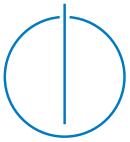
Chair of Network Architectures and Services
Department of Informatics
Technical University of Munich

Incident Handling Systems with Automated Intrusion Response

Nadine Herold

Dissertation

Network Architectures and Services NET-2017-05-2



TECHNISCHE UNIVERSITÄT MÜNCHEN
Institut für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste

Incident Handling Systems with Automated Intrusion Response

Nadine Herold, M.Sc.

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:		Prof. Bernhard Brügge, Ph.D.
Prüfer der Dissertation:	1.	Prof. Dr.-Ing. Georg Carle
	2.	Prof. Dr.-Ing. Tanja Zseby

Die Dissertation wurde am 12.12.2016 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 19.04.2017 angenommen.

Cataloging-in-Publication Data
Nadine Herold
Incident Handling Systems with Automated Intrusion Response
Dissertation, Mai 2017
Network Architectures and Services, Department of Computer Science
Technische Universität München

ISBN 978-3-937201-59-7
ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)
DOI 10.2313/NET-2017-05-2
Network Architectures and Services NET-2017-05-2
Series Editor: Georg Carle, Technische Universität München, Germany

ACKNOWLEDGMENTS

Have you tried turn it off and on again?

The IT-Crowd

I am grateful for the support and help I received from certain individuals without whom this work would not have been possible. I would like to take the opportunity to express my gratitude.

First of all, I would like to thank Prof. Dr.-Ing. Georg Carle for giving me the opportunity to join the Chair for Network Architectures and Services and for supervising the dissertation by giving helpful advices and feedback along the way. I would also like to thank Prof. Dr.-Ing. Tanja Zseby for her feedback on my work and being my second assessor and Prof. Bernhard Brügge for chairing the examination committee.

I would like to thank my co-authors for the valuable feedback, collaboration and worthwhile criticism that helped to improve the work we have done together. A sincere thank you goes to all of them, in particular Stephan-A. Posselt, Dr. rer. nat. Matthias Wachs and Dr. rer. nat. Holger Kinkelin.

Most parts of this thesis were elaborated in the context of the research projects Systemic Security for Critical Infrastructures (SURF) and Decentralized Anomaly Detection (DecADe). Both projects has been supported by the German Federal Ministry of Education and Research (BMBF) under support code 16KIS0145 (SURF) and 16KIS0538 (DecADe). Additionally, some parts of this work are done in collaboration with Airbus Group Innovations (AGI). I want to express my gratitude to the individuals working at AGI supported this thesis.

Last but not least, I thank my family and friends supporting me over several years, always having a sympathetic ear for my complaints and being on hand with help and advice anytime.

ABSTRACT

Computer networks are permanently affected from attacks. In addition to hardening and prevention mechanisms, active *incident handling* is required to ensure resilience during an on-going attack. Handling security incidents in real-time is a complex process consisting of three steps: intrusion detection, alert processing and intrusion response. Those steps can be further divided into tasks encapsulated into modules. For effective and automated incident handling, a holistic approach covering the overall process and tightly intertwined steps are required. Most existing solutions for incident handling merely focus on a dedicated step instead of taking the overall process into account. Within this thesis, the question of how to design and integrate incident handling as a holistic and intertwined process into computer networks is addressed.

This thesis proposes a holistic Incident Handling System (IHS) that allows interleaved and collaborative interaction between the incident handling modules. This is achieved by providing a holistic information model covering all needed information from all steps as foundation for the collaboration of the modules. The interaction of those modules is realized using the Blackboard Pattern to structure the incident handling process and derive an appropriate execution model. The information and the execution model as well as the partitioning of the incident handling process into modules are aligned with each other such that the modules can solve the holistic problem of incident handling following a collaborative and interference-free strategy.

The contribution of this thesis is – besides the overall IHS – the development of the single modules needed for effective incident handling. As basis for automated intrusion response, a reliable Intrusion Detection System (IDS) is needed. We propose an anomaly-based IDS based on Complex Event Processing (CEP), producing only useful alerts for intrusion response. To answer the question of, which responses are suitable to counteract a security incident, possible responses are examined and categorized using available taxonomies. In order to determine an appropriate point in time to start automated intrusion response, possible triggers based on our information model are examined and combination strategies are proposed. To answer the question of which set of available responses is the optimal one, we provide a response selection approach based on Mixed Integer Linear Programming (MILP) and provide a proven optimal combination of responses. Finally, we investigate the question of how to execute responses on the target system. Therefore, we propose a module to automatically generate a response plan for structured execution and dynamic deployment of the selected responses.

To show the applicability of our holistic IHS, the overall system as well as the individual modules are implemented and evaluated. Therefore, we provide different simulations and an example use case based on a data cabin network using the SOME/IP protocol. With our evaluation we investigate the applicability and performance of the proposed IHS for different network settings and attack patterns and show the benefits of integrated, automated intrusion response.

ZUSAMMENFASSUNG

Vernetzte IT-Systeme unterliegen ständigen Angriffen. Zusätzlich zu Härtung und Präventionsmaßnahmen ist eine aktive Behandlung von Sicherheitsvorfällen notwendig, um die Resilienz dieser IT-Systeme auch während eines Angriffs zu erhalten. Eine Echtzeitbehandlung ist ein komplexer Prozess der sich aus der Erkennung, Analyse und Reaktion zusammensetzt und sich in weitere feinere Aktivitäten zerlegen lässt. Für eine effektive und automatisierte Behandlung von Sicherheitsvorfällen ist eine umfassende Sicht auf alle Teilprozesse, sowie deren Verzahnung ineinander notwendig. Bisherige Ansätze fokussieren sich meist nur auf einen Teilprozess anstatt den vollständigen Prozess zu betrachten. In der vorliegenden Arbeit wird daher die Frage untersucht, wie ein umfassender Prozess für die automatisierte Behandlung von Sicherheitsvorfällen gestaltet, umgesetzt und in bestehende IT-Systeme integriert werden kann.

In dieser Arbeit wird ein umfassendes Incident Handling System (IHS) vorgestellt, welches die Kollaboration der einzelnen Aktivitäten ermöglicht. Hierfür wird ein umfassendes Informationsmodell vorgestellt, welches alle Informationen, die für die einzelnen Teilprozesse benötigt werden, abdeckt. Dieses Informationsmodell legt die Grundlage für das vorgestellte Ausführungsmodell, welches auf dem Blackboard Pattern basiert, und die kollaborative Zusammenarbeit der einzelnen Module für die jeweiligen Aktivitäten ermöglicht. Das Informationsmodell, sowie die Strukturierung der Module sind auf einander abgestimmt, sodass die Behandlung von Sicherheitsvorfällen kollaborativ und frei von Konflikten durchgeführt werden kann.

Neben der Erstellung des IHS werden in dieser Arbeit die einzelnen Module umgesetzt. Als Basis für die aktive Behandlung von Sicherheitsvorfällen ist ein zuverlässiges Intrusion Detection System (IDS) notwendig. Im Rahmen dieser Arbeit wird ein Anomaliebasierter Ansatz unter Zuhilfenahme von Complex Event Processing (CEP) Technologien vorgestellt. Mögliche Handlungsoptionen für ein IHS werden im Rahmen dieser Arbeit untersucht und anhand bestehender Taxonomien klassifiziert. Die Entscheidung, wann automatisierte Reaktionen eingeleitet werden sollen, wird mit Hilfe eines Regel-basierten Systems umgesetzt. Für die optimale Zusammenstellung möglicher Reaktionen wird in dieser Arbeit ein Auswahlmechanismus basierend auf Mixed Integer Linear Programming (MILP) vorgestellt. Für die eigentliche Ausführung konkreter Reaktionen im IT-System wird im Rahmen dieser Arbeit eine Beschreibungssprache für die strukturierte Ausführung vorgestellt, die auf dem Zielsystem ausgeführt werden kann.

Um die Anwendbarkeit des vorgestellten IHS sowie der einzelnen Module zu zeigen, wurden alle benötigten Komponenten implementiert und evaluiert. Neben geeigneten Simulationen wurde zudem ein exemplarischer Anwendungsfall, basierend auf einem Kabinennetz unter Verwendung des SOME/IP Protokolls untersucht. Im Rahmen der Evaluation wurde die Performanz des IHS für unterschiedliche Netzwerkkonfigurationen und Angriffsmuster untersucht und der Nutzen des vorgestellten IHS aufgezeigt.

TABLE OF CONTENTS

1. INTRODUCTION

Computer networks are permanently affected by attack attempts. When hardening and prevention mechanisms fail, active *incident handling* is required to ensure service continuity and resilience during an on-going attack. „The history of cybersecurity shows that flawless intrusion prevention and perfectly accurate intrusion detection are practically impossible.“[6] Due to non-interoperable prevention systems and ill-equipped state-of-the-art Intrusion Detection Systems (IDS) raising high numbers of false positives and negatives the need for actively defending computer networks arise [6].

Incident handling is a complex process consisting of comprehensive steps: The underlying infrastructure to protect (target system) is monitored continuously to observe its healthiness. *IDSes* search for signs of intrusions and raise alerts if a security incident is detected. Those alerts are processed and analyzed to find the root cause of an intrusion and extract more meaningful information. This root cause is mitigated by *Intrusion Response Systems (IRS)* with suitable responses and identified security flaws are closed.

As nowadays computer networks are complex and the number and variety of security incidents is constantly rising, incident handling has to be automated. „[. . .] [Computer] systems [. . .] have reached a level of complexity and the attacks directed at them a level of sophistication that manual responses are no longer adequate.“[50] Huge and complex computer networks comprised of heterogeneous subsystems, different operating systems, vendor specific components, and a large number of diverse devices complicate manual interaction with those networks and impede a comprehensive analysis done by administrators in case of security incidents.

Growing distribution of services provides an additional challenge and further increases network complexity. Cloud services that have to be integrated within the own network infrastructure, the increasing number of users having their own mobile devices, or remote access for largely distributed power plants are some examples for increasing distribution of networks. „However, in order to meet the challenges of continuously available trustworthy services from today’s distributed systems, intrusion detection needs to be followed by response actions.“[50]

„In addition, the absence of automated response actions can be overwhelming, especially when dealing with multiple incidents at once.“[100] Automation allows quicker and less error-prone intervention than manual interaction, stops attacks and prevent the target system from further damage [123, 168, 172]. „The longer it takes to detect events and accurately respond to them, the more damage the organization sustains.“[18] Apart from quicker and efficient responses, automated intrusion response capabilities have additional benefits, like „responding to incidents systematically so that the appropriate actions are taken[,] [. . .] use information gained during incident handling to better prepare for handling future incidents and to provide stronger protection for systems and data [. . .] [and] dealing properly with legal issues that may arise during incidents.“[23]

1.1 Problem Statement and Research Questions

Automated intrusion response is a key element to effectively defend a target system under attack. Containing the attack, stopping further spreading, and preventing further damage ensures service continuity and resilience even during an attack. Incident handling combines intrusion detection, alert processing and intrusion response into a continuous process to better cope with security incidents and improve IT security as reaction times are shortened. Especially, systems relying on providing continuous services, like critical infrastructures, benefit from a continuous process of incident handling. Automating this process as a whole is challenging as multiple aspects, different functionality and comprehensive views have to be intertwined tightly to provide cooperative and continuous incident handling.

Considerable effort has been made to improve the single steps of incident handling. However, a comprehensive solution interleaving single steps and gain a broad view is missing. Existing partial solutions for incident handling lack in possibilities to interleave those solutions. Information required in all steps has to be recollected instead of sharing it in advance. Intermediate results helpful in subsequent steps are lost as only final results are passed. To enable automated, useful incident handling a collaborative execution model is required. The overall **research objective** of this thesis is

How to increase security of computer networks by integrating automated intrusion response into a holistic Incident Handling System (IHS)?

This overall research objective can be divided further into research questions that are discussed in more detail in the following.

Research Question RQ1 – What are requirements for an IHS? To answer this research question not only requirements with respect to the overall system but from all single components are of interests. Requirements for the overall system have to ensure compatibility between all needed single steps of the incident handling process. Functional as well as non-functional requirements have to be determined in order to provide an effective and useful IHS.

Research Question RQ2 – How to integrate all single steps into an IHS in a continuous manner? To integrate the single steps of incident handling into a holistic and continuous process, a suitable execution model is needed that can couple the single steps of incident handling. This execution model has to follow the requirements identified to answer Research Question RQ 1. The need to process all steps of incident handling in a continuous way arises from the need that during an ongoing attack the IHS has to react to the detected security incident in order to prevent further damage. Waiting times, discontinuities or interruptions will slow down the system and increase the chance for a successful attack.

Research Question RQ3 – What information elements are required to support all steps of incident handling? Besides the execution model required according to Research Question RQ 2, a suitable information model is required. The single steps of incident handling rely at least partially on the same information. In order to tightly interleave the single steps of incident handling into a comprehensive process, the single steps have to

follow a joint information model to be integrable. The information model has to provide all knowledge that is needed for the overall process and couple the needed information elements to allow cooperative interaction between the steps of incident handling.

Research Question RQ4 – How to reliably identify attacks in a stream of packets? As the detection of a security incident is the first step to enable automated intrusion response, detection has to be as early and reliable as possible. Detecting attacks within a stream of network packets is challenging as an attack can be detected as such not until corresponding packets are identified. For example the absence of a network packet may indicate an attack recently. Checking for the non-existence of packets within a stream is one aspect of this research question. Additionally, reassembling connections and communication between multiple network elements needs to be discussed.

Research Question RQ5 – How to determine when to react? As IDSes produce false positives and false negatives reacting on single alerts produced by an IDS is not suitable. Additionally, alerts may be correlated with each other or describe the same security incident. Therefore, a reasonable processing of alerts raised by IDSes is needed. Furthermore, determining the correct point in time when intrusion response has to be triggered is challenging. Reacting to early may result in an ineffective choice of responses while reacting to late will result in unnecessary damage. A continuous way of processing alerts and coupling alert processing with the trigger of intrusion response is inevitable.

Research Question RQ6 – What are suitable response options and how to identify them? After identifying a security incident and triggering intrusion response, the response options have to be determined. Hereby, the current network situation has to be taken into account. This includes amongst others the identification of suitable responses with respect to the detected attack and the network capabilities with respect to deployed and available responses.

Research Question RQ7 – How to assess and select an optimal set of responses from multiple response options? Given a suitable set of response options the subset of those options that is most effective under the current situation has to be selected. As a security incident may effect multiple victims, a combination of responses is needed. Responses may conflict with each other, such that those responses cannot be used in conjunction. The challenge while answering this research question is to determine a suitable model that can represent relations between responses themselves and with other related entities.

Research Question RQ8 – How to deploy and execute the selected set of responses within the target system? After determining the set of responses that has to be executed to cope with the security incident those responses have to be deployed and launched. Therefore, an adequate description language to define the control flow is needed to appropriately schedule responses and respect interdependencies between responses. Response deployment is a challenge as well, as responses may be applicable to multiple network entities and implementations may change. This last step of the incident handling process is currently neglected in existing approaches.

1.2 Contributions and Chapter Overview

The contribution of this thesis is the study and development of a comprehensive, collaborative and distributed IHS consisting of the following single steps: intrusion detection, alert processing and intrusion response. The IHS provides an execution and information model allowing to tightly interleave those single steps and enables continuous processing during an ongoing attack. As the focus is mainly on automated intrusion response capabilities, the most important functionality within intrusion response are covered in more detail to provide enhanced and automated intrusion response.

The contributions, including the mapping to the research questions raised in Section 1.1, and the structure of this thesis are summarized in Table 1.1. The structure of this thesis follows the research questions introduced in Section 1.1. The contribution to answer a research question can be a model, e.g. an execution model or information model, a simulation that is based on the implementation of a specific part, or a use case, e.g. a case study. In case parts of the presented work are already published, the citation is given for the respective chapter.

Tab. 1.1: Contributions and Structure of this Thesis

Chapter	Contributions and Research Questions	Model	Implementation	Simulation	Use Case	Paper
	RQ1, **RQ2**, **RQ3** – Proposed IHS					
3	Collaborative and distributed IHS using a common information model	•				[62]
	RQ4 – Intrusion Detection					
4	IDS based on Complex Event Processing (CEP)	•	•	•	•	[63]
	RQ5, **RQ6** – Suitable Responses and Triggers					
5	Response identification including a collection of suitable responses	•	•		•	
	RQ7 – Response Selection Strategies					
6	Response selection based on Mixed Integer Linear Programming (MILP)	•	•	•		[64]
	RQ8 – Response Execution and Deployment					
7	Response execution including a response plan description language	•	•			[162]
	RQ2, **RQ3** – Proposed IHS					
8	Overall system implementation and evaluation		•	•	•	[62]

Chapter 1 gives an introduction into this thesis. The research objective is examined and conducted research questions are discussed. The contributions of this thesis are presented and mapped to the research questions and the document structure.

Chapter 2 provides the background for this thesis. Basic terms as well as a short introduction into the single steps of incident handling namely intrusion detection, alert processing and intrusion response is given.

Chapter 3 explains the overall design of the proposed IHS. Research Question **RQ1** is covered in this chapter as functional and non-functional requirements are examined and explained. The execution model of the system is provided and answers Research Question **RQ2**. To cover Research Question **RQ3**, the underlying information model is examined and mapped to the processes of incident handling. This chapter is focusing on the overall design of the IHS. Parts of this work are already published, cf. [62]. The following chapters single out the individual steps of incident handling.

Chapter 4 answers Research Question **RQ4**. We propose a Network-based Intrusion Detection System (NIDS) that is capable of extracting attacks from a stream of network packets and identifying attacks on specific protocols by analyzing the communication between network entities. The design of this NIDS is based on CEP to provide continuous processing. Parts of this work are already published, cf. [63].

In Chapter 5 we answer Research Question **RQ5** and **RQ6**. Possible responses are examined due to a literature search and structured according to relevant features and available taxonomies. Additionally, this chapter shows possibilities on how to trigger responses appropriately. The interfaces with the overall IHS are examined and the usage of the information model is shown.

Chapter 6 focuses on Research Question **RQ7**. We propose a response selection approach based on MILP optimization. Therefore, an appropriate model is introduced that covers relations between responses and other required entities. We provide an implementation and show the applicability of our approach by simulating different input settings. Hereby, we analyze the influencing factors of our system that increase the problem complexity. Additionally, interfacing with the overall IHS and the usage of the information model is shown. Parts of this work are already published, cf. [64].

Chapter 7 is concerned with Research Question **RQ8**. We provide a description language for a response plan that specifies the control flow of interacting responses and allows to specify the interdependencies between responses. Additionally, we provide a response execution framework that is capable of deploying and launching responses. We show, how this system interfaces with our overall IHS and how the information model is utilized. Parts of this work are already published, cf. [162].

Chapter 8 forges the bridge to Chapter 3 as the proposed overall IHS is evaluated. Therefore, the system described in Chapter 3 is implemented and all components from Chapter 5 to 7 are integrated. Using this implementation, the applicability of the proposed IHS is shown. Parts of this work are already published, cf. [62].

2. BACKGROUND

In this chapter the background for this thesis is examined. First, the basic terms are outlined in Section 2.1. Afterwards, an introduction into Intrusion Detection Systems (IDS) is given in Section 2.2. This includes an overview on IDSes as well as the standardized alert format, called *Intrusion Detection Message Exchange Format (IDMEF)*. Required processing steps for alerts generated by IDSes are examined in Section 2.3. Afterwards, classification methods as well as properties and characteristics of responses are examined in Section 2.4. The last section, Section 2.5, addresses Intrusion Response Systems (IRS) including a taxonomy to describe an IRS and the required steps of intrusion response.

2.1 Basic Terms

Computer security was defined by NIST [60] as „the protection afforded to an automated information system in order to attain the applicable objectives of preserving the *Confidentiality, Integrity and Availability (CIA)* of information system resources (includes hardware, software, firmware, information/data, and telecommunications)." Integrity, availability and confidentiality are often called *security properties*. This traditional security properties are extended by *authentication* and *non-repudiation* [90, 143].

A *security attack* is any malicious action that compromises those security properties [143]. Attacks can either be *active* or *passive* [131], whereat active attacks effect the *target system* directly and passive attacks are used to gain information about the target system. The source or potential of an attack refers to as *threat*, while *risk* describes the likelihood of a successful attack against the target system. A *vulnerability* describes a security flaw or weakness within the target system that increases the chance of a successful attack. A detected attack refers to as *security incident* that may contains multiple *security events* and can be comprised of multiple attack steps [131].

Intrusion Detection Systems (IDS) are so-called *security services* monitoring and analyzing the target system to detect security events. They may combine single security events to get a bigger picture of what is going wrong in the target system [131]. *Intrusion Prevention Systems (IPS)* defend attacks before they can harm the target system, e.g. by using different hardening mechanisms, access control or firewalls [90]. *Attack avoidance* is defined as additional security mechanism, whereas, gained information is made unusable for an attacker [90].

IDSes can be extended by *response* capabilities. This means that in case of detecting a security incident a response to counteract the security incident can be triggered. Responses are also referred to as *countermeasures* or *counteractions*. Those systems are called *Intrusion (Detection and) Response Systems (I(D)RS)*. The most simplistic response of an I(D)RS is to trigger an *alert* in case of a detected intrusion that describes the security incident. A common format for alerts is the *Intrusion Detection Message Exchange Format (IDMEF)* [35].

In general, an IDS can be modeled using the *Common Intrusion Detection Framework (CIDF)* [152] as shown schematically in Figure 2.1 with four components fulfilling spe-

cific roles. *Event boxes (E-boxes)* are generating events based on monitoring the target system's state. This may include monitoring log-files, network traffic or communication relations between single components of the target system. *Analysis boxes (A-boxes)* capture events from the event boxes to analyze them and trigger alerts in case of a security incident. A-boxes can take alerts as inputs for further analysis on a higher level, e.g. to correlate or aggregate alerts. *Data boxes (D-boxes)* simply store alerts or events for forensics or persistence reasons. *Response boxes (R-boxes)* are consuming alerts including actions to counteract a security incident.

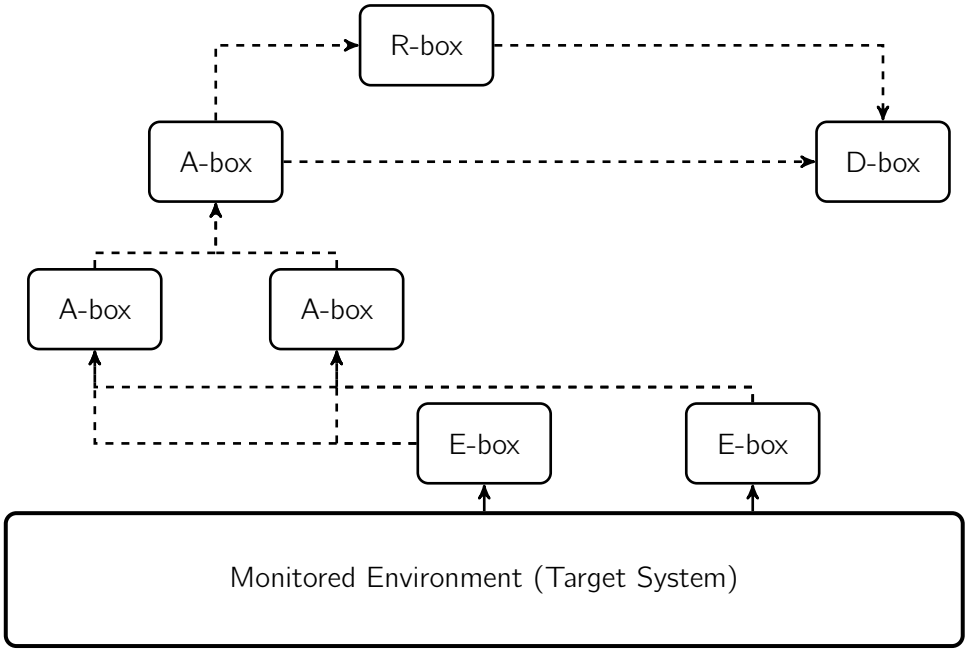


Fig. 2.1: General Structure of an IDS According to [152]

As the number of alerts triggered by an IDS may be enormous, *alert processing* is introduced to reduce the number of alerts and increase the meaning of a single alert. Within this work we refer to *incident handling* as the combination of the three main steps, namely *intrusion detection*, *alert processing* and *intrusion response*. A system providing all these single steps is referred to as *Incident Handling System (IHS)*.

2.2 Intrusion Detection Systems

Different methods for structuring and classifying IDSes are available [36, 90, 122, 143]. Among others the *location* and the *detection method* are functional characteristics for IDSes [36]. Both criteria are examined shortly in the following Subsections 2.2.1 and 2.2.2. As alerts play a central role within intrusion detection the common and standardized format *IDMEF* is described shortly in Subsection 2.2.3.

2.2.1 Host vs. Network-Based Detection

The first criterion IDSes can be divided into, is the *location* an IDS is placed.

A *Host-based Intrusion Detection System (HIDS)* is directly located on the host that is monitored. Such a HIDS has access to the system's resources and information available on a single host, e.g. log files. In case of an infection of the monitored host, the HIDS

may be compromised by an attacker having root access. The information, such a system can gain, is very detailed but limited to a single host instance. In case of multiple hosts in the target system, a HIDS has to be deployed on every single host [36, 90, 122, 143].

SAMHAIN [1] or OSSEC [2] are typical representatives of HIDSes. Both provide a server instance collecting information from agents deployed on and monitoring every single host.

Network-based Intrusion Detection Systems (NIDS) are deployed across the network and are monitoring the communication between hosts. They can gain a much more high-level view on what is going on in the network but have a limited insight into the single hosts deployed in the target system. An attacker has to attack the NIDS directly in order to compromise it, those IDSes are more isolated from the target system than HIDSes. A challenge for NIDSes is the placement of such systems as they need insights in all packets from the network communication. High-speed networks may be also a challenge to those systems as such networks enlarge the number of packets to inspect [36, 90, 122, 143].

Snort [3] and Bro [4] are typical representatives of NIDSes.

Hybrid solutions combine both approaches and analyze events from HIDSes as well as NIDSes to gather more insights into the target system [36].

2.2.2 Intrusion Detection Methods

The second criterion IDSes can be divided into is the method of intrusion detection. The main assumption of intrusion detection is that in case of an intrusion the observations differ from legitimate situations [143]. In [36, 90, 122, 143] the detection methods are split up differently that shows the difficulties arising from defining the scope of detection methods.

IDSes can be distinguished between *knowledge-based* and *behavior-based* methods [36]. The first group uses predefined knowledge for example signatures or rules. Gathering and maintaining this knowledge can be very difficult and time-consuming, but those systems provide a low false alert rate. For the second group the assumption is that an intrusion can be detected as derivation from the normal behavior that is learned during an initialization phase of the system. Those systems suffer from a high false alert rate, changing legitimate behavior and inappropriate learning sets may contain attacks. On the other hand side, those systems are capable of detecting new attacks without time consuming maintenance effort.

Kruegel et al. [90] use the definition of *misused-based* and *anomaly-based* methods. Whereat, misused-based systems follow the paradigm to specify the attack and look for signs that this attack had happened. This is done using signatures or models. Anomaly-based systems in contrast define the normal behavior and are searching for deviations. This may include that mis-configurations are treated as attacks.

Both descriptions overlap and do not clearly distinguish between the features a method provides. Therefore, the following differentiation will be used for clarification. The input of an IDS can be categorized as

- *Knowledge* that has to be available in advance, e.g. protocol specifications
- *Observations* that have been made about the target system, e.g. traffic dumps

The strategy of what an IDS is observing during its analysis can be categorized as

[1] http://www.la-samhna.de/samhain/
[2] http://ossec.github.io/
[3] https://www.snort.org/
[4] https://www.bro.org/

- Observing *normal* activities
- Observing *abnormal* activities

That will result in four different classes of detection methods with more precise features of provided methods.

2.2.3 Intrusion Detection Message Exchange Format (IDMEF)

The *Intrusion Detection Message Exchange Format* [35] defines a data format as well as exchange procedures for alerts triggered by IDSes to provide heterogeneous information among different systems interacting with each other. The format is given in `Extensible Markup Language (XML)` due to various reasons listed in [35].

An IDMEF message is either a *heartbeat* or an *alert*. A heartbeat provides information about the *analyzer* sending the heartbeat, the *creation time* as well as application-specific *additional data*. Whereas, the analyzer is the IDS.

An alert extends this information by the *detection time* of a security event, the *analyzer time*, information about *source* and *target*, the *classification* of the security event and the *assessment* information. The analyzer time is the time the analyzer noticed the event. Both, source and target, may include information regarding the *node*, *user*, *process* and *service* that can be determined.

As different analyzers have different detection capabilities, IDMEF defines so-called core classes as main parts of an alert, namely the analyzer, classification, source, target and additional data. To be compliant with the CIDF [152], the assessment field provides the option to define an action to be executed as a response to the security incident. More detailed information about the structure of IDMEF and provided information can be found in RFC 4765 [35].

2.3 Alert Processing

In [90] alert processing is defined as the „process that takes as input the alerts [...] and provides a more succinct and high-level view of occurring or attempted intrusions“. Single alerts are combined into *meta alerts* also referred to as *alert context* or *hyper alert* [106].

First, alerts from different IDSes have to be *collected*. Even if the deployed IDSes use IDMEF as alert format, they may provide information using different syntax and semantics. Therefore, all incoming alerts need to be *normalized*. Additionally, the alerts have to be *preprocessed* to determine the alert time, source and target of the alert as well as the attack type of the alert. For both steps, normalization and preprocessing, a so-called *ontology database* is utilized.

The next step in alert processing is alert *fusion*, also called *aggregation*, that combines duplicates of alerts. According to [90], alerts are fused in case the alert time differs within a predefined time span t and all other attributes of the alert except the analyzer are equal. In [176] alert aggregation is defined more generic as grouping alerts that have similar features. The features and the calculation for similarity are not further specified. As a measure of similarity the comparison of source, target and attack classification can be used [40]. According to [34] a combination of those features is sufficient for aggregation.

As false-positive reduction is essential when dealing with alerts, alert processing can be useful to undertake this task [27]. According to [90], an IDS can either send an alert in case of a successful attack, an unsuccessful attack attempt or a non-malicious event. The two later ones have to be filtered out by alert *verification*.

In the *attack-thread-reconstruction* phase multiple alerts are mapped to an attacker attacking a single target. This is done by merging alerts with the same source and target. In the *attack session reconstruction* host- and network-based alerts are linked together. A simple way to implement this is linking alerts such that first an attack on a host occurs that launches a network attack afterwards. In the *attack focus recognition* hosts being either source or target are identified. Attacks having a single source but multiple targets are referred to as *one2many* and attacks having multiple sources but a single target are referred to as *many2one* [90].

To find more high-level alert structures *multi-step correlation* is used [90]. Hereby, not the similarity of alerts is crucial but relations due to semantical meanings. A typical example is malware spreading over the network. Starting from a single host, attacks are launched to infect other hosts that for their part launch attacks again. Hereby, typical attack patterns consisting of individual attacks can be observed.

Within the *impact analysis* the effects of an attack on the target system is determined. Based on this impact alerts or alert contexts can be *prioritized* to identify the most urgent ones. The last step of alert processing is alert *sanitization* where sensitive data is removed using *pseudonymization* and *anonymization* [90].

2.4 Structuring Responses

Within this section, the goal is to achieve a closer look on responses. First, we examine possible classification methods for responses in Subsection 2.4.1. Afterwards, we dive into more details on response properties in Subsection 2.4.2.

2.4.1 Classification Methods for Responses

RFC 2828 [131] defines a *countermeasure* as „an action, device, procedure, or technique that reduces a threat, a vulnerability, or an attack by eliminating or preventing it, by minimizing the harm it can cause or by discovering and reporting it so that corrective action can be taken.“ In the following the term *response* will be used in the sense of a countermeasure in form of a concrete action triggered by an alert reported during the intrusion detection phase. As the number and variety of possible responses due to a security incident is huge, first some basic considerations on responses are done. This includes extracting possible criteria on how to structure responses from different taxonomies or ontologies found in literature.

According to [5] a response can either be *active* or *passive*. Active responses are used to minimize the attacker's impact on the system. In the best case the attack can be completely mitigated by using those responses. An active response may set the target system into an unstable state or do some other harm to the system. Therefore, selection of active responses has to be done carefully. Passive responses only notify other components without taking further actions. For example, sending an alert is a typical passive response.

In [5] active responses are further divided into *proactive* and *reactive* responses. Proactive responses are executed before a security incident occurs, while reactive responses are triggered after the detection of a security incident. Proactive responses are used for IPSes, while reactive responses are common in Intrusion Response Systems (IRS). Proactive responses can also be considered as *delayed* responses [128]. A proactive response can be executed during a security incident or after a security incident (recovery) [21]. In [103] passive responses are referred to as trace-back methods while active responses are referred to as incident handling strategies.

In [80] the idea of so called *one-shot* and *sustainable* responses is proposed. A one-shot response is only effective as it is executed, that means its lifetime is limited to a single attack. After launching the response it will be automatically deactivated after a certain time. In contrast a sustainable response will stay effective even after the occurrence of an attack. Sustainable responses can be divided into *defeasible* and *indefeasible* responses. A defeasible response can be deactivated, while an indefeasible response can not be deactivated.

In [128] the use of the location where a response is executed is proposed as an additional feature to categorize responses. The authors name the *attacker machine* or the *intruder machine* as possible location to execute a response. Additionally, they take the *attack path* into account. The authors define that the attack path consists of the starting point (attacker machine), the firewall point (firewall and routers), the midpoint (intermediary machines) and the endpoint (target machine). A response can be executed on all points of this attack path.

In [177] responses are classified as passive, neutral or positive. It is claimed that every response has to fall into one of these categories, but no further description is given.

In [103] responses can be executed in one of the following phases: *containment*, *eradication* or *recovery*. During the containment phase first aid measures are taken to stop the intrusion from spreading over the whole system. Those actions can be taken while an attack is ongoing. After containing the attack, the system has to be cleaned up within the eradication phase to eliminate the attackers access. This may includes changing passwords or re-build or re-install the compromised systems. The recovery phase deals with potential data loss occurred during the security incident. Possible activities within this phase include the restoration of user data from trusted backups or reviewing system configurations and protection mechanisms. Within this thesis the main focus lays on containment activities to stop the intrusion.

In [181] response and recovery action are classified as:

- *Rollback* actions try to bring the components back to a save and secure state.
- *Roll-forward* actions try to find a new state from which the component can be operated securely.
- *Isolation* actions try to perform physical or logical exclusion of the faulty components.
- *Reconfiguration* actions try to switch to spare component or reassign tasks to other available components.
- *Reinitialization* actions try to check or record new configuration or update components or configurations.

Those basic considerations on responses are used for classification and structuring of available responses in Subsection 5.1.3.

2.4.2 Response Properties and Characteristics

Despite the possibility to split up responses into different categories, every response has dedicated properties and characteristics that influence the response model. Some common properties of responses found in literature are listed in Table 2.1.

The *Time To Restore* (TTR) describes the time between the detection of a security incident and bringing the system into service again, measurable in mean (TTR_M) and

Tab. 2.1: Overview of Metrics to Evaluate Responses

Property	Name	Description	Related Work
P1	TTR	Time To Restore	[23, 101, 140]
P2	PRT	Policy Reaction Time	[140]
P3	IRT	Incident Response Time	[140]
P4	DT_C	Downtime of critical components	[23, 67, 140]
P5	DT_T	Downtime during critical time-slots	[140]
P6	E	Effectiveness	[23, 67]
P7	$comp$	Complexity or severity	[101]
P8	S	Error-proneness or likelihood of success	[20, 67, 101]

maximum (TTR_{Ma}). It is made up of the *Policy Reaction Time* (PRT) and the *Incident Response Time* (IRT) by the following definition

$$TTR = PRT + IRT = PRT + IRecT + IResT \tag{2.1}$$

The *Policy Reaction Time* (PRT) describes, hereby, the time the policy engine needs to decide what to do. The *Incident Response Time* (IRT) describes, hereby, the time between the start of recovery steps ($IRecT$) and bringing the target system into service again by appropriate restoration actions ($IResT$).

The *downtime of critical components* (DT_C) or *during critical time-slots* (DT_T) describes that important services are not available during the execution of a response. The *effectiveness* (E) of a response describes the benefit for the affected entity the response is executed on. This may be for example the degree of increasing availability gain. The *complexity or severity* ($comp$) of a response describes how difficult it is to execute this response, e.g. due to additional configuration issues or system complexity. The *error-proneness or likelihood of success* (S) describes the likelihood that after execution of the response the security incident is solved for the affected entity respectively how likely an error occurs during the execution of the response.

2.5 Intrusion Response Systems

Within this section, the goal is to gain a closer look into IRSes. In Subsection 2.5.1, we first examine an IRS taxonomy that was extended by different authors in multiple steps. Afterwards, the single steps of intrusion response are examined in more detail in Subsection 2.5.2.

2.5.1 Overview and Taxonomies

In [142] a taxonomy for *IRSes* is presented that was later on extended and visualized [128, 130].

According to [142] IRSes can be split up with respect to the *activity of responses* they trigger into *active* and *passive* responses. Passive responses notify about the security incident and provide information about the attack and do not attempt minimizing already caused or further damage. Traditionally, IDSes provide this passive response capability by raising an alert in case of an attack. Active systems are more invasive to the target system and try to minimize the damage or prevent attack steps.

Another distinction criterion regarding to [142] is the *degree of automation* that differentiates between simple *notification* systems, *manual response* systems and *automated intrusion response* systems (IRS). A notification system only notifies about the security incident going on, while a manual response system highlights options that can be done in order to cope with the ongoing security incident. An automated IRS do not need any feedback from outside and triggers a response completely automated.

Automated IRSes can be divided further with respect to their *ability to adjust* into *static* or *adaptive* systems. Static means that the response selection mechanism can either be static during an ongoing attack or adaptive to changing environmental conditions [142].

