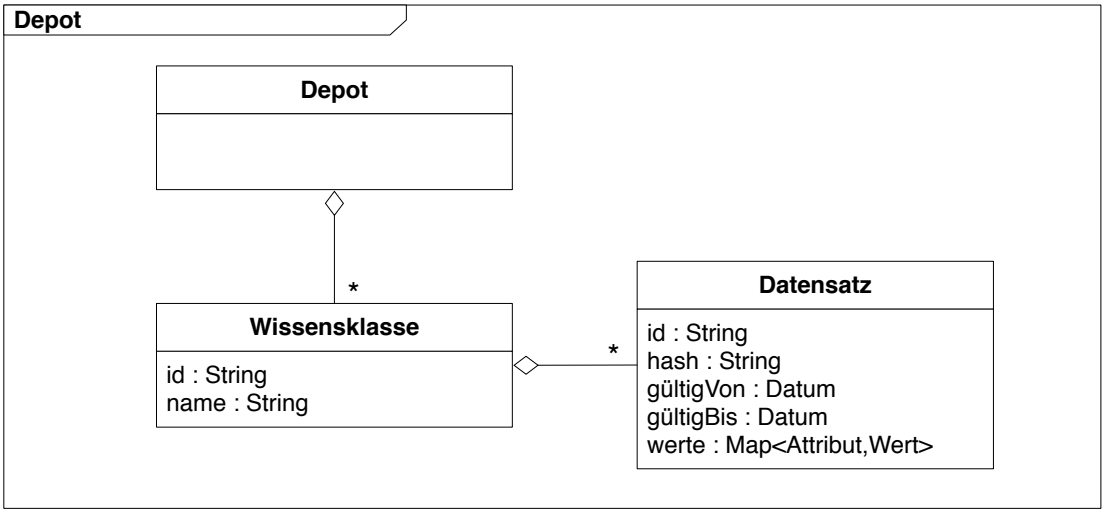


Abb. 6.10: *Das Datensatz-Modell des Depots*

6.2.1.2 Registrieren von Wissensklassen

Das Depot stellt, wie in Abbildung 6.11 modelliert, die Methode `registrieren()` über seine Fassade bereit. Die Fassade des Depots ruft hierfür auf der konkreten Implementierung des Depots die statische Methode `erstellen()` auf. Das Depot ermittelt über `strategie()` die verwendete Implementierung des Depots und ruft auf dieser die `erstellen()`-Methode auf. Die darauf folgenden Aufrufe sind von der konkreten Implementierung des Depots abhängig. Am Beispiel Oracle wird eine `CREATE TABLE` - Anweisung an die Oracle-Datenbank gesendet. Alle Anweisungen erfolgen synchron, und der Aufrufer des Depots wird über die abgeschlossene Registrierung benachrichtigt.

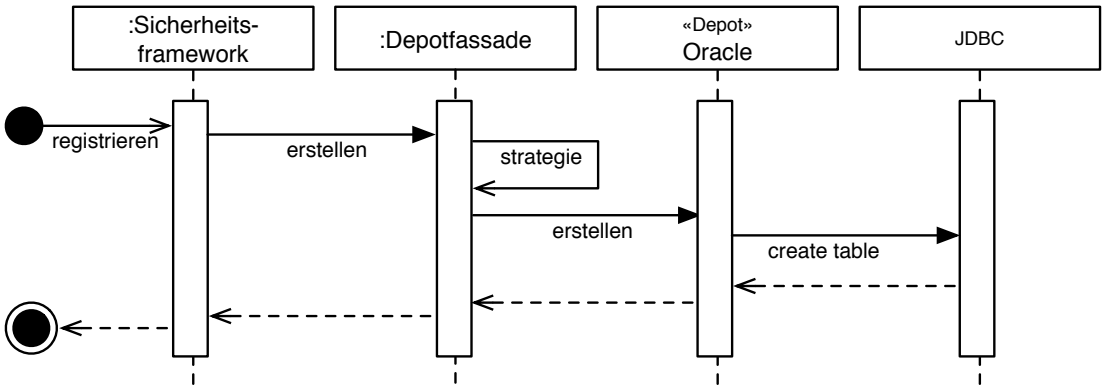


Abb. 6.11: *Registrierung einer Wissensklasse am Depot am Beispiel einer Oracle-basierten Implementierung*

6.2.1.3 Bereitstellen und Historisieren von Wissen

Für das Bereitstellen von Wissen stellt das Depot einen Mechanismus zur Historisierung bereit. Dieser Historisierungsmechanismus erlaubt es, auf jeden historischen Wissensstand des Depots zuzugreifen. Abbildung 6.12 beschreibt diesen Mechanismus am Beispiel einer Oracle-Implementierung des Depots. Initial wird dem Depot über `wissenBereitstellen()` der aktuelle Wissensstand zu einer Wissensklasse übergeben. Daraufhin wird der aktuell im Depot existierende Stand dieses Sicherheitsexperten geladen. Über die private Methode `abgleichen()` ermittelt das Depot die Menge der neuen Datensätze und die der nicht mehr existierenden Datensätze.

Neue Datensätze werden über die Depotfassade mit der Methode `hinzufuegen()` erstellt. Bei jedem Datensatz wird das Attribut `gueltigVon` auf den aktuellen Zeitpunkt gesetzt, `id` wird mit einer eindeutigen Bezeichnung versehen und in `hash` wird ein Hash-Wert über die Werte der Map `werte` gespeichert. Durch die eigenen Identifizierer ist es möglich, zwei inhaltlich identische Datensätze zu unterscheiden und deren Identität festzustellen. Hierzu wird ein `INSERT` - Befehl an den JDBC-Treiber abgesetzt. Die Menge der nicht mehr existierenden Datensätze werden über die Methode `invalidieren()` invalidiert. Für das invalidieren wird `gueltigBis` der Datensätze auf den aktuellen Zeitpunkt gesetzt und mit einen `UPDATE` - Befehl an den JDBC-Treiber geschickt.

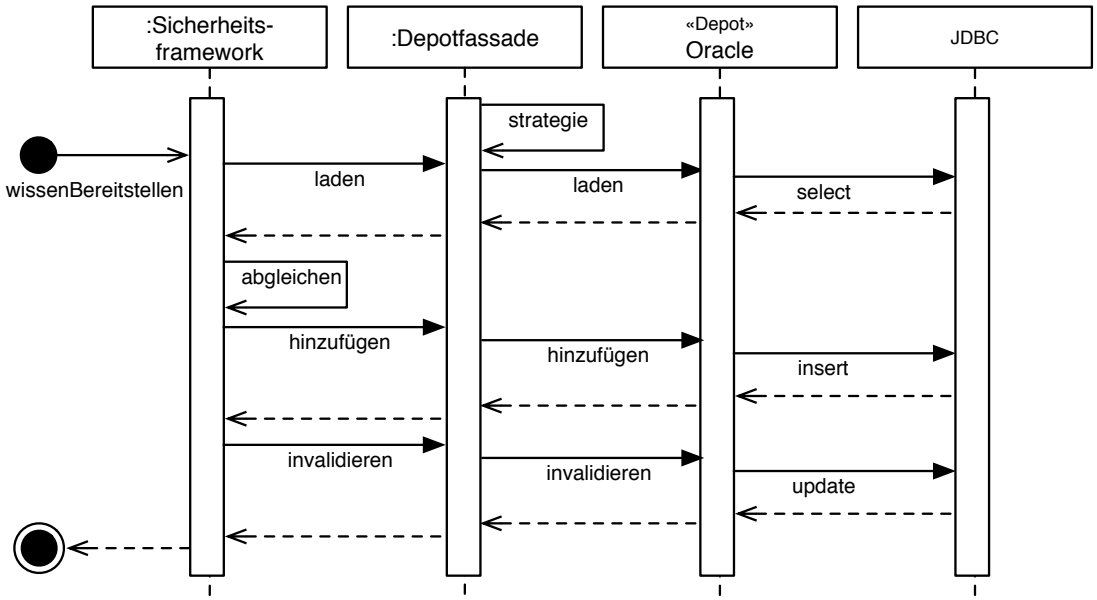


Abb. 6.12: *Bereitstellen und Historisieren von Wissen am Beispiel eines Oracle-basierten Depots*

6.2.1.4 Abfragen von Wissen

Die Schnittstelle für das Anfragen von Wissen ermöglicht erst die Kommunikation zwischen den Sicherheitsexperten. Hierfür steht das in Abbildung 6.13 modellierte Anfragemodell zur Verfügung. Eine Anfrage adressiert immer eine Wissensklasse auf dem Depot, ein Datum und eine Anfrageart. Die Anfrageart `Stand` führt zur Rückgabe des Wissensstands der Wissensklasse zum angefragten Datum. Die Anfrageart `Erstellt` gibt die an dem Datum neu erstellten Datensätze, und die Anfrage `Invalidiert` gibt die seit dem Datum nicht mehr gültigen Datensätze zurück. Filter können das Ergebnis einer Anfrage einschränken und beziehen sich auf ein Attribut, indem sie einen Filterwert definieren. Der Filtertype eines Filters legt fest, wie der Filterwert mit dem Wert des Attributs verglichen wird. Der Filtertype `Equal` filtert beispielsweise alle Datensätze, bei dem der Attributwert exakt dem Filterwert entspricht, und der Filtertyp `Contains` filtert die Datensätze, bei denen der Filterwert ein Teilwort des Attributwerts ist.

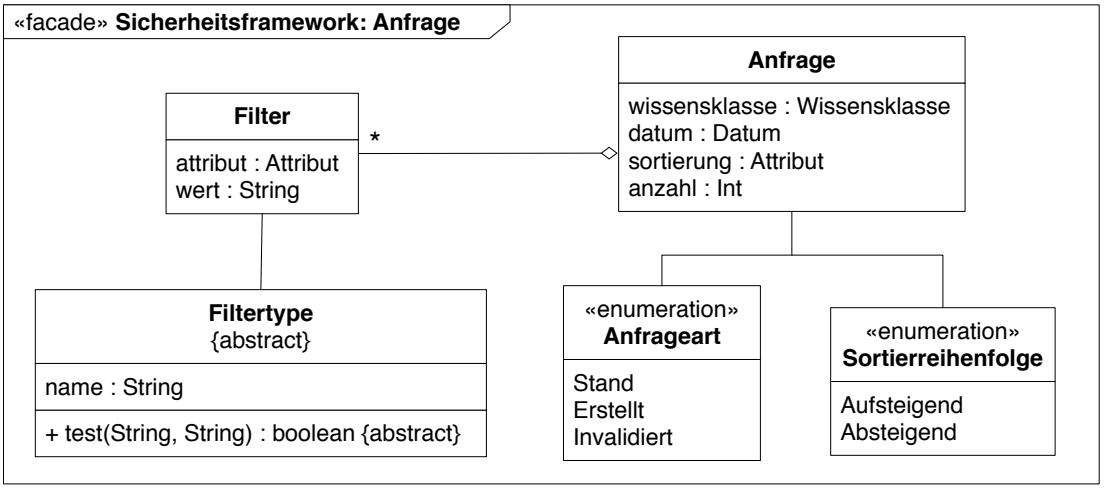


Abb. 6.13: *Modell der Klasse Anfrage und ihrer referenzierten Klassen*

Über den Parameter `anzahl` des Anfrageobjekts kann ein Lazy-Loading-Mechanismus gesteuert werden. Dieser basiert auf der lazy initialization wie sie Fouler beschreibt [Fow02]. Das Setzen dieses Parameters bestimmt, wie viele Datensätze geladen werden sollen. Das Verändern des Parameters führt zu einem Nachladen der gewünschten Datensätze. Das erlaubt es, nur die Anzahl von Datensätze anzufragen, die tatsächlich benötigt werden. Hiervon können nicht die Sicherheitsexperten bei ihren Analysen, sondern auch Spezialisten beim Explorieren der KUT profitieren.

Das Resultat einer Anfrage ist das Objekt Ergebnis, welches eine Liste von Datensätzen hält. Diese Liste ist observierbar, und der Empfänger kann so über geladene Datensätze notifiziert werden. Jeder Datensatz beinhaltet in einem Feld die Werte der Attribute, die von der angefragten Wissensklasse definiert werden. Des Datensatzobjekt ist in Abschnitt 6.2.1.1 (Depot) detailliert beschrieben.

Abb. 6.14: *Modell der Klasse Ergebnis und ihrer referenzierten Klassen*

Das Depot ist eine Datenbank zum Austausch aller bisher bekannten Konfigurationsdaten sowie aller bei Analysen erstellten Ergebnisse. Der Wissens-

austausch zwischen den Sicherheitsexperten findet über das Depot statt. Dies gewährleistet die Modularität und entkoppelt die Sicherheitsexperten. Neben dem reinen Speichern und Bereitstellen von Daten historisiert das Depot auch. Dadurch können die Sicherheitsexperten auch auf alte Datenstände oder bereits invalidierte Hypothesen zugreifen. Das ermöglicht nicht nur die historische Beurteilung der Sicherheitskonfiguration.

Zur Sicherheitsbeurteilung von Konfigurationen in verteilten Systemen müssen unterschiedliche Datenquellen und Gefahrenwissen mehrerer Domänen erfasst und vereint werden. Vor allem jene Gefahrenklassen, die aus der Interaktion der Komponenten eines verteilten Systems hervorgehen, benötigen zur Erkennung das Gefahrenwissen mehrerer Personen. Auch das Auffinden der Instanzen dieser Gefahrenklassen bedarf häufig der Zusammenarbeit mehrerer Experten.

6.2.2 Sicherheitsexperten

Es ist möglich, Sicherheitsexperten dynamisch zur Laufzeit in das Sicherheitsframework einzubinden. Hierfür definiert das Sicherheitsframework die in Abbildung 6.15 modellierte Schnittstelle, welche von den einzubindenden Sicherheitsexperten implementiert werden muss. Diese Schnittstelle ist im Kontext der eingesetzten Plugin-Architektur das „Separated Interface“ [Fow02].

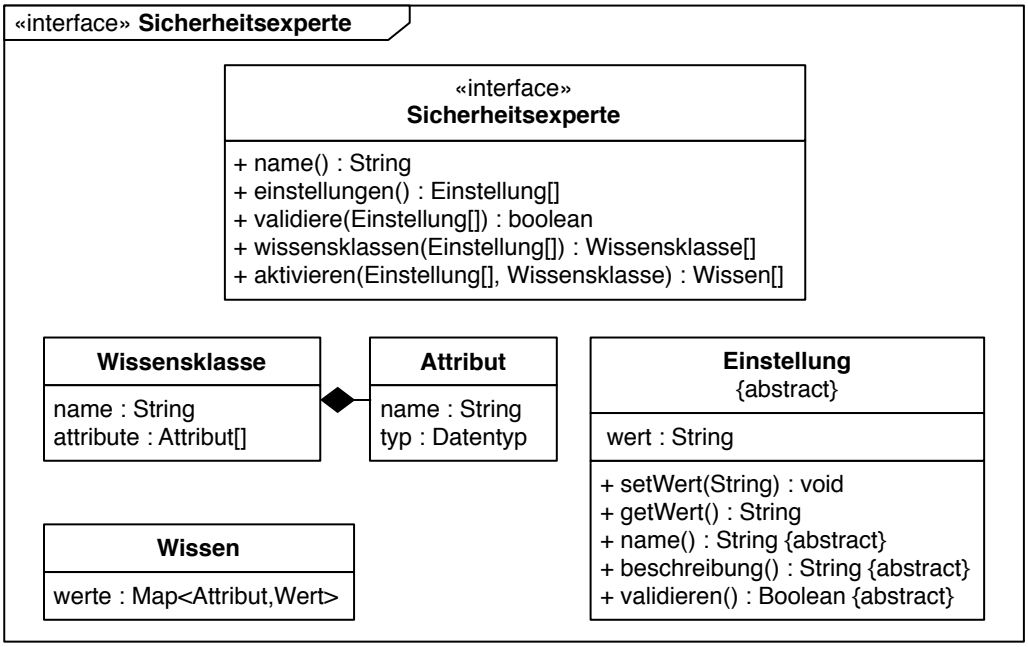


Abb. 6.15: *Definition der Schnittstellen des Sicherheitsframeworks für Sicherheitsexperten inklusive ihrer Schnittstellenobjekte*

Die Schnittstelle Sicherheitsexperte definiert alle Methoden, auf die das Sicherheitsframework zugreift. Der Sicherheitsexperte wird dem Nutzer mit dem `name()` angezeigt. Die Methoden `einstellungen()`, `validiere()` und `wissensklassen()` werden bei der Registrierung des Sicherheitsexperten am Sicherheitsframework benötigt. Diese Registrierung ist in Abschnitt 6.2.2.1 genauer erläutert. Über Methode `stand()` fragt das Sicherheitsframework den aktuellen Wissensstand des Sicherheitsexperten ab. Die Abfrage von Sicherheitsexperten ist in Abschnitt 6.2.2.2 beschrieben.

6.2.2.1 Registrierung von Sicherheitsexperten

Sicherheitsexperten müssen am Sicherheitsframework registriert werden, um auf dem Depot arbeiten zu können. Bei dieser Registrierung wird der Sicherheitsexperte für den konkreten Einsatz im Sicherheitsframework konfiguriert. Zusätzlich gibt der Sicherheitsexperte die Struktur des von ihm publizierten Wissens bekannt. Mit diesen Strukturinformationen kann das Depot für den Einsatz des Sicherheitsexperten präpariert werden.

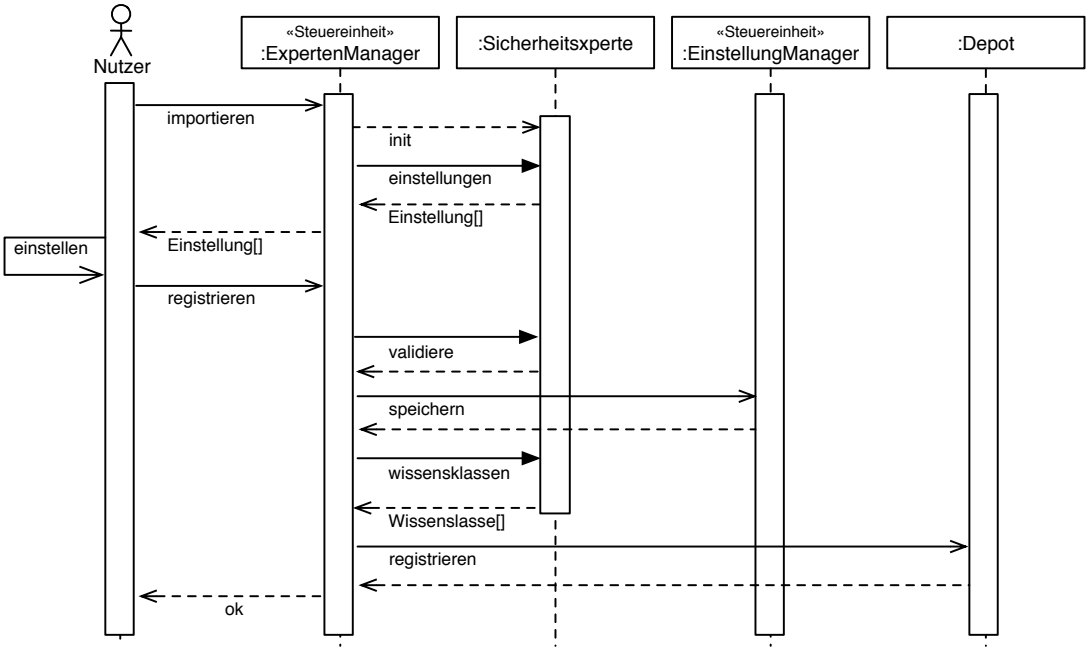


Abb. 6.16: *Registrierung von Sicherheitsexperten am Sicherheitsframework*

In Abbildung 6.16 ist der gesamte Prozess der Registrierung modelliert. Zuerst bindet der Nutzer das Plugin, welches einen Sicherheitsexperten beinhaltet, über Expertenmanager ein. Der Expertenmanager initialisiert daraufhin den Sicherheitsexperten und erfragt die von diesem geforderten Einstellungen, um diese den Nutzer zurückzugeben. Der Nutzer füllt die Einstellungen entsprechend aus und löst daraufhin die Registrierung des Sicherheitsexperten über den Expertenmanager aus. Der Expertenmanager validiert die vom Benutzer vorgenommenen Einstellungen und speichert diese (bei Validität) über den Einstellungsmanager.

In Abbildung 6.17 werden exemplarische Einstellungen für einen LDAP-Experten vorgestellt. Daraufhin erfragt der Expertenmanager am Sicher-

heitsexperten die von ihm bereitgestellten Wissensklassen. Der Expertenmanager übergibt diese Wissensklassen daraufhin dem Depot zur Präparation.

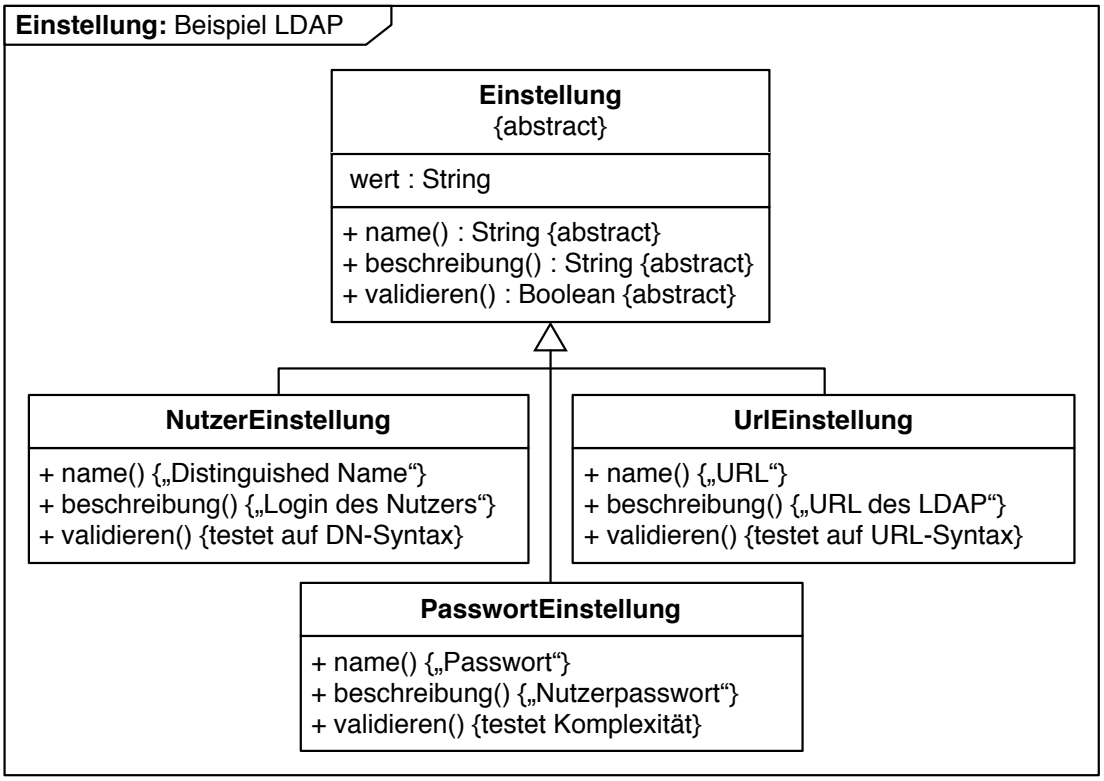


Abb. 6.17: *Definition der Schnittstellen des Sicherheitsframeworks für Sicherheitsexperten inklusive der von der Schnittstelle referenzierten Klassen*

Die exemplarischen, für einen LDAP-Sicherheitsexperten benötigten Einstellungen Nutzer, URL und Passwort sind in Abbildung 6.17 modelliert. Für jede der drei Einstellungen definiert der Sicherheitsexperte eine eigene Klasse, die von der abstrakten Klasse Einstellung des Sicherheitsframeworks abgeleitet ist. Das erlaubt dem Entwickler des Sicherheitsexperten, eine individuelle Validierung der von den Benutzern eingestellten Werten für die Einstellungen vorzunehmen. Das ist notwendig, weil den verschiedenen Einstellungen unterschiedliche Syntaxen zugrundeliegen können. Das ermöglicht es dem Framework, dem Nutzer mitzuteilen, welche der vorgenommenen Einstellungen potenziell falsch sind.

6.2.2.2 Bereitstellen von Wissen auf dem Depot

Ein Sicherheitsexperte kann erst dann sein Gefahrenwissen in Form von Wissen auf dem Depot bereitstellen, wenn er vorher, wie in Abschnitt 6.2.2.1 beschrieben, registriert wurde. Die möglichen Auslöser einer Bereitstellung werden in Abschnitt 6.2.3 beschrieben.

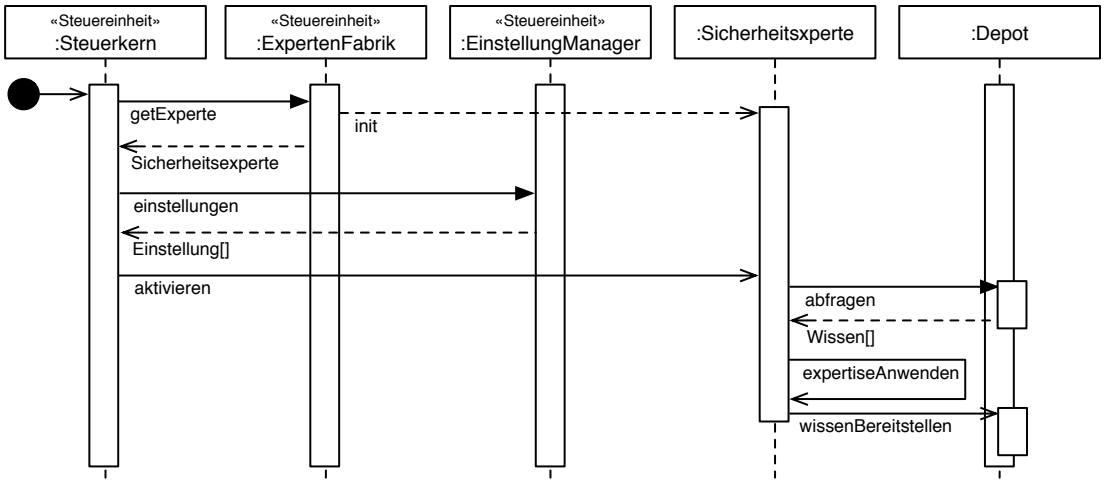


Abb. 6.18: *Bereitstellen von Wissen auf dem Depot durch einen Sicherheitsexperten*

Dem Steuerkern wird beim Aufruf (siehe Abbildung 6.18) der eindeutige Klassenname des Sicherheitsexperten sowie die gefragte Wissensklasse übergeben. Die Instanziierung des Sicherheitsexperten erfolgt unter der Anwendung des „Basic Factory Patterns“, wie Fowler es für Plugin-Architekturen vorschlägt [Fow02]. Nachdem der Steuerkern die Instanz des Sicherheitsexperten von der Expertenfabrik erhalten hat, lädt er die dazugehörigen Einstellungen über den Einstellungsmanager. Daraufhin aktiviert der Steuerkern den Sicherheitsexperten unter Übergabe der Einstellungen und der gefragten Wissensklasse asynchron. Der Sicherheitsexperte fragt vom Depot das für ihn notwendige Wissen ab und erstellt unter Anwendung seines Gefahrenwissen das sich aus der aktuellen Situation ergebende Wissen. Wie Wissen vom Depot abgefragt werden kann, beschreibt Abschnitt 6.2.1.4. Dieses Wissen stellt wer asynchron auf dem Depot bereit (beschrieben in Abschnitt 6.2.1.3).

Das Depot benötigt für das Erkennen von Gefahren in der Konfiguration eines verteilten Systems unterschiedliche Sicherheitsexperten, die als Sicherheitsexperten ihr Gefahrenwissen bereitstellen. Jeder Sicherheitsex-

perte ist ein eigenständiges Subsystem, welches eine bestimmte Aufgabe lösen kann, jedoch bei der Sicherheitsbeurteilung nur einen Teil der Lösung beitragen kann. Erst durch das gemeinsame Bereitstellen ihres Gefahrenwissens auf dem Depot wird dieses komplexe Problem gelöst. Die Sicherheitsexperten kommunizieren ausschließlich mit dem Depot und nicht untereinander. Dieses Konzept entkoppelt die einzelnen Lösungsschritte und macht deren Interaktion von außen steuerbar. Daher können Sicherheitsexperten sowohl unabhängig voneinander von verschiedenen Spezialisten entwickelt werden als auch parallel arbeiten.

6.2.2.3 Sicherheitstests

CUSTODIAN soll die Gefahrenerkennung auf der Basis von Sicherheitstests erkennen (siehe Abschnitt 2.4.2). Sicherheitstests gehen heuristisch vor und generieren Hypothesen zu Gefahren. Gefahrenhypothesen können aufgrund der heuristischen Natur der Sicherheitsexperten inkorrekt sein. Deshalb ist ein Mechanismus zur Invalidierung inkorrekter Gefahrenhypothesen notwendig. Abbildung 6.19 modelliert einen Sicherheitstest, der aus einem Hypothesengenerator und mehreren Hypotheseninvalidatoren besteht. Sowohl der Hypothesengenerator als auch die Hypotheseninvalidatoren basieren auf Heuristiken. Die Hypotheseninvalidatoren können wieder die vom Hypothesengenerator generierten Hypothesen invalidieren. Dieses Konzept ermöglicht es, einen Sicherheitstest bei der Identifikation von inkorrekten Gefahrenhypothesen zu verfeinern.

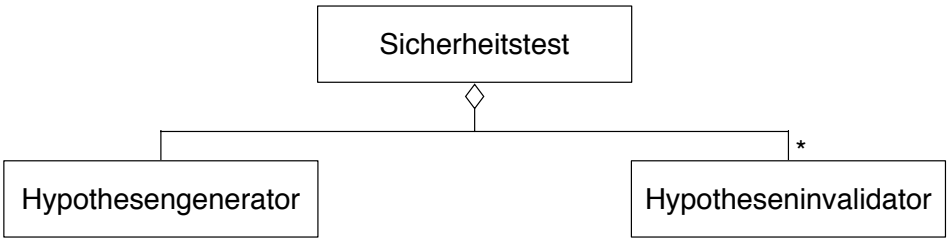


Abb. 6.19: *Aufbau eines Sicherheitstests*

6.2.3 Steuereinheit

Die Steuereinheit bestimmt die Reihenfolge bzw. Parallelität der Abarbeitung verschiedener Sicherheitsexperten. Hierfür prüft die Steuereinheit bei Ereignissen, ob für einen Sicherheitsexperte neue Informationen bereitstehen. Ist dies der Fall, startet die Steuereinheit diesen Sicherheitsexperten in einem eigenen Prozess. Die Abbildung 6.20 zeigt das beispielhaft anhand der Experten `a`, `b` und `c`. Der Ereignisexperte `a` erkennt, dass ein bestimmtes Ereignis eingetroffen ist und gibt das dem Depot bekannt. Das kann beispielsweise das Erreichen eines Zeitpunkts sein. Abbildung 6.2 in Abschnitt 6.1 zeigt eine Taxonomie der möglichen Ereignisse. Das Depot notifiziert die Steuereinheit über die Änderung und startet den Konfigurationsexperten `b`. Der Konfigurationsexperte `b` liest die aktuelle Konfiguration einer Komponente ein und stellt diese auf dem Depot bereit. Das Depot notifiziert die Steuereinheit über die Änderung von `b` und startet den Gefahrenexperten `c`.

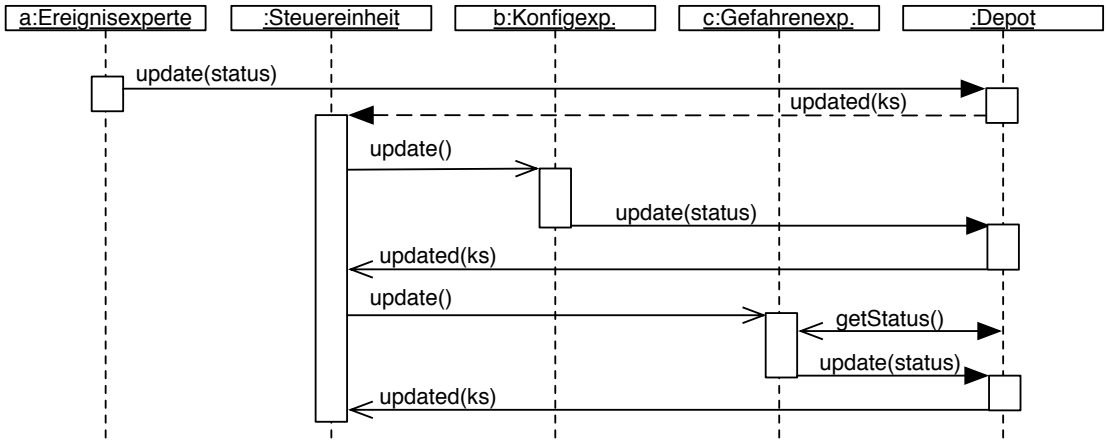


Abb. 6.20: *Prozesssteuerung der Experten auf dem Depot*

Das Sicherheitsframework ist zur Sicherheitsbeurteilung verteilter Systeme konzipiert. Damit ist das System Under Test aus der Sicht des Sicherheitsframeworks eine Instanz eines verteilten Systems, welches auf Gefahren überprüft wird. Ein System Under Test ist, wie in Abbildung 6.21 dargestellt, ein Systemaggregat. Ein System ist entweder eine einfache Komponente oder wiederum ein Systemaggregat, also ein System, welches aus mehreren Systemen besteht. Eine Komponente besitzen eine Konfiguration welche, wie in Abschnitt 2.3 beschrieben, der individuellen Anpassung an den Einsatzzweck dient. Die Konfiguration eines System Under Test ist

das Aggregat der Konfigurationen der einzelnen Komponenten und wird als Konfigurationsaggregat bezeichnet.

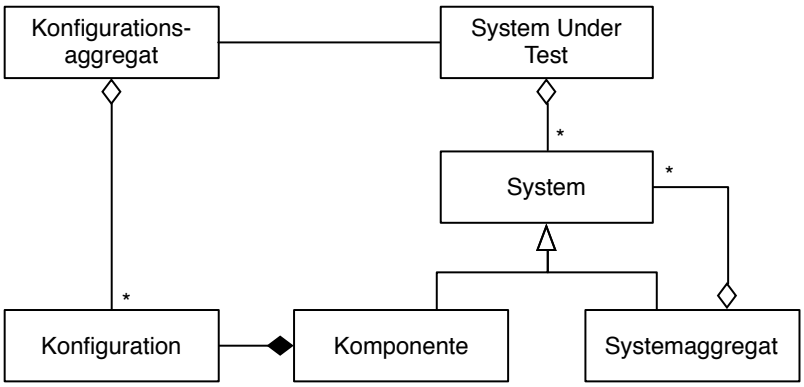


Abb. 6.21: *Modell eines verteilten Systems samt Konfiguration aus der Betrachtungsweise des Sicherheitsframeworks*

Eine rekursive Auflösung des in Abbildung 6.21 beschriebenen Kompositums überführt ein verteiltes System in eine einfache Aggregation konfigurierter Komponenten, die alle Unterkomponenten unmittelbar beinhaltet. Dies ist in Abbildung 6.22 dargestellt. Die konkrete Aggregation der Komponenten führt zu den für das verteilte System relevanten Gefahrenklassen. Unter Einbeziehung der Konfiguration lassen sich die Gefahren ermitteln.

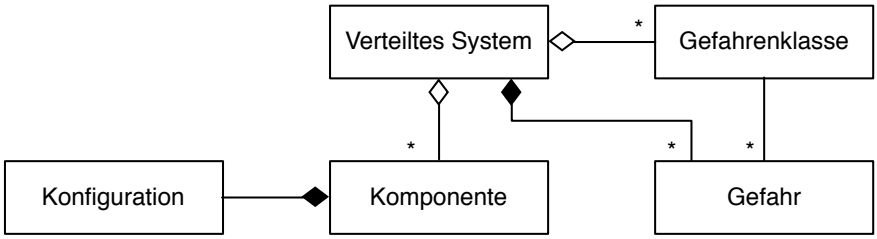


Abb. 6.22: *Vereinfachte Darstellung eines verteilten Systems unter Einbeziehung von Gefahrenklassen und -instanzen*

Zur Gefahrenerkennung kommen im Sicherheitsframework verschiedenartige Sicherheitsexperten zum Einsatz. Sämtliches für die Gefahrenerkennung notwendiges Gefahrenwissen wird dem Sicherheitsframework von Sicherheitsexperten bereitgestellt. Die Taxonomie der Sicherheitsexperten in Abbildung 6.23 zeigt die fundamentalen Arten von Sicherheitsexperten. Aufgrund der Notwendigkeit der Adaption ist die Taxonomie erweiterbar; weitere Sicherheitsexperten können hinzugefügt werden.

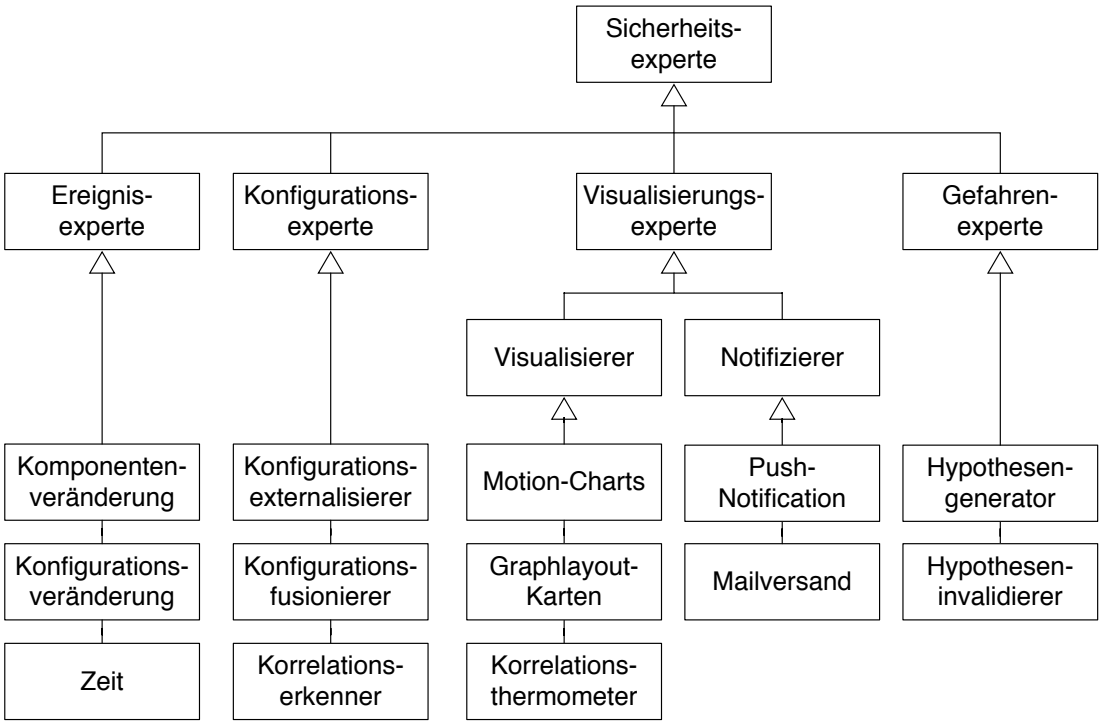


Abb. 6.23: *Taxonomie von Sicherheitsexperten zur Gefahrenerkennung*

Komponentenexperten extrahieren die Konfiguration einer Komponente und stellen diese dem Depot zur Verfügung. Strukturerhaltende Komponentenexperten legen die Konfiguration in der von der Komponente vorgegebenen Struktur auf das Depot. Strukturverändernde Komponentenexperten transformieren die Konfiguration vorher, wie in Kapitel 2.4.5 beschrieben, in eine für die Gefahrenanalyse optimierte Struktur. Ein Analyseexperte führt syntaktische oder semantische Auswertungen der auf dem Depot veröffentlichen Informationen durch. Gefahrenexperten sind auf eine Gefahrenklasse des verteilten Systems spezialisiert. Hypothesengeneratoren erstellen neue Gefahrenhypothesen, und Hypotheseninvalidatoren invalidieren existierende Gefahrenhypothesen.

Neben den Sicherheitsexperten greifen auch Beobachter auf das Depot zu. Beobachter können wie Sicherheitsexperten alle auf dem Depot vorhandenen Erkenntnisse einsehen, sie tragen jedoch keine eigenen Erkenntnisse bei. Das Konzept der Beobachter ist, wie auch das der Sicherheitsexperten, erweiterbar. Prinzipiell kann jedoch zwischen intrinsischen und

extrinsischen Beobachtern unterschieden werden. Extrinsische Beobachter werden von Ereignissen außerhalb des Sicherheitsframeworks ausgelöst und verwenden einen Pull-Mechanismus. Das sind beispielsweise Ansichten (engl. View) wie Tabellen oder Diagramme, die zur Präsentation auf das Wissen des Depots zugreifen. Intrinsische Beobachter werden durch einen Push-Mechanismus aufgrund einer Veränderung des Wissens auf dem Depot ausgelöst. Das können Notifizierer, die via E-Mail oder SMS Verantwortliche informieren, Störungsmelder, die eine Erkenntnis als Störung in einem IT-Incident Management System einstellen, oder Gegenmaßnahmen zur automatischen Beseitigung von Gefahren sein.

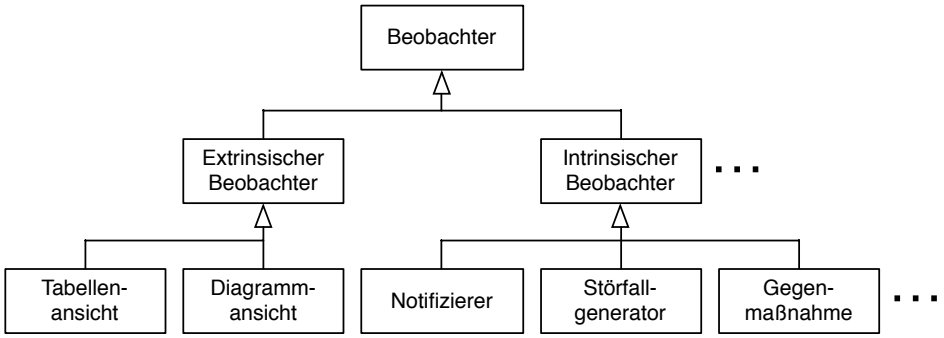


Abb. 6.24: *Taxonomie der Observierer im Sicherheitsframework*

6.2.4 Das Datenmodell

Auf dem Depot bestehen Wissensklassen, die wiederum aus einer Menge von Datensätzen bestehen (siehe Abbildung 6.26). Jede Wissensklasse definiert eine Menge von Attributen, welche in jedem Datensatz der Wissensklasse vorkommen. Das ist am Beispiel der Wissensklasse `Person` mit den Attributen `name` und `geboren` in Abbildung 6.25 modelliert.

Abb. 6.25: *Wissensklassen und deren Datensätze in CUSTODIAN*

Die von den Konfigurationsexperten bereitgestellten Konfigurationsklassen werden als Konfigurationsobjekte auf dem Depot bereitgestellt. Die von Ge-

fahrenexperten erkannten Gefahren der Gefahrenklasse existieren als Gefahrenhypothesen auf dem Depot. Ereignisexperten können Ereignisse bestimmter Ereignisklassen erkennen und diese auf dem Depot bekanntgeben.

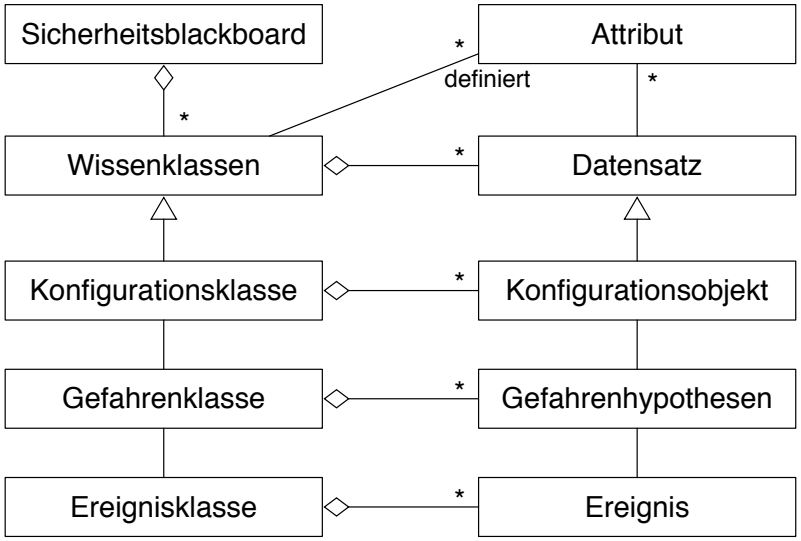


Abb. 6.26: *Analyse-Objektmodell des CUSTODIAN-Sicherheitsblackboards*

6.2.5 Die Wahl des Architekturmusters

Bei der Auswahl der Architektur stand das Zusammenarbeiten mehrerer Experten im Vordergrund. Die in CUSTODIAN arbeitenden Experten müssen gegenseitig auf ihre Ergebnissen aufbauen können. So kann ein Gefahrenexperte Gefahren in Konfigurationen erkennen, die von Konfigurationsexperten bereitgestellt wurden und dann von einem Visualisierungsexperten dargestellt zu werden. Eine flexible Inkorporation der Experten muss daher von CUSTODIAN unterstützt werden. Aufgrund dieser Anforderungen der vielseitigen Erweiterbarkeit und der flexible Inkorporation von Experten bietet sich das Blackboard-Architekturmuster für das CUSTODIAN-Sicherheitsframework an.

Das Blackboard-Architekturmuster kam erstmals bei der Hearsay II, einen an der Carnegie Mellon University (CMU) entwickelten Spracherkennungssystem, zum Einsatz [BELW80]. Buschmann beschreibt das Architekturmuster als eine Menge von Wissensquellen, die auf einem Blackboard arbeiten und von einer Kontrolleinheit gesteuert werden [Bus98].

> *„ Das Blackboard-Muster hilft bei Problemen, für die keine deterministische Lösungsstrategien bekannt sind. In diesem Muster stellen mehrere spezialisierte Subsysteme ihr Wissen zur Verfügung, um eine möglicherweise unvollständige oder nur approximierende Lösung zu erstellen. “*
>
> Frank Buschmann [Bus98]

Die Kontrolleinheit (siehe Abbildung 6.27) wird in einer Schleife gestartet, die permanent überprüft, ob eine Wissensquelle gestartet werden soll. Jede Wissensquelle stellt sein Gefahrenwissen auf dem Sicherheitsblackboard bereit. Auf dieses Gefahrenwissen können alle anderen Wissensquelle anschließend zugreifen. Dadurch sind die Wissensquellen voneinander entkoppelt, obwohl sie gegenseitig auf ihr Gefahrenwissen zugreifen können.

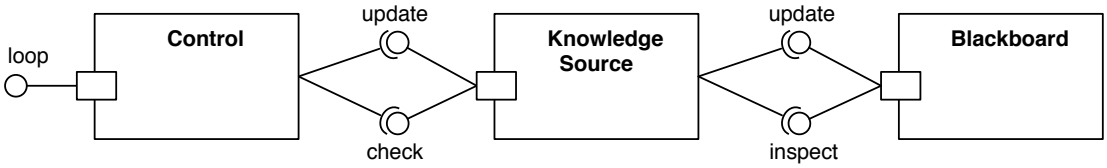


Abb. 6.27: *Der prinzipielle Aufbau einer Blackboard-Architektur in Anlehnung an Buschmann et. al. [Bus98]*

Die Blackboard-Architektur hat bereits in verschiedenen Anwendungsbereichen dabei geholfen, die Zusammenarbeit mehrerer Experten zu koordinieren, die für sich alleine das gegebene Problem nicht lösen können [Nii86, RB07]. Erstmals wurde das Blackboard-Architekturmuster, wie bereits erwähnt, bei der Hearsay II [EHRLR80], einem in den 1980er Jahren entwickelten Spracherkernnungssystem, eingesetzt. In den folgenden Jahren wurde die Blackboard-Architektur auch zur interaktiven Visualisierung von Daten [LHDL98], zur Interpretation von Nachrichten [CMSW01] und zur Navigation in Gebäuden [OUP04] erfolgreich eingesetzt. Dong beschreibt, wie eine ereignisbasierte Blackboard-Architektur bei der Steuerung von Mehragentensystemen eingesetz werden kann [DCJ05]. Diese unterschiedlichen Einsatzgebiete deuten darauf hin, dass sich die Verwendung der Blackboard-Architektur nicht nur in der ursprünglichen Domäne, der Spracherkennung, anbietet.

Blackboard-Architektur definiert, dass Experten als Wissensquellen auf dem Sicherheitsblackboard arbeiten (siehe Abbildung 6.28). Eine Ausnahme

stellt der Historisierungsexperte dar, dessen Aufgaben das Depot wahrnimmt. Ein Depot wird in der Blackboard-Architektur für die Organisation der Datenstruktur eingesetzt [BD04]. Im CUSTODIAN-Framework übernimmt das Depot die Historisierung.

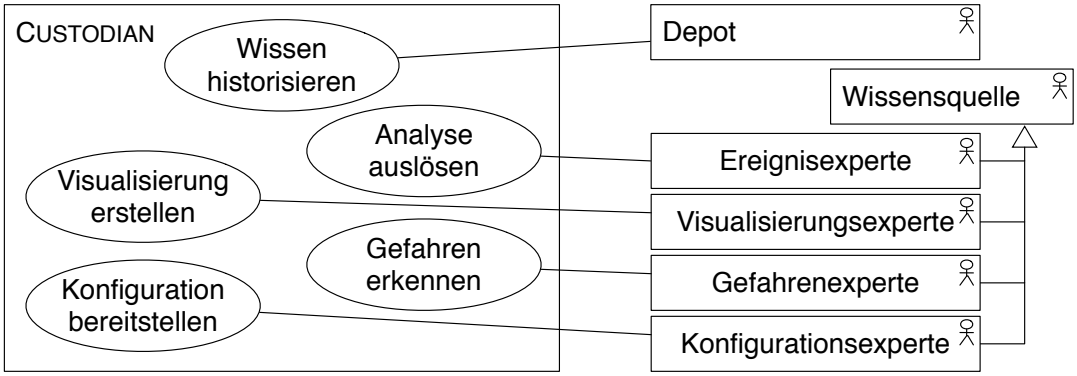


Abb. 6.28: *Verfeinerung der Anwendungsfälle für die Blackboard-Architektur*

Auf der Basis der verfeinerten Anwendungsfälle ergibt sich für CUSTODIAN das in Abbildung 6.29 modellierte Analyse-Objektmodell.

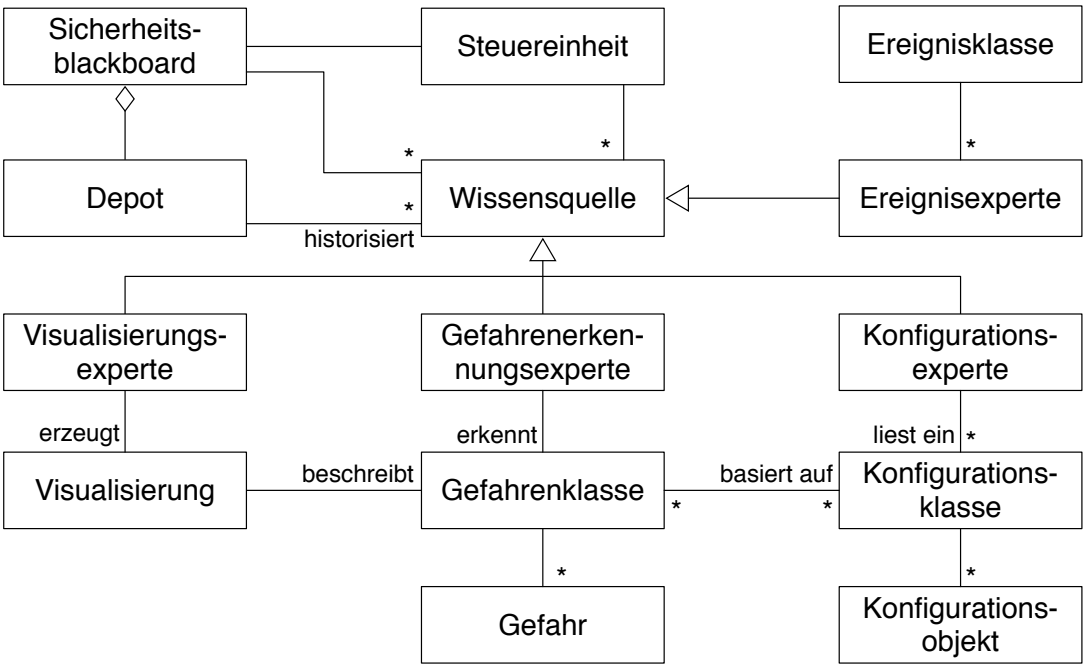


Abb. 6.29: *Analyse-Objektmodell des CUSTODIAN-Sicherheitsframeworks*

Die Grundkomponenten der Blackboard-Architektur sind mit den Wissensquellen, dem Sicherheitsblackboard und der Steuereinheit übernommen.

Wissensquellen heißen bei CUSTODIAN Sicherheitsexperten, da sie bei der Gefahrenerkennung benötigtes Gefahrenwissen bereitstellen. Das Sicherheitsblackboard besteht aus einem Depot, welches für das historisierte Speichern des Gefahrenwissens der Wissensquellen verantwortlich ist. Vier verschiedene Wissensquellen arbeiten auf dem Sicherheitsblackboard. Ereignisexperten können verschiedene Ereignisse erkennen und diese auf dem Blackboard veröffentlichen. Konfigurationsexperten können Konfigurationsklassen von Komponenten auslesen und deren Konfigurationsobjekte auf dem Sicherheitsblackboard bekanntgeben. Gefahrenexperten erkennen die Gefahren der für das verteilte System bekannten Gefahrenklassen. Die Gefahren einer Gefahrenklasse können von einen Visualisierungsexperten mit einer geeigneten Visualisierung dargestellt werden.

6.3 Systementwurf

Die Externalisierung von Gefahrenwissen ist in Abschnitt 3.6 beschrieben. Daraufhin ist das Sicherheitsframework in der Lage, dieses Gefahrenwissen ohne weiteres menschliches Zutun auf dem aktuellen Stand zu halten. So können, wie es Shaw und Garlan beschreiben [SG96], unabhängig operierende Wissensquellen dazu beitragen, das komplexe Problem der Gefahrenerkennung in verteilten Systemen zu lösen. Ziel ist es, dem Sicherheitsframework korrektes und hinreichend umfängliches Gefahrenwissen bereitzustellen. Unter dieser Bedingung erkennt CUSTODIAN Gefahren in der Konfiguration des getesteten verteilten Systems. Das schließt die Fehleranfälligkeit menschlicher Arbeit aus und wirkt der Kostenintensität vieler manueller Analysen entgegen.

Im Sicherheitsframework CUSTODIAN wird der Blackboard-Architekturstil eingesetzt. Die Steuereinheit und das Blackboard, genannt Sicherheitsblackboard, werden durch das Sicherheitsframework gestellt. Neue Wissensquellen, welche im Kontext des Sicherheitsframeworks CUSTODIAN Sicherheitsexperten genannt werden, können mit Hilfe einer Schnittstelle (siehe Abbildung 6.30) eingebunden werden. Hierfür erfragt die Steuereinheit über den Dienst `deklarationen` die Deklarationen des vom Sicherheitsexperten bereitgestellten Wissensklassen ab. Mit Hilfe dieser Wissensklassendeklarationen kann die Steuereinheit den Sicherheitsexperten am Sicherheitsblackboard über den Dienst `experteRegistgrieren` registrieren. Das Sicherheitsblackboard erstellt daraufhin die für den Sicherheitsexperten benötigten Datenstrukturen.

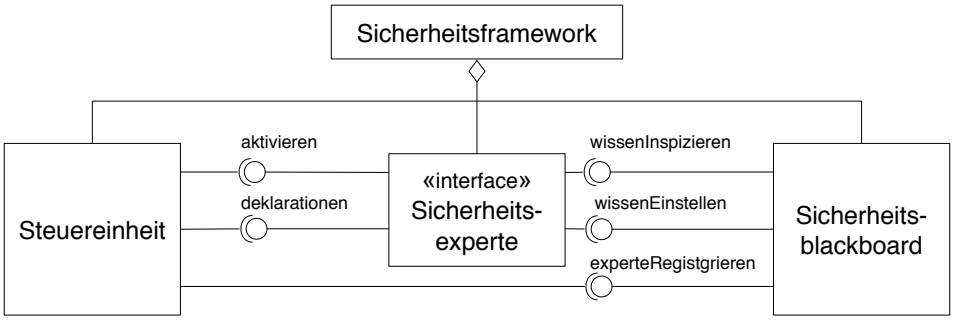


Abb. 6.30: *Die Subsystemdekomposition der Sicherheitsframeworks auf Basis des Blackboard-Architekturmusters*

Registrierte Sicherheitsexperten können von der Steuereinheit über den Dienst `aktivieren` angestoßen werden. Daraufhin erzeugt der Sicher-

heitsexperte den aktuellen Wissensstand. Hierbei kann er über den Dienst `wissenInspizieren` sämtliches Wissen des Sicherheitsblackboards einsehen. Der Sicherheitsexperten stellt den aktuellen Wissensstand auf dem Sicherheitsblackboard mit Hilfe des Dienstes `wissenEinstellen` bereit. Die Blackboard-Architektur ermöglicht das Zusammenspiel der Sicherheitsexperten. Das Gefahrenwissen, das den Sicherheitsexperten zugrundeliegt, muss nur einmal von realen Personen (Spezialisten) externalisiert werden.

Das Sicherheitsframework testet ein verteiltes System, das System Under Test, auf Gefahren. Dazu lesen Konfigurationsexperten die Konfiguration des verteilten Systems aus. Diese legen die Konfiguration in einer für die Analyse geeigneten Form auf dem Sicherheitsblackboard offen. Gefahrenexperten können daraufhin Gefahrenhypothesen generieren. Andere Gefahrenexperten können diese Gefahrenhypothesen bestätigen oder invalidieren. Nicht invalidierte Gefahrenhypothesen gelten nach der dem Sicherheitsblackboard aktuell vorliegenden Gefahrenwissen als Gefahr. Kommunmikationsexperten visualisieren die Gefahren und notifizieren verantwortliche Personen.