The *time instance* of an automated IRS can either be *proactive* also called *preemptive* or *delayed*. In case of proactive responses the system tries to forecast a security incident before happening and to predict how to react. A delayed response is going to be triggered after an attack is confirmed [142].

Another distinction criterion according to [142] is the *cooperation capability* of an automated IRS. An IRS can be a stand-alone system that acts *autonomous* or it can be a distributed system that supports *cooperation* over multiple instances.

A central functionality of an IRS is the *response selection* mechanism. In [142] response selection is divided into *static mapping*, *dynamic mapping* and *cost sensitive* mechanisms. Static mapping maps a predefined response directly to an alert. With respect to system complexity and the huge amount of threat scenarios this approach is claimed to be infeasible [151]. The enhancement of static mapping is to use dynamic mapping that includes attack metrics into the response selection. This can be expressed with Event Condition Action (ECA) rules. This approach is more flexible and the response selection is done just-in-time when an alert is raised. Cost-sensitive mappings try to balance the damage a security incident causes and the costs of a response. Hereby, the optimal response can be determined if the response as well as the damage can be assessed.

In [130] the *costs of a response* can be modeled with a *static*, *static evaluated* or *dynamic evaluated* cost model. Within a static response cost model static costs are assigned to each response. Within static evaluated response cost models the costs of a response are evaluated beforehand by an evaluation mechanisms for each response. In contrast dynamic evaluated response cost model take the network situation into account and determines the response costs online.

With respect to *response execution*, the authors of [130] distinguish between *burst* and *retroactive* strategies. In burst mode all responses are executed together, while in retroactive mode response execution is more fine-grained.

In [128] two additional dimensions for classifying IRSes are examined, namely *applying location* of responses and the *response lifetime*. The location, a response can be applied on, are all parts of the *attack path*, namely the attacker machine, firewalls and routers, intermediate machines and the attacked machine. A response can either be *sustainable* within the system or has to be *deactivated* after a certain time.

In [128] risk assessment is examined as a key capability of IRSes that influences the cost-sensitive mapping within the response selection. *Assessment strategies* can be based on *service dependency graphs*, *attack graphs* or can be *non-graph-based*.

In [66] three classes of automated IRSes are identified. *Adaptive-based* is derived from [142] and describes systems using a feedback loop evaluation previous responses. *Expert-based* systems use metrics to determine a response and *association-based* systems are using decision tables.

According to [142] an ideal IRS is **automated**, **proactive**, **adaptable** and **cost-sensitive**. [130] suggests additionally, that an IRS should be **retroactive** and use a **online**

response cost evaluation model. The authors in [128] extend this recommendation by the **elastic** property to support response activation and deactivation.

2.5.2 Steps for Intrusion Response

In the previous Subsection 2.5.1 it became apparent that intrusion response has to support different tasks. This subsection focuses on examining those different steps in more detail and chronological order.

After a security incident is detected and additional information is gathered during alert processing, intrusion response has to be *triggered*. Here, different approaches can be used, for example, the confidence level of a detected security incident is high enough [141]. Another condition for triggering may be the priority of a security incident exceeding a certain threshold.

Next, the candidate responses have to be *identified*. Candidate responses are all responses that could be potentially used for the given security incident and target system. This means candidate responses have to be:

- Effective and helpful against a certain security incident and
- Deployed such that the affected entity is covered by the response.

After identifying candidate responses those candidate responses have to be *assessed* with respect to certain metrics [128, 130]. This is necessary because from the candidate responses a certain set has to be *selected* [142]. This selected set has to be optimal with respect to the assessed metrics.

After selecting the optimal set of responses a *response plan* has to be *prepared*. This is required in case selected responses have relations like pre- or postconditions or a dedicated execution sequence needs to be enforced.

Finally, the response plan has to be *deployed* on the target system [128, 130]. Here, the execution has to be monitored such that all responses are executed according the response plan and no errors occur.

During the execution of responses the responses have to be *evaluated* in terms of different metrics. Those metrics are a required input for response assessment [128, 130].

3. ANALYSIS AND SYSTEM DESIGN

In this chapter we present the analysis as well as the system design for our proposed Incident Handling System (IHS). First, we introduce functional and non-functional requirement an IHS has to fulfill in Section 3.1. In Section 3.2 we introduce the incident handling process, the underlying pattern, the Blackboard Pattern, used for our system design and the mapping of this pattern to the incident handling domain. Afterwards, we examine the underlying information model of our system, including requirements for such a model and related work in Section 3.3. In Section 3.4 we examine the functional segmentation of the incident handling process and describe our modules for all steps within the incident handling process in more detail. In Section 3.5 we present related work on IHSes, provide an overview on existing execution models and frameworks and compare identified related work against our identified requirements. As the proposed system is already published in [62], in the last section a publication reference determines the own contribution to this system (see Section 3.6).

3.1 Requirements

The requirements for an IHS arise from functional and non-functional needs an IHS has to cope with. Those requirements were identified due to the literature search of available Intrusion Response Systems (IRS). Some requirements are directly stated in literature. In this case the reference is given in the description of the requirement. Other requirements arise from features or shortcomings of available IRSes. First, the functional requirements are examined in Subsection 3.1.1. In the second subsection, Subsection 3.1.2, the non-functional requirements are explained.

3.1.1 Functional Requirements

A functional requirement specifies the system's range of functions and the system's behavior. The functional requirements arise from different areas and are specified in more detail in the following.

Requirement RF1 – Multiple Intrusion Detection Systems (IDS): The system has to be able to cope with multiple input sources (e.g. IDSes) generating alerts (cf. [43]). This is needed as different IDSes can be deployed within a network, e.g. Host-based Intrusion Detection Systems (HIDS) or Network-based Intrusion Detection Systems (NIDS). The need to use multiple IDSes arise from the fact that different locations an IDS can be placed, leads to different insights into the target system. In order to provide a holistic security incident report and a knowledgeable response to the security incident, all generated alerts have to be included into the incident handling process.

Requirement RF2 – Support Alert Processing: The system has to support alert processing capabilities (cf. [43]). As the number of alerts from different IDSes is enormous and may differ in terms of their semantical meaning, automated intrusion response cannot be meaningful without combining alerts together. Otherwise, the automated intrusion

response will overload the target system with unnecessary responses may not having the expected success.

Derived from Subsection 2.5.2, where a description of the needed steps of intrusion response is given, an IHS has to support the following functionalities.

Requirement RF3 – Flexible Triggers for Automated Intrusion Response: The system has to be able to specify different conditions, intrusion response has to be triggered on (cf. [141]). This is needed to support different use cases that need a flexible strategy to describe policies when to use an automated intrusion response. Consequently, not only the ability to trigger an automated intrusion response is essential, but also the adjustment ability of the trigger mechanism.

Requirement RF4 – Response Identification: The system has to support multiple responses that can be mapped to a security incident (cf. [141]). This ensures that different respond strategies can be implemented, deployed and used simultaneously. This is needed because the optimal response to use depends on the system state of the target system. Therefore, a static mapping between attack and response is not sufficient enough.

Requirement RF5 – Cost-Sensitive Response Assessment: The response costs as well as the attack costs has to be determined by the system (cf. [142]). This is done using a cost-sensitive approach as described in Subsection 2.5.1. This is needed to properly identify a suitable response in case more response options are available (cf. RF3). Additionally, the system has to be able to determine whether or not a response is executed.

Requirement RF6 – Optimal Response Selection: The system has to find an optimal solution which responses to use from a given set of candidate responses suitable for a security incident (cf. [141]). This includes not only to find the most efficient responses with respect to certain metrics, but interdependencies between responses have to be considered. For example, responses can conflict with each other, or need pre- or postconditions that have to considered when trying to find the optimal set of responses.

Requirement RF7 – Response Plan Preparation: The system has to be able to deploy and coordinate a complex combination of multiple responses. This includes allowing distributed responses as well as parallel or sequential execution of responses. The system has to, therefore, consider dependencies between responses, e.g. pre- and post conditions or conflicting responses. According to this interdependencies between responses a holistic response plan, combining all responses into a structured control flow of the execution, needs to be generated.

Requirement RF8 – Response Deployment and Execution: The system has to be able to retroactively execute and deploy selected responses following the prepared response plan (cf. RF7). That means the system has to provide mechanisms to communicate with actors in the response plan and trigger the responses appropriately. Requirement RF7 and RF8 are necessary to achieve a retroactive system as recommended in [130].

3.1.2 Non-Functional Requirements

In contrast to functional requirements non-functional requirements describe criteria to estimate the system's operation rather than specifying concrete functionality. Derived from Subsection 2.5.1 an IHS has to fulfill the following non-functional requirements:

Requirement RNF1 – Automated: The IHS has to be automated such that no manual intervention is needed and notification capabilities are extended to response capabilities (cf. [142]). This encompasses all functionality the IHS provides. This is needed as automated incident handling benefits in terms of speed of reaction that leads to less damage as a security incident may be stopped.

Requirement RNF2 – Adaptable: The IHS has to be adaptable such that during intrusion response the response plan can be adapted to changing environmental conditions (cf. [142]). This includes for example to include additional targets into the response plan or re-select responses as they appear inefficient during execution. This is important as during an ongoing attack the decision basis may change.

Requirement RNF3 – Comprehensive View: The system has to be able to cope with multiple targeted hosts or subsystems and consider the security incident as a whole. Additionally, this includes helping as much as possible compromised network elements (cf. [165]). Instead of local decisions that may conflict, a global view of the situation is needed to ensure the effectiveness and efficiency of all selected responses. A global view can combine all available information about the target system, used policies and the security incident itself.

Requirement RNF4 – Generality: The system must not be restricted to a single application domain, but has to be general enough to be used in a flexible manner and multiple domains (cf. [59]). This does not mean that expert knowledge from specific domains is not necessary, but the integration of additional fields of application should be as easy as possible. Therefore, the system design itself must not include any domain specifics.

Requirement RNF5 – Continuity: The system has to provide a continuous processing of all required steps of incident handling in order to cope with a continuous on-going attack. This includes intrusion detection, alert processing as well as intrusion response. Sequential processes, i.e. the sequence of detection, processing and resolving is fixed, are not suited. As every step is done one by one and the results of the previous step are the input of a subsequent step, a bunch of problems occur. First, information get lost from one step to another and subsequent modules cannot access them anymore. Additionally, the order of single steps is not clear, as for example an incoming alert can be prioritized but a correlated alert as well. Another grave issue of such a sequential process is, how to map an infinite stream of alerts to such systems. This infinite stream arise the need of continuously processing information that sequential processing cannot afford.

Requirement RNF6 – Active Responses: The system has to disable detected security incidents directly even during the attack (cf. [165]). This implies that the responses that are executed are active and not passive. Furthermore, the supported responses has to exceed recovery actions reconstructing a safe state after a security incident. The used responses have to be applied during an attack is ongoing.

Requirement RNF7 – Collaborative: Incident handling covers multiple functional aspects, namely intrusion detection, alert processing and intrusion response. Within this processes many tasks have to be executed. All those single tasks rely – at least partially – on the same information or on intermediate results another task produces. Therefore, the system has to support collaborative strategies that allow an interaction between all modules fulfilling such a task. Additionally, a collaboration support allows to run the single modules across the network to distribute the incident handling process.

Requirement RNF8 – Modularity: All tasks of the single steps of the incident handling process have to be exchangeable. That implies that interfaces and behavior of all tasks are well-defined and free of interference as well as conflicts. Moreover, the single modules are not hard-wired with each other or components of the IHS on the one hand. On the other hand the system design does not rely on a certain implementation of a module.

3.2 System Design Overview

The main challenges of the system design for our IHS are to

- *Separate and split up the functionality* of the incident handling process into separate modules that are interference free and
- Provide an information sharing component based on an *information model* to support collaboration between those modules.

Therefore, we first give a short wrap up of the incident handling process in Subsection 3.2.1. Afterwards, we examine a software pattern that suits our need, called *Blackboard Pattern*, in Subsection 3.2.2. This pattern is than mapped to our application domain in Subsection 3.2.3.

3.2.1 Incident Handling

The proposed IHS supports all needed steps of incident handling to cover the whole life cycle of a security incident, beginning from the detection of a security incident, covering the analysis, and finally identify, select, and execute appropriate responses.

Intrusion Detection The infrastructure to be secured (target system) is monitored by different IDSes. This may include HIDSes as well as NIDSes as both systems can deliver different insights into the target system's state. Every IDS will generate alerts that are delivered to the proposed IHS in case an intrusion is detected. The proposed IHS provides multiple interfaces for those alerts.

Alert Processing The alert processing is first responsible for normalizing the incoming alerts. This is needed as multiple systems may have an inconsistent naming or description of the alerts. After normalization the alert is stored in the proposed IHS and is accessible to other components of alert processing. Steps, like aggregation, correlation or prioritization, are than done based on the inserted alerts in parallel. Alert processing is responsible to extract more meaning from single alerts and to identify the underlying security incident of incoming alerts.

Intrusion Response As soon as the intrusion response process is triggered responses capable of solving the security incident are identified and bundled. From this set, the response selection can determine which responses to choose in order to optimize the resulting costs to cope with the security incident. After determining this optimal set those responses can be applied to the target system that is secured by the proposed IHS. During and after execution the selected responses are evaluated. Those results are stored in the proposed IHS to be accessible for new decisions in terms of response assessment.

Holistic Process In the following we refer to the holistic process as *incident handling*. Intrusion detection, alert processing and intrusion response are referred to as *steps*. Those single steps can be further divided into single tasks (see Sections 2.3 and 2.5). A component executing such a task is referred to as *module* or *knowledge source* using the *Blackboard Pattern* phrase.

3.2.2 Blackboard Pattern

We identified the *Blackboard Pattern* as a suitable design pattern for our IHS as it provides an information sharing component and autonomous modules collaboratively working on shared information. Furthermore, this pattern was used for IDSes in earlier work (cf. [32, 33, 102]), but they did not cover alert processing and intrusion response. As this pattern lays the foundation of our system design, we give a short introduction in the following.

The *Blackboard Pattern* [17, 91], further examined in [75], is based on the idea to introduce a shared data structure acting as a global memory. This data structure is called *blackboard*. Modules, called *domain knowledge sources* with different application and domain-specific knowledge work on the blackboard independently. The goal is to divide a problem into subproblems, that can be solved autonomous by those modules. Modules store intermediate and partial results on the blackboard, or combine and delete information from it. The final solution is produced by the collaboration of those modules. Communication between modules is exclusively done on the blackboard. The main components of the Blackboard Pattern are examined in the following and shown in Figure 3.1 in UML-Notation taken from [91].

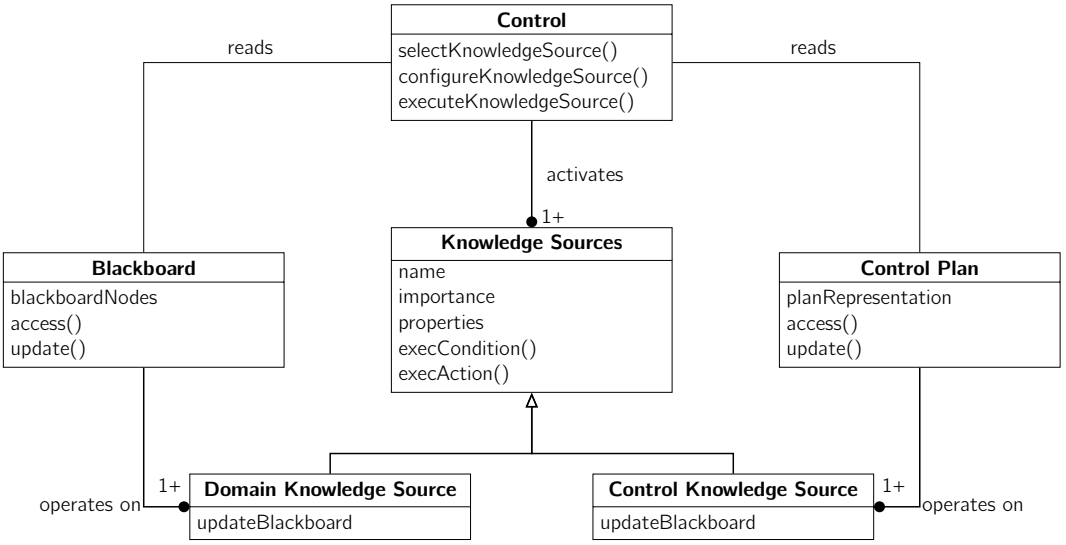


Fig. 3.1: Blackboard Pattern in UML-Notation [91]

The processing of information is data-driven. That means the control flow depends on the blackboard's current state which is monitored by a *controller*. When a change on the blackboard occurs, the controller can activate additional modules for further execution. Task decomposition and dependency analysis are used to improve the controller [170].

The controller follows a *control plan* to determine modules. The execution plan can be provided manually or automatically by so-called *control knowledge sources* [91]. Control knowledge sources can adapt the control plan during execution. Using the Reflection Pattern, the separation between the controller including its control structures and the modules including their data can be improved [133].

To ensure that only authorized modules access the blackboard, basic security mechanisms are introduced. The verification of modules is done by the controller [109].

3.2.3 Blackboard Pattern for IHSes

In this section we give an overview of the design of our IHS that uses a blackboard to share and exchange information between modules. Our system is depicted in Figure 3.2. Our system is divided into four main functional parts:

1. The blackboard as information sharing component, and modules to execute the tasks of the incident handling steps:
2. Monitoring and intrusion detection (dotted boxes),
3. Alert processing (solid boxes), and
4. Intrusion response (dashed boxes).

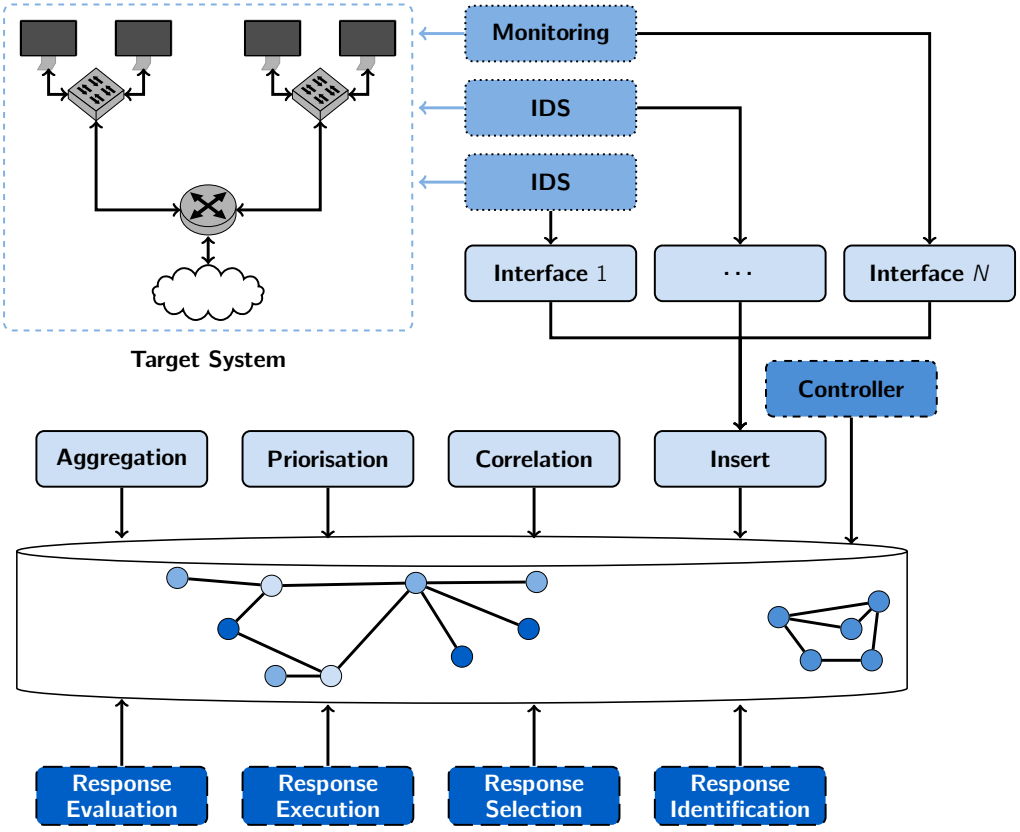


Fig. 3.2: System Overview of the Proposed IHS

The modules for the tasks of all steps of incident handling are depicted in Figure 3.2. The modules can be distributed across the network as long as they can connect to the blackboard and the controller can reach the modules. Each module can be available in different variants, e.g. multiple correlation modules can be used, as long as they follow the information model.

Blackboard The *blackboard* of the proposed IHS is the central information sharing component storing all required information. All modules can access the blackboard by reading information from or writing information on the blackboard. The blackboard can be realized as distributed database system accessible from different points in the network.

The blackboard has to provide an information model to enable information sharing [91]. We propose a graph-based information model where all information elements are represented as nodes. Connections between different elements are modeled as relations. Both, nodes as well as relations, may have attributes further describing the entity, see Subsection 3.3. Providing a graph-based information model follows the recommendation given in [91] where the blackboard is described as a structured global memory containing objects, so-called blackboard nodes, linked to each other.

A further important component of a blackboard is the *controller*. Regarding its control plan, the controller can intervene the control flow. Modules can be activated or deactivated to handle changes in the system state, see Subsection 3.4.6.

Monitoring and Intrusion Detection The target system is continuously monitored by host and network monitoring systems like Zabbix[1] or Nagios[2]. Those systems collect so called *infrastructure information* describing the current state of the target system.

Additionally, IDSes like OSSEC[3] or Snort[4] monitor the target system. In case a security incident is detected, they produce an alert message. Their capabilities and detection methods vary heavily and provide different insights into the target system. As it is common to use multiple IDSes in parallel [32, 99, 102, 178] the different views of the IDSes are consolidated by alert processing in the subsequent steps.

The infrastructure and alert information is inserted on the blackboard using different interfaces to support multiple systems. Within this thesis we provide an IDS that can deliver alerts to our proposed IHS, see Chapter 4 for details.

Alert Processing Collected alerts are handled by alert processing. First, alerts are associated to infrastructure information on the blackboard to locate and identify the security incident in the target system. An important goal of alert processing is to reduce the amount of alerts. This can be achieved by using multiple aggregation methods [34]. Alert correlation allows identifying relationships between alerts like attack paths or root causes. Prioritizing alerts enables the system to identify urgent security incidents to be handled immediately.

For the single tasks in the alert processing steps different tools and methods exist that can be applied independently and simultaneously, see Subsection 3.4.2 for further details.

Intrusion Response Intrusion response utilizes the infrastructure information collected by monitoring systems and the alerts collected by IDSes. First, suitable response available on the target system are determined as candidate responses. From those candidate responses an optimal set is selected to counteract the security incident. The selected responses are executed on the target system and evaluated by that time. The response evaluation is needed as response assessment to determine the optimal set of responses, see Section 3.4.3.

As the main focus of this work is automated intrusion response concrete modules covering those aspects are presented in Chapters 5 to 7.

[1] http://www.zabbix.com/
[2] https://www.nagios.org/
[3] http://ossec.github.io
[4] https://www.snort.org

3.3 Information Model

Within this section, we first examine the requirements that our information model has to fulfill in Subsection 3.3.1. Afterwards, we present our information model in detail by providing the description of the information elements as well as relations in between in Subsection 3.3.2. This includes references to the requirements stated in Subsection 3.3.1 to show that the proposed information model is aligned to the requirements. In Subsection 3.3.3 we present related work on information models related to the security domain and compare them against the requirements stated in Subsection 3.3.1.

3.3.1 Requirements

In the following, we subsume the major requirements that lead to the design of our information model for the blackboard. They arise from the need to combine the steps of incident handling, namely intrusion detection, alert processing, and intrusion response.

Requirement R1 - Information Separation: All modules have to update or add information independently without creating write conflicts resulting in inconsistent information. Hence, the information stored on the blackboard needs to be segmented in disjoint parts. The needed information has to be split up in single information elements that can be processed independently. Consequently, relations between the single information elements need to be modeled. Those relations may store additional information and have to be processable independently as well.

Requirement R2 - Completeness: The information model has to provide the required infrastructure information for all steps of incident handling. This includes all information required for intrusion detection, alert processing and intrusion response. Additionally, the information model has to provide all information elements that are needed to model those steps and the according tasks within the step. The single information elements for the single steps of incident handling need to be connected to the infrastructure information available in the information model.

Requirement R3 - Compatibility: Our information model has to be compliant to standards related to incident handling, e.g. the alert message format Intrusion Detection Message Exchange Format (IDMEF) [35]. This is needed as existing components build upon those standards and should be integrable into our proposed IHS. Additionally, relying on existing standards ensures easier further development as a stable basis is provided.

Requirement R4 - Traceability: The information model has to provide the ability to trace all tasks of incident handling. Hence, alerts and intermediate results like selected, and executed responses have to be stored on the blackboard. Tracing the whole process of incident handling allows the investigation of the process itself. It allows adaptations of single steps and ensures auditability and reproducibility of all ongoing steps. Furthermore, additional recovery actions can be executed and tracked easier.

3.3.2 Information Model Description

The information model we provide is graph-based (cf. Requirement R1). This allows a fine-grained structure of information elements and relations between single information elements. Our information model is depicted in Figure 3.3. A node, representing an information element is represented as box. Dotted edges between nodes, express an *isA* relation. Dashed edges between nodes, express a *has* relation. Solid edges between nodes, express an *association* relation.

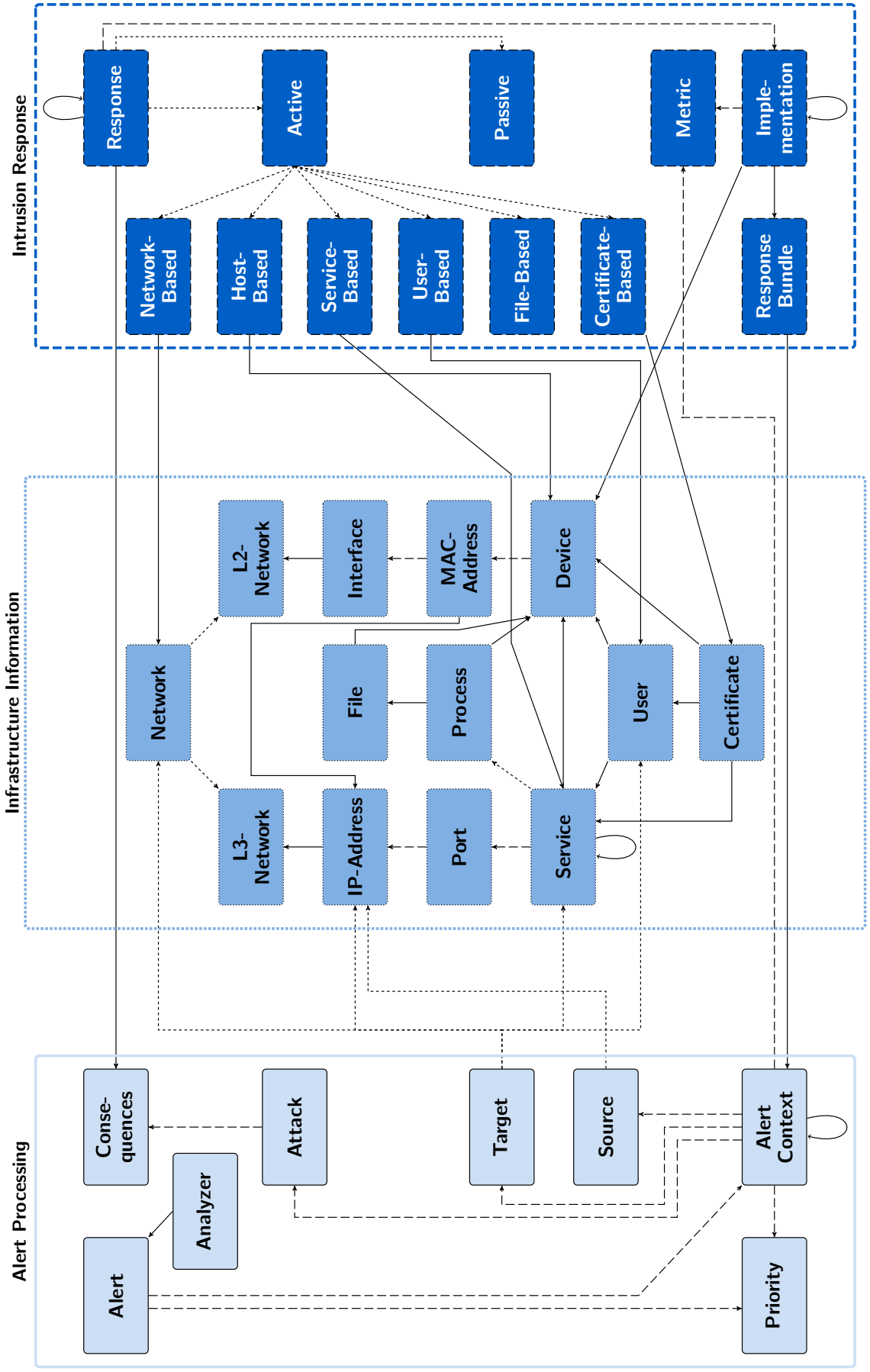


Fig. 3.3: Information Model of the Blackboard to Enable Information Sharing among Modules

To reflect the three types of modules, namely intrusion detection, alert processing, and intrusion response, our information model is split into three groups of nodes (cf. Requirement R2):

1. Infrastructure nodes (dotted boxes in Fig. 3.3) to link intrusion detection, alert processing, and intrusion response,

2. Alert processing related nodes (solid boxes in Fig. 3.3) to cover intrusion detection and alert processing , and

3. Response related nodes (dashed boxes in Fig. 3.3) for intrusion response.

To avoid write conflicts (cf. Requirement R1), modules only write nodes related to their task. Read access is allowed to all nodes to enable collaboration across tasks (cf. Requirement R2). Updating information element is rare and restricted to single modules. The main operation done by modules is adding new information elements including their relations to other information elements.

Infrastructure nodes (dotted box in Fig. 3.3) reflect infrastructure information collected by monitoring tools. A typical network consists of devices (`Device` node), e.g. hosts or routers, and subnetworks (`Network` node). Networks can be switched (`Layer2 Network` node) or routed (`Layer3 Network` node). Devices can be split further, using the `isA` relation, into `Router` nodes, `Host` nodes, `Virtual Machine (VM)` nodes, or `Server` nodes. Typically, a VM is running on a server, such that those nodes are related to each other. For reasons of clarity and comprehensibility we do not depict this kind of inheritance in Figure 3.3.

A device can be connected to several layer2 networks via multiple interfaces (`Interface` node) each having a MAC address (`MAC` node). An IP address (`IP` node) is assigned to a MAC address providing access to a layer3 network. Services (`Service` nodes) are running on a device and are associated with a port (`Port` node) of a specific IP. Services may depend on another service, indicated by the self-reference. A service is a special process (`Process` node) that is running on a device. A process can be based upon multiple files (`File` node) that are executed or used and stored on a device. Users (`User` node) are logged on devices and use services. A user, service and device can be equipped with a certificate (`Certificate` node).

Alert processing related information is marked as solid box in Fig. 3.3. An alert (`Alert` node) is detected by an analyzer (`Analyzer` node), e.g. an IDS. Each analyzer has a certain severity, the security incident is detected with, if an alert is triggered. Therefore, each analyzer provides a `Severity` attribute. Each alert is associated with an alert context (`Alert Context` node) that combines semantically related alerts. Each `Alert Context` node provides a `Solved` attribute to indicate whether the security incident was handled successfully (`TRUE`) or not (`FALSE`). Additionally, an alert context can be equipped with a severity that is calculated from the severity of the analyzer providing the alerts combined within the alert context. Therefore, an `Alert Context` node provides a `Severity` attribute. An alert or alert context can be prioritized using a `Priority` node. Prioritization can alternatively be realized as an attribute to improve performance by reducing the depth of the graph. The relation between an `Alert Context` node and the `Metric` node represents the damage a certain security incident causes. This value is assessed to estimate the harm of a security incident and to find appropriate responses.

An alert context is associated with a `Source`, `Target`, and `Attack` node (cf. Requirement R3). The source describes the supposed attacker, normally determined by its IP.

The target describes the victim in the target system and can be further described by creating references to infrastructure nodes such as IP, User, Device, Service, or Network nodes. Each infrastructure node can be assessed regarding its importance for the target system. Therefore, infrastructure nodes provide a Value attribute. An Attack node classifies the type of a security incident. Attacks are associated to consequences (Consequence node) happening in case of success. Attack and Consequence nodes describe which security incidents our system can handle and are provided by experts.

The last category is response-related information marked as dashed box in Fig. 3.3. A response (Response) can mitigate certain consequences resulting from an attack. Responses can either be Active or Passive. Passive responses (sending alerts or notifications) do not directly mitigate or decrease the damage to the target system, while active responses (blocking traffic, shutting down hosts) do. As we focus on active responses, we divided them with respect to the target they influence during execution. We identified amongst others User-Based, Network-Based, Service-Based, File-Based, Certificate-Based, and Host-Based responses. This list is not complete but exceeds the scope of this thesis to identify all possible fine-grained targets. Information about available responses has to be provided to our system by an expert, knowing the capabilities of the target system.

Responses may be related to each other in two different relations. Responses can either be conflicting (responseconflictswithresponse relation) or depend on each other (responseispreconditionofresponse relation). The first relation excludes a combined execution of two responses. The second relation will force that either both responses are executed or none of them.

For each response different implementations (Implementation node) can be deployed, meaning available, on a device. Additionally, an Implementation node is related to a device indicating that a device is executing an implementation. As responses, Implementation nodes can be related with each other described through conflicts or preconditions. Those implementations have different metrics (Metric node) describing the implementation. The metrics are used for response selection and assess an Implementation node with respect to the costs of the response implementation. A more detailed description on responses and their relations to each other is given in Subsection 5.1.2. A Bundle node is associated with an alert context and indicates response implementations associated with the security incident. This association means that implementations within this bundle can be potentially applied to the security incident.

A bundle has multiple attributes used for communication between different modules in the intrusion response process:

- The Executing attribute indicates if the execution of this bundle is ongoing.
- The Active attribute indicates if this bundle is still in use.
- The Prepared attribute indicates whether or not the execution is ready to start. This attribute is needed to prepare the evaluation of responses during execution.
- The Ready attribute indicates whether or not the execution of a bundle can start.

Additionally, the relation between an Implementation node and a Bundle node carries supplemental information:

- A Selected attribute indicates whether or not a response implementation was selected to be executed.

- A `Round` attribute indicates when the response implementation was selected.
- A `Executed` attribute indicates whether or not a response is yet executed.

This design fulfills Requirement R4 as all subtasks of intrusion response are traceable during the process. Different rounds of execution can be distinguished and the intrusion response process is transparent.

3.3.3 Related Work

As basis of our information model, the needed concepts can be described as ontologies. Therefore, available ontologies are investigated in the following to identify the relevant concepts they provide. First, we examine ontologies build for general purpose within the security domain in Subsubsection 3.3.3.1. Afterwards, we take a closer look on ontologies used in security applications in Subsubsection 3.3.3.2. In Subsubsection 3.3.3.3 we subsume ontologies from other related domains. The last subsubsection, Subsubsection 3.3.3.4, compares the related work against the requirements stated in Subsection 3.3.1.

3.3.3.1 Security-Related Ontologies

In [157, 158] an ontology for attacks and intrusion detection is given. The concepts include amongst others *Actor*, *Attack*, *Goal* and *Effects*. This work does not provide possibilities to include infrastructure information describing the target system. Moreover, response capabilities are not sufficiently represented.

In [108] the first steps for an ontology for the cyber security domain is presented. The work is not finished, yet, but lacks in inclusion of infrastructure information as well as the representation of response capabilities.

In [153, 154] an ontology for IDSes is presented. This ontology models aspects like *Hosts*, *Attack*, *Consequences* and *System Components*. Some infrastructure information is included, and the concept of *Attacks* and *Consequences* is introduces. This work lacks of representing response capabilities and alerts.

In [9] an ontology to represent the current security state of a network is presented. They represent the network with different components such as *Hardware*, *Services* or *Users*. They model *Vulnerabilities* in detail and represent *Attacks*, as well as *Effects*.

In [111] an ontology for network security attacks is presented. They mainly focus on attacks itself, and include concepts like *Security Properties*, different *Attacks* including *Attack Steps* and *Vulnerabilities*. They provides *Attack Profiles* consisting of *Assets*, *Access*, *Actor*, *Motive* and *Outcome* and link them to security properties they attack.

3.3.3.2 Ontologies within Applications

In [29] an ontology for alerts describing the IDMEF-Format is given. This ontology is used in the RED (Reaction after Detection) project. The goal is to instantiate security policies used in case of a security incident by mapping IDMEF alerts into Organization Based Access Control (OrBAC) contexts. Therefore, the policy is expressed in form of an ontology. Additionally, they provide the concept of a *Threat Context* that is mapped to the alert classification and a *Reaction Focus* that is mapped to the source and target of an alert.

In [120] an alert correlation, called *ONTIDS* framework based on an ontology is presented. They model the underlying infrastructure (target system), and *Attacks*, as well as *Attack Contexts*. They include concepts like *Vulnerability*, *IDS* and *Attack Objectives*.

In [94] an ontology for an alert correlation framework is presented. This ontology is capable of reflecting the alert format IDMEF by providing appropriate information elements. Additionally some basic infrastructure information can be reflected. The concept of an *Attack* and *Security State* is defined. The *Attacker* as well as the *System State* is coupled with the *Security State*. Nevertheless, this ontology does not provide information element to reflect intrusion response capabilities. Consequently, chosen responses can not be tracked for auditing.

In [56] an ontology to model Security Information and Event Management (SIEM) information and operations is presented. A so-called information class provides concepts for infrastructure information, like *User*, *Network*, or *System*, and for security related terms, like *Alert*, *Vulnerability*, or *Policy*. A so-called operation class provides concepts for *Detection*, *Decision*, and *Reaction*. The intrusion response capabilities of the proposed ontology are limited such that no tracing of executed responses is possible.