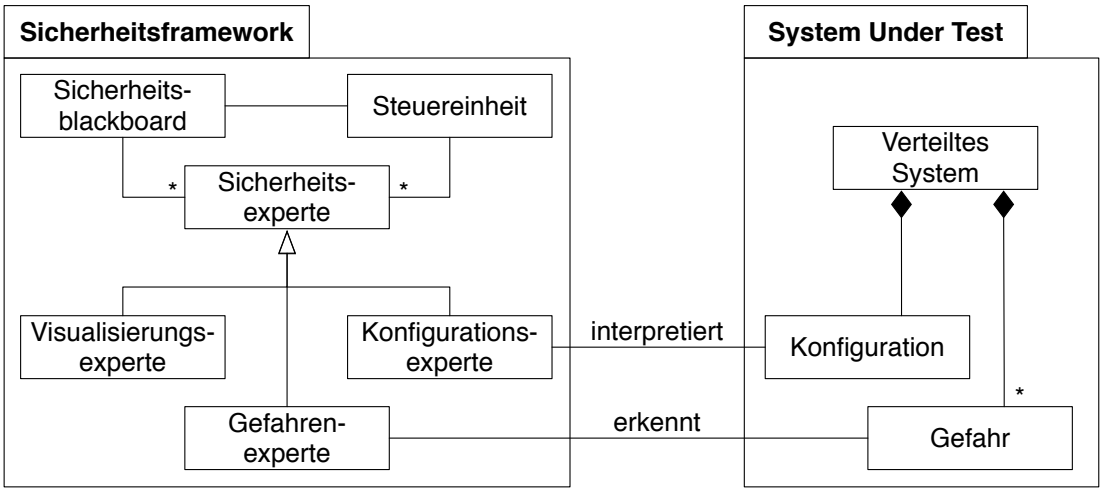


Abb. 6.31: *Prinzipieller Aufbau des Sicherheitsframework im Zusammenspiel mit einem vereinfacht dargestellten verteilten System*

Die Blackboard-Architektur baut, wie in Abbildung 6.31 dargestellt, auf dem Entwurfsmusters des Depots (Repository-Pattern) auf. Das Sicherheitsframework stellt ein Depot und eine Steuereinheit bereit. Für das Einbinden von Sicherheitsexperten stellt das Sicherheitsframework die Schnittstelle Sicherheitsexperte bereit. Abschnitt 6.2.1 geht auf die Funktionsweise und

die Schnittstellen des Depots ein. Die Sicherheitsexperten werden im Abschnitt 6.2.2 detailliert beschrieben. Und Abschnitt 6.2.3 gibt eine Übersicht über die Steuereinheit.

6.4 Referenzimplementierung

Der gewählte Forschungsansatz zur Evaluation des Sicherheitsframeworks CUSTODIAN erfordert eine Referenzimplementierung. Für diese Evaluation war eine Implementierung der CUSTODIAN-Architektur vonnöten. Der folgende Abschnitt beschreibt die Referenzimplementierung namens KUSTOS. Neben der Referenzimplementierung KUSTOS selbst sind für den Betrieb noch weitere Komponenten notwendig (siehe Abbildung 6.32). Die Sicherheitsexperten stellen das Gefahrenwissen bereit, um Gefahren eines verteilten System zu erkennen. Da für unterschiedliche verteilte Systeme verschiedene Sicherheitsexperten benötigt werden, sind die Sicherheitsexperten nicht fester Bestandteil von KUSTOS. Ein Depot stellt die persistente Speicherung der Informationen des Sicherheitsblackboards sicher.

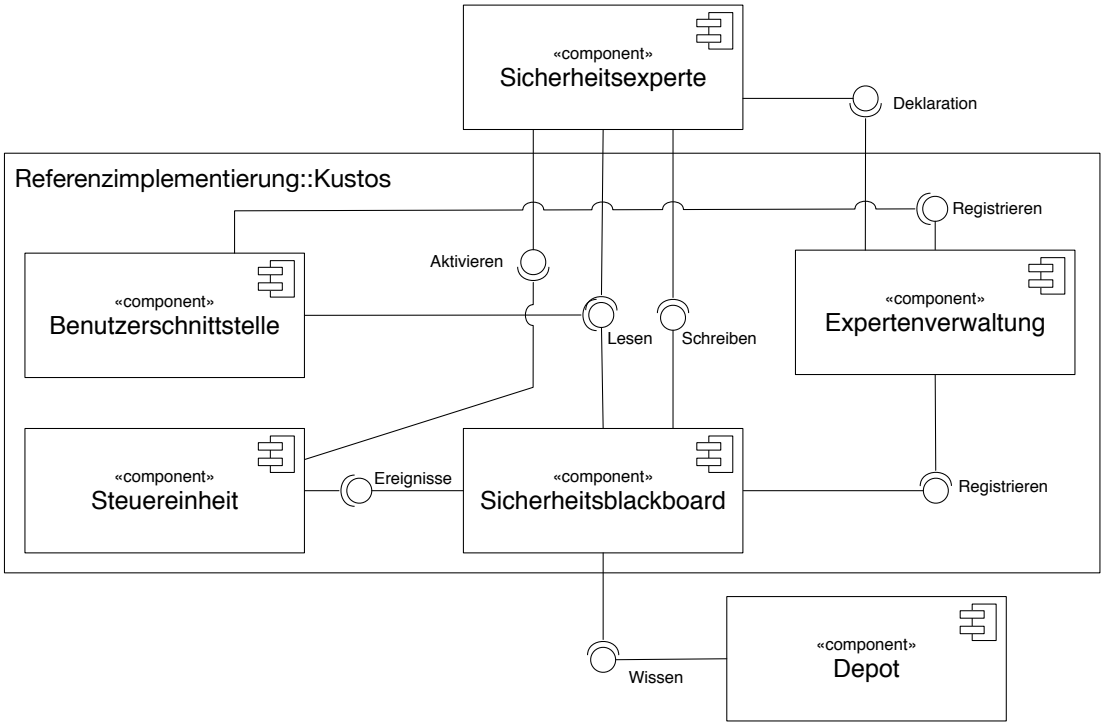


Abb. 6.32: *Die Subsystemdekomposition der Referenzimplementierung KUSTOS*

Die Referenzimplementierung KUSTOS ist in der Programmiersprache Java implementiert. KUSTOS selbst besteht, wie in Abbildung 6.32 modelliert, aus dem zentralen Sicherheitsblackboard, einer Steuereinheit, einer Expertenverwaltung und einer Benutzerschnittstelle. Auf das Sicherheitsblack-

board schreiben Sicherheitsexperten ihre Erkenntnisse. Diese Erkenntnisse werden von der Benutzerschnittstelle und von weiteren Sicherheitsexperten gelesen. Die Steuereinheit observiert die Ereignisse des Sicherheitsblackboards und aktiviert Sicherheitsexperten, damit diese arbeiten können. Über die Expertenverwaltung können Sicherheitsexperten auf dem Sicherheitsblackboard aktiviert werden. Das kann der Benutzer von der Benutzerschnittstelle aus initiieren. Die Benutzerschnittstelle setzt für die Visualisierung das Framework JavaFX ein und erlaubt es dem Benutzer, mit KUSTOS zu interagieren.

Während der Implementierung von KUSTOS wurden mehrere allgemeine Sicherheitsexperten erstellt. Diese Sicherheitsexperten sind aus zweierlei Gründen fest in KUSTOS integriert. Einerseits stehen diese allgemeinen Sicherheitsexperten automatisch in jeder KUSTOS-Installierung zur Verfügung. Andererseits können diese Sicherheitsexperten effizienter auf das Sicherheitsblackboard zugreifen. Denn die integrierten Sicherheitsexperten können nicht nur über die vom Sicherheitsblackboard zu Verfügung gestellten Schnittstellen, sondern auch direkt auf die Informationen des Sicherheitsblackboards zugreifen. Im Abschluss stellt dieses Kapitel die allgemeinen, mit KUSTOS entwickelten, Sicherheitsexperten vor.

Die Tabelle 6.1 zeigt eine Übersicht der CUSTODIAN-Anwendungsfälle. Für jeden Anwendungsfall definiert sie dessen Akteure (siehe Abschnitt 6.2) und referenziert gegebenenfalls auf deren Umsetzung in KUSTOS.

Anwendungsfall	Akteur
Experte bereitstellen	Der Akteur dieses Anwendungsfalls erstellt ein Spezialist eine Wissensquelle. Diese macht sein Gefahrenwissen auf dem Sicherheitsblackboard replizierbar.
Konfiguration bereitstellen	Konfigurationsexperten stellen als Wissensquelle die Konfiguration bereit.
KUT explorieren	Sowohl die Benutzer des Frameworks als auch die Wissensquellen können auf die KUT zugreifen. Den Benutzern steht hierfür der Wissensbrowser bereit (siehe Abschnitt 6.4.4). Wissensquellen können über die Fassade des Depots auf das Wissen zugreifen (siehe 6.2.1.1).
Korrelationen ermitteln	Abschnitt 6.4.2 die Wissensquelle Korrelationsexperte.
Wissen historisieren	Das Wissen auf dem Sicherheitsblackboard wird von dem Depot historisiert (siehe Abschnitt 6.2.1.3)
Gefahrenwissen verbinden	Spezialisten können mit Hilfe des Wissensbinders Gefahrenwissen verbinden (seihe Abschnitt 6.4.3).
Gefahrenklasse definieren	Spezialisten können mit Hilfe des Gefahrenanalytikers Gefahrenklasse definieren (siehe Abschnitt 6.4.5).
Gefahren erkennen	Gefahrenerkennungsexperten stellen als Wissensquelle die Konfiguration bereit.
Sicherheitsüberprüfung auslösen	Ereingisexperten lösen als Wissensquelle Sicherheitsüberprüfung aus.
Visualisierung erstellen	Visualisierungsexperten erstellen als Wissensquelle Visualisierungen. Ein von KUSTOS bereitgestellter Visualisierungsexperte ist der Motion-Chart (siehe Abschnitt 6.4.1).
Gefahren explorieren	Die Benutzer von CUSTODIAN können die erkannten Gefahren explorieren.

Tab. 6.1: *Anwendungsfälle von CUSTODIAN und deren Umsetzung in KUSTOS*

6.4.1 Motion-Charts

KUSTOS appliziert Motion-Charts (siehe Kapitel 3.7.1), um eine große Anzahl von Gefahren in einer Übersicht darzustellen. Abbildung 6.33 zeigt ein Bildschirmfoto eines von KUSTOS visualisierten Motion-Chart. Die acht hellblau unterlegten Zahlen sind nachträglich hinzugefügte Annotationen und dienen der Erklärung.

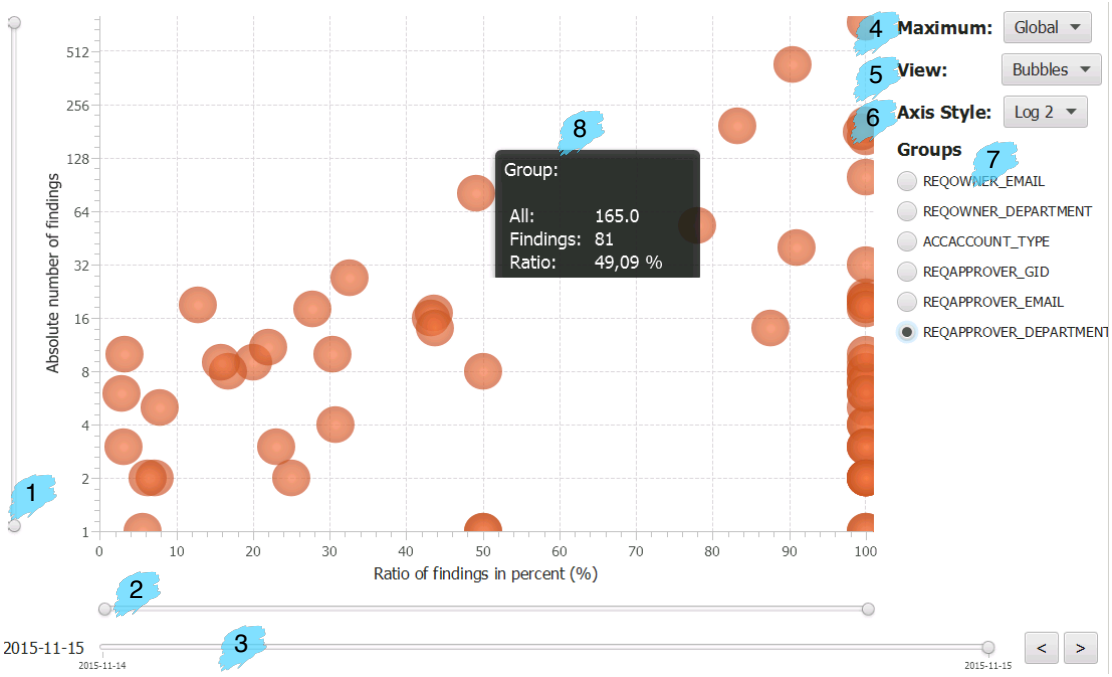


Abb. 6.33: *Exemplarische Darstellung eines Motion-Charts*

Große Anzahlen von Gefahren können mit Hilfe des Motion-Charts aggregiert und als Gruppe dargestellt werden. Jede Gruppe wird als ein Kreis dargestellt, dessen Position durch die Fehlermenge in der Y-Achse und durch die Fehlerhäufigkeit in der X-Achse bestimmt ist:

$$\frac{\text{Anzahl der Gefahren in der Gruppe}}{\text{Anzahl aller Testobjekte in der Gruppe}} \qquad \text{(X-Achse)}$$

$$\text{Anzahl der Gefahren in der Gruppe} \qquad \text{(Y-Achse)}$$

Die Schieber (Annotation [1] und [2]) geben den Benutzer die Möglichkeit, Teile des Charts in vergrößerter Darstellung zu betrachten. Der Schieber

[3] erlaubt es, zwischen den verschiedenen Momentaufnahmen (verschiedene Tage) zu navigieren. Das Navigieren auf einen anderen Datenpunkt führt zum Laden dieses Datenpunkts. KUSTOS animiert die Veränderungen zwischen dem neuen und den alten Datenpunkt. Punkt [4] ermöglicht die Auswahl zwischen einem lokalen oder einem globalen Maximum in der Y-Achse. Unter Annotation [5] kann in eine Tabellenansicht gewechselt werden. Unter Annotation [6] kann gewählt werden, ob die Y-Achse logarithmisch oder linear dargestellt wird. Unter Annotation [7] kann der Diskriminator der angezeigten Gruppen ausgewählt werden. Die Diskriminatoren können für jede Gefahrenklasse individuell definiert werden. Annotation [8] stellt einen Popup dar. Dieser erscheint, wenn der Benutzer mit der Maus über eine Gruppe navigiert. In diesem werden die grundlegenden Informationen über diese Gruppe angezeigt, nämlich Name (im Beispiel geschwärzt), Anzahl der Gruppenmitglieder, Anzahl der Gefahren sowie den prozentualen Anteil der Gefahren. Mit einen Mausklick auf einen Kreis kann zu den vom Kreis repräsentierten Gefahren navigiert werden.

6.4.2 Korrelationsexperte

Unterschiedliche Sicherheitsexperten stellen auf dem Sicherheitsblackboard ihr Wissen in ihren Wissensklassen bereit. Zwischen den verschiedenen Wissensklassen existieren Korrelationen. Die Korrelation wird durch inhaltliche Überschneidungen der Ausprägungen zweier Attribute verschiedener Wissensklassen bestimmt. Das Beispiel der Abbildung 6.34 verdeutlicht Korrelation anhand der Wissensklassen LDAP und BenutzerDB. Beide Wissensklassen haben jeweils zwei Attribute. Das einzige Attributpaar, bei dem gleiche Werte vorkommen, ist $(LDAP.Gruppe, BDB.ID)$. Alle vier verschiedenen Werte des Attributs $LDAP.Gruppe$ kommen im Attribut $BDB.ID$ vor, somit ist die Korrelation 100 Prozent. Von den sechs verschiedenen Werten des Attributs $BDB.ID$ kommen im Attribut $LDAP.Gruppe$ vier vor, somit ist die Gegenkorrelation 66,7 Prozent. Bei allen anderen Attributkombinationen existiert keine Übereinstimmung und somit besteht eine Korrelation von 0 Prozent. Der Korrelationsexperte stellt Korrelationen unter 10 Prozent nicht dar.

LDAP

Gruppe	Mitglied
Admins	P1
Admins	P2
Benutzer	P1
Benutzer	P3
Benutzer	P6

BenutzerDB (BDB)

ID	Name
P1	Stephan Wilms
P2	Djahan Salar
P3	Hannes Kiebler
P4	Matthias Wirsching
P5	Christian Hauck
P6	Christopher Lehmann

Korrelation

FeldA	FeldB	Korrelation
LDAP.Gruppe	BDB.ID	0,000
LDAP.Mitglied	BDB.ID	1,000
LDAP.Gruppe	BDB.Name	0,000
LDAP.Mitglied	BDB.Name	0,000
BDB.ID	LDAP.Gruppe	0,000
BDB.Name	LDAP.Gruppe	0,000
BDB.ID	LDAP.Mitglied	0,667
BDB.Name	LDAP.Mitglied	0,000

Abb. 6.34: *Korrelation zwischen den Attributen zweier Wissensklassen*

Mit steigender Anzahl der Wissensklassen auf dem Sicherheitsblackboard wird es für den Benutzer schwieriger, die Korrelationen selbständig zu erkennen. Für viele der Auswertungen ist es jedoch wichtig, einen Einblick über die Zusammenhänge zwischen den Wissensklassen zu haben. Deshalb bietet KUSTOS einen Korrelationsexperten, der die Korrelationen zwischen den Wissensklassen berechnet und auf dem Blackboard ablegt. Mit diesen Informationen wird der Benutzer dabei unterstützt, Zusammenhänge im verteilten System zu erkennen. Die Abbildung 6.35 zeigt, wie KUSTOS diese Unterstützung anbietet. Nachdem der Benutzer das Attribut `VermittlerID` einer Wissensquelle `Lizenzverwaltung` ausgewählt hat, wird ihm anzeigt, wie stark die Attribute anderer Wissensquellen dazu korrelieren.

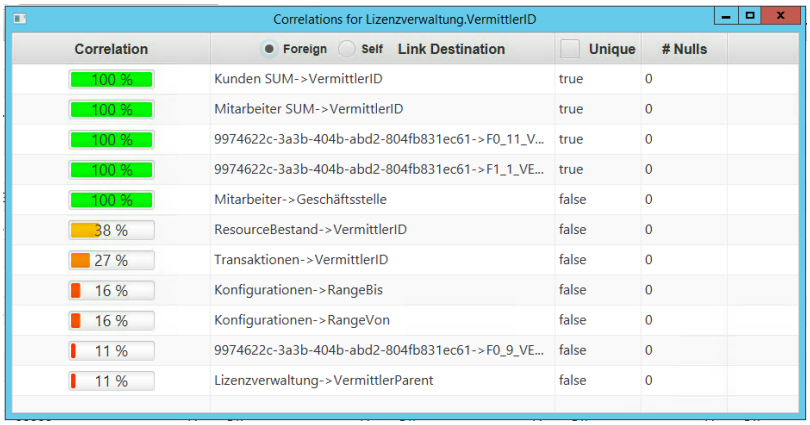


Abb. 6.35: *Der Korrelationsexperte zeigt Korrelationen zwischen Attributen*

6.4.3 Wissensbinder

Die mit KUSTOS durchgeführten Gefahrenanalysen setzen im Regelfall das Gefahrenwissen unterschiedlicher Sicherheitsexperten voraus. Deshalb besteht die Notwendigkeit, das Wissen von mehrerer Wissensklassen zu vereinigen. Das ist möglich, indem man einen neuen Sicherheitsexperten implementiert, der auf das Wissen der bestehenden Wissensklassen zugreift, dieses vereint und auf dem Sicherheitsblackboard zur Verfügung stellt. Jedoch ist dieser immer wiederkehrende Prozess mit viel Aufwand verbunden und kann nur von Programmierern durchgeführt werden. Deshalb stellt KUSTOS den Wissensbinder bereit. Der Wissensbinder ermöglicht das Vereinigen von Wissen, welches in neuen Wissensklassen mündet, über eine graphische Benutzeroberfläche. Der Wissensbinder ist in Abbildung 6.36 anhand eines Beispiels dargestellt, bei dem er zwei Wissensklassen zu einer neuen vereint.

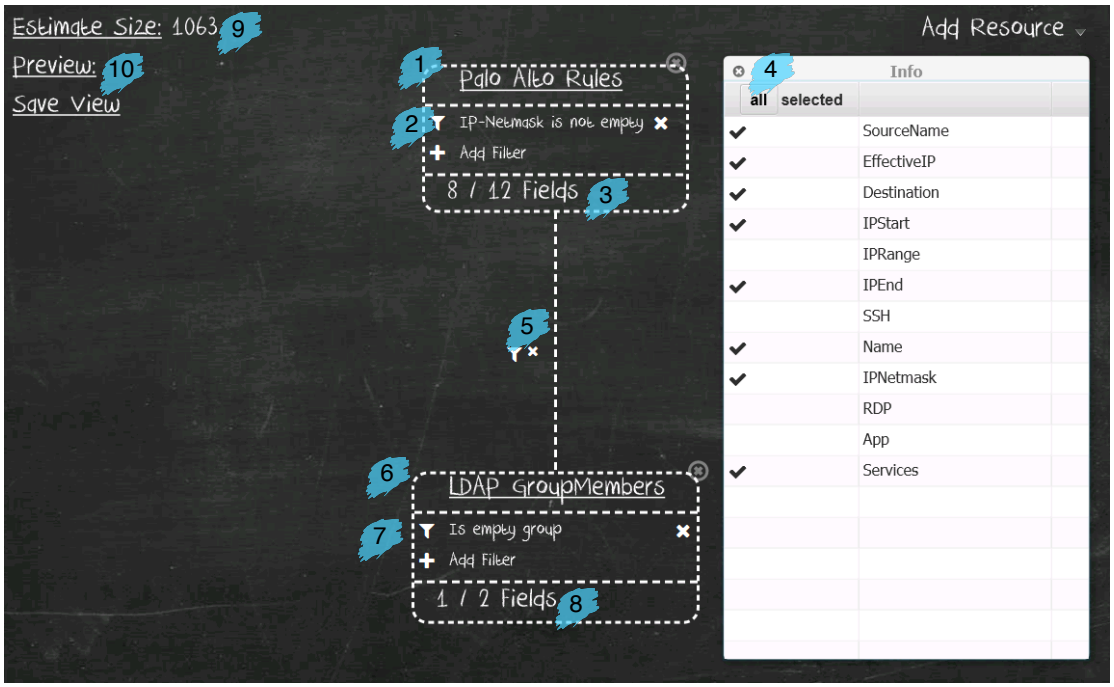


Abb. 6.36: *Erstellen einer neuen Wissensklasse mit dem Wissensbinder*

Die Annotationen [1] und [6] zeigen die beiden Wissensklassen „Palo Alto Rules“ und „LDAP-GroupMembers“, die der Wissensbinder vereint. Es existieren zwei Filter, welche die Anzahl der Datensätze reduzieren (Annotationen [2] und [7]). Die Annotationen [3] und [8] zeigen, wieviele der Attribute

der ursprünglichen Gefahrenklassen in die neue Gefahrenklasse übernommen werden. Annotation [4] zeigt das eingeblendete Fenster, in dem die Attribute der Wissensklasse „Palo Alto Rules“ selektiert werden können. Annotation [5] zeigt den Link zwischen den beiden verwendeten Wissensklassen, der das korrekte Zuordnen der zueinander gehörenden Datensätze sicherstellt. Über [9] kann der Benutzer von KUSTOS abschätzen lassen, wie viele Datensätze in der neuen Wissensklasse entstehen und über [10] kann der Benutzer die neu entstehenden Datensätze einsehen.

Der Wissensbinder bildet das kartesische Produkt mehrerer Mengen (den Datensätzen einer Wissensklasse). Das Ergebnis kann hierbei, äquivalent zu einer SQL-Anweisung, manipuliert werden. Die Datensätze der Wissensklasse können vorgefiltert werden, das entspricht der `WHERE`-Anweisung in SQL (siehe Annotationen [1] und [6]). Für das kartesische Produkt kann ein Gleichverbund definiert werden, das entspricht der `ON`-Anweisung in SQL (siehe Annotation [5]). Und die Attribute der verbundenen Wissensklassen können selektiert werden, das entspricht der `SELECT`-Anweisung in SQL (siehe Annotationen [3] und [8]).

Der Wissensbinder erlaubt es, auch Nicht-Programmierern neue Wissensklassen zu erstellen, die auf dem Wissen der existierenden Wissensklassen basieren. Somit werden sie in ihrer Analyse unabhängiger von Programmierern und können schneller und selbständiger Gefahren erkennen.

6.4.4 Wissensbrowser

Sicherheitsexperten sollen in die Lage versetzt werden, das vorhanden Wissen zu explorieren. Hierfür ist ein generisches Visualisierungskonzept vonnöten, denn das anzuzeigende Wissen ist, aufgrund der Erweiterbarkeit von KUSTOS, nicht im Vorhinein bekannt. Der Wissensbrowser ermöglicht es den Nutzern, das auf dem Sicherheitsblackboard bereitstehende Wissen über ein verteiltes System zu explorieren. Mit dem Wissensbrowser kann man, vergleichbar mit einem Webbrowser, durch Informationen navigieren. Abbildung 8.49b zeigt den Wissensbrowser, der die Wissensklasse namens „AdWorks.Mitarbeiter“ geöffnet hat. Einige Werte sind aufgrund der Vertraulichkeit zensiert.

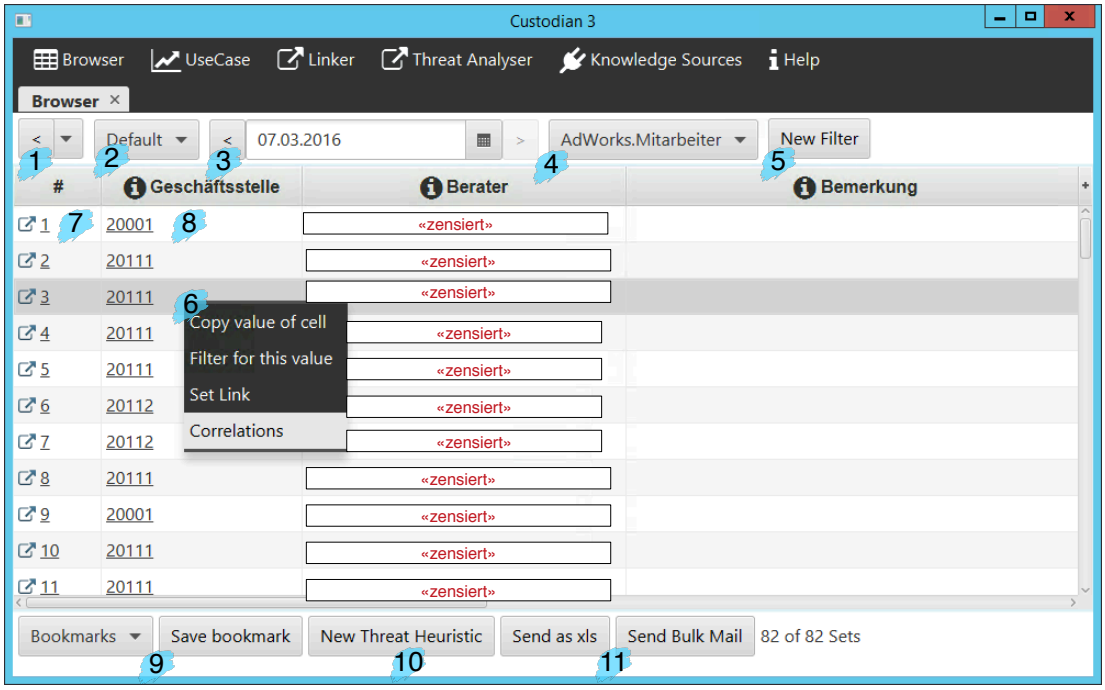


Abb. 6.37: *Der Wissensbrowser zum Explorieren eines verteilten Systems*

[1] Zurückknopf, mit dem man zu einem Stand mit bisherigen Browserverlauf zurückspringen kann.

[2] Mit dieser Schaltfläche kann man die Selektionsart bestimmten, mit welcher Datensätze des unter [3] ausgewählten Datums angezeigt werden: „Default“ (alle existierenden Datensätze), „Created“ (die erstellten Datensätze) und „Deleted“ (die gelöschten Datensätze).

[3] Mit dieser Schaltfläche bestimmt man, von welchem Datum die Datensätze angezeigt werden.

[4] Auswahl der anzuzeigenden Wissensquelle.

[5] Über diese Schaltfläche können Filter hinzugefügt werden, welche die angezeigten Datensätze einschränken.

[6] Durch den Rechtsklick in die Tabelle erscheint dieses Kontextmenü, über das man den Wert der Zelle in den Zwischenspeicher kopieren, einen neuen Filter mit dem Wert der Zelle erstellen, einen neuen Link zu einer anderen Wissensklasse definieren und die Korrelationen

zu Attributen anderer Wissensklassen einsehen kann (siehe Abschnitt 6.4.2).

[7] Über das Auswählen der Spalte „#“ wird die Anzeige des Datensatzes mit einem Visualisierer (siehe Abbildung 6.38) bewirkt. Wenn für diese Wissensquelle kein spezieller Visualisierer existiert, wird der Datensatz im Standard-Visualisierer angezeigt.

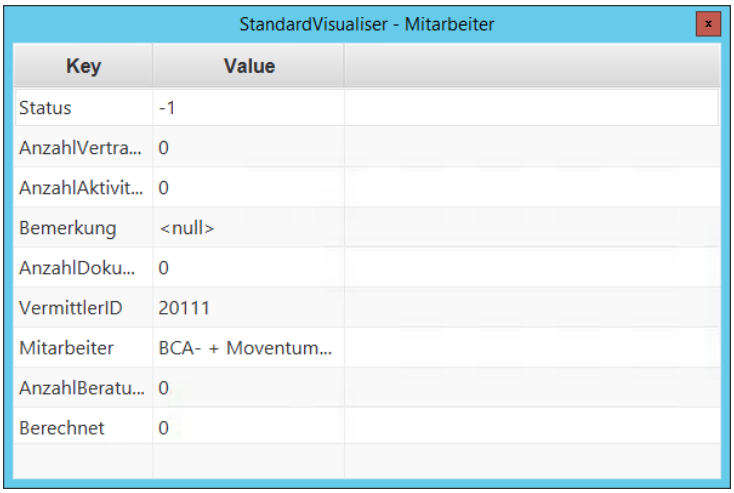


Abb. 6.38: *Anzeige eines Datensatzes mit dem Standard-Visualisierer*

[8] Der Wert des Attributs `Geschäftsstelle`. Durch das Unterstreichen des Wertes wird dem Benutzer angezeigt, dass es sich um einen Link handelt, der zu einer anderen Wissensquelle führt. Folgt er diesem Link, wird die verknüpfte Wissensquelle mit einem entsprechend vorgewählten Filter auf das verknüpfte Attribut (in diesem Fall mit dem Wert `20001`) angezeigt.

[9] Die Einstellungen, die im Wissensbrowser getätigt sind (Wissensklasse, Selektionsart und Filter) können als Bookmark gespeichert und später wieder aufgerufen werden.

[10] Mit dieser Schaltfläche erstellt der Benutzer eine neue Gefahrenklasse, die auf der aktuellen Wissensklasse basiert.

[11] Über die Sendefunktionen können die gezeigten Inhalte als Tabellenkalkulation per E-Mail versendet werden.

6.4.5 Gefahrenanalytiker

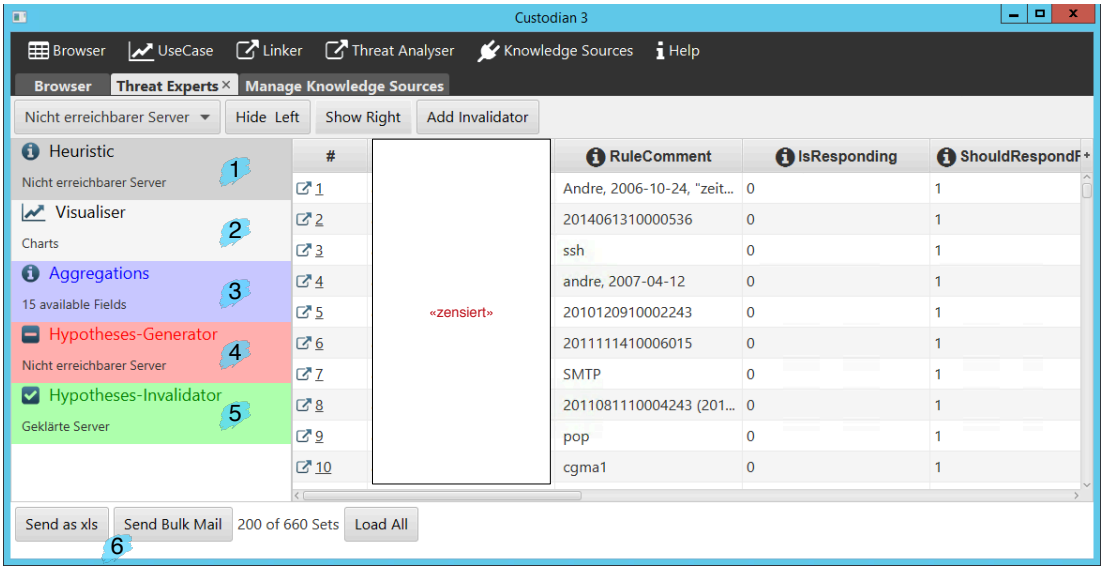


Abb. 6.39: *Der Gefahrenanalytiker zum Generieren und Invalidieren von Gefahrenhypothesen*

Nach dem Erstellen einer Gefahrenklasse (siehe Abschnitt 6.4.4) kann der Benutzer diese anschließend mit dem Gefahrenanalytiker konfigurieren. Der Gefahrenanalytiker implementiert den in der Analyse vorgestellten Mechanismus zur Generierung und Invalidierung von Gefahrenhypothesen (Abschnitt 6.2.2.3). Hierfür muss man zuerst über Schaltfläche [4] ein Hypothesengenerator definieren. Ein Hypothesengenerator definiert mit einem Ausdruck der Prädikatenlogik der ersten Stufe (engl. First-order logic), welche Datensätze der zugrundeliegenden Wissensklasse eine Gefahr im Sinne der Gefahrenklasse darstellen. Auf der Basis der generierten Gefahrenhypothesen können Invalidatoren definiert werden. Ein Invalidator (Schaltfläche [5]) definiert ebenfalls mit einem Ausdruck der Prädikatenlogik der ersten Stufe, welche Gefahrenhypothesen falsch sind und nicht mehr angezeigt werden sollen. Es können gleichzeitig mehrere Invalidatoren existieren, die verschiedene Gefahrenhypothesen invalidieren. Die Invalidatoren und Hypothesengeneratoren bestimmen welche Datensätze dem Benutzer als Gefahr angezeigt werden (erreichbar über Schaltfläche [1]) und definieren somit eine maschinell interpretierbare Definition der Gefahrenklasse. Über die Schaltfläche Aggregationen ([3]) kann bestimmt werden, nach welchen Attributen der Wissensklasse die Gefahr gruppiert werden soll. Diese Ein-

stellung ist für die Motion-Charts (siehe Abschnitt 6.4.1) notwendig, denn in diesem repräsentiert jeder Kreis eine Gruppe. Die Motion-Charts für die Gefahrenklasse können über die Schaltfläche [2] eingesehen werden. Wie auch vom Wissensbrowser kann der Benutzer mit den Schaltflächen [6] die Ergebnisse per Mail versenden.

Beispiel Hypothesengenerator und Invalidator:
Die Wissensklasse „Serverzugriffe“ hat sechs Datensätze (siehe Abbildung 6.40). Ein Datensatz ist ein Tupel (bid, bs, sid, ss) mit der ID des Benutzers bid, dem Status des Benutzers bs, der ID des Servers sid und dem Status des Servers ss.

Serverzugriffe

BenutzerID	BenutzerStatus	ServerID	ServerStatus
b1	extern	s1	geheim
b1	extern	s2	offen
b2	intern	s2	offen
b2	intern	s3	offen
b3	intern	s4	geheim
b4	extern	s4	geheim

Abb. 6.40: *Datensätze der Wissensklasse Serverzugriffe*

Die Gefahrenklasse „ExternAufGeheim“ basiert auf der Wissensklasse „Serverzugriffe“. Der Hypothesengenerator der Gefahrenklasse „ExternAufGeheim“ bestimmt, dass ein Serverzugriff eine Gefahr ist, wenn gilt:

$$\text{gefahr}(bid, bs, sid, ss) \equiv (bs = extern \wedge ss = geheim).$$

Abbildung 6.41 zeigt die beiden Gefahrenhypothesen, die von dem definierten Hypothesengenerator generiert werden.

ExternAufGeheim

BenutzerID	BenutzerStatus	ServerID	ServerStatus
b1	extern	s1	geheim
b4	extern	s4	geheim

Abb. 6.41: *Gefahrenhypothesen auf Basis des Hypothesengenerators*

Der Benutzer mit der ID „b4“ wird von einem Sicherheitsspezialisten als ungefährlich identifiziert. „b4“ darf auf geheime Server zugreifen. Daher muss ein Invalidator erstellt werden:

$$\text{invalidator}(bid, bs, sid, ss) \equiv (bid = b4).$$

Abbildung 6.42 zeigt die Gefahrenhypothesen, die nun auf Basis des Hypothesengenerators und des Invalidators generiert werden.

ExternAufGeheim

BenutzerID	BenutzerStatus	ServerID	ServerStatus
b1	extern	s1	geschlossen

Abb. 6.42: *Gefahrenhypothesen auf Basis des Hypothesengenerators und des Invalidators*

7 Das Vorgehensmodell HARVEST

Dieses Kapitel beschreibt das Vorgehensmodell HARVEST zur Gefahrenerkennung in verteilten Systemen[1]. Die Metapher soll veranschaulichen, dass in verteilten Systemen prinzipiell Gefahren existieren, die nur noch „geerntet“ werden müssen. HARVEST kam bei den in Kapitel 8 beschriebenen Fallstudien zum Einsatz und wurde für die Anwendung von CUSTODIAN in verteilten Systemen entwickelt. HARVEST basiert auf dem agilen Vorgehensmodell Rugby [BKW12, KABW14]. Die in Rugby definierten Workflows Analyse, Entwurf, Implementierung und Test sind in HARVEST übernommen [Kru16]. Das ermöglicht eine agiles Vorgehen, bei dem es erlaubt ist, auf Veränderungen des verteilten Systems oder der Sicherheitsanforderungen sowie auf neue Erkenntnisse über Bedrohungen zu reagieren.

HARVEST definiert speziell auf die Gefahrenerkennung angepasste Workflows. In Abschnitt 7.1 wird der Analyseworkflow, eine spezielle Systemsicherheitsanalyse, vorgestellt. Auf dieser Analyse basiert der Entwurf, der in Abschnitt 7.2 beschrieben ist. Der Entwurf wird während der Implementierung (Kapitel 6.4) durchgeführt. In Abschnitt 7.3 ist beschrieben, wie die definierten Sicherheitstests durchgeführt werden und welche Konsequenzen deren Testergebnisse nach sich ziehen. Kapitel 7.4 beschreibt die Instanziierung sowie das evolutionäre Vorgehen von HARVEST.

[1] HARVEST ist ein Akronym von „Vorge**h**ensmodell zur Gef**a**hrener**r**kennung in **ve**rteilen **S**ys**t**emen“.

7.1 Systemsicherheitsanalyse

Die HARVEST-Systemsicherheitsanalyse fokussiert auf die Bewertung der Sicherheit verteilter Systeme. Die Systemsicherheitsanalyse besteht aus der Komponentenanalyse, der Berechtigungsanalyse und der Gefahrenanalyse. Sie fasst Erkenntnisse über die Struktur, das Berechtigungsmodell und den bisher bekannten Gefährdungen eines verteilten Systems zusammen. Sowohl Spezialisten der Komponenten eines verteilten Systems, als auch Spezialisten zu Gefahren und Sicherheitsanforderungen sind durch die Systemsicherheitsanalyse adressiert. So fasst sie das zur Verfügung stehende Wissen zur Konfiguration eines verteilten Systems zusammen. Veränderungen am verteilten System können sich auf das Ergebnis der Systemsicherheitsanalyse auswirken und erfordern evolutionäre Adaption der Systemsicherheitsanalyse. Den zugrundeliegenden Mechanismus der evolutionären Adaption beschreibt Abschnitt 7.4.

Die Systemsicherheitsanalyse beginnt mit der in Abschnitt 7.1.1 beschriebenen Komponentenanalyse. Die Komponentenanalyse führt zu einer Übersicht der Komponenten des verteilten Systems und einer Beschreibung der einzelnen Komponenten. Abschnitt 7.1.2 geht auf die darauf folgende Berechtigungsanalyse ein. Abschließend wird in Abschnitt 7.1.3 die Gefahrenanalyse beschrieben. Die Gefahrenanalyse ermittelt die Sicherheitsziele und die bekannten Gefahren des verteilten Systems.

7.1.1 Komponentenanalyse

Im Rahmen der Komponentenanalyse beschreiben die Spezialisten die im verteilten System eingesetzten Komponenten, deren Konfigurationsmöglichkeiten und die interkomponentalen Abhängigkeiten. Die Komponentenanalyse beginnt mit einem kompakten Modell des verteilten Systems. Das Modell dient der Kommunikation zwischen den Spezialisten des verteilten Systems. Kommunikationsmodelle sollten, aus Gründen der Verständlichkeit, nicht zu komplex werden [BKW12]. Abschnitt 7.1.1.1 stellt Metriken zur Messung von Komplexität in Diagrammen vor. Abschnitt 7.1.1.2 stellt das im Rahmen dieser Dissertation entwickelte UML-Profil „Sicherheitsdiagramm“ vor, den gewählten Ansatz zur verständlichen Modellierung von verteilte System. Sicherheitsdiagramme zeigen die Komponententypen eines verteilten Systems

mit deren Auftretenshäufigkeiten und den Abhängigkeiten zu den weiteren Komponententypen auf. Anschließend folgt die detaillierte Analyse der einzelnen Komponententypen. Diese Analyse umfasst die Schnittstellen und die Konfigurationsmöglichkeiten der Komponententypen. Die Komponentenanalyse hilft bei der Identifikation der Komponenten, deren Konfigurationen die für die späteren Sicherheitsanalysen notwendig sind.

7.1.1.1 Komplexität von UML-Diagrammen

Es besteht eine Korrelation zwischen der Komplexität und der Verständlichkeit eines Modells. Manso et. al. beschreiben, dass die strukturelle Komplexität eines Diagramms in einer direkt proportionalen Beziehung mit dessen kognitiven Komplexität steht [MGP03]. Darüberhinaus zeigen sie die Abhängigkeit zwischen der kognitiven Komplexität und der Verständlichkeit von Diagrammen. Dabei kommen sie zu dem Schluss, dass Diagramme mit steigender Komplexität unverständlicher werden. In der Arbeit beschreiben Manso et. al. mehrere Metriken zur Messung der strukturellen Komplexität eines UML-Diagramms. Sie unterscheiden hierbei zwischen Größenmetriken und Metriken der strukturellen Komplexität anhand von Klassendiagrammen. Diese Metriken beschreiben jeweils einen Aspekt des Diagramms anhand einer Kenngröße. Die Kenngrößen fließen in die Komplexität des Diagramms ein. Mansos Erkenntnisse werden durch die Beobachtungen von Miller bestätigt [Mil56]. Miller legte anhand seines 7 ± 2 – Gesetzes dar, wie viele Informationseinheiten ein Mensch gleichzeitig erfassen kann.

Mansos Metriken „Klassenanzahl“ (NC), „Attributanzahl“ (NA) und „Assoziationsanzahl“ (NAssoc) sind auf die in diesem Kapitel verwendeten Diagrammtypen übertragbar. NC wird in eine Metrik $G(d)$ überführt, welche die Anzahl der Komponententypen d verkörpert. NA wird in eine Metrik $E(d)$ überführt, welche die Anzahl der Eigenschaft von Gegenständen für das Diagramm d ermittelt. Die Anzahl der Beziehungen des Diagramm d zwischen zwei Gegenständen kann mit der Metrik $B(d)$ (basierend auf NAssoc) ermittelt werden. In Tabelle 7.1 sind die Metriken $G(d)$, $E(d)$ und $B(d)$ für die Diagrammtypen Objektdiagramm, Klassendiagramm mit Tagged Values, Komponentendiagramm und Sicherheitsdiagramm definiert.

Die Metrik $O(d)$ (siehe Formel 7.1) summiert die drei definierten Metriken NC, NA und NAssoc und trifft somit eine verallgemeinerte Aussage zur Kom-

Diagrammtyp	Metrik $G(d)$ (Gegenstände)	Metrik $E(d)$ (Eigenschaften)	Metrik $B(d)$ (Beziehungen)
Objektdiagramm	$\sum Objekte$	$\sum Attribute$	$\sum Objektbeziehung$
Klassendiagramm mit Tagged Values	$\sum Klassen$	$\sum TaggedValues + \sum Multiplizitäten$	$\sum Assoziationen$
Komponentendiagramm	$\sum Komponenten$	$\sum Dienste$	$\sum Kompositionskonnektoren$
Sicherheitsdiagramm	$\sum Komponenten$	$\sum Ikonen + \sum Dienste$	$\sum Kompositionskonnektoren$

Tab. 7.1: *Metrikendefinitionen $G(d)$, $E(d)$ und $B(d)$ der Diagrammtypen*

plexität eines Diagramms d. Es können auch mehrere Diagramme in einem Modell M zusammengefasst werden. Ein Modell M wird als eine Menge von Diagrammen verstanden. Die Komplexität des Modells M ergibt sich aus der Summe der Komplexitäten seiner Diagramme (siehe Formel 7.2). Die durch $O(M)$ ermittelte Komplexität ermöglicht es Modelle zu vergleichen, welche das gleiche verteilte System beschreiben.

$$O(d) = G(d) + E(d) + B(d) \tag{7.1}$$

$$O(M) = \sum_{d \in M} O(d) \tag{7.2}$$

Gefahren für die Gültigkeit: Die durch $O(d)$ ermittelte Komplexität eines Diagramms beruht auf zwei getroffenen Verallgemeinerungen. Beide könnten die Aussage bezüglich der Komplexität verfälschen. Erstens wird keine qualitative Differenzierung zwischen den Gegenständen, Eigenschaften und Beziehungen eines Diagramms vorgenommen. Das impliziert die Annahme, dass sich diese drei Aspekte gleichermaßen auf die Komplexität des jeweiligen Diagrams auswirken. Zweitens wird nicht zwischen den verschiedenen Ausprägungen der Gegenstände, Eigenschaften und Beziehungen in den verschiedenen Diagrammtypen differenziert. Dadurch wird die Annahme impliziert, dass sich diese drei Aspekte in den verschiedenen Diagrammtypen gleichermaßen auf die Komplexität auswirken. Da jedoch für Diagrammtypen gilt, dass eine größere Anzahl (von Gegenständen, Eigen-

schaften oder Beziehungen) zu einer höheren Komplexität des Diagramms führt, stellen $O(d)$ und $O(M)$ dennoch aussagekräftige Kenngrößen dar.

7.1.1.2 Das UML-Profil Sicherheitsdiagramm

Für die Fallstudien fokussieren wir uns auf grundlegende Aspekte eines verteilten Systems. Besonders interessieren die Komponenten des verteilten Systems sowie deren angebotenen und geforderten Dienste. Darüberhinaus ist die Frage, ob eine Komponente individuell konfigurierbar ist, von Bedeutung. Diese Aspekte lassen sich, wie in Beispiel 13 dargestellt, durch die Kombination von Objektdiagramm und Komponentendiagramm modellieren. Eine derartige Darstellung führt bei umfangreichen verteilten Systemen schnell zu sehr großen, unübersichtlichen Modellen. Einhundert Instanzen von K1 (anstelle von fünf) führen bereits zu einem Objektdiagramm mit über einhundert Objekten und Assoziationen. Derartige Diagramme können nach Miller vom Betrachter nur schwer erfasst und verinnerlicht werden.

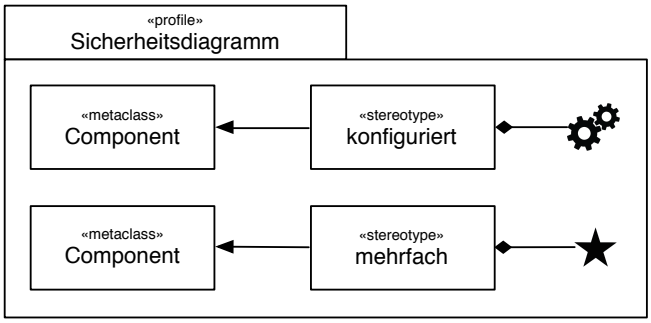


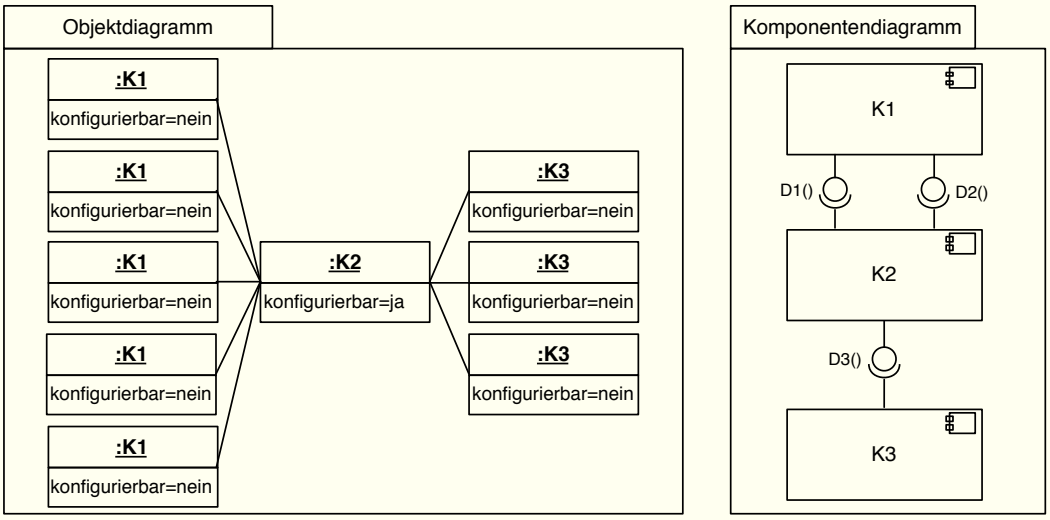
Abb. 7.1: *Das UML-Profil für Sicherheitsdiagramme*

Diese Dissertation stellt das UML-Profil „Sicherheitsdiagramme“ vor. Das UML-Profil Sicherheitsdiagramm (siehe Abbildung 7.1) definiert für Komponenten zwei Stereotypen. Das Stereotyp „konfiguriert“ wird mit einem Zahnradsymbol visualisiert und markiert Komponenten mit nichttrivialen Konfigurationen. Eine triviale Konfiguration ist beispielsweise ein reiner Identifikator einer Komponente. Das Stereotyp „mehrfach“ markiert die Komponententypen, von denen im verteilten System mehrere Instanzen vorkommen. Die Bedeutung des mehrfach-Stereotyps kommt der des im Klassendiagrammen verwendeten Asterisk ($\star$) sehr nahe. Deshalb übernahmen wir im Sicherheitsdiagramm den Asterisk für die Visualisierung des mehrfach-

Stereotyps. Das Sicherheitsdiagramm (Abbildung 15) modelliert das bereits in den Beispielen 13 und 14 modellierte verteilte System.

Beispiel 13

Darstellung des verteilten Systems als Modell m', bestehend aus dem Objektdiagramm o' und dem Komponentendiagramm k'. o' hat eine Komplexität von 26 und k' hat eine Komplexität von 9. Das führt zu der Gesamtkomplexität 35 des Modells m'.



$$
\begin{aligned}
O(o') &= G(o') + E(o') + B(o') &&= 9 + 9 + 8 &&= 26 \\
O(k') &= G(k') + E(k') + B(k') &&= 3 + 3 + 3 &&= 9 \\
O(m') &= O(o') + O(k') &&= 26 + 9 &&= 35
\end{aligned}
$$

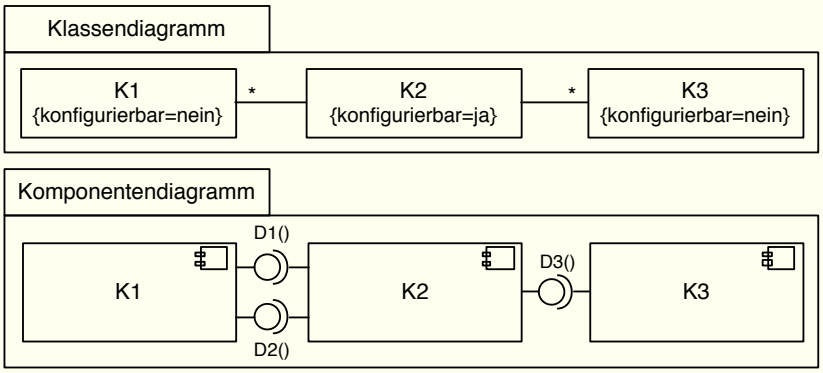
Die verschiedenen Instanzen eines Komponententyps im Beispiel 13 unterscheiden sich nicht. Eine essentielle Eigenschaft, die Objektdiagramme von Klassendiagrammen unterscheidet, wird in Beispiel 13 jedoch nicht genutzt. Der Wert des einzigen Attributs „konfigurierbar“ wird durch den Komponententyp und nicht durch die Instanz determiniert. Dieser Sachverhalt ist eine hilfreiche Erkenntnis und führte zu der Idee, die einzelnen Komponenten zu Komponententypen zu abstrahieren. Äquivalent zum Prinzip von RBAC [SS94, SCFY96], bei dem man die autorisierten Subjekte zu Rollen abstrahiert, werden nun auch die Objekte (auf die autorisiert wird) abstrahiert. Wir überführen das Objektdiagramm, durch eine Art Faltung, in ein Klassendiagramm. In Klassendiagrammen besitzen Attribute jedoch keine Wer-

te. Deshalb ist das Attribut „konfigurierbar“ (siehe Beispiel 13) samt Wert als Tagged-Value-Stereotyp modelliert. Folglich ersetzen wir das Objektdiagramm durch ein Klassendiagramm, welches jeden Komponententyp nur einmal enthält (siehe Beispiel 14).

In Beispiel 14 ist das verteilte System mit diesem Ansatz erneut modelliert. Betrachtet man dieses Modell, so wird eine weitere negative Eigenschaft deutlich. Die einzelnen Komponententypen und Assoziationen sind redundant modelliert. Das ist durch die Verteilung des Modells auf zwei Diagrammtypen begründet. Das führt dazu, dass Komponententypen und Assoziationen doppelt auf die Komplexität einwirken. Diese Eigenschaft motivierte die Entwicklung eines optimierten UML-Profils des Sicherheitsdiagramms. Das Sicherheitsdiagramm stellt die relevanten Aspekte eines verteilten Systems vereint dar. Sie modellieren die Komponententypen eines verteilten Systems mitsamt ihrer Schnittstellen, ihrer Multiplizitäten und ihrer Konfigurierbarkeit.

Beispiel 14

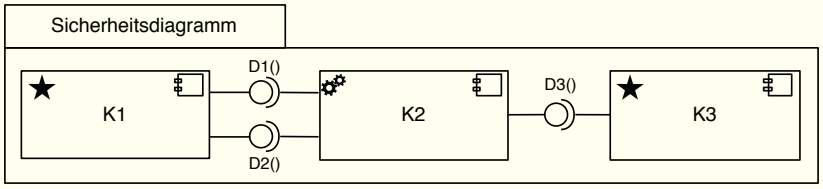
Die Darstellung des verteilten Systems als Modell m'', bestehend aus dem Klassendiagramm kl'' (gefaltetes Objektdiagramm o') und dem Komponentendiagramm k''. Das Diagramm kl'' besitzt die Komplexität 10 und k'' die Komplexität 9. Die Gesamtkomplexität des Modells m'' addiert sich somit zu 19.



$$
\begin{aligned}
O(kl'') &= G(kl'') + E(kl'') + B(kl'') &&= 3+5+2 &&= 10\\
O(k'') &= G(k'') + E(k'') + B(k'') &&= 3+3+3 &&= 9\\
O(m'') &= O(kl'') + O(k'') &&= 10+9 &&= 19
\end{aligned}
$$

Beispiel 15

Darstellung des verteilten Systems als Modell m''', bestehend aus dem Konfigurationsdiagramm ko'''. Das Diagramm ko''' besitzt eine Komplexität von 12, was gleichzeitig der Gesamtkomplexität des Modells m''' entspricht.



$$O(m''') = O(ko''') = G(ko''') + E(ko''') + B(ko''') = 3 + 6 + 3 = 12$$

In den Beispielen 13, 14 und 15 ist stets das gleiche verteilte System modelliert. Die Tabelle 7.2 gibt einen Überblick über die Komplexität der drei Modelle. Das Modell aus Beispiel 13 hat mit einem Wert von 35 die höchste Komplexität. Dieser Wert ist der Ausgangswert für den prozentualen Vergleich der letzten Tabellenspalte. Die in Beispiel 13 vorgenommene Überführung vom Objektdiagramm zum Klassendiagramm führt bereits fast zur Halbierung der Komplexität. Durch die Verschmelzung der Konzepte von Komponenten- und Klassendiagrammen wird die Komplexität des Modells in Beispiel 15 auf etwa ein Drittel der ursprünglichen Komplexität reduziert.

Diagrammtypen	Beispiel	$O(M)$	$O(M)$ in %
Objektdiagramm und Komponentendiagramm	Beispiel 13	35	100 %
Klassendiagramm und Komponentendiagramm	Beispiel 14	19	$\approx$ 54 %
Sicherheitsdiagramm	Beispiel 15	12	$\approx$ 34 %

Tab. 7.2: *Metriken $G(d)$, $E(d)$ und $B(d)$ für verschiedene Diagrammtypen*

Sicherheitsdiagramme fungieren als Analysemodell, indem sie sicherheitsrelevante Aspekte eines verteilten Systems zusammenfassen. Außerdem fungieren sie als Kommunikationsmodelle [BKW12] für die Interaktion zwischen den verschiedenen Stakeholdern, welche bei der Sicherheitsanalyse

des verteilten Systems interagieren. Sicherheitsdiagramme sollen die Lesbarkeit, im Vergleich zu Zugriffsmatrizen und deren Repräsentationsmöglichkeiten (siehe Abschnitt 7.1.2), verbessern. Aufgrund dieser Eigenschaften werden verteilte Systeme in dieser Dissertation mit Sicherheitsdiagrammen modelliert.