3.3.3.3 Ontologies from Different Domains

In [11] an ontology to represent network infrastructures is presented. This ontology is used in the proposed *IO Tool* to scan network infrastructures. They focus on the representation of the underlying network infrastructure and do not consider security incident or intrusion response capabilities. Nevertheless, they provide a rich model to present infrastructure information that can be included in the information model provided in this thesis.

Some related work is focusing on special use cases, domains or attacks. In [160] an ontology for attacks especially for web services is presented. They focus on a concrete domain and provide a deeper insight in possible attacks on web services. In [180] an ontology focusing on attacks in wireless sensor networks is presented. They map attack, intention, movement, target and result to possible attack scenarios in wireless sensor networks. In [159] an ontology to classify network Denial of Service (DOS) attacks is presented. They provide an overview on protocols vulnerable to DOS attacks and link appropriate attacks to those protocols.

3.3.3.4 Summary, Comparison and Conclusion

A comparison of the generic approaches of the related work is given in Table 3.1. Approaches focusing on special domains or attacks are not included, as the needed information model has to be generic for all scenarios and attacks. Specialized work can be used to extend our information model with respect to a more detailed view of certain attacks.

This comparison clearly shows that none of the existing information models is a suitable foundation for our IHS. Non of the presented approaches covers the requirement of traceability (cf. Requirement R4). Additionally, the presented approaches do not completely cover all required information elemente for all steps of the incident handling process and most often lack within intrusion response capabilities. Some concepts of those information models are used and influence the proposed information model but can not be used directly without adaptations.

Tab. 3.1: Comparison of Related Work Based on the Requirements Stated in Subsection 3.3.1

Approach	R1 – Separation	R2 – Completeness	R3 – Compatibility	R4 – Traceability
Attack Ontology [157, 158]	✗	✗	✗	✗
Cyber Security Ontology [108]	✗	✓	✗	✗
IDS Ontology [153, 154]	✗	✗	✓	✗
Security State Ontology [9]	✗	✗	✓	✓
Network Security Ontology [111]	✗	✗	✗	✗
RED Ontology [29]	✗	✗	✓	✗
ONTIDS [120]	✓	✓	✗	✗
IO Tool[11]	✓	✗	✗	✗
Alert Correlation Ontology [94]	✓	✗	✓	✗
SIEM Ontology [56]	✓	✓	✓	✗

Some relevant concepts of the proposed ontologies are utilized as basis for the own information model. Therefore, parts of the concepts of the presented ontologies are used, and combined such that they can be used together. Missing concepts were added to be able to fulfill all needed requirements. Newly added concepts mainly cover infrastructure information and concepts for intrusion response. The proposed information model fulfills those requirements, as explained in the description of our information model in Subsection 3.3.2.

3.4 Functional Separation

In this section we examine the tasks of incident handling in detail and present the modules implementing the tasks. The modules of our system reflect the knowledge sources of the Blackboard Pattern. The modules are independent from each other and provide functional disjoint tasks of incident handling. This segmentation has several advantages:

- Modules can be distributed across the network as no direct communication between modules is required.
- Modules can be exchanged easily as they implement self-contained tasks.
- Collaboration between modules can be managed easily using a central information sharing component.
- Additional management of modules can be done easily using a controller, e.g. authentication or module selection.

In the following, we describe the modules in detail and therefore, follow the structure given from the single steps of the incident handling process. First, monitoring and IDS modules are covered in Subsection 3.4.1. Alert processing modules are covered in Subsection 3.4.2 followed by modules of intrusion response in Subsection 3.4.3. In Subsection 3.4.4, we

give a short example to illustrate the functional interaction of the modules covering the incident handling process. In Subsection 3.4.5 a cleanup module – the garbage collector – is introduced. As access to the blackboard is managed by the controller, this component including the control plan are examined in Subsection 3.4.2.

3.4.1 Monitoring and Intrusion Detection

These modules collect infrastructure information about the target system (e.g., IP or MAC addresses of devices) and store the current state of the target system on the blackboard. If new information is found, a new instance of the infrastructure node is added on the blackboard. Edges between the new node and existing nodes are added if needed. If a change is detected, the corresponding existing node or edge is updated. A continuous monitoring of the target system ensures that infrastructure information is up-to-date.

Multiple monitoring tools can be used in parallel distributed across the target system. For example, dedicated tools processing log files, monitoring packet flows or scanning subnetwork can be deployed. So far, we incorporated a limited set of infrastructure information. However, the capabilities of the IHS can be extended and adapted easily to other needs by adding additional interfaces to the blackboard of the IHS.

In Chapter 4 we propose an IDS that can be used as module within the proposed IHS.

3.4.2 Alert Processing

Alert processing is well covered in this domain and multiple different approaches are available [19, 26, 28, 40, 43, 44, 94, 106, 156, 169, 176]. The steps of alert processing, we present in the following, are adapted from the given related work.

Initial Insert Once an IDS raises an alert, the alert is inserted on the blackboard. The insertion module provides one or more *interfaces* for available IDSes to receive the alerts. The insertion proceeds as follows. First, a new `Alert` node is inserted. The source, target, and attack classification are extracted from the alert. A suitable `Alert Context` node is searched. This means source, target and attack classification match. If existent, a new edge between alert context and alert is inserted. Otherwise, a new `Alert Context` node is created and linked to corresponding `Target`, `Source` and `Attack` nodes.

Within this step an implicit normalization is done. The attack classification itself is not stored in the `Alert Context` node but is linked to available `Attack` nodes. Additionally, an implicit first aggregation is done. Alerts with the same source, target and attack classification are combined into an alert context [34]. This aggregation is loss-free in terms of information lost as alerts remain on the blackboard, but additional information, the `Alert Context` node representing the aggregation, is added.

Aggregation Typical systems aggregate alerts in a lossy process. The raw alert information, that is the input for aggregation, is discarded, but aggregates are passed. In our IHS, aggregation is creating edges between existing `Alert Context` and `Alert` nodes with similar features. That may be the same source, target, attack classification, or a combination of those features. Aggregation with the same target and attack classification are especially helpful for distributed attacks like distributed DOS attacks. As distributed attacks have the same target and attack classification, many alerts are combined into a single `Alert Context` node. Because of the loss-free aggregation, additional aggrega-

tion features are possible, e.g., alerts from the same IDS. Alert aggregation can also be handled using clustering techniques [72, 73, 74].

The aggregation module is triggered after a new `Alert Context` node is inserted. The module checks, if the new node can be aggregated with existing `Alert Context` nodes. If so, the new node either belongs to an existing aggregate or a new `Alert Context` node as aggregate is needed. An edge between the newly inserted `Alert Context` node and the `Alert Context` node representing the aggregate is inserted.

Correlation This module combines multiple different `Alert Context` nodes. In contrast to aggregation, correlation does not focus on feature similarity of alerts or alert contexts, but tries to find more high-level relations. A typical example is an attack path: a host is first the target of an attack, then becomes a source of a subsequent attack. For alert correlation a bunch of possibilities [30, 92, 99, 112, 121, 155, 171, 179] exists, that can be combined within the proposed IHS.

This module is triggered after a new `Alert` or `Alert Context` node is inserted. The processing procedure is analogous to aggregation, but using correlation rules instead.

Prioritization This module calculates or updates the priority of an `Alert` or `Alert Context` node. If a new priority is added, this new priority value has to be propagated through connected `Alert Context` nodes.

Prioritization can be separated into two modules. One module calculates the priority value of a single alert, for example, considering available infrastructure information like the importance of affected hosts. This module is triggered after a new `Alert` node is inserted. Approaches to calculate priorities for alerts can be found for example in [53, 105].

Another module propagates newly inserted or updated priorities through the `Alert Context` node hierarchy, meaning through connected `Alert Context` nodes. For propagation, different calculations methods can be applied, e.g., using the weighted average of all incoming priority values. The priority propagation module is triggered after updating or adding priority information to an `Alert` or `Alert Context` node.

3.4.3 Intrusion Response

To implement automated intrusion response, specialized functionality from a completely different domain than alert processing is required. However, intrusion response needs the same infrastructure information as required by alert processing. Additionally, intermediate solutions from alert processing are helpful for intrusion response.

Response Identification This module examines a set of possible responses mitigating a specific and open (meaning unsolved) alert context. This is done by combining the available information as follows. The module follows the edge from the `Alert Context` node to the `Target` node and further to a concrete network entity, e.g., a `User`, `Network`, `Service`, or `IP` node. From this entity, edges to `Response` nodes exists. As the responses are split up into the targets they influence, only applicable responses are reached, e.g., a `User` node is only related to `User-Based` response nodes. In case a flexible deployment mechanism is available, those responses can be used, even they are not deployed on the target, as they can be deployed and executed in a single step.

Next, this set of candidate responses is further limited to useful responses. The module follows the edge from the `Alert Context` to the `Attack` node and further to the `Consequence` nodes. Finally, we determine responses that mitigate a certain

consequence by following all edges between the identified Consequence and Response nodes. This results in a second set of applicable Response nodes.

Calculating the intersection between both calculated sets of responses lead to a set of candidate responses helpful against certain consequences of an attack and effect the target of the security incident. Afterwards, this component follows the edges between all Response and Implementation nodes that are available on the device associated with the target. For example, a user is logged in on a certain device means that only response implementations available on this host remain in the candidate set. After determining applicable Response nodes a new Bundle node is added on the blackboard. The Bundle node is connected to the corresponding Alert Context node and the previously identified Implementation nodes.

To trigger the response identification module, multiple options are available. Different infrastructure information can be utilized as trigger, e.g., a highly valuable element in the target system is affected by an attack. Additionally, alert processing information can be utilized, e.g., an alert context exceeds a certain priority or number of connected alerts.

In Chapter 5 we describe the response identification module in more detail and provide possible responses that can be used within the IHS. Additionally, possible triggers for automated intrusion response are examined.

Response Selection This module selects the most suitable subset of the candidate responses. It is triggered after insertion or updates of a Bundle node where the Active attribute is set to TRUE. This module can be implemented using optimization techniques for optimal solutions or heuristics in case of less relevant threats. For the selection process, metrics associated with the implementations of the candidate responses are utilized. This module updates the edges between the Bundle and the Implementation node by setting the Selected attribute to TRUE. The Active attribute of the Bundle node is set to FALSE.

In case the selected responses were not efficient against a certain attack, the existing bundle can be re-used as candidate responses are already identified. The response selection module only takes responses into account that have not been tried out yet (the Selected attribute is FALSE). Our approach has the advantage that all intermediate steps are completely documented on the blackboard, which allows accountability of the IHS. In case no targets are connected to the Alert Context node, the response selection module will set the Solved attribute to TRUE. This happens, if an aggregated alert context is in process, whereas all connected Alert Context nodes have already been solved.

In Chapter 6 we provide a closer look into the response selection module and propose an approach to find the optimal set of responses to be used for a security incident.

Response Execution This module examines suitable actors able to execute the responses, generates a response plan and starts coordinated response execution. It is triggered by updates on the Selected property of the edge between an Implementation node of a response and the corresponding Bundle node. All selected, not yet executed, responses and the correct sequence of execution are determined. The resulting response plan is than executed on the target system. Questions like scheduling and coordination of responses are done by this module. Additionally, this module is responsible for deploying needed responses on the device that are executing responses in case of a security incident.

After preparing the response plan for a Bundle node the Ready attribute of the Bundle node is set to TRUE. As the execution starts, the Executing attribute in the

Bundle node is set to TRUE and is triggered by a Bundle node with the Prepared attribute set to TRUE. After the execution, the Executing attribute is set to FALSE.

In Chapter 7 we provide a framework to execute multiple responses including a language to specify the response plan for coordinated and structured execution of responses.

Response Evaluation This module measures response implementations during their execution and updates related Metric nodes, e.g. effectiveness or duration. Those metrics are stored on the blackboard in form of additional nodes related to Implementation nodes. Those metrics are used for response selection and response assessment. Those metrics have to be updated after the implementation was used within the response execution module. Additionally, the Executed attribute of the edges between Bundle and Implementation node are set to TRUE after their execution.

Additionally, the response evaluation module examines if a response bundle was successful to mitigate a security incident. If so, the Alert Context node is closed by setting the Solved attribute to TRUE. Otherwise, the Active attribute in the Bundle node is set to TRUE to trigger a new response selection. A response bundle is interpreted as successful, if no update or insert on the Alert Context node is done during a defined time span. This time span is the duration a response needs to be effective within the target system. The response evaluation is started after the Executing attribute in the Bundle node is set to TRUE and stopped after this attribute is set to FALSE.

In order to be able to prepare the evaluation of response implementations, communication between the response execution module and the response evaluation module is needed. As the response execution module has generated the response plan, the Ready attribute of the Bundle node is set to TRUE. The response evaluation module will prepare everything to be able to measure the implementations during execution. Afterwards, the Prepared attribute is set to TRUE, to indicate that response execution can be started. The response execution module will wait until the Prepared attribute is set to TRUE and than continue with its execution as described above.

3.4.4 Example for the Interaction of Modules

In Figure 3.4 an excerpt of the instance of the information model is given to illustrate the interaction of modules described in Subsection 3.4.1 to 3.4.3.

The infrastructure information is limited to three different IP-Addresses (IP1, IP2, and IP3). We assume that two alerts are triggered (AL1 and AL2). Those alerts contain the same attack type (classification) and target, but are originated from different sources.

The initial insert module will insert both Alert nodes and additionally two Alert Context nodes (AC1 and AC2) including the relation between the Alert and Alert Context nodes. The aggregation module identifies AC1 and AC2 to have the same target of the attack and insert the Alert Context node AC3 including the relations between AC1 and AC3 and between AC2 and AC3.

The response identification module identifies all suitable candidate responses applicable to the attack and deployed on the devices connected to the IPs. The response identification module does not care about relations between Response or Implementation nodes, like the Conflict relation between R2 and R3 or I2 and I3. All implementations connected to these responses are linked to the Bundle node B indicating that B solves AC3. The inserted Bundle node has initially the following attributes set: Active to TRUE, Executing to FALSE, Prepared to FALSE, and Ready to FALSE. All relations

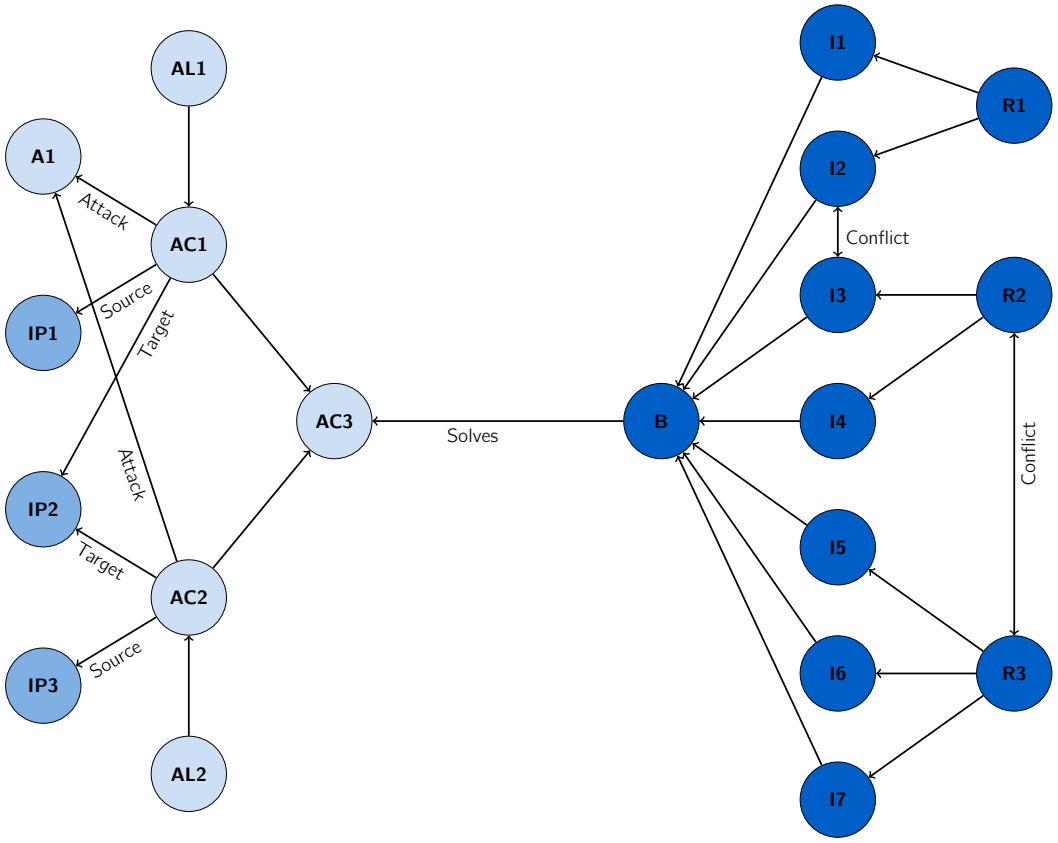


Fig. 3.4: Interaction of Modules from all Steps of the Incident Handling Process – Example

between the `Bundle` node and an `Implementation` node initially have the following attributes set: `Selected` to `FALSE`, `Executed` to `FALSE`, and `Round` to 0.

In Figure 3.4 an excerpt of the blackboards state after the explained steps is shown. Relations between responses and implementations with the infrastructure information are left out. During further execution the response identification module may append additional `Implementation` nodes to the bundle and adds relations between `Alert Context` nodes the bundle solves.

The response selection module is triggered after the `Active` attribute of an `Alert Context` node is set to `TRUE`. The response selection module will select the optimal subset of implementations to solve the security incident and reset the `Active` attribute to `FALSE`. For example, `I2` and `I5` are chosen. The edges between `I2` and `B` and between `I5` and `B` are updated as follows: `Selected` to `TRUE` and `Round` to 1.

An edge between a `Bundle` and an `Implementation node` with a `Selected` attribute set to `TRUE` and an `Executed` attribute set to `FALSE` triggers the response execution module. The latest (with the maximum `Round` attribute) selected and not yet executed implementations are collected and a response plan is generated. After generating the response plan, the `Bundle` is updated as follows: the `Ready` attribute is set to `TRUE`. The response plan is scheduled and waits for its execution.

The response evaluation module is triggered by the `Bundle` node with the `Ready` attribute set to `TRUE`. This module will set this attribute back to `FALSE` and arranges the measurement of the implementations to be executed. When everything is ready, the `Prepared` attribute will be set to `TRUE`.

The `Prepared` attribute set to `TRUE` in the `Bundle` node triggers the response

execution again. The previously scheduled response plan will now be executed and the `Prepared` attribute is set back to `FALSE`. Before execution the `Executing` attribute is set to `TRUE`. This triggers response evaluation starting the evaluation of the implementations. After the execution of the response plan is finished, the `Executing` is set back to `FALSE` that is stopping the response evaluation module.

The response evaluation module decides whether or not the executed responses were helpful or not. If they mitigated the security incident the `Solved` attribute of `AC3` is set to `TRUE`. Otherwise, the `Active` attribute is set to `TRUE` triggering response selection again. Additionally, response evaluation will set the `Executed` attribute on the edges between the executed implementations to `TRUE` and renews the values for the metrics of the implementations.

During the execution of the system the response identification, the response execution, and the response evaluation module have to hold additional state. In case of the response identification module, already seen, but unsolved `Alert Context` nodes are stored, to reduce querying the blackboard actively. The response execution module has to schedule its response plan but needs to wait for the response evaluation module. On the other side, the response evaluation module prepares for measurements and waits for the response execution module for the final execution. As both modules have to work non blocking, they have to store the needed information in the meantime they wait for the other module.

3.4.5 Garbage Collector

All modules described so far, only insert or update nodes. Hence, the underlying blackboard will be soiled by outdated or no longer required information over time. Therefore, a single dedicated module for garbage collection is needed. The *garbage collector* uses a fine-grained rule set to determine which information is still required on the blackboard and for how long. The rule set depends on the use case as it determines how long a view into the past is possible. As the garbage collector is the only module allowed to delete information, no conflicts with other modules occur.

For example, `Alert Context` nodes with the `Solved` attribute set to `TRUE` can be deleted from the blackboard using the garbage collector. This includes `Alert Context` nodes and `Alert` nodes connected to the solved alert context being a subset of this alert context. Additionally, connected `Bundle` nodes as well as all relations between the bundle and the alert context with each other and other nodes can be deleted as no module no longer requires this information. As this information is may required for forensics or accountability reasons this information can be copied to and stored on an additional longterm storage.

3.4.6 Controller and Control Plan

The controller in the proposed design notifies appropriate modules regarding its control plan in case information on the blackboard is added or updated by other modules. A simplistic controller activates all modules and passes detected changes to all modules no matter what task they implement.

A more sophisticated controller can use *publish-subscribe* mechanisms to reduce the amount notifications. A change on the blackboard is noticed by the controller that publishes the change only to subscribed modules. As the controller needs to select the subscribers, *symmetric publish/subscribe* can be used. Thus, the controller can specify which modules get a notification of the blackboard's change. This avoids deactivation of

modules that must not get informed. Hence, no inconsistencies on the blackboard occur as aborted modules wrote preliminary results on the blackboard.

Putting the decision logic of responsibilities for particular information into the modules instead of the controller has certain disadvantages. The decision logic has to be distributed over multiple modules complicating maintenance. However, a single, isolated module with a local, limited view, cannot decide about its own responsibilities. Furthermore, overlapping executions of modules within the same domain might result in inconsistencies.

One challenge using a central control mechanism is to select suitable modules. The control plan defines under which conditions a module has to be used. Therefore, the controller determines and interpret the situation first. Second, the controller maintains knowledge about the capabilities and properties of modules to select the correct ones.

For example, if only a few alerts arrive on the blackboard the work load will be low. The controller can select modules with extensive analyzing capabilities. If the alert rate increases, a faster decision is needed. In this example, the controller has to be able to determine the rate of incoming alerts and needs to assess the work load for the system. With respect to the assessed expected workload appropriate modules are selected.

3.5 Related Work

In this section related work from different research areas is examined. Existing solutions focusing on one or more single steps of incident handling are presented in Subsection 3.5.1. The main focus of this presentation is to examine the underlying processing models of the existing solutions. We show, why existing execution models are not appropriate for subsequent intrusion response. Afterwards, selected IRSes are introduced in Subsection 3.5.2. First, we introduce fundamental approaches for IRSes either being early work or comprehensive frameworks. Secondly, we examine other approaches providing partial solutions for intrusion response. In Subsection 3.5.3 we compare the fundamental approaches and frameworks against each other using the requirements identified in Section 3.1.

3.5.1 Execution Models for Incident Handling

Within the related work, we identified the following execution models, examined in more detail in the following:

- Sequential execution,
- Processing based on Complex Event Processing (CEP),
- Agent-based systems, and
- Systems using an information sharing component.

3.5.1.1 Sequential Execution

Most execution models are sequential [43, 79, 103, 110, 120, 132, 177, 181], i.e., the sequence of all tasks for intrusion detection, alert processing and intrusion response is fixed. Every task is done independently and the result of the previous task is input of a subsequent task. This is problematic as information is discarded between tasks. Hence, subsequent tasks have less information available. Additionally, the order of tasks is not deterministic, as alerts or aggregated alerts can be prioritized. Another severe issue is

processing a virtually endless stream of alerts in a sequential process expecting a finite input.

Examples: The *CIDS* framework [43] and other work proposed in [90, 156], describe tasks of the alert processing step. Tasks are executed in sequence to minimize the amount of alerts raised by IDSes. The *ONTIDS* framework [120] proposes an alert correlation framework combining different information sources. *OutMet* [132] combines alert correlation and prioritization and represents correlated alerts as directed acyclic graphs. In [176] alerts gained from agents are correlated into a so-called attack scene. For alert processing they use normalization, aggregation, verification and correlation. In [28] the CRIM Architecture is presented. The focus lies on alert processing and implements alert clustering, alert merging, alert correlation and intention recognition. The RED Framework [79] is examined in Subsubsection 3.5.2.4. ADEPTS [51, 168] is examined in Subsubsection 3.5.2.5.

3.5.1.2 Techniques based on CEP

To cope with endless streams of alerts, CEP can be used. Systems based on CEP are capable of processing streams of alerts, and offer a continuous way of alert processing. The drawback of CEP is the need of pre-defined alert windows defining how long alerts last in the system. As it is a priori unknown how long alerts are relevant, determining the window size is challenging.

Examples: In [49], a generic *Intrusion Detection and Diagnosis System (ID^{2}S)* is proposed. The system supports alert correlation for detecting and analyzing intrusions in large-scale critical infrastructures.

3.5.1.3 Agent-Based Systems

More collaborative approaches use agent-based systems. The single steps are done by autonomous agents but a central master component controlling, managing, and providing information to the agents is required.

Examples: Emerald [113] is further examined in Subsubsection 3.5.2.2. CSM [166] is further examined in Subsubsection 3.5.2.1. AAIRS [115] is further examined in Subsubsection 3.5.2.3. CITRA [124] and IDIP [123] are further examined in Subsubsection 3.5.2.6

3.5.1.4 Information Sharing Component

Some approaches utilize a dedicated component for information sharing to overcome the central controller with manifold management capabilities. All modules in the systems work independently without being managed by a central master.

Examples: In [32, 33] an IDS based on learning strategies is introduced. This work focuses on the pure detection process and collaboration between different detection techniques. SPIDeR [102], an IDS, is based on multiple agents using self-organizing maps, a neural network technique used in machine learning. The agents perform tasks independently and store the results on an information sharing component. In [163] a system to detect and asses misconfigurations is proposed.

3.5.1.5 Summary

Sequential processes and CEP-based systems suffer from limited possibilities of information sharing between the single steps of incident handling. Agent-based systems heavily rely on

a master component controlling the agents and distribute information. Hereby, most of the application logic is present within the central master component that needs to perform important tasks of the IHS. Existing solutions providing a component for information sharing are immature and do not cover the whole incident handling process.

The presented systems propose partial solutions to incident handling resulting in different issues. The systems cannot be distributed easily across the network as tasks are not decoupled. The functionality and communication between tasks is tightly coupled to the proposed system. Hence, single tasks cannot be used as standalone components. Information sharing and collaboration of those systems are limited and no possibilities of integrating missing steps of incident handling are proposed.

3.5.2 Selected IRSes

Within this subsection fundamental approaches for IRSes are examined in Subsubsection 3.5.2.1 and 3.5.2.2 Comprehensive frameworks are presented in Subsubsections 3.5.2.3 to 3.5.2.6. In Subsubsection 3.5.2.7 we introduce other approaches providing a partial solution to intrusion response or a high-level approach.

3.5.2.1 Cooperative Security Managers (CSM)

In [166] an early Intrusion (Detection and) Response System (I(D)RS) called *Cooperative Security Managers (CSM)* is introduced. The authors propose to deploy a CSM on every machine in the network working cooperative and autonomously. A CSM consists of an *intrusion detection component (IDS)*, an *Intruder Handling (IH) component*, that determines actions to execute, and a *Security Manager (SECMGR)* that tracking users.

Security Managers can communicate with each other in order to appropriatly track users and examine the current target system state. Within this work, some exemplary responses are listed and weighted with respect to their impact on the user. They propose to use the severity as measure to decide which response to use. In case the suspicion level of a certain action exceeds a certain threshold, a response is executed by the IH module.

As this is an early work within this field it lacks in certain details as no explicit response selection is done, the cooperation in terms of executing different actions in a cooperative manner is left out and no response assessment is provided.

3.5.2.2 Event monitoring Enabling Responses to Anomalous Live Disturbance (EMERALD)

In [113] *EMERALD (Event monitoring Enabling Responses to Anomalous Live Disturbance)*, one of the first IRSes, is described. Within this early work the focus is mainly on distributing monitoring and intrusion detection capabilities. They provide a hierarchical layered approach based on *service analysis*, *domain-wide analysis* and *enterprise-wide analysis*. They propose so-called *Service Monitors* that can be distributed across the network and monitor different resources. Those resources are abstracted network entities, such that an individual adjustment to the monitoring target is possible. Each resource includes *Response Handlers* executable on the target.

Information, collected by the Service Monitors, can be shared amongst them and is propagated to the domain monitors and further to the enterprise-wide analysis. This layering allows to reduce information amongst the hierarchy. A so-called *Resolver* is the coordinator of all analysis reports as well as the implementor of all response policies. The Resolver of EMERALD is described as an expert system invoking responses provided by

the resources. Furthermore, they describe evaluation metrics a response has to provide in order to allow a selection. The Resolver uses the threshold and severity metric to formulate policies for response execution.

This early work lacks in much details, the metrics are not clearly explained. Furthermore, no concrete responses are given and the mapping between response and attack stays unclear. The focus is mainly on distributed intrusion detection and alert processing instead of intrusion response. Nevertheless, basic concepts of intrusion response are examined. This early work is further adapted to the domain of critical infrastructure to detect network traversal attacks [14].

3.5.2.3 Adaptive Agent-based Intrusion Response System (AAIRS)

In [71, 115] the *Adaptive Agent-based Intrusion Response System (AAIRS)* is proposed. They provide response adaption by weighting responses with respect to their success probability in the past. They support multiple IDSes interfaced by *Interface Agents* maintaining the false-positive rate of each individual IDS that builds the attack confidence metric and normalizes incoming alerts from different IDSes. Alerts and confidence metrics are handed over the *Master Analysis Agent* determining if a security incident is new or ongoing.

For each new security incident an *Analysis Agent* is created, for ongoing security incidents the new information is handed over to the responsible agent. The Analysis Agent decides what response to use by invoking the *Response Taxonomy Agent* and the *Policy Specification Agent*. Therefore, a response plan consisting of a response goal and multiple plan steps with associated tactics is created. To create this response plan the following inputs are used: response goal, confidence and success metric, incident report and history, available tactics and policies. To further limit responses, the Response Taxonomy Agent uses the time, implications and type of attack, attacker type and degree of suspicion. The selected response is handed over to the *Tactics Agent* consisting of multiple *Implementations* that translates a high-level response plan to executable actions and invokes the *Response Toolkit* for execution.

This work mostly provides a methodology for intrusion detection and intrusion response but lacks in much details. For example, the response selection is unclear, no concrete responses are given and no strategies for alert processing is explained. In [20] the AAIRS is expanded to be able to deal with uncertainty within the detection process. Therefore, the false-positive rates of IDSes are calculated more precisely.

3.5.2.4 REaction after Detection (RED)

In [29, 37, 38, 39] the *RED architecture (REaction after Detection)*, approach based on the OrBAC security model and on ontologies, is proposed. Within their architecture they provide the following elements. The *Alert Correlation Engine* receives alerts from the network and is capable of further diagnosis. The results are send to the *Policy Instantiation Point* that instantiates new security policies. Those security policies are sent to the *Policy Decision Point* to be deployed. The *Policy Enforcement Point* or *Reaction Enforcement Point* is responsible to enforce those deployed policies. Actions executed within this loop are called *high-level reactions*.

Two shortcuts are possible: one from the Alert Correlation Engine into a *Reaction Decision Point* into the Reaction Enforcement Point for *mid-level actions*. Additionally, the Reaction Enforcement Point can decide and execute low-level actions directly.

To determine which response to take they provide a so-called *threat context activation* that maps to a classification of an alert. To determine the *reaction focus* meaning where

to execute, they use the source and target definition of an alert. The Policy Decision Point takes both as input to determine a suitable response.

In [77] the work done in [78, 79] is integrated in the OrBAC concept to include risk assessment and cost-based response selection. Within this work the benefit of different reaction levels is unclear. Moreover, no details for the single components are provided. Neither suitable actions are proposed nor a concrete strategy for execution is given.

3.5.2.5 ADEPTS

In [51, 168] an automated adaptive intrusion containment and disruption tolerant system, called *ADEPTS*, is introduced. They model the spreading of a failure through the system using an acyclic graph, called *I-Graph (Intrusion Graph)*. Nodes of the graph represent sub-goals, while edges represent pre- and post conditions of the attack (AND, OR and Quorum). From the I-graph an attack subgraph is created for every attack instance.

They compute how likely an intrusion goal within the I-Graph can be reached using the *Compromised Confidence Index (CCI)*. Therefore, they utilize the alert confidence level and the edges of the I-Graph.

They used a statically determined set of responses from which an appropriate one is selected. They use responses targeting nodes where the current attack is happening and most likely will spread to. The *Response Control Center* storing available responses within an *Response Repository* chooses responses applicable to the attack and computes the *Response Index (RI)*. The RI is calculated by subtracting the *Disruptiveness Index (DI)* from the *Effectiveness Index (EI)* of the response. The response with the highest RI is chosen for each node.

Within their work they do not consider interdependencies between responses nor provide a structured response plan or response execution process. They do not consider alert processing and split up the information they use into the I-Graph and the Response Repository. Interconnection and interchangeability of separate modules is not possible.

3.5.2.6 Cooperative Intrusion Traceback and Response Architecture (CITRA) and Intruder Detection and Isolation Protocol (IDIP)

The *Cooperative Intrusion Traceback and Response Architecture (CITRA)* [124] is based on the application layer IRS, named *Intruder Detection and Isolation Protocol (IDIP)* [123]. They structure their system into administrative domains, called *IDIP Communities*, capable of detecting and responding to security incidents. Directly connected IDIP systems are referred to as *Neighborhoods*. Connections between neighborhoods are referred to as *Boundary Control Devices*.

If an intrusion is detected, an IDIP system responds locally. The collected information and the response decision is spread to the neighborhood. A central entity called *Discovery Coordinator* correlates all available information and provides the big picture of the intrusion. Additionally, this component is capable of canceling or adding responses. Within their work they discuss secure communication and trust between the different IDIP Communities and IDIP nodes. An IDIP node can either be the Discovery Coordinator or an IDIP agent. An IDIP agent provides detection, audit or response functionality.

Within the CITRA framework a simple cost model is used to determine the five basic responses using thresholds. The systems allows to use those responses as immediate response. Additionally, administrators can interact with the system manually. They provide boundary controllers to block traffic of an attack as the only active response. The main focus is on the traceback of an attacker to the source of the security incident.

CITRA and IDIP provide limited response capabilities. Most responses are passive and only one is active. The assessment of responses and attacks is simplistic. The response selection mechanism is based on thresholds and can not be adapted.

3.5.2.7 Other Approaches towards IHSes

Sequential Approaches Most of the related work within the field of IHSes is based on sequential execution. Besides already presented frameworks, the following related work, additionally falls into this category with respect to the execution model.

In [173] a three-layer IRS based on *contextual fuzzy cognitive maps (FCM)* is presented. The first layer is used to map incoming alerts to four pre-defined classes of intrusions, namely *DOS*, *Probing*, *R2L* and *U2R* taken from [95]. This layer is called *Intrusion Recognition Layer* and is implemented using a conceptual graph ontology. This conceptual graph can represent any knowledge by defining concepts and relations, in this case relations between possible intrusions and the intrusion class. The second layer, called *Diagnosis Layer*, is responsible for providing information about the influence of intrusion impacts on the target system. They use a fuzzy cognitive map that defines concepts as nodes and relations between those concepts as weighted relations. This map reflects how certain attacks will degrade security goals, like data integrity and confidentiality. They map those consequences to resources within the system using dependency relations between resources. The last layer, called *Agent Response Layer* interfaces with the response capabilities of the system. They provide specialized agents responsible for one intrusion class. As the second layer determined the target the response has to be applied on, the agent in the third layer can execute the response. This system does not provide information on how to select a concrete response within the agents. No response nor intrusion assessment is given. Moreover, the proposed system lacks in details about the knowledge that has to be provided within the system. Most of this information is expert knowledge and has to be mapped directly into the system without an information model.

In [76] an IRS for *relational databases (RDB)* is proposed. The system consists of an anomaly-based IDS and an anomaly-based IRS. Anomalies are detected based on access profiles of different roles and users. In order to realize the IRS, they define an *Event Condition Action (ECA)* language, whereas the event describes the anomaly occurred and the action represents the response to be executed under certain conditions. They propose several responses especially for databases with different severities. They propose conservative actions having low severity, aggressive actions having high severity and fine-grained actions having medium severity. Additionally, they propose an administration model for response policies as well as possible attacks and protection strategies.

In [86] the *DSS framework* for collaborative response based on *Case-Based Reasoning (CBR)* is presented. The process consists of three steps: *Protection*, *Detection* and *Reaction*. In order to automate this process intelligent agents are deployed for intrusion detection. The basic idea is to manage case profiles describing who, how, where, and when a security incident occurs. The CBR-engine is responsible for managing those profiles and in case of new security incidents mapping them to the most similar case seen before and triggering the response accordingly. The main focus is on performing the analysis on how to generate the profile based on log analysis and how to calculate the similarity between the cases. A structured process to identify possible responses and to match responses and security incident is not presented, but relies on expert knowledge.

In [150] the *RAIM* framework, a SCADA security framework for critical infrastructures, is proposed. The abbreviation RAIM is compsed of the four key components: Real-time

monitoring, Anomaly detection, Impact analysis, and Mitigation strategies. Information is gathered during real-time monitoring. This collected data is used for anomaly detection. Hereby, different correlation techniques are used to detect the security incident, e.g. temporal correlation or spatial correlation. Afterwards, an impact analysis examines the effects to the security incident. Based on those effects, the mitigation strategy is examined for the most vulnerable part of the system. The main contribution of this paper is the identification of possible security enhancements based on a vulnerability assessment. The work lacks in terms of details regarding execution and selection of responses.

In [149] the *Risk Assessment for Intrusion Detection with Automated Response (RADAR) Framework* is presented. They provide an *IDS* component, a *Risk Assessment System (RAS)*, and an *IRS*. Their main focus is in integrating IDSes with poor detection capabilities, managing a high false positive rate and try to minimize the execution of unnecessary responses. Instead they try to execute only effective responses. Therefore, they provide a huge list of performance metrics used to estimate available IDSes. Within their RAS they provide mechanisms to evaluate the risk of an attack and the effectiveness of a response. They provide a list of responses they assume to have automated. Their selection strategy is restricted to decide which percentage of the intrusion is handled automated, the rest is handed over to the administrator. They do not provide a selection mechanism which response to use and they do not cover structured execution of responses on the target system.