7.1.2 Berechtigungsanalyse

Das in Abschnitt 7.1.1.2 vorgestellte Sicherheitsdiagramm beinhaltet unter anderen die Informationen einer Zugriffsmatrix [Lam74, SS94]. Folglich kann ein Sicherheitsdiagramm auch in eine Zugriffsmatrix überführt werden. Überführt man das Sicherheitsdiagramm des Beispiels 15, so entsteht die in Tabelle 7.3 abgebildete Zugriffsmatrix. Die Überführung von Sicherheitsdiagrammen in eine Zugriffsmatrix ist nicht verlustfrei. Denn Zugriffsmatrizen treffen weder zur Konfigurierbarkeit noch zur Multiplizität der Komponenten entsprechende Aussagen. Diese Informationen gehen bei der Überführung verloren. Aufgrund der verlustbehafteten Überführung zur Zugriffsmatrix ist eine vollständige (Rück-)überführung einer Zugriffsmatrix in ein Sicherheitsdiagramm nicht möglich.

bietet an / erfordert	Komponente 1		Komponente 2
	Dienst 1	Dienst 2	Dienst 3
Komponente 1			
Komponente 2	✓	✓	
Komponente 3			✓

Tab. 7.3: *Zugriffsmatrix der Komponenten eines verteilten Systems*

Die Zugriffsmatrix in Tabelle 7.3 ist eine für HARVEST optimierte Derivation zur Berechtigungsanalyse. Einerseits soll das Konzept dieser Zugriffsmatrix für UML-kundige Betrachter durch die im Achsenursprung dargestellten Lollipop-Notation verdeutlicht werden. Andererseits werden in den Spaltenköpfen nicht nur die Komponenten, sondern auch deren bereitgestellten Schnittstellen geführt. In den Zellen selbst werden nur noch die Schnittstellen markiert, die der zugreifende Akteur erfordert. Das ist eine

Platzoptimierung, mit der auch komplexere verteilte Systeme kompakt dargestellt werden können.

Für Zugriffsmatrizen existieren drei verschiedene Repräsentationsarten [BD04]. Diese alternativen Repräsentationen sind globale Zugriffstabellen, Zugriffskontrolllisten und Berechtigungen. Diese Repräsentationen können über Modelltransformationen erstellt werden. Der Ansatz der Modelltransformationen ist im Bereich der Modellierung in der UML bereits verbreitet. Einige gebräuchliche Modelltransformation auf Basis von UML-Diagrammen sind in Tabelle 7.4 den auf Sicherheitsdiagrammen basierenden Modelltransformationen gegenübergestellt. UML-Klassendiagramme lassen sich beispielsweise in Object Role Modeling (ORM) überführen [HB99]. Aus solchen ORM-Diagrammen kann wiederum eine DDL[2] automatisiert erzeugt werden.

Ausgangsmodell	**Transitionsmodell**	**Transformiertes Modell**
Klassendiagramm	ORM-Diagramm	DDL (SQL)
Klassendiagramm	Ecore-Modell [BCHK07]	Java-Code Swift-Code
Sicherheitsdiagramm	Zugriffsmatrix	globale Zugriffstabelle Zugriffskontrollliste Berechtigungen

Tab. 7.4: *Beispiele für Modelltransformationen mittels Transitionsmodell*

Ein Sicherheitsdiagramm, das am Anfang der Analyse erstellt wurde, kann über eine Zugriffsmatrix in spezielle technischere Sichten überführt werden. Diese Repräsentationen basieren auf unterschiedlichen Konzepten der Berechtigungsverwaltung, welche unterschiedliche Vor- und Nachteile haben. In der Berechtigungsanalyse kann unter diesen Visualisierungskonzepten zur Beschreibung der Berechtigungen im verteilten System frei gewählt werden. Tabelle 7.5 zeigt exemplarisch eine Zugriffskontrollliste die von der Zugriffsmatrix aus Tabelle 7.3 überführt wurde.

Die Zugriffskontrollliste in Tabelle 7.5 enthält, im Vergleich zur Zugriffsmatrix, zusätzliche Informationen. Die verwendeten Standards und eine Beschreibung der bereitgestellten Schnittstelle erlauben einen detaillierteren Einblick in das verteilte System. Die freie Wahl der Repräsentation gewährt

[2] Die DDL ist eine Teilsprache der SQL

Konsument	Anbieter	Schnittstelle	Standards	Beschreibung
Komponente 2	Komponente 1	Dienst 1	ftp, xml	Vollständige Synchronisierung der Messwerte
Komponente 2	Komponente 1	Dienst 2	http, json	Inkrementelle Synchronisierung der Messwerte
Komponente 3	Komponente 2	Dienst 3	WebSocket, json	Anbieten von Steuerinformationen

Tab. 7.5: *Beispiel einer Zugriffskontrollliste eines verteilten Systems*

beim Modellieren Flexibilität, um die essenziellen Aspekte des konkreten verteilten Systems herauszustellen.

7.1.3 Gefahrenanalyse

Der Prozess der Meta Threat Facility (siehe Abbildung 5.3 in Kapitel 5) definiert drei Strategien zur Erkennung von gefährlichen Konfigurationen. Der Ausgangspunkt der Gefahrenanalyse sind die in dem MTF-Prozess erkannten Gefahrenklassen des verteilten Systems. Das Ziel ist die Erkennung der konkreten Gefahren dieser Gefahrenklasse, die im untersuchten Konfiguration Unter Test (siehe Abschnitt 2.4.4) existieren.

Während der Gefahrenanalyse können verschiedene, mit dem verteilten System in Verbindung stehend, Personengruppen einbezogen werden. Neben Experten, Verantwortlichen und Benutzern sind noch weitere Kreise denkbar. Das Ziel der Gefahrenanalyse ist die Beschreibung von Gefahrenklassen, welche aus einer fehlerhaften Konfiguration des verteilten Systems resultieren. Diese Definition besteht aus einem aussagekräftigen Namen, einer abstrakten Definition, Beispielen und einer Gefahrenbewertung der Gefahrenklasse. Eine solche Definition kann wie in Beispiel 16 auf vorgefertigte Karteikarten geschrieben werden und in einem Projektraum aufgehängt werden. Die abstrakte Definition soll die Gesamtheit der Gefahren der Gefahrenklassen umfassen. Als Beispiele können neben fiktiven Begebenheiten auch real (ehemals) existierende Konfigurationen herangezogen werden. Und aus der Gefahrenbewertung soll hervorgehen, warum von der beschriebenen Gefahrenklassen eine Gefahr ausgeht.

Beispiel 16

Gefahrenklassen Witwe	
Beschreibung	Eine Person darf über das Netzwerk auf einen Server zugreifen, darf sich auf diesen jedoch nicht anmelden.
Beispiele	Herr Mustermann hat Netzwerkzugriff auf den IP-Bereich 10.10.42.96/28. In diesem Bereich liegt jedoch der Server der Personalverwaltung mit der IP-Adresse 10.10.42.107. Auf diesen Server kann (und darf) sich Herr Mustermann nicht anmelden.
Gefahren-bewertung	Das Prinzip der doppelten Sicherheit, das für einen Dienst zwei wirksame Schutzmaßnahmen fordert, ist gebrochen. Durch eine neue Sicherheitslücke des Betriebssystems des Servers könnte er somit Zugriff erlangen.

7.2 Sicherheitsentwurf

Der Sicherheitsentwurf fußt auf den Ergebnissen des Analyse-Workflows Analyse. Während durch die Komponentenanalyse und die Berechtigungsanalyse wichtige Informationen über die aktuelle Gestalt des verteilten Systems zusammengefasst sind, werden durch die Gefahrenanalyse die Gefahrenklassen identifiziert, deren Gefahren zukünftig vom Sicherheitsframework erkannt werden sollen. Dazu wird für jede Gefahrenklasse ein Prozess durchlaufen, dessen vier aufeinander aufbauende Stufen in Abbildung 7.2 dargestellt sind. An der in Beispiel 16 beschriebenen Gefahrenklassen der „Witwe“ werden diese Prozessschritte in Beispiel 17 durchlaufen.

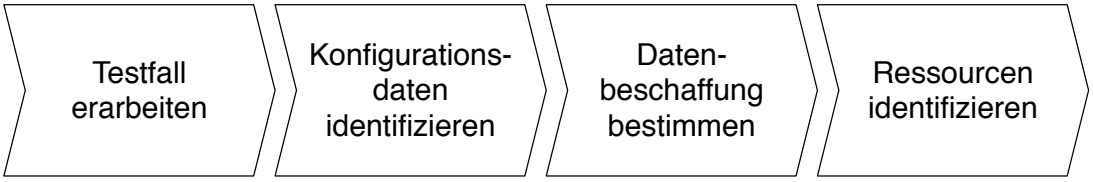


Abb. 7.2: *Prozessschritte des Sicherheitsentwurfs-Workflow*

Sicherheitstest erarbeiten: Das Muster, mit der die Gefahren einer Gefahrenklasse erkannt werden sollen, ist zunächst nicht zwingend offensichtlich. Eine naheliegende Möglichkeit geeignete Muster zu finden ist die Zusammenarbeit mit den Spezialisten, die sich mit Gefahrenklasse und den relevanten Komponenten des verteilten Systems auskennen. Hierbei können auch mehrere Sicherheitstests für die Gefahrenklasse erarbeitet werden. Dann liegt es in der Entscheidungsgewalt der Stakeholder wie fortgefahren wird. Es ist nicht immer ideal, nur einen Sicherheitstest zu verfolgen. Denn wenn man mehrere Sicherheitstests identifiziert, kann man ausprobieren, welche der Beste ist. Ebenfalls können sich Sicherheitstests ergänzen und nur die Kombination beider führt zu Erfolg. Wird keine geeignetes Muster gefunden, kann dieser Schritt hinten angestellt werden. Die folgenden Prozessschritte erfolgen dann ohne definierten Sicherheitstest. Das geschieht mit dem Ziel, weitere hilfreiche Erkenntnisse auf dem Weg zu einem geeigneten Sicherheitstest zu entwickeln.

Konfigurationsdaten identifizieren: In diesem Schritt werden die für die Erkennung der Gefahren der Gefahrenklasse notwendigen Konfigurationsdaten bestimmt. Wenn im vorhergehenden Schritt ein Sicherheitstest defi-

niert wurde, ist im Regelfall auch schlüssig, um welche Daten es sich handelt. Liegt noch keine Sicherheitstest vor, wählt man die Komponenten, die wahrscheinlich bei der Erkennung der Gefahren der Gefahrenklasse eine Rolle spielen. Hierfür wird zusammengefasst, um welche Konfigurationsdaten es sich handelt, auf welchen Komponententypen diese Konfigurationen vorliegen und welche Instanzen der Komponententypen wichtig sind. Es ist beispielsweise sinnvoll, die Instanzen von Test- und Entwicklungsumgebungen nicht mit in die Auswertung miteinzubeziehen.

Datenbeschaffung bestimmen: Die Eigenarten und Einsatzkontexte der verschiedenen Komponententypen können sich erheblich voneinander unterscheiden. Deshalb muss für jeden Komponententyp eine individuelle Strategie zur Datenbeschaffung ausgearbeitet werden. Bei der Wahl der Strategie spielen viele Faktoren eine Rolle. Das Übertragungsprotokoll und das Datenformat der Exportschnittstelle geben die Technologien vor, mit denen die Konfigurationen ausgelesen werden müssen. Hat der Auslesevorgang negative Auswirkungen auf den produktiven Betrieb, können Restriktionen, wie die Beschränkung auf ein bestimmtes Zeitfenster, gelten. Die Dauer eines Auslesevorgangs bestimmt, wie häufig neue Daten von der Komponente bezogen werden können. Und die Volatilität der Konfigurationsdaten beeinflusst, wie häufig neue Daten von der Komponente bezogen werden sollten. Darüber hinaus können noch weitere Faktoren Einfluss auf die Beschaffung der Konfigurationsdaten nehmen. Neben der genauen Beschreibung, wie die Daten ausliest, hält man in diesem Prozessschritt auch das Rational, also die Entscheidungsgrundlage, fest. Hierfür bietet sich in größeren Projekten mit vielen Beteiligten der Einsatz eines Rational-Management-Werkzeugs an, wie es Wolf und Dutoit [WD05] anheimstellen.

Ressourcen identifizieren: Im letzten Prozessschritt werden die für die Umsetzung benötigten Ressourcen ermittelt. Es handelt sich in erster Linie um die Identifikation der Spezialisten, deren Gefahrenwissen (wie in Abschnitt 3.6 beschrieben) zu entsprechenden Experten externalisiert werden müssen. Wenn für das geforderte Gefahrenwissen bereits ein Experte besteht, ist dieser Experte als Ressource zu nennen. Darüber hinaus müssen weitere notwendige Ressourcen wie beispielsweise erforderliche Berechtigungen, Exportmechanismen, Softwarelizenzen und Hardware identifiziert werden. Die Ressourcen sind mit den notwendigen Vorlaufzeiten und Verfügbarkeiten aufzulisten.

Beispiel 17
Der Sicherheitsentwurf der Gefahrenklasse Witwe

Sicherheitstest erarbeiten: Für jeden Benutzer werden die Server, auf die er über das Netzwerk zugreifen kann, sowie die Server, an denen er sich anmelden kann, ermittelt. Wenn für einen Benutzer auf einem Server Netzwerkzugriff aber kein Anmelderecht besteht, ist das eine Witwe.
Konfigurationsdaten identifizieren: Die Benutzerinformationen können aus der Benutzerverwaltung, die Netzwerkberechtigungen aus der Firewall-Konfiguration und die Anmeldeberechtigungen aus den Server-Konfigurationen bezogen werden.
Datenbeschaffung bestimmen: Die Server werden zentral über eine Berechtigungsverwaltung administriert, deren Daten ebenfalls in einer Datenbank liegen. Die Daten der Benutzerverwaltung und die der Berechtigungsverwaltung liegen in einer nicht direkt aufrufbaren Datenbank. Deshalb müssen die Konfigurationen über das mitternächtlich stattfindende Backup der Datenbank bezogen werden und 2 Uhr morgens mit den Backupwerkzeugen in eine eigene CUSTODIAN-Datenbank kopiert werden. Von dort aus kann CUSTODIAN über JDBC auf die Daten zugreifen. Die UB-Firewall hat eine Exportfunktion, mit der man die Konfigurationen auf andere Systeme zeitgesteuert exportieren lassen kann. Im zeitlichen Einklang mit der Datenversorgung der Benutzerverwaltung wird die Exportfunktion so konfiguriert, dass die Konfigurationen als XML-Datei auf das CUSTODIAN-System exportiert werden.
Ressourcen identifizieren: Für die drei Komponenten Benutzerverwaltung, UB-Firewall und Berechtigungsverwaltung wird für die Auswertung jeweils eins Spezialist der jeweiligen Fachabteilung benötigt; diese können bei Bedarf spontan angesprochen werden. Ebenso muss zwischen den drei Systemen und dem CUSTODIAN-System jeweils eine Firewallfreischaltung, welche eine Vorlaufzeit von bis zu fünf Werktagen, für den Export eingerichtet werden. Für die Benutzer- und Berechtigungsverwaltung muss auf dem CUSTODIAN-System eine Datenbank bereitgestellt werden, das dauert bis zu 12 Werktage. Das Gefahrenwissen der Gefahrenklasse Witwe können die Verantwortlichen der Komponenten Benutzerverwaltung und Berechtigungsverwaltung gemeinsam aufbringen.

7.3 Sicherheitstest

Aus dem Workflow Umsetzung resultiert eine für die Gefahrenklasse abgestimmte Anordnung von Experten auf dem Sicherheitsblackboard. Eine solche Expertenanordnung entspricht einem Sicherheitstest der zu überprüfenden Gefahrenklasse. Ein Sicherheitstest ordnet die zu überprüfenden Testobjekte (siehe Abschnitt 2.4.3) entweder der Nullhypothese (es liegt keine Gefahr vor) oder der Gegenhypothese (es liegt eine Gefahr vor) zu. Im Workflow Sicherheitstest (siehe Abbildung 7.3) wird der Sicherheitstest im ersten Schritt ausgeführt und das verteilte System validiert. Die Anwendung des Sicherheitstests erzeugt die Nullmenge und Gegenmenge, welche disjunkt sind. Liegt dem Sicherheitstest eine ideale Erkennungsheuristik zugrunde, so entspricht die Gegenmenge der Menge der tatsächlich im verteilten System existierenden Gefahren. Im zweiten Schritt des Sicherheitstest-Workflows ist die Gegenmenge auf α-Fehler und die Nullmenge auf β-Fehler zu überprüfen. Durch diese Überprüfung werden die Null- und die Gegenmenge jeweils in zwei Mengen unterteilt.

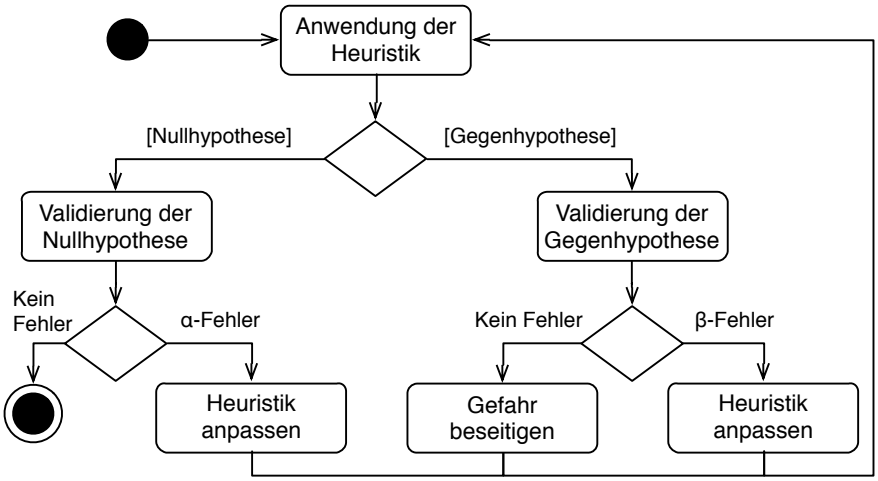


Abb. 7.3: *Der Sicherheitstest-Workflow als Aktivitätsdiagramm*

Aus dem Test-Workflow resultieren vier Mengen. Auf die unterschiedlichen Mengen muss unterschiedlich reagiert werden. Die notwendigen Maßnahmen sind in der Entscheidungstabelle 7.6 dargelegt. Auf Testobjekten bei denen weder eine Gefahr noch eine fehlerhafte Erkennung vorliegt, ist keine weitere Maßnahme notwendig. Stellt das Testobjekt eine Gefahr dar und die Erkennung ist korrekt, muss die Gefahr im verteilten System beseitigt und der Test erneut gestartet werden. Wenn eine fehlerhafte Erkennung vor-

liegt, muss man die Heuristik des Sicherheitstests anpassen. Hierfür muss man die Expertenanordnung auf dem Sicherheitsblackboard korrigieren. Basierend auf den neuen Erkenntnissen begibt man sich wieder in den Analyseworkflow und korrigiert die Analyse, den Entwurf und die Umsetzung, um danach den Test erneut zu durchlaufen.

	keine Gefahr	**Gefahr**
Sicherheitstest ergibt Nullhypothese	Richtige Entscheidung ⇒ Keine Maßnahme	β-Fehler ⇒ Sicherheitstest anpassen
Sicherheitstest ergibt Gegenhypothese	α-Fehler ⇒ Sicherheitstest anpassen	Richtige Entscheidung ⇒ Gefahr beseitigen

Tab. 7.6: *Entscheidungstabelle für Maßnahmen nach dem Test*

Die sich daraus ergebenden Entwicklungsiterationen werden durch α- und β-Fehler ausgelöst. Das Erkennen von β-Fehlern führt jedoch wieder zur Problematik dieser Dissertation: dem Erkennen von Gefahren in verteilten Systemen. β-Fehlern können jedoch durch eine bereits bekannte Technik aus der Testtechnik und der Entwicklung robuster Systeme erkannt werden. Die Fehlerinjektion (siehe Abschnitt 9.2.2) ist eine Möglichkeit, mit der man zwar nicht die β-Fehler selbst, aber zumindest mögliche Ursachen von β-Fehlern aufdecken kann. Durch die gezielte Fehlkonfiguration des verteilten Systems kann man überprüfen, ob ein Sicherheitstest die injizierten Fehler erkennt. Um das produktive, verteilte System nicht zu gefährden, bietet es sich bei der Fehlerinjektion an, auf eine Testumgebung zurückzugreifen.

7.4 Das HARVEST-Ereignismodell

Die in Abbildung dargestellte Taxonomie 7.4 klassifiziert die Ereignisse, die den HARVEST-Prozess aufwecken können. HARVEST ist somit in der Lage, auf Veränderungen des verteilten Systems, auf zeitliche Degeneration von Konfigurationen, auf neues Gefahrenwissen von Spezialisten und auf neue Gefahrenklassen in der MTF zu reagieren. Das entspricht den Ereignissen, die auch die Sicherheitsüberprüfung durch CUSTODIAN initiieren (siehe Abbildung 6.2 des Kapitels 6.1). Zusätzlich lösen auch die gewonnenen Ergebnisse der HARVEST-Unterprozesse den Hauptprozess aus.

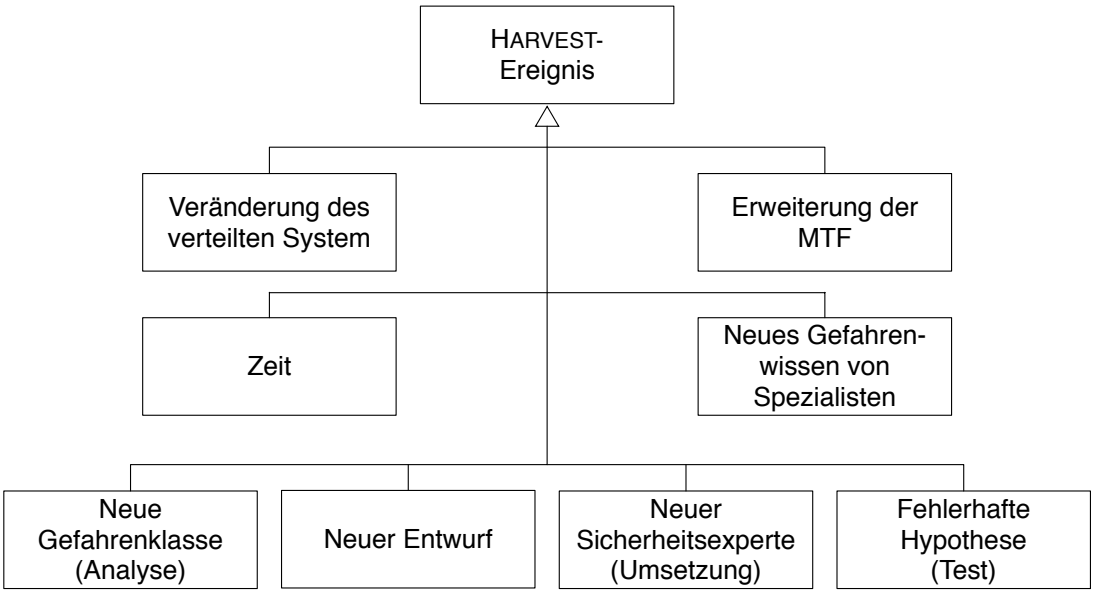


Abb. 7.4: *Taxonomie der Auslöser des Harvest-Hauptprozesses*

HARVEST erlaubt ein paralleles Durchlaufen der vier Workflows. Deshalb bietet es sich an, agile Methoden, welche einen schnellen und unkomplizierten Informationsaustausch der beteiligten Akteure ermöglichen, einzusetzen. Das Rugby-Modell [KABW14] spricht die Schnittstellen zwischen den einzelnen Workflows gezielt an: Szenarien können eingesetzt werden, um die Schnittstelle zwischen Analyse und Entwurf zu schließen. Die Schnittstelle zwischen Entwurf und Umsetzung kann mit Prototypen und Issue-Management geschlossen werden. Kontinuierliche Integration und Lieferung (engl. continuous integration/delivery) erlauben es den Spezialisten,

stets den aktuellen Entwicklungstand bereitzustellen, denn deren Gefahrenwissen ist bei der Validierung der α- und β-Fehler gefragt.

Abbildung 7.5 zeigt den Prozessablauf von HARVEST exemplarisch anhand von drei Veränderungen:

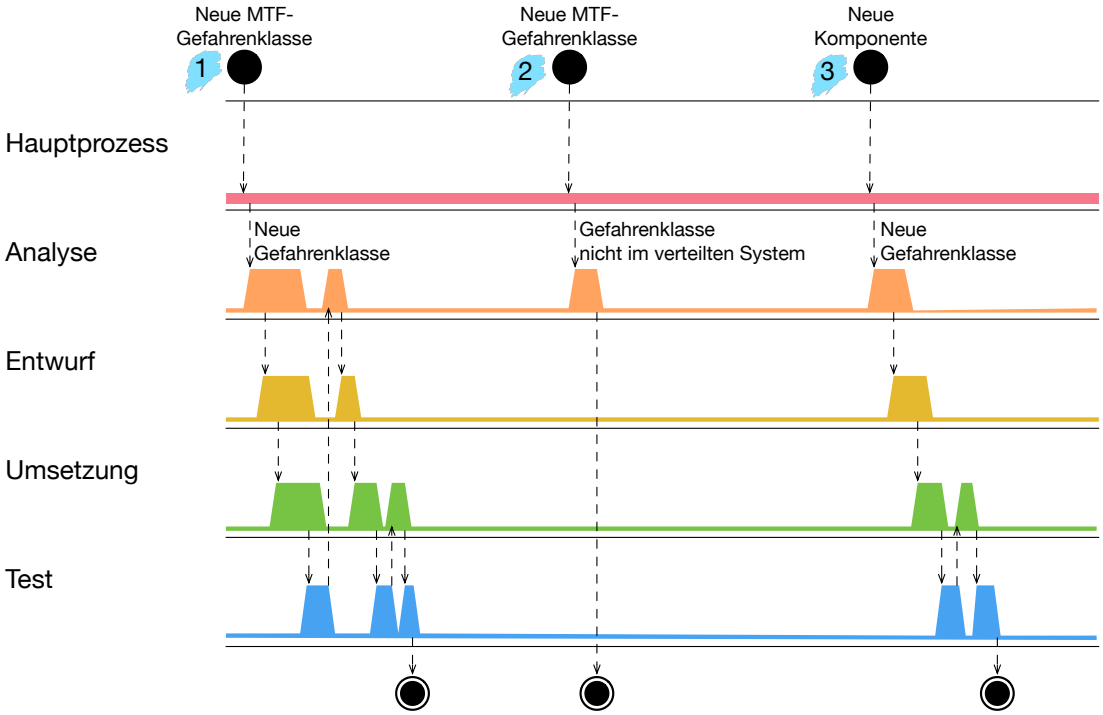


Abb. 7.5: *Exemplarischer Lebenszyklus des Harvest-Prozessmodells*

[1] Eine neue Gefahrenklasse der MTF löst die erste Veränderung aus, und der Hauptprozess startet den Analyseprozess. Die Analyse kommt zu dem Ergebnis, dass die neue MTF-Gefahrenklasse auch in diesem verteilten System existiert. Schon während der Analyse führen erste Teilergebnisse der Analyse zur Notifikation des Hauptprozesses, der wiederum den Entwurfsprozess startet[3]. Die Abbildung verdeutlicht, dass die HARVEST-Unterprozesse parallel aktiv sein können. Die ersten Entwurfsergebnisse führen zur Initiierung des Umsetzungsprozesses, der einen Sicherheitstest hervorbringt. Der Sicherheitstest wird im Testprozess geprüft, und einige

[3] Zur Vereinfachung des Modells in Abbildung 7.5 sind die Notifikationen des Hauptprozesses durch Unterprozesse sowie die darauf folgenden Notifikationen weiterer Unterprozesse als direkte Notifikation zwischen den beiden Unterprozessen modelliert.

Gefahrenhypothesen können invalidiert werden. Ein weiterer Durchlauf des Analyse-, des Entwurfs- und des Umsetzungsprozesses bringt einen verfeinerten Sicherheitstest hervor. Auch bei diesem verfeinerten Sicherheitstest können im Testprozess Gefahrenhypothesen invalidiert werden. Ein weiterer Durchlauf des Umsetzungsprozesses bringt einen Sicherheitstest hervor, dessen Gefahrenhypothesen im Testprozess nicht invalidiert werden können. Deshalb startet der Hauptprozess keinen weiteren Unterprozess. Somit steht auf dem CUSTODIAN-Sicherheitsblackboard, in Form von Sicherheitsexperten, das notwendige Gefahrenwissen zur Verfügung, um die Gefahren der neuen MTF-Gefahrenklasse erkennen zu können.

[2] Eine weitere neue Gefahrenklasse der MTF löst die zweite Veränderung aus und weckt den Hauptprozess erneut auf. Dieser startet den Analyseprozess, der zum Ergebnis hat, dass die neue MTF-Gefahrenklasse für dieses verteilte System nicht relevant ist.

[3] Eine neue Komponente verändert die Struktur des verteilten Systems, und der Hauptprozess wacht auf. Der Hauptprozess startet den Analyseprozess, indem erkannt wird, dass eine neue Gefahrenklasse auf das verteilte System einwirkt. Die Analyseergebnisse initiieren den Entwurfsprozess und dessen Ergebnisse den Umsetzungsprozess. Der Umsetzungsprozess führt zu einem Sicherheitstest, dessen Gefahrenhypothesen im Testprozess teilweise invalidiert werden. Eine weitere Iteration des Umsetzungsprozesses führt zu einem im Testprozess nicht invalidierten Sicherheitstest. Ein weiterer Umsetzungsprozess muss nicht durchlaufen werden.

8 Evaluation

Dieses Kapitel beschreibt die Evaluation der Forschungshypothesen dieser Dissertation mit Hilfe des Sicherheitsframeworks CUSTODIAN und der Referenzimplementierung KUSTOS. Abbildung 8.1 zeigt die Forschungsmethoden die bei der Evaluation durch drei Fallstudien zum Einsatz kamen. Die Fallstudien evaluieren, ob CUSTODIAN in der Lage ist, bisher unbekannte Gefahren in den Konfigurationen verteilter Systeme zu erkennen (siehe Ausgangshypothese in Abschnitt 4) [RH09].

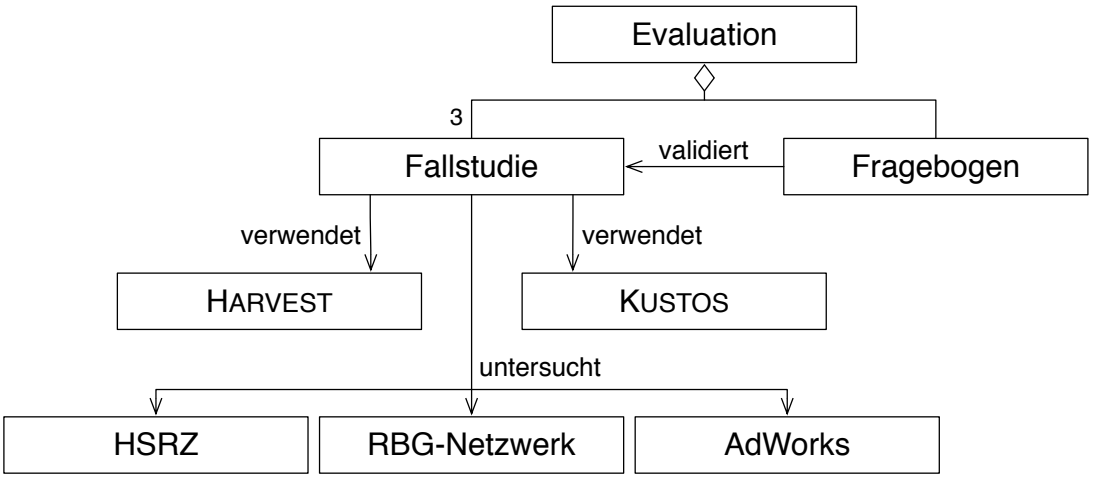


Abb. 8.1: *Überblick über die bei der Evaluation eingesetzten Methoden und die evaluierten Artefakte*

Im Rahmen der ersten Fallstudie einem Hochsicherheitsrechenzentrum (kurz HSRZ), kam die Referenzimplementierung KUSTOS erstmals zum Einsatz. Der Betreiber des HSRZ ist ein Technologieunternehmen mit einer hierarchisch geführten Organisation und einer komplexen Abteilungsstruktur. Der Hauptfokus ist die teils geheime Forschung an neuen Technologien sowie die sichere Gewährleistung der von dem Technologieunternehmen vertriebenen Produkte.

Die zweite Fallstudie wurde in einem thematisch verwandten verteilten System, dem von der Rechnerbetriebsgruppe (RBG) verwalteten Netzwerk, durchgeführt. Die Fakultät für Mathematik und Informatik (FMI), für die die RBG Netzwerkdienstleistungen bereitstellt, wie auch die Technische Universität München, der die Fakultät für Mathematik und Informatik (FMI) angehört, haben keine hierarchische Struktur. Die Unterorganisationen, wie Lehrstühle, Verwaltungseinheiten oder Fachschaften, haben weitgehende eigene Entscheidungsbefugnisse, die sie auch bei der Organisation ihrer Netzwerke ausschöpfen. Als Universität möchte die Technische Universität München (TUM) ihre Lehre offen vermitteln und ihre Lehrstoffe zur freien Verfügung stellen. Die TUM ist eine Organisation, die das offene Vermitteln von Wissen vorantreibt und ihre Lehrstoffe möglichst frei bereitstellen möchte. Gleichzeitig müssen jedoch persönliche Daten, Zensuren und Forschungsergebnisse vor unbefugten Zugriff geschützt werden. Wie auch beim HSRZ müssen für das RBG-Netzwerk Firewallregel ausgewertet werden. Ein wesentlicher Unterschied ist, dass die RBG die Server in diesen Netzwerken vorwiegend nicht selbst betreibt. Auf die RBG wurden wir aufmerksam, da wir am eigenen Lehrstuhl feststellen mussten, dass wir keinen Überblick über die Firewallregeln unsers Netzwerks haben und für den Lehrstuhl somit Gefahren entstanden sind.

Die dritte Fallstudie fand in einer unterschiedlichen Domäne der financeTec AG satt. Die financeTec AG betreibt mit einem verteilten System den Online-Dienst AdWorks2go und ist eine Organisation mit wenigen Abteilungen und flachen hierarchischen Strukturen. Das Ziel der financeTec AG ist, die Software AdWorks stetig weiterzuentwickeln und den Dienst AdWorks2go sicher und korrekt zur Verfügung zu stellen. Bei der financeTec AG war der Anstoß der Untersuchungen die Frage, ob die Rechnungen für AdWorks2go korrekt gestellt werden und ob von diesen Gefahren ausgehen.

Die Fallstudie beim HSRZ ist in Abschnitt 8.1, die Fallstudie mit der RBG ist in Abschnitt 8.2 und die AdWorks-Fallstudie ist in Abschnitt 8.3 beschrieben. Der nach der Durchführung der Fallstudien ausgehändigte Fragebogen ist in Abschnitt 8.4.1 beschrieben, und in Abschnitt 8.4 befindet sich die Auswertung der Fallstudien.

8.1 Fallstudie Hochsicherheitsrechenzentrum

Von einem Technologieunternehmen werden einige Informationstechnologie-Dienste bereitgestellt. Diese Dienste unterstützen unter anderen die interne Verwaltung, die Entwicklung und die Produktion oder können auch selbst ein von dem Technologieunternehmen angebotenes Produkt darstellen. Diese Dienste, die besonders hohen Sicherheitsanforderungen haben, betreibt das Technologieunternehmen in einem Hochsicherheitsrechenzentrum (im Folgenden HSRZ genannte).

Die vom HSRZ bereitgestellten Dienste stellen unterschiedliche Sicherheitsanforderungen. Einige für den Deutschen Staat bereitgestellte Dienste erfordern die Einhaltung der Sicherheitsvorgaben des Bundes für Verschlusssachen[1] erfüllt werden. Andere Dienste erfordern die Einhaltung internationaler Standards wie die des amerikanische Sarbanes-Oxley Act (SOA) [SO02]. Eine Reihe von internen Standards, wie beispielsweise das Zwei-Rechte-Prinzip (siehe Abschnitt 8.1.3.5), gilt für alle im Rechenzentrum betriebenen Dienste. Wenn die von einem Dienst erforderlichen Standards nicht eingehalten werden, hat das Konsequenzen für die Sicherheit des Dienstes. Deshalb ist ein permanenter Prozess notwendig, mit dem man Risiken identifiziert und eliminiert. In dieser Fallstudie wird die Realisierbarkeit dieses Prozesses anhand des Sicherheitsframeworks CUSTODIAN und der Referenzimplementierung KUSTOS untersucht.

[1] Verschlusssachen - Nur für den Dienstgebrauch (VS-NfD) [Deu80]

8.1.1 Komponentenanalyse

Das Sicherheitsdiagramm in Abbildung 8.2 ist ein Modell des HSRZ. Die Server stellen die vom HSRZ angebotenen Dienste zur Verfügung. Deshalb sind die Server sowohl der funktionale Kern des Rechenzentrums als auch das am stärksten zu beschützende Gut. Eine direkte Verbindung von außerhalb des Rechenzentrums ist prinzipiell verboten. Nutzer (z. B. Kunden) müssen deshalb zuerst mit Citrix ein Terminalverbindung initiieren. Vom Terminal aus können sie auf ihre Server im Rechenzentrum zugreifen. Das setzt jedoch eine Freischaltung auf der User-Firewall sowie eine entsprechende Gruppenmitgliedschaft im Lightweight Directory Access Protocol (LDAP) voraus. Derartige Freischaltungen können im Berechtigungsverwaltungssystem über die Antragsverwaltung beantragt werden. Ein Nutzer kann auf den für ihn erreichbaren Servern die für ihn freigeschalteten Dienste verwenden. Die Nutzung eines Dienstes erfordert somit neben der Berechtigung am Dienst selbst auch einen entsprechende Netzwerkfreischaltung.

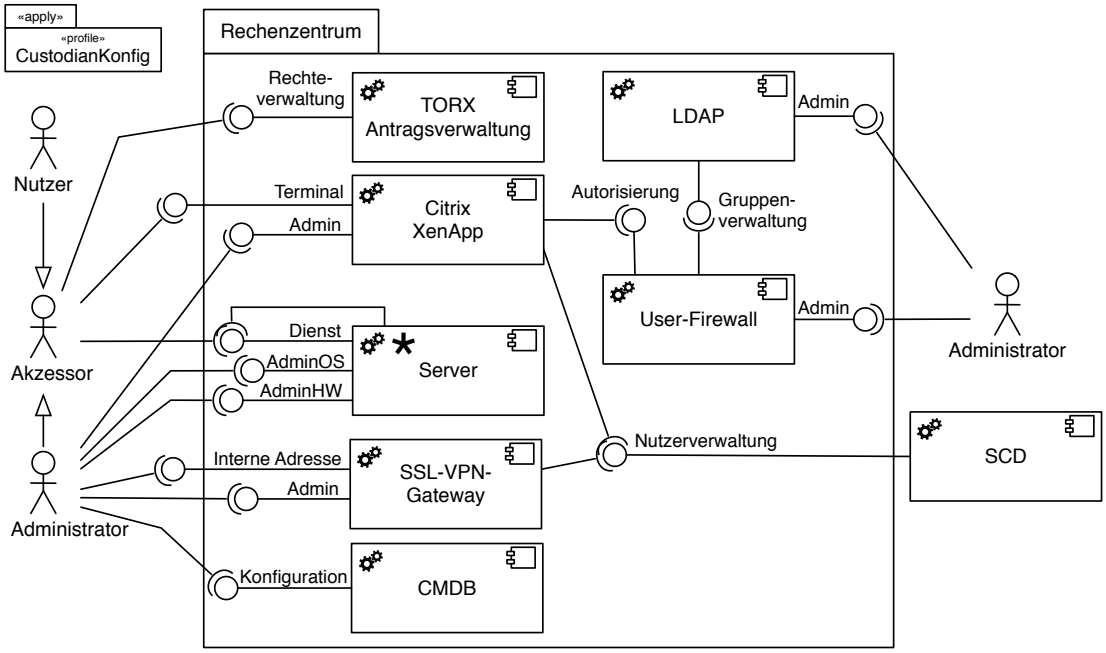


Abb. 8.2: *Sicherheitsdiagramm des HSRZ*

Administratoren sind für den Betrieb der Dienste verantwortlich. Um dieser Verantwortung nachkommen zu können, benötigen sie Berechtigungen, die über die des normalen Nutzer hinausgehen. Deshalb besteht für die-

se die Möglichkeit, durch eine VPN-Verbindung über die SSL-VPN-Gateway eine Interne IP-Addresse des Rechenzentrum zu erhalten. Abhängig von ihrem SSL-VPN-Profil können sie dann Netzwerkverbindungen zu Teilen oder dem gesamten Netzwerk des Rechenzentrums aufbauen. Die Administratoren müssen alle vorgenommenen strukturellen Änderungen in der CMDB pflegen. Alle von den Administratoren benötigten Rechte müssen ebenfalls über die Antragsverwaltung beantragt werden. In Abschnitt A.2 des Appendix sind die einzelnen Komponenten detailliert beschrieben.

8.1.2 Berechtigungsanalyse

Kunden dürfen mit Hilfe der Citrix-Komponente eine Terminalverbindung zum HSRZ herstellen. Mit einer hergestellten Terminalverbindung dürfen die Kunden die von Servern angebotenen Dienste ausführen, die ihnen zugewiesen sind und auf die Rechteverwaltung zugreifen. Administratoren können alternativ zur Terminalverbindung auch eine SSL-VPN-Verbindung mit dem HSRZ aufbauen. Administratoren dürfen wie auch Kunden auf die Rechteverwaltung und, soweit ihnen das Recht explizit zugeordnet ist, auf Dienste der Server zugreifen. So ist beispielsweise KUSTOS selbst als Dienst auf einem Server installiert und darf von einigen individuell berechtigten Administratoren verwendet werden.

Für die Ressourcen der weiteren Komponenten müssen den Administratoren ebenfalls individuell berechtigt werden. Auf Servern dürfen sie sowohl das Betriebssystem und die Hardware administrieren. Administratoren dürfen Citrix, die User-Firewall, das LDAP, das SSL-VPN und die CMDB administrieren. Die Citrix-Komponente darf sich an der User-Firewall mit dem Benutzerkontext des Akzessors autorisieren. Die User-Firewall darf die im LDAP gespeicherten Gruppeninformationen abfragen. Sowohl Citrix als auch das SSL-VPN dürfen die von ihnen benötigten Nutzerinformationen aus dem AD abfragen.

In Abschnitt A.2.1.1 des Appendix befindet sich die von HARVEST beschriebenen Erhebung der Konfigurationsdaten, Organisation der Datenbeschaffung und Planung der Ressourcen.

erfordert / bietet an	Server: Dienst	Server: Admin OS	Server: Admin Hardware	Citrix: Terminal	Citrix: Administration	User-FW: Autorisierung	User-FW: Administration	LDAP: Gruppenverwaltung	LDAP: Administration	TORX: Rechteverwaltung	SSL-VPN: Interne Adresse	SSL-VPN: Administration	CMDB: Konfiguration	AD: Nutzerverwaltung
Kunde	✓			✓						✓				
Administrator	✓	✓	✓	✓	✓		✓		✓	✓	✓	✓	✓	
Citrix						✓								✓
User-Firewall								✓						
SSL-VPN														✓

Tab. 8.1: *Zugriffsmatrix der Komponenten im HSRZ*

8.1.3 Gefahrenanalyse

Im Rahmen der Fallstudie traten sieben Gefahrenklassen auf. Diese lassen sich den drei Hauptkategorien Fehlautorisierung, Fehlauditierung und Komplexität zuordnen. Diese Zuordnung sowie die Diversifizierung in vier weitere Unterkategorien ist in der Taxonomie mit Mehrfachvererbung in Abbildung 8.3 modelliert. Eine Gefahr der Hauptkategorie Fehlautorisierung liegt vor, wenn ein Akzessor fälschlich Berechtigungen inne hat. Unter Fehlauditierung verstehen sich Gefahren, die zu einem negativen Resultat bei einer Auditierung führen können. Überkomplexität liegt vor, wenn die Konfiguration komplexer als notwendig ist.

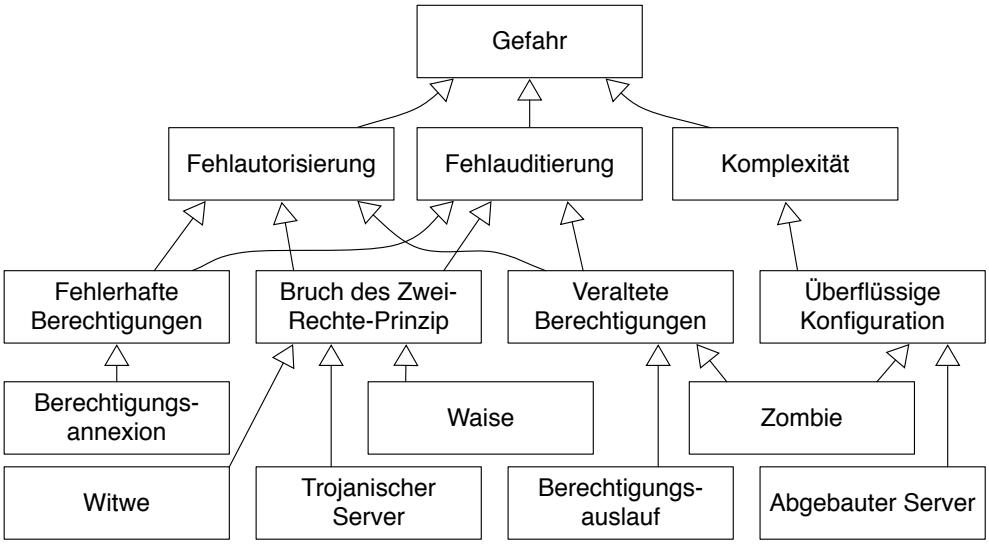


Abb. 8.3: *Taxonomie der Gefahrenklassen im HSRZ*

Mit Hilfe von zwei Methoden gelang es, sieben Gefahrenklassen zu identifizieren. Für einige Gefahrenklassen konnte schon vor Beginn der Fallstudie eine Deklaration erstellt werden. Die anderen Gefahrenklassen wurden erst durch die Arbeiten der Fallstudie aufgedeckt. Die Taxonomie in Abbildung 8.4 gibt einen Überblick über die Entdeckungsarten Gefahrenklassen. Im Folgenden werden die Gefahrenklassen beschrieben. Die für die Gesamtheit der Gefahrenklassen relevanten Ergebnisse sind im anschließenden Abschnitt A.2.1.1 zusammengefasst.

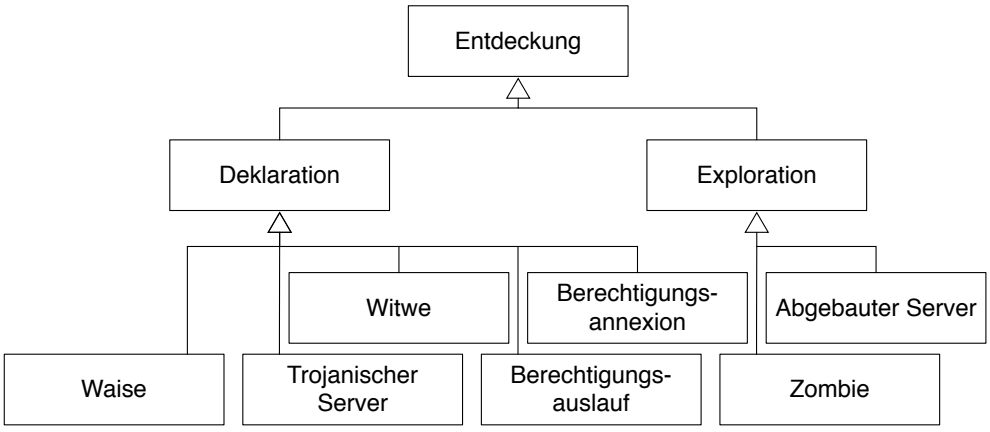


Abb. 8.4: *Taxonomie der Entdeckungsarten der Gefahrenklassen*

8.1.3.1 Berechtigungsauslauf

Anmeldeberechtigungen werden im HSRZ über sogenannte Anträge verwaltet, für die der in Abbildung 8.5 modellierte Berechtigungsprozess definiert ist. Dieser Prozess wird von der TORX-Antragsverwaltung (siehe Abschnitt A.2.1) initiiert.

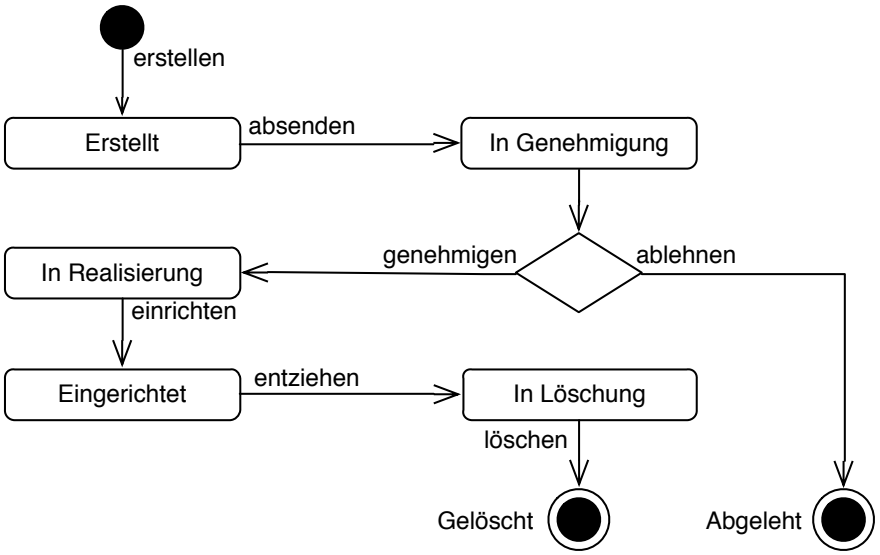


Abb. 8.5: *Prozess für Berechtigungen auf Dienste im HSRZ*

Eingerichtete Berechtigungen sind in diesem Prozess solange aktiv, bis sie vom Genehmiger entzogen werden. Eine lediglich schriftlich niedergelegte Richtlinie verpflichtet die Genehmiger, diese Regeln einmal im Geschäftsjahr zu überprüfen. Für jeden Server ist ein Genehmiger definiert. Diese Richtlinie bezieht sich auf Anmelderechte, die über die TORX-Antragsverwaltung beantragt wurden, Netzwerkrechte sind explizit ausgenommen. Die Überprüfung muss über die TORX-Antragsverwaltung erfolgen und wird in deren Datenbank festgehalten. Kurz vor dem Ablauf des Geschäftsjahrs werden die Genehmiger über eine Email an ihre Aufgabe erinnert. Ein Berechtigungsauslauf liegt vor, wenn eine Berechtigung auf einen aktiven Server über 365 Tage nicht überprüft wurde. Die initiale Genehmigung wird in diesem Kontext wie eine Überprüfung gewertet.

Ausschlaggebend für die Identifikation und Untersuchung der Gefahrenklassen waren der Sicherheitsverantwortliche des HSRZ und die der Verantwortliche der TORX-Antragsverwaltung. Diese vermuteten, dass die Verantwortlichen der Server ihren Verpflichtungen ungenügend nachkommen.

Da die TORX-Antragsverwaltung keine entsprechende Managementsicht bereitstellt und die Daten der Datenbank der Antragsverwaltung allein nicht hinreichend für die Auswertung waren, bot sich eine Untersuchung mit KUSTOS an.

Beispiele:
Einem Benutzer wurde vor 411 Tagen die Berechtigung zum Anmeldung an einem Server genehmigt. Diese Berechtigung wurde seitdem nicht überprüft.
Ein Benutzer hat eine Berechtigung, die vor 543 Tagen das letzte Mal genehmigt wurde.

Gefahrenbewertung:
Einst valide Rechte können mit der Zeit obsolet oder gar gefährlich werden. Je länger ein Recht nicht überprüft wird, desto höher ist die Wahrscheinlichkeit einer Gefahr. Dies kann zur Folge haben, dass eine nunmehr unberechtigte Person Zutritt zu einem Server hat und geschützte Informationen einsehen oder verändern kann. Da dann ein Richtlinienverstoß vorliegt, besteht zusätzlich die Gefahr, dass dem Betreiber des HSRZ notwendige Zertifizierungen nicht erteilt werden.

Der Sicherheitstest für Berechtigungsausläufe:
Betrachtet man die Gefahrenklasse Berechtigungsauslauf, so ist folgendes von Bedeutung:

- Eine Menge A der Akzessoren.
- Eine Menge S von Servern.
- Die Menge AS der Antragsstatus $\{$ 'Erstellt', 'In Genehmigung', 'In Realisierung', 'Eingerichtet', 'In Löschung', 'Gelöscht', 'Abgelehnt' $\}$.
- Die Menge SS der Serverstatus $\{$ 'Activated', 'Disassembled' $\}$.
- Eine Menge T bestehend aus den Tagen seit der Eröffnung des Rechenzentrums und $nDef =$ 'undefinierter Tag'.
- Eine Menge L von Anträgen (Logins). Ein Antrag l wird mit einer eindeutigen Nummer id identifiziert und ist ein Tripel (id, s, b) mit dem Server $s \in S$ und dem dem Besitzer $b \in A$.

Die Frage, ob die Berechtigung eines Antrags $a = (id, s, b)$ zum Zeitpunkt ta ausgelaufen ist, hängt von folgende Größen ab:

- dem Datum $gt \in T$ der Genehmigung von a,

- dem Datum $lt \in T$ der letzten Prüfung von a,
  ist noch nicht geprüft worden, so setzen wir $lt = nDef$,
- dem Antragsstatus $as \in AS$ von a und
- dem Serverstatus $ss \in SS$ von a

Diese Situation fassen wir zum *Zustand* $z = (a, ta, gt, lt, as, ss)$ zusammen. Mit Z sei die Menge aller Zustände bezeichnet. Es sei

$$v(t) = \begin{cases} 366 & \text{wenn } t = nDef, \\ \text{Anzahl der Tage von } t \text{ bis } ta & \text{sonst.} \end{cases}$$

Ein Zustand $z = (a, ta, gt, lt, as, ss)$ heißt *Berechtigungsauslauf*, falls gilt:

$$v(gt) > 365 \wedge v(lt) > 365 \wedge as = \text{'Eingerichtet'} \wedge ss = \text{'Activated'}$$

Die Menge aller Berechtigungsäuslaufe ist somit gegeben durch

$$\text{berechtigungsausläufe}(Z) =_{\text{def}} \{z \in Z \mid z \; ist \text{ Berechtigungsauslauf}\}.$$

Für die Ermittlung und Darstellung der Berechtigungsausläufe sind, wie in Abbildung 8.6 modelliert, drei Konfigurationsklassen erforderlich. Die Akzessoren, welche aus dem AD bezogen werden, treten in zwei Rollen auf: als Besitzer und als Genehmiger. Bei den Anträgen aus TROX sind nur die Logins für Windows und Unix relevant. Jeder dieser Anträge bezieht sich auf einen Server, dessen Konfiguration aus der CMDB bezogen wird.

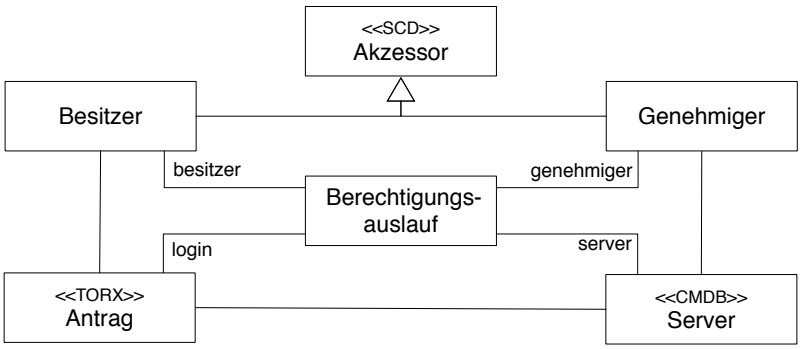


Abb. 8.6: *Modell eines Berechtigungsauslauf*

Die Menge der Berechtigungsausläufe kann über die in Quelltext 8.1 abgebildete, an Hibernate Query Language (HQL) angelehnte Abfrage ermittelt werden. In Zeile 3 dieser Abfrage wird auf die Anträge gefiltert, die den Status „Eingerichtet“ haben. Denn nur eingerichtete Anträge repräsentie-

ren ein aktives Login und somit eine potentielle Gefahr. In den Zeilen 4 und 5 wird validiert, ob die letzte Prüfung bzw. die Genehmigung älter als 365 Tage ist. In Zeile 6 wird geprüft, ob der Server aktiv ist, denn von inaktiven (abgebauten) Servern geht keine Gefahr im Sinne dieser Gefahrenklasse aus. Eingerichtete Rechte auf inaktiven Servern stellen eine Gefahr der Gefahrenklasse „Abgebauter Server“ dar (siehe Abschnitt 8.1.3.4).

```
1 SELECT ln.besitzer, ln, ln.server, ln.server.genehmiger
2 FROM login ln
3 WHERE ln.status = 'Eingerichtet'
4   AND (current_date - ln.genehmigung) > 365
5   AND (current_date - ln.letzte_pruefung) > 365
6   AND ln.server.status = 'Activated'
```

Quelltext 8.1: *Abfrage für die Ermittlung der Berechtigungsausläufe*

Abbildung 8.7 zeigt, wie die Testmenge für Berechtigungsausläufe mit Hilfe des KUSTOS-Wissensbinders (siehe Abschnitt 6.4.3) zusammengestellt werden kann. Für diese Gefahrenklassen müssen alle Anträge aus der Antragsverwaltung überprüft werden. Hierfür wird die Konfigurationsklasse Anträge auf die Anträge gefiltert, die den Status „Eingerichtet“ haben. Die Konfigurationsklasse User liefert weitere Informationen über den Besitzer des Antrags. Weitere Informationen über den Server des Antrags und dessen Verantwortlichen (der Genehmiger der Anträge für diesen Server) werden mit Hilfe der Konfigurationsklassen Server und User ermittelt.

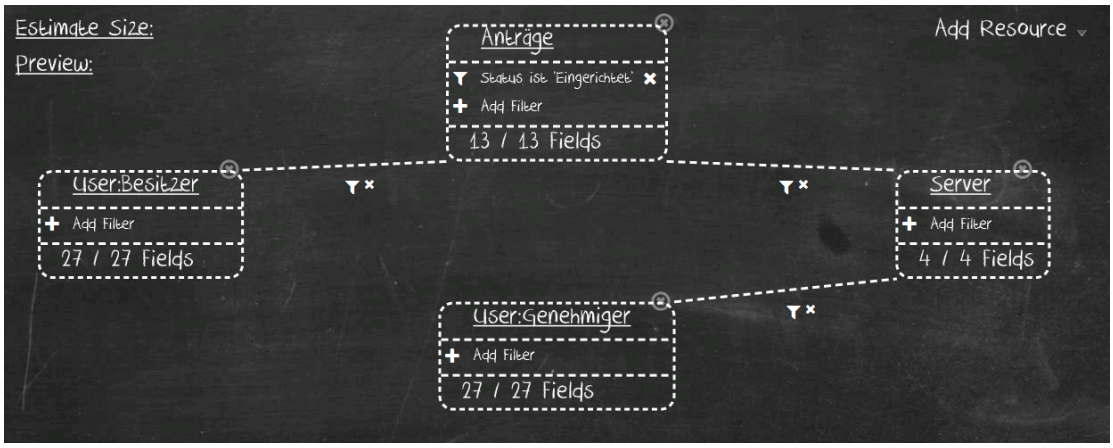


Abb. 8.7: *Definition der Testmenge für Berechtigungsausläufe via Wissensbinder*

Auswertung Bei der ersten Auswertung der Gefahrenklasse Berechtigungsauslauf am 11. Mai 2014 untersuchte KUSTOS 40.982 Testobjekte und identifizierte dabei 19.407 Gefahren. Die Vermutung, dass einige der Verantwortlichen ihren Verpflichtungen nur ungenügend nachkommen, bestätigte sich mit gut 47,4 Prozent ausgelaufenen Berechtigungen sehr eindeutig. Im Folgenden sind fünf Momentaufnahmen der Auswertung der Berechtigungsausläufe als Motion-Chart (siehe Abschnitt 6.4.1) visualisiert. Die verantwortlichen Genehmiger wurden nach ihren Abteilungen gruppiert[2]. Jeder in Motion-Charts dargestellte Kreis repräsentiert eine Abteilung. Die Kreise sind auf der Y-Achse logarithmisch angeordnet. Während die Abteilung, die in Chart 8.8 am oberen Ende der Y-Achse visualisiert ist, 10.050 abgelaufene Berechtigungen zu verantworten hat, sind es bei Abteilung auf Rang zwei „nur“ 1.343 abgelaufene Berechtigungen.