In [110] a holistic system, called *REASSESS (Response Effectiveness Assessment)*, is proposed combining alert processing and response capabilities. Their system is based on several stages that are recommended by [23] and follows a sequential execution. Within their system they name, but not further describe, the following components: IDSes raising alerts in case of a detected intrusion, alert processing consisting of normalization, aggregation, and correlation, response selection and response execution. Besides their proposed system is sequential, they do not further describe and cover the single tasks of incident handling, but focus on the response selection process.

Agent-Based Approaches The number of agent-based approaches is limited. Those approaches realize the IHS as distributed system, where information is shared among the agents of the IHS. An issue of such an approach is the amount of information to be shared amongst the agents. Additionally, those agents need to implement the same functionality, otherwise the would need to know about what other agent needs which information. The implementation within the agents is again sequential.

In [148] *Kinesis* a security incident response and prevention system for wireless sensor networks is introduced. They utilize distributed components running an IDS and an additional Kinesis system distributed across the target system. Different information is shared among the Kinesis entities regarding the security state of the neighbors based on history data. The response selection is done based on a mixture of a policy-based selection and response assessment. the execution is done by distributed monitors with free capacities for execution. For high severity actions, first, a consensus of all Kinesis entities is needed.

Approaches Focusing on Specific Aspects The following approaches provide either only a high-level description of their IHS or focus on a concrete problem specification. Therefore, they will not be listed in the final comparison.

In [118, 119] the *Generic Authorization and Access Control API (GAA-API)* is introduced. Within this work, the authors provide a new policy description language called *Extended Access Control List (EACL)* that allows to define access policies before, during,

and after access to an object. The proposed framework is able to adapt policies during runtime. They propose how to in-cooperate with an IDS by identifying possible information that can be reported by an IDS. This reported information can be included within the policies specified in EACL. Possible responses within GAA-API are to allow or to deny access to an object. As this solution is policy-based, we do not include this solution into our comparison of related work. Additionally, the response capabilities are limited such that a comparison with more comprehensive solutions is not applicable.

In [88] the use of the *OODA loop* [13] consisting of the four stages *Observe*, *Orient*, *Decide* and *Act*, to model a holistic cyber defense process. The observation phase is needed to gather information about the underlying target system. This includes active and passive monitoring, as well as the use of IDSes to gather security relevant information like alerts. The orientation phase is needed to substantiate the security incident and identify responses. This includes reducing the amount of messages and information about the security incident to a manageable amount of events. The decision phase is used to select an identified response with respect to certain criteria. The action phase will eventually execute the selected response. Within their work they only map the concept of the OODA loop to IRSes and describe related work within this phase. They provide a process-based view on the intrusion detection and intrusion response process on a very high-level point of view. Their work lacks in concrete implementation strategies for the single processing steps. They neither provide an underlying information model, nor an interaction description of their processing steps. As this work is very high-level and lacks in details, we do not include it in the final comparison of related work.

In [126, 127] alert and response capabilities are modeled using the *Department of Defense Architecture Framework (DoDAF)*. This architecture framework provides different views for different stakeholders and provides models for developing and representing architecture descriptions. In this work, a very high-level description of services for managing alerts, responses and both in conjunction is given. The presented works lacks in much details about the concrete implementation, used mechanisms and strategies. Therefore, this work is not included within our comparison as this overview does not provide deeper insights but instead shows how to model known concepts within the DoDAF framework.

In [165] a tracing-based active IRS is proposed. They focuses on strategies to trace back the attacker in real time to apply better responses. The authors propose some high-level response strategies but do not propose a whole system to cope with the attack.

Manual Approaches or Recommendations Additionally to automated approaches, some related work provides partially manual solutions or simply gives recommendations how to handle incidents. this type of related work is covered in the following.

The NIST Computer Security Incident Handling Guide [23] proposes four stages as intrusion response life cycle: *Preparation*, *Detection and Analysis*, *Containment and Eradication and Recovery* as well as *Post-Incident Activities*. The preparation phase includes preventive tasks as well as the installation of appropriate monitoring and analysis tools. During detection and analysis a security incident or abnormal behavior is identified and classified. The security incident is documented and prioritized in this stage. As a last task a notification has to be send to the people in power. In the third stage a containment strategy is chosen and evidence is gathered. After identifying the attacker, the eradication and recovery tasks can be triggered. The post-incident activity is used to document the security incident and the incident handling process and evaluate the quality of the process. The whole process is given in form of a textual description and single activities are not structured. Especially, the third phase is not covered in detail. Moreover, this process is

not automated, but describes the business process that has to be done manually.

In [85] a framework for collaborative incident response and investigation, called *Palantir*, is introduced. This framework is not automated but provides a business process model. Responsibilities and roles are assigned to groups within the organization. The single steps to be executed during incident handling are covered in a generic and less detailed manner. The tool that is provided supports the administrators doing the intrusion response manually during documentation to ensure traceability of the incident and allow further learning steps after incident handling.

3.5.3 Summary, Comparison and Conclusion

The IRSes described in Subsection 3.5.2 are compared with respect to the requirements stated in Section 3.1.

Tab. 3.2: Comparison of Related Work Based on the Requirements Stated in Section 3.1

Approach	RF1 – Multiple IDS	RF2 – Alert Processing	RF3 – Trigger Response	RF4 – Identification	RF5 – Assessment	RF6 – Selection	RF7 – Preparation	RF8 – Execution	RNF1 – Automated	RNF2 – Adaptable	RNF3 – Comprehensive	RNF4 – Generality	RNF5 – Continuity	RNF6 – Active Responses	RNF7 – Collaborative	RNF8 – Modularity
CSM [166]	✓	✓	✗	✓	✗	✗	✗	✓	✓	✗	✗	✓	✗	✓	✓	✗
EMERALD [113]	✓	✓	✗	✓	✗	✗	✗	✓	✓	✗	✗	✓	✗	✓	✓	✗
AAIRS [20, 71, 115]	✓	✓	✗	✗	✗	✗	✓	✓	✓	✗	✗	✓	✓	✗	✗	✗
RED [29, 37, 38, 39]	✓	✓	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✗	✗	✗	✗
ADEPTS [51, 168]	✓	✗	✗	✓	✓	✓	✗	✓	✓	✗	✗	✓	✓	✓	✗	✗
CITRA [124], IDIP [123]	✓	✓	✗	✓	✗	✓	✓	✓	✓	✗	✓	✓	✗	✓	✓	✗
RDB [76]	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗	✗	✓	✓	✓	✗	✗
FCM [173]	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✗	✗	✗	✗
DSS [86]	✓	✓	✗	✗	✗	✓	✗	✗	✓	✗	✗	✓	✓	✓	✗	✓
RAIM [150]	✓	✓	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✗	✗	✗	✗
RADAR [149]	✓	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✓	✗	✓	✗	✗
REASSESS [110]	✓	✓	✗	✗	✓	✓	✓	✓	✓	✗	✓	✓	✗	✓	✗	✓
Kinesis [148]	✓	✗	✗	✓	✓	✓	✗	✓	✓	✗	✗	✗	✓	✓	✓	✓

First we would like to point out, that the presented IRSes do not provide a central component for information sharing. Mostly, they are sequential or agent-based using a master component with excessive functionality complicating collaboration between the modules and leading mostly to non-exchangeable modules. The shortcoming of such executions models are examined and discussed in Subsection 3.5.1. Approaches using an information sharing component do not provide a full IHS but are limited to intrusion detection and partially alert processing.

Moreover, none of the presented approaches is able to fulfill all functional and non-functional requirements. Most of the presented approaches support the use of multiple IDSes and are automated, but non of them has a flexible response trigger. They do not cover how to couple alert processing and subsequent intrusion response such that those processes are executed in a continuous manner to cope with the continuously incoming stream of security events occurring during an on-going security incident.

Even if it seams that most of the presented approaches are able to execute responses (Requirement RF8), most of them lack in details how they achieve response execution but only name the functionality. A structured execution using a response plan that is prepared before execution is not discussed in the presented related work. Preparation actions are limited to intervening responses or mapping selected responses to implementations.

3.6 Publication Reference

Parts of the content of this chapter are already published on WISCS (Workshop on Information Sharing and Collaborative Security) 2016 (cf. [62]). The own contribution of this paper was the idea of the execution and information model, the implementation, and the evaluation of the proposed system. The presented information model within this thesis is extended to better integrate the designed modules and includes additional information elements compared to the presented paper.

4. INTRUSION DETECTION

An *Intrusion Detection System (IDS)* is the second line of defense after intrusion prevention and hardening mechanisms failed. The quicker an intrusion can be detected, the more damage may be prevented and the sooner recovery actions can be triggered [143]. intrusion detection includes the identification of the source of an attack, the target of the attack as well as the classification or type of an attack. In case of a detected intrusion, an alert is generated. A standardized format for an alert description is given by the standardized *Intrusion Detection Message Exchange Format (IDMEF)* [35].

In this chapter an IDS detecting protocol violations based on Complex Event Processing (CEP) strategies is presented. As an example protocol, the Scalable service-Oriented MiddlewarE over IP (SOME/IP) protocol was chosen. A short description of this protocol, the underlying attacker model as well as a use case description is given in Section 4.1 including the requirements of the system. The design of the IDS is given in Section 4.2. The implementation details are described in Section 4.3. In Section 4.4 the evaluation of the proposed IDS is examined and the results are presented. As the proposed IDS is already published in [63], in the last section a publication reference is given to determine the own contributions to this system (see Section 4.6).

4.1 Analysis

In this section the SOME/IP protocol is introduced in Subsection 4.1.1. This section further investigates possibles attacks on this protocol in order to show how different violations against the protocol specification can be misused in Subsection 4.1.2. As this work was done in cooperation with Airbus Group Innovations (AGI) their specialized use case is summed up in Subsection 4.1.3. The requirements arising from the protocol and the use case are explained in Subsection 4.1.4.

4.1.1 SOME/IP Protocol Description

SOME/IP [7] specifies a middle-ware for Remote Procedure Call (RPC) on top of the TCP/IP protocol stack, and a serialization format for the messages between RPC clients and RPC servers. SOME/IP was standardized by the AUTOSAR project and is service-oriented [161]. It was designed for embedded devices in the automotive domain, but can also be adapted to the aerospace domain.

The underlying transport protocol can either be the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP). A TCP connection as well as a single UDP packet may contain more than one SOME/IP message. SOME/IP specifies 30491 as dynamic client port. The first server instance should use port 30501. In case more server instances are active, subsequent ports should be used.

The SOME/IP message format is shown in Figure 4.1. The 32 bit long message ID is used to distinguish between RPC calls a client can execute on the server instances.

This message ID is composed of

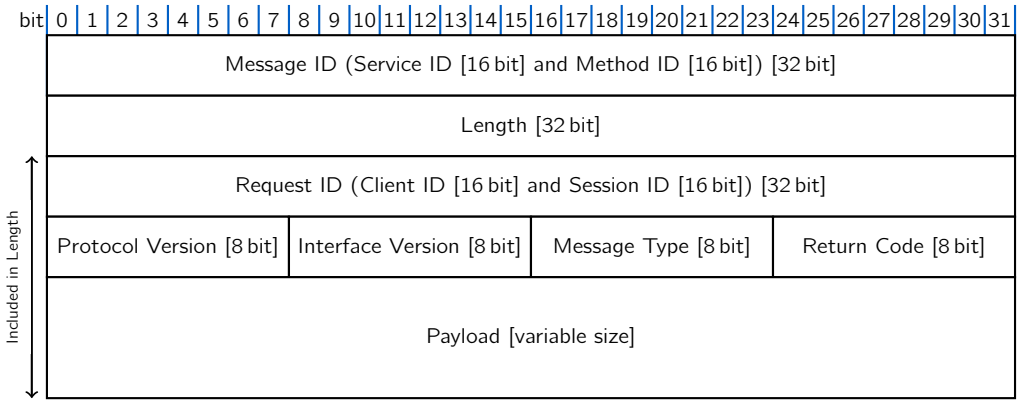


Fig. 4.1: SOME/IP Message Format as Specified in [7]

- A 16 bit service ID to address the service available on a server instance
- A single bit (1 bit) to distinguish between an event (Bit is set to 1) or a method call (Bit is set to 0)
- And 15 bit event or service ID to distinguish between single events occurred or the method that has to be called on the server side.

The 32 bit length field covers all subsequent header fields as well as the payload. The 32 bit request ID is used to distinguish between multiple open RPCs initiated from the client. Therefore, the client can call the same method multiple times without waiting for an appropriate response. The server on the other hand side can differentiate between incoming RPCs and map them to the right client.

The request ID is composed of

- A 16 bit client ID to identify a client uniquely over the system, further divided into a 8 bit prefix and a 8 bit client ID
- And a 16 bit session ID to be able to identify every open RPC

The 8 bit protocol version is currently set to 0*x*01. The 8 bit interface version describes the application specific major version of a service interface provided by the server.

SOME/IP supports the different message types that are listed and explained in Table 4.1. The 8 bit return codes of a `REQUEST`, `REQUEST_NO_RETURN`, and `NOTIFICATION` have to be set to 0*x*00 (`E_OK`) indicating no error occurred. In case of an error, different error codes can be applied. Those error codes cover for example unknown services or methods requested by clients, timeouts, wrong message settings e.g. wrong interface or protocol version, or message types as well as communication issues like the service is not ready to use or the system is currently not available.

The protocol standard does not specify any security measures, such as authentication or encryption. Therefore, the devices have no easy way of determining the authenticity and integrity of messages. This situation facilitates an attacker to compromise a valid device and execute attacks from inside the network, for example by plugging compromised hardware into available USB ports within the data cabin. Whether or not a communication partner is malicious, can not be decided by a device on its own. A central network monitor, on the other hand, can correlate packets from all devices. If specialized to the SOME/IP protocol, it may detect attacks and misconfigurations of services.

The SOME/IP standard proposes using security measures of underlying protocols but not keeping additional overhead in mind that is not feasible in an embedded domain.

Tab. 4.1: Supported Message Types of the SOME/IP Protocol

Number	Value	Description
0x00	REQUEST	A RESPONSE is expected.
0x01	REQUEST_NO_RETURN	No other message is expected.
0x02	NOTIFICATION	Requests a notification service and expects no message to be returned.
0x40	REQUEST_ACK	Can be send optionally as response to REQUEST.
0x41	REQUEST_NO_RETURN_ACK	Can be send as additional information as response to REQUEST_NO_RETURN.
0x42	NOTIFICATION_ACK	Can be send as additional information as response to NOTIFICATION.
0x80	RESPONSE	Is sent as return message for a REQUEST.
0x81	ERROR	Is sent as return message for a REQUEST in case an error occurred.
0xC0	RESPONSE_ACK	Can be send as an acknowledgment to a RESPONSE.
0xC1	ERROR_ACK	Can be send as an acknowledgment to an ERROR.

4.1.2 Possible Attacks on the SOME/IP Protocol and Attacker Model

We analyzed the protocol design described in Subsection 4.1.1 with respect to possible attacks and identified the following classes of misbehavior that can be a harm to the SOME/IP protocol:

Malformed Packets are packets whose structure does not comply with the standard. Such packets may be inserted intentionally by an attacker or by an improperly operating device. For example, the payload length may be shorter than indicated in the header, the version number may be different from 1, or an invalid combination of return code and request ID may be set. Malformed packets can be used to trigger bugs in the protocol implementation, but can also be used as a covert channel to exchange information undetected or for protocol fuzzing attacks.

Protocol Violations include all deviations from the specification in a conversation between two devices. For example, a regular request with message type 0x00 is expected to be answered with a single response with message type 0x80. Two responses to one request violate the protocol, and may indicate a misconfigured device or the presence of an attacker who tries to impersonate the server and injects additional packets.

System-Specific Violations are violations of additional, scenario specific constraints, which are not defined by the standard, but by the system under observation. For example, in our real-world use case, SOME/IP service discovery is not utilized or each device offers exactly one service.

Timing Issues are the subset of system specific violations which are concerned with deviations from the expected packet inter-arrival times. For example, keep-alive messages may be missing. A sudden increase in the number of messages may indicate a Denial of Service (DOS) attack.

In this work we focus on the SOME/IP protocol itself and do not include the service discovery capabilities SOME/IP SD that may introduce security risks of their own.

An IDS is placed in the target system such that it receives all traffic from all devices in the cabin. With the knowledge of the specification and a live view of the network traffic, we chose the following attacker as a possible opponent to the proposed system. The attacker is able to:

- Compromise a known device within the target system. Thus, the attacker has a valid MAC address, IP address, and service ID,
- Eavesdrop on all traffic within the target system, and
- Send packets to all clients and all servers, and thus impersonate other SOME/IP devices and services.

All non-attacker devices are assumed to behave according to the specification. If a device does not comply, it will simply be flagged as a possible attacker.

4.1.3 Use Case Description

This work was done in cooperation with the research department of Airbus Group Innovations (AGI) who explore possibilities to introduce IP-based networks in future airplane cabins. A simplified structure of AGI's data cabin network is shown in Figure 4.2. This network topology is not deployed in today's airplanes but serve as research platform for future deployments. The cabin network includes, for example, the smoke detectors, and the passenger service units with lights, and service buttons. Those devices are included into single seats or seat rows and represent a single client for each functionality. All devices are coupled using switches lined up to cope with the huge number of single devices. Those line switches are bundled into the main cabin switch since every communication has to cross through.

This main cabin switch offers a monitoring port, a Network-based Intrusion Detection System (NIDS) can be plugged in and all network traffic can go through the NIDS. Moreover, all servers available are connected to the main cabin switch in order to provide services for all clients in the network. Communication between servers is not going through this switch but instead a separate backbone switch is used.

One envisioned network is based on SOME/IP, but without any service discovery features. Instead, the clients know in advance all possible services and methods offered by the servers, as well as their MAC and IP addresses. Those services are all located on the central servers connected to the main cabin switch. The services and methods, each client is allowed to use, are restricted. Those restrictions include access restrictions to whole services or are limited to certain methods. All restrictions are known in advance.

Some clients send notifications to specific servers on a regular basis. For those services, the exact intervals used, as well as the corresponding services and methods, are also known in advance. Additionally, the servers awaiting those notifications are known.

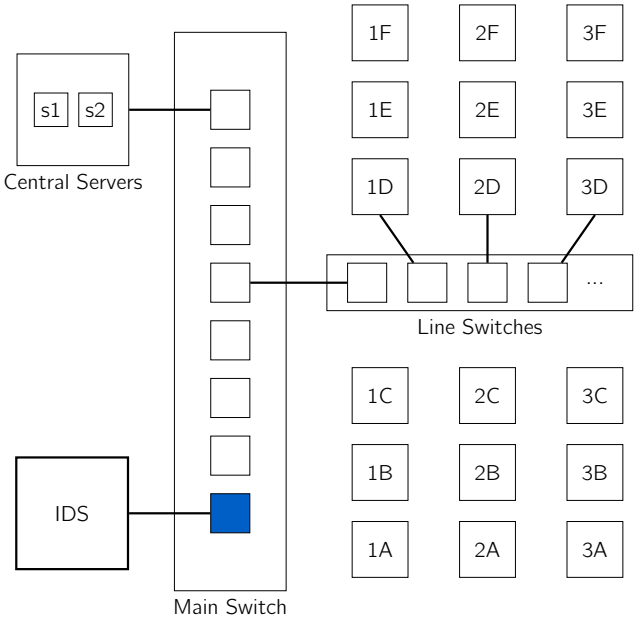


Fig. 4.2: Simplified Airbus Data Cabin Network

4.1.4 Requirements

In the following we describe the requirements an IDS has to fulfill in order to generate alerts that are the basis for automated intrusion response and to be applicable to our use case scenario, where a given protocol needs to be analyzed in detail.

Requirement R1 – Protocol Behavior Description: The ability to describe the protocol behavior is the first requirement. This includes describing packet relations that are expected as well as packet relations that are not allowed. An example for expected behavior is that a response will be followed by a request. An example for forbidden behavior is that an error message is never answered by another error message. Additionally, header field definitions based on single header fields as well as the combination of header fields within a single packet need to be addressed.

Requirement R2 – Misbehavior Detection: The system has to detect protocol misbehavior or anomalies that means deviations from the specified model. The ability to specify the model of the protocol to observe is stated in Requirement R1. The system must not rely on attack signatures that have to be updated continuously, but rely on the complete specification of the expected behavior and specification of the protocol.

Requirement R3 – Continuity: The system has to work in a continuous manner, meaning processing the incoming data and generate an alert just-in-time. This means that as soon as an event happens, meaning a packet arrives, the analysis should be triggered and malicious actions have to be identified. The system should not rely on a block by block execution, as this introduces blind spots the attacker can misuse to attack the system unnoticed. As soon as the attack is visible the alert has to be triggered.

Requirement R4 – Low False-Positive Rate: The system should not produce false-positives as false information will lead to wrong decisions in downstream processes and unnecessary responses will be taken that overload the system and may lead to wrong configurations. Learning-based mechanisms to derive a model tend to higher false-positive rates as they may not have the complete information they need to derive the model correctly. Explicitly specifying the model according to an available protocol specification

can overcome this issue. False-positives do not include misconfigurations as they can be handled in subsequent intrusion response.

Requirement R5 – Usability: Rules for the IDS should be easy to write and deploy without complex adaptations to the system itself. Adding additional rules should not require to adapt the system itself or add additional components to the IDS. Writing rules should be comfortable even for complex relations between single packets within a stream.

4.2 System Design

As the proposed system is based on CEP, first, a short introduction into CEP is given in Subsection 4.2.1. The proposed system design is presented in Subsection 4.2.2. The required input providing necessary knowledge to the system is described in Subsection 4.2.3.

4.2.1 Complex Event Processing

CEP is part of the *Information Flow Processing (IFIP) domain* that comprises distributed applications that process continuously flowing data from different sources that may be geographically distributed [25]. This domain is split up into *complex event processing* and *data stream processing*. Data stream processing models this continuously flowing data as data streams, while complex event processing refers this data as single items like notifications or events [25].

An event can be treated as any occurrence from the real physical world, e.g. a single SOME/IP packet in the network. As those events occur in an infinite sequence that is ordered and linear, such sequences are called stream [10]. CEP describes methods, techniques as well as tools dealing with events as they occur. This means CEP is real-time and continuous [41, 98].

CEP is able to extract additional information from events by combining them (*Event Patterns*) for example correlating or aggregating them. Those new compositions are called *complex events*. Those correlations are often temporal relations, for example the existence of a later event or the non-existence of such an event [16] or spatial [10].

4.2.2 Proposed System Design

The proposed design is based on the traditional rule-based NIDS architecture. The system runs on a device which can monitor all traffic in a network, e.g. by using the monitoring port of a central switch. A knowledge-based approach minimizes the amount of false positive alerts and is well-suited for tightly specified environments such as aircraft cabins or internal car communication. The basic idea is to specify and model the expected protocol behavior, as this information is available within the standardization documents. Deviations from this specified behavior indicate an attack and is therefore considered as a security incident.

Inspecting single packets in isolation can only determine whether a message is well-formed. This includes the correct setting of single header fields as well as the correct combination of header field settings. For example the current protocol version of SOME/IP has to be set to $0x01$. An example for the correct combination of header field settings would be the following. In case a `RESPONSE` is sent the error code has to be set to $0x00$. This example may be more complicated but still, only one single message is needed to verify if a packet fulfills this requirement.

In order to analyze a whole conversation between two devices, information from multiple packets is needed. To this end, the proposed system feeds the received messages to

a CEP engine [42] as a stream of events, one event per message. The event is parsed and reduced to its meta data, i.e. the header fields of the SOME/IP protocol as well as the underlying protocols. This includes the MAC header information, the Internet Protocol (IP) as well as TCP or UDP header information. The payload itself is not part of an event. Of course this scheme is applicable to all protocols providing header information and not limit to the use of the SOME/IP protocol.

Instead of single events, the CEP engine applies its rule set to a sliding window on the stream of incoming messages. Such a sliding window can be based upon timing definitions, e.g. all packets arriving in a certain amount of time, or a simple packet counter. This means that the view of the CEP has to be limited to a certain amount of packets. The rules applied to those windows only trigger if the condition they describe is fulfilled within this window. As the rules are evaluated every time a packet arrives, the system reacts in a timely manner, meaning just-in-time in case of a security incident. In case the window is full the last seen event leaves the window if a new event occurs. The rules applied to the windows may aggregate, join, and group messages in the window allowing to apply more complex rules than checking only a single packet.

As the window size is an important factor of CEP, as the more events are stored in a window the more events have to be checked to investigate whether a rule has to be triggered or not, the window size has to be limited as much as possible. Prefiltering events that are going to be stored within the event window is be a good option. For example matching on certain header fields to distinguish whether or not a packet is of interest limits the size of events that have to be stored for a longer time.

4.2.3 Knowledge as Input

In order to write the rules the system has to apply, two categories of knowledge are required:

- The protocol specification and
- System or use case related meta data.

The protocol specification is the basis for writing rules as packet sequences and header fields as well as their combinations are laid down. Additionally, meta data can be used to render rules more precisely or do some optimizations within the rules.

Within our use case, AGI's data cabin network, the following information is known before-hand:

- MAC addresses and IP addresses of all servers
- The offered service and method IDs, and expected message types of each server
- MAC addresses, IP addresses, and client IDs of all clients
- The services and methods of which server a client is allowed to use
- Timing definitions of notifications (whose message type is either NOTIFICATION or REQUEST_NO_RETURN)

This information allows us to implement more efficient rules, as the window length of the rules can be shortened. The details are described in Section 4.3.

4.3 Implementation

A working prototype is implemented in Java to demonstrate the feasibility of the approach. Esper [46] is chosen as the CEP engine. It accepts rules written in the *Event Processing Language (EPL)* [47], a subset of the Structured Query Language (SQL), extended with features for stream processing. Thus, rules can be written in a familiar, easy-to-learn, high-level language. Esper and EPL are described in Subsection 4.3.1. The implementation itself, as well as the SOME/IP packet generator used in the evaluation is described in Subsection 4.3.2. In Subsections 4.3.3 to 4.3.5, we illustrate how the classes of protocol violations from Subsection 4.1.2 can be realized with complete EPL examples.

4.3.1 Esper and EPL

Esper [46] is a CEP engine written in Java to analyze series of events. Esper is implemented as real-time engine, that means an action is triggered in case a predefined condition within the event stream occurs. Events are represented as Java Objects like Plain Old Java Objects (POJO) or JavaBeans. EPL is used to define queries that are registered within the Esper engine to describe the triggering conditions. In case an EPL rule triggers, a listener class is invoked to execute an action.

In contrast to an in-memory database, Esper does not store the data itself and performs queries on-top, but stores the queries and let the data flow through the queries. Esper turns the database concept upside-down, to allow a more continuous execution procedure.

Within Esper, events can be modeled and represented. For an event, representation types of information are available, called *event type*. Events can be configured statically within the configuration at startup or dynamically during runtime. An event is defined as an immutable record that describes an action or change within the state occurred in the past. Each event may have different properties characterizing the event itself. Those properties can be static, dynamic, or even nested.

The feature list of Esper and EPL is extensive, therefore, only a few selected and used features are examined in short. EPL supports multiple kinds of data windows, for example sliding windows based on time or length as well as combined windows providing for example an intersection of multiple windows. Additionally, traditional tables can be used as global data structures to perform co-aggregation of multiple statements. Esper allows parallelization for the same and for sub-queries, and is multithread-safe.

4.3.2 SOME/IP – Analyzer and Generator

The implementation consist of the SOME/IP – Analyzer written in Java and using Esper and of the SOME/IP – Generator for producing SOME/IP Packets written in Python3. Both components are shortly examined in the following.

4.3.2.1 SOME/IP – Analyzer

For analyzing and event generation, that means transforming a network packet into an event, two separate threads are used, namely `EventSender` and `Analyzer`.

The `EventSender` can either read the packets to analyze from a network interface or from a given `.pcap` file. To be able to interpret SOME/IP packets as events, the network packets need to be parsed. The network packets are acquired and parsed with the library `jnetpcap`. A deserializer for SOME/IP packets was added, since `jnetpcap` does not have one. Currently, only UDP packets which contain exactly one SOME/IP packet are

supported. The SOME/IP protocol potentially allows containing several SOME/IP messages in one UDP packet or using a TCP stream. Events representing a single SOME/IP packet are transformed into a dedicated event class. This event class provides attributes for all needed header information from the MAC, IP, UDP and SOME/IP headers.

The `Analyzer` starts the Esper engine and registers the SOME/IP events given as POJO. Additionally, a rules file defines the rules to be loaded into the engine. For each rule a listener is instantiated that triggers in case the rules matches. Besides the rules file the configuration file is read that specifies the predefined knowledge described in Subsection 4.2.3. This configuration is given in `Extensible Markup Language (XML)` format and defines server and client instances including their MAC and IP addresses.

As the implementation of the used EPL rules is a central part of this work, they are examined in separate Subsections 4.3.3 to 4.3.5. The implementation of the proposed system, the rules file, and an example configuration are available on GitHub [136].

4.3.2.2 SOME/IP – Generator

The packet generator is written in Python3 and generates valid SOME/IP packets and integrates predefined attacks within the valid packet stream. The generator allows to spawn multiple server, client and attacker instances running in different processes. The attacker acts as a man-in-the-middle and can execute attacks based on incoming messages. The attacker can simulate the following attacks:

- `sendErrorOnError` will send an error message as reply to a previous error
- `sendErrorOnEvent` will send an error message as reply to a previous notification
- `fakeResponse` will send a spoofed response message to a client
- `disturbTiming` will add or remove heart beat messages
- `wrongInterface` will change the predefined interface information within a valid packet
- `fakeClientID` will send packets with a client ID of other valid clients

All server and client instances will behave according to the protocol specification and their configurations. Clients as well as servers can be configured using two configuration files given in `XML` format. Within the `device.xml` all available devices (server, client and attacker) are specified by name, type, IP, MAC and sending as well as receiving port. Within the `services.xml` all available services are defined including their ID, a method description and servers offering those services. A detailed description of the Generator can be found here [1]. The implementation of the generator as well as example configurations are available on GitHub [137].

4.3.3 Malformed Packets

As a first example, a packet is examined in isolation. Checks of the *malformed packets* class, for example whether a packet is malformed, or for allowed IP addresses or port numbers, are implemented this way. The rule of Listing 4.1 fires when the interface version is not a predefined value.

[1] http://some-ip-generator.readthedocs.io/en/latest/index.html

Listings 4.1: Check for Malformed Packets

```
1 SELECT * FROM SomeIPPacket.win:length(1)
2     WHERE interfaceVersion != INTERFACE
```

In Line 1 we define the stream and the window size to look at. The stream *SomeIP-Packet* contains all SOME/IP packets arriving at the Esper engine. In this case we look at only one packet at a time, so the window size is 1. In Line 2 we compare the interface version set in the packet with the predefined allowed value. This predefined value is configured as an Esper internal variable at engine startup.

Using this kind of rules, several checks on packets well-formdness can be done. Those checks are not limited to only one header field, but can be extended to check for the correct combinations of header fields.

4.3.4 Protocol and System-Specific Violations

In order to check for protocol or system-specific violations, more EPL features are used. In the following we will show, how checks for certain changes in the system can be observed and how conversations can be validated.

4.3.4.1 Checking for Changes

The AUTOSAR standard requires the client ID to be unique within the system. In this example, we use Esper's *Named Tables* to track which client, identified by its IP address, uses which client ID. One table maps IP addresses to client IDs, another table holds the reverse mapping from IDs to IP addresses. For each incoming message, both tables are checked. If the pair of IP address and client ID of the incoming message is not contained in the tables, it is added to both tables. If there is already a mapping and the mapping differs, an alert is generated, and the existing mapping is updated.

The query in Listing 4.2 checks whether a device changed its client ID. The complementary query, checking whether a client ID is used by a different device, is not shown as only a few adaptations with respect to the checked table are required.

Listings 4.2: Check of Changed Client ID/IP Assignment

```
1  ON SomeIPPacket s MERGE clientMappingIP cm
2      WHERE (s.srcIP IN (select client_ip FROM clientMappingIP)
3          AND (s.srcIP IN (clientIPs))
4          AND (s.srcIP = cm.client_ip AND s.clientID != cm.client_id))
5  WHEN MATCHED THEN
6      UPDATE SET cm.client_id = setClientID(s)
7  WHEN NOT MATCHED
8      AND s.srcIP NOT IN (SELECT client_ip FROM clientMappingIP)
9      AND s.srcIP IN (clientIPs) THEN
10         INSERT INTO clientMappingIP SELECT s.srcIP AS client_ip, s.clientID AS ↩
               client_id
```

In Line 1 of Listing 4.2 the SOME/IP event is automatically merged into the named table created beforehand. In case of a match (cf. Line 5) the new entry is added, if no match is found (cf. Line 7) the corresponding values are updated and an alert can be triggered. The query in Listing 4.2 uses the predefined variable `clientIPs` in Line 3 representing an array with all allowed client IP addresses read from the configuration. In Line 6 a user-defined function is called, that throws the alert and returns the new client ID the table is updated to.

4.3.4.2 Validating conversations

A sequence of corresponding messages between two devices can be easily analyzed with EPL. Thus, the system can check for protocol violations. The remaining listings will implement the following requirements from the SOME/IP standard:

- An error message should not be answered with another error message.
- Notifications should not be answered with an error message.
- Requests have to be answered with either a response or an error message.
- A response should only be sent to answer an open, earlier request.

Currently, the implementation does not handle retransmits.

The rule in Listing 4.3 checks whether an error was answered with another error. To check whether a notification is replied to with an error, only Line 3 has to be modified to check for the type `ERROR`.

Listings 4.3: Check for Correct Error Behavior

```
1  SELECT * FROM SomeIPPacket(type = ERROR).win:length(1) s1
2      WHERE NOT EXISTS
3          (SELECT * FROM SomeIPPacket(type = REQUEST OR type = ↩
               NOTIFICATION OR type = REQUEST_NO_RETURN).win:length(100) s2
4              WHERE s1.serviceID = s2.serviceID
5              AND s1.methodID = s2.methodID
6              AND s1.requestID = s2.requestID
7              AND s1.srcIP = s2.dstIP
8              AND s1.dstIP = s2.srcIP
9              AND s1.srcMAC = s2.dstMAC
10             AND s1.dstMAC = s2.srcMAC
11             AND s1.srcPort = s2.dstPort
12             AND s1.dstPort = s2.srcPort
13             AND s1.time_stamp > s2.time_stamp
14             AND s2.time_stamp < s1.time_stamp + δ)
```

For every incoming error (see Line 1) we check for a corresponding request (or other packet) (Line 3) with the same settings (Line 4 to 12). We use the prefiltering capabilities of Esper in Line 1 and Line 3 allowing to reduce the stream size of considered events. In Lines 13 to 14 we check for the timestamps, that means the current timestamp is greater than the potential match and within the maximum response time of the servers, δ microseconds. We use a length definition instead of a time definition in Line 3 for the window size because we are analyzing a captured dump instead of a live capture. Note that the window size of stream s2 in Line 3 is selected based on the number of incoming packets as we read the information from a `.pcap` file. In case we would listen to an interface, choosing the window size based on times, e.g. the server responding times, would be more appropriate.

The following statement in Listing 4.4 finds requests that were never answered. Line 11 is an abbreviation for the Lines 4 to 12 in Listing 4.3. The difficulty with checking this requirement is that events have to be checked after some time has passed. The first event coming in in the client sending the request, if a request will finally arrive is unknown. Therefore, we have to assume that after a certain time, no response will arrive anymore.

Listings 4.4: Check for Missing Responses

```
1  SELECT * FROM
2      SomeIPPacket.win:length(2) s1,
3      SomeIPPacket.win:length(2) s2,
4      SomeIPPacket(type = REQUEST).win:length(100) s
5      WHERE s1.time_stamp > s2.time_stamp
6      AND s3.type = SomeIPPacket.REQUEST
7      AND s3.time_stamp < (s1.time_stamp − δ)
8      AND s3.time_stamp > (s2.time_stamp − δ)
9      AND NOT EXISTS
10     (SELECT * FROM SomeIPPacket(type = RESPONSE OR type = ERROR).win:↩
           length(50) s4
11         WHERE (s3 corresponds to s4)
12         AND s3.time_stamp < s4.time_stamp)
```

A request can be answered within a certain time window. Only afterwards the decision can be made that the request was never answered. Therefore, we utilize the timestamps of the last two packets that arrived in the stream (s1 and s2) to check for request packets lying in the past in stream s3. The window we check in the past corresponds to the maximum response time of the server. This window selection is done in Lines 5 to 8. In case we find a request in this window, we check if the corresponding response arrived afterwards (see Lines 9 to 12). Note that in case that no packets arrived so far or no packets exist in the considered time window the rule will not be triggered.

The last example checks whether a response was sent twice in Listing 4.5.

Listings 4.5: Check for Missing Requests

```
1  SELECT * FROM SomeIPPacket(type = RESPONSE).win:length(1) s1
2      WHERE NOT EXISTS (
3          SELECT * FROM SomeIPPacket(type = REQUEST).win:length(100) s2
4              WHERE (s1 corresponds to s2)
5              AND s1.time_stamp > s2.time_stamp
6              AND s2.time_stamp < s1.time_stamp + δ))
7          OR
8            ((SELECT COUNT(*) FROM SomeIPPacket(type = REQUEST).win:length↩
                 (50) s
9                WHERE (s1 corresponds to s)
10               AND s1.time_stamp > s.time_stamp
11               AND s.time_stamp < s1.time_stamp + δ)
12             =
13            (SELECT COUNT(*) FROM SomeIPPacket(type = ERROR OR type = ↩
                 RESPONSE).win:length(50) s
14               WHERE (s1 equals s)
15               AND s1.time_stamp > s.time_stamp
16               AND s.time_stamp < s1.time_stamp + δ
17               AND s.time_stamp > minValue )))
```

The query shown in Listing 4.5 uses an abbreviation for checking if packets are identical in Line 14. In contrast to the correspondence check, were we check for the answer to a packet, we check for packets with identical settings. First, we check for every incoming packet if a requests exists that corresponds to the response (Line 1 to 6) in a given time window that reflects the maximum server response time.