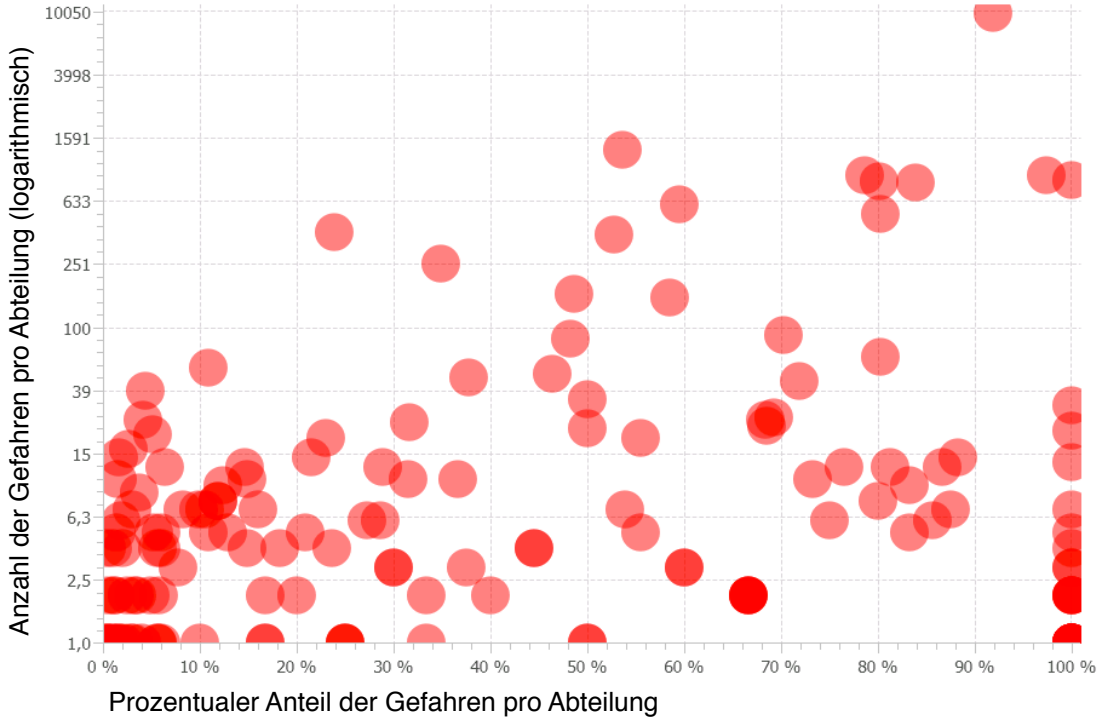


Abb. 8.8: *Motion-Chart der Berechtigungsausläufe am 11. Mai 14*

Diese initiale Auswertung fand eine große Aufmerksamkeit im Management. Durch sie wurde verdeutlicht, dass der aktuelle Auditprozess für Berechti-

[2]Aus Datenschutzgründen durften keine Erhebungen mit personalisierten Daten vorgenommen werden.

gungen nicht eingehalten wird. Einzelne Berechtigungen waren seit über drei Jahren abgelaufen. Aufgrund des hohen Anteils der abgelaufenen Berechtigungen wurde von einem systematisches Problem des Auditprozesses und nicht nur von einem fehlerhaften Verhalten der verantwortlichen Genehmiger ausgegangen. Es wurden zwei Ursachen für das systematische Problem identifiziert: Erstens war es für den Genehmiger nicht einfach, die von ihm zu prüfenden Berechtigungen herauszufinden. Zweitens wurde der Genehmiger während der Anmeldung bei der Antragsverwaltung nicht auf abgelaufene Berechtigung hingewiesen.

Zur Beseitigung der Gefahren wurden zwei Strategien verfolgt. Eine kurzfristige, um die abgelaufenen Berechtigungen schnell zu beseitigen und eine langfristige, um das systematische Problem zu beheben. In Tabelle 8.2 sind alle Analyseergebnisse zusammengefasst. Da fortlaufend Anträge gestellt werden, basieren die Analysen auf unterschiedliche Testmengen. Die ersten Analyse war noch durch keine Strategie beeinflusst, die drei darauf folgenden waren durch die kurzfristige Strategie beeinflusst, und die letzte Analyse war durch beide Strategien beeinflusst.

Datum	Testobjekte	Gefahren	Anteil	Abbildung	Strategie
11. Mai 14	40982	19407	47,35 %	Abb. 8.8	
16. Mai 14	41207	6106	14,82 %	Abb. 8.9	kurz
03. Juni 14	41543	3502	8,43 %	Abb. 8.10	kurz
13. Juni 14	43273	1722	3,98 %	Abb. 8.11	kurz
16. Mai 15	47724	176	0,36 %	Abb. 8.13	kurz, lang

Tab. 8.2: *Zusammenfassung aller Analysen der Berechtigungsausläufe*

Die kurzfristige Strategie besteht darin, die Abteilungen über ihren Prüfungsrückstand in Kenntnis zu setzten. Bereits 5 Tage nach der ersten Analyse waren die Anzahl der Gefahren auf 6106 gesunken (siehe Abbildung 8.9). Das entspricht 14,8 Prozent der neuen 41207 Testobjekte. Mehr als zwei Drittel der Gefahren wurde in dieser Zeit beseitigt.

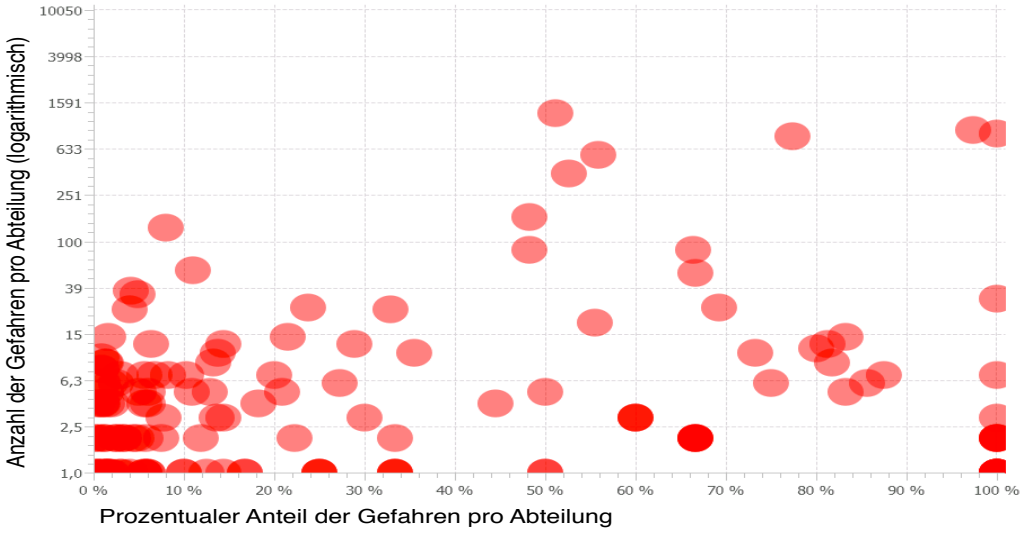


Abb. 8.9: *Motion-Chart der Berechtigungsausläufe am 16. Mai 14*

Nach 23 Tage waren die Anzahl der Gefahren bereits auf 3502 gesunken (siehe Abbildung 8.10). Das entspricht 8,4 Prozent der neuen 41543 Testobjekte.

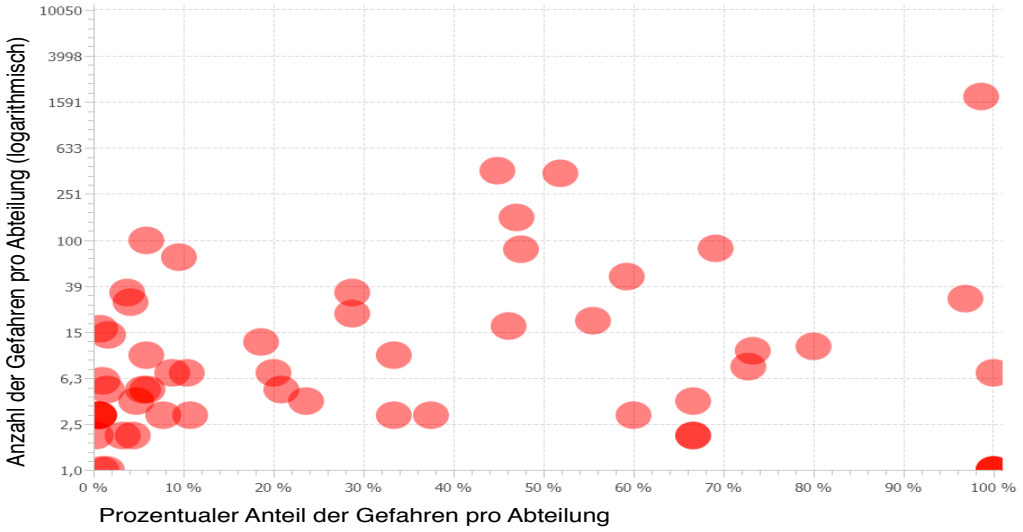


Abb. 8.10: *Motion-Chart der Berechtigungsausläufe am 3. Juni 14*

Abbildung 8.11 zeigt die 1722 die nach weiteren 10 Tagen noch bestehenden Gefahren. Das entspricht 3,98 Prozent der neuen 43273 Testobjekte.

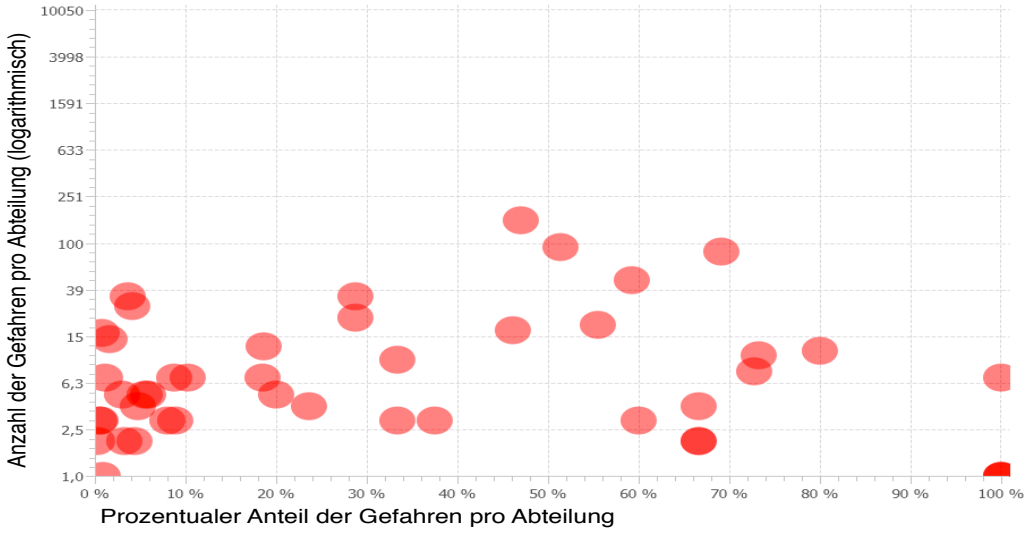


Abb. 8.11: *Motion-Chart der Berechtigungsausläufe am 13. Juni 14*

In der folgenden Zeit wurde die langfristige Strategie umgesetzt. Hierfür wurde die Startseite der Antragsverwaltung um eine aktuelle Übersicht der abgelaufenen und der bald ablaufenden Berechtigungen erweitert (siehe Abbildung 8.12).

Abb. 8.12: *Neue Startseite der TORX-Antragsverwaltung*

Nach der Erweiterung der Startseite der Antragsverwaltung wurde das Thema der auslaufenden Berechtigungen für die verantwortlichen Genehmiger

viel sichtbarer. Nach dieser Erweiterung sank die Anzahl auf 176, also auf unter ein Prozent der ursprünglichen Gefahren[3].

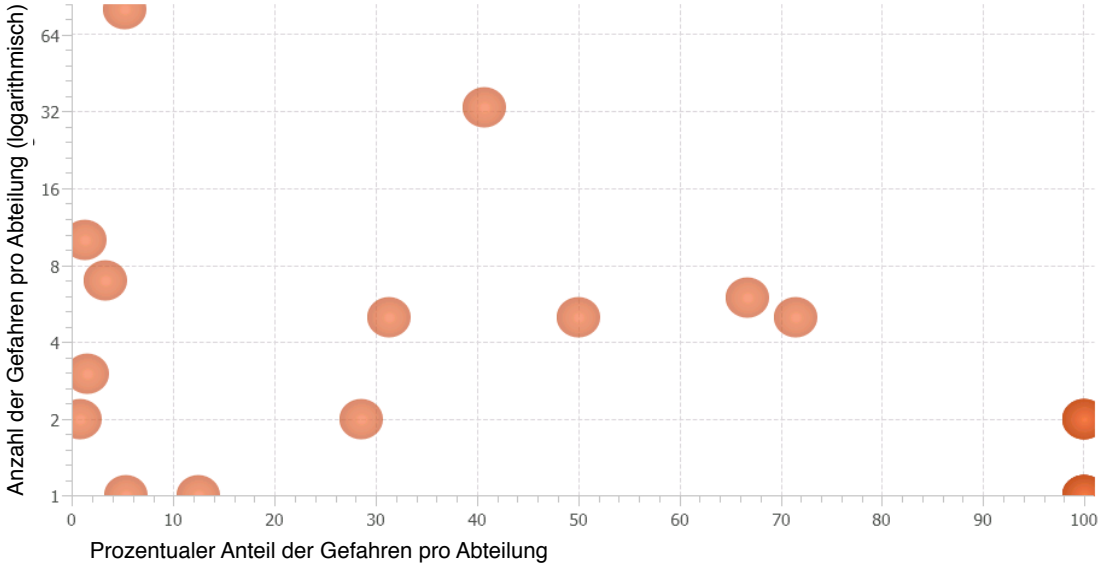


Abb. 8.13: *Motion-Chart der Berechtigungsausläufe am 23. November 15*

[3]Das unterschiedliche Design in Abbildung 8.13 ist der Weiterentwicklung von KUSTOS geschuldet.

8.1.3.2 Berechtigungsannexion

Die Gefahrenklasse Berechtigungsannexion modelliert die besonderen Anforderungen die für VS-NfD-Server gelten. Auf diesen Anforderungen basiert bereits die Gefahrenklasse Trojaner (siehe Abschnitt 8.1.3.3). Eine allgemeine Vorschrift lautet, dass nur betriebszugehörige Personen mit einen deutschen Arbeitsvertrag auf solche Server zugreifen dürfen. Somit werden durch diese Gefahrenklasse die Anmeldeberechtigungen (Logins) für VS-Server erkannt, bei denen der Berechtige diese Kriterien nicht erfüllt.

Beispiele: Die externe Person A mit inländischem Vertrag mit einem Login [1] auf einem VS-Server entspricht einer Berechtigungsannexion. Die interne Person B mit inländischem Vertrag mit einem Login [2] auf einem VS-Server entspricht keiner Berechtigungsannexion. Die externe Person C mit ausländischem Vertrag mit einem Login [3] auf einem VS-Server entspricht einer Berechtigungsannexion. Die interne Person D mit ausländischem Vertrag mit einem Login [4] auf einem VS-Server entspricht einer Berechtigungsannexion.

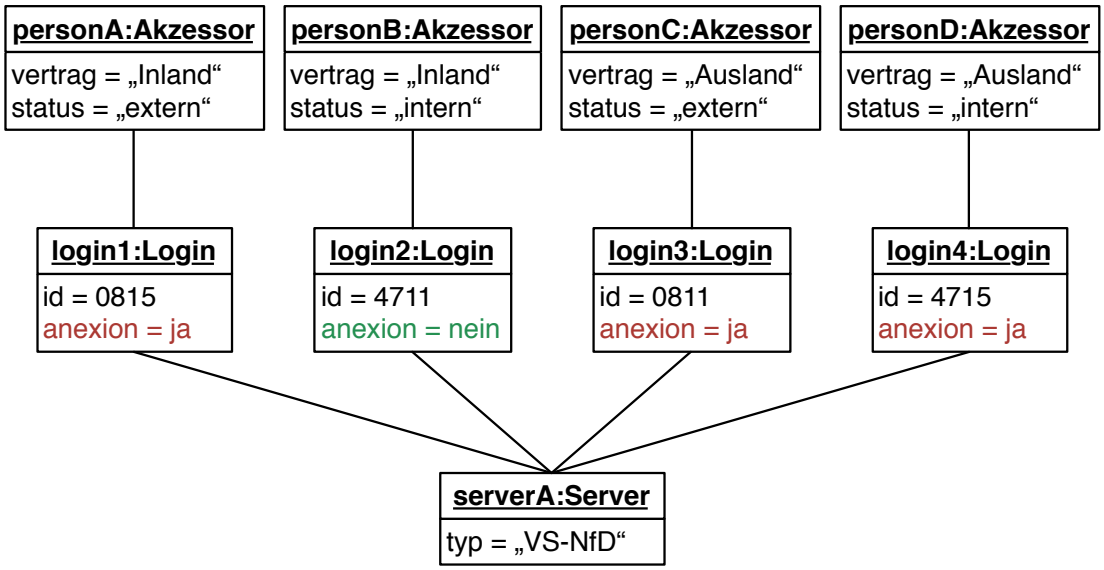


Abb. 8.14: *Reguläre und annektierte Berechtigungen als Objektdiagramm*

Gefahrenbewertung: Eine Konstellation, bei der eine unberechtigte Person Zugriff auf VS-Daten hat, sei laut Betreiber vom HSRZ sehr kritisch. Einerseits besteht die Gefahr, dass die Daten offengelegt oder kompromittiert werden. Andererseits kann dies zu einem negativen Befund bei einer VS-NfD-Auditierung führen.

Der Sicherheitstest für Berechtigungsannexionen: Aus Sicht der Gefahrenklasse Berechtigungsannexion existieren in einem Rechenzentrum folgende Grundobjekte:

- Die Menge V { 'Inland', 'Ausland' } der Vertragsarten.
- Die Menge F { 'intern', 'extern' } der Firmenzugehörigkeiten.
- Eine Menge A von Akzessoren. Ein Akzessor $a \in A$ ist ein Paar (v, f) mit der Vertragsart $v \in V$ mit der Firmenzugehörigkeit $f \in F$.
- Die Menge T { 'VS-NfD', 'Standard' } der Servertypen.
- Eine Menge S von Servern. Ein Server $s \in S$ besitzt den Servertyp $t(s) \in T$.
- Eine Menge L von Logins. Ein Login $l \in L$ ist ein Paar (a, s) mit dem Akzessor $a \in A$ und dem Server $s \in S$. Nach Auflösung von a stellt sich das Paar als $((v, f), s)$ dar.

Eine Login gilt als Berechtigungsannexion falls

$$\text{annexion}((v, f), s) =_{\text{def}} ((v = \text{'Ausland'} \vee f = \text{'extern'}) \wedge t(s) = \text{'VS-NfD'}).$$

Die Menge aller Berechtigungsannexion kann ermittelt werden durch

$$\text{annexion}(L) =_{\text{def}} \{((v, f), s) \in L \mid \text{annexion}((v, f), s)\}.$$

Testmenge: In Abbildung 8.15 ist die Konfiguration des Wissensbinders zur Erstellung der Testmenge dargestellt. Für die Erzeugung der Testmenge sind drei Wissensklassen notwendig. Ein Antrag verkörpert einen Login, ein User einen Akzessor und ein Server einen Server im Sinne des Sicherheitstests. Prinzipiell sind nur die Anträge interessant, die sich auf einen VS-Server beziehen. Deshalb wird für die Wissensklasse Server der Filter „IS VS“ erstellt. Dieser Filter reduziert die Testmenge von 6651 Testobjekten auf 614 Testobjekte.

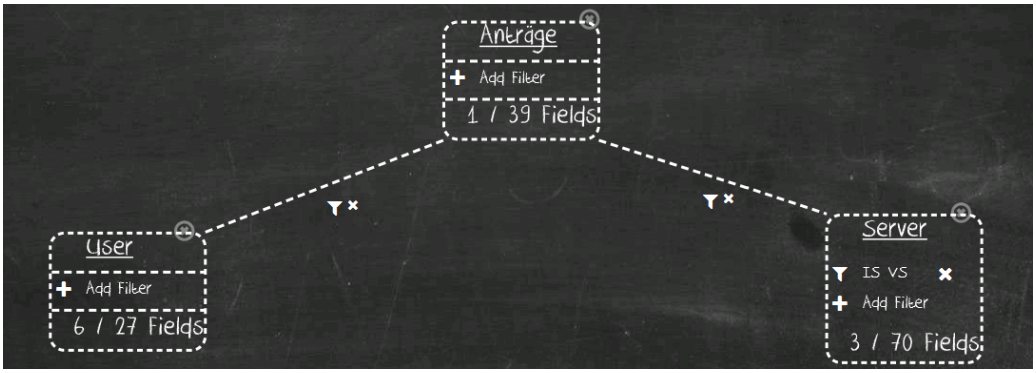


Abb. 8.15: *Testmengendefinition für Berechtigungsannexion auf dem Wissensbinder*

Auswertung: Die erste Auswertung des Sicherheitstests führte für den Zeitraum von von Anfang September bis zum 1. November stets 19 bis 20 Hypothesen für Berechtigungsannexionen. Nach der Überprüfung aller Hypothesen sollte sich herausstellten, dass es sich bei 18 der 20 Hypothesen um tatsächliche Gefahren handelte. Die zugrundeliegenden Berechtigungen dieser 18 Berechtigungsannexionen wurden umgehend entzogen und die Anzahl der Hypothesen sank, wie in Abbildung 8.16 dargestellt, auf zwei.

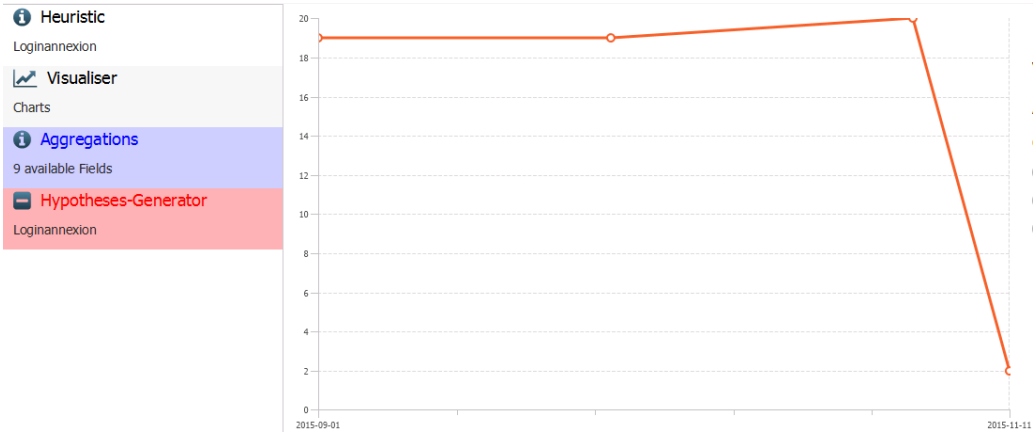


Abb. 8.16: *Entwicklung der Berechtigungsannexionen vor und nach der Analyse*

Bei einem Teil der Berechtigungsannexionen handelte es sich um Portugiesische Mitarbeiter des Rechenzentrums. Diese dürfen aufgrund ihres nicht auf deutschem Recht basierenden Arbeitsvertrags nicht auf VS-Server zugreifen. Der andere Teil der Berechtigungsannexionen ging von Berechtigungen für Werkstudenten und Praktikanten aus. Um diese Art von Berechtigungsannexionen generell auszuschließen, wurde das Antragsformular für Berechtigungen erweitert.

Die verbleibenden beiden Hypothesen bezogen sich auf Berechtigungen für Personen, die einer Tochterunternehmen beschäftigt sind. Da diese Tochterunternehmen ebenfalls VS-NfD-zertifiziert sind geht von diesen beiden Berechtigungen keine Gefahr aus. Ein neu eingeführter Invalidator verhindert in Zukunft derartige Hypothesen automatisch.

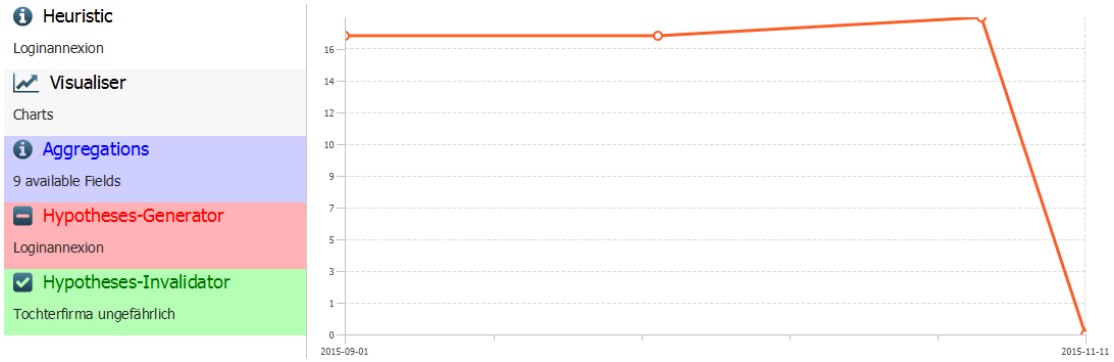


Abb. 8.17: *Entwicklung der Berechtigungsannexionen unter Berücksichtigung des Invalidators*

Durch die Einführung des Invalidator reduziert sich die Anzahl der nicht invalidierten Hypothesen über den gesamten Zeitraum um exakt die beiden invalidierten Berechtigungen. Das ist in Abbildung 8.17 erkennbar.

8.1.3.3 Trojanische Server

Diese Gefahrenklasse basiert nicht auf der ebenfalls als Trojaner bzw. Trojanisches Pferd bezeichneten Schadsoftware. Vielmehr beruht sie auf der Möglichkeit, dass man von einem Server, zu dem man eine Verbindung aufgebaut hat, Verbindungen zu anderen Servern aufbauen kann. Diese Technik heißt in dieser Dissertation Serverhopping. Im HSRZ ist Serverhopping möglich, denn das HSRZ-Netzwerk ist in Subnetze aufgeteilt, innerhalb derer Server beliebige Netzwerkverbindungen zueinander aufbauen können.

Das Prinzip des Serverhoppings ist in Abbildung 8.18 dargestellt. Der Akzessor darf aufgrund der Firewallregeln zu Server A, jedoch nicht zu Server B eine Netzwerkverbindung aufbauen[4]. Meldet sich der Akzessor jedoch am Server A an, kann er von dort aus eine Netzwerkverbindung zu Server B aufbauen. Das Serverhopping wird somit durch die Zugehörigkeit der beiden Server zum selben Subnetz ermöglicht.

Serverhopping ist nicht in jedem Fall gefährlich. Im Normalfall stehen nur jene Server im gleichen Subnetz, welche auch zu der gleichen Anwendung gehören, von den selben Personen administriert und den gleichen Nutzer verwendet werden dürfen. Innerhalb eines Subnetzes werden keine Firewallregeln erstellt. Diese Entscheidung sei dem sonst zu hohen Aufwand der Pflege geschuldet. Im Fall von VS-NfD-klassifizierten-Servern kann Serverhopping jedoch gefährlich sein. Diese besteht, wenn von einem nicht-VS-NfD-Server zu einem VS-NfD-Server eine Verbindung aufgebaut werden kann. Zur Vereinfachung werden VS-NfD-Server ab sofort als VS-Server und nicht-VS-NfD-Server als Standardserver bezeichnet.

Beispiele: Das in Abbildung 8.18 dargestellte Beispiel zeigt zwei Server in einem Subnetz. Der VS-Server Dokumentenverwaltung hat vier IP-Adressen, zu denen die IP 10.10.10.25 gehört. Der Standardserver Personalverwaltung hat ebenfalls vier IP-Adressen, zu denen die IP 10.10.10.27 gehört. Die beiden IP-Adressen 10.10.10.25 und 10.10.10.27 gehören dem Subnetz 10.10.10.24/29 an. Deshalb können zwischen den beiden Servern der Dokumentenverwaltung und der Personalverwaltung beliebige Netzwerkverbindungen aufgebaut werden.

[4] Im HSRZ-Kontext entsprechen die Firewallregeln den Aggregat der Regeln des SSL-VPNs und des Citrix-Gateways.

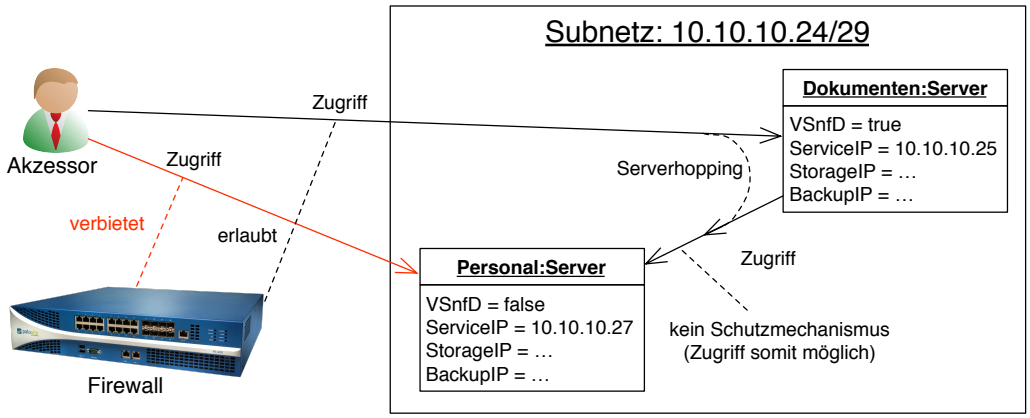


Abb. 8.18: *Beispiel für den Aufbau einer Verbindung zu einem eigentlich nicht erreichbaren Server mittels Serverhopping*

Gefahrenbewertung: Nicht VS-NfD-berechtigte Personen können mittels Serverhopping eine indirekte Netzwerkverbindung zu einem VS-Server aufbauen. Somit liegt ein Bruch des Zwei-Rechte-Prinzips vor. Daraus folgt, dass unberechtigte Personen leichter VS-NfD-Daten einsehen oder verändern können.

Der Trojanersicherheitstest: Betrachtet man die Gefahrenklasse Trojaner, so sind folgende Objekte von Bedeutung:

- Eine Menge I der IP-Adressen. Eine IP-Adressen $i \in I$ ist ein Paar (a, n), wobei a eine Adresse und n ein Subnetz ist
  Beispiel: $i = (\text{'10.10.10.25'}, \text{'10.10.10.24/29'})$.
- Eine Menge V { 'VS-Server', 'Standardserver' } der VS-nfD-Status.
- Eine Menge S von Servern. Ein Server ist ein Tripel $s = (id, v, I')$ wobei id ein eindeutiges Wort zur Benennung eines Servers, $v \in V$ und $I' \subseteq I$ ist.
- Eine Menge SN von allen Servernnamen.

Eine Netzzugehörigkeit nz ist ein Tripel $nz = (id, v, n)$, wobei id eine Sevename, $v \in V$ und n ein Subnetz ist. Die Menge der Netzzugehörigkeiten eines Servers s ergibt sich aus dem Kreuzprodukt

$$\text{serveradressen}(s) =_{\text{def}} \{(id, v, n) | (a, n) \in I'\}.$$

Die Menge der Netzzugehörigkeiten NZ aller Server von S ist

$$NZ =_{\text{def}} \bigcup_{s \in S} \text{netzzugehörig}(s).$$

Die Mengen der VS-Netzzugehörigkeiten NZ_v und der Standardnetzzugehörigkeiten NZ_n ergeben sich aus

$$NZ_v =_{\text{def}} \{(id, v, n) \in NZ \,|\, v = 'VS - Server'\} \text{ und } NZ_n =_{\text{def}} NZ \backslash NZ_v.$$

Eine VS-Netzzugehörigkeit $nz = (id, v, n)$ hat Trojaner falls

$$\text{trojaner}(id, v, n) =_{\text{def}} \{SN \times v \times \{n\} \cap NZ_n \neq \emptyset\}.$$

Die Menge aller VS-Netzzugehörigkeiten mit Trojaner ergibt sich aus

$$\text{trojaner}(NZ_v) =_{\text{def}} \{(id, v, n) \in NZ_v \,|\, trojaner(id, v, n)\}.$$

Testmenge: Der Wissensbinder ermöglichte das erstellen der Testmenge aus aus drei Wissensklassen (siehe Abbildung 8.19). Die Wissensklasse Server liefert neben dem Servernamen noch weitere, für die Auswertung interessante Informationen. Da diese Gefahrenklasse eine Aussage über gemeinsam in einem Subnetz stehende Serverpaare trifft, ist die Wissensklasse Server zweimal eingebunden. Über die Wissensklasse IP-Adressen kann ermittelt werden, über welche IP-Adressen ein Server erreichbar ist. Somit muss auch diese Wissensklasse zweimal eingebunden werden. Über die Wissensklasse Subnetze werden letztlich beide Server-IP-Adressen-Paare verbunden.

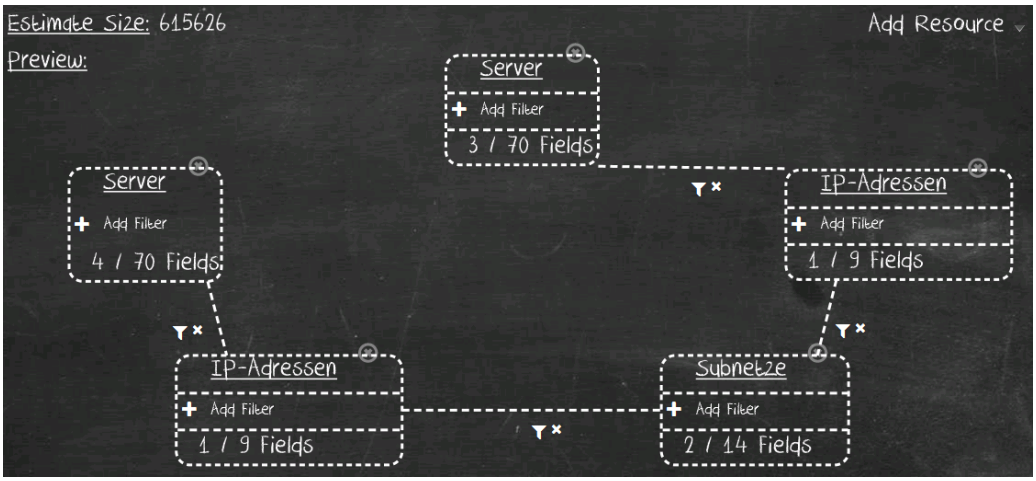


Abb. 8.19: *Definition der Testmenge für Trojaner mit dem Wissensbinder*

Die mit dem Wissensbinder erstellte Testmenge enthielt 615626 Testobjekte der Art $(Server, IP, Subnetz, IP, Server)$. Es existierten insgesamt 307813

Kombinationen, das entspricht der Hälfte. Denn eine Kombination zweier Servern $server'$ und $server''$ im gleichen Subnetz ist mit den Quintupeln $(server', IP, Subnetz, IP, server'')$ und $(server'', IP, Subnetz, IP, server')$ in der Testmenge vertreten. Für die Generierung Testmenge gab es zwei Filter (siehe Abbildung 8.20). Der erste Filter ($Is\ VS$) reduziert die erste eingebunden Menge von Servern auf die Menge der VS-Server. Der zweite Filter ($No\ VS$) reduziert die zweite eingebunden Menge von Servern auf die Menge der Standardserver.

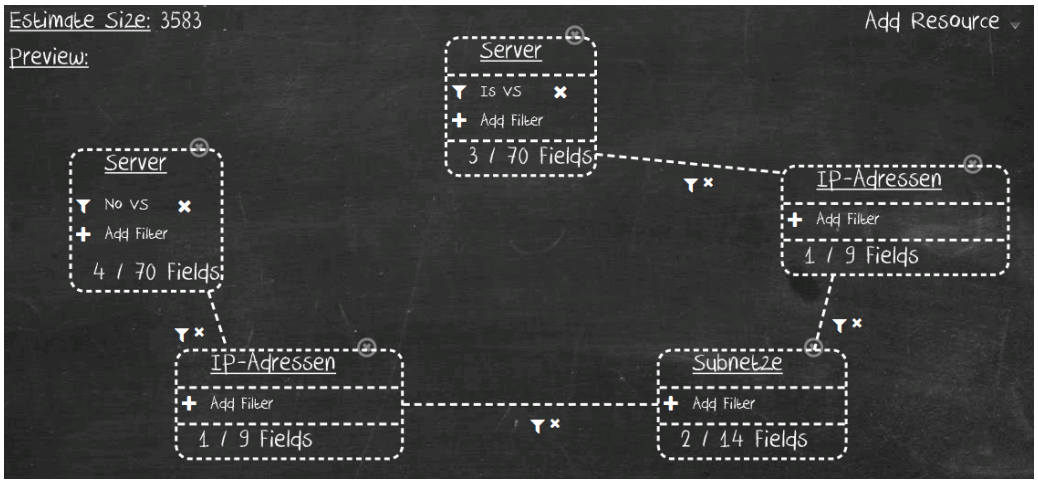


Abb. 8.20: *Definition der Testmenge für Trojaner mit dem Wissensbinder*

Mit Hilfe der Filter werden doppelte auftretende Serverkombinationen eliminiert und eine eindeutige Zuordnung von VS-Server und Standardserver gewährleistet. Ein Testobjekt entspricht somit einem Quintupel $(VS\text{-}Server, IP, Subnetz, IP, Standardserver)$. Die sich dann ergebende Testmenge hatte eine Größe von 3583 Testobjekten.

Auswertung: Für jedes der Testobjekte wird von dem Hypothesengenerator eine Gefahrenhypothese generiert. Somit entspricht die Menge B in der Tabelle 8.3 sowohl der verwendeten Testmenge als auch der Menge der generierten Hypothesen. Die Tabelle 8.3 gibt eine Übersicht über die beschriebenen Testmengen sowie über die folgenden Invalidationsmengen und die verbleibenden Menge der nicht invalidierten Hypothesen.

Im Liniendiagramm der Abbildung 8.21 ist ersichtlich, wie sich die Anzahl der Trojaner über den Zeitraum von gut 8 Monaten vor der Analyse entwickelt haben. Das abgebildete Diagramm ist von den Ergebnissen der Unter-

$\{\ \}$	Beschreibung	Größe	Anteil
A	Gesamte Testmenge	615626	–
B	Eingegrenzte Testmenge (mit Filter $Is\ VS$ und $No\ VS$)	3583	0,58 % von A
C	Invalidator „Test“	294	8,21 % von B
D	Invalidator „Integration“	180	5,02 % von B
E	Invalidator „Backup“	148	4,13 % von B
F	Invalidator „Storage“	128	3,57 % von B
G	$B \setminus (C \cup D \cup E \cup F)$	2917	81,41 % von B
H	Invalidator „iRMC“	3190	89,03 % von B
I	$B \setminus (C \cup D \cup E \cup F \cup I)$	105	2,93 % von G

Tab. 8.3: *Übersicht über Test-, Hypothesen-, und Invalidationsmengen*

suchung unbeeinflusst und gibt einen Einblick in die Volatilität der Konfiguration des verteilten Systems.

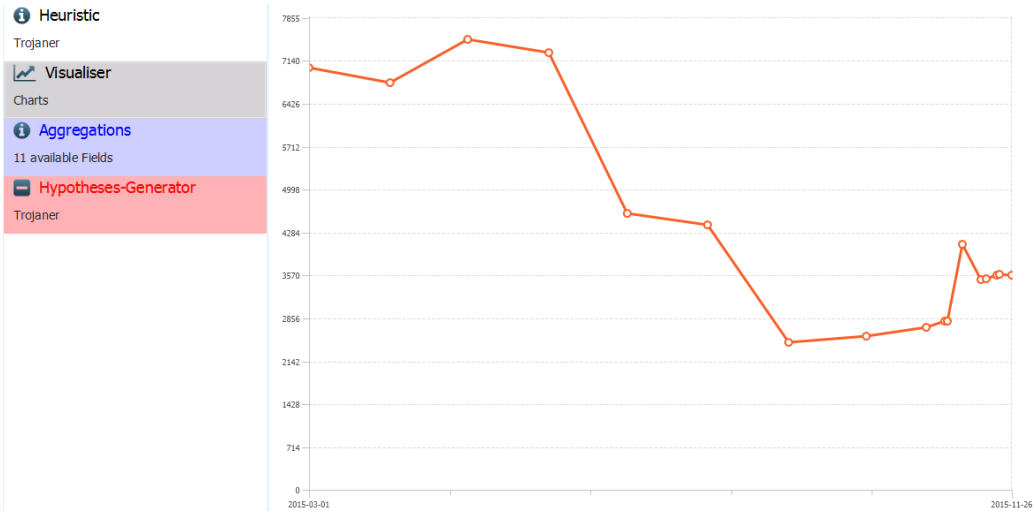


Abb. 8.21: *Entwicklung der Trojaner-Hypothesen*

Die Testmenge B beinhaltete eine Menge von ungefährlichen Trojanern. Einige betroffenen VS-Systeme waren Test- oder Integrationssysteme, auf denen sich keine sensiblen Daten befanden. Für die Test- und Integrationssysteme wurden zwei Invalidatoren zu der Heuristik hinzugefügt.

Wie in Modell 8.18 aufgezeigt, besitzt ein Server mehr als eine IP-Adresse. Für jeden Server gibt es die Service-IP-Adresse, über die Nutzer und Admi-

nistratoren zugreifen können. Zusätzlich können noch die Storage-IP-Adresse, über die externer Speicher eingebunden wird, und die Backup-IP-Adresse, über die Datensicherungen vorgenommen werden, bestehen. Von diesen beiden möglichen IP-Adressen geht laut Einschätzung der Spezialisten keine Gefahr aus. Deshalb wurden weitere zwei Invalidatoren für diese beiden Netze zu der Heuristik hinzugefügt.

Insgesamt wurden somit vier Invalidatoren für ungefährliche Trojaner erstellt. Die Anzahl der invalidierten Hypothesen sind in Tabelle 8.3 unter den Mengen C, D, E und F angegeben. Die vier Invalidatoren sind in Abbildung 8.22 dargestellt. Auch nach diesen Invalidierungen verblieben noch gut 89 Prozent der Gefahren.

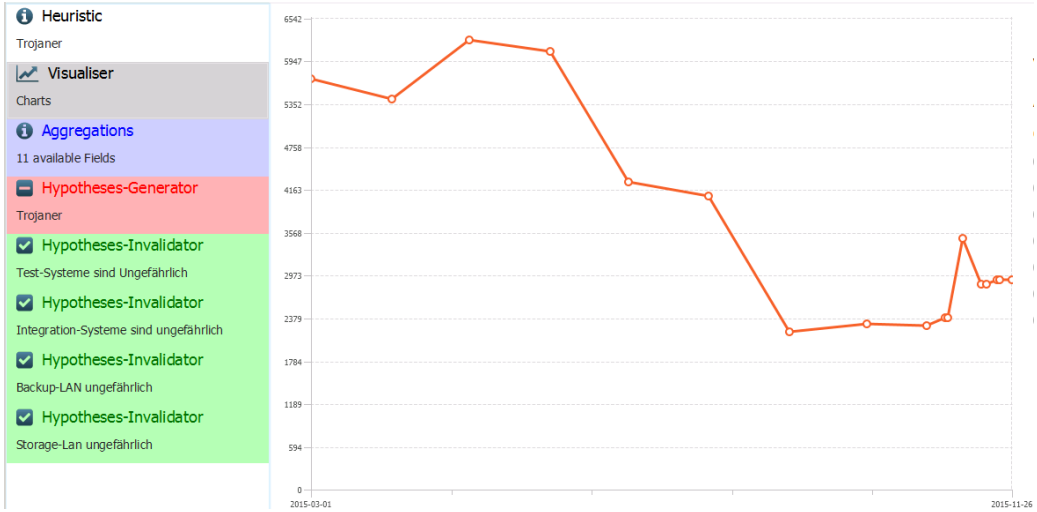


Abb. 8.22: *Entwicklung der Trojaner-Hypothesen nach Invalidierung der ungefährlichen Testobjekte*

Bei der Analyse der verbleibenden Gefahren fiel die Existenz von bisher unbeachteten IP-Adressen auf. Neben den Service-, Backup und Storage-IP-Adressen haben Server noch eine weitere IP-Adresse. Diese IP-Adresse ist durch die Administrationsschnittstelle[5] der Serverhardware begründet. Die iRMC-IP-Adresse wird der Hardware des Servers, und nicht dem auf dem Server installierte Betriebssystem, zugeordnet. Dieser Umstand ist in Abbildung 8.23 modelliert.

[5]integrated Remote Management Controller (iRMC)

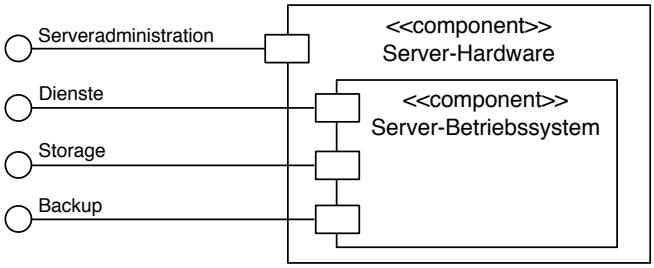


Abb. 8.23: *Das ätriebssystem als Subkomponente einer Hardware*

Mit dem iRMC können Administratoren direkt auf die Hardware des Servers zugreifen. Von der Hardware kann man wiederum auf das Betriebssystem selbst zugreifen. Bei der VS-Sicherheitsbewertung fiel die Tatsache, dass die iRMC-IP-Adressen von VS-Servern und Standardservern in den gleichen Netzwerken liegen nicht auf. Die Sicherheitsverantwortlichen führten auf Basis dieser Erkenntnis eine neue Richtlinie ein und setzten diese um: Nur die Personen, die VS-berechtigt sind, dürfen einen Zugang zu der Serveradministration erhalten. Das führte zur Invalidierung der Menge der Trojaner aus iRMC-Netzen (siehe Abbildung 8.24) .

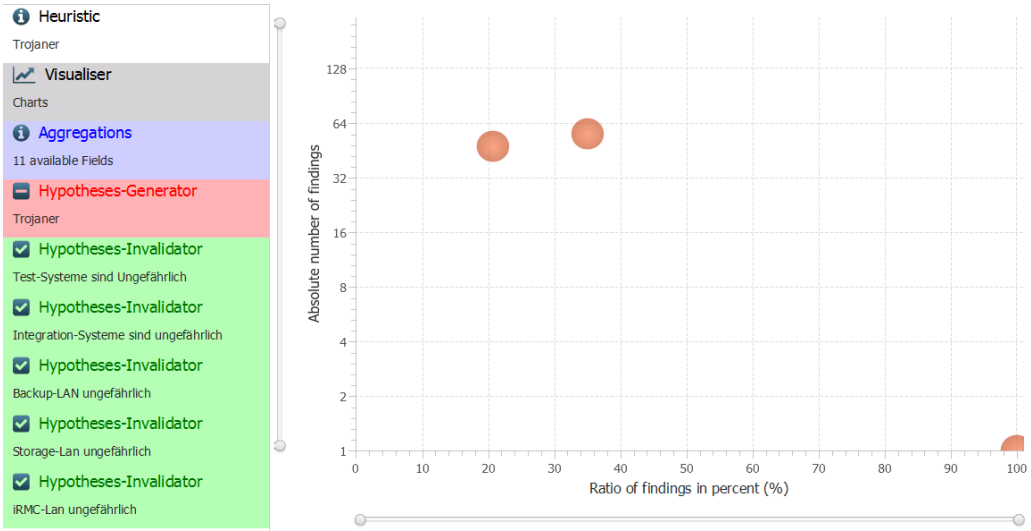


Abb. 8.24: *Darstellung der verbleibenden gefährlichen Trojaner als Motion-Chart*

Die verbleibenden Trojanern sind in dem Motion-Charts nach Systemlinien aggregiert. Jeder angezeigte Kreis repräsentiert eine Systemlinie. Letztendlich blieben nur 105 gefährliche Trojaner übrig welche auf drei Systemlinien verteilt waren. Eine Systemlinie, welche eine Zusammenfassung von Servern darstellt, ist detailliert im Abschnitt A.7 des Appendix beschrieben.

8.1.3.4 Abgebauter Server

Die Berechtigung, sich an einem Server anzumelden, wird über die TORX-Antragsverwaltung (siehe Abschnitt A.2.1) beantragt. Zunächst realisieren Administratoren die in der Antragsverwaltung beantragten Rechte auf den Servern (siehe Abbildung 8.5). Die Informationen zu den Servern führt das HSRZ in der Configuration Management Database (CMDB). Sobald ein Server in der CMDB angelegt ist, können für diesen Rechte über die Antragsverwaltung beantragt werden. Wenn Server abgebaut werden, dokumentieren dies die Administratoren in der CMDB. Die bestehenden Berechtigungen für abgebaute Server bleiben in der Antragsverwaltung jedoch erhalten. Somit gibt die Antragsverwaltung nicht den tatsächlichen Stand aller Berechtigungen auf Servern wider.

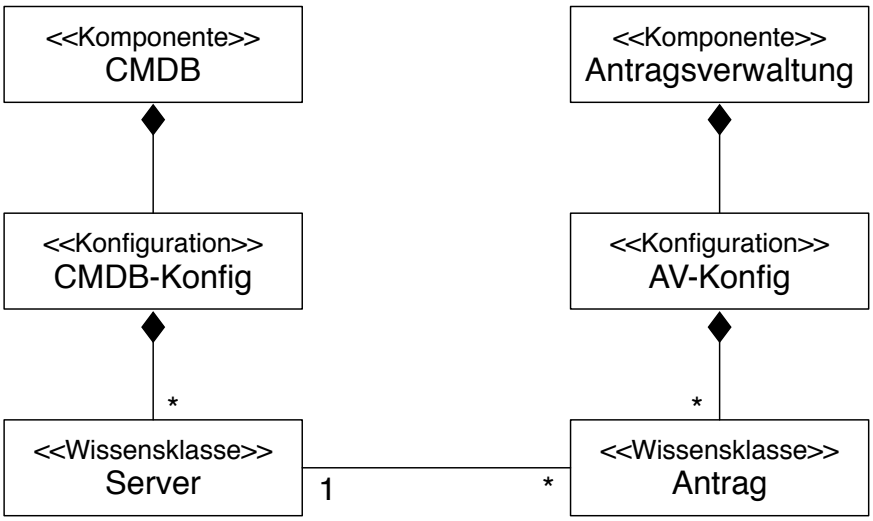


Abb. 8.25: *Abhängigkeit zwischen der Antragsverwaltung und der CMDB*

Beispiele: Abteilung A1 beantragt einen Server im Rechenzentrum (siehe Abbildung 8.26). Die Administratoren bauen diesen auf und dokumentieren dies in der CMDB. Nach dem Aufbau beantragen Mitarbeiter der Abteilung A1 Berechtigungen für den Server. Nach einer Zeit ist der Server obsolet, und die Abteilung A1 beantragt den Abbau. Die für diesen Server erstellten Berechtigungen haben jedoch weiterhin den Antragsstatus „Eingerichtet“.

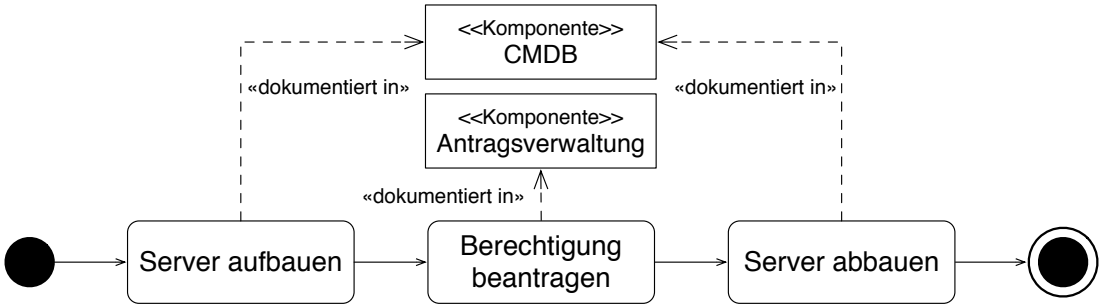


Abb. 8.26: *Der Server-Workflow der CMDB*

Gefahrenbewertung:
Durch abgebaute Server entstehen keine gefährlichen Möglichkeiten, auf Server zuzugreifen. Es können jedoch Entscheidungen beeinflusst werden. Das kann geschehen, wenn der Akzessor Berechtigte des Antrags einen neuen Antrag stellt und der Genehmiger aufgrund der Tatsache, dass der Akzessor schon eine Berechtigung hat, diesen Antrag bewilligt. Darüberhinaus verfälschten Berechtigungen auf nicht existierende Server mehrere Statistiken und führen zu einer zu komplexen Konfiguration.

Der Abgebaute-Server-Sicherheitstest:
Aus Sicht der Gefahrenklasse Abgebauter Server existieren in einem Rechenzentrum folgende Grundobjekte:

- Eine Menge S von Servern.
- Die Menge AS { 'Erstellt', 'In Genehmigung', 'In Realisierung', 'Eingerichtet', 'In Löschung', 'Gelöscht', 'Abgelehnt' } der Antragsstatus.
- Die Menge SS { 'Activated', 'Disassembled' } der Serverstatus.
- Ein Server s ist ein Paar (sid, ss) mit sid, einem eindeutigen Wort zur Identifikation des Servers, und dem Serverstatus $ss \in SS$.
- Eine Menge L von Anträgen (Logins). Ein Antrag l wird mit einer eindeutigen Nummer aid identifiziert und durch das Tripel $(aid, (sid, ss), as)$ mit dem Server $(sid, ss) \in S$ und dem dem Antragsstatus $as \in AS$ beschrieben.

Eine Berechtigung gilt als vakant falls

$$\text{vakant}(aid, (sid, ss), as) =_{\text{def}} (as =' \text{Eingerichtet}' \wedge ss =' \text{Disassembled}').$$

Die Menge aller als vakant geltenden Berechtigungen ermitteln sich durch

$$\text{vakant}(L) =_{\text{def}} \{l \in L \,|\, \text{vakant}(l)\}.$$

Testmenge: Das Erstellen der Testmenge benötigt die Wissensklasse Antrag aus der TORX-Antragsverwaltung sowie die Wissensklasse Server aus der CMDB (siehe Abbildung 8.27). Der Filter reduziert die Testmenge auf die Anträge mit den Status Eingerichtet. Dadurch entstand eine Testmenge von 4219 Testobjekten.

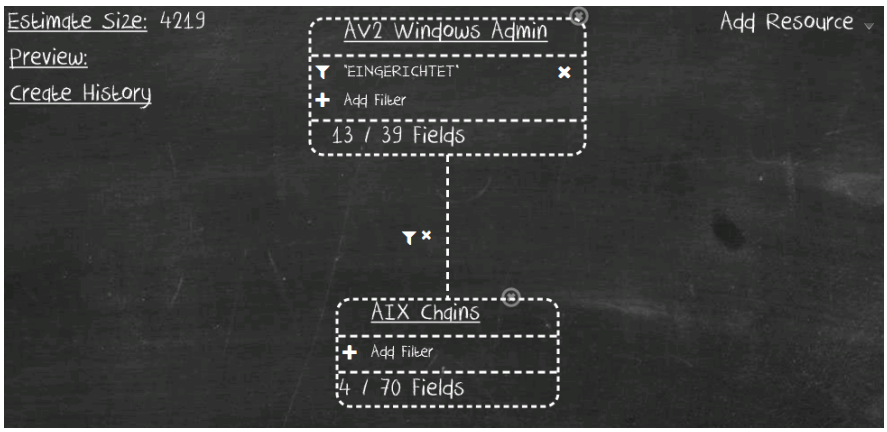


Abb. 8.27: *Definition der Testmenge für abgebaute Server mit dem Wissensbinder*

Auswertung: Die erste Analyse auf Basis der Testmenge ergab 135 Gefahren. Diese Gefahren waren, wie in Abbildung 8.28 dargestellt, auf Server von 135 verschiedenen Abteilungen verteilt.

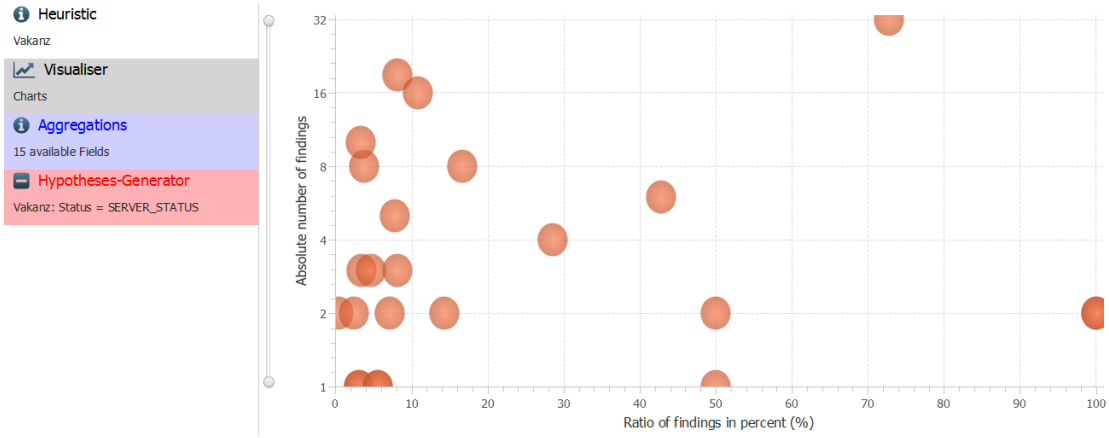


Abb. 8.28: *Darstellung der abgebauten Server als Motion-Chart*

Die Auswertungen der abgebauten Server wurden den Betreibern der TORX-Antragsverwaltung zur Verfügung gestellt. Diese konnte auf der Basis

der Auswertung sämtliche abgebauten Server eliminieren, sodass am nächsten Tag keine abgebauten Server mehr existierten. Die Betreiber der TORX-Antragsverwaltung erhalten seither regelmäßig und automatisiert eine aktualisierte Version der Auswertung von KUSTOS zugesendet.