In case we found corresponding requests, we have to check if those requests were already answered with responses (Line 8 to 17). As the time window we check can contain a response that was sent for a previous request not in the checked time window, we have to limit the scope of requests taking into account to the last response seen. Otherwise, the query would fail. Therefore, we use the variable `minValue` in Line 17.

To set `minValue` to the timestamp of the last response taken into account we use the following query in Listing 4.6.

Listings 4.6: Helper Query to Set the Minimum Timestamp to be Considered

```
1 ON SomeIPPacket (type = RESPONSE) AS s1
2     SET minValue = (
3         SELECT MIN(s.time_stamp) FROM SomeIPPacket(type = REQUEST).win:↩
              length(100) s
4             WHERE (s1 equals s)
5             AND s1.time_stamp > s.time_stamp
6             AND s.time_stamp < s1.time_stamp + δ
```

To ensure that the helper query is executed before the main query, both queries are annotated with a priority, which tells Esper the execution order of the queries.

4.3.5 Timing Issues

Some components will send notifications to another device at a fixed frequency, for example a sensor to a controller. If no message was sent for a predetermined amount of time, the device may be offline. On the other hand, if the device sends notifications too frequently, it may indicate a broken device, or a DOS attack by the means of flooding.

In the following example, we show how we can check those timing constraints in EPL. The rule in Listing 4.7 checks whether messages are sent more often than defined by an interval δ.

Listings 4.7: Check for Timing Constraints

```
1 SELECT * FROM SomeIPPacket(clientID = id, methodID = x, serviceID = y).win:length↩
      (1) AS s1
2     WHERE NOT EXISTS
3         (SELECT * FROM SomeIPPacket(clientID = id, methodID = x, serviceID = y).↩
              win:length(2) AS s2
4             WHERE s2.time_stamp < s1.time_stamp − δ)
```

In Line 1 we filter the stream to only include the specific clients and notifications we are interested in. We select those messages from a window of size 1 (Line 1), and a window of size 2 (Line 3), which will also contain the previous message. Line 4 checks whether more than δ microseconds have passed between those messages. We can add a rule like this one for each timing constraint in the system specification.

4.4 Evaluation

Within this evaluation section, we first compare our system to the requirements stated in Subsection 4.1.4 (see Subsection 4.4.1). Further, we provide a performance analysis as a quantitative analysis covering execution time and memory consumption within this section. The used test setup for this evaluation is explained in more detail in Subsection 4.4.2. We evaluated the time consumption of the system using only a single rule

(see Subsection 4.4.3) and multiple rules in parallel (see Subsection 4.4.4). Finally, we investigate the memory consumption using multiple rules (see Subsection 4.4.5) and with a varying window size (see Subsection 4.4.6).

4.4.1 Requirement Alignment

In Subsection 4.1.4 the requirements an IDS has to fulfill if it has to be integrated into an Incident Handling System (IHS) and if it has to be suitable to our use case are stated. In this subsection our proposed IDS is compared to this requirements.

Within the proposed IDS the rules given in EPL reflect the protocol behavior of SOME/IP [7] used exemplary (cf. Requirement R1). In Section 4.3, we have shown that malformed packets, protocol and system specific violations as well as timing issues can be reflected using the proposed IDS. This includes validating conversations and checking for changes. The defined expected behavior defined in the standardization documents can be reflected using the proposed IDS.

The proposed IDS is throwing an alert in case misbehavior is detected (cf. Requirement R2). Misbehavior can either presented by violating explicitly forbidden or allowed actions. This misbehavior is either an attack or a misconfiguration. As both is unwanted in the target system, attacks as well as misconfigurations can be handled by automated intrusion response. The underlying model used to detect deviations is explicit, meaning that no learning phase is required. As no attack signatures but the expected protocol behavior is specified, rules need only to be adapted if the underlying protocol changes but not in case of new attacks.

The proposed IDS can detect an attack in a continuous manner (cf. Requirement R3). As CEP is used and the SOME/IP packets are fed into the system directly, an alert can be triggered as soon as the attack happens. In case the attack is only visible if an expected packet can not be observed, the alert is triggered as soon as the attack is clear.

The proposed IDS provides a low false-positive rate (cf. Requirement R4). As long as the components behave according the protocol no false alert in terms of attacks will be raised. Besides security incidents a misconfiguration of a network element can be detected if this misconfiguration leads to a wrong behavior. A distinction between security incident and misconfiguration can not be detected but a paling IHS can handle a misconfiguration likewise a security incident as a misconfigured device is unwanted behavior of the target system as well. Therefore, responses reconfiguring devices or resetting devices need to be integrated into the IHS.

As the proposed IDS uses EPL the usability is high (cf. Requirement R5). All rules to reflect the protocol behavior can be reflected using EPL. Therefore, no additional components or recompilation are needed if new rules are added to the IDS. As EPL is based on the well-known SQL the language to write rules can be learned easily as complex structures within the stream of packets can be expressed easily in a known language.

4.4.2 Test Setup

In order to evaluate the proposed system, we utilize our SOME/IP packet generator written in Python 3. The generator simulates the behavior of multiple clients and servers, and an attacker performing various attacks. The clients and servers behave according to the AUTOSAR standard [7]. The attacker is capable of all actions and implements the attacks described in Subsection 4.1.2 . The implementation of the packet generator, including configuration, is available on GitHub [137].

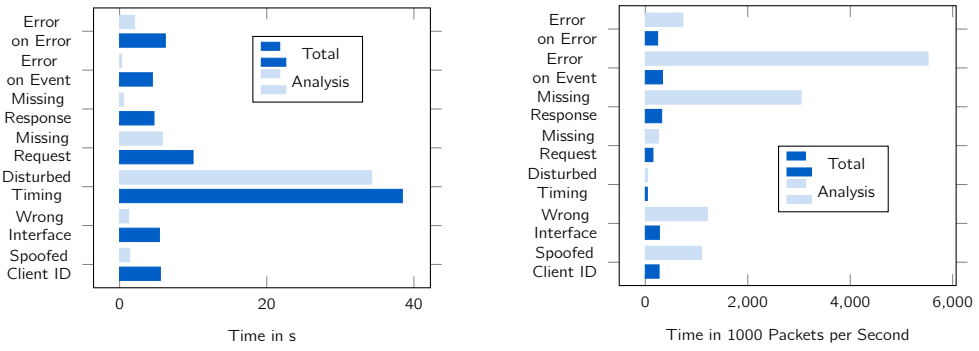
For evaluation, we generated a *libpcap* dump file. This dump contains around 12.000 attacks, with a size of 122.4 MB and containing around 1.49 million packets. All figures show the average over five runs. Memory consumptions were sampled at an interval of 100 ms during execution.

The measurements were executed on a Ubuntu 15.10 system with an Intel Xeon E3-1275v3 CPU at 3.5 GHz and 16 GB of RAM. As of now, the analyzer is single-threaded. The reading and parsing of the *libpcap* dump is executed on a separate thread.

4.4.3 Time Consumption of Single Rules

For each rule, we measure the time needed for analyzing the dump. We calculate the packet rate and the data rate. As the SOME/IP payload is only a few bytes in our test case, we assumed a size of 64 B per frame, the minimum size of an Ethernet frame.

The time to only deserialize all packets and feed them into Esper, without any rules being evaluated, is 4.18 s. About 357.000 packets can be deserialized per second which results in a data rate of 183 Mbit/s. Table 4.2 lists the time measurements for each rule. A graphical representation is given in Figure 4.3a and 4.3b.



(a) In Seconds

(b) In 1000 Packets per Second

Fig. 4.3: Time Comparison for Single Rules

The times we show for *analysis'* is the *total* time measured to analyze the dump minus the constant deserialization time. Times are given in seconds (s). σ is the standard deviation of the total time. The packet rate is given in 1.000 packets per second (kpkt/s). The data rate is given in Megabit per second (Mbit/s). The rule *Disturbed Timing* is actually a bundle of 5 separate rules following the definition of Listing 4.7.

Some rules, in particular Listing 4.1 (wrong interface) and Listing 4.7 (disturbed timing), only have a minor effect on the packet rate, whereas Listing 4.4 (missing response) has a huge impact. The results of these benchmarks indicate that the execution time of a rule increases particularly with the number of sub-queries and the window size, which is to be expected, just like in SQL.

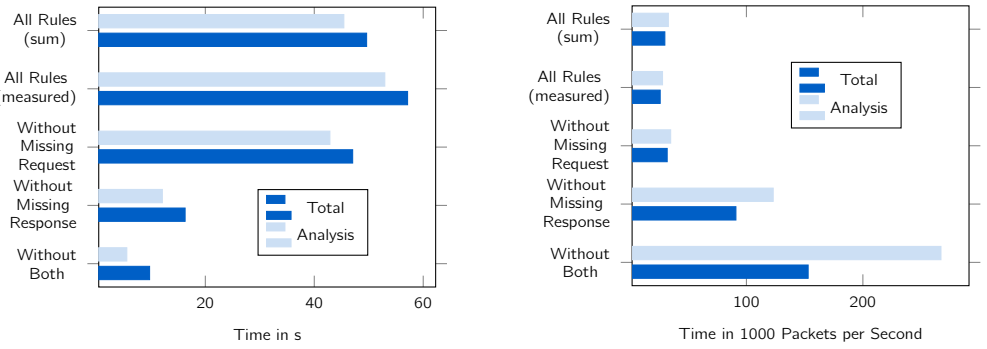
AGI is building a cabin network mock-up for testing experimental devices and protocols. At the moment, the cabin part of the mock-up is equipped with 100 Mbit/s lanes. Therefore, most of the measured rules could handle live monitoring at line rate. The other rules would overwhelm the monitoring system if the lanes are saturated. This would not happen during normal operation, but may happen when a device misbehaves.

Tab. 4.2: Time Consumption per Rule in Seconds

Rule	**Analysis**	**Total**	σ	kpkt/s	Mbit/s
Error on Error (Listing 4.3)	1.36	5.54	0.09	269	137.92
Error on Event (Listing 4.3)	1.23	5.41	0.04	276	141.28
Missing Response (Listing 4.4)	34.25	38.43	0.96	39	19.92
Missing Request (Listing 4.5)	5.80	9.98	0.37	150	76.64
Disturbed Timing (Listing 4.7)	0.49	4.66	0.07	320	163.92
Wrong Interface (Listing 4.1)	0.27	4.45	0.08	335	171.68
Spoofed Client ID (Listing 4.2)	2.03	6.21	0.15	241	123.2

4.4.4 Time Consumption of Multiple Rules

We evaluated the performance of the system when several or all of the rules are enabled. The results are shown in Table 4.3. A graphical representation is given in Figure 4.4a and 4.4b.



(a) In Seconds **(b)** In 1000 Packets per Second

Fig. 4.4: Time Comparison for Multiple Rules

As Listing 4.4 (missing response) and Listing 4.5 (missing request) are comparatively expensive, we first enable all but those two rules (Case 1). Then, we only disable one of those rules (Case 2 and 3), and finally, all rules are enabled (Case 4).

Tab. 4.3: Time Consumption with Multiple Rules Activated

Considered Cases	**Analysis**	**Total**	σ	δ	kpkt/s	Mbit/s
1: without 4.4 & 4.5	5.59	9.76	1.09	0.21	153	78.24
2: without 4.4	12.11	16.29	0.21	0.93	92	46.96
3: without 4.5	42.87	47.05	0.22	3.24	32	16.24
4: all rules	52.95	57.13	0.67	7.52	26	13.36

We are interested in the additional overhead multiple rules may generate. Therefore,

we calculated the difference (δ in seconds) between the sum of the single analysis times of each rule and the analysis time when these rules were enabled in the same run.

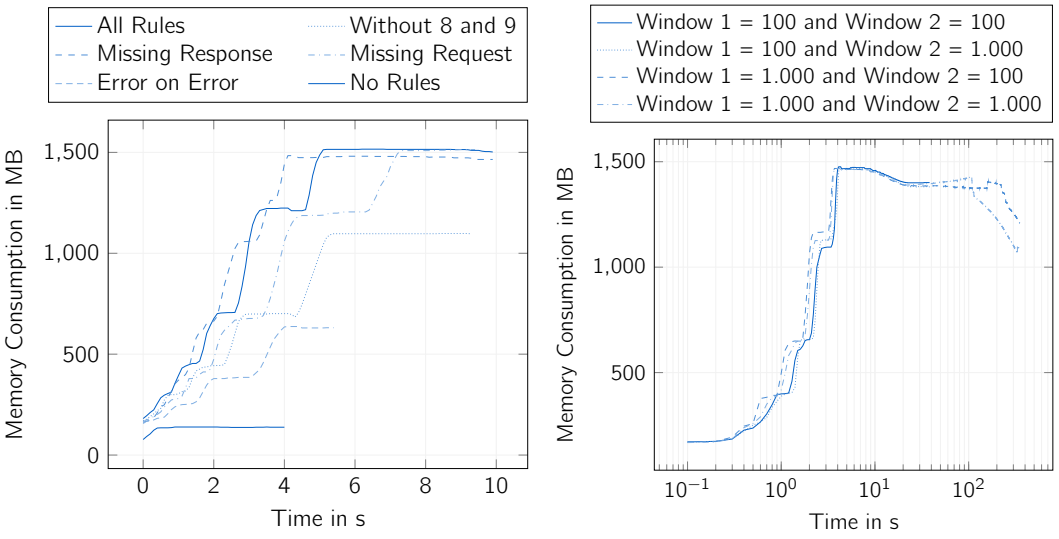
In Case 1, when only inexpensive rules were activated the additional overhead δ was negligible. When more expensive rules were activated, the overhead increased from 3.7 % of the analysis time (Case 3) to 14 % (Case 4).

Regarding the cabin network mock-up, we conclude that the combined data rates are under the 100 Mbit/s maximum data rate. Either, only a subset of the rules can be enabled, or the traffic has to be pre-filtered to decrease the data rate to the IDS. Additionally, multiple instances of the IDS can be deployed using a suitable rule subset.

4.4.5 Memory Consumption with Multiple Rules

Another interesting aspect is the memory consumption of single rules as well as in combination. In Figure 4.5a the results of the memory consumption measurements are shown. We only plot the resident memory in this figure. Shared memory remains below 20 MB at all times. The single process, which includes the Java runtime and Esper besides the analyzer, allocates about 7350 MB of virtual memory in advance. As this do not change over time, we do not include it.

As the memory consumption is sampled at fixed intervals, the number of measurements depends on the running time of the analysis. We only show the first 10 seconds if the process took longer. Even when all rules were activated, the memory consumption did not increase after the first 10 seconds.



(a) Selected Rules

(b) Varying Window Sizes

Fig. 4.5: Memory Consumption for Different Scenarios

Only the checks for missing requests (Listing 4.5) and responses (Listing 4.4) have a noticeably higher memory consumption than the other rules. This is the case even if the remaining other rules are combined with each other. The memory consumption of all rules combined is barely higher than the memory consumption of using only the single rule for missing responses (Listing 4.4) or missing requests (Listing 4.5). As expected, rules with a large window require the CEP engine to keep more packets in memory. However, windows with equal conditions are shared, so adding more expensive rules of the same kind does not increase the memory consumption.

4.4.6 Memory and Time Consumption with Varying Window Sizes

The window size on a stream may have a strong impact on performance if the rule requires a join on large windows, or performs a sub-query for each element in a window. The *missing response* rule (Listing 4.4) works over a large window (Line 4) and performs a sub-query over another large window (Line 10). We evaluated this rule in Table 4.4.

Tab. 4.4: Time Consumption for Listing 4.4 with Varying Window Size

Query Window	Sub-Query Window	Total	σ
100	100	38.9	0.59
100	1000	39.5	0.49
1 000	100	360.5	4.67
1 000	1000	364.0	4.05

In our dump, the cost of the size of the sub-query window (Line 10) is negligible compared to the cost of the main query window (Line 4). Esper has to keep all packets of a window in memory. Figure 4.5b shows the memory consumption for varying window sizes. Note that the x-axis is set to log-scale.

A large window in a sub-query has a negligible impact when compared to a large window size on the main query. This is likely because those sub-queries are executed rarely by the engine. The engine may short-circuit the evaluation of the rule as soon as a *where'*-clause of the main query fails, or a match is found if the sub-query has an `EXISTS` predicate.

As the impact of large windows is significant, the window size should be chosen as small as possible. Correctly pre-filtering the packet stream allows for smaller windows. In future work, generic queries like Listing 4.3 may be made more specific to the content of the stream as for example in Listing 4.7.

4.5 Related Work

A common method for detecting attacks is to collect network traffic and feed it to a NIDS [90]. Popular IDSes, such as Snort [134] or Bro [15], do not ship rules for SOME/IP or other sparse protocols, provide particularly no support for use-case-specific requirements or do not provide an easy and adequate description language for protocols or other requirements. Implementing custom rules in Snort, especially when relations between multiple packets are to be considered, is burdensome. Additionally, Snort is used to define and specify attack signatures and therefore, describes the unwanted behavior instead of the expected system behavior. Consequently, Snort rules have to be updated and adapted for all new attacks. CEP offers a promising alternative for anomaly detection. Those approaches are examined in the following, followed by model-based approaches.

CEP-Based Approaches The authors of [2, 3, 96] have implemented a specialized algorithm for detecting port scans on top of a CEP engine. In [52] a general IDS for detection attack on the network based on CEP is introduced. The authors provide an attack pattern definition for DOS [52]. In contrast, we apply the CEP paradigm to a wide variety of protocol-specific issues as we do not specify concrete attacks to look for in the event stream, but specifying the correct behavior of a protocol and looking for deviations

from this behavior. The advantage of this approach is that even unknown attacks can be detected without adding new rules to the system.

In [10] a solution for object tracking based on CEP for wireless sensor networks is introduces. The focus of this work is the detection and tracking of objects using multiple sensors. Even if they claim to propose an IDS no IDS functionality is implemented or tested within their system.

In [49] a generic Intrusion Detection and Diagnosis System (ID^2S) based on CEP is proposed. The system supports alert correlation for detecting and analyzing complex intrusion scenarios in large-scale complex critical Infrastructures. This work focuses on the collection and interpretation of raw attack information to generate a more high-level view on the network situation rather than detecting protocol misbehavior.

Model-Based Approaches In contrast to model checking [97], the proposed system is used for testing and analyzing implementations of single network components rather than verifying the protocol design itself. It does not require a formal model, but behavioral rules from the standardization documents.

In [125] a specification-based approach towards anomaly detection within computer networks is proposed. They specify the protocol behavior defined within RFCs or other specifications using an EFSA (extended finite state automata). An EFSA is similar to a finite-state machine but transitions can be done based on events and state variables can be utilized. For each packet a new instance of the state machine is needed and additionally handed over to all existent state machines. A state machine reaching a finite state is removed from the list of available state machines. The authors admit many instances of those state machines will be created. That why they combine their approach with statistical machine learning techniques. They map statistical characteristics of the packet stream, like frequency, most common values and value distribution to properties of the EFSA. This approach does not enable the system to identify misbehavior of the network entities regarding the protocol definition, but instead deviations from statistical values regard the protocol.

SYMBEXNET [138, 139] describes a method to verify protocol implementations based on symbolic execution and rule-based specifications. The idea is to generate special packets covering a huge code space of the implementation This system is based on replaying these generated packets and is not designed as an IDS.

Summary, Comparison and Conclusion The requirements stated in Subsection 4.1.4 are mapped to the presented approaches. The results are summarized in the following Table 4.5. Summing up the related work we can come to the conclusion that common and used IDSes do not provide an easy specification language for protocol or use case requirements. Specification-based approaches produce to many instances to be managed such that their capabilities are limited to statistical analysis. Approaches based on CEP only specify attacks to look for within the stream of packets, instead of providing rules for the expected behavior. Protocol verification approaches can not be used for anomaly detection.

Tab. 4.5: Comparison of Related Work Based on the Requirements Stated in Subsection 4.1.4

Approach	R1 – Protocol Model	R2 – Protocol Checks	R3 – Continuity	R4 – Low False Positive Rate	R5 – Simplicity
Snort [134]	✗	✗	✓	✓	✗
Bro [15]	✗	✗	✓	✗	✗
Port Scan Detection [2, 3, 96]	✗	✗	✓	✓	✓
Attack Detection [52]	✗	✗	✓	✓	✓
Wireless Sensors [10]	✗	✗	✓	✓	✓
ID^{2}S [49]	✗	✗	✓	✓	✓
State Machines [125]	✓	✗	✓	✓	✗
SYMBEXNET [138, 139]	✗	✗	✗	✓	✗

Concluding the related work, the unique features of the presented approach are the specification of the correct protocol behavior using CEP rules and finding intrusions by looking for deviation from this modeled behavior.

4.6 Publication Reference

AGI provided the use case of the data cabin network within an airplane, protocol selection to use SOME/IP and the assumptions of given information used as input for the proposed IDS. This data cabin network is not deployed within airplanes but serves as research platform for future cabin networks. The content of this chapter is published on DISSECT (Workshop on Security for Emerging Distributed Network Technologies) 2016 held in conjunction with the Network Operations and Management Symposium (NOMS) (cf. [63]). The own contribution of this paper is the basic idea of using CEP as method for protocol behavior checks, the implementation of the overall IDS and an essential part of the design of the rules as well as the evaluation.

5. RESPONSES IDENTIFICATION

The response identification module is responsible for

- Determining at which point in time automated intrusion response has to be triggered,
- Determine all affected entities, and
- Determine applicable responses (candidate responses) with respect to the affected entities and the security incident.

To be able to realize this functionality, an underlying model for responses in general is needed. In Section 5.1 we cover the response model derived from our information model given in Section 3.3 in more detail. Additionally, we identify requirements the response identification module has to fulfill in order to realize the listed functionality. Further, we explore available responses and classify them according to the features and taxonomies shown in Subsection 2.4.1. In Section 5.2 we introduce the system design of our response identification module and give a detailed overview on how and when to trigger responses appropriately. The implementation of our response identification module is presented in Section 5.3 including how to integrate this module into our Incident Handling System (IHS) proposed in Chapter 3. Within our evaluation in Section 5.4, we show how our response identification module aligns to the requirements stated in Section 5.1.

5.1 Analysis

First, we analyze the requirements a response identification module has to fulfill (see Subsection 5.1.1). Afterwards, a model is deviated from our information model given in Section 3.3 that covers the important aspects of responses in Subsection 5.1.2. In literature a huge amount of different responses is available, but those descriptions and listings are unstructured or domain specific. For selecting appropriate responses, a literature search was done in Subsection 5.1.3 that structures and examines available responses.

5.1.1 Requirements

To identify suitable requirements for the response identification module the needed steps of intrusion response (cf. Subsection 2.5.2) and available related work are taken into account. The following requirements need to be fulfilled for response identification:

Requirement R1 – Flexible Response Triggers: Depending on the use case or the underlying target system, automated intrusion response is applicable under varying conditions. The response identification module has to provide the ability to adjust under which conditions automated intrusion response is applied. Therefore, possible features of the `Alert Context` node have to be collected and evaluated against specified rule sets. The use of triggering responses is proposed in [141].

Requirement R2 – Continuous Execution: During automated intrusion response the conditions may change. For example, a security incident that is handled by the system

is extended by additional information, like additional victims within the target system or additional consequences observed. The response identification module has to be able to cope with changing environmental conditions with respect to the ongoing security incident. To achieve an adaptable Intrusion Response System (IRS) as proposed in [142] the response identification module has to work continuously.

Requirement R3 – Low Concurrent Response Execution: As it is not suitable to automatically respond to every single incoming alert, the response identification module has to balance between a prompt initiation of responses and to find an operation base as large as possible. This means that the number of entities covered with automated intrusion response has to be large enough such that synergy effects can appear, e.g. responses effective for multiple targets.

Requirement R4 – Consider the Target System: To identify the candidate responses applicable to a security incident, the response identification module has to identify all affected entities in the target system. This allows automated intrusion response to cope with the whole security incident instead of responding to single attacks on single network elements. Therefore, the response identification module needs to filter and combine all targets from the `Alert Context` node triggering automated intrusion response including connected `Alert Context` nodes, either connected as superset or subset.

Requirement R5 – Consider the Security Incident: The response identification module has to be able to take the concrete security incident into account. The type of an attack limits the course of actions automated intrusion response has.

Requirement R6 – Identify Applicable Candidate Responses: The response identification module has to be able to determine all applicable responses that are a course of action. Therefore, the system has to be able to combine available (cf. Requirement RQ3) and suitable responses (cf. Requirement RQ4).

5.1.2 Response Model

From the information model (cf. Section 3.3) and the background information on responses (cf. Section 2.4), we extract the following features for a response model.

Execution Target Each response has a specific target. This means a network entity it is designed for. The network entity being the target of a response is part of the infrastructure information of the information model described in Section 3.3. In this model we presented the following targets, `Network` nodes, `Device` nodes, `Service` nodes, `User` nodes, `Certificate` nodes, `Virtual Machine (VM)` nodes, `Process` nodes, `Connection` nodes, or `File` nodes. Those categories can be subsumed from the list of existing responses prepared from the related work. This information is independent from the underlying target system and applies to all use cases. This information has to be provided by an expert designing responses.

For example, a typical response is to restart a system. The system to restart may be a device or VM, service, process or connection. Other mentioned targets, e.g. a user, are not applicable. The response has a clear semantical meaning with known output and implications on the target system. It can not be applied directly to the target system as different implementations may be possible.

Available Implementations Each generic response has to has at least one concrete implementation realizing a designed response. A designed response can have multiple re-

sponse implementations that differ in certain metrics. A response implementation depends on the underlying target system as it utilizes the given environment.

Restarting a system can be implemented in multiple ways. For example, restarting a VM depends on the virtualization solution that is used. Restarting a device depends on the operating system used. Additionally, restarting a VM can be done by utilizing the hypervisor or logging on the machine and using the underlying operation system.

Providing multiple implementations for a single response helps to adapt more precisely to the current situation a response has to be executed. For example, logging on a VM using `ssh` and shutdown the VM may be a virtualization agnostic solution, but the targeted VM has to be reachable and still under control of the operator.

Conflict Relations Responses can be related with each other. An important relation is the *conflict* relation expressing that two responses can not be executed together based on their design. If two responses are conflicting all their implementations are conflicting. Apart from conflicting responses that express conflicting concepts, implementations of responses can conflict. Those conflicts are based on the realization of a response.

For example, restarting a VM and restoring a VM are conflicting by design. The VMs can be restarted with either the old image or a fresh image. Taking both actions together will result in a non-deterministic state of the target system. Consequently, all implementations of restarting a VM conflict with all implementations of restoring a VM.

Conflicts can additionally arise from the use case or the target system itself, for example, due to resource restrictions. For example, due to resource restrictions on the hosting server only a limited number of VMs can be migrated at the same time.

Dependency Relations *Dependency* relations are used to express that before a response can be executed, some preconditions in form of additional actions are required. Those additional actions can be considered as responses as well not helping an entity to be freed from an attack (no execution target). This relation is useful to describe responses in a more flexible manner as they can be designed more fine-grained. Additionally, responses can benefit from synergy effects in case multiple responses need the same preconditions. Precondition actions do not target a network entity, but instead prepare the target system.

Analog to the conflict relation, dependency relations can be applicable to responses or implementations. In case a response is precondition of another response, one implementation of the precondition response has to be executed before an implementation of the response itself.

For example, migrating a VM offering a services can only be done if the routing or load balancing is adapted and the migration is announced to the users of the service. Instead of offering a single response that bundles all those single actions, they can be split up into smaller more manageable parts that are reusable for other responses.

Executing Entity A concrete implementation of a response is executed by a specific device within the target system. This implies that the response implementation is available on the executing entity or can be deployed on this entity. This relation is specific to the underlying target system and is needed in order to eventually execute the response on the target system, for example a specific firewall or router.

Deployment Target The deployment target describes which entities a response implementation can reach and effect during execution. In contrast to the execution target

describing a potential class of network entities applicable to a response, the deployment target describes the concrete applicability of a concrete response.

For example, a response targeting a network can be potentially executed on every network within the target system. But a response implementation has to be deployed on a network entity that reaches the network, e.g. isolating a concrete network can only be done by a router (executing target) connected to that specific network. The deployment target in this case is the isolated network.

Mitigated Consequences A response can mitigate certain consequences occurring as a result of a security incident. Instead of binding a response to a detected security incident, modeling consequences allows to re-use responses helpful against the same consequence resulting from different attacks.

For example, certain attacks can compromise the RAM of a system. A helpful response is to restart the system as the infection is limited to non-permanent memory. The type of an attack (classification) does not matter, as only the implications are considered.

Metrics A response implementation is determined by different metrics defining the implementation with respect to different features. Depending on the use case those metrics may differ. These metrics are used later to determine the optimal subset of responses to be chosen for execution. For example, the duration of a response or resulting downtime for affected service can be used as metrics for a response implementation.

5.1.3 Collection of Responses

In this section responses available in literature are examined. Because a huge amount of responses is available, structuring those responses is challenging as the following problems appeared within the literature search. Responses are often a loose collection with no structure. Moreover, they are often described within a limited view and are very domain specific. No model or generalization is available that would ease the use and description of responses. In literature a huge amount of duplicates and inconsistent naming appears. The goal of this section is to structure available responses in order to ease choosing appropriate responses for different use cases and to instantiate our proposed response model.

The identified responses are summarized in Table 5.1. A checkmark (✓) indicates that the given characteristic is fulfilled. A cross (✗) indicates that the given characteristic is not fulfilled. A dash (–) indicates that the characteristic is not applicable.

The examined characteristics from Section 2.4 are used for structuring identified responses and are shortly reviewed in the following:

Response First, we give a description of the response that can be found in column *Response*.

Target The *Target*-column shows which system entities will be affected by a response that is executed. A target can be a *Network (N)*, a *Host (H)*, a *Virtual Machine (V)*, a *Service (S)*, a *User (U)*, a *Process (P)*, a *Session (Se)* or Connection, a *File (F)*, a *Certificate(C)*, or the *Attacker (A)*. Due to naming inconsistencies the term *Host* will be used analog to the term *Device*. The same goes for *Service*, *Application* and *Software*. The distinction between the different targets is subsumed from the related work and is useful to identify suitable responses for the underlying target system in order to determine the execution target.

Location To describe the location a response can be executed on the definition found in [128] is used. As short notation *T* is used for the target machine, *M* is used for the mid points, *F* is used for firewall or router points and *I* is used for the attacker (intruder) machine. This information is needed to instantiate the executing entity.

Properties From [5] the distinction in active and passive responses is used. The *A*-column indicates whether or not a response can be considered as active. As only reactive responses are of interests for this thesis the differentiation between reactive and proactive is set aside. The description of sustainable and indefeasible responses from [80] is indicated by the *S*-column and *I*-column. This information is needed to decide whether or not a response is applicable to a specific use case. If a response is not sustainable, than the *I*-column is not applicable indicated by the dash (–).

Classification The classification given in [181] is described in the *C*-column. The following short notation is used. *RB* is used for rollback, *RF* is used for roll-forward, *I* is used for Isolation, *RC* is used for reconfiguration and *RI* is used for reinitialization. This information is needed to decide whether or not a response is applicable to a specific use case.

References This column lists the references the response is listed, described, or exemplary used.

Some of the presented related work have special concerns, like special use cases, attack scenarios, or dedicated environments the responses are proposed for. In the following we shortly list this kind of related work and point out their special concern:

- In [57] the use case of *Mobile Money Transfer Service (MMTS)* is considered. They assume an account take-over as concrete attack.
- In [177] the *Darpa intrusion detection data set LLDoS 1.0* is considered and for attacks within this data set exemplary responses are selected.
- In [80] password cracking and partial Denial of Service (DOS) attacks are considered.
- In [77] a DOS attack on *VoIP services* is explained.
- [76] focuses on database specific responses.
- In [87] some responses for mobile ad-hoc networks are specified and parameterized.
- In [83, 84] a service dependency graph is used to determine the appropriate response for services under attack.
- In [48] the use case of *Advanced Metering Infrastructures (AMI)* is described.

Tab. 5.1: List of Available Responses Found in Literature

Response	Target										Location				A	S	I	C	Reference
	N	H	V	S	U	P	Se	F	C	A	I	M	F	T					
Suspend or halt	✗	✓	✓	✓	✗	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	I	[21, 76, 103, 124, 135, 142, 177]
Shut down, kill, abort, or close	✗	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✗	I	[6, 8, 21, 23, 76, 83, 84, 87, 103, 124, 129, 142, 146, 149, 166, 177]
Reboot or restart	✗	✓	✓	✓	✗	✓	✓	✗	✓	✗	✗	✗	✗	✗	✓	✗	–	RB	[8, 48, 124, 129, 142, 149, 181]
Change passwords	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	RI	[79, 80, 103, 149]
Disabling, deactivation or lock	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	I	[21, 23, 57, 87, 103, 124, 129, 142, 166]
Revoke or deny privileges	✗	✗	✗	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗	–	I	[21, 76, 142]
Change privileges	✗	✗	✗	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗	RC	[8, 21, 83, 84, 129, 174]
Quarantine or sandbox	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	I	[8, 21, 23, 48, 76, 83, 84, 87, 103, 129, 142, 146, 165]
Restore	✗	✓	✓	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗	RI	[6, 8, 103, 142, 149, 181]
No response	–	–	–	–	–	–	–	–	–	–	–	–	–	–	✗	–	–	–	[57, 76, 124, 177]
Traceback	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	–	–	[21, 48, 124, 142, 177]
Scrutinizing ports	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	–	–	[177]
Block or disconnect	✓	✓	✓	✓	✗	✓	✓	✗	✓	✗	✗	✗	✓	✓	✓	✓	✓	I	[6, 8, 21, 23, 48, 83, 84, 87, 129, 142, 146, 149, 165, 177, 181]
Remove, revoke or format	✗	✓	✓	✗	✓	✗	✗	✓	✓	✗	✗	✗	✗	✓	✓	✓	✗	I	[8, 48, 83, 84, 87, 142, 149, 177]
Send an alert, report, log, or notification	–	–	–	–	–	–	–	–	–	–	–	–	–	–	✗	–	–	–	[8, 21, 48, 57, 76, 124, 142, 166, 174, 177]

Continued on next page

Table 5.1 – *Continued from previous page*

Response	Target										Location				A	S	I	C	Reference
	N	H	V	S	U	P	Se	F	C	A	I	M	F	T					
Notify neighborhood	–	–	–	–	–	–	–	–	–	–	–	–	–	–	✗	–	–	–	[107]
Deny	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗	–	I	[57]
Activate filter (reduce, shape, drop)	✓	✗	✗	✗	✗	✗	✓	✗	✗	✓	✗	✗	✗	✓	✓	✓	✓	RF	[6, 8, 23, 48, 57, 65, 77, 79, 80, 83, 84, 124, 142, 149, 172, 174]
Additional security measures	✗	✓[1]	✓	✗	✓[2]	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	✓	✓	–	[21, 23, 48, 57, 76, 124, 142, 146, 149, 165]
Slow down or delay	✗	✗	✓	✗	✗	✓	✓	✗	✓	✗	✗	✗	✗	✓	✓	✗	–	I	[8, 21, 135, 142, 146, 172]
Redirect or adjust routing	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✓	✓	✗	✓	✓	✓	RF	[23, 83, 84, 87, 149]
Create backup	✗	✓	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✓	✗	✗	–	–	[21, 87, 142]
Unmount file system	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	I	[8, 83, 84]
Change owner	✗	✗	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	RI	[8, 83, 84]
Update or patch	✗	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✗	RF	[48, 149, 181]
Use redundant system	✗	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	RC	[23, 48, 181]
Profile behavior	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✓	✗	✓	✓	–	[48]
Verify ARP caches	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗	–	–	[6, 48]
Jam or attack the attacker	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	✓	✗	–	–	[48]
Reinstall, rebuild, or renew	✗	✓	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✗	✓	✓	✓	✗	I	[103, 129, 149]
Provide shadow system	✗	✓	✓	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓	✓	✗	–	RC	[8, 21, 142, 165, 172]
Warn the attacker	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	✓	✗	–	RC	[21]
Attack the attacker	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	✓	✗	–	RC	[21, 23]
Encrypt traffic	✗	✗	✗	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗	–	RC	[79]
Migrate	✗	✗	✓	✓	✓	✓	✗	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	RC	
Limit resources	✗	✗	✓	✓	✓	✓	✓	✗	✗	✓	✗	✗	✗	✓	✓	✓	✓	RF	

[1] e.g. additional IDS, increase audit
[2] e.g. multi-factor authentication

5.2 System Design and Use Case

In Subsection 5.2.1 we give an overview of our response identification module. As intrusion response has to be triggered appropriately, we identify possible triggers that can be derived from our information model given in Section 3.3 (see Subsection 5.2.2). Afterwards, we show how our use case given in Subsection 4.1.3 can be treated with respect to applicable responses in Subsection 5.2.3.

5.2.1 Design Overview

For the design of the response identification module the following challenges occur:

- Determine the point in time automated intrusion response is triggered,
- Identify the highest `Alert Context` node within the hierarchy,
- Consider changes within the hierarchy of `Alert Context` nodes during an ongoing security incident,
- Compose `Alert Context` nodes concerning the same or overlapping targets within the target system in order to consider the security incident as a whole, and
- Identify the correct applicable responses (candidate responses).

The response identification module is triggered every time a new `Alert Context` node appears on the blackboard or the properties of an `Alert Context` node change. Each time the response identification module is triggered, a function following the definition of Algorithm 1 is called and the `Alert Context` node is handed over to. To react on every single insertion or update of an `Alert Context` node is not appropriate. Therefore, some trigger conditions have to be fulfilled (cf. Line 1.3). A closer look on possible trigger conditions and their combination possibilities is given in Subsection 5.2.2.