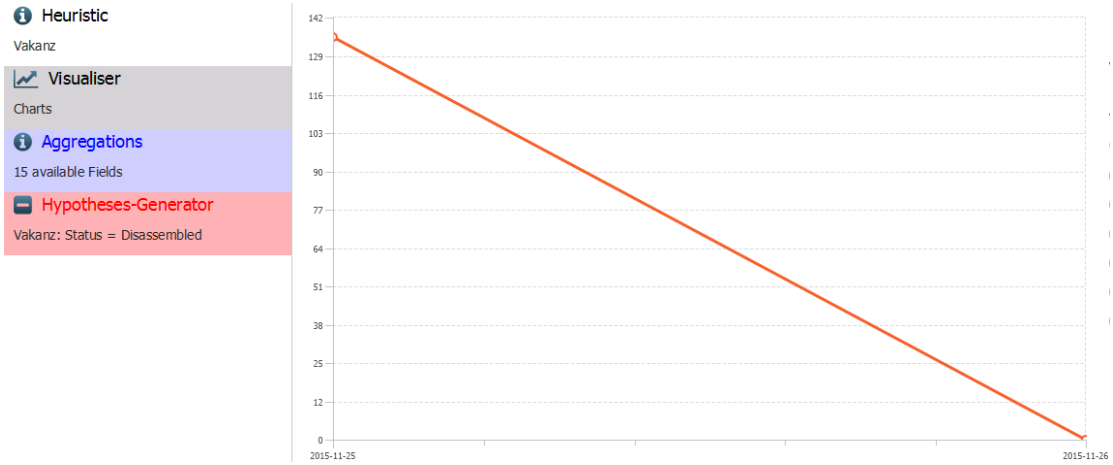


Abb. 8.29: *Entwicklung der abgebauten Server als Liniendiagramm*

8.1.3.5 Witwe

Ein allgemein im HSRZ geltender Grundsatz ist die Vermeidung von Single Point of Failure (SPOF). Dieser Grundsatz gilt sowohl bei der Ausfallsicherheit als auch bei der Zugriffssteuerung. Jedes Recht muss mit zwei unabhängigen Sicherheitsmechanismen geschützt sein. Die fehlerhafte Konfiguration oder die Fehlfunktion eines Sicherheitsmechanismus führt dadurch noch nicht zu einer unbeabsichtigten Zugriffsmöglichkeit. Das hat zur Folge, dass man nur dann auf eine Ressource zugreifen darf, wenn man mindestens zwei entsprechende Autorisierungen besitzt.

Der Grundsatz, SPOFs zu vermeiden, wird Zwei-Rechte-Prinzip genannt. Das Zwei-Rechte-Prinzip wird im HSRZ umgesetzt, indem man für den Zugriff auf Server des Rechenzentrums zwei Berechtigungen benötigt. Neben der Anmeldeberechtigung muss auch ein entsprechender Netzwerkzugriff bestehen. Der Netzwerkzugriff darf ausschließlich über SSL-VPN-Gateway oder das Citrix-Terminal erfolgen. Es müssen immer Bi-Tupel von Netzwerk- und Anmeldeberechtigung bestehen.

Definition: Die Möglichkeit, den SPOF-Grundsatz zu brechen, wird mit der Gefahrenklasse Witwe beschrieben. Eine Witwe liegt vor, wenn eine Person sich an einem Server anmelden darf, aber keine entsprechende Berechtigung hat, über das Netzwerk an diesen Server zu gelangen (siehe Abbildung 8.30). Das Recht, sich am Server anzumelden, sei als Anmeldeberechtigung definiert. Das Recht, auf einen Server über das Netzwerk auf die Anmeldeschnittstelle zuzugreifen, sei als Anmeldezugriff definiert.

Abb. 8.30: *Überblick über die Gefahrenklasse Witwe*

Beispiele: Eine Person darf sich an dem Server der Personalverwaltung (IP-Adresse 10.10.42.107) anmelden. Es besteht für diese Person jedoch weder über Citrix noch über SSL-VPN ein entsprechender Netzwerkzugriff.

Im Objektdiagramm der Abbildung 8.31 ist ein einfaches Beispiel mit einer Witwe modelliert. In dem dargestellten verteilten System existiert ein Anmeldezugriff $z1$ und die Anmeldeberechtigungen $b1$ und $b2$. Für die An-

meldeberechtigung $b1$ existiert der Anmeldezugriff $z1$ als Pendant und somit ist $b1$ keine Witwe. Für die Anmeldeberechtigung $b2$ existiert kein Anmeldezugriff und somit ist $b2$ eine Witwe.

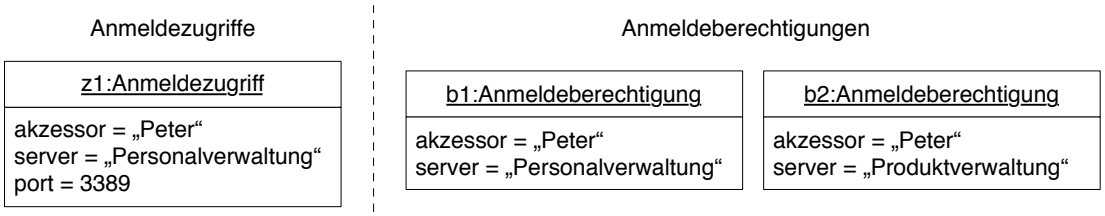


Abb. 8.31: *Objektmodell des HSRZ mit einer Witwe*

Gefahrenbewertung: Ein fehlender Netzwerkzugriff könnte darauf hinweisen, dass Herr Meier über einen unerlaubten Weg, eine Sicherheitslücke, auf den Server zugreift. Im Falle eines inkorrekten Anmelderechts könnte Herr Meier (bei einer fehlerhaften Citrix- oder SSL-VPN-Konfiguration) unberechtigt auf den Server der Personalverwaltung zugreifen.

Der Witwen-Sicherheitstest: Aus Sicht der Gefahrenklasse Witwe existieren in einem Rechenzentrum folgende Grundobjekte:

- Eine Menge A von Akzessoren.
- Eine Menge G von LDAP-Gruppen $g \subseteq A$.
- Eine Menge S von Servern.
- Eine Menge I von IP-Adressen. Unter einer IP-Adresse i kann höchstens ein Server erreichbar sein. Es sei $ser(i)$ derjenige Server, der unter der Adresse i erreichbar ist, falls es einen solchen gibt. Andernfalls ist $ser(i)$ nicht definiert. Die Menge der Server, die unter mindestens einer Adressen von $I' \subseteq I$ erreichbar ist, wird mit $ser(I') =_{def} \{ser(i) \mid i \in I \wedge ser(i) \text{ definiert}\}$ bezeichnet.
- Eine Menge P von Ports.

Damit ein Akzessor auf einen Server tatsächlich zugreifen kann, muss sowohl eine entsprechende *Netzwerkregel* (Regeln des SSL-VPN oder des Citrix) als auch ein entsprechende *Anmeldeberechtigung* (gültiger Benutzeraccount auf dem Server) bestehen. Es sei N die Menge der vorhandenen Netzwerkregeln.

Eine Netzwerkregel ist ein Tripel $n = (G', P', I')$ mit $G' \subseteq G$, $P' \subseteq P$ und $I' \subseteq I$. Sie legt fest, dass jeder Akzessor $a \in \bigcup_{g \in G'} g$ über jeden Port $p \in P'$ auf jeden Server $s \in ser(I')$ über das Netzwerk zugreifen darf. Eine Netzwerkregel (G', P', I') definiert somit die Menge

$$\text{zugriff}(n) =_{\text{def}} (\bigcup_{g \in G'} g) \times P' \times ser(I')$$

von erlaubten Zugriffen (a, p, s). Die Gesamtheit der Regeln von N erlauben die Zugriffe aus

$$\text{zugriff}(N) =_{\text{def}} \bigcup_{n \in N} \text{zugriff}(n).$$

Eine Anmeldung kann bei Windows-Servern mittels RDP über Port 3389 und bei Linux-Servern mittels SSH über Port 22 erfolgen. Die Menge der erlaubten Anmeldezugriffe AZ ist unter diesen Voraussetzungen gegeben durch

$$\text{anmeldeZugriff}(N) =_{\text{def}} \{(a, s) \mid \{(a, 22, s), (a, 3389, s)\} \cap \text{zugriff}(N) \neq \emptyset\}.$$

Eine Anmeldeberechtigung ist ein Paar $l = (a, s)$ mit $a \in A$ und $s \in S$. Eine Anmeldeberechtigung legt fest, dass sich der Akzessor a am Server s einloggen darf. Es sei L die Menge aller Anmeldeberechtigungen.

Eine Anmeldeberechtigung $l = (a, s)$ heißt Witwe bezüglich AZ, falls $(a, s) \notin AZ$. Die Menge aller Witwen ist somit gegeben durch

$$\text{witwen}(AZ, L) =_{\text{def}} \{(a, s) \in L \mid (a, s) \text{ Witwe bzgl. } AZ\}.$$

Auswertung Der Witwen-Sicherheitstest wurde erstmal mit einer Testmenge von 4389 Akzessor-Netzwerkregel-Tupeln ausgeführt. Der Sicherheitstest generierte auf Basis dieser Testmenge 645 Gefahrenhypothesen. Aufgrund der mit 645 Gefahrenhypothesen bereits recht unübersichtlichen Gefahrenmenge bietet sich die aggregierte Darstellung als Motion-Charts (siehe Abschnitt 6.4.1) an. Der in Abbildung 8.32 dargestellte Motion-Chart, wie auch die in den folgenden Abbildungen, sind nach dem betroffenen Server gruppiert.

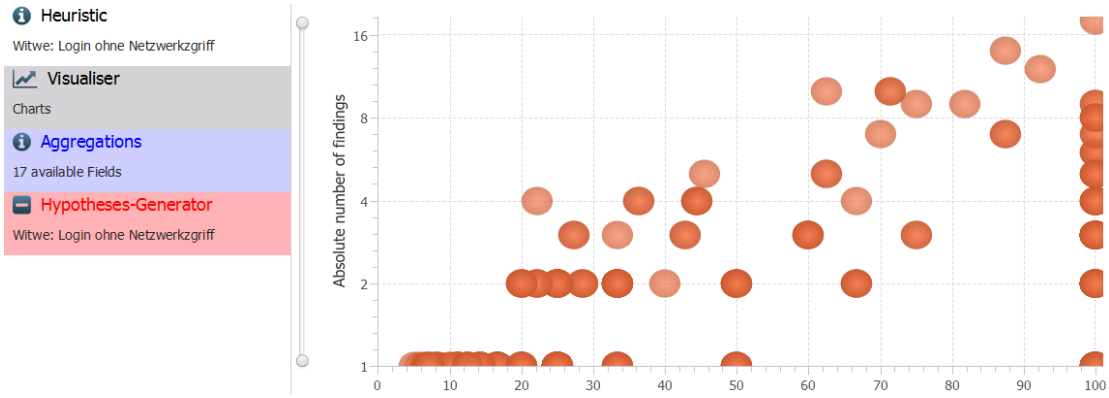


Abb. 8.32: *Motion-Chart der Ergebnisse des Witwen-Sicherheitstests*

Innerhalb dieser ersten Ergebnisse konnten Spezialisten einige β-Fehler identifizierten. 134 der Hypothesen basierten auf Logins für Server, die mittlerweile außer Betrieb genommen sind. Diese 134 Logins stellen somit keine Gefahr im Sinne der Gefahrenklasse Witwe dar, denn auf diesen Servern kann sich mit Sicherheit niemand mehr anmelden. Jedoch stellen solche Logins dennoch ein Problem dar. Die in TORX verwalteten Login-Anträge entsprechen nicht der Realität. TORX soll aber das führende System hinsichtlich der Login-Anträge sein. So entdeckten die Spezialisten während der Analysearbeiten der Wissensklasse Witwe die neue Wissensklasse „Abgebauter Server“ (siehe Abschnitt 8.1.3.4).

Die Spezialisten erweiterten deshalb den Sicherheitstest um einen Hypotheseninvalidator. Der Hypotheseninvalidator invalidiert alle Gefahrenhypothesen, die sich auf einen inaktiven Server beziehen Die verbleibenden 510 Gefahren sind in Abbildung 8.33 als Motion-Chart dargestellt und ebenfalls nach der Abteilung des Akzessors gruppiert.

Die Spezialisten identifizierten auch sogenannte funktionalen Zugänge als ein weiteres Ausschlusskriterium für Witwen. Mit diesen Zugängen können Akzessoren sich nicht remote am Server anmelden. Sie dienen ausschließlich der automatisierten, zeitbasierten Ausführung von Prozessen auf dem Server. Diese Erkenntnis erlaubte eine erneute Erweiterung des Sicherheitstests um einen entsprechenden Hypotheseninvalidator. Das führte zu eine Gesamtanzahl von 326 Gefahrenhypothesen (siehe Abbildung 8.34), das entspricht 51 Prozent der ursprünglichen Hypothesen.

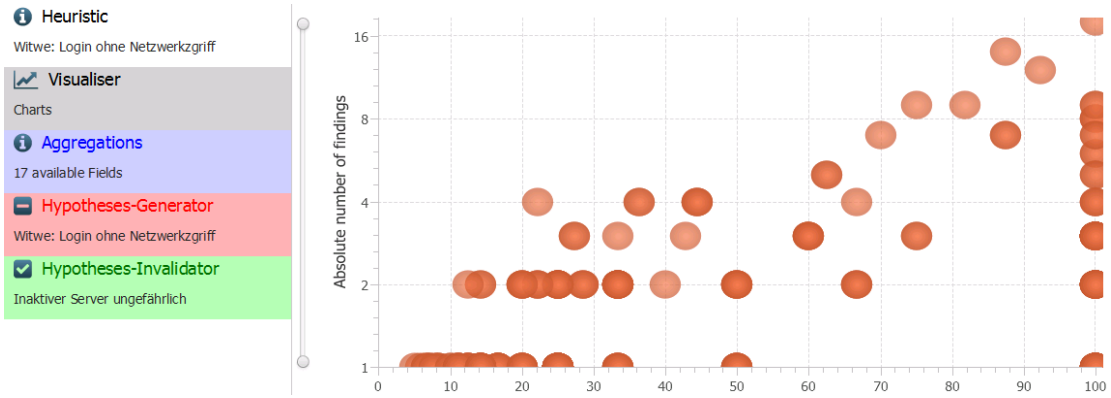


Abb. 8.33: *Motion-Chart nach der Invalidierung der Hypothesen inaktiver Server*

Der Prozess der Invalidierung führte zu einen Sicherheitstest mit zwei Hypotheseninvalidator. Anstatt der ursprünglich 645 Gefahrenhypothesen generiert der Sicherheitstest nur noch 326 Gefahrenhypothesen. All diese Gefahrenhypothesen stellen nach Auffassung der Spezialisten Gefahren dar. Diese sollen mit der Umstellung des Berechtigungsprozesses beseitigt werden[6].

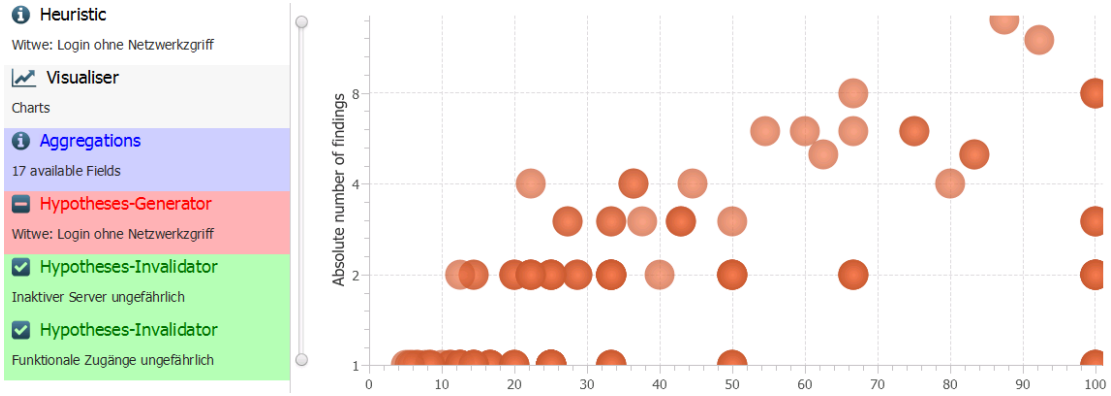


Abb. 8.34: *Gefahrenhypothesen nach der Invalidierung der funktionaler Zugänge*

[6]Die Umstellung des Berechtigungsprozesses ist im Rahmen der Gefahrenklasse Waise (siehe Abschnitt 8.1.3.6) vorgestellt. Diese Lösung geht sowohl die Problematik der Witwen als auch die der Waisen an.

8.1.3.6 Waise

Waisen resultieren aus dem Bruch des SPOF-Grundsatzes[7]. Eine Person darf über das Netzwerk auf einen Server zugreifen, auf dem sie sich jedoch nicht anmelden darf.

Beispiel: Ein Akzessor darf auf den IP-Bereich 10.10.42.96/28 über das SSL-VPN zugreifen. Alternativ: Ein Akzessor darf über Citrix auf die IP-Adresse 10.10.42.107 zugreifen. Der Server der Personalverwaltung hat die IP-Adresse 10.10.42.107 und kann somit vom Akzessor über das Netzwerk erreicht werden. Der Akzessor kann (und darf) sich jedoch auf dem Server der Personalverwaltung nicht anmelden.

Gefahrenbewertung: Wenn der Akzessor tatsächlich ein Anmelderecht besitzen darf, liegt keine Gefahr vor. Im Falle eines inkorrekten Netzwerkrechts könnte ein Akzessor (bei einer fehlerhaften Serverkonfiguration) unberechtigt auf den Server der Personalverwaltung zugreifen.

Der Waisen-Sicherheitstest: Aus Sicht dieser Gefahrenklasse Waise existieren in einem Rechenzentrum folgende Grundobjekte:

- Eine Menge A von Akzessoren.
- Eine Menge G von LDAP-Gruppen $g \subseteq A$.
- Eine Menge S von Servern.
- Eine Menge I von IP-Adressen. Unter einer IP-Adresse i kann höchstens ein Server erreichbar sein. Es sei $ser(i)$ derjenige Server, der unter der Adresse i erreichbar ist, falls es einen solchen gibt. Andernfalls ist S' nicht definiert. Die Menge der Server, die unter mindestens einer Adressen von $I' \subseteq I$ erreichbar ist, wird mit $ser(I') =_{def} \{ser(i) \mid i \in I' \wedge ser(i) \text{ definiert}\}$ bezeichnet.
- Eine Menge P von Ports.

Damit ein Akzessor auf einen Server tatsächlich zugreifen kann, muss sowohl eine entsprechende *Netzwerkregel* (Regeln des SSL-VPN oder des Citrix) als auch ein entsprechender *Login* (Benutzeraccount auf dem Server) bestehen. Es sei N die Menge der vorhandenen Netzwerkregeln.

[7] Der SPOF-Grundsatz ist in Abschnitt 8.1.3.5 (Witwen) beschrieben

Eine Netzwerkregel ist ein Tripel $n = (G', P', I')$ mit $G' \subseteq G$, $P' \subseteq P$ und $I' \subseteq I$. Sie legt fest, dass jeder Akzessor $a \in \bigcup_{g \in G'} g$ über jeden Port $p \in P'$ auf jeden von jedem Server $s \subseteq S$ auf das Netzwerk zugreifen darf. Eine Netzwerkregel (G', P', I') definiert somit die Menge

$$\text{zugriff}(n) =_{\text{def}} (\bigcup_{g \in G'} g) \times P' \times \{S'\}$$

von erlaubten Zugriffen (a, p, S').c Die Gesamtheit der Regeln von N erlauben die Zugriffe aus

$$\text{zugriff}(N) =_{\text{def}} \bigcup_{n \in N} \text{zugriff}(n).$$

Eine Anmeldung kann bei Windows-Servern mittels RDP über Port 3389 und bei Linux-Servern mittels SSH über Port 22 erfolgen. Die Menge der erlaubten Anmeldezugriffe ist unter diesen Voraussetzungen gegeben durch

$$\text{anmeldeZugriff}(N) =_{\text{def}} \{(a, S') \mid \{(a, 22, S'), (a, 3389, S')\} \cap \text{zugriff}(N) \neq \emptyset\}.$$

Ein Login ist ein Paar $l = (a, s)$ mit $a \in A$ und $s \in S$. Ein Login legt fest, dass sich der Akzessor a am Server s einloggen darf. Es sei L die Menge aller Logins.

Ein Anmeldezugriff (a, S') heißt *Waise* bzgl. L, falls $\forall s(s \in S' \rightarrow (a, s) \notin L)$ gilt. Die Menge aller Waisen ist somit gegeben durch

$$\text{waisen}(N, L) =_{\text{def}} \{(a, S') \in \text{anmeldeZugriff}(N) \mid (a, S') \text{ Waise bzgl. } L\}.$$

Abbildung 8.35 zeigt die Konfigurationsklassen, die für die Bestimmung der Waisen mit Hilfe des beschriebenen Sicherheitstests notwendig sind. Die Netzwerkregeln inklusive der Ports stellen eine Besonderheit dar, denn diese werden aus dem SSL-VPN-Gateway und aus der User-Firewall ausgelesen und vereint. Eine Netzwerkregel ist somit entweder eine SSL-VPN-Regel oder eine Firewallregel. Die Konfigurationen der Logins werden nicht von den Servern, auf den sie eingerichtet sind, sondern gemeinsam von der Antragsverwaltung bezogen.

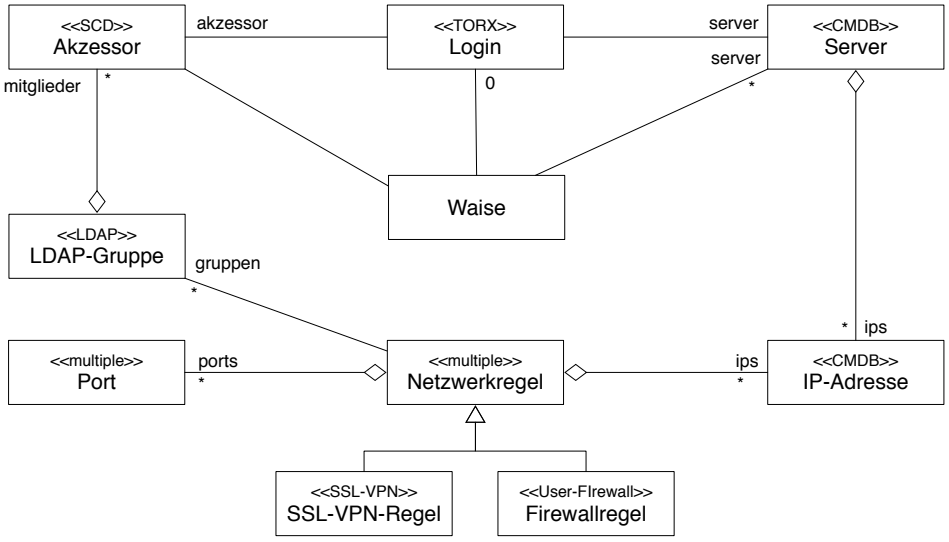


Abb. 8.35: *Modell von Waisen in der Konfiguration des Rechenzentrums*

In Quelltext 8.2 definiert eine Abfrage[8] für die Menge aller Waisen. In Zeile 1 werden die Netzwerkregeln und deren zugehörige Akzessoren selektiert. Die Zeile 4 der Abfrage werden definiert den Filter, dass nur die Netzwerkregeln, die auf mindestens einen der Port 22 und 3389 verweisen, relevant sind. In den Zeilen 5 bis 7 definieren für jedes Akzessor-Netzwerkregel-Tupel die Anzahl der existierenden Logins ermittelt. Nur die Akzessor-Netzwerkregel-Tupel, für die kein Login existiert (siehe Zeile 8), gelten als Waise.

```
1 SELECT reg.gruppen.mitglieder akzessor, reg regel,
2 FROM netzwerkregel reg
3 WHERE (
4         reg.port IN (22, 3389) AND
5         (       SELECT COUNT (*) logins FROM login login
6                 WHERE login.akzessor = akzessor
7                 AND login.server IN regel.ips.server
8         ) = 0
```

Quelltext 8.2: *Abfrage für die Ermittlung der Waisen*

Auswertung Die erste Auswertung des Sicherheitstests brachte eine Anzahl von 2071 Gefahrenhypothesen zu Waisen hervor. Diese Hypothesen

[8]Die Abfrage ist an die objektorientierte Sprache HQL angelehnt

sind, nach dem Namen der Netzwerkregel gruppiert, im Abbildung 8.36 als Motion-Chart dargestellt. Die Implementierung der Motion-Charts ist in Abschnitt 6.4.1 beschrieben.

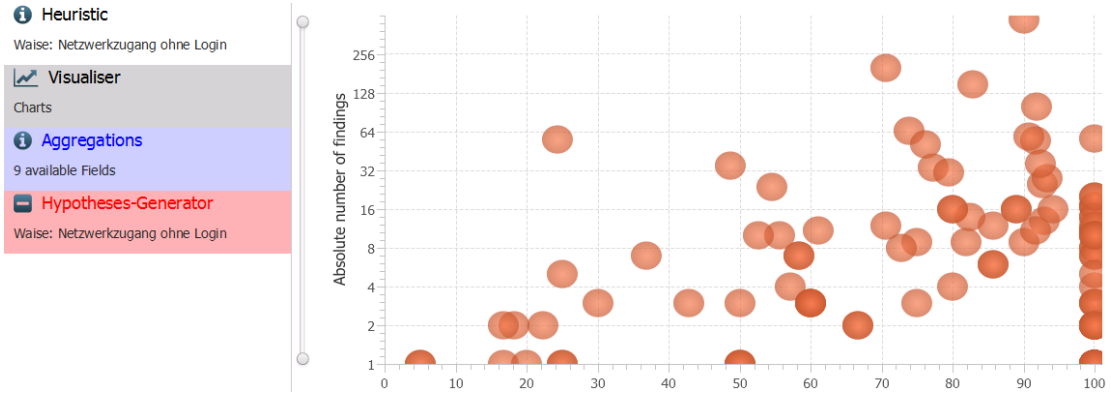


Abb. 8.36: *Die vom Sicherheitstest hypothetisierten Waisen*

Mehrere Gefahrenhypothesen wurde von Spezialisten als ungefährliche Netzwerkregeln identifiziert. Diese Netzwerkregeln waren für die Durchführung der Systemsicherungen notwendig und konnten nicht gefährlich werden. Deshalb wurden die Heuristik des Sicherheitstests um ein Hypotheseninvalidator für diese Backupregeln erweitert. Dadurch konnten 57 Gefahrenhypothesen invalidiert werden (siehe Abbidlung 8.37).

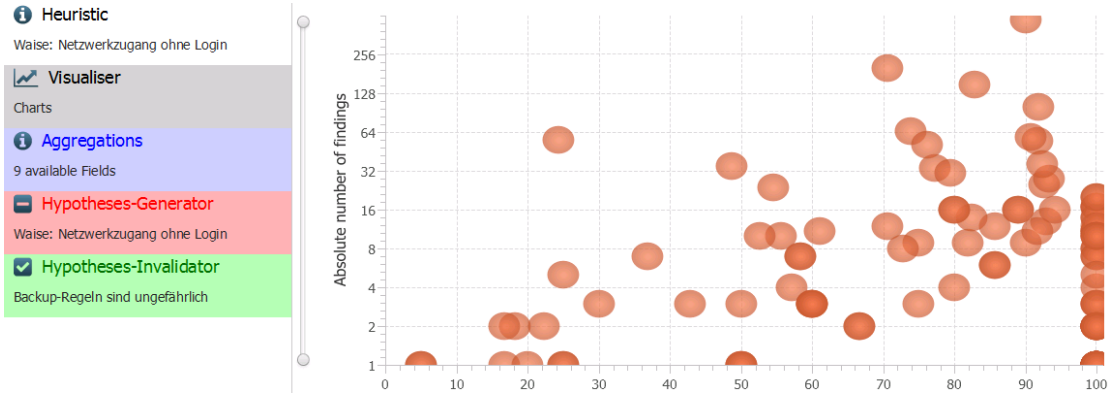


Abb. 8.37: *Hypothesen nach der Invalidierung der Backupregeln*

Zusätzlich konnten die Netzwerkregeln als ungefährlich identifiziert werden, die sich nicht auf Netzbereiche des Rechenzentrums bezogen. Es handelt sich dabei um Netzwerkregeln, die den Zugriff von Servern des Rechen-

zentrums auf andere interne Systeme des Unternehmens erlauben. Auch für diese falschen Hypothesen wurden Hypotheseninvalidator erstellt. Dies führte zu einer Reduzierung der Hypothesen um etwa 30 Prozent (siehe Abbidlung 8.38).

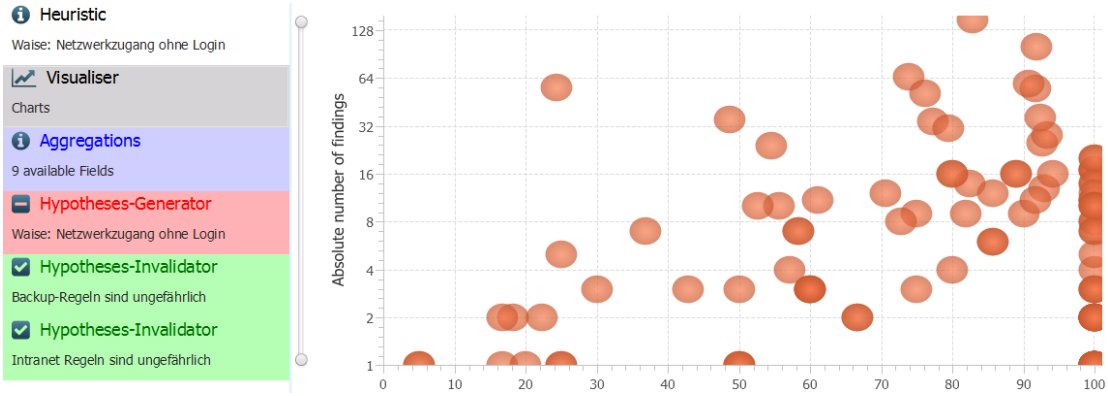


Abb. 8.38: *Hypothesen nach der Invalidierung der Intranetregeln*

Die 1336 verbleibenden Gefahrenhypothesen können nicht kurzfristig beseitigt werden. Sowohl bei Witwen als auch bei Waisen dürfen die Administratoren die Rechte nicht eigenmächtig entziehen. Denn das könnte den produktiven Betrieb empfindlich stören. Eine mittelfristige Lösung des Problems ist eine Umstellung des Berechtigungsprozesses. Netzwerkberechtigungen werden dann immer im gleichen Antrag mit den dazugehörigen Anmeldeberechtigungen beantragt werden. Sobald das Recht annulliert wird, können sowohl Netzwerkberechtigung als auch Anmeldeberechtigungen gleichzeitig entzogen werden. Eine Diskrepanz zwischen den Anmeldeberechtigungen und den Netzwerkberechtigungen wird somit verhindert[9].

[9]Wie und bis wann ein solche Prozess umgesetzt wird, ist bis dato noch nicht entschieden.

8.1.3.7 Zombie

Die Gefahrenklasse der Zombies beschreibt Unstimmigkeiten zwischen den Konfigurationen der Netzwerkkomponenten (SSL-VPN oder Citrix) und dem LDAP. Diese Komponenten werden zwar von verschiedenen Abteilungen getrennt gewartet, erfüllen jedoch durch ihr Zusammenspiel eine gemeinsame Aufgabe. Während die Netzwerkkomponenten definieren, welche Benutzergruppen welche Netzwerkberechtigungen besitzen, definiert das LDAP, welche Benutzer den Benutzergruppen angehören. Benutzergruppen können jedoch auch wieder gelöscht werden. Netzwerkregeln, die auf eine gelöschte Benutzergruppe verweisen, zeigen somit „ins Leere“ . Solche Verweise heißen Zombies.

Beispiele:
In Citrix existiert eine Regel „AccessPAN“. „AccessPAN“ verweist auf die Benutzergruppe „PanAccess“. Die Gruppe „PanAccess“ wird zu einem späteren Zeitpunkt in die beiden Gruppen „PanUser“ und „PanAdmins“ überführt. Die Gruppe „PanAccess“ selbst wird dabei gelöscht und die Administratoren des Citrix werden beauftragt, die beiden neuen Benutzergruppe in die Regel „AccessPAN“ aufzunehmen. Die Citrix-Regel „AccessPAN“ verweist danach auf drei Benutzergruppen von denen die Gruppe „PanAccess“ nicht mehr existiert. Dieser Zombie ist im Objektdiagramm der Abbildung 8.39 modelliert. Da das Objekt „PanAccess“ nicht mehr existiert, ist es durch gestrichelten Linien dargestellt.

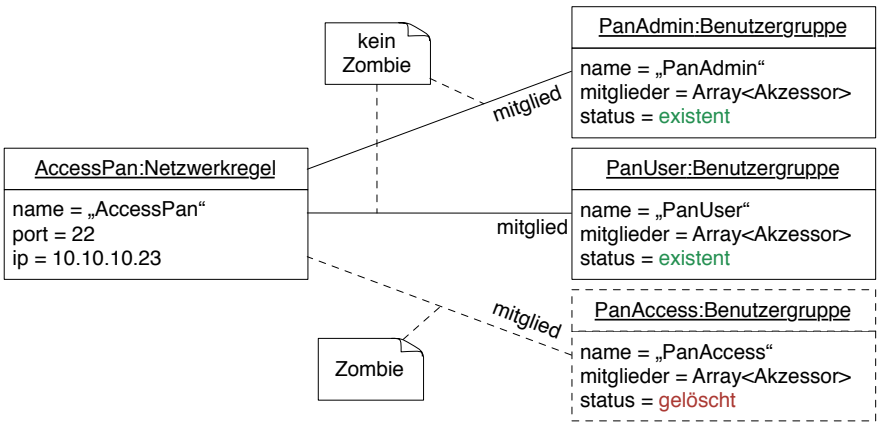


Abb. 8.39: *Modell eines Zombies*

Gefahrenbewertung:
Zombies sind aus zwei Gesichtspunkten eine Gefährdung:

Komplizierte und umfangreiche Konfigurationen wurden vom Betreiber des HSRZ als eine Ursache von Fehlkonfigurationen, und somit als eine Gefahr identifiziert. Deshalb ist ein erklärtes Ziel für das HSRZ, die Konfigurationen der Komponenten nicht unnötig kompliziert oder zu umfangreich zu gestalten. Wenn Netzwerkregeln auf nicht existierende Benutzergruppen verweisen, so liegt eine zu umfangreiche Konfiguration vor.

Eine größere Gefahr geht von der Möglichkeit aus, dass diese LDAP-Gruppe „gekapert“ wird. Denn nur für existierende LDAP-Gruppen kann definiert werden, wer diese administrieren darf. Eine nicht existierende Gruppe kann somit von einer Person angelegt werden, welche nicht berechtigt ist, LDAP-Gruppen für Netzwerkregeln zu administrieren. Dadurch wäre diese Person in der Lage, sich oder andere Personen unberechtigt zu einer Netzwerkregel hinzuzufügen.

Der Zombie-Sicherheitstest:
Aus Sicht der Gefahrenklasse Zombie existieren in einem Rechenzentrum folgende Grundobjekte:

- Eine Menge M von LDAP-Gruppen l, die je existiert haben.
- Eine Menge $L \subseteq M$ der aktuell existierenden LDAP-Gruppen.
- Eine Menge N der Netzwerkregeln.

Eine Netzwerkregel $n \in N$ ist ein Paar (i, M') mit dem Identifizierer der Netzwerkregel i (ein Wort) und den berechtigten Gruppen $M' \subseteq M$.

Die Menge der Netzwerkregelzuweisungen Z_n der Netzwerkregel n ergibt sich aus dem Kreuzprodukt $zuweisung(n) = \{i\} \times M'$. Eine Netzwerkregelzuweisung ist somit ein Paar (i, l). Die Menge Z aller Netzwerkregelzuweisungen bezüglich N ergibt sich aus $Z = \bigcup_{n \in N} zuweisung(n)$.

Eine Netzwerkregelzuweisung (i, l) ist ein Zombie, falls $l \notin L$. Die Menge aller Zombies in N wird wie folgt ermittelt:

$$zombies(Z, L) =_{\text{def}} \{(i, l) \in Z \mid l \notin L\}.$$

Auswertung: Die Heuristik wurde bei ihrer ersten Anwendung auf der Basis von 2310 existierenden Testobjekten, welche Netzwerkregelzuweisungen sind, eingesetzt. 379 dieser Testobjekte wurden als Zombie identifiziert (siehe Abbildung 8.40).

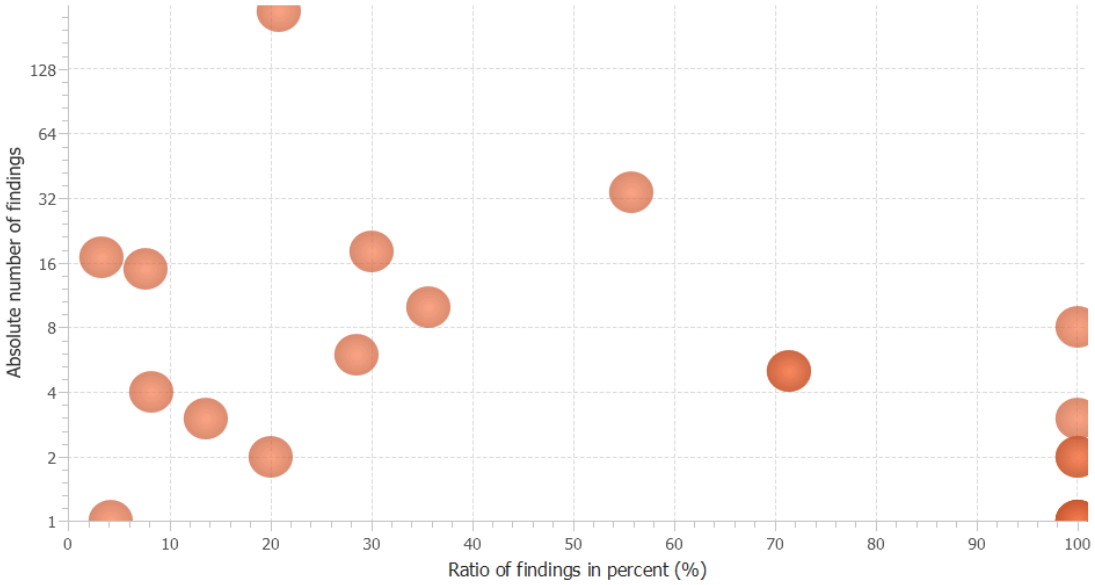


Abb. 8.40: *Motion-Chart der Zombies nach der ersten Analyse*

Diese Auswertung wurde der für Citrix und SSL-VPN zuständigen Netzwerkgruppe übermittelt, die daraufhin die Netzwerkregeln überarbeiteten. In Abbildung 8.40 sind die 379 Zombies, nach ihrer Zugehörigkeit zu Netzwerkregeln gruppiert, dargestellt. Abschnitt 6.4.1 beschreibt, wie der in dieser Abbildung 8.40 dargestellten Motion-Chart zu interpretieren ist.

Zwei Arbeitstage nach der ersten Analyse wurde die Heuristik erneut angewandt. Bei dieser zweiten Analyse wurden keine Zombies identifiziert. Laut Aussage eines Administrators der Netzwerkgruppe half die Auswertung bei der Administration der Netzwerkregeln. Nicht mehr existierende Benutzergruppen in Netzwerkregeln konnten somit effizient identifiziert und eliminiert werden. KUSTOS erzeugt seitdem regelmäßig und automatisiert einen Bericht über die aktuellen Zombies an die Netzwerkgruppe.

8.2 Fallstudie: RBG-Netzwerk

> *„ Wir [die RBG] bieten als IT-Abteilung für die Fakultäten Mathematik und Informatik als Rechen- und IT-Kompetenzzentrum ein breites Spektrum an Informations-/Kommunikations- und Mediendienstleistungen für die Wissenschaft “*
>
> Internetauftritt der RBG

Die RBG ist der interne IT-Dienstleister der Fakultäten Mathematik und Informatik FMI an der Technischen Universität München. Das Betreiben der Netzwerkinfrastruktur ist ein Dienst, den die RBG den einzelnen Organisationseinheiten (Lehrstähle und die Arbeitsgruppen) anbiett. Neben der Verwaltung der IP-Adressen und den zugehörigen Host-Einträgen betreibt die RBG auch die Firewall. Dadurch können die Organisationseinheiten ihre Systeme sowohl für den gewünschten Zugriff freigeben, als auch vor ungewünschten Zugriff schützen. Die Möglichkeit, die Erreichbarkeit selbstverantwortlich und individuell zu gestalten, ist vor allem für die Lehrstühle sehr wichtig. Denn die Lehrstühle möchten zugleich die freie Verfügbarkeit ihrer Lehrmaterialien und den Schutz ihrer Forschungsergebnisse. Bei den Lehrstühlen besteht in zweierlei Hinsicht eine hohe Fluktuation: Ein Teil der Systeme werden für einzelne Forschungsprojekte oder Lehrveranstaltungen bereitgestellt und oft schon nach weniger als einem Jahr wieder abgebaut. Und das Personal, sowohl die wissenschaftlichen Mitarbeiter als auch die wissenschaftlichen Hilfskräfte, ist oft nur wenige Jahre am Lehrstuhl beschäftigt. Die Kombination einer selbstverantwortlichen und individuellen Entscheidungsgewalt auf der einen Seite und Volatilität auf der anderen Seite zieht Gefährdungen mit sich. Das häufige Wechseln mancher Systeme erfordert einerseits, dass der Konfiguration der Netzwerkinfrastruktur eine hohe Sorgfalt entgegengebracht werden muss. Der häufige Austausch des Personals führt andererseits dazu, dass die Entscheidungsgrundlage für existierende Konfigurationen niemanden bekannt ist. Der Druck durch steigende Studentenzahlen und wichtige Forschungsprojekte erlaubt den Sicherheitsverantwortlichen keine gewissenhafte Einarbeitung in die bestehende Infrastruktur. Und die Sorge, durch die Veränderung bestehender Konfigurationen Schaden herbeizuführen, sind zwei Gründe dafür, dass veraltete Konfigurationen nicht eliminiert und somit gefährlich werden.

8.2.1 Komponenten- und Berechtigungsanalyse

Die RBG bietet mit ihrem Netzwerkmanagement die Infrastruktur, mit der die Organisationseinheiten ihre Server betreiben. Das Sicherheitsdiagramm in Abbildung 8.41 zeigt die von der RBG bereitgestellten Komponenten für das Netzwerkmanagement sowie die von den Organisationseinheiten im Netzwerk betriebenen Komponenten. Für jede angeschlossene Organisationseinheit betreibt die RBG eine Firewall, welche die Administratoren der RBG warten. Eine gemeinsame, netzwerkübergreifende Host-Datenbank erlaubt die Verwaltung von IP-Adressen, Hostnamen und Mac-Adressen. Jede Organisationseinheit betreibt in ihren Netzwerk eine Menge von Geräten (z. B. Server, Drucker und Arbeitsplatzrechner).

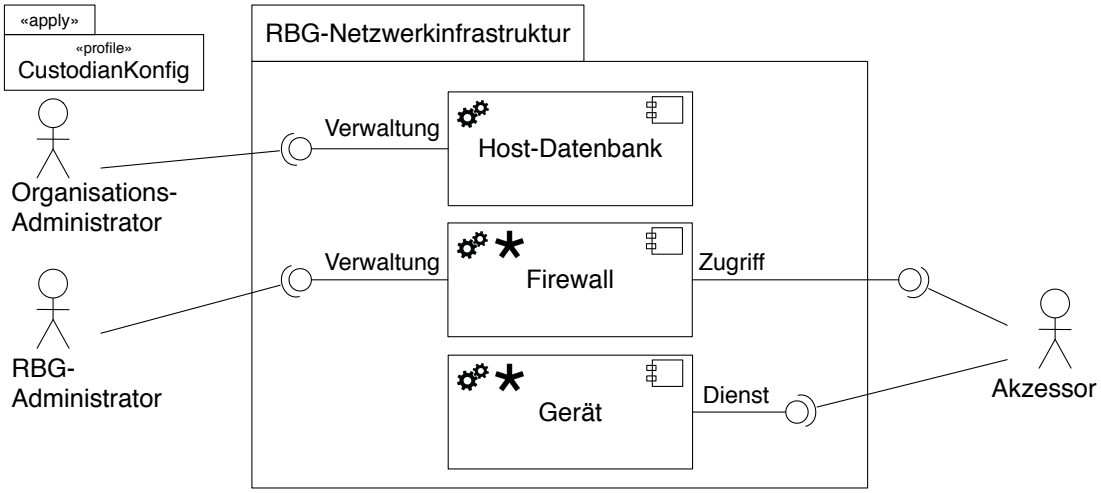


Abb. 8.41: *Sicherheitsdiagramm der Netzwerkdienstleisters RBG*

Die Berechtigungsanalyse führt zu einer Zugriffsmatrix, welche drei Komponententypen mit vier Diensten sowie drei Zugreifern definiert (siehe Tabelle 8.4). Die Administratoren der RBG verwalten die Firewall im Auftrag der Organisationseinheiten. Die Administratoren der Organisationseinheiten haben Zugriff auf die Verwaltung der Host-Datenbank. Akzessoren können mit Hilfe der Firewall auf das Netzwerk und die von den Geräten der Organisationseinheiten bereitgestellten Dienste zugreifen.

erfordert / bietet an	**Host-DB**	Verwaltung	**Firewall**	Verwaltung	Zugriff	**Gerät**	Dienst
RBG-Administrator				✓			
Organisations-Administrator		✓					
Akzessor					✓		✓

Tab. 8.4: *Zugriffsmatrix der Komponenten der RBG-Netzwerkdienste*

8.2.2 Gefahrenanalyse

Den Anstoß der Sicherheitsanalyse durch CUSTODIAN stellte die Erkenntnis am Lehrstuhl Brügge dar, dass viele der Firewallregeln veraltet waren. Diese Regeln wurden für Server erstellt, die mittlerweile nicht mehr existieren. Das führte zu der Problematik, dass diese Firewallregeln auch für neue Server, bei denen die IP-Adressen nicht mehr existierender Server wiederverwendet wurden, noch aktiv waren. Dadurch existieren Freischaltungen, die für diese neuen Server nicht vorgesehen waren und diese gefährden konnten. Diese Gefahrenklasse der abgebauten Server, welche bereits in der MTF existierte, ist in Abschnitt 8.2.2.1 beschrieben. Während der Auswertungen der Firewallregeln fiel darüberhinaus auf, dass einzelne Firewallregeln existieren, welche einen sehr umfangreichen Zugriff auf die Netzwerke gewähren. Manche Firewallregeln schalten alle IP-Adressen eines Netzwerks samt aller ihrer Ports frei und gewähren beispielsweise allen Rechnern des Münchner Wissenschaftsnetzes einen kompletten Zugriff auf das Netzwerk einer Organisationseinheit. Da das Münchner Wissenschaftsnetzes relativ groß ist und auch nicht vertrauenswürdige Quellen[10] einschließt, stellen diese Regeln eine Gefahr für ein Netzwerk dar. Abschnitt 8.2.2.2 beschreibt diese Gefahrenklasse.

[10]Viele Wohnheime sind an das Münchner Wissenschaftsnetzes angeschlossen

8.2.2.1 Abgebauter Server

Das Abbauen von bestehenden Geräten[11] erfordert von dem Administrator der Organisationseinheit mehrere Arbeitsschritte. Zunächst muss der Server selbst außer Betrieb genommen werden und dann müssen der Eintrag in der Host-Datenbank und die betreffenden Firewallregeln in der Firewallkonfiguration gelöscht werden. Diese Schritte werden nicht immer durchgeführt. Das kann vor allem zu einer Gefahr führen: Firewallregeln werden vor der Neuvergabe der IP-Adresse nicht gelöscht und gelten dann für einen neuen Server. Da eine solche Firewallregel nicht für den neuen Server konzipiert ist kann sie eine Gefahr verursachen.

Die Gefahrenklasse „Abgebauter Server“ beschreibt Firewallregeln mit einer Ziel-IP-Adresse, die keinem aktuell existierenden Server zugewiesen ist.

Beispiel: Der Server mit der IP-Adresse 131.159.39.93 wird nicht mehr gebraucht und abgebaut. Eine Firewallregel für die IP-Adresse 131.159.39.93, die für jeden beliebigen Zugreifer den Port 80 freigibt, bleibt jedoch bestehen. Diese verbleibende Firewallregel stellt eine Gefahr nach der Definition dieser Gefahrenklasse dar. Denn es ist möglich, dass zu einem späteren Zeitpunkt ein neuer Server die IP-Adresse 131.159.39.93 erhält. Aufgrund der alten Firewallregel könnte weltweit auf den Server zugegriffen werden, auch wenn die dahinterstehenden Informationen nur für interne Zwecke bestimmt sind.

Gefahrenbewertung: In manchen Fällen liegt keine Gefahr vor, in anderen Fällen gefährden die Firewallregeln die im Netzwerk betriebenen Komponenten immens. Die Hilfe der Administratoren der betroffenen Organisationseinheit ermöglicht eine Gefahrenbewertung. Selbst diesen liegen oft nicht alle Informationen vor, um die Gefährdung alleine zu bestimmen.

Der Abgebaute-Server-Sicherheitstest: Aus Sicht der Gefahrenklasse „Abgebauter Server“ besteht das RBG-Netzwerk aus folgenden Grundobjekten:

- Die Menge I aller 2^{32} IPv4-Adressen.
- Die Menge P aller 2^{16} verfügbaren Ports.
- Eine Menge O von Organisationseinheiten.

[11] Die meisten Geräte sind Server, deshalb wird im Folgenden von Servern gesprochen.

- Eine Menge N von Netzwerken. Eine Netzwerk $n \in N$ ist ein Paar (o, na) mit dem Organisationseinheit $o \in O$ und den Namen des Netzwerks na.
- Eine Menge F von Firewallregeln für das Netzwerk $n \in N$. Eine Firewallregel $f \in F$ ist ein Tupel (n, rn, SI, SP, DI, DP) mit dem Regelnamen rn. Die Source-IP-Adressen $SI \subset I$ und die Source-Ports $SP \subset P$ bestimmen die Berechtigten. Die Source-IP-Adressen $DI \subset I$ und die Source-Ports $DP \subset P$ bestimmen, auf welche IP-Adressen mit welchen Ports in Netzwerk $n \in N$ zugegriffen werden darf.

Für diese Gefahrenklasse werden nur die Firewallregeln F' überprüft, die Zugriff auf genau eine IP-Adresse gewähren. Die Funktion singular(F) ermittelt diese Firewallregeln (n, rn, SI, SP, di, DP) mit $di \in P$.

$$F' = \text{singular}(F) =_{\text{def}} \{(n, rn, SI, SP, DI, DP) \in F \mid |DI| = 1\}$$

Die boolesche Funktion berechtigt(i, f) stellt fest, ob die IP-Adresse i von der Firewallregel f berechtigt wird:

$$\text{berechtigt}(i, f) \mathrel{\widehat{=}} \text{berechtigt}(i, (n, rn, SI, SP, di, DP)) =_{\text{def}} i \in SI$$

Die boolesche Funktion erreichbar(i, f) wird von dem Rechner mit der IP i ausgeführt. Sie versucht, eine Telnet-Verbindung mit dem Host di über einen der Ports $p \in DP$ aufzubauen. Wenn die Funktion beim Verbindungsaufbau keine Antwort erhält, gibt sie `falsch` und sonst `true` zurück. Diese Funktion ist die einzige Funktion in dieser Dissertation, welche ein Black-Box-Verfahren einsetzt, um Informationen über das verteilte System zu erlangen (siehe Abschnitt 3.5). Das war notwendig, weil die RBG keinen Informationen über die aktuell existierenden Server in einem Netzwerk hat.

Die Funktion abgebaut ermittelt die Menge aller potenziell abgebauten Server:

$$F' = \text{abgebaut}(i, F) =_{\text{def}} \{f \in F' \mid \neg\text{berechtigt}(i, f)\}$$

Die Auswertungen der Gefahrenklassen „Abgebauter Server“ und „Generalisierung“ fasst Abschnitt 8.2.2.3 zusammen.

8.2.2.2 Generalisierung

> *„Das Problem ist, dass man nie einen Überblick über die Konfiguration der Firewall hat und Gefahren nicht einschätzen kann.“*
>
> – Helma Schneider, Administratorin

Der Einsatz einer Firewall soll gewährleisten, dass Server nur unter den notwendigen Ports erreichbar und von außerhalb ihres Netzwerks erreichbar sind. Je konsequenter diese Prämisse verfolgt wird, desto expliziter sind die Freischaltungen, die durch eine Firewallregel gewährt werden. Einige Organisationseinheiten verfolgen diese Prämisse jedoch nicht konsequent. Denn die pauschale Freischaltung ganzer Netzwerkbereiche reduziert den Aufwand, der mit der Verwaltung der Firewalls verbunden ist. Dementsprechend existieren Regeln, die beispielsweise alle 512 IP-Adressen eines Netzwerks die gesamten 65.536 Ports für das gesamte Münchner Wissenschaftsnetz (MWN) freischalten. Aus diesem Münchner Wissenschaftsnetz wurden bereits Angriffe auf Infrastrukturen registriert. Das ermöglicht Angreifern, das Netzwerk auf über 33 Millionen verschiedene Port-IP-Adress-Kombinationen nach Verwundbarkeiten zu durchsuchen. Mit Brute-Force-Attacken können Angreifer beispielsweise schwache Passwörter von SSH-Zugängen erraten [OJ08]. Angreifer können ebenfalls bekannte Sicherheitslücken von Webservern ausnutzen [KV03] und so auf vertrauliche Informationen auf dem Webserver zugreifen, oder von dort aus Zugriff auf andere Komponenten des Netzwerks erlangen.

Beispiele: Folgende Beispiele konnten während der Analyse der Firewallregeln entdeckt und als gefährlich identifiziert werden.

- Eine Firewallregel gibt für Zugreifer aus dem MWN den Port 22 für alle IP-Adressen des gesamten Netzwerks einer Organisationseinheit frei.
- Eine Firewallregel gibt für alle weltweiten Zugreifer alle Ports einer IP-Adressen im Netzwerk einer Organisationseinheit frei.
- Eine Firewallregel gibt für Zugreifer aus dem MWN alle Ports auf allen IP-Adressen des gesamten Netzwerks einer Organisationseinheit frei.

Gefahrenbewertung:
In manchen Fällen liegt keine Gefahr vor, in anderen Fällen gefährden die Fi-

rewallregeln die im Netzwerk betriebenen Komponenten immens. Die Hilfe der Administratoren der betroffenen Organisationseinheit ermöglicht eine Gefahrenbewertung. Selbst diesen liegen oft nicht alle Informationen vor, um die Gefährdung allein zu bestimmen.

Der Generalisierungs-Sicherheitstest:
Aus Sicht der Gefahrenklasse „Generalisierung“ besteht das RBG-Netzwerk aus folgenden Grundobjekte:

- Die Menge I aller 2^{32} IPv4-Adressen.
- Die Menge P aller 2^{16} verfügbaren Ports.
- Eine Menge O von Organisationseinheiten.
- Eine Menge N von Netzwerken. Eine Netzwerk $n \in N$ ist ein Paar (o, na) mit dem Organisationseinheit $o \in O$ und den Namen des Netzwerks na.
- Eine Menge F von Firewallregeln für das Netzwerk $n \in N$. Eine Firewallregel $f \in F$ ist ein Tupel (n, rn, SI, SP, DI, DP) mit den Regelnamen rn. Die Source-IP-Adressen $SI \subset I$ und die Source-Ports $SP \subset P$ bestimmen die Berechtigten. Die Source-IP-Adressen $DI \subset I$ und die Source-Ports $DP \subset P$ auf welche IP-Adressen mit welchen Ports im Netzwerk $n \in N$ zugegriffen werden darf.

Die Kardinalität einer Firewallregel, also die Anzahl der Zugriffswege die sie in ein Netzwerk freigibt, ergibt sich aus.

$$\text{kardinal}(f) \mathrel{\widehat{=}} \text{kardinal}(n, rn, SI, SP, DI, DP) =_{\text{def}} |DI| * |DP|$$

Alle Firewallregeln, die eine Kardinalität größer 1 haben, sollen von den Administratoren der betroffenen Organisationseinheit überprüft werden. Die Menge aller Firewallregeln mit eine Kardinalität größer 1 ergibt sich aus:

$$\text{kardinal}(F) =_{\text{def}} \{f \in F \mid \text{kardinal}(f) > 1\}$$

Die Auswertungen der Gefahrenklassen „Abgebauter Server“ und „Generalisierung“ fasst Abschnitt 8.2.2.3 zusammen.

8.2.2.3 Auswertung

Die Auswertung basiert auf der Analyse von sechs Lehrstuhlnetzwerken. Dabei fokussiert die Auswertung auf die Lehrstuhlnetzwerke mit besonders vielen Firewallregeln. CUSTODIAN kann die gefährlichen Firewallregeln von den ungefährlichen Firewallregeln nicht selbständig unterscheiden. Hierfür ist zu viel individuelles Hintergrundwissen zu den Firewallregeln, dem Netzwerk und seinen Servern und Diensten notwendig. Jedoch ist CUSTODIAN in der Lage, potenziell gefährliche Firewallregeln zu identifizieren um diese den Administratoren als Entscheidungsgrundlage bereitzustellen. Wir präsentierten den Administratoren der Lehrstühle die Auswertungen der Gefahrenklassen „Abgebauter Server“ und „Generalisierung“ als Zusammenfassung.

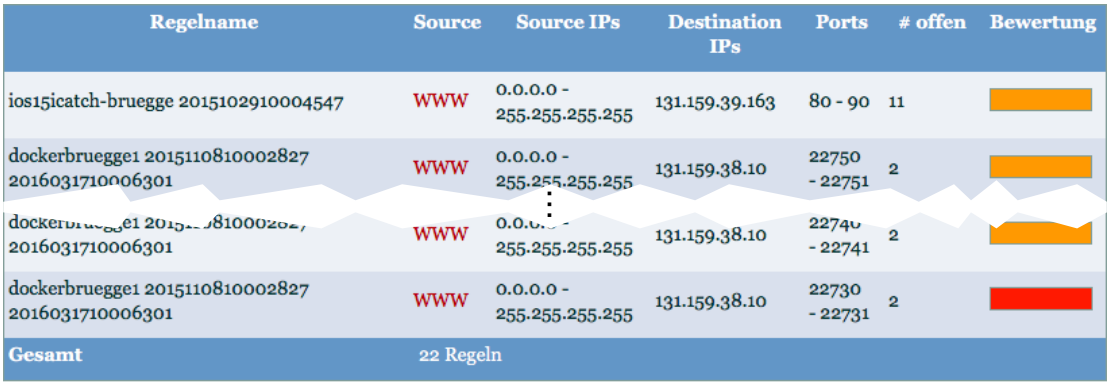
Abbildung 8.42 zeigt einen Auszug der potenziell abgebauten Server, die KUSTOS den Administratoren des Lehrstuhls Brügge bereitgestellte. Die Analyse enthält für jeden potenziell abgebauten Server den Regelnamen, die IP-Adresse, die geöffneten Ports und den Hostnamen, welcher der IP-Adresse zugewiesen ist.