Algorithm 1 Response Identification – Main Function

```
1:  procedure incommingNotification(ac)
2:      openIssues → {}                                  ▷ currently open issues
3:      if trigger(ac) then                              ▷ trigger condition applies
4:          super → getSuperContext(ac)                  ▷ traverse up
5:          for i in super do
6:              if i not in openIssues then
7:                  if max(i_level) then                 ▷ superset node
8:                      openIssues.add(newIssue(i))
9:                  else
10:                     openIssues.add(None)
11:             else
12:                 openIssues.i.restartTimer()
13:             sub → getSubContext(i)                   ▷ traverse down
14:             for j in sub do
15:                 if i not in openIssues then
16:                     openIssues.i → None
```

The Alert Context node handed over to the response identification module, may be part of a larger Alert Context node hierarchy. This means multiple Alert Context nodes are related with each other, e.g. reflecting aggregated or correlated alerts. Consequently, the Alert Context node triggering automated intrusion response may be located at an arbitrary position within the alert context hierarchy. As a security incident has to be considered as a whole, solving the security incident partially is not suitable. Additionally, a relation between multiple Alert Context nodes indicates that they belong to the same security incident. Therefore, the Alert Context node with no outgoing edges (*superset node*) connected to the incoming Alert Context node has to be identified (cf. Line 1.4).

In Figure 5.1 this behavior is depicted. Nodes in the lowest level indicate victims targeted by the attack that is depicted as Alert Context node. All other nodes above the lowest level indicate Alert Context nodes that are connected with each other, e.g. due to aggregation. Within the first step, the marked Alert Context node triggers automated intrusion response. This Alert Context node has multiple other Alert Context nodes as super nodes. The response identification module traverses those Alert Context nodes until a node is found not having incoming edges. This *superset node* is the basis to create a new Issue within the response identification module. An Issue is a specialized Alert Context node combining Alert Context nodes that are handled together by automated intrusion response.

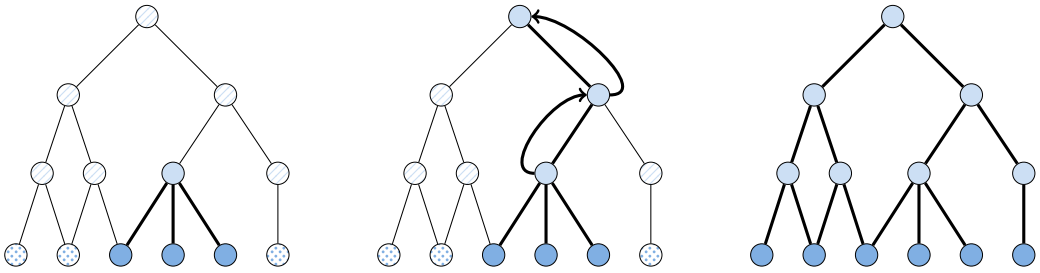


Fig. 5.1: Hierarchy of Alert Context Nodes During Response Identification

The function in Line 1.4 returns all Alert Context nodes on the way to the superset node. For each Alert Context node on the path, we check if we reached the superset node (cf. Line 1.7). If this is the case, we add a new Issue element into the list of open issues, meaning unsolved alert contexts. Otherwise, we add an empty element for the Alert Context node used as key. For each Alert Context node on the way to the superset node we additionally, traverse back down (cf. Line 1.13). This is the last step depicted in Figure 5.1. Each *subset node* of the superset node is added as empty entry for the Alert Context node as open issue. Adding an empty entry into the list of open issues has the advantage that the system knows, that the Alert Context node is in process and needs no further treatment (cf. Line 1.6).

Another challenge within the response identification module is to cope with the changes in the underlying information. Our information and execution model are designed to be additive. This means, that in case two distinct Alert Context nodes are combined into a single Alert Context node, the system can not remove relations between those node and the potentially applied issue. Therefore, the response identification module has to work with stable Alert Context nodes being superset nodes. Additionally added subset nodes can be integrated directly by inserting an edge between the Alert Context node representing the issue and the Alert Context node representing the subset node.

To rely on relatively stable superset nodes, we introduce a timer within the Issue

element. Only if the timer of the issue expires, the callback function (see Algorithm 2) is executed. The timer is configurable and is started if a new `Issue` element is initiated (cf. Line .1.8). In case an update with respect to the issue is noticed, the timer is reseted as the information base can not be considered as stable enough (cf. Line 1.12).

Within Algorithm 2 representing the callback function that is called for an issue after its timer exceeded, applicable responses are selected.

Algorithm 2 Response Identification – Callback Function

```
1:  procedure callback(ac)
2:      issue → getNonSolvedIssue(ac)                    ▷ overlapping issues
3:      if ≠ issue then
4:          issue = newAlertContextNode()
5:          bundle = newBundleNode()
6:      else
7:          bundle = getBundle(issue)
8:      connect(ac, issue)
9:      impls → []
10:     sub → getSubContext(ac)                          ▷ traverse down
11:     for i in sub do
12:         e → getEffected(i)                           ▷ get affected entities
13:         ie → getImplsEffected(e)        ▷ get implementations deployed on affected
    entities
14:         ia → getImplsAttack(i)                       ▷ get implementations for attack
15:         impls → impls ∪(ie ∩ ia)
16:     for i in impls do
17:         connect(i, bundle)
```

To further reduce the amount of issues within the system triggering automated intrusion response independently, we first verify that no issue exists with overlapping targets (cf. Line 2.2). Combining overlapping `Alert Context` nodes has the additional advantage that the selection of responses can leverage from synergy effects. A large set of victims that needs to be freed by automated intrusion response will may benefit from responses that cover multiple targets at once. Within Line 2.2 we query for an `Alert Context` node that is an issue that is not already solved (the `Solved` attribute is set to `FALSE`). If no such issue exist, we create a new `Issue` node and a `Bundle` node. Otherwise, we will use the identified existing issue and its corresponding bundle.

Afterwards, we identify all `Implementation` nodes that need to be connected to the `Bundle` node (cf. Line 2.17). First, we identify all subset nodes of the `Alert Context` node (cf. Line 2.10). Within Lines 2.11 to 2.15 appropriate implementations are identified.

For each subset node we identify the affected entities. This is done by following all edges describing a target relation ending up in an arbitrary infrastructure node. Starting from this affected entities, all deployed responses are examined. This is done by following the `deployedOn` edge between an infrastructure and an `Implementation` node. This step acquires all responses that can be executed on the target. Following the relations from the `Alert Context` node to the `Attack` node, further to the `Consequences` nodes and the `Response` nodes will lead to a second set of `Implementation` nodes that are connected to the response. Finally the set of applicable implementations is the union of all intersections of those described sets.

5.2.2 Triggering Responses

Intrusion response is not useful as every alert or alert context will trigger the execution of a response. Therefore, appropriate trigger mechanisms are needed. Within this section we provide possible triggers for intrusion response that can be derived from our information model given in Section 3.3. First, possible criteria are listed in Subsubsection 5.2.2.1. Possibilities to combine the criteria are examined in Subsubsection 5.2.2.2.

5.2.2.1 Decision Criteria

Using the information model given in Section 3.3 we can derive certain criteria that can be used to launch automated intrusion response. The decision at which point in time to trigger intrusion response has to be based on an existing `Alert Context` node (*ac*) and related nodes that can be reached from the alert context. Based on the alert context, the response identification module calculates a `Bundle` node with applicable response implementations. Therefore, the `Alert Context` node has to has properties warranting executing responses.

Structure of the Alert Context An `Alert Context` node describes a relation between different attacks detected within the target system. The more `Alert` nodes or `Alert Context` nodes are summarized within a single `Alert Context` node, the more relevant the alert context may be. Additionally, the higher the hierarchical level of an `Alert Context` node the more information is summarized. Consequently, a high portion of the security incident is covered if this alert context is mitigated. From the structure of an `Alert Context` node the following features are relevant for deciding if a security incident is mitigated:

Number of Alerts: If a predefined number of `Alert` nodes is obtained that are combined within an `Alert Context` node, automated intrusion response is triggered (count(*ac*.*alert*)).

Number of Alert Contexts: If a predefined number of `Alert Context` nodes is collected that are bundled within an `Alert Context` node, intrusion response is triggered. Hereby, all incoming edges of the `Alert Context` node within the complete hierarchy are of interest (count(*ac*.*ac*)).

Alert Frequency: If a predefined rate of alerts per time interval is reached, automated intrusion response is triggered (frequency(*ac*.*alert*)). This value can be determined using the number of alerts and the timestamp information of the alert. This metric is used in [149].

Hierarchical Level: If a predefined hierarchical level is reached, intrusion response is triggered. Hereby, the outgoing edges from an `Alert Context` node to another `Alert Context` node are traversed until an `Alert` node is reached (*ac*.*level*).

Alert Context Priority or Severity Each `Alert Context` node has a priority indicating the relevance of the security incident. This information can be used to trigger intrusion response (*ac*.*priority*) as recommended in [141]. Therefore, the `Priority` node or attribute can be utilized directly by checking whether or not a predefined value is reached. In an analogous way the severity of an `Alert Context` can be used (*ac*.*severity*). The severity of an `Alert Context` node describes the reliability that the detected attack is no false positive.

Targets Within the Target System Each `Alert Context` node is related to a target representing the victim of the security incident. Following the edge between the `Alert Context` node and the target different information can be derived, usable as trigger.

Number of Targets: If a predefined threshold of affected entities (targets) is reached, automated intrusion response is triggered. Hereby, the number of distinct edges between an `Alert Context` node including all related `Alert Context` nodes is counted (count(*ac.target*)).

Value of Target: Each network entity within the target system can be assessed to evaluate its importance to the target system. Therefore, to each information element representing infrastructure information, a `Value` attribute can be added. This attribute can be used for triggering, intrusion response if an `Alert Context` node is related to a network entity with a `Value` attribute higher than a predefined threshold(*ac.target.value*). This metric is used for example in [89].

Specific Target: In case an `Alert Context` node is related to specific predefined targets (either by name or type of the target), intrusion response can be triggered. For example, intrusion response is triggered directly if a router or a network nodal point is target of an attack (*ac.target.name* or *ac.target.type*).

Sources of Alert Context The source of an attack can be treated in an analogous manner as the target of an attack. Additionally, the source can be checked, whether or not, the source is inside the own network. As insider attackers may have greater impact than outsider attackers an insider attack has to be handled more quickly.

Attack Types of Alert Context Extracting the attack type of an `Alert Context` node can be implemented by following the edge between the `Alert Context` node and the `Attack` node (*ac.attack.name*). If a predefined attack is related to the `Alert Context` or any subset nodes, intrusion response is triggered.

Analyzer Information Each alert as basis of an `Alert Context` node is provided by a specific analyzer or Intrusion Detection System (IDS). Using information from the analyzer detected the security incident, the following criteria can be derived:

Number of Distinct Analyzers: A predefined number of different analyzers detecting the same security incident independently, can be used as trigger. Hereby, the following edges have to be traversed: from the `Alert Context` node to all `Alert` nodes and from the `Alert` nodes to their `Analyzer` nodes (count(*ac.analyzer*)).

Analyzer Severity: Each analyzer may have a severity indicating how reliable this analyzer can detect security incidents. Only if an analyzer with a severity that is high enough (predefined value) has confirmed a security incident, intrusion response is triggered (*ac.analyzer.severity*). This metric is used in [149].

Specific Analyzer: Additionally, specific analyzers can be specified (either by name or type of the analyzer). In case such an analyzer raises an alert the corresponding `Alert Context` nodes are used for intrusion response (*ac.analyzer.name* or *ac.analyzer.type*).

Timestamps-Based Decisions Each `Alert` node is equipped with a timestamp indicating the freshness of the information. The longer an `Alert` node resides in the system the less relevant is the carried information. Therefore, only `Alert Context` nodes that are up-to-date are handled with an automated response (*ac.timestamp*). The threshold the timestamp can lay in the past has to be predefined.

Potential Damage With each `Alert Context` node a damage value is associated using the relation between `Alert Context` and `Metric` (*ac.* < *metric* >.*value*). If the expected damage of a security incident is higher than a predefined value intrusion response is triggered.

5.2.2.2 Combination of Criteria

Providing the combination of multiple criteria, different use cases can be reflected more fine-grained and flexible. Each of the presented criteria can be evaluated to `TRUE` or `FALSE`. The result of an evaluated combination has to be either `TRUE` (start automated intrusion response) or `FALSE` (do not start automated intrusion response). Therefore, *Boolean Logic* can be used to combine multiple criteria.

To define multiple rules that can be applied to an `Alert Context` node, the `OR` operator is used. If multiple criteria have to hold at the same time, the `AND` operator is used. The negotiation is allowed as well. All rules for the bracket usage apply as in Boolean Logic.

For example, automated intrusion response is triggered in case the priority of an `Alert Context` node is higher than 90 or at least 20 different targets are victims of the security incident and the severity of the `Alert Context` node is higher than 80%. This rule is constructed as follows:

$$ac.priority > 90 \wedge (\text{count}(ac.target) \geq 20 \vee ac.severity > 0.8)$$

Each given property is calculable from the `Alert Context` node that may trigger automated intrusion response. If the given equation evaluates to `TRUE` applicable responses are going to be identified, otherwise not.

Another possibility for combining the listed triggers is to integrate existing approaches. In [149] a decision tree to determine the response strategy is proposed. This strategy encompasses which percentage of the intrusion is handled automatically, while the remaining part is handled manually. Therefore they utilize the efficiency of an IDS, the alert frequency, and the maximum risk of a security incident. The first two criteria are named as triggers in Subsubsection 5.2.2.1. The maximum risk needs to be calculated additionally.

Additionally, the work done in [12] can be integrated. They present an algorithm that decides whether or not, an alert is forwarded to the administrator or is handled automatically. Therefore, they utilize a cost-sensitive approach to balance the response costs and the costs of the security incident. The costs of the security incident can be extracted directly from our information model, as an `Alert Context` node is associated with a `Metric` node, carrying the cost per metric regarding the related security incident. The costs of the response can be extracted directly from our information model as well as they are associated with the available implementations a response provides.

5.2.3 Use Case Applicability

In Subsection 4.1.3 the use case of the data cabin network is introduced. This use case is based on the research department of Airbus Group Innovations (AGI) and describes a possible future data cabin network not yet deployed in todays aircrafts.

All devices within the data cabin network communicate with the central servers using the Scalable service-Oriented MiddlewarE over IP (SOME/IP) protocol. Servers as well as clients are running in a separate VM. For automated intrusion response it is assumed that those VMs can be controlled from the outside and inside. Additionally, responses within the network can be executed.

In Subsection 4.1.2 we identified possible attacks on the SOME/IP protocol. In Subsection 4.3.3 to 4.3.5 we showed the rules that can identify those attacks. Based on those attacks, we now identify their consequences in Subsubsection 5.2.3.1 to be compliant with our information model introduced in Section 3.3 and examine possible responses listed (see Subsection 5.1.3) in Subsubsection 5.2.3.2. The resulting dependency graph including the mapping between responses and consequences is shown in Subsubsection 5.2.3.3.

5.2.3.1 Attacks and Consequences

Based on possible attacks on the SOME/IP protocol we can identify the following implications and consequences:

Malformed Packet A malformed packet (**A1**) in the target system may indicate the existence of a hidden channel (Consequence **C1**). This hidden channel can be used to exchange information between multiple attacker instances or between the attacker and external networks. This may lead to an information leakage within the target system.

Additionally, a malformed packet may indicate a misconfigured device (Consequence **C2**) left behind from an unsuccessful attack attempt. Misconfigured devices can be handled as well as a security incident as they lead to unwanted behavior within the target system.

Changed Client ID and IP Mapping A changed mapping between a client ID and an IP **A2**) indicates that an attacker tries to impersonate a device. Consequently, the attacker is located in the internal network (Consequence **C3**). Attackers located within the target system have a broader influence within the target system than outsiders.

Further, malicious traffic appears in the target system (Consequence **C4**). Malicious traffic may disturb other devices or services within the target system.

Error on Error If an error message occurs as response to another error (**A3**), this may be due to a misconfiguration (Consequence **C2**) or indicates a hidden channel (Consequence **C1**). Additionally, an attacker may tries to disturb the network by sending malicious traffic (Consequence **C3**)

Error on Notification If an error message occurs as response to a notification (**A4**), this may be due to a misconfiguration (Consequence **C2**) or indicates a hidden channel (Consequence **C1**). Additionally, an attacker may tries to disturb the network by sending malicious traffic (Consequence **C3**)

Missing Response If a client does not get a response on a sent request (**A5**), this may indicates that the server is overloaded and is unable to respond (Consequence **C5**) to incoming requests. In case of a DOS attack the goal of the attacker is to overload the system such that legitimate requests are not handled anymore.

Missing Request A missing request of a server for a send response (**A6**) indicates that an attacker tries to impersonate a service. Consequently, the attacker is located in the internal network (Consequence **C3**). Furthermore, malicious traffic appears in the target system (Consequence **C4**).

Timing Issue If a client sends to many notifications to the server (**A7**), this may indicates a DOS to overload the servers (Consequence **C5**) Additionally, timing issues may indicate a misconfigured device (Consequence **C2**).

5.2.3.2 Selected Responses

Possible responses for the use case have to be *active* and need to target either the network, a service or a VM. In case the response is *sustainable*, it has to be *infeasible*. From the list of responses given in Subsection 5.1.3 we can derive the following responses according to the given limitations:

Suspend the VM or Service A VM containing a SOME/IP client or service can be paused (**R1.1**). As long as this response is activated, the SOME/IP client or service is not available in the target system. As we assume that each service and client is running in a separate machine, no other services or clients are affected directly. Additionally, a SOME/IP service can be stopped directly (**R1.2**), instead of stopping the whole VM the service is running on. Stopping a service or a VM a service is running, indirectly effects clients using this service.

Reboot the VM or Restart the Service This response is specified analog to Response **R1.1** and **R1.2**. For a reboot either the whole VM (**R2.1**) or the service (**R2.2**) can be used. The effect of this response according to other entities within the target system is limited to the time to reboot or restart the system.

Deny Clients or Service Privileges Denying privileges for clients (**R3.1**) restricts access from the single devices in the data cabin network to the central servers located in the backbone. Denying privileges for services (**R3.2**) will restrict the access of services on each other within the backbone network and lead to service limitations for clients.

Block Network, VM, or Service Blocking a whole network (**R4.1**) will lead to no communication flows into this network from the inside as well as to the outside. To restrict communication more flexible, a VM (**R4.2**) running a client or services can be blocked from communication or a single service (**R4.3**) provided within the target system.

Filter Traffic for Network, VM, or Service Instead of blocking all communication, filtering or shaping traffic from and to a whole network (**R5.1**), VM (**R5.2**) or service (**R5.3**) can be used. Those responses are less intrusive then Responses **R4.1** to **R4.3**, and still allow communication between entities even though performance is reduced.

Slow Down VMs As VMs are used, their execution can be slowed down (**R6**). Consequently, the execution of the client or service running within the VM is delayed and less traffic is generated. This response can be used, if the amount of traffic has to be reduced.

Unmount File System of a VM To unmount a file system can protect data leakages. But may lead to unpredictable crashes of services and clients. Therefore, this response will not be included within possible courses of action.

Use Redundant VM or Service A redundant service or VM can be used to compensate performance shortcomings in case of a high number of requests. Within the use case we assume that all available services are known before hand. Therefore, this response is not considered as a course of action.

Provide Shadow Systems for a VM or Service Shadow systems can be used to trap the attacker or to relieve the system from overload. Within the use case, we assume that all available services are known before hand. Therefore, this response is not considered as a course of action.

Encrypt Traffic of Service Activate encryption for the communication with services would prevent data leakage that is based on sniffing packets within the network. Within the use case no encryption is considered, yet. Therefore, this response is not considered as a course of action.

Migrate a VM The migration of a VM can be used to compensate performance shortcomings in case of a high number of requests in case of performance issues caused due to the underlying hypervisor. Within the use case, we assume that all VMs are running on fixed server (one server for the backbone functionality and a second one for all devices). Therefore, this response is not considered as a course of action.

Limit Resources of VM or Service Limiting the resources of a VM (**R7.1**) or service (**R7.2**) leads to a slower execution of the VM or service and less traffic is generated. This response can be used if the amount of traffic has to be reduced.

5.2.3.3 Complete Dependency Graph

The dependencies between attacks, consequences and possible responses are illustrated in Figure 5.2.

As hidden channels (Consequence **C1**) lead to information leakages, preventing the system from further losing information is stopping network packets from the target. This can be done by shutting down the device (Response **R1**) or blocking all traffic from the target (Response **R4**).

A misconfigured device (Consequence **C2**) can be mitigated by suspending the affected device (Response **R1**) or blocking it from the rest of the network (Response **R4**). In case the misconfiguration is not sustainable, a reboot of the device (Response **R2**) may reset its configuration.

An attacker established in the internal network (Consequence **C3**) has to be excluded from the internal network, by either suspending the device (Response **R1**) or blocking all

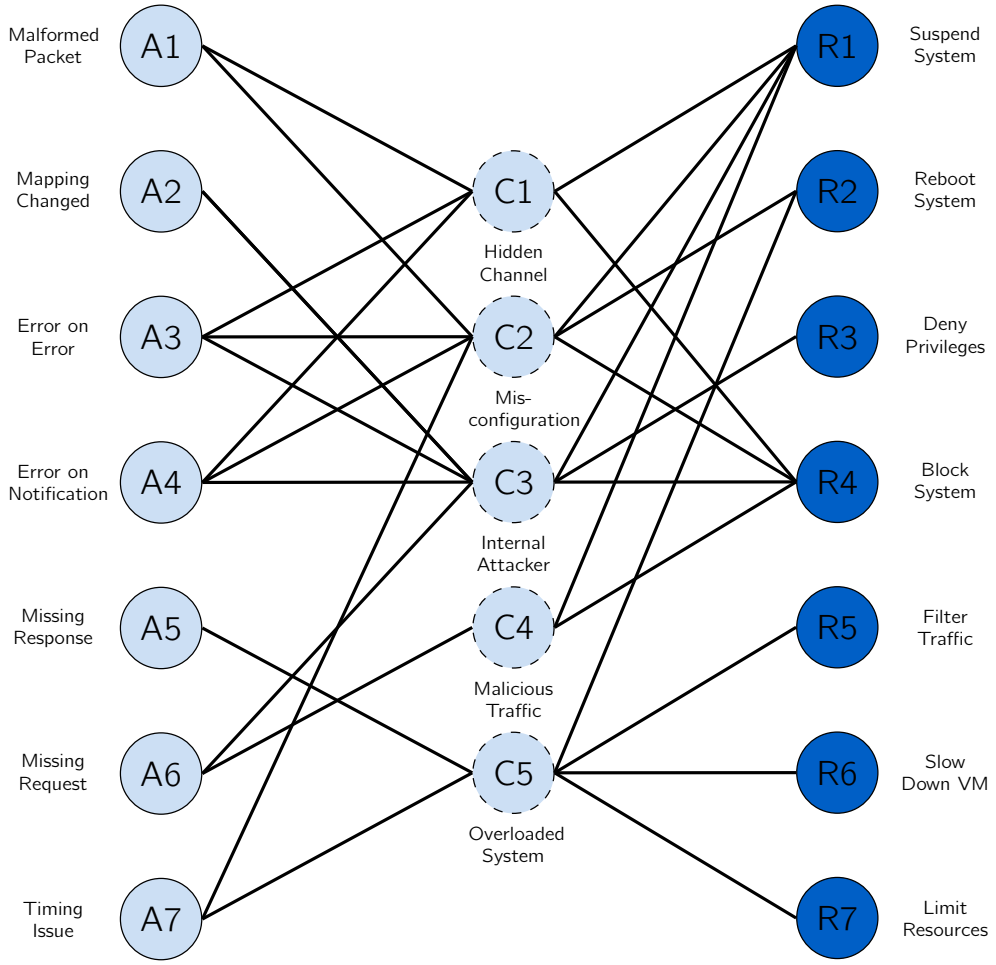


Fig. 5.2: Dependency Graph of Attacks, Consequences and Responses for the Data Cabin Network Use Case

traffic from the victim device (Response **R4**). Additionally, privileges of the device can be denied (Response **R3**) to limit the attackers access.

Malicious and disturbing traffic (Consequence **C4**) can be excluded from the network by either suspending the device (Response **R1**) or blocking all traffic from the compromised device (Response **R4**).

To handle an overloaded system (Consequence **C5**) the attacking system can be blocked by suspend the attacked system during the attack (Response **R1**) or filter traffic to the target system (Response **R5**). If the attacker is an internal attacker the attacker machine could be slowed down (Response **R6**) or its resources can be limited (Response **R6**) to reduce the attackers power.

5.3 Implementation

Embedding response identification into the IHS is done by providing a dedicated module. This module is available on GitHub[3] as part of the implementation of the overall IHS. This module is written in Python 3 and follows the algorithmic definition given in Section 5.2.1.

An `issue` is implemented using a dedicated class definition and is identified by the

[3] https://github.com/Egomania/BlackboardIDRS

ID of the `Alert Context` node triggering the initiation of the issue. Additionally, this class uses Pythons built-in timers, that are realized as threads, to implement the delay the callback function is called to get a more stable `Alert Context` node as superset node. Initially setting the timer and restarting the timer in case of unstable `Alert Context nodes` can be set independently.

To traverse up and down the `Alert Context` node hierarchy, the used queries have to be adjusted to the underlying database. In case a relational database is used, recursive queries are needed. A connection between two `Alert Context` nodes are stored within a table storing the superset and the subset node referencing the `Alert Context` table. In order to rebuild the hierarchy from a given `Alert Context` node as starting point, all edges between this start node being a subset node need to be followed. The resulting superset node is going to be the subset node in the next round. To traverse down superset and subset nodes have to be exchanged. In Listing 5.1 the query for PostgreSQL is shown.

Listings 5.1: Query to Traverse Up the Alert Context Hierarchy for PostgreSQL

```
1  WITH RECURSIVE contextTree (fromnode, level, tonode)
2  AS (
3          SELECT id, 0, id
4          FROM alertcontext
5          WHERE id = <start node>
6                  UNION ALL
7          SELECT cTree.tonode, cTree.level + 1, context.tonode
8          FROM contexttocontext context, contextTree cTree
9          WHERE context.fromnode = cTree.fromnode)
10 SELECT DISTINCT tonode
11 FROM contextTree
12 WHERE level = (SELECT MAX(level) FROM contextTree);
```

The return value of the query is only the most upper layer within the hierarchy as the return value is restricted (see Line 12). The recursive table `contextTree` is generated for each query with a different starting value (see Line 1 and 5) temporarily. For each round of execution the level is incremented (see Line 7). This level reflects the level of the hierarchy starting from the start node and reflects the recursion depth.

5.4 Evaluation

Within this evaluation, we show that the proposed system is aligned to the requirements stated in Subsection 5.1.1. A further evaluation of this module is done in the evaluation of the overall IHS in Section 8.2 as the alert processing modules are required.

The proposed response identification module provides flexible response triggers (cf. Requirement R1). In Subsection 5.2.2 we propose possible triggers that can be derived from our information model (cf. Section 3.3). Those triggers are based on the `Alert Context` node and directly or indirectly connected information elements. Thus, triggering automated intrusion response purely depends on the alert context itself. Additionally, single triggers can be combined to provide flexible adaptations to the underlying use case.

The proposed response identification module provides a continuous execution (cf. Requirement R2). As soon as a stable `Issue` can be generated, subset nodes can be added continuously. This includes adding additional, not yet established edges between additional `Implementation` nodes and the `Bundle` node connected to the issue. On the other hand side obsolete `Alert Context` nodes are neglected.

The proposed response identification module ensures that the number of concurrent response execution processes is low (cf. Requirement R3). Therefore, only stable `Alert Context` nodes are used as superset nodes to generate the `Issue`. Further, `Alert Context` nodes with the same or overlapping targets are composed into a single issue.

The proposed response identification module considers the target system (cf. Requirement R4). The identified responses only cover responses that are deployed on the set of affected entities within the target system (see Algorithm 2 in Line 2.12 and 2.13). Those responses are identified with the aid of those targets connected to the `Alert Context` node. Within the implementation a target can be a `Device`, `User` or `Service` node to be as specific as possible.

The proposed response identification module considers the security incident (cf. Requirement R5). The identified responses only cover responses that are effective against the attack of the alert context (see Algorithm 2 in Line 2.14). Those responses are gained by including the `Attack` node connected to the `Alert Context` node as well as the `Consequence` nodes related to the `Attack` node.

The proposed response identification module proposes applicable candidate responses (cf. Requirement R6). The identified responses only cover candidate responses (see Algorithm 2 in Line 2.15). For each affected entity the intersection of applicable (cf. Requirement R4) and effective (cf. Requirement R5) responses is identified. The candidate responses are combined as the union of those intersections.

6. RESPONSE SELECTION

Response selection describes the process of choosing an appropriate set of responses to be used in order to counteract the security incident. The selection is based on a set of candidate responses that are evaluated as effective against a certain attack. Each of those candidate responses has to be assessed against certain metrics in order to be able to select the most suitable subset of those candidate responses.

In a first step, possible strategies of response assessment strategies are discussed in Section 6.1. This includes providing requirements the response assessment has to fulfill, covering related work and propose an own metric to assess response costs. Within Section 6.2 we analyze requirements the response selection module has to fulfill and give a short introduction into Linear Programming (LP). Additionally, we provide an example to better illustrate the challenges of response selection. In Section 6.3 we provide a mathematical set-based model compliant to our information model given in Section 3.3 and the more detailed response model given in Subsection 5.1.2. This model is transformed into a Mixed Integer Linear Programming (MILP) problem that can be solved using optimization techniques. In Section 6.4 we examine our implementation supporting two different solvers for the optimization problem and two heuristics. The implementation is used in Section 6.5 to show the applicability of our approach by investigating the influencing factors with respect to the underlying target system. In Section 6.6 the related work is examined and compared against the requirements stated in Section 6.2. As this system is already published in [64], in the last section a publication reference is given to determine the own contributions to the approach (see Section 6.7).

6.1 Response Assessment Strategies

A central question when selecting appropriate responses is who to assess a single response. Responses are typically assessed by calculating their costs. For calculating response costs, the different properties for responses can be taken into account. Those properties are already examined in Subsection 2.4.2. First, the requirements for a response assessment strategy are examined in Subsection 6.1.1 Based on this requirements, in Subsection 6.1.2 a response assessment strategy is proposed. In the following Subsection 6.1.3, strategies to assess response costs are examined and the proposed approaches are compared against the identified requirements.

6.1.1 Requirements

Within this subsection we cover requirements the cost assessment has to fulfill for response selection. The requirements arise from recommendations stated in the related work or because a more precise estimation is needed. The assessment is restricted to the assessment of responses. Additionally, the assessment of the damage a security incident causes is needed to balance the response costs against the damage costs. The assessment of the damage of a security incident is not covered within this work.

Requirement R1 – Properties of Responses: The response assessment strategy should include the response properties given in Subsection 2.4.2. Those properties are examined from multiple related work ([20, 23, 67, 101, 140]) and are listed in Table 2.1. The given recommendations do often overlap such that they reflect common possibilities to assess responses. Those properties describe essential features a response has and are used as metrics to assess a response. As those properties are not directly comparable with each other, a response assessment strategy has to combine those features reasonable.

Requirement R2 – Include Dependencies: The response assessment strategy should reflect the underlying target system. Therefore, the dependencies between system entities should be reflected. Those dependencies may include services used in the target system relying on subservices provided across the network. Responses effecting those subservices may lead to higher costs as they influence other parts of the network indirectly. This requirement arise from the need to include the underlying target system to make the response assessment more precise and provide more appropriate estimations.

Requirement R3 – Include Redundancy: Additionally, redundancy aspects should be covered. Responses effecting network entities having redundant equivalents may lead to lower costs as the redundant component can be used during the execution of a response. The reason for this requirement is the same as for Requirement R2, the costs of a response can be assessed more precisely including information available on the target system.

Requirement R4 – Automated Elicitation: All influencing factors of the assessment should be determinable in an automated manner. First, this includes that the influencing factors are specified accurately. Additionally, they should not rely on expert knowledge estimating the influencing factors. The reason to include this requirement is that relying on expert knowledge depends solely on the experience of the expert itself. The authors in [57] point out that such assessments using experts knowledge highly depend on the estimation quality of the expert and name this as a limitation of their approach.

6.1.2 Proposed Response Assessment Strategy

Our cost metric to assess responses is derived from the properties given in Subsection 2.4.2. First, we show how our metric is assembled using those properties. Afterwards, we show a use case example on how to calculate the proposed metric within a service-based environment.

Derivation from Examined Properties One of the most common metric used is the downtime t_{g_i} of component g_i as a side-effect of executed responses [23, 67, 140]. In [140], this metric is split up into the downtime of critical components and at critical points in time. Another prominent metric is the success probability p_r of response r, or its counterpart, the error-proneness $(1 - p_r)$, expressing how reliable a response is to cope with the security incident [21, 67, 101]. The effectiveness e_r of a response expresses, how exhaustive the security incident is resolved, or how high the improvement of availability is after applying the response [23, 67]. Other metrics to measure a response were proposed including the complexity or severity of a response during execution [101].

The metric used for estimations depends on the system and may be fine-tuned towards special use cases. However, the following cost assessment is a good starting point:

$$m = \frac{1}{p_r} \cdot \frac{1}{e_r} \cdot \sum_{g_i} (t_{g_i} \cdot w_{g_i} \cdot c_{g_i}). \tag{6.1}$$

The first term $\frac{1}{p_r}$ is the inverse success probability of the response. We assume that the execution of responses are independent events and factor in the number of executions to get an expectancy value of 1. The second term $\frac{1}{e_r}$ represents the effectiveness of a response. We assume that coping with the remainder of the security incident will be proportional to the calculated costs. The last term includes additional costs: w_{g_i} is a *weighting* factor which translates the unit and scale of the *cost* c_{g_i} to embed it into the single cost function. t_{g_i} is the duration over which the cost occurs. This is well suited to resources such as bandwidth and memory.

Determining Costs in a Service-Based Scheme To model and calculate the costs per downtime for a response, we propose the following service-based scheme. *Users* or customers are paying the amount of U_c per time unit to use different *Services*. The value S_v of a specific service is the sum of all customer payments for this particular service. To provide those services, different *Subservices* are offered by *Hosts*, which may be redundant and deployed as Virtual Machine (VM) on a *VM server*.

Responses can influence those entities during execution. For example, blocking a single user from a service will affect neither the service as a whole nor the VM server the host is running on. Blocking connections on a firewall may affect a service as a whole, or just one host providing the service. Therefore, a special relation between a response and those entities is needed to reflect this influence. $i_{e,r} = influences : E \times R \rightarrow \{FALSE, TRUE\}$, where r is the response and e the entity influenced by this response.

We use relations between the entities *service* and *subservice* to model service dependencies. For example, a service may depend on a database and a mail service as subservices to fulfill its tasks. To model those dependencies we use first-order logic and express those relations as follows: $S = (x)A \wedge (y)B$, where S represents the service and A and B the subservices used. x and y are the degree of dependency. For example $S = (0)A \wedge (0)B$ means that service S fully depends on the functionality of both services. $S = (0.5)A \wedge (0.7)B$ means that S can provide 50 % of it's functionality in case A is not available and 70 % in case B is not available.

To model redundancy we use the entities *subservice* and *host* in an analogous way. The following formula $S = (x)A \vee (y)B$ expresses that a subservice S runs on host A and B with different degrees of efficiency. For example $S = (1)A \vee (0.9)B$ means that a shutdown of host A will have no effect, but a shutdown of host B will lead to a service degradation to 90 %. To calculate the response costs r_c for a response the following applies: The response influences a

- *User*: $r_c^u = U_c$
- *Service*: $r_c^s = S_v$
- *Subservice*: $r_c^{sub} = \sum_{A_i} x_i \cdot S_v^j$ where S^j depends on A_i and $i(r, A_i) = TRUE$
- *Host*: $r_c^h = \sum_{A_i} (1 - x_i) r_c^{sub}$ where A_i depends on h and $i(r, A_i) = TRUE$
- *Server*: $r_c^{ser} = \sum_h r_c^h$ where h depends on ser and $i(r, ser) = TRUE$

At this point, we like to point out that those calculations are independent from the response selection process and other metrics can be used as well.

6.1.3 Related Work

The following related work presents possibilities to assess the costs of a response as well as to assess the potential damage of a security incident. First, we shortly explain how to integrate the related work into the overall Incident Handling System (IHS) proposed in this work. Afterwards, we examine the related work by following categories:

- Basic approaches
- Including system information, like the importance of the target of an attack
- Including probabilities, like success probabilities of an attack
- Including Intrusion Detection System (IDS) capabilities

Lastly, we give a comparison and a short summary of the related work and conclude the findings.

Integration into the Proposed IHS Instead of the proposed response assessment strategy given in Subsection 6.1.2, the following strategies can be integrated into our proposed IHS. Each presented response assessment strategy can be used as dedicated metric. The evaluated response costs are stored on the edge between the `Response` node and the `Metric` node. The evaluated costs of the system damage can be stored on the edge between the `Alert Context` node and the `Metric` node.

Basic Approaches In [146] a response cost model is proposed. The response costs are calculated as follows:

$$RC = OC + RSI - RG$$

With OC describing the *Operational Costs*, RG describing the *Response Goodness* (who god a response mitigates a security incident) and RSI describing the *Response Impact* on the system. The operational costs have to be determined manually using expert knowledge. The response goodness is calculated by combining available responses on the system resources with their weights expressing the importance of a system. The response goodness is evaluated by selecting applicable responses and compute the goodness of these responses in terms of the attack.

In [48] a cost model for responses regarding *Advanced Metering Infrastructure (AMI)* is given. They propose to calculate the costs of a response as follows:

$$C_{Action} = C_{Impact} + C_{Operation} + C_{Attack}$$

$C_{Operation}$ refers to the operational or managerial costs like costs for recovery, labor, initiating the response and needed resources. They propose to include the duration of a response as well, as they assume that the longer a response will last, the higher are the operational costs. C_{Impact} describes positive as well as negative effects on the security goals Confidentiality, Integrity and Availability (CIA). C_{Attack} describes the costs of the attack's consequences. Additionally, they present how to convert the impact on the security goals to financial values. They propose to use market pricing, service level agreements and empirical data for pricing CIA degradations. Their presentation lacks of

concrete implementations, how an integrity degradation is determined, how the degree can be identified, and how to map this to certain costs. Moreover, it is unclear why the response costs include the attack impact.