Regelname	IP-Adresse	Ports	Host
ios15br-bruegge 2015102910004547	131.159.38.188	80	ios15br-bruegge.informatik.tu-muenchen.de
pom15bruegge 2015102910004547	131.159.39.113	80	pom15bruegge.informatik.tu-muenchen.de
ios15ntt-bruegge 2015102910004547	131.159.38.185	80	ios15ntt-bruegge.informatik.tu-muenchen.de
⋮			
monitorbruegge 2015102910004547	131.159.38.103	80	monitorbruegge.informatik.tu-muenchen.de
monitorbruegge 2015102910004547	131.159.38.103	443	monitorbruegge.informatik.tu-muenchen.de
Gesamt	30 Regeln		

Abb. 8.42: *Analyse der abgebauten Server am Lehrstuhl Brügge*

Abbildung 8.43 zeigt einen Auszug der potenziellen Generalisierungen für den Lehrstuhl Brügge. Die Analyse enthält den Regelnamen, eine Kategorisierung der berechtigten Source-IP-Adressen (FMI, MWN oder WWW) und die definierten Source-IP-Adressen. Die Destination-IP-Adressen und die Ports geben an, auf welche Ressourcen im Netzwerk aufgrund der Regel zugegriffen werden darf. Die Kardinalität der Firewallregel ist unter „offen“ angegeben. Eine farbliche Bewertung visualisiert die Größe der potentiellen Gefahr mit Farben zwischen gelb und rot. Die zugrundeliegende Heuristik

zieht hierfür die Kriterien Source, Destination-IPs und Ports zur Berechnung heran.



Regelname	Source	Source IPs	Destination IPs	Ports	# offen	Bewertung
ios15icatch-bruegge 2015102910004547	WWW	0.0.0.0 - 255.255.255.255	131.159.39.163	80 - 90	11	
dockerbruegge1 2015110810002827 2016031710006301	WWW	0.0.0.0 - 255.255.255.255	131.159.38.10	22750 - 22751	2	
dockerbruegge1 2015110810002827 2016031710006301	WWW	0.0.0.0 - 255.255.255.255	131.159.38.10	22740 - 22741	2	
dockerbruegge1 2015110810002827 2016031710006301	WWW	0.0.0.0 - 255.255.255.255	131.159.38.10	22730 - 22731	2	
Gesamt	22 Regeln					

Abb. 8.43: *Analyse der Generalisierungen am Lehrstuhl Brügge*

Bei allen sechs Lehrstühlen erkannte KUSTOS potenziell gefährliche Firewallregeln. Die Reaktion auf die zur Verfügung gestellten Analysen war sehr unterschiedlich (siehe Tabelle 8.5). Bei zwei Lehrstühlen haben wir bisher noch keinen zuständigen Administratoren erreicht. Bei weiteren zwei Lehrstühlen identifizierten die Administratoren Gefahren, sie müssen jedoch noch prüfen, inwieweit die Firewallregeln eingeschränkt werden können. Den Administratoren empfahl ich den Kontakt zur RBG aufzunehmen, eine Reaktion habe ich jedoch noch nicht erhalten. Und bei den letzten zwei Lehrstühlen half die Auswertung den Administratoren, um das Regelwerk der Firewall zu bereinigen.

Lehrstuhl	Status
Brügge	Persönliches Gespräch mit den Administratoren. Erste Regeln sind gelöscht, weitere sollen folgen.
Bungartz	Administrator bisher nicht persönlich erreichbar.
Knoll	Persönliches Gespräch mit dem Administrator.
Krcmar	Persönliches Gespräch mit dem Administrator. Gefährliche Regeln sind identifiziert und gelöscht.
Navab	Administrator bisher nicht persönlich erreichbar.
Pretschner	Persönliches Gespräch mit dem Administrator.

Tab. 8.5: *Identifizierte Generalisierung der einzelnen Lehrstühle*

KUSTOS generierte beim Lehrstuhl Brügge 30 Gefahrenhypothesen für „Abgebauter Server“ und 22 Gefahrenhypothesen für „Generalisierung“ (siehe

Tabelle 8.6). 27 der 30 Firewallregeln waren tatsächlich für nicht mehr existierende Server und deren Löschung wurde von den Administratoren beauftragt. Bei den 22 Generalisierungen identifizierten die Administratoren 10 gefährliche Regeln, von denen sie bei fünf die Löschung beauftragten. Bei den verbleibenden fünf gefährlichen Regeln muss noch geprüft werden, ob sie durch weniger generelle Substitutregeln ersetzt werden müssen.

Lehrstuhl	Anzahl der gesamten Firewallregeln	Abgebauter Server		Generalisierung	
		initial	gelöscht	initial	gelöscht
Brügge	145	30	27	22	5 (+5)
Bungartz	107	14	–	23	–
Knoll	168	32	–	54	–
Krcmar	503	281	281	42	20
Navab	136	30	–	47	–
Pretschner	105	32	–	33	–

Tab. 8.6: *Identifizierte Generalisierung der einzelnen Lehrstühle*

Für den Lehrstuhl Krcmar generierte KUSTOS 281 Gefahrenhypothesen für „Abgebauter Server“ und 42 Gefahrenhypothesen für „Generalisierung“ (siehe Tabelle 8.6). Nach der Prüfung der Gefahrenhypothesen stellten die Administratoren fest, dass alle 281 Firewallregeln, die zu abgebauten Servern führten, obsolet sind und gaben deren Löschung in Auftrag. Ebenso konnten sie 20 der 42 Generalisierungsregeln als überflüssig ausmachen und haben auch deren Löschung in Auftrag.

Bei den Lehrstühlen Bungartz und Navab kann bisher noch keine Aussage über die Korrektheit der Gefahrenhypothesen gemacht werden. Nur ein Gespräch mit deren Administratoren kann dies klären. Bei den Lehrstühlen Knoll und Pretschner wurden laut Aussage der Administratoren beunruhigende Funde gemacht. Die Administratoren hatten zum Gesprächszeitpunkt noch nicht die notwendigen Informationen, um bestimmen zu können, ob sie Firewallregeln löschen können.

8.3 Fallstudie: AdWorks2go

Die financeTec AG (kurz financeTec) entwickelt und vertreibt die Software AdWorks, eine Kundenverwaltungs-Software, die speziell für Vermögensberater und Versicherungsvermittler (allgemein Berater genannt) konzipiert ist. Diese Software wurde jedoch zuerst nicht an die Berater selbst, sondern nur an Beraterorganisationen vermietet, welche als Zwischenhändler auftreten und die notwendigen Systeme betreiben. Diese Organisationen erhalten eine uneingeschränkte Lizenz von AdWorks. Mit dieser Lizenz dürfen sie AdWorks auf ihren Systemen betreiben und an Ihre Kunden weitervermieten. Aufgrund der uneingeschränkten Lizenzvergabe der Software AdWorks, war bis zu diesem Zeitpunkt das Lizenzmanagement bei der financeTec trivial.

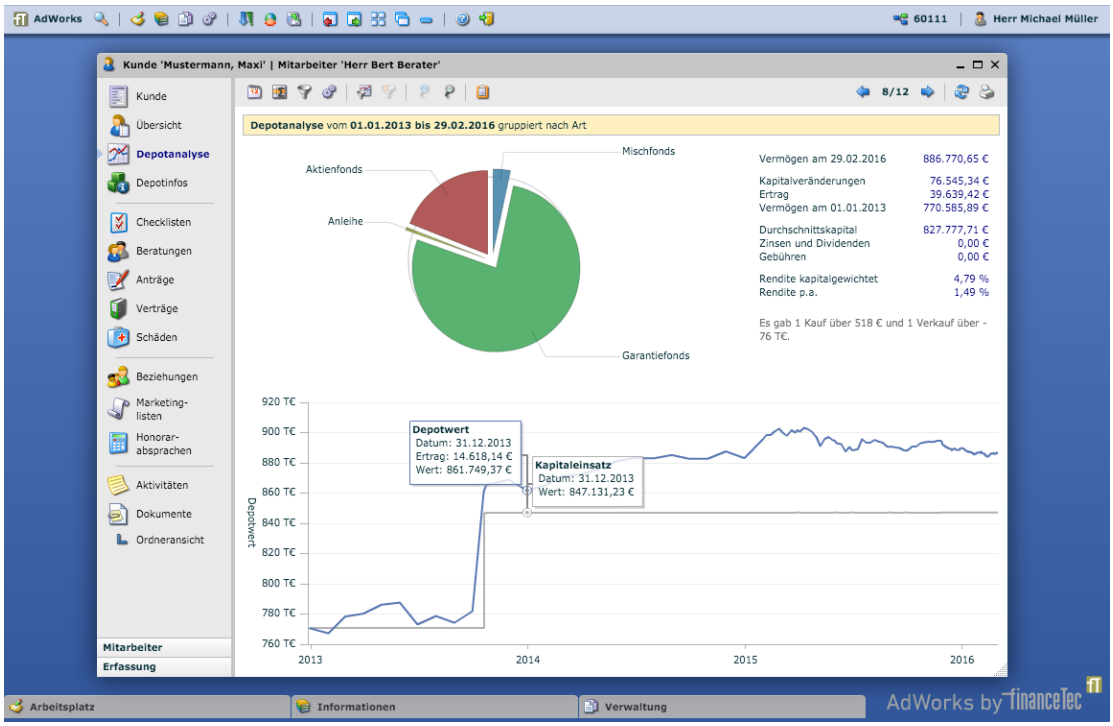


Abb. 8.44: *Depotanalyse eines Kunden in AdWorks*

Diese ursprüngliche Vermarktungsstrategie ermöglichte eigenständigen Berater, die keiner Organisation angehören, die Nutzung von AdWorks nicht. Um auch diese Berater als Kunden zu erreichen, bietet financeTec AdWorks

nun auch als Dienst an, der auf eigenen Servern betrieben wird. Dieser Dienst wird unter dem Namen AdWorks2go angeboten. AdWorks2go kann seitdem von eigenständigen Beratern direkt von financeTec gemietet werden. Im Gegensatz zu den Lizenzgebühren der Zwischenhändler sind die Nutzungsgebühren nicht pauschal veranlagt. Die Nutzungsgebühren sind von den in Anspruch genommenen Modulen und der Anzahl der aktiven Nutzer abhängig. AdWorks2go ist, wie in Abbildung 8.45 modelliert, eine Installation der Software AdWorks.

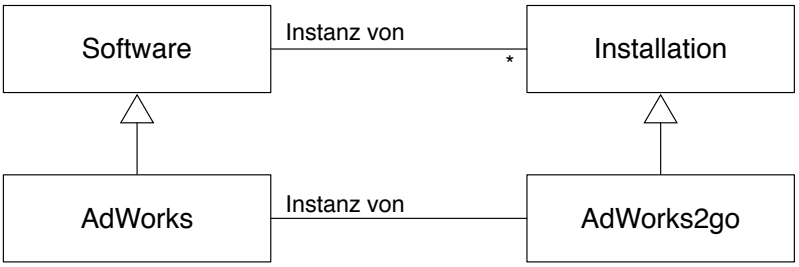


Abb. 8.45: *AdWorks2go als Installation der Software AdWorks*

Aufgrund der ursprünglichen pauschalen Vermarktung ist AdWorks technisch nicht auf die modulbasierte Fakturierung optimiert. Die hierfür notwendigen Informationen werden nicht in einer hinreichend übersichtlichen Form bereitgestellt. Für die monatlich stattfindende Rechnungsstellung müssen die erbrachten Leistungen für jeden Kunden ermittelt werden. Da dieser Prozess sehr zeitaufwändig ist, wird er nicht immer vollständig durchgeführt. Das führte dazu, dass Leistungen nicht korrekt oder gar nicht berechnet werden. Ein Hauptproblem ist, dass die benötigten Informationen auf verschiedene Stellen des Systems verteilt sind. Es dauert beispielsweise zwischen ein und zwei Minuten, um für einen einzelnen Berater herauszufinden, ob er aktiv ist.

8.3.1 Komponentenanalyse

In Abbildung 8.46 ist die Systemumgebung von AdWorks2go modelliert. Administratoren, Berater und Kunden können als Akteure mit AdWorks2go interagieren. Die Administratoren sind Mitarbeiter von financeTec, die das System für die Berater konfigurieren. Die Berater sind Mitarbeiter der Beratungsunternehmen, die AdWorks2go mieten, und setzen das System für ihre Arbeit ein. Kunden sind die Personen, die Dienstleistungen von den Beratern beziehen. Administratoren, Berater und Kunden können sind an

der Komponente `Session-Manager` anmelden, um danach auf die Funktionalitäten und Daten zugreifen zu können. Die Anmeldung erfolgt entweder über ein Flash-basiertes Programm, welches über einen Browser aufgerufen wird, oder über eine mobile iPhone-Applikation.

Im `Lizenzmanager` kann von Administratoren eingestellt werden, welche Module die einzelnen Berater gebucht haben. Das System stellt jedoch nicht konsequent sicher, dass Berater nur die Leistungen nutzen können, die auch gebucht sind. Somit sind die ausgewählten Lizenzen lediglich der Sollzustand und nicht der Istzustand. In der Komponente `Logging` werden verschiedene Ereignisse, wie beispielsweise das Eröffnen und das Beenden einer Session, gespeichert. In den Komponenten `Kunden` und `Berater` werden alle Informationen zu den beratenen Kunden und den Beratern gespeichert. AdWorks2go ist eine White-Label-Produkt, das heißt, es kann optisch an die Anforderungen der Berater angepasst werden. Diese optischen Anpassungen (Designs) werden auf dem Konfigurationsserver verwaltet.

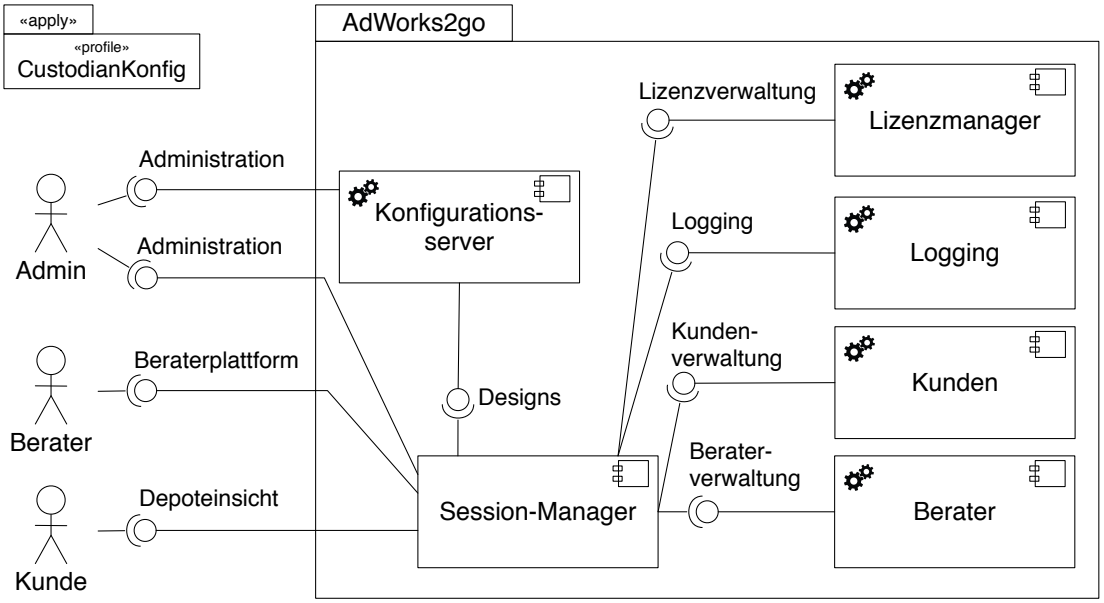


Abb. 8.46: *Sicherheitsdiagramm für AdWorks2go*

8.3.2 Berechtigungsanalyse

In Abbildung 8.7 ist die Zugriffsmatrix von AdWorks dargestellt. Administratoren können sowohl den Session-Manager als auch Konfigurationsserver über spezielle Administrationsschnittstellen administrieren. Berater können sich über die Schnittstelle Beraterplattform des Session-Managers am System anmelden. Und der Session-Manager kann auf die Schnittstellen der Lizenzverwaltung, des Loggings der Kundenverwaltung und der Beraterverwaltung zugreifen. Darüberhinaus kann der Session-Manager Designs vom Konfigurationsserver abfragen.

erfordert / bietet an	**Session-Manager**	Administration	Beraterplattform	Depoteinsicht	Lizenzverwaltung	Logging	Kundenverwaltung	Beraterverwaltung	**Konfigurationsserver**	Administration	Designs
Administrator		✓								✓	
Berater			✓								
Kunde				✓							
Session-Manager					✓	✓	✓	✓			✓

Tab. 8.7: *Zugriffsmatrix der Komponenten bei AdWorks2go*

8.3.3 Gefahrenanalyse

> *„Wenn Rechnungen fehlerhaft erstellt werden kann das zu Mindereinnahmen oder zu Regressforderungen führen. Darüberhinaus beschädigen sie unser professionelles Image und provozieren Konflikte mit Kunden. Somit stellen inkorrekte Rechnungen für die financeTec AG eine Gefahr dar.“*
>
> Michael Müller – Vorstand financeTec AG

Den Anstoß der Gefahrenüberprüfung durch CUSTODIAN stellte die Befürchtung des Vorstandes der financeTec dar, dass fehlerhafte Rechnungen Schaden herbeiführen können. Die Struktur und die Verteilung der für die Rechnungsstellung notwendigen Daten ist schwer auszuwerten und führte zu einer Diskrepanz zwischen den in Anspruch genommenen und den berechneten Leistungen.

Im Rahmen der Überprüfungen wurden tatsächlich zwei verschiedene Klassen von fehlerhaft berechneten Leistungen identifiziert. Der monatlich kostenpflichtige Onlinezugriff wurde von einigen Beratern für Kunden freigeschaltet, jedoch nicht berechnet. Diese unbezahlten Onlinezugriffe sind in Abschnitt 8.3.3.1 beschrieben. Die einmaligen Gebühren für individuelle Designs wurden häufig nicht fakturiert. Solche unbezahlten Designs sind in Abschnitt 8.3.3.2 beschrieben.

Während der Überprüfungen der gestellten Leistungen wurde noch eine sicherheitsrelevante Gefahrenklasse entdeckt. Berater konnten auf Geschäftsstellen zugreifen, für die sie nicht autorisiert sind. Diese Gefahrenklasse ist in Abschnitt 8.3.3.3 beschrieben.

8.3.3.1 Unbezahlter Onlinezugriff

Die Kunden der Berater können sich an AdWorks2go anmelden um ihre Bestände einzusehen und mit den Beratern in Kontakt zu treten. Dem Beratern wird hierfür das Modul Online-Depoteinsicht berechnet. Das Modul Online-Depoteinsicht wird den Beratern monatlich und pauschal für alle seine Kunden berechnet. In der Komponente Lizenzmanager können die Administratoren einstellen, welche Berater die Online-Depoteinsicht gebucht haben (siehe Abbildung 8.47).

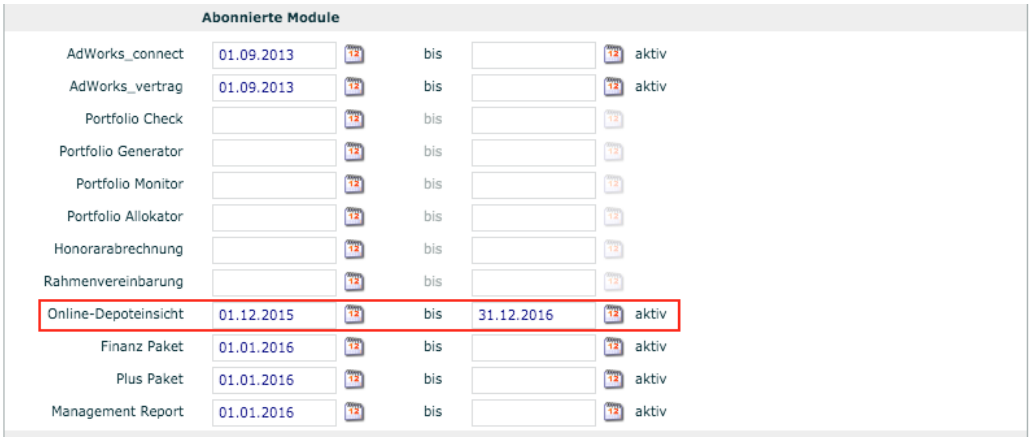


Abb. 8.47: *Aktivierung des Moduls Online-Depoteinsicht*

Einer Geschäftsstelle sind, wie in Abbildung 8.48 modelliert, Berater und Kunden zugeordnet. Jeder Kunde hat eine Menge von Logins, die er bisher durchführte.

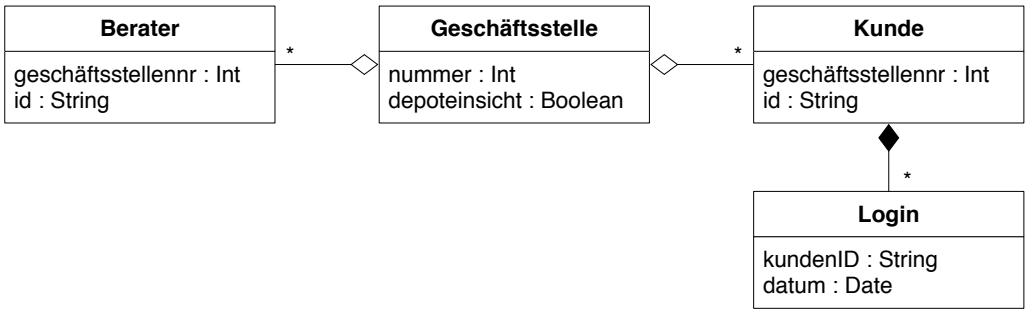


Abb. 8.48: *Modell einer Geschäftsstelle in AdWorks2go*

Die Einstellung im Lizenzmanager hat einen rein informativen Charakter. Sie steuert nicht, ob ein Berater seinen Kunden die Online-Depoteinsicht zur Verfügung stellen kann. Dies wird vom Berater für jeden Kunden individuell

konfiguriert. Es ist demnach allen Beratern technisch möglich die Online-Depoteinsicht zu nutzen. Deshalb ist für eine korrekte Rechnungsstellung das alleinige Prüfen des Lizenzmanagers nicht ausreichend. Hierfür ist es für alle Berater notwendig, bei jeden Kunden zu prüfen, ob er sich entweder im betreffenden Monat eingeloggt hat oder ob seine Online-Depoteinsicht aktiviert ist. Eine manuelle Überprüfung ist bei aktuell über 3.500 verwalteten Kunden mit vertretbarem Aufwand nicht möglich.

Beispiel:

- Ein Berater hat das Modul Online-Depoteinsicht nicht gebucht. Dennoch hat er aktuell bei ein oder mehreren seiner Kunden die Online-Depoteinsicht freigeschaltet.
- Ein Berater hat das Modul Online-Depoteinsicht nicht gebucht. Dennoch hat sich einer seiner Kunden während des Monats angemeldet. Die Freischaltung der Online-Depoteinsicht hat der Berater mittlerweile deaktiviert.

Gefahrenbewertung: Durch nicht berechnete Module der „Online-Depoteinsicht“ entsteht financeTec ein finanzieller Schaden.

Der Unbezahlter-Onlinezugriff-Sicherheitstest: Aus Sicht der Gefahrenklasse „Unbezahlter Onlinezugriff“ besteht AdWorks2go aus folgenden Grundobjekte:

- Eine Menge G von Geschäftsstellen. Eine Geschäftsstelle $g \in G$ ist ein Paar (gn, og) mit der Geschäftsstellennummer gn und den booleschen Wert og, der aussagt, ob für die Geschäftsstelle das Modul Online-Depoteinsicht gebucht ist.
- Eine Menge K von Kunden. Eine Kunden $k \in K$ ist ein Tripel (g, i, ok) mit seiner zugeordneten Geschäftsstelle g, seiner ID i und ok, den booleschen Wert, der aussagt, ob der Kunde für die Online-Depoteinsicht freigeschaltet ist.
- Eine Menge L von Logins. Eine Login $l \in L$ ist ein Paar (ik, d) mit der ID des Kunden ik und dem Datum der Anmeldung d.
- Eine Menge M diejenigen Monate, in denen mindestens ein Login stattgefunden hat (zb: 2016-01).

Die Funktion login ermittelt, ob sich ein Kunde k im Monat m eingeloggt hat:

$$\text{login}(m, k, L) =_{\text{def}} \forall(ik, d) \in L \rightarrow (ik \neq k \vee d \text{ liegt nicht in } m)$$

Die Funktion aktivK ermittelt ob einem Kunden $k \equiv (g, i, ok)$ entweder momentan Online-Depoteinsicht freigeschaltet ist oder ob es sich im Monat m angemeldet hatte:

$$\text{aktivK}(m, (g, i, ok), L) =_{\text{def}} ok = true \vee login(m, k, L)$$

Die Funktion kunden ermittelt alle Kunden M' einer Geschäftsstelle g':

$$K' = \text{kunden}(g', K) =_{\text{def}} \{(g', i, ok) \in K\}$$

Die Funktion aktivG ermittelt ob eine Geschäftsstelle g im Monat m mindestens einen aktiven Kunden hatte:

$$\text{aktivG}(g, m, L, K) =_{\text{def}} \exists\, k(k \in \text{kunden}(g, K) \wedge \text{aktivK}(m, k, L))$$

Die Funktion modul ermittelt, ob eine Geschäftsstelle $g \equiv (gn, og)$ im Monat m für das Modul Online-Depoteinsicht bezahlen muss, dieses jedoch nicht gebucht ist:

$$\text{modul}(g, m, L, K) =_{\text{def}} \{aktivG(g, m, L, K) \wedge og = false\}$$

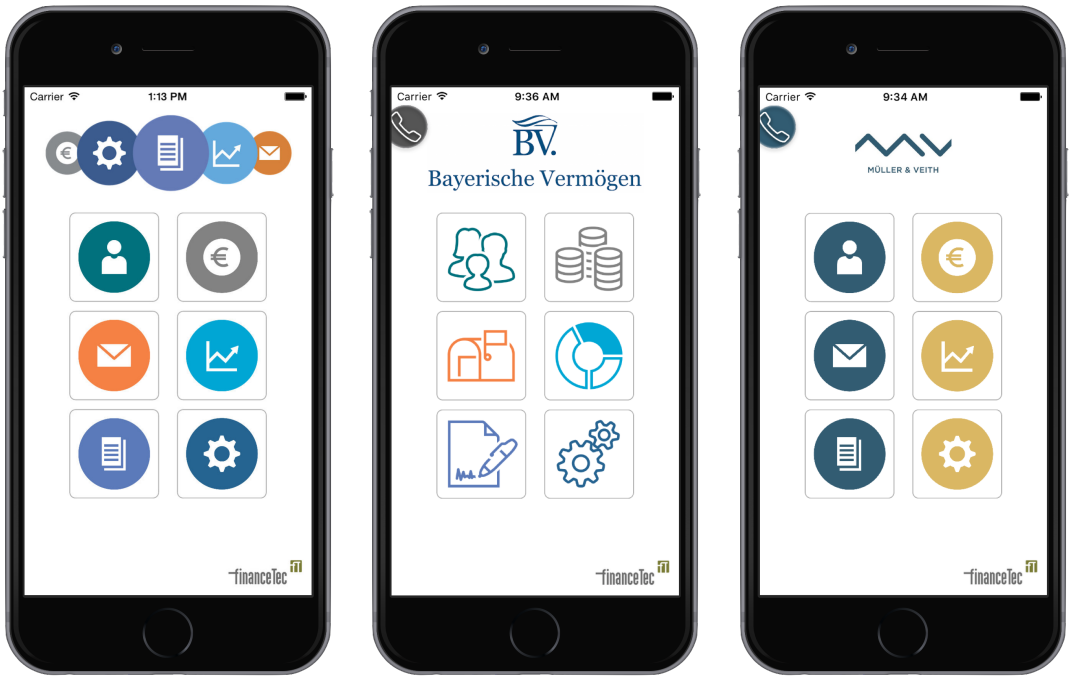
Letztendlich ermittelt die Funktion module die Menge G' der Geschäftsstellen, die im Monat m für das Modul Online-Depoteinsicht bezahlen muss, dieses jedoch nicht gebucht haben:

$$G' = \text{module}(G, m, L, K) =_{\text{def}} \{\, g \in G \,|\, \text{modul}(g, m, L, K)\}$$

Auswertung: Bei der Auswertung traten insgesamt drei Geschäftsstellen auf, bei denen Kunden Onlinezugriffe gewährten wurden, jedoch das Modul Online-Depoteinsicht nicht gebucht war. Dadurch entsteht der financeTec monatlich finanzieller Schaden. Die Auswertung wird es der financeTec in Zukunft ermöglichen, das Modul Online-Depoteinsicht korrekt in Rechnung zu stellen. Mittelfristig ist eine Änderung der Implementierung geplant, die es Beratern ohne den Modul Online-Depoteinsicht, verwehrt ihren Kunden den Onlinezugriffe bereitzustellen.

8.3.3.2 Unbezahltes Design

Die Kunden der Berater können sich, soweit freigeschaltet, an AdWorks2go anmelden um ihre Bestände einzusehen und mit den Beratern in Kontakt zu treten. Für diesen Kundenzugang stellt financeTec ein kostenfreies Standarddesign bereit (siehe Abbildung 8.49a). Neben diesem Standarddesign können Berater auch eigene Designs bestimmen und die Kundenoberfläche mit ihrem individuellen Logo und Farbschema zu gestalten (siehe Abbildungen 8.49b und 8.49c).



(a) Standarddesign (b) Individuelles Design 1 (c) Individuelles Design 2

Abb. 8.49: *Verschiedene Designs der AdWorks2go iPhone-Anwendung*

Der Aufwand für das Erstellen eines Designs wird von financeTec einmalig in Rechnung gestellt. Jedoch ist es schwierig, einen Überblick darüber zu erlangen, welche Designs schon berechnet sind und welche nicht. Im Konfigurationsserver sind die Designs mit der Geschäftsstelle des Servers identifiziert. Und in den Rechnungen werden die Berater ohne ihre Geschäftsstellen erwähnt.

Beispiel:
Ein Berater bestellt ein individuelles Design. Dieses wird erstellt, jedoch nicht berechnet.

Gefahrenbewertung:
Durch nicht berechnete Designs entsteht financeTec ein finanzieller Schaden.

Der Unbezahltes-Design-Sicherheitstest:
Aus Sicht der Gefahrenklasse „Unbezahltes Design“ besteht AdWorks2go aus folgenden Grundobjekten:

- Eine Menge B von Beratern. Eine Berater $b \in B$ ist ein Paar (gb, n) mit der Geschäftsstellennummer gb und dem Namen des Beraters n.
- Eine Menge D von Designs. Eine Design $d \in D$ ist ein Paar (gd, i) mit der Geschäftsstellennummer gd und der Individualisierung i.

Die Menge der qualifizierten Designs QD ist definiert durch

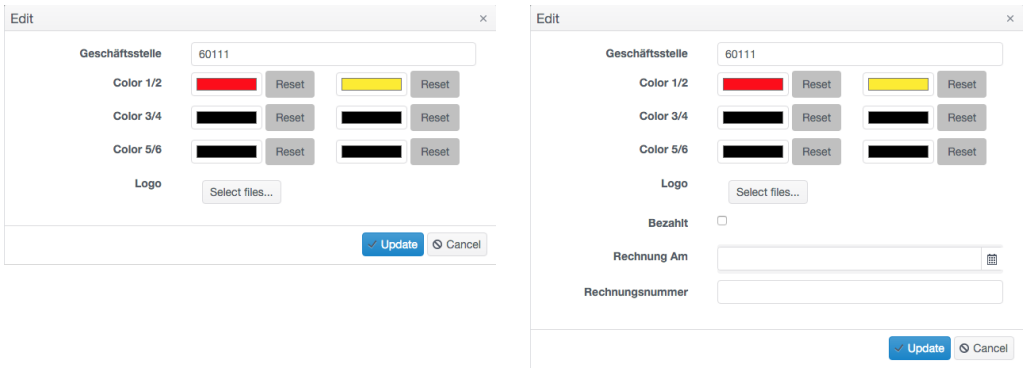
$$QD =_{\text{def}} \{(g, n, i) \mid (g, n) \in B \wedge (g, i) \in D\}$$

Diese Zusammenfassung ermöglicht bei der Rechnungsstellung den Abgleich zwischen den erstellten Designs und den gestellten Rechnungen.

Auswertung:
Der Sicherheitstest brachte 166 qualifizierte Designs mit ihrem Beratern hervor. Beim Abgleich mit den bisher gestellten Rechnungen fielen 43 nicht berechnete Designs auf. Die financeTec stellte daraufhin diese Designs in Rechnung.

Das mit Hilfe von CUSTODIAN ermittelte Delta zwischen Soll und Ist stellte sich größer heraus als erwartet. Deshalb entschied man sich, den Konfigurationsserver zu überarbeiten. Die ursprüngliche Eingabemaske des Konfigurationsserver ist in Abbildung 8.50a dargestellt. In Zukunft besteht ein Design $d \in D$ aus dem Tupel (gd, i, rd, rn, bez) mit der bisher existierenden Geschäftsstellennummer gd und der Individualisierung i. Zusätzlich wurden die Designs um $rd = Rechnungsdatum$, $rn = Rechnungsnummer$ und $bez = bezahlt$ erweitert. Bei Rechnungsstellung werden die Felder rd und rn gesetzt und bei Zahlungseingang wird der boolesche Wert bez auf „wahr“ gesetzt. Der erweiterte Konfigurationsserver ist in Abbildung A.13b dargestellt. Dadurch kann financeTec zukünftig mit Hilfe von CUSTODIAN mittels

einer Übersicht überprüfen, ob und unter welcher Rechnungsnummer die Leistung berechnet wurde und ob eine Bezahlung eingegangen ist.



(a) Ursprüngliche Eingabemaske (b) Erweiterte Eingabemaske

Abb. 8.50: *Verschiedene Versionen des Konfigurationsserver*

8.3.3.3 Berechtigungsannexion

Die Geschäftsstellen in AdWorks sind hierarchisch als Kompositum aufgebaut (siehe Abbildung 8.51). Jede Geschäftsstelle kann weitere Untergeschäftsstellen besitzen. Berater, die einer Geschäftsstelle zugeordnet sind, können somit auch auf ihre Untergeschäftsstellen zugreifen. Dadurch können komplexe Berechtigungsstrukturen entstehen.

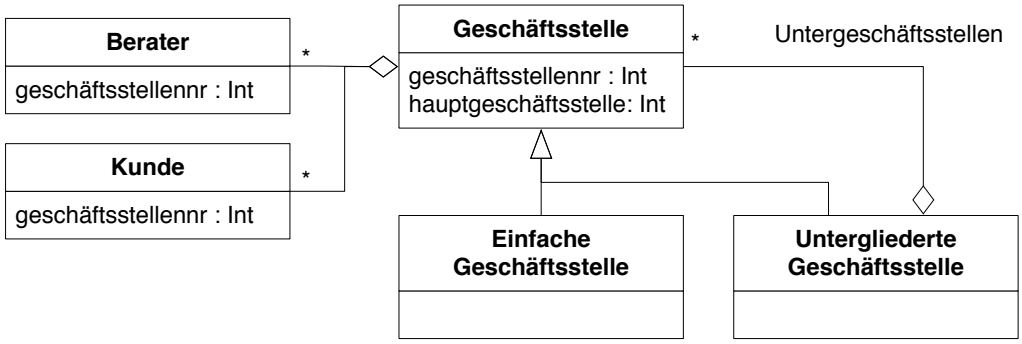


Abb. 8.51: *Modell der Geschäftsstellenhierarchie in AdWorks2go*

Während der Überprüfungen der gestellten Leistungen wurde noch eine sicherheitsrelevante Gefahrenklasse entdeckt. Diese ist auf die hierarchischen Berechtigungsstrukturen der Geschäftsstellen zurückzuführen.

Beispiel:
Ein Berater ist der Geschäftsstelle 99900 zugeordnet. Diese Geschäftsstelle 99900 hat die Untergeschäftsstellen 20111 und 20112, auf die der Berater somit zugreifen kann. Jedoch darf der Berater die Informationen der Geschäftsstelle 20112 nicht einsehen.

#	Geschäftsstelle	Berater
46	99900	Müller, Michael
47	20112	Müller, Michael
48	20111	Müller, Michael

Abb. 8.52: *Mehrfachzuordnung eines Beraters durch die Geschäftsstellenhierarchie in AdWorks2go*

Gefahrenbewertung:
Durch Berechtigungsannexionen können Datenschutzprobleme auftreten, welche die Privatsphäre der Kunden angreifen. Darüberhinaus können wirtschaftliche Interessen anderer Berater verletzt werden.

Der Berechtigungsannexions-Sicherheitstest:
Aus Sicht der Gefahrenklasse „Berechtigungsannexion“ besteht AdWorks2go aus folgenden Grundobjekte:

- Eine Menge G von Geschäftsstellen. Eine Geschäftsstelle $g \in G$ ist ein Paar (n, hn) mit der Geschäftsstellennummer n und der Geschäftsstellennummer der Hauptgeschäftsstelle hn. Wenn es keine Hauptgeschäftsstelle gibt, ist $hn = \epsilon$.
- Eine Menge B von Beratern. Eine Berater $b \in B$ ist ein Paar (gb, i) mit der Geschäftsstellennummer gb und den identifizierenden Namen des Beraters i.

Die rekursive Funktion `ugs` löst alle direkten und indirekten Untergeschäftsstellennummern N der Geschäftsstelle $g \equiv (n, hn)$ auf:

$$N = \text{ugs}((n, hn), G) =_{\text{def}} \{(\text{ugs}(n', hn') \mid (n', hn') \in G \wedge n = hn') \cup \{n\}\}$$

$$N = \text{ugs}((n, hn), G) =_{\text{def}} \bigcup_{(n*,n)\in G} \text{ugs}(n', n) \ \cup \{n\}$$

Die Funktion `gest` löst alle Geschäftsstellen auf, auf die der Berater $b \equiv (gb, i)$ zugreifen kann, und gibt diese als Menge Z der Paare (n, i) zurück:

$$Z = \text{gest}(b, G) \mathrel{\widehat{=}} \text{gest}((gb, i), G) =_{\text{def}} \{(n, i) \mid n \in \text{ugs}(i)\}$$

Die Funktion `gestAll` gibt die Menge Z_{All} der Paare (n, i) für alle Berater zurück:

$$Z_{All} = \text{gestAll}(B, G) =_{\text{def}} \bigcup_{b\in B} \text{gest}(b, G)$$

Auswertung:
Bei der Auswertung der Berater konnten die Mitarbeiter der financeTec bereits drei Berechtigungsannexionen identifizieren. Es ist geplant, für jeden AdWorks2go-Kunden zukünftig einen Report zu erstellen. Dadurch können diese sich selbst einen Überblick über ihre Berechtigungsstrukturen verschaffen.

8.4 Auswertung

Wir führten in verschiedenen natürlichen Umgebungen Fallstudien durch. Das Ziel der Fallstudien war herauszufinden, ob es in existierenden verteilten Systemen Fehler gibt, die zu Gefahren führen können und ob CUSTODIAN die Gefahren mit dem notwendigen Expertenwissen automatisiert erkennen kann. Und ob es Defizite bei der Erkennung der existierenden Gefahren gibt, die mit dem Sicherheitsframework CUSTODIAN im Allgemeinen und mit der Referenzimplementierung KUSTOS im Speziellen behoben werden können (siehe Abbildung 4.1 des Kapitels 4). Darüberhinaus war von Interesse, ob und wie vielseitig CUSTODIAN und KUSTOS einsetzbar sind. Für den Kontext der MTF analysierten wir, ob wir neuartige Gefahrenklassen in verteilten Systemen erkennen können und ob wir Gefahrenklassen, die in der MTF definiert sind, in verteilten Systemen wiederverwendet können.

Abschnitt 8.4.1 stellt den Fragebogen zur Evaluierung der Fallstudien mit seinen Ergebnissen vor. Abschnitt 8.4.2 legt die Entwicklung der MTF während der Fallstudien dar. Eine Übersicht über die entdeckten Gefahren und Gefahrenklassen gibt Abschnitt 8.4.3.

8.4.1 Fragebogen

Nach der Durchführung jeder Fallstudie wurden die in die Analyse der verteilten Systeme einbezogenen Sicherheitsexperten, gebeten einen Fragebogen auszufüllen. Der Fragebogen wurden in jeder Fallstudie von mindestens einer Person ausgefüllt (siehe Abbildung 8.53). Die Referenzimplementierung KUSTOS wurde den Sicherheitsexperten unter dem Namen CUSTODIAN vorgestellt. Er zielt mit den Fragen 1 bis 6 auf die in Abschnitt 4.1 definierten Forschungshypothesen ab. Er validiert, ob die in dieser Dissertation beschriebenen Beobachtungen auch die Erfahrungen der Sicherheitsexperten widerspiegeln. Eine Gegenüberstellung der Forschungshypothesen und der Fragen des Fragebogens ist in Tabelle 8.8 aufgezeigt.

	Frage		Hypothese
1	In dem verteilten System existieren fehlerhafte Konfigurationen.	H_{1a}	Verteilte Systeme bergen fehlerhafte Konfigurationen.
2	Fehlerhafte Konfigurationen können das verteilten System gefährden.	H_{1b}	Fehlerhafte Konfigurationen verteilter System können diese gefährden.
3	Für die Erkennung von fehlerhaften Konfigurationen sind Spezialisten notwendig.	H_{2a}	Die Erkennung vor Gefahren in verteilte Systemen setzt Gefahrenwissen zu Komponenten und den Gefahren voraus.
4	Die Expertise der Spezialisten konnte durch CUSTODIAN automatisiert werden.	H_{2b}	Gefahrenwissen zur Erkennung von Gefahren in verteilten Systeme kann externalisiert und in ein spezialisiertes System überführt werden.
5	Bei der Arbeit mit CUSTODIAN wurden neue Gefahrenklassen identifiziert.	H_{3a}	In verteilten Systemen existieren unerkannte Fehlkonfigurationen.
6	Bei der Arbeit mit CUSTODIAN wurden bisher unentdeckte Gefahren erkannt.	H_{3b}	Der Einsatz der Blackboard-Architektur unterstützt bei der Erkennung von fehlerhaften Konfigurationen eines verteilten Systems.
	(eine äquivalente Frage)	H_{3c}	Eine Lösung auf Basis der Blackboard-Architektur kann bei mehreren, verschiedenartigen verteilten Systemen zur Erkennung von fehlerhaften Konfigurationen eingesetzt werden.

Tab. 8.8: *Gegenüberstellung der Forschungshypothesen mit den Fragen*

Evaluation des CUSTODIAN-Frameworks

Name: _____

Aufgabe: _____

Verteiltes System: _____

Bitte beantworten Sie folgende Fragen zu dem untersuchten verteilten System.

1. In dem verteilten System existieren fehlerhafte Konfigurationen.
trifft überhaupt nicht zu ☐ −2 ☐ −1 ☐ 0 ☐ +1 ☐ +2 trifft voll und ganz zu

2. Fehlerhafte Konfigurationen können das verteilten System gefährden.
trifft überhaupt nicht zu ☐ −2 ☐ −1 ☐ 0 ☐ +1 ☐ +2 trifft voll und ganz zu

3. Für die Erkennung von fehlerhaften Konfigurationen sind Spezialisten notwendig.
trifft überhaupt nicht zu ☐ −2 ☐ −1 ☐ 0 ☐ +1 ☐ +2 trifft voll und ganz zu

4. Die Expertise der Spezialisten konnte durch CUSTODIAN automatisiert werden.
trifft überhaupt nicht zu ☐ −2 ☐ −1 ☐ 0 ☐ +1 ☐ +2 trifft voll und ganz zu

5. Bei der Arbeit mit CUSTODIAN wurden neue Gefahrenklassen identifiziert.
trifft überhaupt nicht zu ☐ −2 ☐ −1 ☐ 0 ☐ +1 ☐ +2 trifft voll und ganz zu

6. Bei der Arbeit mit CUSTODIAN wurden bisher unentdeckte Gefahren erkannt.
trifft überhaupt nicht zu ☐ −2 ☐ −1 ☐ 0 ☐ +1 ☐ +2 trifft voll und ganz zu

7. Wie hat Ihnen CUSTODIAN bei Ihrer Arbeit geholfen?

_____

_____

_____

8. Welche Verbesserungsvorschläge haben Sie für CUSTODIAN?

_____

_____

_____

Abb. 8.53: *Evaluationsbogen für die Evaluation des CUSTODIAN-Frameworks*

Bei den Fragen 1 bis 6 kam eine fünfgliedrige Likert-Skala zum Einsatz, bei der der Befragte jeweils fünf Antwortmöglichkeiten zwischen „trifft überhaupt nicht zu“ (-2) bis „trifft voll und ganz zu“ (+2) hat [Lik32]. Diese ersten sechs Fragen dienen der Validierung der Ergebnisse der Fallstudien. Somit sind diese der zentrale Teil des Fragebogens. Die Fragen 7 und 8 sind offen formuliert und geben eine Übersicht über die Zufriedenheit und die Verbesserungsvorschläge der Sicherheitsexperten als Benutzer von KUSTOS.

Gefährdung der Validität (threats to validity):
Die im Folgenden beschriebenen Verzerrungseffekte, welche die Gültigkeit der Ergebnisse des Fragebogens gefährden können, beziehen sich auf die Antworten der Fragen 1 bis 6.

1. **Geringe statistische Signifikanz**. Die Anzahl von sechs Befragten ist nicht ausreichend, um statistisch signifikante Aussagen abzuleiten. Die Ergebnisse belegen die durchgeführten Fallstudien lediglich mit anekdotischer Evidenz [Hoe01].

2. **Unbewusste Empathiediskrepanz** (engl. empathy gap), Empathie zu anderen Personen kann sich auf die Objektivität der Befragten auswirken [GI12]. Da mich jede der befragten Personen kannte, ist es möglich, dass Sympathie zu mir oder Antipathie mir gegenüber die Befragten unbewusst beeinflusst haben.

3. **Bewusste Empathiediskrepanz**. Befragte Personen können auch bewusst falsch antworten. Eine bewusste Beeinflussung ist ebenfalls möglich, denn welche Antworten für meine Arbeit positiv sind, ist wahrscheinlich für die Befragten erkennbar.

4. **Selbstschutztendenz**. Antworten, welche die Befragten selbst oder ihre Kollegen kompromittieren, können vermieden werden. Da die befragten Personen sicherheitsverantwortlich sind, kann es möglich sein, dass sie beispielsweise nicht zugeben wollen, dass es Gefahren gibt oder dass sie diese nicht kannten.

5. **Zentraltendenz** (engl. central tendency). Bei Skalen wie der Likert-Skala tendieren die Befragten oft dazu, Werte nahe des Zentrums zu wählen [Hol10]. Somit ist es möglich, dass die tatsächlichen Ergebnisse extremer ausgefallen wären.

6. **Beeinflussung durch Erklärung**. Während des Ausfüllens der Fragebögen stand ich den Befragten für Rückfragen zur Verfügung. Es ist möglich, dass meine Antworten auf die gestellten Rückfragen einen manipulierenden Einfluss hatten.

7. **Variationstendenz**. Bei drei Befragten fällt auf, dass die Antworten, abgesehen von einer Variation, immer identisch sind. Es könnte möglich sein, dass die Befragten eine durchgehend identische Bewertung für undifferenziert halten und diese (bewusst oder unbewusst) verhindert haben.

Die Fragebögen wurden insgesamt von sechs Personen beantwortet. Dies waren vier Sicherheitsverantwortliche der HSRZ-Fallstudie, ein Sicherheitsverantwortlicher der RBG-Fallstudie und ein Sicherheitsverantwortlicher der AdWorks2go-Fallstudie.

		Frage						
Fallstudie	Position	1	2	3	4	5	6	∅
HSRZ	Head of Datacenter Security	+1	+2	+1	+1	+1	+2	+1,33
HSRZ	Verantwortlicher Interne Audits	+2	+1	+2	+2	+2	+2	+1,83
HSRZ	Head of Datacenter Security	+2	+2	+1	+2	+2	+2	+1,83
HSRZ	Leitung Administration Firewalls	+2	+2	+1	+2	+2	+2	+1,83
AdWorks2go	Chief Executive Officer	+2	+2	+1	+2	+1	+2	+1,67
RBG	Leitung Netzwerkgruppe	+2	+1	+2	+2	-2	+2	+1,17

Tab. 8.9: *Antworten der einzelnen Umfrageteilnehmer*

Die nahezu durchgehend positiven Bewertungen stärken die Forschungshypothesen dieser Dissertation. Die einzig negative Bewertung gab der Leiter der RBG-Netzwerkgruppe bei Frage 5 ab. Dieses Phänomen ist durch die Eigenschaft der RBG-Fallstudie zu erklären. In der RBG waren die von den Firewall- und Hostdatenbank-Konfigurationen ausgehenden Gefahrenklassen bereits bekannt. Es ging bei RBG-Fallstudie vielmehr darum, die Administratoren der einzelnen Organisationseinheiten über mögliche Gefahren aufzuklären. Das gelang, die sehr positive Antwort des Leiters der RBG-Netzwerkgruppe auf Frage 6 ist darauf zurückzuführen, dass in der RBG viele Änderungen an den Firewallkonfigurationen aufgrund der Analyse beantragt wurden.

#	Frage	∅
1	In dem verteilten System existieren fehlerhafte Konfigurationen.	+ 1,83
2	Fehlerhafte Konfigurationen können das verteilten System gefährden.	+ 1,67
3	Für die Erkennung von fehlerhaften Konfigurationen sind Spezialisten notwendig.	+ 1,33
4	Die Expertise der Spezialisten konnte durch CUSTODIAN automatisiert werden.	+ 1,83
5	Bei der Arbeit mit CUSTODIAN wurden neue Gefahrenklassen identifiziert.	+ 1,00
6	Bei der Arbeit mit CUSTODIAN wurden bisher unentdeckte Gefahren erkannt.	+ 1,83
∅	Durchschnitt	+ 1,61

Tab. 8.10: *Auswertung der Fragen 1 bis 6 des Fragebogens*

8.4.2 Evolution der MTF

Zwischen den Fallstudien und der MTF herrschte eine iterative Beziehung: Einerseits wurde das der Fallstudie zugrundeliegende verteilte System bezüglich der in der MTF definierten Gefahrenklassen analysiert, und die Fallstudie profitiert vom Wissen der MTF. Andererseits wurden neuartige Gefahrenklassen, die während der Analyse des verteilten Systems entdeckt werden, in die MTF aufgenommen, und die MTF profitiert von den Erkenntnissen, die während der Fallstudie gewonnen werden. Ausgangspunkt für die drei durchgeführten Fallstudien war der Kern der MTF, wie er in Abbildung 8.54 dargestellt ist.

Abb. 8.54: *MTF 1.0: Der Kern der Meta Threat Facility*

Im Folgenden ist aufgezeigt, wie sich die MTF während der Fallstudien entwickelt hat. Dabei werden die in Abbildung 8.55 vorgestellten Farbnotationen eingesetzt. Existierende Gefahrenklassen der MTF, die jedoch in der konkret untersuchten Fallstudie nicht wiederverwendet werden konnten, sind schwarz gekennzeichnet. Gefahrenklassen, die in der Fallstudie neu entdeckt und in die MTF aufgenommen wurden, sind grün gekennzeichnet. Kamen existierende Gefahrenklassen der MTF in einer Fallstudie wieder zum Einsatz, so sind diese violett gekennzeichnet.

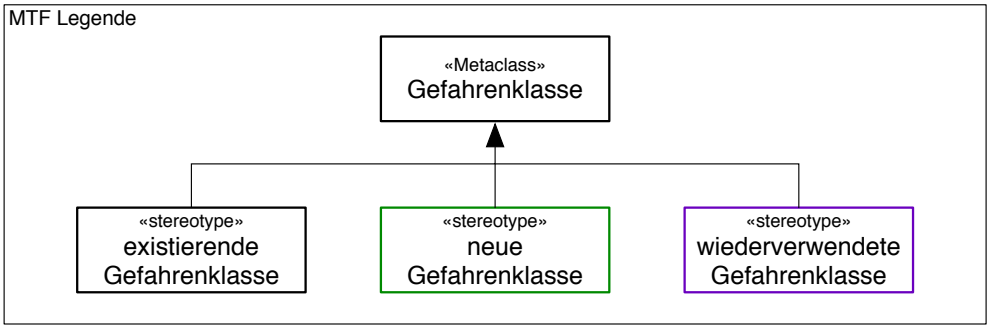


Abb. 8.55: *MTF Legende: Farbnotationen für Gefahrenklassen der MTF*

Vor der ersten durchgeführten Fallstudie im HSRZ existierte nur der MTF-Kern, und es konnten somit keine Gefahrenklassen wiederverwendet werden. Während der Fallstudie wurden insgesamt sieben Gefahrenklassen neu erkannt und in die MTF überführt. Es handelt sich um die Gefahrenklassen Berechtigungsannexion, Witwe, Waise, Zombie, Trojanischer Server, Berechtigungsauslauf und Abgebauter Server, die in Abschnitt 8.1 beschrieben sind. Die dadurch entstandene MTF 1.1 ist in Abbildung 8.56 dargestellt.

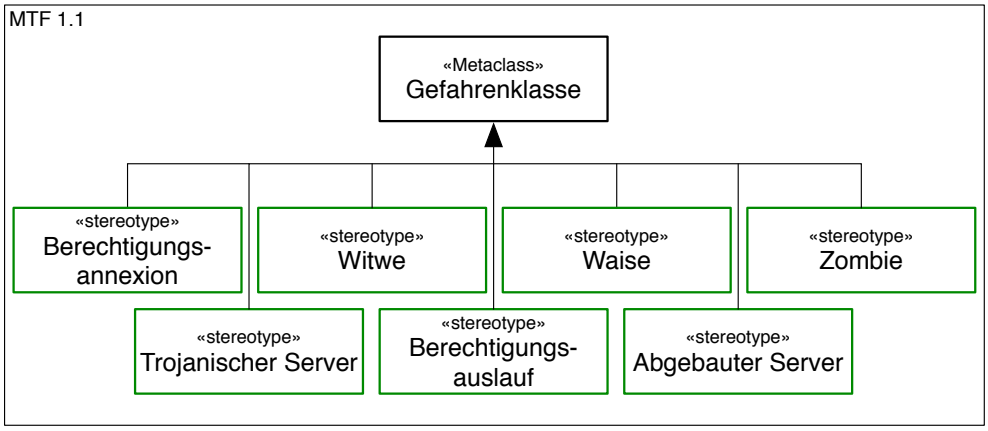


Abb. 8.56: *MTF 1.1: MTF nach der HSRZ-Fallstudie*

In der zweiten, bei der RBG durchgeführten, Fallstudie (siehe Abschnitt 8.2) existierten bereits 7 konkrete Gefahrenklassen in der MTF. Somit war es möglich, dass bestehende MTF-Gefahrenklassen auch im untersuchten verteilten System vorkamen. Dies war der Fall, denn die Gefahrenklasse „Abgebauter Server“, die bei der HSRZ-Fallstudie entdeckt wurde, kam auch im RBG-Netzwerk vor. Zusätzlich wurde die neue Gefahrenklasse „Generalisierung“ entdeckt und in die MTF aufgenommen. Daraus ergab sich die neue Version 1.2 der MTF, welche in Abbildung 8.57 dargestellt ist.

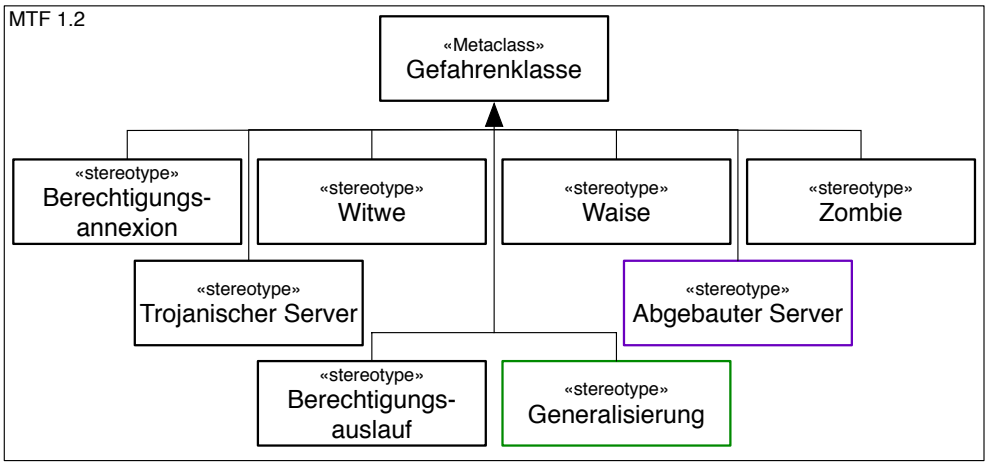


Abb. 8.57: *MTF 1.2: MTF nach der RBG-Fallstudie*

Bei der AdWorks2go-Fallstudie bestand die MTF somit aus acht Gefahrenklassen. Aufgrund der unterschiedlichen fachlichen Domäne, in der die Fallstudie durchgeführt wurde, war zu erwarten, dass potenziell neuartige Gefahrenklassen auftreten. Das war mit den neuen Gefahrenklassen „Unbezahlter Onlinezugriff“ und „Unbezahltes Design“ auch der Fall. Die beiden Gefahrenklassen erweiterten die MTF, die somit in der Version 1.3 besteht (siehe Abbildung 8.58). Auch in dieser Fallstudie wurde eine MTF-Gefahrenklasse im verteilten System identifiziert. Die Gefahrenklasse „Berechtigungsannexion“, welche aus der HSRZ-Fallstudie herrührt, tritt auch im verteilten AdWorks2go-System auf.

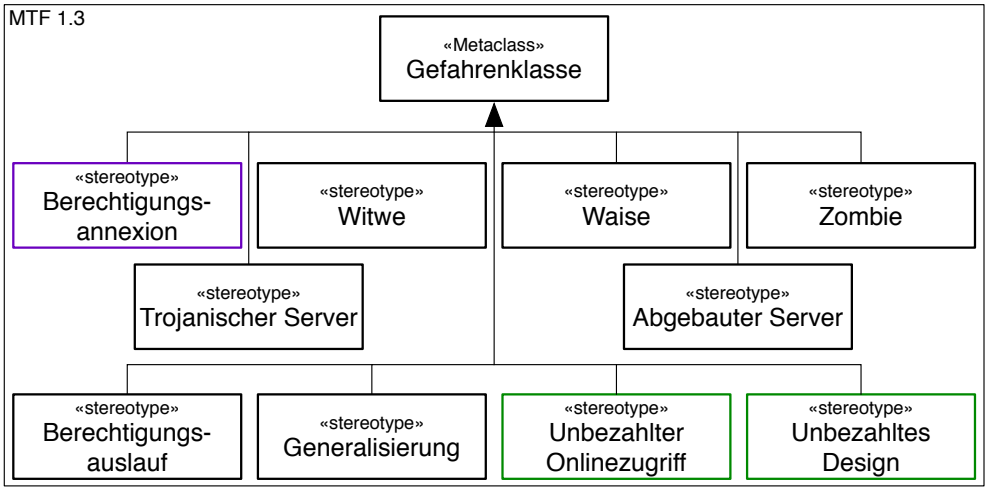


Abb. 8.58: *MTF 1.3: MTF nach der AdWorks2go-Fallstudie*

8.4.3 Die entdeckten Gefahrenklassen und Gefahren

Die Tabelle 8.11 gibt einen Überblick über die Gefahrenklassen aller Fallstudien. Beim HSRZ war sowohl Anzahl der Gefahrenklassen als auch die Anzahl der gefundenen Gefahren am höchsten. Die Sicherheitsverantwortlichen beseitigten mit Hilfe von KUSTOS bei den meisten Gefahrenklassen nahezu alle Gefahren. Nur bei den Gefahrenklassen Witwe und Waise konnten die Anzahl der Gefahren während der Fallstudie nicht weitgehend eingeschränkt werden. Durch die geplante Umstellung des Berechtigungsprozesses werden diese beiden Gefahrenklassen laut Einschätzung der Sicherheitsverantwortlichen komplett beseitigt.

Gefahrenklasse	Typ	ursprüngliche Gefahren	beseitigte Gefahren	Anteil beseitigter Gefahren
Fallstudie HSRZ				
Berechtigungsauslauf	neu	19.407	19.231	99 %
Berechtigungsannexion	neu	18	18	100 %
Trojanischer Server	neu	2.917	2.812	98 %
Abgebauter Server	neu	135	135	100 %
Witwe	neu	645	319 (645[10])	49 % / (100 %)
Waise	neu	1.336	0 (1.336[10])	0 % / (100 %)
Zombie	neu	379	379	100 %
Fallstudie RBG-Netzwerk				
Abgebauter Server	MTF	308[11]	308	100 %
Generalisierung	neu	25[11]	25	100 %
Fallstudie AdWorks2go				
Berechtigungsannexion	MTF	3	3	100 %
Unbezahlter Onlinezugriff	neu	43	43	100 %
Trojanischer Server	neu	3	3	100 %

Tab. 8.11: *Überblick über die Gefahrenklassen und Gefahren aller Fallstudien*

Die Tabelle gibt über die Gefahren im RBG-Netzwerk keinen vollständigen Überblick. Einerseits wurden während der Fallstudie nur die Lehrstühle mit besonders vielen Firewallregeln kontaktiert. Andererseits beurteilten nicht alle kontaktierten Sicherheitsverantwortlichen während der Fallstudie die

[10] Die erste Zahl beziffert die bis zum Abschluss der Fallstudie beseitigten Gefahren. Die zweite Zahl beziffert die voraussichtliche Anzahl der nach der Umstellung des Berechtigungsprozesses

[11] Berücksichtigt nur die von den Lehrstühle als gefährlich bestätigten Firewallregeln

Gefahrenhypothesen. Somit beschränkt sich die Anzahl der in der Tabelle ausgewerteten Daten auf die Firewallregeln von zwei Lehrstühlen. Auch nach dem Abschluss der Fallstudie werde ich in Zusammenarbeit mit der RBG auf die Lehrstühle zugehen um gemeinsam mit ihnen die Gefahrenhypothesen zu evaluieren und tatsächliche Gefahren zu eliminieren.

Bei der AdWorks2go-System entdeckte KUSTOS am wenigsten Gefahren, die jedoch vollständig beseitigt wurden. Bei der erwarteten zukünftigen Steigerung der Kundenzahl wird für die financeTec AG auch der von KUSTOS ausgehende Nutzen steigen.

9 Fazit und Ausblick

Diese Dissertation richtet sich an die Betreiber komplexer verteilter Systeme. Sie adressiert das Problem, dass in diesen verteilten Systemen Fehlkonfigurationen aufgrund der Verteilung der Konfigurationen schwer zu erkennen sind. Da diese Fehlkonfigurationen zu Gefahren führen, ist deren Erkennung wichtig. Kapitel 6 stellt das Sicherheitsframework CUSTODIAN vor, dass die Teilkonfigurationen eines verteilten Systems zusammengeführt. CUSTODIAN erlaubt die ganzheitliche Betrachtung der Konfigurationen eines verteilten Systems Das ermöglicht die Identifikation von Fehler respektive von Gefahren. Architektonisch basiert das Framework CUSTODIAN auf dem Blackboard-Muster, auf dessen Sicherheitsblackboard unterschiedliche Wissensquellen arbeiten. Die Wissensquellen stellen im Prozess der Gefahrenerkennung das notwendige Wissen bereit. Wissen um beispielsweise die Konfigurationen der einzelnen Komponenten korrekt auszulesen und zu interpretieren, um bestimmte Gefahren in der Gesamtkonfiguration zu erkennen oder um die erkannten Gefahren aussagekräftig zu kommunizieren. Die Erweiterbarkeit der Architektur erlaubt die Reaktion auf strukturelle Änderungen eines verteilten Systems sowie die Adaption auf neuartige verteilte Systeme. Neben dem Sicherheitsframework CUSTODIAN stellt diese Dissertation das erweiterbare Gefahrenmodell MTF vor. Die MTF ist ein MOF-konformes UML-Profil, das für alle bisher identifizierten Gefahrenklassen ein Stereotyp definiert. Eine Gefahrenklasse ist eine Abstraktion, welche eine Menge von gleichartigen Gefahren beschreibt. Dadurch stellt die MTF sogleich eine Wissensdatenbank über die bekannten Gefahrenklassen dar und erlaubt es, diese Gefahrenklassen bei der Modellierung in UML einzusetzen.

Im Rahmen einer Evaluation erkannte CUSTODIAN zehn verschiedene Gefahrenklassen in drei unterschiedlichen verteilten Systemen. Bei zwei verteilten Systemen unterstützte die MTF bei der Erkennung von Gefahrenklassen, indem eine in der MTF definierte Gefahrenklasse wiedererkannt wurden. CUSTODIAN unterstützte nach dem Erkennen der Gefahrenklasse auch bei der Identifikation ihrer Gefahren. Dabei wurden bei einer Gefahrenklasse über 19.000 Gefahren identifiziert. Mit diesen erzeugten Wissen

war es den Betreibern der verteilten Systeme möglich, auf die Gefahren zu reagieren. Die Betreiber waren somit in der Lage, bei allen identifizierten Gefahrenklassen die erkannten Gefahren entweder erheblich zu reduzieren oder sogar vollständig zu eliminieren.

Diese Dissertation untersuchte die Frage, ob die Blackboard-Architektur bei Identifikation von Gefahrenklassen und Gefahren in verteilten Systemen hilfreich ist. Sie legt in verschiedenen Bereichen einen Grundstein für weitere mögliche Entwicklungs- und Forschungsatkivitäten. Noch offene Fragen zu der im Kapitel 5 vorgestellten MTF werden in Abschnitt 9.1 erörtert. Der Abschnitt 9.2 geht auf mögliche Weiterentwicklungen und Evaluierungen des Sicherheitsframeworks CUSTODIAN ein.

9.1 Meta Threat Facility

Die MTF definiert einen Katalog für Gefahrenklassen. Betreiber verteilter Systeme können sich diesen Katalog zu Nutzen machen indem sie ihr verteiltes System auf die im Katalog beschriebenen Gefahrenklassen untersuchen. Dadurch können sie in ihrem verteilten System Gefahren aufdecken, die sie sonst eventuell erst später oder überhaupt nicht gefunden hätten. Um diesen Nutzen zu maximieren, muss die MTF möglichst viele Gefahrenklassen beinhalten und stets aktuell gehalten werden. Deshalb wäre es sinnvoll, eine Community zu gründen, welche die MTF nutzt und aktiv zu ihrer Weiterentwicklung beiträgt. Die MTF könnte über eine kollaboratives Informationssystem im Stil von Wikipedia oder GitHub publiziert und weiterentwickelt werden.

Eine weitere Möglichkeit, die MTF zu erweitern, ist die Applikation bereits etablierter Disziplinen der Gefahrenerkennung. Hierbei können die Prüf- und Zielvorgaben von IT-Sicherheitszertifizierungen für Unternehmen der Ausgangspunkt neuer Gefahrenklassen sein. Die internationale verbreitete Norm ISO/IEC 27001 [iso05] definiert beispielsweise einen IT-Grundschutz-Katalogen von Standard-Sicherheitsmaßnahmen. Jede der Sicherheitsmaßnahme beschreibt eine Gefahrenklasse, die durch ihre Nichteinhaltung definiert ist. Die Auditierungssstandards SAS 70 [sas92] und SSAE 16 [ssa10] werden beispielsweise verwendet, um einen IT-Dienstleister auf die Kon-

formität zum Sarbanes-Oxley Act [SO02], der internationale Bedeutung genießt, zu überprüfen.

9.2 CUSTODIAN

9.2.1 Evaluierung der Einsetzbarkeit

Kapitel 4 definiert sowohl eine schwache als auch eine starke Portabilitätshypothese. Durch die Evaluation wird jedoch nur die schwache Portabilitätshypothese belegt. Mit einer geeigneten Auswahl von Fallstudien wäre es eventuell möglich auch die starke Portabilitätshypothese mit aussagekräftigen Evidenz zu belegen. Hierfür bieten sich Fallstudien in den Bereichen an, die aktuell im Fokus wissenschaftlicher Entwicklungen stehen. Sowohl das IoT als auch Cyber-physical system (CPS) stellen somit mögliche Schwerpunkte für Fallstudien dar. Weitere Fallstudien können jedoch auch zur Invalidierung die starke Portabilitätshypothese führen. Hierfür müsste ein komplexes verteiltes System gefunden werden, in dem durch die von CUSTODIAN eingesetzte Blackboard-Architektur keine Gefahren gefunden werden können. In diesem Fall ist es von Interesse, Diskriminatoren zu beschreiben, die bestimmen, ob der Einsatz von CUSTODIAN sinnvoll ist.

9.2.2 Erkennung von Beta-Fehlern durch Fehlerinjektion

> *„Program testing can be used to show the presence of bugs, but never to show their absence!“*
>
> – Edsger Wybe Dijkstra [Dij72]

Dijkstra stellte 1972 heraus, dass Programmtests lediglich die Existenz von Fehlern, jedoch nicht Abwesenheit beweisen können. Das Programm einer Komponente setzt sich aus zwei Teilen zusammen, der Fireware und der Konfiguration. Die Fireware ist der vom Hersteller deklarierte Teil und die Konfiguration der vom Anwender deklarierte Teil. Gemeinsam bestimmen sie auf der Basis der Hardware einer Komponente das Verhalten der Komponente (siehe Abschnitt 2.2). Nita und Notkin beschreiben diese Zweitei-

lung folgendermaßen: „Roughly defined, a configurable software system is a shared code base from which a set of executable configurations can be produced. The system is characterized by a set of configuration options, each of which represents a dimension that can be configured“ [NN09]. Somit gilt nach Dijkstras Aussage auch eine Konfiguration als Programm.

Im Allgemeinen sucht man mit Tests nach Defekten, im Rahmen dieser Dissertation nach Fehlkonfigurationen (siehe Abschnitt 2.4). Das Auffinden von Fehlkonfigurationen gibt jedoch keinen direkten Hinweis auf die Menge der noch noch existierenden Fehler. Dieses Ergebnis bleiben Tests oft schuldig, denn die nicht gefundenen Fehler sind nicht nur qualitativ sondern auch quantitativ noch unbekannt. Die Fehlerinjektion ist eine Technik, die es erlaubt, Hinweise auf die Effizienz eines Tests zu gewinnen [HTI97, ZHM97]. Hierbei werden vor dem Test gezielt Fehler im getesteten Objekt platziert. Durch das Vergleichen der platzierten und der aufgedeckten Fehler kann man die Aufdeckungsrate ermitteln. Unter der Annahme, dass die so ermittelte Aufdeckungsrate repräsentativ für die Aufdeckungsrate aller Fehler im System ist, kann man Schlussfolgerungen über die noch im System verbleibenden Fehler treffen.

Das Prinzip an sich ist auch für die Gefahrenerkennung in verteilten Systemen übertragbar. Man kann es einsetzen, um den Anteil der Gefahren, der durch eine Untersuchungsmethode entdeckt wird, zu prognostizieren. Dazu müssen bewusst gefährliche Konfigurationen in das verteilte System injiziert werden. Der Prozentsatz der durch eine Untersuchungsmethode aufgedeckten Gefahren wird dann als repräsentativ angesehen.

Die Schwäche der Technik Fehlerinjektion liegt in der Tatsache, dass die injizierten Fehler bzw. Gefahren als repräsentativ für tatsächlich aufzudeckenden Fehler angesehen werden. Es ist jedoch nicht erwiesen, dass ein Test nicht über- oder unterproportional viele der injizierten Fehler aufdeckt. Das beeinfluss die Korrektheit der Rückschlüsse, welche auf die aufzudeckenden Fehler gezogen werden. Trotz dieser Problematik wird dieses Prinzip in der Softwarentwicklung [HTI97, Voa97] und in der Hardwareentwicklung [GKT89] verfolgt. Basierend auf dem Sicherheitsframework CUSTODIAN könnte der Ansatz der Fehlerinjektion eingesetzt werden. Eine offene Frage ist, wie es möglich ist, in eine explizite Testumgebung Fehler zu injizieren, ohne dabei das System zu gefährden.

9.2.3 Einbeziehung von Akteuren in die Konfiguration

CUSTODIAN zielt auf die Gefahrenerkennung in verteilten Systemen. Jedoch können bei weitem nicht alle Gefahren mit CUSTODIAN erkannt werden. Whistleblower, wie beispielsweise Edward Snowden, können beispielsweise nicht identifiziert werden. Das liegt unter anderem daran, dass CUSTODIAN die Benutzer eines verteilten Systems weder als Teil der Konfiguration noch als Teil des verteilten System selbst sieht. Jedoch sind die Nutzer ein wesentlicher Sicherheitsfaktor und sollten bei der Evaluation von Sicherheit stets mit in Betracht gezogen werden. Deshalb ist es von wissenschaftlichem Interesse, wie der Benutzer als Teil der Konfiguration eines verteilten Systems verstanden werden kann, um ihn in die Sicherheitsbewertung mit einzubeziehen.

9.2.4 Automatisierte Erkennung der Gefahrenklassen

Auf der Basis der MTF können Sicherheitsverantwortliche ihr verteiltes System auf bekannte Gefahrenklassen untersuchen. Dieser Prozess ist jedoch mit manuellen Aufwand verbunden. Je mehr Gefahrenklassen die MTF umfasst, desto länger und teurer wird diese Analyse. Eine automatisierte Erkennung der Gefahrenklassen ist erstrebenswert. Ein Ansatz, eine solche automatisierte Erkennung zu ermöglichen, basiert auf einem abstrakten Modell für verteilte Systeme auf dessen Basis die Gefahrenklassen in einer maschinenlesbaren Weise definiert werden. Jedes untersuchte verteilte System müsste danach auf das abstrakte Modell abgebildet werden. Diese Abbildung zwischen dem abstrakten Gefahrenmodell und den Modellen der verteilten Systeme könnte beispielsweise über eine ontologiebasierte Modellintegration erreicht werden wie Glas diese in seiner Dissertation vorstellt [Gla13].

A Appendix

Der Appendix ist in zwei Abschnitte untergliedert. In Abschnitt A.1 wird, aufgrund der Relevanz der Fallstudien HSRZ und RBG auf grundlegende Aspekte der IT-Sicherheit eingegangen. In Abschnitt A.2 sind weitere Details zu den in den Fallstudien vorkommenden Komponenten beschrieben.

A.1 IT-Sicherheit

A.1.1 Intrusion Detection System

Ein IDS (dt. Angriffserkennungssystem) ist ein System, dass darauf spezialisiert ist, Angriffe auf Systeme oder Netzwerke zu erkennen. Die verschiedenen Arten von IDS sowie deren historische Entwicklung stellt Vigna in seiner Arbeit vor [VK02]. Neben diesen klassischen Erkennungssystemen existieren auch Lösungen, welche bei erkannten Angriffen selbständig Gegenmaßnahmen einleiten. Diese Systeme heißen Intrusion Prevention System (IPS). Ierace beschreibt den Ansatz eines IPS's und zeigt, wie es Systeme wirkungsvoll schützen kann [IUB05]. Vulnerability-Scanner können mögliche Angriffsziele eines Systems identifizieren [KKKJ06].

A.1.2 Autorisierung und Authentifizierung

Die kontextsensitive Authentifizierung stellt einen Forschungsbereich mit vielen verschiedenen Ansätzen dar. Day beschreibt [Dey01] bereits 2001, was unter dem Kontext zu verstehen ist und wie er von IT-Systemen genutzt werden kann. Einige der Ansätze sind aus dem Blickwinkel der IT-Sicherheit, andere aus dem Blickwinkel der Usability Engineerings motiviert.

Ein vor allem bei Betriebssystemen verbreiteter Ansatz für die Autorisierung stellt die Access Controll List (ACL) dar. Eine ACL besteht aus einer Liste von Tripeln (r, b, a) mit der Ressource r, auf die berechtigt wird, einem Benutzer b, der die Berechtigung erhält und einer Aktion a, die ausgeführt werden darf.

Lampson stellt mit der Zugriffsmatrix eine weitere Methode zur Autorisierung dar [Lam74]. Mit ihr kann definiert werden, welche Aktionen A die Zugreifer Z auf die Ressourcen R haben. Die Zugriffsmatrix definiert eine zweidimensionale Matrix, in er die erste Dimension durch die Zugreifer $z \in Z$ und die zweite Dimension durch die Ressourcen $r \in R$ definiert ist. Die Werte der Matrix werden durch die Aktionen $a \in A$ belegt, die der definierte Zugreifer z auf der definierten Ressource r ausüben darf.

Ein weiteres, mächtigeres Konzept der Autorisierung ist Role-Based Access Control RBAC, welches Ferraiolo und Kuhn beschreiben [FK92]. Bei RBAC werden die Berechtigungen nicht direkt den Personen, sondern den Rollen zugewiesen. Diese Rollen stehen somit mediativ zwischen den Berechtigungen und Berechtigten. Eine Rolle definiert somit ein Profil von Berechtigungen, welches mehreren Personen zugewiesen werden kann. RBAC ermöglicht es, in komplexen Berechtiungsstrukturen übersichtlicher darzustellen als beispielsweise ACLs das können.

Das Konzept der Generalized Role-Based Access Control (GRBAC) [CMA00, MA01] erweitert RBAC um das Umfeld, in dem der Zugang gewährt werden soll. Damit erlaubt GRBAC eine kontextsensitive Definition von Rechten, bei der beispielsweise Zeit oder andere Prozesse mit einbezogen werden können [CLS+01]. Basierend auf GRBAC wird ein Architekturmodell vorgestellt [CFZA02], welches dabei hilft, Services sicher und kontextabhängig zur Verfügung zu stellen.

SESAME wird als dynamischer, kontextsensitiver Kontrollmechanismus großer vernetzter Applikationen vorgestellt [ZP03]. Es erweitert bestehende Authentifizierungsmechanismen um den aktuellen Kontext des Benutzers. Dies erlaubt es, dem Benutzer kontextabhängig Rechte zu gewähren und zu entziehen. Es basiert auf RBAC und erweitert dies um den Kontext des Benutzers wie es bereits von RBAC's Schöpfern vorgeschlagen wurde [FSG+01]. Es wird als dynamic role based access control DRBAC vorgestellt.

Wullems et al. [WLC04] stellen eine sicherheitsgetriebene Architektur vor, die in bestehende Netzwerke integriert werden kann. Sie zeigen auf, wie der

Kontext bei der Instanziierung einer Autorisierung herangezogen werden kann. Dies ermöglicht beispielsweise, den Aufenthaltsort des Ausübenden, die genutzten Hardware und Software und die Art der Netzwerkverbindung bei der Autorisierung mit zu berücksichtigen.

Masoumzadeh et al. [MAJ06, MAJ07] zeigen auf, dass es zur fachgerechten Bewertung der Kontexte verschiedenen Experten benötigt. Ihr Ansatz löst die Konflikte zwischen den eingebundenen Experten und hilft, eine Entscheidung auf Basis des gesamten zur Verfügung stehenden Wissens herbeizuführen.

A.1.3 Firewalls

Jede Firewall stellt ein Filter dar, der den Datenverkehr in einem System auf Basis gewisse Kriterien zulässt. Firewalls, die den Datenverkehr in einem Netzwerk überprüfen, werden Hardware-Firewall genannt. Eine Firewall, welche direkt auf einem Rechner installiert ist und den ein- und ausgehenden Datenverkehr dieses Rechners überprüft, ist eine Personal-Firewall. Die Firewalls unterscheiden sich nach der Art der Kriterien, nach denen sie filtern [Abi00].

Es gibt eine Reihe von Technologien nach denen Firewalls arbeiten und unerwünschten Datenverkehr herausfiltern [SH09]. Mehrere dieser Technologien können in einer Firewall zum Einsatz kommen.

Paketfilterung ermöglicht es die Herkunfts- und Zieladresse sowie Netzwerkprotokoll und Netzwerkport als Filterkriterien heranzuziehen. Bei der zustandsorientierten Paketüberprüfung (Stateful Packet Inspection) wird der Zustand der Verbindungen von der Firewall gespeichert und in die Analyse mit einbezogen.
Applikations-Firewalls sind zusätzlich in der Lage, den Inhalt der Pakete zu bewerten. Einerseits können erlaubte Inhalte definiert werden, andererseits ist die Firewall auch in der Lage zu überprüfen, ob der Inhalt für den verwendeten Port typisch ist.
Applikations-Proxys agieren als Agent zwischen den Systemen. Somit ist die Adresse der Systeme für die jeweiligen Gegenseite nicht sichtbar.
Virtual Private Networks erfordern eine Authentifizierung an der Fire-

wall, um auf das Netzwerk hinter der Firewall zugreifen zu können.
Network Access Control überprüft bei eingehenden Verbindungen den zugreifenden Klienten auf Gefahren.
Next Generation Firewalls oder auch User Based Firewalls können anhand der Identität erkennen, ob Datenverkehr erwünscht oder unerwünscht ist [PY15].

A.1.4 Data Center Infrastructure Management

„Werkzeuge, die es den Betreibern eines Rechenzentrums erlauben diese komplexe Umgebung effektiv und effizient zu betreiben, wurden in eine Gruppe klassifiziert die im Allgemeinen als DCIM bezeichnet wird [Col12].“ Diese Gruppe von Werkzeugen umfasst viele Bereiche wie Asset Management (z.B. CMDBs) und Real-Time Monitoring. Eine Möglichkeit, die Berechtigungen in einem Rechenzentrum zu definieren, ist über Policies und dem sogenannten Policy-Based Management, auf welches in Abschnitt A.1.4.1 eingegangen ist. Wenn die Komponenten eines Systems diese Policies selbst bestimmen, nennt man diese autonom (siehe Abschnitt A.1.4.2). Gemeinsam für alle Komponenten können die Policies in einer Policy Information Base verwaltet werden (siehe Abschnitt A.1.4.3).

A.1.4.1 Policy-Based Management

In verteilten Systemen sind die Policies das Regelwerk, welches das Verhalten einer Komponente definiert [Str03]. Der Ansatz des PBM zielt darauf ab, dass diese Regelwerk eigenständig definiert und verwaltet wird, um dann von den Komponenten interpretiert zu werden. Dieser Ansatz hat (im Vergleich mit hartcodierten Regelwerken) den Vorteil, dass Änderungen des Regelwerks keine Änderungen der Komponenten nach sich ziehen [Slo94]. Das erlaubt den Administratoren eine einfache und flexible Definition des Regelwerks. Solche Policies können beispielsweise mit Ponder, einer Spezifikationssprache, definiert werden [DDLS01].

Im Kontext der PBM wird zwischen dem Informationsmodell und dem Datenmodell unterschieden [SHC+01a]. Das Informationsmodell (auch abstraktes oder konzeptuelles Modell genannt [PS03]) entspricht einem

Kommunikationsmodell[BKW12]. Mit einem Informationsmodell wird abstrakt beschrieben, wie Policies lauten können. Das definiert die verwalteten Objekte (engl. managed objects) und Abhängigkeiten zwischen Objekten auf einer konzeptionellen Ebene, ist jedoch noch von konkreten Protokollen unabhängig. Ein Datenmodell ist eine Instanz eines Informationsmodell und dient der Realisierung, es enthält bereits implementierungs- und protokollspezifische Informationen.

A.1.4.2 Autonome Konfiguration

Policy-based Management kann auch autonom durchgeführt werden. Das bedeutet, dass jede Komponente selbst bestimmt, welche Sicherheitsanforderungen sie hat. Der Ansatz der autonomen Konfiguration ermöglicht es den Komponenten eines verteilten Systems, selbständig alle sicherheitsrelevanten Konfigurationen vorzunehmen. Dies hat den Vorteil, dass die Sicherheit einer Komponente ohne die Betrachtung der Konfigurationen anderer Komponenten evaluiert werden kann. Klenk, Niedermayer, Masekowsky und Carle stellen das Extensible Security Adaptation Framework (ESAF) vor, welches Komponenten erlaubt, selbst festzulegen, welche Sicherheitsanforderungen für die Kommunikation mit anderen Komponenten gelten [KNMC06]. Die ESAF-Nutzer definieren die Sicherheitsanforderungen mit der Requirements Description Language (RDL) und das Framework kann auf dieser Basis eine geeignete Kommunikationsverbindung verhandeln. Diese strikte Trennung von Sicherheitsanforderungen und Konfiguration erlaubt es den ESAF-Nutzern, stets mit der aktuell bestmöglichen Konfiguration zu agieren. Eine autonom konfigurierte Komponente ist daher unabhängiger von der Konfiguration anderer Komponenten, und die Sicherheitsbeurteilung kann im Mikrokosmos der Komponente selbst stattfinden. Für die Interaktion zweier Komponenten setzt das ESAF voraus, dass beide Komponenten das ESAF implementieren.

A.1.4.3 Policy Information Base

Eine PIB ist eine Art Datenbank für Policies. Die Policies werden in dieser Datenbank im Format sogenannter Provisioning Classes gespeichert, welche eine geordnete Menge von Attributen repräsentiert [SHC+01a]. Es gibt

verschiedene implementierte PIBs, so stellen Squair et. al. beispielsweise [SJN05] eine PIB vor, die speziell für das Verwalten von RBAC-Autorisierungsregeln entwickelt wurde.

A.2 Komponentenbeschreibungen der verteilten Systeme

Dieser Abschnitt stellt die Komponententypen der untersuchten verteilten Systeme vor. Hierfür ist für jede Fallstudie das Sicherheitsdiagramm (siehe Abschnitt 7.1.1.2) abgebildet. Das Sicherheitsdiagramm gibt eine Übersicht über das Zusammenspiel der Komponenten. Anschließend sind die einzelnen Komponententypen beschrieben.

A.2.1 Fallstudie Hochsicherheitsrechenzentrum

Die Abbildung A.1 zeigt das Sicherheitsdiagramm des Hochsicherheitsrechenzentrum (kurz HSRZ). Im Anschluss sind die einzelnen Komponenten des HSRZ beschrieben.

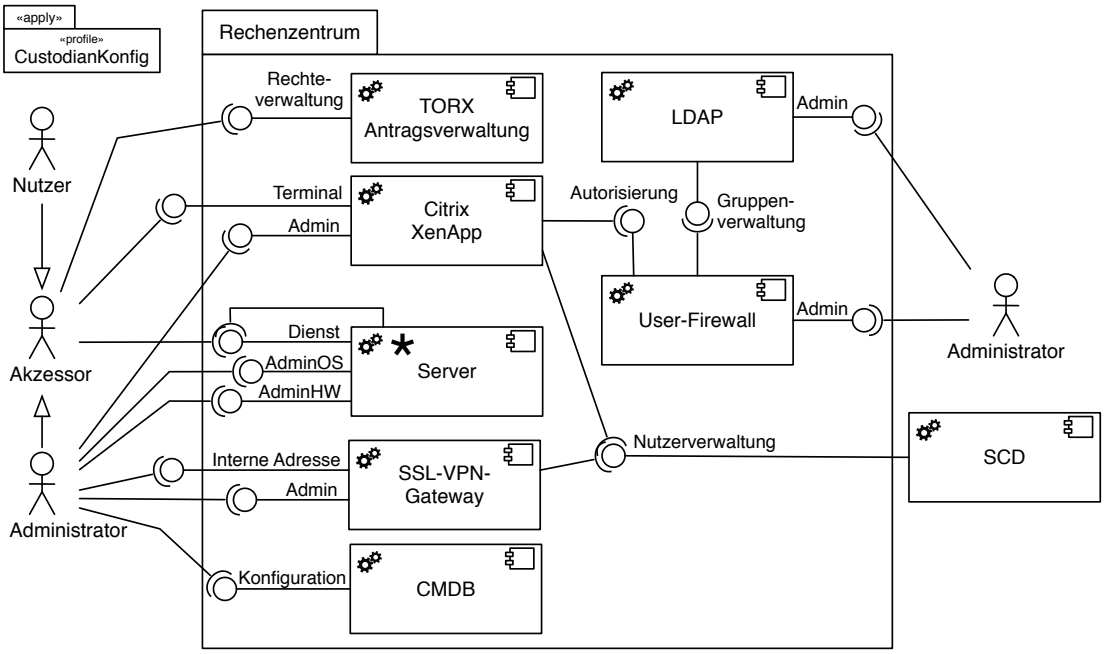


Abb. A.1: *Sicherheitsdiagramm des HSRZ*

Die Server sind der zu beschützende Kern des HSRZ und stellen den Nutzern der Kunden die gewünschten Funktionalitäten bereit. Server bieten, wie in Abbildung A.1 modelliert, die drei Typen von Schnittstellen Dienst,

AdminBetriebssystem und AdminHardware an. Über Dienste werden den Nutzern Funktionalitäten bereitgestellt. Über die Administrationsschnittstelle des Betriebssystems können das Betriebssystem und die darauf befindlichen Applikationen gewartet werden. Die Administrationsschnittstelle steht für den Notfall zur Verfügung, dass ein Betriebssystem nicht mehr reagiert. Über diese Schnittstelle hat man all die Möglichkeiten, die man auch hat, wenn man physikalisch im Rechenzentrum vor dem Servers steht. Der Betrieb der verschiedenen Kundenanwendungen erfordert, wie in Abbildung A.2 dargestellt, den Einsatz verschiedener Servern. Diese unterteilen sich in dedizierte und allgemeine Server. Dedizierte Server gehören zu genau einer dem Benutzer bereitgestellten Anwendung. Allgemeine Server stellen anwendungsübergreifende Dienste bereit.

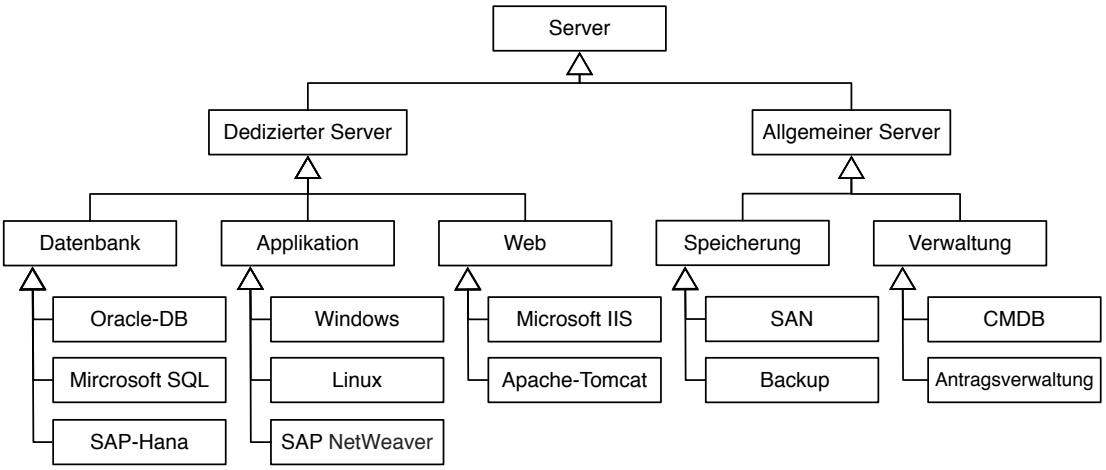


Abb. A.2: *Servertaxonomie im HSRZ*

Als Applikationsserver, also Server, auf denen individuelle Anwendungen betrieben werden können, stehen Windows- und Linux-Server zur Verfügung. Für die SAP-Anwendungen wird der dafür optimierte Applikationsserver SAP NetWeaver eingesetzt. Für den Nutzer gibt es prinzipiell zwei Möglichkeiten, auf die von einem Applikationsserver angebotenen Dienste zuzugreifen. Er kann sich über eine Remote-Desktopverbindung auf dem Server anmelden und von dort aus die Applikation starten. Und er kann auf dem Terminalserver einen Rich-Client starten, der eine Verbindung mit dem Applikationsserver aufbaut.

Mit Microsoft IIS und Apache Tomcat existieren zwei verschiedene Webserverlösungen zum Betreiben von webbasierten Anwendungen. Diese Weban-

wendungen kann der Nutzer vom Terminal aus mit dem Browser aufrufen. Als Datenbanklösungen werden die drei Datenbanksystemen OracleDB, Mircrosoft SQL-Server und SAP-Hana betrieben. Die Dienste der Datenbanken werden oft nicht direkt von den Kunden, sondern von einem Web- oder Applikationsserver in Anspruch genommen. In seltenen Fällen verbinden sich auf dem Terminal gestartete Rich-Clients direkt mit einer Datenbank.

Sie stellen entweder Administratoren oder anderen Servern Dienste bereit. Zum allgemeinen Speichern von Daten steht ein Storage Area Network (SAN) zur Verfügung. Alle Nutzdaten der Datenbankserver, Applikationsserver und Webserver werden auf der SAN abgelegt. Spezielle Backupserver sichern in regelmäßigen Abständen die Datenstände der anderen Server, damit diese im Falle eines Datenverlusts wiederhergestellt werden können. Die beiden Servertypen CMDB und Antragsverwaltung stellen keine Kundenapplikationen dar. Sie dienen der internen Verwaltung und spielen für den Betrieb des Rechenzentrums eine wichtige Rolle. Deshalb sind sie, obwohl die Antragsverwaltung eine Applikation auf Basis eines Webservers und der CMDB auf Basis eines Applikationsservers laufen, gesondert modelliert.

Der XenApp-Server von Citrix ist neben dem SSL-VPN die einzige vorgesehene Möglichkeit, eine Verbindung in das Rechenzentrum aufzubauen. Hierfür muss sich der Akzessor mittels Zwei-Faktor-Authentifizierung [HA03] einloggen. Besitz und Wissen des Akzessors werden als Faktoren der Authentifizierung herangezogen. Mit diesen Faktoren kann ein Akzessor sich an einem von XenApp-Server bereitgestellten Portal anmelden. Dieses Portal (siehe Abb. A.3) wird als frei über das Internet erreichbare Internetseite bereitgestellt.

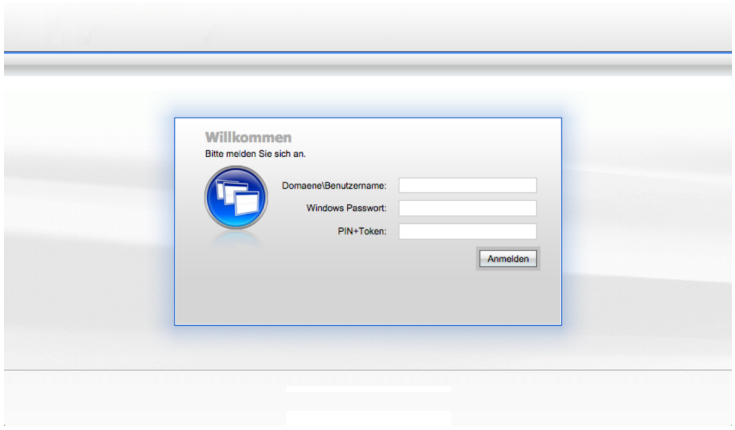


Abb. A.3: *Loginmaske des Citrix Terminals*

Der Faktor Wissen setzt sich aus den drei Komponenten Benutzername, Password und Pin zusammen. Der Benutzername identifiziert eine Person und wird im AD geführt. Da der Benutzername über das AD unter der Bezeichnung „GUID“ eingesehen werden kann, gilt er nicht als geheimes Wissen. Das Passwort wird durch den Akzessor definiert und ist von ihm als Geheimnis zu bewahren. Die Pin wird dem Akzessor initial exklusiv mitgeteilt und ist ebenfalls ein Geheimnis. Der Faktor Besitz wird über einen Hardware-Token [Rub96], wie in Abbildung A.4 dargestellt, realisiert. Dieser Token erstellt für kurze Zeit gültige, aus 6 Ziffern bestehende Einmalkennwörter. Diese Einmalkennwörter stehen nur dem Besitzer des Tokens und dem Server zur Verfügung. Wählt man sich mit diesen vier Komponenten erfolgreich bei der Loginmaske an (Abb. A.3), wird man zur Serverübersicht in Abbildung A.5 weitergeleitet. Durch das Auswählen eines Servers baut sich eine Remote Desktop Connection (RDC) zu einer von Citrix verwalteten und im Rechenzentrum laufenden Windows-Instanz auf. Das Terminal zeigt die Nutzeroberfläche der Server-Instanz im Browser des Benutzer an und leitet die Benutzereingaben an die Windows-Instanz weiter. Vor dieser Windows-Instanz aus kann man die Dienste im Rechenzentrum erreichen. Damit liegen nicht nur die Server und ihre Dienste, sondern auch das auf die Dienste zugreifende System in der Kontrolle des Rechenzentrums.

Abb. A.4: *Der Token stellt den Faktor Besitz bei der Authentifizierung bereit*

Diese Methode erlaubt es den Betreibern des Rechenzentrums, auch Kontrolle über die zugreifenden Systeme ausüben zu können. Die zugreifenden Systeme werden im Rechenzentrum selbst betrieben. Dadurch können sie auf Schadsoftware wie Vieren, Trojaner oder Spionagesoftware überprüft werden. Jedoch besteht, wenn man sich über Citrix mit dem Rechenzentrum verbunden hat, noch kein freier Zugriff auf alle Server des Rechenzentrums. Um auf einen Server von Citrix aus zu erreichen, muss dieser für den Benutzer auf der User-Firewall freigeschaltet sein.

Abb. A.5: *Serverübersicht des Citrix Terminals*

Das Active Directory (AD) ist das zentrale Personenverzeichnis des Technologieunternehmens mit dem die Basisdaten zu Personen verwaltet werden. Jeder, der einen personalisierten Zugang zu dem Technologieunternehmen hat, ist im AD registriert. Eine Teilmenge dieser Personen sind Akzessoren. Das AD ist ein System, das nicht im HSRZ betrieben wird. Deshalb steht es auch nicht im Verantwortungsbereich des HSRZ. Dennoch basieren die Autorisierungen von Personen auf den AD-Einträgen.

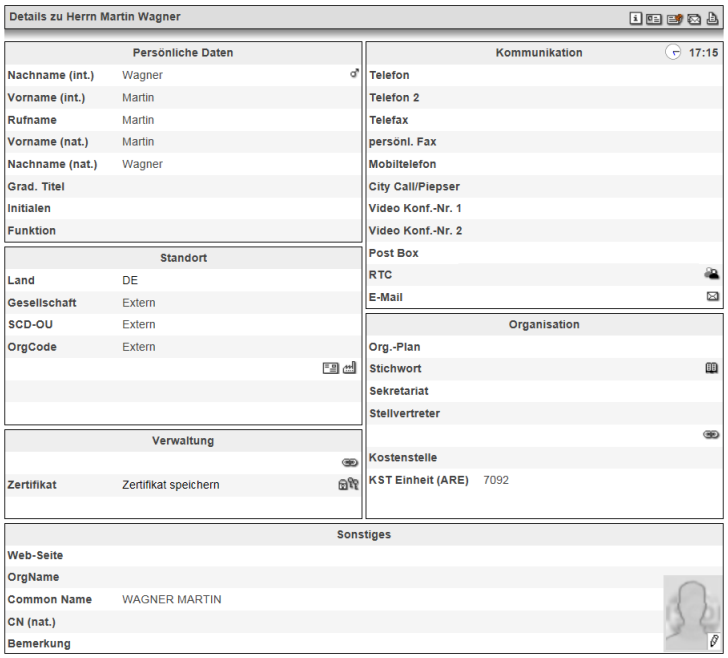


Abb. A.6: *Personenbasisdaten im AD*

Das AD stellt Schnittstellen bereit, über die andere Systeme Informationen zu den registrierten Personen erfragen können. Über diese Schnittstellen bezieht die TORX-Antragsverwaltung regelmäßig aktuelle Basisdaten zu den Akzessoren. Das AD kann innerhalb des Intranets des Technologieunternehmens auch von Personen erreicht werden. Über eine interne Webseite können Informationen zu den registrierten Personen eingesehen werden. Abbildung A.6 zeigt einen Bildschirmabzug von einem AD-Eintrag auf dieser internen Webseite.

Die User-Firewall (auch User-Based Firewall genannt) ist ein Produkt der Firma Palo Alto. Diese Firewall ist im Gegensatz zu klassischen Firewalls in der Lage, Netzwerkverkehr auf der Basis des Benutzers, von dem der Netzwerkverkehr ausgeht, zu filtern. Um dies zu erreichen, interagiert die User-Firewall eng mit dem Citrix XenApp-Server und dem LDAP. Der Citrix XenApp-Server reichert jede in das Rechenzentrum gerichtetes Datenpaket um den Authefizierungskontext an. Dadurch ist der User-Firewall die Identität des zugreifenden Nutzers bekannt. Basierend auf ihrem Regelwerk ermittelt die User-Firewall die Regeln, die Zugriff auf das Ziel der Anfrage erlauben. Jede der Regeln verweist auf Menge von Gruppen im LDAP. Die User-Firewall ermittelt am LDAP die Mitglieder der Gruppe und kann so entscheiden, ob die Regel für ein konkretes Datenpaket wirksam ist. Die

User-Firewall stellt somit sicher, dass die Benutzer ausschließlich auf die für sie bestimmten Servern über das Netzwerk zugreifen können.

Das Lightweight Directory Access Protocol – kurz LDAP – steht, obwohl sich hinter dem Begriff lediglich das verwendete Zugriffsprotokoll verbirgt, für einen Verzeichnisdienst, der auf einer hierarchisch organisierte Datenbank basiert. Die im HSRZ verwendete Implementierung ist das „Active Directory“ von Microsoft. Im LDAP werden alle Gruppenmitgliedschaften verwaltet, die von der User-Firewall benötigt werden. Durch die hierarchische Natur des LDAP können Gruppen nicht nur Benutzer, sondern weitere Gruppen beinhalten. Benutzer sind dadurch nicht nur Mitglied der Gruppen, denen sie direkt zugeordnet sind, sondern auch aller derer Übergruppen.

Das SSL-VPN ist ein Virtual Private Network (VPN) und ist, neben der Citrix XenApp, der zweite bereitgestellte und erlaubte Weg, um eine Verbindung mit dem Rechenzentrum aufzubauen. Es handelt sich bei einem SSL-VPN um ein VPN, welches, wie bereits durch den Name angedeutet, durch das Verschlüsselungsprotokoll Secure Sockets Layer (SSL) geschützt ist. Das VPN bietet die Möglichkeit, einen eigentlich externen Rechner in das Netzwerk des Rechenzentrums einzubinden. Hierfür baut der VPN-Client eine Verbindung zum VPN-Gateway im Rechenzentrum auf und bekommt eine interne Netzwerkadresse des Rechenzentrums zugewiesen. Ab diesem Zeitpunkt agiert der Rechner wie ein tatsächliches System des Rechenzentrums und der gesamte Netzwerkverkehr wird über das Netzwerk des Rechenzentrums geleitet. Auf dem VPN-Gateway existiert für jeden registrierten Benutzer ein Nutzerprofil. Dieses Nutzerprofil definiert, auf welche Ressourcen des Rechenzentrums dieser Benutzer zugreifen kann. Ein solcher VPN-Zugang wird ausschließlich den Administratoren des HSRZ bereitgestellt.

Das HSRZ setzt eine CMDB ein. Die CMDB-Anwendung wird auf einen nur über Citrix erreichbaren Server betrieben. In der CMDB werden mehrere Informationen zur Konfiguration der Server des Rechenzentrums verwaltet. Die wichtigsten dieser Informationen sind die IP-Adressen, die Zugehörigkeiten zu Hierarchieren und Klassifizierungen von Servern.

Server haben im Regelfall 4 IP-Adressen: Eine IP-Adresse, über die seine Dienste und die Administrationsoberfläche des Betriebssystems erreichbar sind. Eine, über die man auf die Administrationsoberfläche Hardware zugreifen kann. Und jeweils eine IP-Adresse mit er mit dem SAN- beziehungsweise dem Backup-Server kommuniziert.

Jeder Server des Rechenzentrums ist über eine Hierarchie einer Systemlinie Zugeordnet (siehe Abbildung A.7). Eine Systemlinie definiert einen im Rechenzentrum beherbergten Anwendungstyp wie zum Beispiel ein konkretes Buchhaltungssytem. Eine Systemlinie hat ein oder mehrere Lösungen, wobei eine Lösung eine Instanz des Anwendungstyps ist. Mehrere Lösungen einer Systemlinie können beispielsweise durch mehreren Kunden exklusiv bereitgestellte Instanzen oder durch den zusätzlichen Betrieb einer Entwicklungs- oder Testumgebung erforderlich werden. Eine Lösung benötigt immer mindestens einen Dienst. Typischerweise wird von einer Lösung ein Datenbankdienst und ein Web- bzw. Applikationsdienst beansprucht. Ein Dienst wird von ein oder mehreren Servern bereitgestellt. Die Bereitstellung eines Dienstes durch mehrere Server kann beispielsweise zur Gewährleistung hoher Kapazitäten oder Ausfallsicherheit notwendig sein. Eine solche Hierarchie ist in Beispiel 18 anhand eines Buchhaltungssystems beschrieben.

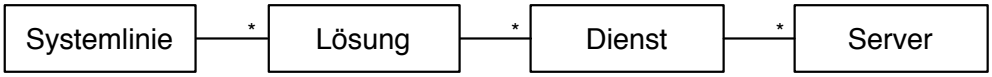
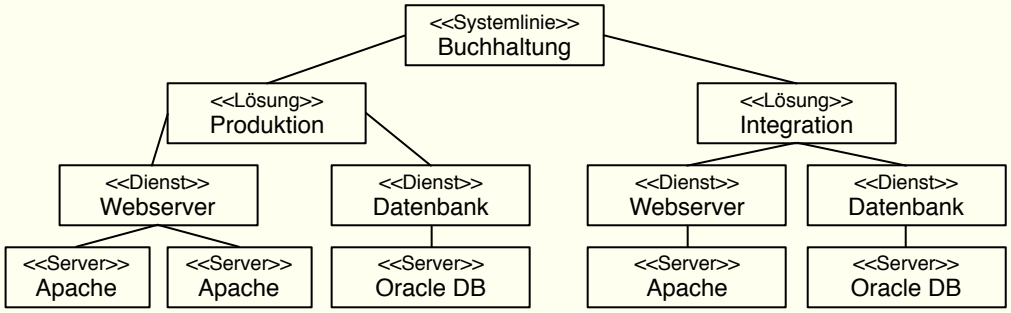


Abb. A.7: *Hierarchische Strukturierung von Servern im Rechenzentrum*

Beispiel 18

Von einem im Rechenzentrum laufenden Buchhaltungssystem werden zwei Lösungen betrieben. Die erste Lösungen gewährleistet den produktiven Betrieb, die zweite steht für die Integration, also die Entwicklungsarbeiten, zur Verfügung.



Eine Lösung des Buchhaltungssystem benötigt die Dienste Webserver und Datenbank. Der Webserver-Dienst wird in der Produktion von zwei und in der Integration von einem Apache-Server bereitgestellt. Der Datenbank-Dienst wird in beiden Lösungen von jeweils einer Oracle-DB realisiert.

Zusätzlich zu den IP-Adressen und den hierarchischen Zuordnungen werden in der CMDB drei Klassifizierungen auf der Hierarchieebene der Systemlinien geführt. Systemlinien können mit den Kennzeichen VS-NfD, SOA und Golden Nugget (GN) versehen werden. VS-NfD kennzeichnet Systeme, bei denen der Bund als Kunde auftritt und besondere Anforderungen an die Geheimhaltung stellt. SOA kennzeichnet Systeme, die, um den Anforderungen des amerikanischen Marktes gerecht zu werden, besonderen Regulationen unterliegen. Systeme, die aus der Sicht des Technologieunternehmens als besonders wichtig gelten, werden mit GN gekennzeichnet.

In der CMDB werden wichtige Informationen der im Rechenzentrum beherbergten Systeme gepflegt. Die Klassifizierungen der Systemlinien und die hierarchischen Beziehungen zu deren Servern werden ausschließlich in der CMDB geführt. Deshalb ist die CMDB als Datenquelle für zur Gefahrenbewertung unersetzlich.

Die Antragsverwaltung ist das zentrale System zur Verwaltung von Berechtigung im HSRZ. Alle Berechtigungen auf Servern, für das Technologieunternehmen als Betreiber des Rechenzentrums verantwortlich ist, werden über dieses System verwaltet.

Akzessoren sind die Personen, die auf Systeme des HSRZ zugreifen. Jede Person, die Rechte im HSRZ besitzt, ist daher ein Akzessor. Akzessoren besitzen einen sogenannten Global Unique Identifier (GUID) zur eindeutigen Identifizierung. Die GUID ist ein bei dem Technologieunternehmen gebräuchlicher Terminus und wird für interne und externe Mitarbeiter sowie Mitarbeiter von Kunden vergeben. Ohne eine GUID kann man weder selbständig das Betriebsgelände des Technologieunternehmens betreten oder Dienste des HSRZ zugreifen.

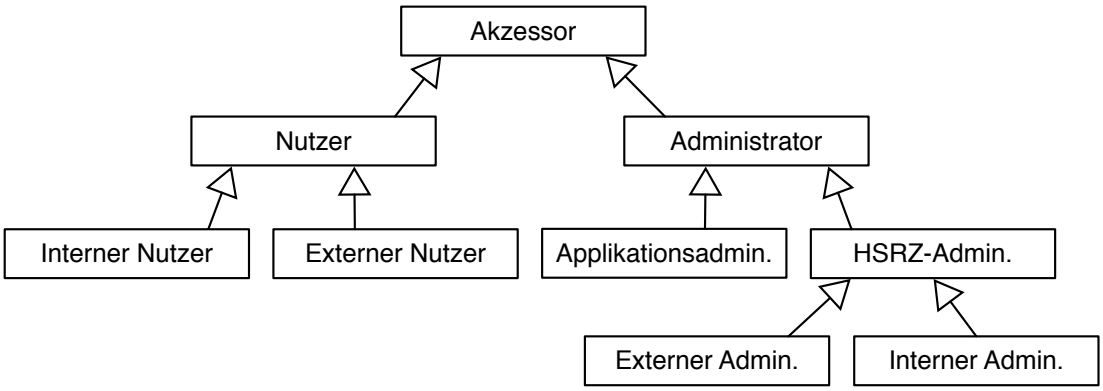


Abb. A.8: *Taxonomie der Akzessoren des HSRZ*

A.2.1.1 Konfigurationsdaten, Datenbeschaffung und Ressourcen

Der Workflow Entwurf des Prozessmodells HARVEST beschreibt wie die benötigten Konfigurationsdaten erhoben, die Datenbeschaffung zu organisieren und die Ressourcen zu planen sind (siehe Abschnitt 7.2). Im Folgenden sind die Ergebnisse dieses Prozesses am Beispiel der HSRZ-Fallstudie aufgeführt.

Konfigurationsdaten: Die folgende Tabelle zeigt welche Konfigurationsdaten der Heuristik bereitstehen müssen und von welchen Komponenten diese bezogen werden können.

Benötigte Konfiguration	**Führende Komponente**
Akzessoren	AD
Gruppenmitgliedschaften	LDAP
Servern	CMDB
IP-Adressen	CMDB
Netzwerkregeln	SSL-VPN und User-Firewall
Logins	Antragsverwaltung

Tab. A.1: *Erforderliche Konfigurationsdaten des verteilten Systems*

Datenbeschaffung: In Tabelle A.2 sind die Rahmenbedingungen für die Beschaffung der Konfigurationen dargelegt.

Komponente	**Aktualität**	**Bereitstellung**	**Schnittstelle / Format**
AD	02:00 am	Datenbank	JDBC
LDAP	immer aktuell	LDAP-Dienst	LDAP
CMDB	02:00 am	Datenbank	JDBC
SSL-VPN	02:00 am	Datei	XML
User-Firewall	02:00 am	Datei	XML
Antragsverwaltung	02:00 am	Datenbank	JDBC

Tab. A.2: *Rahmenbedingungen für die Beschaffung der Konfigurationen*

Die Konfigurationsdaten des AD, der CMDB und der Antragsverwaltung liegen in Datenbanken, auf die von KUSTOS nicht direkt zugegriffen werden darf. Deshalb wird täglich um 2 Uhr eine Kopie dieser Datenbanken auf dem

KUSTOS-Server eingespielt. Da der Export der Datenbanken die Leistung des produktive System beeinflusst, darf dieser nur in der Nacht durchgeführt werden. KUSTOS darf direkt auf das LDAP-Verzeichnis zugreifen und kann somit stets den aktuellen Stand abfragen. Auch auf die Komponenten SSL-VPN und User-Firewall darf KUSTOS nicht direkt zugreifen. Deshalb legt eine externe Backup-Routine täglich einen Abzug der XML-Konfiguration dieser beiden Komponenten auf dem KUSTOS-Server ab.

Ressourcen: In Tabelle A.3 sind die für die Beschaffung der Konfigurationsdaten benötigten Ressourcen aufgeführt.

Komponenten	Vorlauf	Ressourcen
AD CMDB Antragsverwaltung	15 Tage	2pt CMDB-Experten 2pt Antragsverwaltungs-Experten 5pt Datenbank-Administrator 1pt Server-Administrator
LDAP	5 Tage	2pt LDAP-Experten 1pt LDAP-Administrator
SSL-VPN User-Firewall	3 Tage	1pt Server-Administrator 3pt User-Firewall-Experten 2pt SSL-VPN-Experten

Tab. A.3: *Konfigurationsdaten der Gefahrenklasse Witwe*

A.2.2 Fallstudie RGB-Netzwerk

Die Abbildung 8.41 zeigt das Sicherheitsdiagramm des RBG-Netzwerks. Im Anschluss sind die einzelnen Komponenten des RBG-Netzwerks beschrieben.

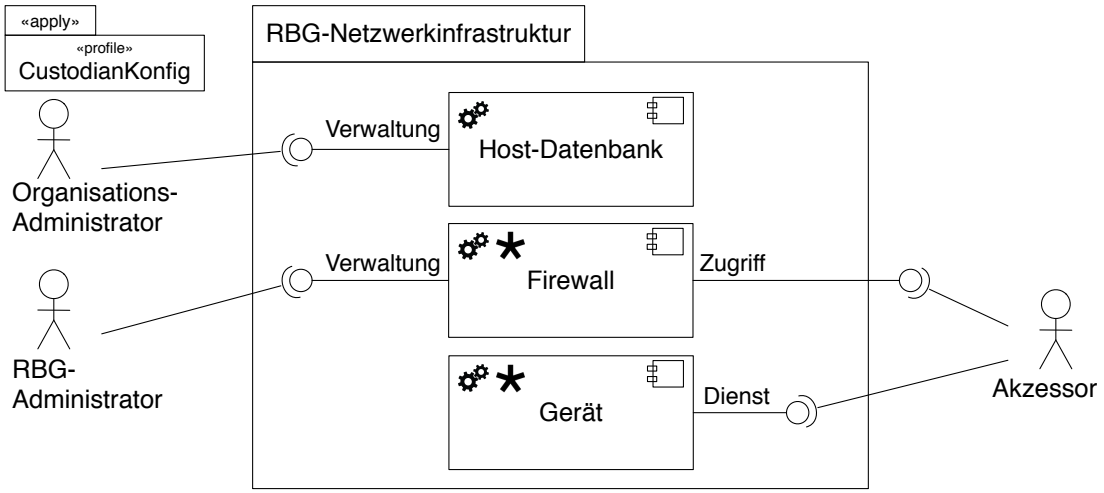


Abb. A.9: *Sicherheitsdiagramm der Netzwerkdienstleisters RBG*

Die RBG betreibt für jedes von ihr bereitgestellte Netz eine eigene logische Firewall. Die logischen Firewalls sind virtuell und werden gemeinsam auf einer Firewall-Hardware des Typs Brocade MLX-8 betrieben. Sowohl die logischen Firewalls als auch die physikalischen Firewall werden von RBG-Mitgliedern administriert. Jede Organisationseinheit (Lehrstühle und Arbeitsgruppen), die den Firewalldienst der RBG in Anspruch nimmt, erhält eine eigene Firewall, deren Konfiguration sie, mit Hilfe der RBG selbst bestimmt. In Abbildung A.10 ist ein Auszug einer virtuelle-Firewall-Konfiguration dargestellt.

```
remark jirabruegge 2015102910004547
permit tcp any host 131.159.39.205 eq 80
permit tcp any host 131.159.39.205 eq 443
remark repobruegge 2015102910004547
permit tcp any host 131.159.39.230 eq 80
permit tcp any host 131.159.39.230 eq 443
permit tcp any host 131.159.39.230 eq 7999
```

Abb. A.10: *Auszug aus einer Konfiguration einer logischen Firewall*

Über die Host-Datenbank können die Administratoren der Organisationseinheiten IP-Adressen vergeben sowie die zugehörigen Mac-Adressen Hostnamen bestimmen. Eine Intranetseite (siehe Abbildung A.11) ermöglicht das Anmelden an der Host-Datenbank.