In [83] the question whether a response is worthwhile to be executed is discussed. Therefore, they define the *RORI index (Return-On-Response-Investment)* as follows:

$$RORI = \frac{RG - (CD + OC)}{CD + OC} = \frac{[IC_b - RC] - OC}{CD + OC}$$

Where RG describes the *Response Goodness*, CD the response' *Collateral Damage* and OC the response' *Operational Costs*. The response goodness RG can be composed of $IC_b - IC_a$, whereas IC_b are the costs of the security incident in case no response is executed and IC_a are the remaining costs of a security incident after a response is executed. The authors admit, that IC_a is hard to determine and therefore propose RC that represents the combined impact of security incidents and response. OC has to be determined statically, while RC, IC_b and CD are determined on-line. Additionally, they propose a method to quantify costs for security incidents and responses by using privilege infection and revocation. Propagations are mapped by service dependencies. In [84] they had already shown how to propagate CIA within a dependency graph.

In [57] the authors claim to improve the known *Return on Response Investment (RORI)*. The calculation includes the *Annual Loss Expectancy* (ALE), the *Risk Mitigation Level* (RM), the *Annual Response Costs* (ACR), as well as the *Annual Infrastructure Value* (AIV). They authors identified that most parameters need to be estimated appropriate by an expert as a limitation of this solution.

In [67, 68] a graph-based approach for determining costs is introduced. To calculate the response costs the following properties are included: the expected success, the expected effort, the expected error-proneness, and the expected durability of a response. They include dependencies between resources and calculate negative impacts of responses with respect to availability of connected responses. They are able to model different classes of dependencies using their dependency graph.

In [129] *Orcef* is introduced and a cost-sensitive approach to balance the response costs and the cost for a security incident is proposed. They consider the positive as well as negative effects of a response. To determine the negative effects of a response, a service dependency graph is utilized. They ascertain the positive impact on the CIA properties and on the speed of the system. Negative impacts consist of impacts on hosts, external communication, network, network user, local user and the setup costs. Most of the required metrics are statically computed parameters. Additionally, they need the value of the affected resources. This value is determined by an expert's opinion. The damage cost of a security incident is statically assigned for their 4 categories, namely U2R, R2L DOS and PROBE ([95]).

Approaches Including System Importance In [101] a bunch of response metrics for the *AIRS* is proposed that are used with respect to the importance of a compromised system, that can be low, medium or high. They define the *Damage Reduction Metric* to balance security incident and attack costs similarly as defined in [141]:

$$I_i \cdot C_{IDS} \geq I_r$$

Whereas, I_i denotes the impact of the security incident, C_{IDS} denotes the confidence level of an IDS, and I_r denotes the impact of a response. Each single response within

the set of candidate responses has to fulfill this requirement. Additionally, they propose a *Minimum Cost Metric* that does not depend on the success of a response with the objective to minimize the following equation:

$$Cost_T = I_R + Cost_d$$

Whereas, $Cost_T$ denotes the *Response Costs*, I_R denotes the impact of a response during execution, and $Cost_d$ denotes the deployment costs of a response. For more valuable attack targets the propose the *Highest Severity and Highest Efficiency Metric* that maximizes the response severity and success:

$$RE \cdot S_{abs_r} \geq S_i$$
$$Max\{RE \cdot S_{abs_r}\}$$

Whereat, RE is the *Response Efficiency* calculated from previous executions, S_i denotes the severity of the IDS detected the intrusion, and S_{abs_r} denotes the absolute severity of a response previously set by the administrator.

In [110] *REASSESS (Response Effectiveness Assessment)* is proposed. For each response its effectiveness is calculated taking negative and positive effects of a response into account. The negative effects lead from possible service degradation, penalty costs can arise from Service Level Agreement (SLA) violations and alert priorities are taken into account. The effectiveness of a response is the difference between its positive and negative effects. Negative effects are calculated by taking the service reduction and the importance of a service into account. Positive effects depend on the response success rate.

In [93, 167] a cost-sensitive method for distributed intrusion response is presented. It includes the *Damage Costs*, describing the costs of the security incident, and is quantified by the criticality (importance of the target) and the lethality (degree of potential damage). The *Response Costs* depend on the response mechanism and the *Penalty Costs* measure the costs of a false-positive. The *Operational Costs* are the costs arising due to the use of IDSes. To determine the response costs fixed values are used that are estimated by the authors.

Approaches Using Probabilities In [78, 79] the response assessment is based on the success likelihood of an ongoing attack. The response reducing the success likelihood at most, is considered as the best response to choose. For this mechanism, an attack graph is needed. A node in this attack graph represents an observed or expected attack. For each detected attack, a new instance of the attack graph is instantiated. The root of the attack represents the attackers objective. The closer the current ongoing attack is at the root node, the more urgent it is to find a response and the higher is the success likelihood of an attack. The attack's success likelihood SL is calculated as follows:

$$SL_x = -20 \cdot log_{10} \cdot \frac{MTAO_X - 1}{MTAO_X}$$

Whereas, $MTAO$ is the *Mean Time to Attack Objective* of attack objective X. The closer the attacker comes it's goal in the attack graph the higher is the success probability.

In [141] a cost sensitive model for response selection is proposed. They use utility theory and compute the expected value (EV) of a response r_s to a sequence S as follows:

$$EV(r_s) = (Pr_{succ}(S) \cdot SF) + (Pr_{risk}(S) \cdot (-RF))$$

Whereas, $Pr_{succ}(S)$ describes the probability that a sequence S will occur, $Pr_{risk}(S) = 1 - Pr_{succ}(S)$, SF denotes the success factor, and RF denotes the risk factor.

Approaches Including IDS Capabilities In [69] a cost model including false-positives is considered. They assume that both IDSes and Intrusion Response Systems (IRS) produce false-positives. The two states on the IDS are: no intrusion or a real intrusion. The three states on the IRS are: no response, false response and true response. They propose an *Intrusion-Response-Matrix* presenting all responses and security incidents available. An entry in this matrix represents the costs of a response to a certain security incident. They add an additional row and column to provide information about costs of false-negative responses and false-positive responses. They need an additional *Response-Cost-Matrix* as starting point. Within this matrix experts have to manually assign costs to each response with respect to the security incident. Afterwards, they recalculate those costs with respect to false positives. Within this work no real cost assessment was done, but only a refinement of already manually specified costs.

In [177] two cost-based security metrics characterizing the response and current situation are introduced: the *Maintenance Costs* of a response ($Cost_m$) and the *Costs of Failures* due to an attack ($Cost_f$). The first metric ($Cost_m$) is composed of the significance of a component or service in the network (D_s) and the costs when the response is executed (D_d). The second metric ($Cost_f$) is also based on the significance estimation (D_s), but instead of D_d the threat degree of the attack (D_a) is used. In order to choose a response the response maintenance costs and costs of an attack should be balanced. The authors include the uncertainty of IDSes due to false-positives by introducing *Penalty Costs* ($Cost_p$) for executing responses in case they were not needed. A *Partially Observable Markov Decision Process (POMDP)* is used to put all things together and maximizing the reward for changing the system state. The authors do neither describe how to impose the described metrics nor discussing possible responses that can be implemented and how to assess them.

In [172] the costs for active and passive responses are calculated in different ways. The costs of active responses are calculated by summing up the costs arising due to delaying benign events, the average damage costs of undetected attacks and investigation capacity per time. The costs of passive responses are calculated by summing up the average damage costs while an alert is investigated, the average damage costs of undetected attacks and investigation capacity per time.

Summary, Comparison and Conclusion The requirements stated in Subsection 6.1.1 are mapped to the presented approaches. The results are summarized in the following Table 6.1.

Tab. 6.1: Comparison of Related Work Based on the Requirements Stated in Subsection 6.1.1

Approach	R1 – Response Properties	R2 – Dependencies	R3 – Redundancy	R4 – Automated Elicitation
Response Costs [146]	✗	✗	✗	✗
AMI [48]	✗	✗	✗	✗
RORI [57, 83]	✗	✗	✗	✗
Graph-based [67, 68]	✓	✓	✗	✓
Orcef [129]	✗	✓	✗	✗
AIRS [101]	✗	✗	✗	✗
REASSESS [110]	✗	✓	✗	✗
Distributed [93, 167]	✗	✗	✗	✗
Success Likelihood [78, 79]	✗	✗	✗	✗
Utility theory [141]	✗	✗	✗	✗
Response-Cost-Matrix [69]	✗	✗	✗	✗
POMDP [177]	✗	✗	✗	✗
Active and Passive [172]	✗	✗	✗	✗

The related work can be structured into basic approaches using mostly predefined values that need to be evaluated by experts. Those systems mainly provide a mere definition of response costs based on the costs of a security incident and the positive effects of executing a response. Some approaches take the importance of the targeted component into account, that leads to a more precise estimation, but most often, service dependencies are not covered. Additionally, approaches including success probabilities of future events exist. They try to foresee the next steps within a sequence of attacks or rely on additional knowledge like vulnerability assessment. The last category of related work are response assessment strategies including IDS capabilities. They include additional knowledge into the response assessment in order to cope with unreliable IDSes.

Concluding the related work presented in this section against the requirements stated in Subsection 6.1.1 shows that none solution can fulfill those requirements. The main problems are the automated elicitation of needed parameters and the reflection of the properties for responses given in Subsection 2.4.2. The most promising approach [67, 68] is limited to availability.

6.2 Analysis

Within this section, we first examine the requirements the response selection strategy has to fulfill (see Subection 6.2.1). As our response selection approach is based on linear optimization, we give a short introduction into this topic in Subsection 6.2.2. Additionally, we give an illustrative example for response selection and the main challenges in Subsection 6.2.3.

6.2.1 Requirements

The requirements for the response selection module arise from the related work or functionalities found within related approaches. For response selection the following requirements have to be fulfilled:

Requirement R1 – Cost-Sensitive Approach: Response selection has to take the impact of a response as well as the impact of the security incident into account. Responses that are chosen have to balance those costs. Otherwise, responses that are more expensive than the damage caused by the security incident are chosen. In this case, taking no action is more suitable than choosing an expensive response. The recommendation for a cost-sensitive approach towards response selection is given in [142].

Requirement R2 – Comprehensive View: The selection mechanism has to be able to take the whole security incident into account. A security incident may consists of multiple attack steps, or targets. The response selection has to be designed such that not only a single security incident is covered but multiple related security incidents. The selection mechanism has to be able to select multiple responses that cover the security incident when they are executed in combination as no single response may cover multiple attack steps.

Requirement R3 – Conflict-Free Selection: In case multiple responses are executed in combination, those responses must not conflict with each other. Conflicting responses may override each other and the responses do not have the expected impact on the target system. In [148] existing response plans are post-processed in order to eliminate conflicts between responses.

Requirement R4 – Preparation Actions: Some responses may require the execution of some actions to prepare the target system. Multiple responses may use the same actions for preparation. The response selection mechanism should support to model pre- and postconditions and take those relations into account during response selection. In case a response needs an expensive preparation, this response may still an option in case other responses need this preparation as well.

Requirement R5 – Synergy Effect: The response selection should model responses that are helpful against a certain attack for more than one victim. For example, a response targeting a whole subnetwork, may mitigates a security incident for multiple victims. Whereas, a self-healing response is only helping a single entity. Consequently, a more expensive response helping multiple victims may be more reasonable than multiple single self-healing responses helping only a single victim. This requirement is needed in order to satisfy new network management capabilities, like software-defined networking (SDN), allowing more options to interact with a network as a whole.

Requirement R6 – Flexible Cost Assessment: The response selection should allow to exchange the cost assessment for responses as well as the security incident. Response assessment should, therefore, be decoupled from the selection process and should not be tightly interleaved. A flexible cost assessment is needed in order to provide an online response cost evaluation model as recommended in [130].

6.2.2 Linear Programming

LP is a well-known and well-researched technique originating from the field of operations research. While an LP problem can be solved in polynomial time, the restriction of (some) variables to integral values makes the problem one of Karp's 21 NP-complete problems [81]. Well-known methods to solve LP problems are *Basic-Exchange* algorithms, for example the *Simplex*-algorithm proposed by Dantzig [31], or *Interior Point* methods [81].

A beneficial property of these algorithms is the possibility of a *warm start-over*, where an optimal solution from a previous solution run can be re-used. This property has a positive impact on the solution performance when a problem has to be solved repeatedly with modified matrix coefficients.

To solve an *Integer Linear Programming (ILP)*, first, the corresponding linear problem, called the relaxation of the ILP, is solved. Based on the optimal solution of the relaxation, exact methods like *Branch and Bound* algorithms or *Cutting-Plane* methods can be applied to find the integer optimal solution.

In linear optimization, we try to optimize a given objective function subject to certain constraints. Both, the objective function and the constraints, are formulated as linear equations. The goal of LP is to maximize or minimize the value of the objective function within the constraints of the given constraint equations. Geometrically, the linear constraints define the feasible region, which is a convex polyhedron. A linear function is a convex function, which implies that every local minimum is a global minimum; similarly, a linear function is a concave function, which implies that every local maximum is a global maximum.

An optimal solution need not exist, for two reasons. First, if two constraints are inconsistent, then no feasible solution exists: For instance, the constraints $x \geq 2$ and $x \leq 1$ cannot be satisfied jointly; in this case, we say that the LP is infeasible. Second, when the polytype is unbounded in the direction of the gradient of the objective function (where the gradient of the objective function is the vector of the coefficients of the objective function), then no optimal value is attained.

The standard form of an LP problem consists of three parts:

- A linear function to be maximized or minimized, called the *objective function*:

$$f(x_1, x_2, x_2) = c_1 x_1 + c_2 x_2 + c_3 x_3$$

- A set of *problem constraints* in the form:

$$\begin{aligned} a_{11}, x_1 + a_{12} x_2 + a_{13} x_3 &\leq b_1 \\ a_{21}, x_1 + a_{22} x_2 + a_{23} x_3 &\leq b_2 \\ a_{31}, x_1 + a_{32} x_2 + a_{33} x_3 &\leq b_3 \end{aligned}$$

- A set non-negative variables

$$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0$$

An LP problem is often described in the matrix form and expressed as:

$$\max\{c^T x | Ax \leq b \wedge x \geq 0\}$$

If the set of non-negative variables is linear, the optimization problem is called an LP problem. If a subset of these variable are required to be integer (which is including binary values) the problem is called a *MILP problem* and if all variables are required to be integer it is called an *ILP problem*.

A property of linear optimization is that feasibility and optimality is mathematically ensured and provable: for every system of linear equalities we can, by looking at the rank of the extended matrix problem Ab, state if the system of equalities is feasible or infeasible and if the system has no solution, a unique solution or infinitely many solutions. If a system has multiple solutions, it is possible to iterate all optimal and all feasible solutions as described in [82].

6.2.3 Illustrative Example

In Figure 6.1 an illustrative network topology is shown. The network entities are a firewall fw, a router r, two switches s_1 and s_2, and 4 hosts h_1 to h_4. In this network a Denial of Service (DOS) attack was detected. This attack affected a service running on host h_1 and a service running on host h_2.

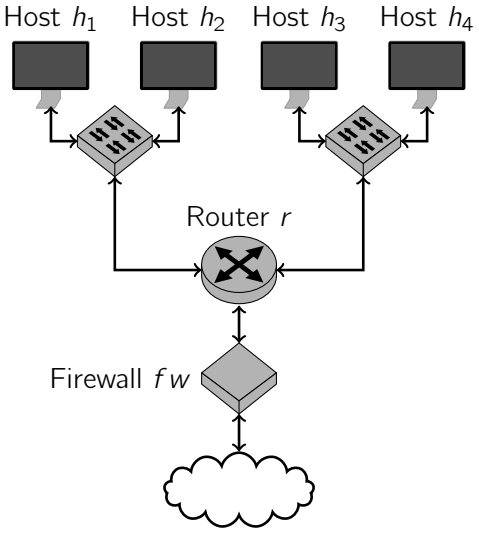


Fig. 6.1: Network Topology Example

The firewall has the following options to automatically respond to this attack: it can block traffic going from or to all hosts of the system ($r_{block}(host)$). Additionally, the firewall can be configured, such that the whole subnetwork, h_1 and h_2 reside in, is blocked ($r_{block}(network)$). Moreover, host h_3 provides a cold-standby for the two affected services, and can provide one of the services running on host h_1 and host h_3, but not both ($r_{migrate}(service)$). If the service is running on another host, the router has to be reconfigured in advance to provide the new routing to this service ($r_{reroute}$). Host h_2 can reconfigure its own service in order to work correctly even it is under attack by providing more resources ($r_{reconfigure}$).

The goal is to find an appropriate selection of responses, such that the DOS attack is mitigated for both services under attack. Hereby, the self-healing response of host h_2 ($r_{reconfigure}$) is conflicting with the blocking of the firewall ($r_{block}(h_2)$) as the reconfiguration would be nullified. The adaptations of the routes ($r_{reroute}$) are a precondition for migrating the service ($r_{migrate}(service)$).

The description of the network, consisting of attacked entities as well as entities capable of executing a response, and the information about suitable responses as well as their relations have to be transferred from this informal description to a mathematical model in order to provide an LP problem. After, introducing our system model the given example will be mapped to this proposed model (see Subsection 6.3.2).

6.3 System Model

In this section, we define a model of the relationships between network entities (infrastructure information) and information elements needed for automated intrusion response (see Subsection 6.3.1). The idea is that a real-world IRS will generate an instance of this model from the parameters of a current security incident and available infrastructure

information. Afterwards, we show in Subsection 6.3.2 how to apply this set-based model to our example given in Subsection 6.2.3. The automatic transformation of the proposed model into a *MILP* problem represented as linear equations of an optimization problem, which can be solved automatically, is examined in Subsection 6.3.3. The goal is to find the cheapest combination of responses which mitigates the security incident on all affected hosts. This problem can be solved using *MILP*. As a result, the IRS obtains an optimal subset of responses to execute.

6.3.1 Definition of Elements and Relations

The focus is to determine suitable response strategies in case of a detected security incident in the target system. These strategies can involve multiple options executable by different and distributed entities in the target system. Therefore, the goal is to find the optimal subset of responses from a given set of candidate responses to be executed by these network entities in order to counteract the security incident. Within this work, an *optimal* response strategy involves only necessary responses to counteract the security incident. In addition, the selected solution generates minimum costs with respect to a given set of cost metrics. In the following the set-based model is introduced. The used sets and symbols including a short description are summarized in Table 6.2.

Tab. 6.2: Sets and Symbols Used in the System Model

Symbol	Set	Meaning
s_i	S	Set system entities
a_i	$A \subseteq S$	Set of entities affected by the security incident
e_i	$E \subseteq S$	Set of entities executing responses
r_i	R	Set of responses
m_i	M	Set of metrics
$TRUE, FALSE$	B	Set of boolean values
c	$R \times M \to \mathbb{R}$	Cost of a response in metric
d	$M \to \mathbb{R}$	Cost of inaction ('damage')
x	$E \times R \to \mathbb{B}$	Is response executed by entity?
f	$A \times R \to \mathbb{B}$	Does response 'free' entity?
o	$R \times R \to \mathbb{B}$	Do responses conflict?
p	$R \times R \to \mathbb{B}$	Do responses have preconditions?

A network consists of various entities, such as hosts, routers, and firewalls. In this model, all of these entities are contained in the set $S = \{s_1, s_2, \ldots\}$. All network entities can be identified uniquely.

A subset of the entities is affected by the security incident: $A = \{a_1, a_2, \ldots\} \subseteq S$. All network entities ($S$) capable of executing a response are contained in the separate set $E = \{e_1, e_2, \ldots\} \subseteq S$.

The set $R = \{r_1, r_2, \ldots\}$ contains all responses which are available to the IRS. All responses can be identified uniquely. A response, if successfully executed, will mitigate the effects of the security incident to one or several entities of A. This is captured by the function $f : A \times R \to \mathbb{B}$. $f(a, r)$ evaluates to 1 for *true* if r "frees" entity a, and otherwise to 0 for *false*. Using this formulation Requirements R2 and R5 can be fulfilled.

The function $x\colon E \times R \to \mathbb{B}$ describes by which entity a response can be *executed*. A response can only be executed by exactly one entity. Thus, the following property holds:

$$\forall r \in R\colon \sum_{i=1}^{|E|} x(e_i, r) = 1$$

If another entity can execute an equivalent response, we regard both responses as two distinguishable responses. For example, rerouting can be done by every router entity within the target system. This will be reflected in multiple responses dependent on the number of routers within the network. From function x (executing) the set E describing all entities that can execute responses can be calculated:

$$\forall e \in E\colon \exists x(e, r_i) = 1$$

A response may itself incur costs when executed. Costs are measured by a set of metrics: $M = \{m_1, m_2, \ldots\}$. The metrics may be chosen freely by the system administrator. An example for such a metric is the execution time of a response. The rationale is that a service may be down while the response is still running. The *cost* of a response is given by the function $c : R \times M \to \mathbb{R}$. There is also a cost associated with not mitigating the security incident at all. This is the *damage* $d\colon M \to \mathbb{R}$, measured using the same metrics as responses are assessed. The damage limits the effort put into the security incident mitigation: if the total cost of the selected responses exceeds the damage, it is cheaper to do nothing. This part of the design fulfills Requirements R1 and R6.

Metrics of different domains are, in general, not comparable. To be able to choose the "best" responses, we convert and weight all metrics to get a single cost domain with the range $[0, \infty)$.

A response may conflict with another response (cf. Requirement R3). This is the case, if an entity can only execute one but not both responses, or if the effect of one response negates the effect of another response. The function $o\colon R \times R \to \mathbb{B}$ describes which responses conflict. o is naturally symmetric and non-reflexive, so

$$\forall r_1, r_2 \in R\colon o(r_1, r_2) = o(r_2, r_1) \wedge \forall r \in R\colon o(r, r) = 0$$

hold. The function is evaluated to $true$ if the given responses conflict, otherwise to $false$.

A response may need another response to be executed in advance (cf. Requirement R4). Defining those preconditions allows to provide more flexible and fine-grained response plans to be defined. If two responses require the same preconditions, synergy effects can be used and the costs of a response plan can be determined more precisely. The function $p\colon R \times R \to \mathbb{B}$ describes which responses have a precondition relation. The precondition relation is not symmetric as a response is not a precondition of itself and non-reflexive as no cycles must occur:

$$\forall r_1, r_2 \in R\colon p(r_1, r_2) \neq p(r_2, r_1) \wedge \forall r \in R\colon p(r, r) = 0$$

The function is evaluated to $true$ if the given response r_1 is precondition of response r_2, otherwise to $false$. A response being a precondition does not necessarily frees entities in case of a security incident. This means that a precondition r may not appear within the function f such that $\exists a \in A(a, r) = 1$.

6.3.2 Illustrative Example

In this subsection we map the example given in Subsection 6.2.3 to the set-based model given in Subsection 6.3.1.

The set of network entities $S = \{fw, r, h_1, h_2, h_3, h_4, s_1, s_2\}$ describes all elements of the network. Attacked network elements are summarized in the set $A = \{h_1, h_2\}$. Network entities capable of executing a response are described in set $E = \{fw, r, h_2, h_3\}$. The set of all possible responses that can be executed in this network to cope with the detected attack is composed as follows: $R = \{r_{block}(h_1),\ r_{block}(h_2) r_{block}(network),\ r_{migrate}(h_1), r_{migrate}(h_2), r_{reroute}(h_1), r_{reroute}(h_2), r_{reconfigure}\}$.

The next step is to transform the relations between the elements of the sets A, E and R given in the informal description into the formal function $x(e, r)$, $f(a, r)$ $o(r, r)$, and $p(r, r)$. Entries, that are not given in these relations explicitly, are evaluated to 0 ($false$), given tuples are evaluated to 1 (true). For function o only one direction is given, symmetric dependencies are not listed. The relations are listed in the following:

- $x = \{(fw,\ r_{block}(h_1)),\ (fw,\ r_{block}(h_2)),\ (fw,\ r_{block}(network)),\ (h_3,\ r_{migrate}(h_1)),\ (h_3,\ r_{migrate}(h_2)),\ (r,\ r_{reroute}(h_1)),\ (r,\ r_{reroute}(h_2)),\ (h_2,\ r_{reconfigure})\}$
- $f = \{(h_1,\ r_{block}(h_1)),\ (h_2,\ r_{block}(h_2)),\ (h_1,\ r_{block}(network)),\ (h_2,\ r_{block}(network)),\ (h_1, r_{migrate}(h_1))(h_2, r_{migrate}(h_2))(h_2, r_{reconfigure})\}$
- $o = \{(r_{migrate}(h_1), r_{migrate}(h_2))\}$
- $p = \{(r_{reroute}(h_1), r_{migrate}(h_1)), (r_{reroute}(h_2), r_{migrate}(h_2))\}$

As described previously, host h_3 can only be used for one service to be migrated. In consequence, both migration responses are conflicting. As mentioned before, migration of a service needs reconfigured routes as preconditions.

The last part of the model is to describe the metrics used to determine the best combination of responses that fulfills the following properties:

- All entities in A are freed from the security incident.
- No conflicting responses are executed.
- Preconditions are included.
- The effort in form of different metrics has to be minimal.

In this example, the duration of the execution and the monetary costs that will arise if a response is executed, are chosen as metrics. All given responses including metrics, affected and executing entities are summarized in Table 6.3.

From Table 6.3 it can be seen that more combinations will fulfill the first two properties. First, the firewall can be reconfigured in a single step ($r_{block}(network)$) or in two separate steps ($r_{block}(h_1)$ and $r_{block}(h_2)$). Second, a service could be migrated and the other one can be secured be reconfiguring the firewall. The last option is the self healing process on host h_2 in combination with a migration or firewall configuration. All responses differ with regard to their metrics, in this case duration and costs. The goal is to chose the optimal solution with respect to given metrics.

Another observation is that executing two different non-conflicting responses freeing the same target, will have no additional benefit. For example, if the service is migrated from host h_1 ($r_{migrate}(h_1)$) and the firewall is reconfigured for both victims

Tab. 6.3: Possible Responses with Executing Entities, Effected Entities and Metrics

Response	Executing	Effected	Duration in ms	Monetary Costs
$r_{block}(h_1)$	fw	h_1	1.2	20
$r_{block}(h_2)$	fw	h_2	1.2	20
$r_{block}(network)$	fw	h_2, h_1	1.5	30
$r_{migrate}(h_1)$	h_3	h_1	30	100
$r_{migrate}(h_2)$	h_3	h_2	30	100
$r_{reconfigure}$	h_2	h_2	5	10
$r_{reroute}(h_1)$	r	–	0.5	1
$r_{reroute}(h_2)$	r	–	0.5	1

($r_{block}(network)$) costs and duration will increase, but the migration is an unnecessary response. This examples shows, how important a decision process over the overall response set can be. Responses executed without coordination will lead to inefficient operations with respect to the entire target system.

6.3.3 Formulating the Optimization Problem

The model from Subsection 6.3.1 describes the network environment, the security incident, and responses available. The corresponding optimization problem formalizes the following question:

In a given model instance, which subset from the set of responses available

- *Frees all affected entities from a security incident,*
- *Has minimal cost within the given set of metrics, and*
- *Has lower cost than the security incident being unmitigated?*

When transforming a problem to the form required to solve it as an LP problem, it is common to distinguish between two kinds of constraints: *feasibility constraints* and *optimality constraints*. Feasibility constraints force the solution to be within the constraints of a valid solution, whereas, the optimality constraints drive the solution into the desired direction of minimization or maximization. The constraints defined in Subsubsection 6.3.3.4 and Subsubsection 6.3.3.5 are explicitly formulated to be linear and directly applicable to a MILP problem.

6.3.3.1 Input

The different inputs needed for the response selection can either originate from infrastructure information and policy definitions known in advance (marked with •) or information extracted from the security incident (marked with ∗) or collected during response execution (marked with †). From the system model in Subsection 6.3.1, we get the following inputs to the optimization problem:

- The set of entities in the network: S (•)
- The set of entities affected by the security incident: A (∗)

- The set of entities capable of executing responses: E ($\bullet$)
- The set of responses to counter the security incident: R ($\bullet$)
- The set of metrics responses can be evaluated under: M ($\bullet$)
- The cost each response has with respect to the metrics: c ($\dagger$)
- The damage if the security incident is not countered: d ($*$)
- Information on which responses are in conflict: o ($\bullet$)
- The entity a response can be executed on: x ($\bullet$)
- Information on which hosts a response frees: f ($\bullet$)
- Information if a response has preconditions: p ($\bullet$)

This input information has to be gathered from the information model examined previously in Section 3.3. The set of entities in the network (S) is given as infrastructure information. Hereby, not only a `Device` node is a network entity, but includes `Service` or `User` nodes as well. The set of entities affected by the security incident (A) are all nodes connected with all `Alert Context` nodes using the `alertcontexthastarget` relation. The set of responses to counter the security incident (R) are preselected by the response identification module and are connected to the `Bundle` node that is currently processed by the response selection module. The set of entities capable of executing responses (E) are all nodes connected with the `Implementation` nodes of the `Bundle` node by the `implementationisdeployedondevice` relation.

The set of metrics (M), responses can be evaluated under, is stored in the `Metric` nodes. The cost each response has with respect to the metrics (c) can be evaluated using the relation between an `Implementation` and a `Metric` node storing the metric value.

The information which responses are in conflict (o) can be gathered by the self-reference of the `Response` node (`responseconflictswithresponse` relation). The same applies for the information if a response has preconditions (p) using the `responseispreconditionofresponse` relation The entity (x), a response can be executed on, can be determined by the relation between the `Implementation` and the `Device` node (`implementationisexecutedbydevice` relation). The information if a response frees a target (f) is given by the `implementationisdeployedondevice` relation between the `Implementation` node and the target.

6.3.3.2 Output

The solution of the optimization problem states a vector $\overrightarrow{n}$ for each response $r_i \in R$ whether it should be executed ($n_i = 1$) or not ($n_i = 0$). Since these are integral values, the problem is a MILP problem. In addition, the solution also offers the total cost c_{total} of all selected responses.

This output has to be mapped appropriately to the information model examined previously in Section 3.3. As response selection is executed on a certain bundle (`Bundle` node) with connected `Implementation` nodes the relation in-between is utilized. The `Implementation` nodes reflect the responses of the optimization problem. As the output of the optimization problem is the vector of all responses including the selection decision (selected ($true$) or not ($false$)) each edge from the `Bundle` node to an `Implementation` node can be queried and the `Selected` attribute is set appropriately.

6.3.3.3 Objective Function

The goal of the optimization problem is to minimize the cost of the responses selected to counter the security incident. Therefore the objective function is defined as:

$$\begin{aligned} min(\sum_{i=1}^{|R|}\sum_{j=1}^{|M|} n_i c_{i,j} w_j) &= min(n_1 \cdot c_{1,1} \cdot w_1 + \cdots + n_{|R|} \cdot c_{|R|,|M|} \cdot w|M|) \\ &= min(n_1 \cdot (c_{1,1} \cdot w_1 + \cdots + c_{1,|M|} \cdot w|M|) + \cdots + \\ &\quad n_{|R|} \cdot (c_{|R|,1} \cdot w_1 + \cdots + c_{|R|,|M|} \cdot w|M|)) \end{aligned} \tag{6.2}$$

Whereas, n_i is a binary variable for each response within the set of candidate responses, that can either be $true$ or $false$. The metrics for each response are reflected in $c_{i,j}$. Whereas, $c_{i,j}$ is metric j of response i. Each metric j can be weighted using the parameter w_j. Within the objective function the weights for a certain metric is constant, meaning the metrics are weighted the same between responses.

6.3.3.4 Feasibility Constraints

Without any feasibility constraints, the optimal solution would be to select no responses at all. Then, the total cost of the selected responses would be 0. The following feasibility constraints force the optimization towards a practically useful solution.

All Affected Entities are Freed Every entity affected by the security incident has to be freed. Such an entity has to be affected by at least one selected ($n_j = 1$) response:

$$\forall_{a \in A} : \sum_{j=1}^{|R|} n_j f_{a,j} \geq 1 \tag{6.3}$$

This constraint ensures that the whole security incident is considered at once, such that a comprehensive view is provided (cf. Requirement R1). This constraint does not exclude that an entity is freed due to multiple responses. For example, if a response r_1 frees entities e_1 and e_2 and another response r_2 frees entities e_1 and e_3, both responses can be applied simultaneously, if not conflicting.

Each Response Can Only Be Executed Once We define that a response $r \in R$, if executed once, is fully effective and does not need to be executed multiple times:

$$\forall_{r_i \in R} : 0 \leq n_i \leq 1 \tag{6.4}$$

As responses are defined as binary values, the response can either be set to 0 or 1. In the first case, the response is not selected in the later one, the response is selected.

Total Cost of Responses Has to Be Below Cost of Damage Based on the damage a security incident inflicts in the network, we do not want responses to generate more cost than the security incident itself. Therefore, we add a constraint limiting the cost of the responses to be below the expected cost of damage:

$$\forall_{m \in M} : \sum_{i=1}^{|R|} n_i c_{i,m} \leq d_m \tag{6.5}$$

If no damage is given for a metric m_i, it is set to $d_i = \infty$. Within this formula, we assume that for each metric a certain upper bound is given. A selection of responses is only valid, if all boundaries for all metrics can be adhered. Otherwise, no solution can be found for the given problem and taking no action is preferred.

No Conflicting Responses are Executed We prevent conflictive responses to be selected at the same time:

$$\sum_{i=1}^{|R|} \sum_{j=1}^{|R|} o_{i,j} n_i n_j = 0 \tag{6.6}$$

This constraint ensures that either r_i or r_j can be selected from the set of candidate responses if they are conflicting, that means the function $o(r_i, r_j)$ is evaluated to 1 ($true$).

Execute Preconditions We have to ensure that preconditions of a response are executed in case the response is selected:

$$\forall_{(r_1,r_2)\in p} : r_1 \leq r_2 \tag{6.7}$$

This constraint ensures that response r_1 will be set to 1, if r_2 is chosen. Response r_1 reflects the precondition of response r_2. Together, with the objective function (see Subsubsection 6.3.3.3) and the optimality constraints (see Subsubsection 6.3.3.5) r_1 will not be executed if response r_2 is not executed, as this response would be unnecessary.

6.3.3.5 Optimality Constraints

The objective of this optimization problem is to determine the set of responses with minimal costs fulfilling the feasibility constraints defined in Subsubection 6.3.3.4.

$$\forall_{m\in M} : c_{m_{total}} = \sum_{i=1}^{|R|} n_i c_{i,m} \Rightarrow$$

$$\forall_{m\in M} : c_{m_{total}} - \sum_{i=1}^{|R|} n_i c_{i,m} = 0 \tag{6.8}$$

Based on this formalization, the size of the resulting optimization problem depends on the size of input sets:

- Number of variables: $|M| + |R|$
- Number of constraints: $2|M| + 2|R| + |A|$
- Number of non-zero elements in the problem matrix:

$$\sum_{i=1,j=1}^{|A|,|R|} f_{i,j} + 2|R||M| + |M| + 2 \sum_{i=1,j=1}^{|R|,|R|} h_{i,j}$$

Since the number of metrics is expected to be more or less constant and small, the size of the problem and its complexity is defined by the number of available responses and the number of affected entities. Additionally, the number of conflicts and preconditions, as well as the coverage factor of a response, the number of entities a response frees, influence the problem size and complexity.

6.4 Implementation

To be able to evaluate the applicability and performance of the presented approach, we implement the proposed design and use this implementation for the evaluation in Section 6.5. The implementation is available on GitHub [1].

The main tasks of this implementation are to generate network environment and incident scenario datasets, to transform these datasets into a linear optimization problem, to instrument established MILP solvers to solve this problem and obtain the computed solution. As storage for all required and generated data, a PostgreSQL database is used.

This functionality is realized by a controller written in Python3. This controller executes the following tasks in the given order:

- Check Database for needed tables within the database
- Create a network environment (test data) with respect to given user inputs
- Generate specific incident scenarios by specifying the number of affected entities
- Generates the corresponding optimization problem, as described in Section 6.3
- Instrument solvers to solve the optimization problem
- Store results and measurement data into the database

First, the controller checks if all needed tables are available in the database. This includes tables to store the network environment as well as meta-data from the single experiments executed. In case a table is missing, the controller will add the table with corresponding columns and constraints.

In a next step, the network environment is generated. Within the configuration file of the controller, it can be specified how many entities, responses and conflicts have to be generated. Additionally, the maximum coverage factor of a response can be determined. For each response the coverage factor is selected randomly and uniformed distributed between 1 and this specified maximum coverage factor. As each response needs costs for different metrics, within the configuration file the name and the upper bound of the specific metric has to be specified. For each given metric a randomized value between 0 and 1 using uniformed distribution is assigned to each response that is generated. All generated incident scenarios are based on this network settings. Therefore, in case different network settings shall be used, the use of multiple databases is recommended. As the system supports the automatic setup of a database, no additional effort is needed.

The next step is setting up incident scenarios from the given network environment. Within the controller's configuration file the user can specify the following values for entities, responses as well as conflicts: the upper bound, a start value and a step size. The controller will determine different incident scenario from those settings, whereas an incident scenario is defined by the number of infected entities (E), the number of responses (R) and the number of conflicts (C). For each incident scenario the controller determines the infected entities by selecting the first entries from the database until E entries are fetched. Afterwards, the corresponding responses are fetched until the controller has collected R responses. The controller will only fetch distinct responses where at least one of the E infected entities benefits from. At last the controller fetches C conflicts, whereas only conflicts are included between the previously fetched R responses.

[1] https://github.com/Egomania/ResponseSelection

This procedure allows a fine-grained regulation of different influence factors within the optimization problem.

In the next step the incident scenarios are handed over one after another to the specific solvers. The solvers to use can be specified within the controller's configuration file and are loaded dynamically during runtime. In case new solvers have to be integrated, the implementation of the new solver has to follow the given interface definition.