Abb. A.11: *Eintrag eines Servers in der Host-DB*

Die Organisationseinheiten betreiben in ihren Netzwerken eine Menge von Geräten. Ein Gerät kann ein herkömmliche Server, wie ein Web-Server oder ein Datenbankserver, sein. Zusätzlich können aber auch Drucker oder andere Geräte über das Netzwerk erreichbar sein. Für die Organisationseinheiten gibt es zwei Möglichkeiten, um Server zu betreiben. Erstens ist es möglich, virtuelle Maschinen bei der RBG zu beantragen. Zweitens können die Organisationseinheiten auch eigene Server in ihren Räumlichkeiten betreiben. Beide Arten von Servern können in das Netzwerk der Organisationseinheit aufgenommen werden. Somit können sie durch die Firewalls beschützt und über die Host-DB verwaltet werden.

A.2.3 Fallstudie AdWorks2go

Die Abbildung 8.46 zeigt das Sicherheitsdiagramm der AdWorks2go-Netzwerks. Im Anschluss sind die einzelnen Komponenten der AdWorks2go-Netzwerks beschrieben.

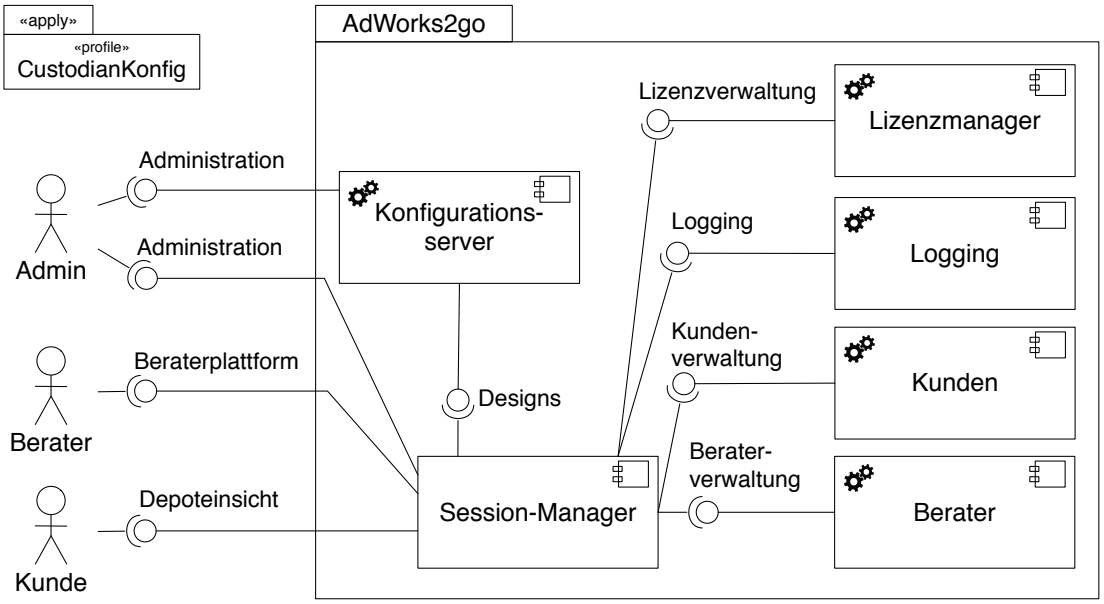
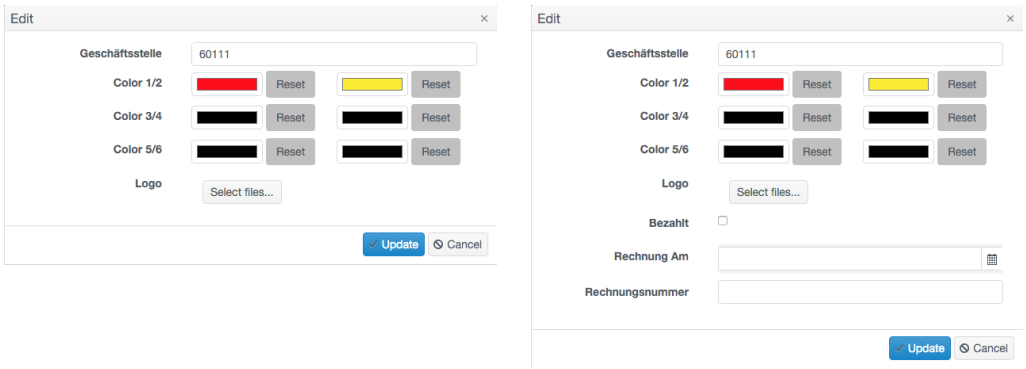


Abb. A.12: *Sicherheitsdiagramm für AdWorks2go*

Der Konfigurationsserver erlaubt es, verschiedene Designs für verschiedene Kunden zu definieren. Vor der Fallstudie bestand die Eingabemaske für ein Design aus den Feldern `Geschäftsstelle` und `Color 1` bis `Color 6` sowie der Möglichkeit ein Logo hochzuladen. Aufgrund der Erkenntnisse Fallstudie (siehe Gefahrenklasse „Unbezahlte Designs“ in Abschnitt 8.3.3.2) der wurde der Konfigurationsserver um die Attribute `Bezahlt`, `Rechnung Am` und `Rechnungsnummer` erweitert. Der Konfigurationsserver wird ausschließlich von Mitarbeitern der financeTec AG administriert.



(a) Ursprüngliche Eingabemaske (b) Erweiterte Eingabemaske

Abb. A.13: *Verschiedene Versionen des Konfigurationsserver*

Der Session-Manager ist eine Fassade, die den Nutzern von AdWorks2go alle Dienste über ein REST-Interface zur Verfügung stellt. Der Session-Manager vereint die Dienste, die die Komponenten Lizenzmanager, Logging-DB, Kunden-DB und Berater-DB bereitstellen. Im Lizenzmanager kann von Mitarbeitern der financeTec AG eingestellt werden, welche Module einem Berater zur Verfügung stehen. Momentan wird nicht für jeden einzelnen Dienst sichergestellt, dass er dem Berater nicht zur Verfügung steht, wenn er im Lizenzmanager nicht ausgewählt ist (siehe Gefahrenklasse Unbezahlter Onlinezugriff in Abschnitt 8.3.3.1). Die Abbildung A.14 zeigt die Einstellungen, die für einen Berater im Lizenzmanager vorgenommen werden können.

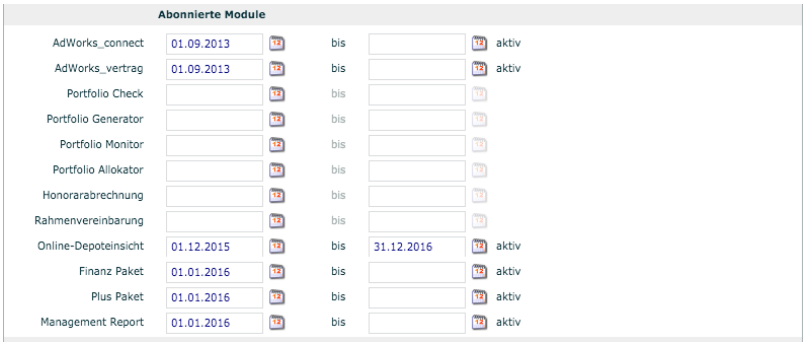


Abb. A.14: *Einstellungen für einen Berater im Lizenzmanager*

In der Logging-DB werden alle Ereignisse protokolliert, die später ausgewertet werden sollen. Neben auftretenden Fehlern werden auch erfolgrei-

che Logins protokolliert. Die Abbildung A.15 zeigt einen Ausschnitt von Ereignissen, die in der Logging-DB protokolliert wurden.

Level	Message	Vermittler	App	Version	Model	Device	Time
	User is logging in						2016-01-19T07:30:21…
	Login Successful		de.financete…	9.2.1	iPhone7,2	iPhone	2016-02-04T14:30:46…
	Login Successful		de.financete…	9.2.1	iPad5,4	iPad	2016-02-06T10:28:31…

Abb. A.15: *Auszug von Logeinträgen in der Logging-DB*

In der Kunden-DB werden sowohl die Kundenstammdaten also auch die Depotdaten der Kunden der Berater gespeichert. In der Berater-DB werden alle notwendigen Informationen über die mit AdWorks2go arbeitenden Berater gespeichert.

Abkürzungen

ACCD	Access Controll Class Diagram
AD	Active Directory
ACL	Access Controll List
API	Application Programming Interface
AWS	Amazon Web Services
CMDB	Configuration Management Database
CMU	Carnegie Mellon University
CPS	Cyber-physical system
DCIM	Data Center Infrastructure Management
DDL	Data Definition Language
DML	Data Manupulation Language
ESAF	Extensible Security Adaptation Framework
IaaS	Infrastructure as a Service
GUID	Global Unique Identifier
DIN	Deutsche Industrie-Norm
FMI	Fakultät für Mathematik und Informatik
GN	Golden Nugget
HQL	Hibernate Query Language
IMIS	Institutional Management Information System
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
IoT	Internet of Things
iRMC	integrated Remote Management Controller
ISO	International Organization for Standardization
KUT	Konfiguration Unter Test
LDAP	Lightweight Directory Access Protocol

MTF Meta Threat Facility
MWN Münchner Wissenschaftsnetz
NVD National Vulnerability Database
OCL Object Constraint Language
OWA Open World Assumption
OWL Web Ontology Language
ORM Object Role Modeling
PIB Policy Information Base
SML Security Modelling Language
SSL Secure Sockets Layer
TUM Technische Universität München
PBM Policy-Based Management
RBG Rechnerbetriebsgruppe
RDL Requirements Description Language
RBG Rechnerbetriebsgruppe
RDP Remote Desktop Protocol
RDC Remote Desktop Connection
RBAC Role-Based Access Control
SDN Software-Defined Networking
SOA Sarbanes-Oxley Act
SPOF Single Point of Failure
SQL Structured Query Language
SUT System Under Test
SSH Secure Shell
ULSS Ultra-Large-Scale System
UML Unified Modeling Language
VPN Virtual Private Network
VS-NfD Verschlusssachen - Nur für den Dienstgebrauch

Abbildungsverzeichnis

1.1 Zwischen 2010 und 2014 stieg die Anzahl der bei der NVD gemeldeten Sicherheitslücken um 65 Prozent 3
1.2 Exemplarische Evolution einer IT-Infrastruktur 5

2.1 Termini der Gefahrenerkennung in verteilten Systemen 9
2.2 Komponenten als Teil eines verteilten Systems 13
2.3 Spezifizierte Funktionsweise, beobachtete Funktionsweise, Konfiguration und Verhalten einer Komponente 13
2.4 Die Konfiguration eines verteilten Systems 15
2.5 Die Konfigurationsklassen eines LDAP-Verzeichnis 15
2.6 Herleitung und Zusammenhang von Fehlkonfigurationen, Gefahren und Schadensfällen 16
2.7 Taxonomie von Konfigurationen 18
2.8 Exemplarische Taxonomie der Gefahrenklassen eines verteilten Systems 19
2.9 Der Zusammenhang zwischen Gefahrenklasse, Sicherheitstest und Gefahr 21
2.10 Die Erhebung von Testmengen in verteilten Systemen 23
2.11 Komposition eines verteilten Systems als System Under Test . . 25
2.12 Die Konfiguration Unter Test als Teil des System Under Test . . 25
2.13 Eine Taxonomie zur Synchronisation zwischen Komponenten . 29
2.14 Erweiterbare Taxonomie über Wissen hinsichtlich der Gefahrenerkennung 30
2.15 Taxonomie über Kapazitäten 31
2.16 Eine Taxonomie über Gefahrenwissen und Kapazitäten 32
2.17 Ein Modell der Komponenten, Spezialisten und des Gefahrenwissens einer Gefahrenklasse 32

3.1 Äquivalenzen der Bodenhofschen Wissenspyramide [Bod06] und der Gefahrenpyramide 38

3.2 Gefahrenmanagement unter der Annahme existierender Gefahren . . . 39
3.3 Gefahrenwissenstransfer von Spezialisten zu Experten . . . 43
3.4 Motion-Chart zur Visualisierung von Abhängigkeiten zwischen Lebenserwartung, Einkommen, Bevölkerungsanzahl und Staaten [RRR05] . . . 45
3.5 Clusterkarte einer Anwendungslandschaft mit logischen Domänen [Mat08] . . . 46
3.6 Politische Landschaft: Eine Anordnung Schweizer Gemeinden abhängig von deren Wahlergebnissen. Übernommen aus [Lor11] 47

4.1 Hypothesenbaum zur Validierung der Ausgangshypothese . . . 50

5.1 Der Kern der Meta Threat Facility (MTF) als UML-Profil . . . 58
5.2 Beispielhafte Taxonomie in MTF über Schadsoftwarearten (Ebene M2) . . . 58
5.3 Prozess zur Gefahrenerkennung in verteilten Systemen und der Gefahrenklassenerkennung für MTF . . . 60
5.4 Beispielhafter Einsatz von CUSTODIAN in einem verteilten System. Der Arbeitsaufwand der drei Aktivitäten Replikation, Exploration und Akquise sind in den oberen drei Bahnen der Abbildung modelliert. Die Anzahl der in MTF definierten Gefahrenklassen ist in der vierten Bahn modelliert. Und die Anzahl der Gefahrenexperten auf dem Sicherheitsblackboard von CUSTODIAN ist in der letzten Bahn modelliert. . . . 61
5.5 Die Erweiterung von MTF um Spionagemethoden (Ebene M2) . 62

6.1 Erkennung von Gefahren in verteilten Systemen mit CUSTODIAN 64
6.2 Taxonomie der Auslöser einer Sicherheitsüberprüfungen . . . 65
6.3 Anwendungsfälle für das Explorieren der Konfiguration Unter Test . . . 68
6.4 Anwendungsfälle für das Explorieren von Gefahren . . . 69
6.5 Anwendungsfälle zur Erkennung von Gefahren . . . 69
6.6 Anwendungsfälle für die Visualisierung von Gefahren . . . 70
6.7 Die Spezialfälle des Anwendungsfalls „erweitern um Experten“ 70
6.8 Definition der Fassade des Depots . . . 71
6.9 Die Schnittstelle Depot mit vier möglichen Implementierungen und deren Zugriffsfassade Depotfassade . . . 72
6.10 Das Datensatz-Modell des Depots . . . 73

6.11 Registrierung einer Wissensklasse am Depot am Beispiel einer Oracle-basierten Implementierung 74
6.12 Bereitstellen und Historisieren von Wissen am Beispiel eines Oracle-basierten Depots 75
6.13 Modell der Klasse Anfrage und ihrer referenzierten Klassen . . 76
6.14 Modell der Klasse Ergebnis und ihrer referenzierten Klassen . 76
6.15 Definition der Schnittstellen des Sicherheitsframeworks für Sicherheitsexperten inklusive ihrer Schnittstellenobjekte 78
6.16 Registrierung von Sicherheitsexperten am Sicherheitsframework 79
6.17 Definition der Schnittstellen des Sicherheitsframeworks für Sicherheitsexperten inklusive der von der Schnittstelle referenzierten Klassen 80
6.18 Bereitstellen von Wissen auf dem Depot durch einen Sicherheitsexperten 81
6.19 Aufbau eines Sicherheitstests 82
6.20 Prozesssteuerung der Experten auf dem Depot 83
6.21 Modell eines verteilten Systems samt Konfiguration aus der Betrachtungsweise des Sicherheitsframeworks 84
6.22 Vereinfachte Darstellung eines verteilten Systems unter Einbeziehung von Gefahrenklassen und -instanzen 84
6.23 Taxonomie von Sicherheitsexperten zur Gefahrenerkennung . . 85
6.24 Taxonomie der Observierer im Sicherheitsframework 86
6.25 Wissensklassen und deren Datensätze in CUSTODIAN 86
6.26 Analyse-Objektmodell des CUSTODIAN-Sicherheitsblackboards . 87
6.27 Der prinzipielle Aufbau einer Blackboard-Architektur in Anlehnung an Buschmann et. al. [Bus98] 88
6.28 Verfeinerung der Anwendungsfälle für die Blackboard-Architektur 89
6.29 Analyse-Objektmodell des CUSTODIAN-Sicherheitsframeworks . 89
6.30 Die Subsystemdekomposition der Sicherheitsframeworks auf Basis des Blackboard-Architekturmusters 91
6.31 Prinzipieller Aufbau des Sicherheitsframework im Zusammenspiel mit einem vereinfacht dargestellten verteilten System . . 92
6.32 Die Subsystemdekomposition der Referenzimplementierung KUSTOS 94
6.33 Exemplarische Darstellung eines Motion-Charts 97
6.34 Korrelation zwischen den Attributen zweier Wissensklassen . . 99
6.35 Der Korrelationsexperte zeigt Korrelationen zwischen Attributen 99

6.36 Erstellen einer neuen Wissensklasse mit dem Wissensbinder . . 100
6.37 Der Wissensbrowser zum Explorieren eines verteilten Systems 102
6.38 Anzeige eines Datensatzes mit dem Standard-Visualisierer . . . 103
6.39 Der Gefahrenanalytiker zum Generieren und Invalidieren von Gefahrenhypothesen 104
6.40 Datensätze der Wissensklasse Serverzugriffe 105
6.41 Gefahrenhypothesen auf Basis des Hypothesengenerators . . . 105
6.42 Gefahrenhypothesen auf Basis des Hypothesengenerators und des Invalidators 106

7.1 Das UML-Profil für Sicherheitsdiagramme 111
7.2 Prozessschritte des Sicherheitsentwurfs-Workflow 119
7.3 Der Sicherheitstest-Workflow als Aktivitätsdiagramm 122
7.4 Taxonomie der Auslöser des Harvest-Hauptprozesses 124
7.5 Exemplarischer Lebenszyklus des Harvest-Prozessmodells . . . 125

8.1 Überblick über die bei der Evaluation eingesetzten Methoden und die evaluierten Artefakte 127
8.2 Sicherheitsdiagramm des HSRZ 130
8.3 Taxonomie der Gefahrenklassen im HSRZ 133
8.4 Taxonomie der Entdeckungsarten der Gefahrenklassen 133
8.5 Prozess für Berechtigungen auf Dienste im HSRZ 134
8.6 Modell eines Berechtigungsauslauf 136
8.7 Definition der Testmenge für Berechtigungsausläufe via Wissensbinder 137
8.8 Motion-Chart der Berechtigungsausläufe am 11. Mai 14 138
8.9 Motion-Chart der Berechtigungsausläufe am 16. Mai 14 140
8.10 Motion-Chart der Berechtigungsausläufe am 3. Juni 14 140
8.11 Motion-Chart der Berechtigungsausläufe am 13. Juni 14 141
8.12 Neue Startseite der TORX-Antragsverwaltung 141
8.13 Motion-Chart der Berechtigungsausläufe am 23. November 15 142
8.14 Reguläre und annektierte Berechtigungen als Objektdiagramm 143
8.15 Testmengendefinition für Berechtigungsannexion auf dem Wissensbinder 145
8.16 Entwicklung der Berechtigungsannexionen vor und nach der Analyse 145
8.17 Entwicklung der Berechtigungsannexionen unter Berücksichtigung des Invalidators 146

8.18 Beispiel für den Aufbau einer Verbindung zu einem eigentlich nicht erreichbaren Server mittels Serverhopping 148
8.19 Definition der Testmenge für Trojaner mit dem Wissensbinder . 149
8.20 Definition der Testmenge für Trojaner mit dem Wissensbinder . 150
8.21 Entwicklung der Trojaner-Hypothesen 151
8.22 Entwicklung der Trojaner-Hypothesen nach Invalidierung der ungefährlichen Testobjekte 152
8.23 Das ätriebssystem als Subkomponente einer Hardware 153
8.24 Darstellung der verbleibenden gefährlichen Trojaner als Motion-Chart 153
8.25 Abhängigkeit zwischen der Antragsverwaltung und der CMDB 154
8.26 Der Server-Workflow der CMDB 155
8.27 Definition der Testmenge für abgebaute Server mit dem Wissensbinder 156
8.28 Darstellung der abgebauten Server als Motion-Chart 156
8.29 Entwicklung der abgebauten Server als Liniendiagramm 157
8.30 Überblick über die Gefahrenklasse Witwe 158
8.31 Objektmodell des HSRZ mit einer Witwe 159
8.32 Motion-Chart der Ergebnisse des Witwen-Sicherheitstests . . . 161
8.33 Motion-Chart nach der Invalidierung der Hypothesen inaktiver Server 162
8.34 Gefahrenhypothesen nach der Invalidierung der funktionaler Zugänge 162
8.35 Modell von Waisen in der Konfiguration des Rechenzentrums . 165
8.36 Die vom Sicherheitstest hypothetisierten Waisen 166
8.37 Hypothesen nach der Invalidierung der Backupregeln 166
8.38 Hypothesen nach der Invalidierung der Intranetregeln 167
8.39 Modell eines Zombies 168
8.40 Motion-Chart der Zombies nach der ersten Analyse 170
8.41 Sicherheitsdiagramm der Netzwerkdienstleisters RBG 172
8.42 Analyse der abgebauten Server am Lehrstuhl Brügge 178
8.43 Analyse der Generalisierungen am Lehrstuhl Brügge 179
8.44 Depotanalyse eines Kunden in AdWorks 181
8.45 AdWorks2go als Installation der Software AdWorks 182
8.46 Sicherheitsdiagramm für AdWorks2go 183
8.47 Aktivierung des Moduls Online-Depoteinsicht 186
8.48 Modell einer Geschäftsstelle in AdWorks2go 186
8.49 Verschiedene Designs der AdWorks2go iPhone-Anwendung . . 189
8.50 Verschiedene Versionen des Konfigurationsserver 191

8.51 Modell der Geschäftsstellenhierarchie in AdWorks2go 192
8.52 Mehrfachzuordnung eines Beraters durch die Geschäftsstellenhierarchie in AdWorks2go 192
8.53 Evaluationsbogen für die Evaluation des CUSTODIAN-Frameworks 196
8.54 MTF 1.0: Der Kern der Meta Threat Facility 199
8.55 MTF Legende: Farbnotationen für Gefahrenklassen der MTF . 200
8.56 MTF 1.1: MTF nach der HSRZ-Fallstudie 200
8.57 MTF 1.2: MTF nach der RBG-Fallstudie 201
8.58 MTF 1.3: MTF nach der AdWorks2go-Fallstudie 201

A.1 Sicherheitsdiagramm des HSRZ 217
A.2 Servertaxonomie im HSRZ 218
A.3 Loginmaske des Citrix Terminals 220
A.4 Der Token stellt den Faktor Besitz bei der Authentifizierung bereit 221
A.5 Serverübersicht des Citrix Terminals 221
A.6 Personenbasisdaten im AD 222
A.7 Hierarchische Strukturierung von Servern im Rechenzentrum . 224
A.8 Taxonomie der Akzessoren des HSRZ 226
A.9 Sicherheitsdiagramm der Netzwerkdienstleisters RBG 229
A.10 Auszug aus einer Konfiguration einer logischen Firewall 229
A.11 Eintrag eines Servers in der Host-DB 230
A.12 Sicherheitsdiagramm für AdWorks2go 231
A.13 Verschiedene Versionen des Konfigurationsserver 232
A.14 Einstellungen für einen Berater im Lizenzmanager 232
A.15 Auszug von Logeinträgen in der Logging-DB 233

Tabellenverzeichnis

3.1 Gefahren nach Quelle und Verursacher 34
3.2 Sicherheitsdimensionen nach Loch 35
3.3 Gefahren nach den Loch'schen Sicherheitsdimensionen 36

6.1 Anwendungsfälle von CUSTODIAN und deren Umsetzung in KUSTOS 96

7.1 Metrikendefinitionen $G(d)$, $E(d)$ und $B(d)$ der Diagrammtypen 110
7.2 Metriken $G(d)$, $E(d)$ und $B(d)$ für verschiedene Diagrammtypen 114
7.3 Zugriffsmatrix der Komponenten eines verteilten Systems . . . 115
7.4 Beispiele für Modelltransformationen mittels Transitionsmodell 116
7.5 Beispiel einer Zugriffskontrollliste eines verteilten Systems . . 117
7.6 Entscheidungstabelle für Maßnahmen nach dem Test 123

8.1 Zugriffsmatrix der Komponenten im HSRZ 132
8.2 Zusammenfassung aller Analysen der Berechtigungsausläufe . 139
8.3 Übersicht über Test-, Hypothesen-, und Invalidationsmengen . 151
8.4 Zugriffsmatrix der Komponenten der RBG-Netzwerkdienste . . 173
8.5 Identifizierte Generalisierung der einzelnen Lehrstühle 179
8.6 Identifizierte Generalisierung der einzelnen Lehrstühle 180
8.7 Zugriffsmatrix der Komponenten bei AdWorks2go 184
8.8 Gegenüberstellung der Forschungshypothesen mit den Fragen 195
8.9 Antworten der einzelnen Umfrageteilnehmer 198
8.10 Auswertung der Fragen 1 bis 6 des Fragebogens 199
8.11 Überblick über die Gefahrenklassen und Gefahren aller Fallstudien 202

A.1 Erforderliche Konfigurationsdaten des verteilten Systems . . . 227
A.2 Rahmenbedingungen für die Beschaffung der Konfigurationen 227
A.3 Konfigurationsdaten der Gefahrenklasse Witwe 228

Literaturverzeichnis

[AAB+13] Andrienko, Gennady ; Andrienko, Natalia ; Bak, Peter ; Keim, Daniel ; Wrobel, Stefan: *Visual analytics of movement*. Springer Science & Business Media, 2013

[AAW07] Andrienko, Gennady ; Andrienko, Natalia ; Wrobel, Stefan: Visual analytics tools for analysis of movement data. In: *ACM SIGKDD Explorations Newsletter* 9 (2007), Nr. 2, S. 38–46

[ABCC08] Alfaro, Joaquin G. ; Boulahia-Cuppens, Nora ; Cuppens, Frédéric: Complete analysis of configuration rules to guarantee reliable network security policies. In: *International Journal of Information Security* 7 (2008), Nr. 2, S. 103–122

[Abi00] Abie, Habtamu: An overview of firewall technologies. In: *Telektronikk* 96 (2000), Nr. 3, S. 47–52

[AH09] Antoniou, Grigoris ; Harmelen, Frank van: Web ontology language: OWL. In: *Handbook on ontologies*. Springer, 2009, S. 91–110

[BCHK07] Brügge, Bernd ; Creighton, Oliver ; Helming, Jonas ; Kögel, Maximilian: Unicase–an Ecosystem for Unified Software Engineering Research Tools. In: *Third IEEE International Conference on Global Software Engineering, ICGSE* Bd. 2008 Citeseer, 2007

[BD04] Brügge, Bernd ; Dutoit, Allen H.: *Objektorientierte Softwaretechnik: mit UML, Entwurfsmustern und Java*. Pearson Studium, 2004

[BELW80] Balzer, Robert ; Erman, Lee D. ; London, Philip ; Williams, Chuck: HEARSAY-II: A Domain-Independent Framework for Expert Systems. In: *AAAI* Bd. 1, 1980, S. 108–110

[Ben13] Bengel, Günther: *Verteilte Systeme: client-server-computing für Studenten und Praktiker*. Springer-Verlag, 2013

[Bin96] Binder, Robert V.: Testing object-oriented software: a survey. In: *Software Testing, Verification and Reliability* 6 (1996), Nr. 3-4, S. 125–252

[BKW12] Brügge, Bernd ; Krusche, Stephan ; Wagner, Martin: Teaching Tornado: from communication models to releases. In: *Proceedings of the 8th edition of the Educators' Symposium* ACM, 2012, S. 5–12

[BLHL+01] Berners-Lee, Tim ; Hendler, James ; Lassila, Ora u. a.: The semantic web. In: *Scientific american* 284 (2001), Nr. 5, S. 28–37

[BMH06] Burke, Bill ; Monson-Haefel, Richard: *Enterprise JavaBeans 3.0*. O'Reilly Media, Inc., 2006

[BMS+10] Babar, Sachin ; Mahalle, Parikshit ; Stango, Antonietta ; Prasad, Neeli ; Prasad, Ramjee: Proposed security model and threat taxonomy for the internet of things (IoT). In: *Recent Trends in Network Security and Applications*. Springer, 2010, S. 420–429

[Bod06] Bodendorf, Freimut: Daten und Wissen. In: *Daten-und Wissensmanagement* (2006), S. 1–5

[BRJ11] Booch, Grady ; Rumbaugh, James ; Jacobson, Ivar: Unified Modeling Language (UML). (2011)

[Bus98] Buschmann, Frank: *Pattern-orientierte Software-Architektur: Ein Pattern-System*. Pearson Deutschland GmbH, 1998

[CB09] Chowdhury, NM Mosharaf K. ; Boutaba, Raouf: Network virtualization: state of the art and research challenges. In: *Communications Magazine, IEEE* 47 (2009), Nr. 7, S. 20–26

[CCS12] Chen, Hsinchun ; Chiang, Roger H. ; Storey, Veda C.: Business Intelligence and Analytics: From Big Data to Big Impact. In: *MIS quarterly* 36 (2012), Nr. 4, S. 1165–1188

[CDK02] Coulouris, George ; Dollimore, Jean ; Kindberg, Tim: *Verteilte Systeme: Konzepte und Design*. Addision-Wesley, 2002

[CDK05] Coulouris, George ; Dollimore, Jean ; Kindberg, Tim: *Distributed systems: concepts and design*. Pearson Education, 2005

[CFZA02] Covington, Michael J. ; Fogla, Prahlad ; Zhan, Zhiyuan ; Ahamad, Mustaque: A context-aware security architecture for emerging applications. In: *Computer Security Applications Conference, 2002. Proceedings. 18th Annual* IEEE, 2002, S. 249–258

[CL85] Chandy, K M. ; Lamport, Leslie: Distributed snapshots: determining global states of distributed systems. In: *ACM Transactions on Computer Systems (TOCS)* 3 (1985), Nr. 1, S. 63–75

[CLS+01] Covington, Michael J. ; Long, Wende ; Srinivasan, Srividhya ; Dev, Anind K. ; Ahamad, Mustaque ; Abowd, Gregory D.: Securing context-aware applications using environment roles. In: *Proceedings of the sixth ACM symposium on Access control models and technologies* ACM, 2001, S. 10–20

[CMA00] Covington, Michael J. ; Moyer, Matthew J. ; Ahamad, Mustaque: Generalized role-based access control for securing future applications. (2000)

[CMSW01] Cavazza, Marc ; Mead, Steven J. ; Strachan, Alexander I. ; Whittakeri, Alex: A Blackboard System for Interpreting Agent Messages. In: *Proceedings GameOn 2000: International Conference on Intelligent Games &Simulation*, 2001

[Coh87] Cohen, Fred: Computer viruses: theory and experiments. In: *Computers & security* 6 (1987), Nr. 1, S. 22–35

[Col12] Cole, Dave: Data center infrastructure management. In: *Data Center Knowledge* (2012)

[DCJ05] Dong, Jing ; Chen, Shanguo ; Jeng, Jun-Jang: Event-based blackboard architecture for multi-agent systems. In: *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on* Bd. 2 IEEE, 2005, S. 379–384

[DDLS01] Damianou, Nicodemos ; Dulay, Naranker ; Lupu, Emil ; Sloman, Morris: The ponder policy specification language. In: *Policies for Distributed Systems and Networks*. Springer, 2001, S. 18–38

[Deu80] Deutscher Bundestag: Geheimschutzordnung. In: *Geschäftsordnung des Deutschen Bundestages* (1980)

[Dey01] Dey, Anind K.: Understanding and using context. In: *Personal and ubiquitous computing* 5 (2001), Nr. 1, S. 4–7

[Dij72] Dijkstra, Edsger W.: Structured Programming. London, UK : Academic Press Ltd., 1972, Kapitel Chapter I: Notes on Structured Programming, S. 1–82

[DIN87] Norm DIN VDE 31000 Dezember 1987. *Allgemeine Leitsätze für das sicherheitsgerechte Gestalten von Produkten*

[DIN94] Norm DIN 19226 Februar 1994. *Leittechnik; Regelungstechnik und Steuerungstechnik; Begriffe zum Verhalten dynamischer Systeme*

[DKA+14] Durumeric, Zakir ; Kasten, James ; Adrian, David ; Halderman, J A. ; Bailey, Michael ; Li, Frank ; Weaver, Nicolas ; Amann, Johanna ; Beekman, Jethro ; Payer, Mathias u. a.: The matter of Heartbleed. In: *Proceedings of the 2014 Conference on Internet Measurement Conference* ACM, 2014, S. 475–488

[DTM10] Dawoud, Wesam ; Takouna, Ibrahim ; Meinel, Christoph: Infrastructure as a service security: Challenges and solutions. In: *Informatics and Systems (INFOS), 2010 The 7th International Conference on* IEEE, 2010, S. 1–8

[EGPQ06] Elkind, Edith ; Genest, Blaise ; Peled, Doron ; Qu, Hongyang: Grey-box checking. In: *Formal Techniques for Networked and Distributed Systems-FORTE 2006*. Springer, 2006, S. 420–435

[EHRLR80] Erman, Lee D. ; Hayes-Roth, Frederick ; Lesser, Victor R. ; Reddy, D R.: The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. In: *ACM Computing Surveys (CSUR)* 12 (1980), Nr. 2, S. 213–253

[ESSD08] Easterbrook, Steve ; Singer, Janice ; Storey, Margaret-Anne ; Damian, Daniela: Selecting empirical methods for software engineering research. In: *Guide to advanced empirical software engineering*. Springer, 2008, S. 285–311

[EZ01] Eronen, Pasi ; Zitting, Jukka: An expert system for analyzing firewall rules. In: *Proceedings of the 6th Nordic Workshop on Secure IT Systems (NordSec 2001)*, 2001, S. 100–107

[FK92] Ferraiolo, David F. ; Kuhn, D R.: Role-Based Access Controls. In: *National Computer Security Conference* 15 (1992), Januar, S. 554–563

[FLR+98] Fuchs, Henry ; Livingston, Mark A. ; Raskar, Ramesh ; Keller, Kurtis ; Crawford, Jessica R. ; Rademacher, Paul ; Drake, Samuel H. ; Meyer, Anthony A. u. a.: *Augmented reality visualization for laparoscopic surgery*. Springer, 1998

[Fow02] Fowler, Martin: *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002

[FSG+01] Ferraiolo, David F. ; Sandhu, Ravi ; Gavrila, Serban ; Kuhn, D R. ; Chandramouli, Ramaswamy: Proposed NIST standard for role-based access control. In: *ACM Transactions on Information and System Security (TISSEC)* 4 (2001), Nr. 3, S. 224–274

[GCH03] Garg, Ashish ; Curtis, Jeffrey ; Halper, Hilary: Quantifying the financial impact of IT security breaches. In: *Information Management & Computer Security* 11 (2003), Nr. 2, S. 74–83

[GE97] Gershon, Nahum ; Eick, Stephen G.: Information visualization. In: *IEEE Computer Graphics and Applications* (1997), Nr. 4, S. 29–31

[GEMT06] Gutiérrez, Javier J. ; Escalona, María José ; Mejías, Manuel ; Torres, Jesús: An approach to generate test cases from use cases. In: *Proceedings of the 6th international conference on Web engineering* ACM, 2006, S. 113–114

[GHM+05] Greenberg, Albert ; Hjalmtysson, Gisli ; Maltz, David A. ; Myers, Andy ; Rexford, Jennifer ; Xie, Geoffrey ; Yan, Hong ; Zhan, Jibin ; Zhang, Hui: A clean slate 4D approach to network control and management. In: *ACM SIGCOMM Computer Communication Review* 35 (2005), Nr. 5, S. 41–54

[GI12] Gutsell, Jennifer N. ; Inzlicht, Michael: Intergroup differences in the sharing of emotive states: neural evidence of an empathy

gap. In: *Social cognitive and affective neuroscience* 7 (2012), Nr. 5, S. 596–603

[GKT89] Gunneflo, Ulf ; Karlsson, Johan ; Torin, Jan: Evaluation of error detection schemes using fault injection by heavy-ion radiation. In: *Fault-Tolerant Computing, 1989. FTCS-19. Digest of Papers., Nineteenth International Symposium on* IEEE, 1989, S. 340–347

[Gla13] Glas, Martin: *Ontology-based Model Integration for the Conceptual Design of Aircraft*, Technische Universität München, Dissertation, 2013

[GLLR05] Gordon, Lawrence A. ; Loeb, Martin P. ; Lucyshyn, William ; Richardson, Robert: *CSI/FBI computer crime and security survey*. Computer Security Institute San Francisco, CA, 2005

[GLM12] Godefroid, Patrice ; Levin, Michael Y. ; Molnar, David: SAGE: whitebox fuzzing for security testing. In: *Queue* 10 (2012), Nr. 1, S. 20

[GNSS06] Gabriel, Richard P. ; Northrop, Linda ; Schmidt, Douglas C. ; Sullivan, Kevin: Ultra-large-scale systems. In: *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications* ACM, 2006, S. 632–634

[Gra00] Grabowski, Jens: TTCN-3-A new Test Specification Language for Black-Box Testing of Distributed Systems. In: *Proceedings of the" 17th International Conference and Exposition on Testing Computer Software (TCS'2000), Theme: Testing Technology vs. Testers' Requirements", Washington DC, June*, 2000

[HA03] Ho, Purdy ; Armington, John: A dual-factor authentication system featuring speaker verification and token technology. In: *Audio-and Video-Based Biometric Person Authentication* Springer, 2003, S. 128–136

[HAS06] Hamed, Hazem ; Al-Shaer, Ehab: Taxonomy of conflicts in network security policies. In: *Communications Magazine, IEEE* 44 (2006), Nr. 3, S. 134–141

[HB99] Halpin, Terry A. ; Bloesch, Anthony C.: Data modeling in UML and ORM: a comparison. In: *J. Database Manag.* 10 (1999), Nr. 4, S. 4–13

[HG15] Harris, Mark ; Geng, Hwaiyu: Data Center Infrastructure Management. In: *Data Center Handbook* (2015), S. 601–618

[HKD07] Haeberlen, Andreas ; Kouznetsov, Petr ; Druschel, Peter: PeerReview: Practical accountability for distributed systems. In: *ACM SIGOPS Operating Systems Review* Bd. 41 ACM, 2007, S. 175–188

[Hoe01] Hoeken, Hans: Anecdotal, statistical, and causal evidence: Their perceived and actual persuasiveness. In: *Argumentation* 15 (2001), Nr. 4, S. 425–437

[Hof98] Hofte, GH t.: Working apart together: foundations for component groupware. In: *Telematica Instituut fundamental research series (ISSN 1388-1795* (1998), Nr. 001

[Hol10] Hollingworth, Harry L.: The central tendency of judgment. In: *The Journal of Philosophy, Psychology and Scientific Methods* 7 (1910), Nr. 17, S. 461–469

[How95] Howes, Timothy A.: The lightweight directory access protocol: X. 500 lite. (1995)

[HTI97] Hsueh, Mei-Chen ; Tsai, Timothy K. ; Iyer, Ravishankar K.: Fault injection techniques and tools. In: *Computer* 30 (1997), Nr. 4, S. 75–82

[HZ06] Hussein, Mohammed ; Zulkernine, Mohammad: UMLintr: a UML profile for specifying intrusions. In: *Engineering of Computer Based Systems, 2006. ECBS 2006. 13th Annual IEEE International Symposium and Workshop on* IEEE, 2006, S. 8–pp

[IKP+95] Ilgun, Koral ; Kemmerer, Richard ; Porras, Phillip u. a.: State transition analysis: A rule-based intrusion detection approach. In: *Software Engineering, IEEE Transactions on* 21 (1995), Nr. 3, S. 181–199

[iso05] ISO/IEC 27001: IT-Sicherheitsverfahren – Informationssicherheits-Managementsysteme – Anforderungen. Internationale Organisation für Normung, 2005

[IUB05] Ierace, Nick ; Urrutia, Cesar ; Bassett, Richard: Intrusion prevention systems. In: *Ubiquity* (2005), Nr. June, S. 2–2

[JCDZ11] Jiang, Li ; Chen, Hao ; Deng, Fei ; Zhong, Qiusheng: A security evaluation method based on threat classification for web service. In: *Journal of Software* 6 (2011), Nr. 4, S. 595–603

[Jür02] Jürjens, Jan: UMLsec: Extending UML for secure systems development. In: *UML 2002—The Unified Modeling Language*. Springer, 2002, S. 412–425

[KABW14] Krusche, Stephan ; Alperowitz, Lukas ; Brügge, Bernd ; Wagner, Martin O.: Rugby: an agile process model based on continuous delivery. In: *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering* ACM, 2014, S. 42–50

[KAF+08] Keim, Daniel ; Andrienko, Gennady ; Fekete, Jean-Daniel ; Görg, Carsten ; Kohlhammer, Jörn ; Melançon, Guy: *Visual analytics: Definition, process, and challenges*. Springer, 2008

[KC99] Kon, Fabio ; Campbell, Roy H.: Supporting automatic configuration of component-based distributed systems. In: *Proceedings of the 5th conference on USENIX Conference on Object-Oriented Technologies & Systems-Volume 5* USENIX Association, 1999, S. 13–13

[Kee13] Keet, C M.: Open World Assumption. In: *Encyclopedia of Systems Biology*. Springer, 2013, S. 1567–1567

[KELN10] Kuhn, Adrian ; Erni, David ; Loretan, Peter ; Nierstrasz, Oscar: Software cartography: Thematic software visualization with consistent layout. In: *Journal of Software Maintenance and Evolution: Research and Practice* 22 (2010), Nr. 3, S. 191–210

[KF13] Kim, Hyojoon ; Feamster, Nick: Improving network management with software defined networking. In: *Communications Magazine, IEEE* 51 (2013), Nr. 2, S. 114–119

[Kir13] Kirkpatrick, Keith: Software-defined networking. In: *Communications of the ACM* 56 (2013), Nr. 9, S. 16–19

[KKKJ06] Kals, Stefan ; Kirda, Engin ; Kruegel, Christopher ; Jovanovic, Nenad: Secubat: a web vulnerability scanner. In: *Proceedings of the 15th international conference on World Wide Web* ACM, 2006, S. 247–256

[KLM+97] Kiczales, Gregor ; Lamping, John ; Mendhekar, Anurag ; Maeda, Chris ; Lopes, Cristina ; Loingtier, Jean-Marc ; Irwin, John: *Aspect-oriented programming*. Springer, 1997

[Klo07] Klostermann, Tanja: *Optimierung kooperativer Dienstleistungen im technischen Kundendienst des Maschinenbaus*. Springer, 2007

[KM85] Kramer, Jeff ; Magee, Jeff: Dynamic configuration for distributed systems. In: *Software Engineering, IEEE Transactions on* (1985), Nr. 4, S. 424–436

[KNMC06] Klenk, Andreas ; Niedermayer, Heiko ; Masekowsky, Marcus ; Carle, Georg: An architecture for autonomic security adaptation. In: *Annales des télécommunications* Bd. 61 Springer, 2006, S. 1066–1082

[Kru16] Krusche, Stephan: *Rugby - A Process Model for Continuous Software Engineering*, Technische Universität München, Diss., 2016

[KV03] Kruegel, Christopher ; Vigna, Giovanni: Anomaly detection of web-based attacks. In: *Proceedings of the 10th ACM conference on Computer and communications security* ACM, 2003, S. 251–261

[Lam74] Lampson, Butler W.: Protection. In: *ACM SIGOPS Operating Systems Review* 8 (1974), Nr. 1, S. 18–24

[LBD02] Lodderstedt, Torsten ; Basin, David ; Doser, Jürgen: SecureUML: A UML-based modeling language for model-driven security. In: *UML 2002—The Unified Modeling Language*. Springer, 2002, S. 426–441

[LCJ87] Leblang, David B. ; Chase Jr, Robert P.: Parallel software configuration management in a network environment. In: *Software, IEEE* 4 (1987), Nr. 6, S. 28–35

[LCW92] Loch, Karen D. ; Carr, Houston H. ; Warkentin, Merrill E.: Threats to information systems: today's reality, yesterday's understanding. In: *MIS Quarterly* (1992), S. 173–186

[LHDL98] Liere, Robert van ; Harkes, Jan ; De Leeuw, Wim: A distributed blackboard architecture for interactive data visualization. In: *Proceedings of the conference on Visualization'98* IEEE Computer Society Press, 1998, S. 225–231

[Lik32] Likert, Rensis: A technique for the measurement of attitudes. In: *Archives of psychology* (1932)

[LLS+13] LaValle, Steve ; Lesser, Eric ; Shockley, Rebecca ; Hopkins, Michael S. ; Kruschwitz, Nina: Big data, analytics and the path from insights to value. In: *MIT sloan management review* 21 (2013)

[LMW05] Lankes, Joscf ; Matthes, Florian ; Wittenburg, André: Softwarekartographie: systematische darstellung von anwendungslandschaften. In: *Wirtschaftsinformatik 2005*. Springer, 2005, S. 1443–1462

[Lor11] Loretan, Peter: *Software cartography*, MS Thesis, University of Bern, Diss., 2011

[LW14] Li, Gang ; Wei, Mingchuan: Everything-as-a-service platform for on-demand virtual enterprises. In: *Information Systems Frontiers* 16 (2014), Nr. 3, S. 435–452

[MA01] Moyer, Matthew J. ; Abamad, Mustaque: Generalized role-based access control. In: *Distributed Computing Systems, 2001. 21st International Conference on.* IEEE, 2001, S. 391–398

[MAJ06] Masoumzadeh, Amir R. ; Amini, Morteza ; Jalili, Rasool: Context-aware provisional access control. In: *Information Systems Security*. Springer, 2006, S. 132–146

[MAJ07] Masoumzadeh, Amirreza ; Amini, Morteza ; Jalili, Rasool: Conflict detection and resolution in context-aware authorization.

In: *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on* Bd. 1 IEEE, 2007, S. 505–511

[Mat08] Matthes, Florian: Softwarekartographie. In: *Informatik-Spektrum* 31 (2008), Nr. 6, S. 527–536

[Met15] *Meta Object Facility (MOF) 2.5 Core Specification*. 2015. – Version 2.5

[MGP03] Manso, Ma E. ; Genero, Marcela ; Piattini, Mario: No-redundant metrics for UML class diagram structural complexity. In: *Advanced Information Systems Engineering* Springer, 2003, S. 127–142

[Mil56] Miller, George A.: The magical number seven, plus or minus two: some limits on our capacity for processing information. In: *Psychological review* 63 (1956), Nr. 2, S. 81

[MRF+13] Monsanto, Christopher ; Reich, Joshua ; Foster, Nate ; Rexford, Jennifer ; Walker, David u. a.: Composing Software Defined Networks. In: *NSDI*, 2013, S. 1–13

[MWC04] Mao, Yuxin ; Wu, Zhaohui ; Chen, Huajun: Semantic browser: an intelligent client for dart-grid. In: *Computational Science-ICCS 2004*. Springer, 2004, S. 470–473

[NAB+92] Nicolaides, KH ; Azar, G ; Byrne, D ; Mansur, C ; Marks, K: Fetal nuchal translucency: ultrasound screening for chromosomal defects in first trimester ofpregnancy. In: *Bmj* 304 (1992), Nr. 6831, S. 867–9

[NFG+06] Northrop, Linda ; Feiler, Peter ; Gabriel, Richard P. ; Goodenough, John ; Linger, Rick ; Longstaff, Tom ; Kazman, Rick ; Klein, Mark ; Schmidt, Douglas ; Sullivan, Kevin u. a.: Ultra-large-scale systems: The software challenge of the future / DTIC Document. 2006. – Forschungsbericht

[Nii86] Nii, H P.: Blackboard application systems, blackboard systems and a knowledge engineering perspective. In: *AI magazine* 7 (1986), Nr. 3, S. 82

[NN09] Nita, Marius ; Notkin, David: White-box approaches for improved testing and analysis of configurable software systems. In: *Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on* IEEE, 2009, S. 307–310

[NT03] Noble, James ; Tobkin, Chris: *Check Point NG VPN-1/FireWall-1 Advanced Configuration and Troubleshooting*. Syngress Publishing, 2003

[NW11] Nürnberger, A ; Wenzel, Constanze: Wisdom-the blurry top of human cognition in the dikw-model. In: *Proceedings of the EUSFLAT conference, Aix-Les-Bains, France* Bd. 1, 2011, S. 584–591

[OJ08] Owens Jr, James P.: *A study of passwords and methods used in brute-force SSH attacks*, Clarkson University, Diss., 2008

[OL10] Ott, Lyman ; Longnecker, Michael: *An introduction to statistical methods and data analysis*. Bd. 6. Brooks/Cole, 2010

[OUP04] Ono, Yuki ; Uchiyama, Hajime ; Potter, W: A mobile robot for corridor navigation: a multi-agent approach. In: *Proceedings of the 42nd annual Southeast regional conference* ACM, 2004, S. 379–384

[Pop63] Popper, Karl R.: Vermutungen und Widerlegungen: Das Wachstum der wissenschaftlichen Erkenntnis, Teilband I: Vermutungen. In: *Originalausgabe „Conjectures and Refutations “, London* (1963)

[PP06] Pfleeger, Charles P. ; Pfleeger, Shari L.: *Security in Computing*. 4th. Prentice Hall Professional Technical Reference, 2006. – ISBN 0132390779

[Pre94] Pree, Wolfgang: Meta patterns—A means for capturing the essentials of reusable object-oriented design. In: *Object-oriented programming*. Springer, 1994, S. 150–162

[PS03] Pras, Aiko ; Schönwälder, Juergen: On the difference between information models and data models / RFC 3444, January. 2003. – Forschungsbericht

[PV74] Popper, Karl R. ; Vetter, Hermann: *Objektive Erkenntnis*. Hoffmann und Campe Hamburg, 1974

[PY15] Pescatore, John ; Young, Greg: *Defining the next-generation firewall*. `http://bradreese.com/blog/palo-alto-gartner.pdf`, 2009 (letzter Zugriff: 11. Februar 2015)

[RB07] Rudenko, D ; Borisov, A: An overview of blackboard architecture application for real tasks. In: *Scientific Proceedings Of Riga Technical University, Ser* Bd. 5, 2007, S. 50–57

[RBG92] Reiter, Michael ; Birman, Kenneth ; Gong, Li: Integrating security in a group oriented distributed system. In: *Research in Security and Privacy, 1992. Proceedings., 1992 IEEE Computer Society Symposium on* IEEE, 1992, S. 18–32

[RH09] Runeson, Per ; Höst, Martin: Guidelines for conducting and reporting case study research in software engineering. In: *Empirical software engineering* 14 (2009), Nr. 2, S. 131–164

[RRR05] Rosling, Hans ; Rosling, Rönnlund A ; Rosling, Ola: New software brings statistics beyond the eye. In: *Statistics, Knowledge and Policy: Key Indicators to Inform Decision Making. Paris, France: OECD Publishing* (2005), S. 522–530

[Rub96] Rubin, Aviel D.: Independent one-time passwords. In: *computing Systems* 9 (1996), Nr. 1, S. 15–27

[sas92] Statement on Auditing Standards No. 70. American Institute of Certified Public Accountants, 1992

[SCFY96] Sandhu, Ravi S. ; Coyne, Edward J. ; Feinstein, Hal L. ; Youman, Charles E.: Role-based access control models. In: *Computer* (1996), Nr. 2, S. 38–47

[SEH13] Sommestad, Teodor ; Ekstedt, Mathias ; Holm, Hannes: The cyber security modeling language: A tool for assessing the vulnerability of enterprise system architectures. In: *Systems Journal, IEEE* 7 (2013), Nr. 3, S. 363–373

[SG96] Shaw, Mary ; Garlan, David: *Software architecture: perspectives on an emerging discipline*. Bd. 1. Prentice Hall Englewood Cliffs, 1996

[SH09] Scarfone, Karen ; Hoffman, Paul: Guidelines on firewalls and firewall policy - Recommendations of the National Institute of Standards and Technology. In: *NIST Special Publication* 800 (2009), S. 41

[Sha01] Sha, Lui: Using simplicity to control complexity. In: *IEEE Software* 18 (2001), Nr. 4, S. 20–28

[SHC99] Stiemerling, Oliver ; Hinken, Ralph ; Cremers, Armin B.: The EVOLVE tailoring platform: supporting the evolution of component-based groupware. In: *Enterprise Distributed Object Computing Conference, 1999. EDOC'99. Proceedings. Third International* IEEE, 1999, S. 106–115

[SHC+01a] Scherling, Mark ; Huynh, An-Ni ; Carlson, Mark ; Westerinen, Andrea ; Quinn, Bob ; Herzog, Shai ; Strassner, John ; Schnizlein, John: Terminology for Policy-Based Management. In: *Terminology* (2001)

[SHC01b] Stiemerling, Oliver ; Hallenberger, Michael ; Cremers, Armin B.: A 3D interface for the administration of component-based, distributed systems. In: *Autonomous Decentralized Systems, 2001. Proceedings. 5th International Symposium on* IEEE, 2001, S. 119–126

[SJN05] Squair, Timothy E. ; Jamhour, Edgard ; Nabhen, Ricardo C.: An RBAC-based policy information base. In: *Policies for Distributed Systems and Networks, 2005. Sixth IEEE International Workshop on* IEEE, 2005, S. 171–180

[SL02] Sloman, Morris ; Lupu, Emil: Security and management policy specification. In: *Network, IEEE* 16 (2002), Nr. 2, S. 10–19

[Slo94] Sloman, Morris: Policy driven management for distributed systems. In: *Journal of network and Systems Management* 2 (1994), Nr. 4, S. 333–360

[Sma88] Smaha, Stephen E.: Haystack: An intrusion detection system. In: *Aerospace Computer Security Applications Conference, 1988., Fourth* IEEE, 1988, S. 37–44

[SO02] Sarbanes, Paul ; Oxley, Mike: Sarbanes-Oxley Act - An Act to protect investors by improving the accuracy and reliability of corporate disclosures made pursuant to the securities laws, and for other purposes. (2002), Juli

[SO05] Sindre, Guttorm ; Opdahl, Andreas L.: Eliciting security requirements with misuse cases. In: *Requirements engineering* 10 (2005), Nr. 1, S. 34–44

[SS94] Sandhu, Ravi S. ; Samarati, Pierangela: Access control: principle and practice. In: *Communications Magazine, IEEE* 32 (1994), Nr. 9, S. 40–48

[ssa10] Statement on Standards for Attestation Engagements 16. American Institute of Certified Public Accountants, 2010

[Ste03] Steen, Maarten van: Distributed Systems Principles and Paradigms. In: *Network* 3 (2003), S. 26

[Str03] Strassner, John: *Policy-based network management: solutions for the next generation*. Morgan Kaufmann, 2003

[Sve15] Svensson, Carl: Threat modelling of historical attacks with CySeMoL. (2015)

[TC+06] Thomas, James J. ; Cook, Kristin u. a.: A visual analytics agenda. In: *Computer Graphics and Applications, IEEE* 26 (2006), Nr. 1, S. 10–13

[Tic15] Tichy, Walter: *Software Configuration Management Overview*. `http://www.ida.liu.se/~petfr/princprog/cm.pdf`, 1988 (letzter Zugriff: 24. Februar 2015)

[Tur36] Turing, Alan M.: On computable numbers, with an application to the Entscheidungsproblem. In: *J. of Math* 58 (1936), Nr. 345-363, S. 5

[TVR85] Tanenbaum, Andrew S. ; Van Renesse, Robbert: Distributed operating systems. In: *ACM Computing Surveys (CSUR)* 17 (1985), Nr. 4, S. 419–470

[TVS07] Tanenbaum, Andrew S. ; Van Steen, Maarten: *Distributed systems*. Prentice Hall, 2007

[Uni10] *OMG Unified Modeling Language (OMG UML), Infrastructure*. 2010. – Version 2.3

[Ven01] Venkateswaran, Ramachandran: Virtual private networks. In: *Potentials, IEEE* 20 (2001), Nr. 1, S. 11–15

[VK02] Vigna, Giovanni ; Kemmerer, Richard A.: Intrusion detection: a brief history and overview. In: *Computer* 35 (2002), Nr. 4, S. 0027–30

[Voa97] Voas, Jeffrey: Fault injection for the masses. In: *Computer* 30 (1997), Nr. 12, S. 129–130

[Vol36] Vollmer, Wilhelm: *Wörterbuch der Mythologie aller Nationen: Mit 129 Tafeln*. Hoffmannsche Verlagsbuchhandlung, 1836

[Wag14] Wagner, Martin Otto W.: ACCD - Access Control Class Diagram. In: *Software Engineering 2014, Fachtagung des GI-Fachbereichs Softwaretechnik*, 2014, S. 131–136

[WD05] Wolf, Timo ; Dutoit, Allen H.: *Supporting traceability in distributed software development projects*. na, 2005

[Wel04] Wellings, Andrew: Concurrent and Real-Time Programming in Java. (2004)

[Whi03] Whitman, Michael E.: Enemy at the Gate: Threats to Information Security. In: *Communications of the ACM* 46 (2003), August, Nr. 8, 91–95. http://dx.doi.org/10.1145/859670.859675. – DOI 10.1145/859670.859675. – ISSN 0001–0782

[WHS+10] Wendler, Thomas ; Herrmann, Ken ; Schnelzer, Andreas ; Lasser, Tobias ; Traub, Joerg ; Kutter, Olivier ; Ehlerding, Alexandra ; Scheidhauer, Klemens ; Schuster, Tibor ; Kiechle, Marion u. a.: First demonstration of 3-D lymphatic mapping in breast cancer

using freehand SPECT. In: *European journal of nuclear medicine and molecular imaging* 37 (2010), Nr. 8, S. 1452–1461

[Wik15] Wikimedia, Foundation I.: *System*. `http://de.wikipedia.org/wiki/System`, 2015 (letzter Zugriff: 9. Oktober 2015)

[Wil85] Wilkes, Maurice: *Memoirs of a computer pioneer*. Massachusetts Institute of Technology, 1985

[WLC04] Wullems, Christian J. ; Looi, Mark H. ; Clark, Andrew J.: Towards context-aware security: An authorization architecture for intranet environments. (2004)

[WM10] Whitman, Michael E. ; Mattord, Herbert J.: The Enemy is Still at the Gates: Threats to Information Security Revisited. In: *2010 Information Security Curriculum Development Conference*. New York, NY, USA : ACM, 2010 (InfoSecCD '10). – ISBN 978–1–4503–0202–9, 95–96

[Woo04] Wool, Avishai: A quantitative study of firewall configuration errors. In: *Computer* 37 (2004), Nr. 6, S. 62–67

[WY15] Wu, Yongzheng ; Yap, Roland H.: Simple and Practical Integrity Models for Binaries and Files. In: *Trust Management IX*. Springer, 2015, S. 30–46

[ZE08] Zambrano, Raul N. ; Engelhardt, Yuri: Diagrams for the masses: Raising public awareness–from Neurath to Gapminder and Google Earth. In: *Diagrammatic Representation and Inference*. Springer, 2008, S. 282–292

[ZE+11] Zikopoulos, Paul ; Eaton, Chris u. a.: *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011

[ZHM97] Zhu, Hong ; Hall, Patrick A. ; May, John H.: Software unit test coverage and adequacy. In: *Acm computing surveys (csur)* 29 (1997), Nr. 4, S. 366–427

[ZP03] Zhang, Guangsen ; Parashar, Manish: Dynamic context-aware access control for grid applications. In: *Grid Computing, 2003. Proceedings. Fourth International Workshop on* IEEE, 2003, S. 101–108