The controller interfaces within the current implementation with well-established and optimized LP solutions: *GNU Linear Programming Kit (GLPK)*[2] and *IBM ILOG CPLEX Optimization Studio (CPLEX)*[3]. To interface with GLPK the controller uses *Python-GLPK*[4] and *pycpx*[5] to interface with CPLEX.

Solving a MILP problem is often stated as computationally expensive and time consuming. To be able to compare the MILP-based approach, the controller additionally implements two simplistic heuristics to find a solution for the security incident as a point of reference for the performance of the optimization:

The **Cheapest-First-Algorithm** orders all available responses with respect to their cost. For each affected entity, the cheapest response is selected. In case the selected response covers more than one entity, no response will be selected for the other entities since they are already covered.

The **Coverage-First-Algorithm** sorts the available responses with respect to their coverage. The responses that help the most affected entities are considered first. On a tie, the cheapest is selected. Responses are drawn as long as there are hosts left which are not covered. Responses which help only hosts which are already covered are ignored.

The solver module for all supported solvers is capable of creating as well as storing the problem definition, creating as well as storing the solution and reading previously generated problem files. The concrete behavior, e.g. not creating a problem but reading from a description file and storing the results into the database can be configured using the controller's configuration file. As the execution can be done multiple times for evaluation purposes it is recommended to create the problem description first and than execute the problem solving within a separate run.

In a last step the results of each solver are stored within the database back-end. This not only includes the concrete results, like which responses are used and what costs are calculated, but additional meta-data describing the incident scenario that was used or the specific solver settings, as well as used metrics and boundaries that can be configured as well. Within the implementation we do currently not support preconditions and weights for different metrics.

6.5 Evaluation

The goal of this evaluation is to analyze whether the presented approach relying on MILP is suitable for response selection, how MILP compares to simplistic approaches like heuristics and to determine how different problem properties influence the performance of both the MILP and heuristic approaches towards response selection. First, we compare our approach to the requirements stated in Subsection 6.2.1 in Subsection 6.5.1. Afterwards, we introduce the evaluation methodology in Subsection 6.5.2 that is used within our evaluation. Our evaluation considers the performance of our approach (see Subsection 6.5.3)

[2] https://www.gnu.org/software/glpk/
[3] http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/
[4] http://www.dcc.fc.up.pt/~jpp/code/python-glpk
[5] http://www.stat.washington.edu/~hoytak/code/pycpx/index.html

as well as the solution quality (see Subsection 6.5.4).

6.5.1 Requirement Alignment

The approach provided for response selection is cost-sensitive (cf. Requirement R1). The costs of responses regarding multiple metrics are taken into account. Hereby, the metrics can be chosen freely. For each metric a boundary exists reflecting the expected damage of the attack within the target system. To ensure that the response costs are below the damage, Equation 6.5 is used. Responses are selected according to their costs and are not mapped statically to a specific security incident.

The approach provided for response selection provides a comprehensive view (cf. Requirement R2). Our response selection approach is not limited to a single victim or a single attack but is generic in this sense. The security incident is solved as a whole and not partially. To ensure this behavior Equation 6.3 is used.

The approach provided for response selection provides a conflict-free selection of responses (cf. Requirement R3). Using Equation 6.6 ensures that only non-conflicting responses can be used. This equation allows only to use one of two conflicting responses but not both within the same response plan.

The approach provided for response selection supports preparation actions (cf. Requirement R4). To ensure that all preconditions of a responses are executed if the response itself is executed, Equation 6.6 is used. The costs of the preconditions are also included within the total costs as preconditions are represented as responses.

The approach provided for response selection supports responses helping multiple victims at the same time (cf. Requirement R5). For each victim the Equation 6.3 is defined. A response can be a term within multiple of those equations, reflecting that a response is effective for multiple victims if this response is selected.

The approach provided for response selection provides flexible cost assessment (cf. Requirement R6). To calculate the costs to solve the security incident, Equation 6.8 is used. Within this equation, multiple metrics can be used. The metric itself is not bound to the selection approach and is therefore, exchangeable with other cost assessment strategies with respect to the response costs. As the damage cost of a security incident is reflected as boundary for our approach, this assessment is not bounded to our system as well.

6.5.2 Evaluation Methodology

Based on the implementation presented in Section 6.4, we analyze and compare two important criteria: *(a) the performance* of the MILP approach and both heuristics and *(b) the quality* of the solution (i.e. the cost to counteract the security incident) of all approaches. We analyze the behavior of the presented MILP approach and both heuristics in different incident scenarios by increasing problem complexity. We raise the problem complexity by increasing (one at a time) the

a. Number of responses,

b. Number of entities,

c. Number of conflicts, and

d. Number of entities a response is applicable to (coverage factor)

in the problem while keeping the number of remaining problem parameters fixed.

We employ seven datasets for network environments with differing average and maximum *response coverage factors* as shown in Table 6.4. The response coverage factor g is the number of hosts a response frees. If not stated differently, dataset D3 is used in the evaluation since it represents expected average values for responses.

Tab. 6.4: Evaluation Datasets and Response Coverage Factors

Setting	D1	D2	D3	D4	D5	D6	D7
$avg(g)$	1	5	10	15	20	25	50
$max(g)$	1	10	20	30	40	50	100

In order to avoid overhead from Python bindings and the Python interpreter itself, we use the command line clients provided by the GLPK and CPLEX solver and give them generated problem description files in CPLEX format as input.

We execute the evaluation using a system equipped with an Intel Xeon E3-1275 CPU running at 3.5 GHz with 4 physical cores and Hyper-Threading enabled and 16 GB of RAM. The operating system is Ubuntu 15.10 64-Bit with Python 3.4, GLPK 4.55, and CPLEX 12.6.1. GLPK does not support multiple CPU cores and therefore uses only one of the CPU cores. CPLEX supports multiple cores and can therefore benefit from the multicore system used in this evaluation. All test runs are executed five times. In the remainder of this evaluation, all figures depict the average creation, calculation and execution time as well as the response costs for each measurement. Additional information including standard deviation is described in the corresponding text for each measurement. As CPLEX was measured using its inbound time measurement facility, the resolution is bound to 0.01s. Therefore, a standard deviation of 0 is the consequence for small measurements. In such cases, the standard deviation of CPLEX is not listed explicitly.

6.5.3 Evaluation of Solver Performance

To evaluate the applicability of the presented approach and its performance and to be able to analyze the impact of different problem properties, we evaluate the execution time based on different input settings with increasing complexity. The execution time of both solvers includes reading the problem from the file, setting up the optimization problem and calculating the optimal solution. This evaluation has shown that problem creation and reading from a file has only negligible impact on the execution time. The execution time of both heuristics includes the ordering of the data as described in Section 6.4.

6.5.3.1 Increasing Number of Responses

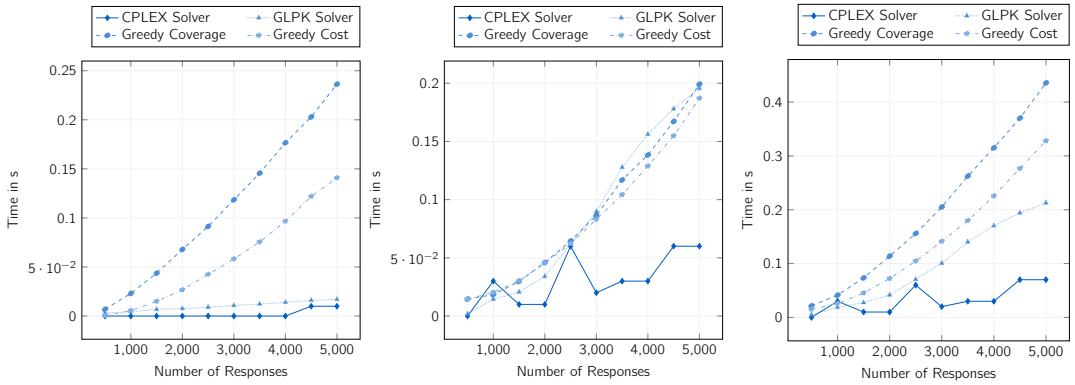
With the first evaluation, we analyze the impact of responses in the incident scenario. The evaluation starts with 500 responses which is raised up to 5000 responses using a step size of 500. The number of entities in the scenario is fixed to 500 and the number of conflicts between responses is 100. The resulting average calculation, creation and execution time is depicted in Figure 6.2. The standard deviation of the execution time for all runs of all solvers is shown in Table 6.5. All numbers are accurate to 5 decimal places.

First, we examine the standard deviation of the considered scenario (cf. Table 6.5). With an increasing number of responses the standard deviation is slightly increasing. Except for the first case (500 responses) GLPK has the most stable results. The maximum

Tab. 6.5: Standard Deviation for an Increasing Number of Responses

# Responses	500	1000	1500	2000	2500	3000	3500	4000	4500	5000
Greedy Cost	0.00012	0.00039	0.00073	0.00056	0.00131	0.00439	0.00326	0.00662	0.0033	0.00477
Greedy Coverage	0.00013	0.00072	0.00053	0.00128	0.00121	0.00193	0.00402	0.00653	0.00459	0.00332
CPLEX	-	-	-	-	-	-	-	-	-	-
GLPK	0.00075	0.0004	0.00049	0.00136	0.0004	0.00102	0.0011	0.00183	0.00141	0.00162

standard deviation for all iterations, all approaches and all complexity settings is 0.007 s. Therefore, the results are very stable.



(a) Creation Time **(b)** Calculation Time **(c)** Execution Time

Fig. 6.2: Performance with a Varying Number of Responses and a Fixed Number of Entities and Conflicts

Taking a closer look on the measurement result (cf.Figure 6.2) we can examine the following observations. The number of responses increases the problem size in multiple dimensions: with each additional response, the number of terms in the objective function grows, additional constraints are added (cf. Equation 6.4), and the number of terms in the constraints increases (cf. Equation 6.3). But still, both the GLPK and the CPLEX solver are faster than both heuristics. We assume that the reason for this is that both heuristics are implemented in Python. As both heuristics have to order the list of responses with respect to costs and in case of the coverage-first algorithm additionally with respect to the coverage factor, the execution time increases with the number of responses. As both algorithms are dominated by the search, the growth is expected to be $n\,log(n)$.

6.5.3.2 Increasing Number of Entities

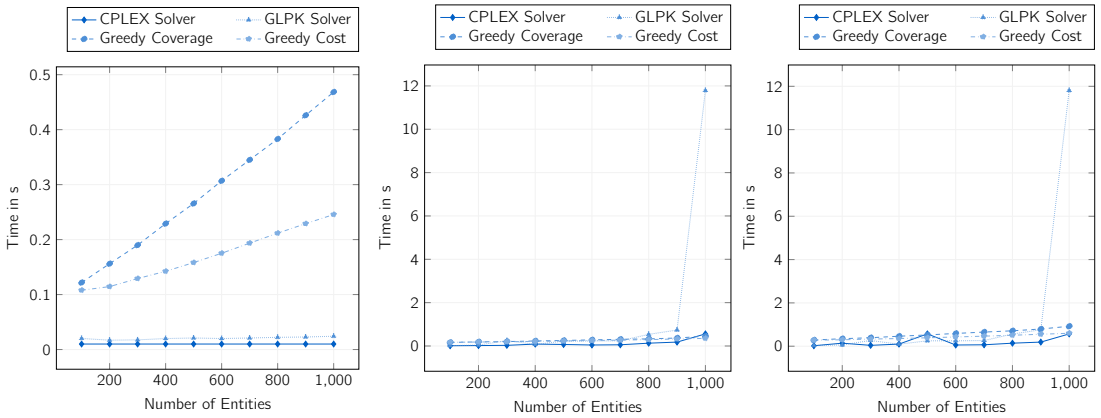
Next, we analyze the impact of entities in the incident scenario. The evaluation starts with a number of 100 entities which is raised up to 1000 using a step size of 100. The number of responses in the scenario is fixed to 5000 and the number of conflicts between responses is 100. The resulting creation, calculation and execution time is depicted in Figure 6.3. The standard deviation of the execution time for all runs of all solvers is shown in Table 6.6. All numbers are accurate to 5 decimal places.

First, we examine the standard deviation of the considered scenario (cf. Table 6.6). Within this table, no clear trend can be identified. The standard deviation is very small such that no clear statement is possible. The maximum standard deviation for all iterations, all approaches and all complexity settings is 0.045 s. Therefore, the results are very stable.

Taking a closer look on the measurement result (cf.Figure 6.3) we can examine the

Tab. 6.6: Standard Deviation for an Increasing Number of Entities

# Entities	100	200	300	400	500	600	700	800	900	1000
Greedy Cost	0.00205	0.0052	0.00833	0.00209	0.0019	0.00452	0.00433	0.00594	0.00612	0.00634
Greedy Coverage	0.00465	0.00427	0.00416	0.00703	0.00998	0.01277	0.01608	0.00492	0.00869	0.01221
CPLEX	0.004	-	-	0.004	0.01721	0.0049	0.0049	-	-	0.01720
GLPK	0.002	0.0008	0.00316	0.00049	0.00271	0.00271	0.00431	0.00595	0.0026	0.04497



(a) Creation Time **(b)** Calculation Time **(c)** Execution Time

Fig. 6.3: Performance with a Varying Number of Entities and a Fixed Number of Responses and Conflicts

following observations. The number of constraints increases with the number of hosts (cf. Equation 6.3). Those constraints may have several terms as they reflect which response can help which entity.

However, this seems to have a moderate impact on the performance of CPLEX for small values apart from the measurement with a number of 500 entities. But only after a number of 800 entities the execution time increases significantly. With CPLEX we have solver capable of solving the problem faster than both heuristics. In comparison GLPKs execution time increases significantly with increased problem complexity. In the final measurement for GLPK with 1000 attacked entities the execution time is many times higher than with 900 entities.

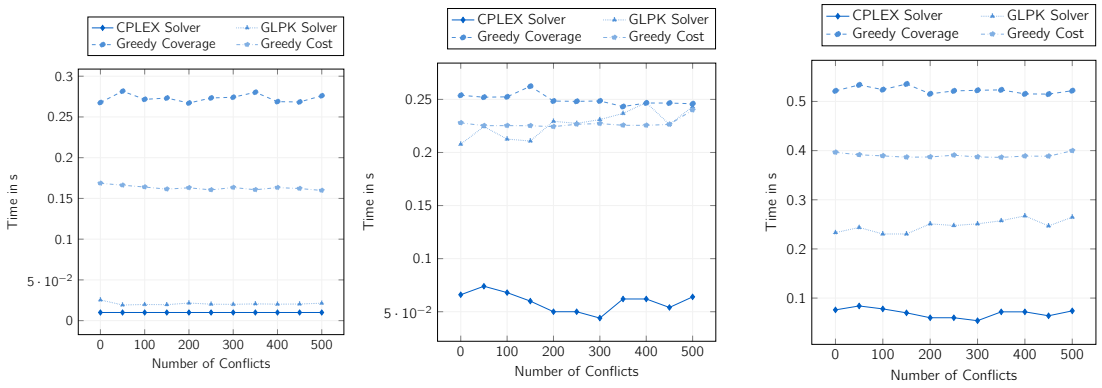
6.5.3.3 Increasing Number of Conflicts

In this setting, we analyze the impact of conflicts between responses in the incident scenario. The evaluation starts with a number of 0 conflicts which is raised up to 500 using a step size of 50. The number of responses in the scenario is fixed to 5000 and the number of entities is 500. The resulting creation, calculation and execution time is depicted in Figure 6.4. The standard deviation of the execution time for all runs of all solvers is shown in Table 6.7. All numbers are accurate to 5 decimal places.

Tab. 6.7: Standard Deviation for an Increasing Number of Conflicts

# Conflicts	0	50	100	150	200	250	300	350	400	450	500
Greedy Cost	0.02348	0.01297	0.00963	0.00510	0.00367	0.00411	0.00493	0.00177	0.00827	0.00478	0.02020
Greedy Coverage	0.00700	0.03084	0.01521	0.01639	0.00784	0.01473	0.01352	0.02155	0.00766	0.00428	0.01058
CPLEX	0.00490	0.00490	0.00400	-	-	-	0.00490	0.00400	0.00400	0.00490	0.00490
GLPK	0.00611	0.00496	0.00496	0.00496	0.00276	0.00338	0.00228	0.00859	0.00598	0.00160	0.00136

First, we examine the standard deviation of the considered scenario (cf. Table 6.7). For CPLEX all measurements are very stable as they have a low standard deviation. For all other solvers the standard deviation fluctuates, but is very small. The maximum standard deviation for all iterations, all approaches and all complexity settings is 0.03 s. Therefore, the results are very stable.



(a) Creation Time **(b)** Calculation Time **(c)** Execution Time

Fig. 6.4: Performance with a Varying Number of Conflicts and a Fixed Number of Entities and Responses

Taking a closer look on the measurement result (cf.Figure 6.4) we can examine the following observations. Comparing the results from this and the previous section, the measurement results look very different. Adding more conflicts leads to an increased number of constraints (cf. Equation 6.6) but the structure of those constraints is different compared to constraints that are added with a rising number of entities. Constraints describing conflicts have a limited number of terms and have, therefore, a simpler structure.

This observation is reflected in the measurements, as an increasing number of conflicts has no significant impact on the execution time of all tested approaches. As a conclusion, conflicts do not harm performance, as long as only few responses conflict with each other.

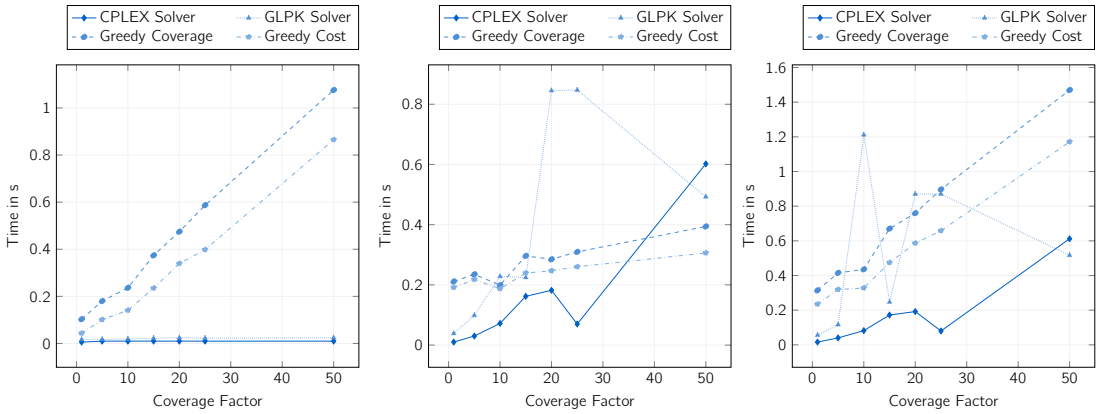
6.5.3.4 Increasing Coverage Factor

For this analysis, we use datasets with varying coverage for responses, as listed in Table 6.4. The maximum value for the coverage factor is increased from 1 to 50 using a step size of 10. Additionally, a maximum coverage factor of 100 is used for the final measurement. For all tests the number of entities is fixed to 500, the number of responses is fixed to 5000, and a number of 100 conflicts is used. The results are shown in Figure 6.5. The standard deviation of the execution time for all runs of all solvers is shown in Table 6.8. All numbers are accurate to 5 decimal places.

Tab. 6.8: Standard Deviation for an Increasing Coverage Factor

AVG Coverage	**1**	**10**	**20**	**30**	**40**	**50**	**100**
Greedy Cost	0.00190	0.00597	0.00477	0.00663	0.00476	0.00683	0.66970
Greedy Coverage	0.02046	0.00419	0.00332	0.01023	0.01256	0.02307	0.01166
CPLEX	0.00490	-	0.00400	0.00400	0.01470	0.00632	0.01166
GLPK	0.00133	0.00133	0.02576	0.00141	0.00708	0.00708	0.00232

First, we examine the standard deviation of the considered scenario (cf. Table 6.8). For CPLEX all measurements are very stable as they have a low standard deviation. For all other solvers the standard deviation fluctuates, but is very small. The maximum standard deviation for all iterations, all approaches and all complexity settings is 0.038 s. Therefore, the results are very stable.



(a) Creation Time **(b)** Calculation Time **(c)** Execution Time

Fig. 6.5: Performance with a Varying Coverage Factor and a Fixed Number of Entities, Responses and Conflicts

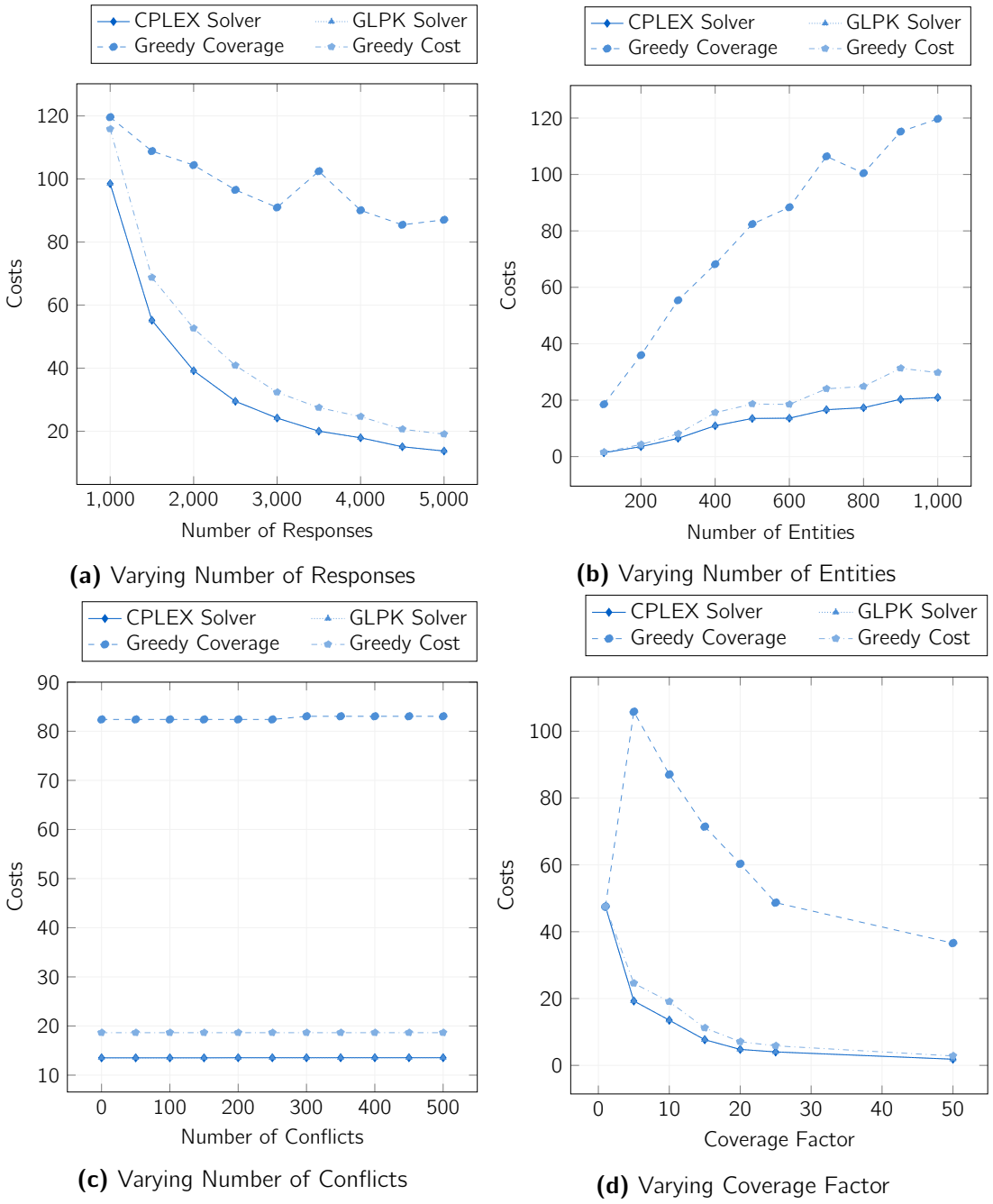
Taking a closer look on the measurement result (cf.Figure 6.5) we can examine the following observations. A higher coverage factor for responses leads to a higher number of terms in the constraints as described in Equation 6.3 as a response will appear more often within those constraints as the response is capable of freeing a higher number of entities. An increasing coverage factor has the highest impact on performance within the evaluation. The time consumption of both heuristics is growing rapidly, but linear with an increasing coverage factor. From the measurements the behavior of GLPK is unclear as the time consumption goes up and down without clear trend. CPLEX has one outlier with a coverage factor of 25. The rest of the measurements show a rising tendency. Nevertheless, CPLEX performance beats both heuristics and shows that the presented approach is usable even with complex network environments.

6.5.4 Solution Quality

Besides performance with respect to execution time, another important aspect to consider is the quality of solutions provided by MILP in comparison to solutions of heuristic solvers. While the solution provided by MILP solvers is optimal within the given objective function, the quality of a heuristic need not be optimal in any way. We therefore compare the provenly optimal solutions provided by the MILP approach and the solutions provided by the Greedy Coverage-First and the Greedy Cheapest-First heuristic for all scenarios used for the performance analysis. In Figure 6.6a the number of responses is increased, in Figure 6.6b the number of entities, in Figure 6.6c the number of conflicts between responses and in Figure 6.6d the coverage factor of a response is increased.

In the following, we point out the potential cost benefits in case an optimizer instead of the Cheapest-First heuristic is used. The Coverage-First metric has worse results in all scenarios than the Cheapest-first metric.

In case the number of responses increases, the average cost saving potential is nearly 33 %. At minimum 17.6 % costs can be saved, at maximum around 39 % are possible.



(a) Varying Number of Responses

(b) Varying Number of Entities

(c) Varying Number of Conflicts

(d) Varying Coverage Factor

Fig. 6.6: Cost Evaluation with Increasing Problem Complexity in One of the Following Dimensions: Number of Responses, Entities, Conflicts, or Coverage Factor

The percentaged cost saving possibilities slightly decrease during the scenario.

In case the number of entities increases, the average cost saving potential is around 36 % and lies between 15 % at minimum and 54 % at maximum. The percentaged cost saving potential increases slightly with an increasing number of entities.

In case the number of conflicts increases, the cost saving potential stays nearly constant at around 38 %. Within the datasets the optimal solution was rarely impacted with additional conflicts, but the problem complexity was increased.

In case the coverage factor increases, the percentaged cost saving potential increases slightly. The first measurement shows, that all approaches find the optimal solution in case one response can free only one entity. With an increasing coverage factor the cost calculated drift apart. At minimum a 27 %, at maximum 53 %, and in average 37 % cost can be saved by using an optimizer.

In summary an average cost saving of 36 % is possible for all tested scenarios. The smallest potential saving is around 15 %, which is still a notable amount. The minimum, maximum and average costs savings of the optimized solution compared to the Cheapest-First metric is summarized in Table 6.9.

Tab. 6.9: Minimum (min), Maximum (max) and Average (avg) Cost Saving between Optimized Solution and Cheapest-First Metric in %

Scenario	avg	max	min
Increasing Number of Responses	33.23	38.88	17.56
Increasing Number of Entities	36.25	54.08	15.15
Increasing Number of Conflicts	37.74	37.89	37.65
Increasing Coverage Factor	37.25	52.75	27.45

6.6 Related Work

Response selection is an essential part in the field of IRSes. Different surveys on IRSes [4, 50, 130] show the evolution IRSes went through with respect to response selection strategies. Earlier IRSes started with static mappings that connect an alert to a predefined response. Approaches, using static mappings, are examined in more detail in Subsection 6.6.1. As static mappings are most often based on simple tables, are not a sufficient solution for complex networks, and cannot cope with environmental changes [151], IRSes became more flexible using more dynamic response selection strategies.

Later on, cost-sensitive mappings [93] were introduced that use both estimated damage and intrusion costs and try to balance them against each other to find a good solution. This cost-sensitive approach was improved over time to become more precise in terms of estimating response costs and damages [145, 146]. Based on these approaches a schema to estimate potential damage and costs in distributed systems was proposed [167]. In Subsections 6.6.2 and 6.6.3 those approaches are examined in more detail. Approaches presented in Subsection 6.6.3 combine cost-sensitive approaches with additional methods, like graphs or game-theory.

Some other approaches based for example on attack trees or different probabilities and approaches for particular domains are examined in more detail in Subsection 6.6.4.

However, these approaches lack in finding an optimal solution employing a holistic approach to counteract the whole security incident. Neither the possibility that a single

response can cover more than one network entity nor conflicting responses are respected by the work described before. A more detailed comparison of the presented approaches against the requirements given in Subsection 6.2.1 is given in Subsection 6.6.5.

6.6.1 Policy-Based Selection

In [29] the response selection is done via policies specified in Organization Based Access Control (OrBAC). The systems determines on which components to react and how to react as well as the granularity of the reaction. The security policy is inferred from the ontology using rules written in the Semantic Web Rule Language (SWRL) and determine the action to take in case of an intrusion. The focus of the attack is determined using the mapping from the Intrusion Detection Message Exchange Format (IDMEF) alert field source into the specified role attacker.

In [76] a policy following the Event Condition Action (ECA) scheme is used to implement the policy that select a response. Each rule can be expressed as follows:

Listings 6.1: Event Condition Action (ECA) Rules to Implement Policies

```
1 On <Event>
2 IF <Condition>
3         THEN <Action>
4         CONFIRM <Confirmation Action>
5                 ON SUCCESS <Resolution Action>
6                 ON FAILURE <Error-Handling Action>
```

Policies can be activated, dropped or suspended during lifetime. The policy is claimed to be protected against several attack, like malicious policy updates or malicious policy deletion. The Response Action is selected using the Most-Serve-Policy or Least-Serve-Policy Principle depending on the severity level.

In [124] a set of different policies is introduced to cope with different aspects of the intrusion detection and intrusion response process. This includes Detection Control Policies, Core Response Policies, Boundary Control Policies, Vulnerability Policies as well as Vulnerability Scanning Policies.

In [24] a hybrid framework for data loss prevention and detection is proposed. The main idea is to identify anomalous behavior of users. In case a transaction is identified as anomalous the transaction is blocked, otherwise the transaction is accepted. In case no decision can be made, the transaction is allowed but a warning is generated.

In [86] case-based reasoning is used to identify similar security incident and apply the same rule set to those identified security incidents. They mainly adapt the firewall configuration to block IPs with similar misbehavior as previously identified ones.

6.6.2 Basic Cost-Sensitive Approaches

The most cost-sensitive approaches have in common that an action is selected only if the costs of the response are lower or equal the damage costs. Additionally, most of them include the confidence level of the intrusion, so that in fact the costs of the response do not overshadow the benefits. The main idea of the following approaches, is to select one single response with the lowest cost. To determine the lowest costs, different aspects are taken into account.

The response selection in [57] is done in the *Countermeasure Evaluation* step. They use their metric called *RORI (Return on Response Investment)* described earlier in Section 6.1. In this step, the *RORI* for the first response on the list is calculated and

compared to a default *RORI*, set to zero. If the calculated *RORI* is higher than this default value, the response is used instead of the default value. In case of equality the the one with the lowest ARC (*Anual Response Costs*) value is selected. In case the calculated value is lower than the default value, the response is neglected. Using this scheme the complete list of available responses is tested for selection. The authors themselves claimed that interdependencies of responses are not considered as limitation. Moreover, only one response can be selected and the use of multiple responses is not possible.

In [141] the authors determine the optimal response from a set of responses by applying the following strategy: First, all possible responses are selected by choosing all responses having costs below the expected damage multiplied with the confidence level of the attack. From those candidate responses an optimal response has to be chosen. An optimal response should provide the maximum benefit (high *Success Factor*) at the lowest risk (low *Risk Factor*). They calculate the expected value of a response using the success and risk factor and select the response with the highest expected value. The approach in [141] is adaptive in terms of the success factor that is increased in case of a successful execution but lacks in selecting multiple responses for a distributed security incident. A similar approach is presented in [147]. The response costs are assessed using the *Response Goodness* and the impact of the response on the system. Likewise, simply the cheapest one is chosen. In [145, 146] a similar approach is presented. The main contribution of those two works is the improvement of the cost calculation for response costs and the costs of the security incident.

[151] presents an approach using dependency trees to model network configurations and a cost model to estimate the effects of responses. A *response action* is defined as a set of *response items*. Response actions can be assessed with an impact evaluation function and already executed response actions are combined in a *response configuration*. Responses are assessed using a *penalty function* describing the negative impact on the system a response has during its execution, e.g. downtime of services. Each time a response has to be selected, all available responses are added to the response configuration and the resulting impact is calculated. The response action with the lowest negative impact stays in the response configuration. This allows to reach a local optimum, but not a global optimum for sure.

In [8] a response agent, called *ARB (Automated Response Broker)* is introduced. For response selection they model their target system using a *resource type hierarchy* and a *system map*. The hierarchy structures available system resources and possible responses executable on those resources. The map reflects dependencies between resources using a directed graph. Each node in the graph reflects an important system resources and stores possible responses gained from the hierarchy. Additionally, an activation condition (list of applicable attacks) and side-effects on other resources are stored. The costs of a node are partially assigned by the system administrator and reflect the importance of the resources. Missing node costs are calculated based on the dependencies. The attack costs are the sum of all costs of the affected nodes, the response benefit are the sum of costs a response frees, and the response costs, are the sum of costs a response effects because of side effects. For the selection criterion they chose the *Minimax risk criterion*. Additionally, they propose strategies to calculate the costs in case of uncertainty, what an attacker really did.

In [110] the positive and negative effects of a response are combined to calculate the effectiveness of a response. The response with the highest effectiveness is chosen. In case two equal responses are available, an arbitrary one of both is selected.

6.6.3 Advanced Cost-Sensitive Approaches

The approaches presented in the following go beyond traditional cost-sensitive approaches. They do not simply calculate the possible damage of the security incident and the cost of a single response and chose the cheapest one, but try to model the response selection strategy in a more complex manner.

In [83, 84] the costs of a response are calculated using a dependency graph. The impact of an attack and a response are compared to choose the best response. The decision whether a response is in the set of candidate responses is passed to the IRS system when an attack is detected. The attack impact IV_r is evaluated using the *Common Vulnerability Scoring System (CVSS)*. The investigates the benefits of all candidate responses in case they would be applied. They support only two classes of responses: *component deactivation*, hereby the target resource is made unavailable, and *dependency alteration*, hereby only the dependency is made unavailable but not all children of the targeted resource.

In [79] the response with the highest negative effect on the success probability of an attack is used. First, candidate responses are selected by the use of an anti-correlation between a response C and an attack A. A response is assumed to be effective against an attack in case the postcondition of C matches the precondition negation of attack A. This is modeled with LAMBDA. In the next step all candidate responses are are simulated and the effect of the success likelihood is determined.

In [181] the so-called *Response and Recovery Engine (RRE)* implements a game-theoretic approach for response selection using a multi-step, sequential, hierarchical, non-zero-sum, two-player stochastic game. An *attack-response tree (ART)* is used to determine a security incident and select a response based on Boolean logic. For each step in the game, a new ART is generated and converted into a *Partially Observable Competitive Markov Decision Process (POCMDP)*. The optimal response is chosen by solving a POCMDP computed from the ART such that the maximum damage an attacker can gain in the future is minimized. The proposed approach can be used locally or on a global basis using global ARTs. As traditional attack trees, ARTs can grow very quickly in huge and complex networks. Additionally, the knowledge to build up an appropriate ART has to be provided by experts or system administrators.

In [177] response selection is modeled as *Partially Observable Markov Decision Process (POMDP)* that tries to maximize the reward calculated from the cost and benefit function. They mainly focus on the uncertainty during intrusion detection. Therefore, they consider the following cases to be happen during intrusion detection: correct detection, false positive, false negative, normal operation and wrong diagnosis. For those given cases different costs arise: maintenance costs, failure costs and penalty costs. They model their system using different observable system states with assigned security properties. Possible actions including the rewards and transition probabilities are needed for the proposed model. The goal is then to maximize the reward for an optimal decision process. The authors admit, that that the accuracy of the model relies on the given expert knowledge. In addition to this approach, the authors of [174] extend the strategy using POMDP (*AIR_POMDP, POMDP for Automated Intrusion Response*). Instead of using only attack related information as proposed in [177], the system state during runtime is included to select a more appropriate response policy [174]. This work was further enhanced in [175], where *Hiddem Markov Models (HMM)* are used to extend the given POMDP model to circumvent false responses generated due to false negatives of the IDS.

In [129] the Orcef framework is introduced. They map each incoming alert to an attack graph and examine possible security defense points. Available responses on those security defense points are considered as candidate responses. For theses candidate responses

the response costs are determined. They use fuzzy techniques to calculate the response cost matrix, to include a set of metrics describing the response costs. They determine the Pareto-optimal set from the candidate responses to find a tradeoff between positive and negative effects of a response. Within this proposed selection mechanism, only a single response can be selected. A combination of responses to counteract the whole security incident is not possible. Additionally, relations between responses, like conflicts and preconditions, cannot be reflected.

6.6.4 Other Approaches

Pre- or Post-Processing-based Approaches Within this category of related work, complete response plans are provided instead of selecting a single response. Those approaches rely on either pre-calculating possible response plans or on iterating over generated response plans afterwards.

The work done in [57] is improved by considering a complete response plan [55, 58]. They calculate the RORI of each available response plan and select the most appropriate one. The drawback of their work is, that all response plans have to be considered has to be available to the system. That means they have to be calculated in advance.

Kinesis [148] is a security incident response and prevention system for wireless sensor networks. They combine a policy-based system using ECA rules with a cost-sensitive approach. First possible responses are examined using the policy definition. In case of multiple candidate responses, the most cost effective one is chosen. The costs of a response are calculated based upon the *Confidence Index* and the *Impact Index*. In case multiple alerts are raised, the union of the selected responses is used. In order to consider dependencies between responses, an *Action Precedence Graph (APG)* is utilized. Conflicting actions as well as equivalent actions can be erased. The disadvantage of this approach is that the dependencies between responses are considered after the generation of the response set. This procedure does not allow to benefit from synergy effects, in case a single response covers multiple incidents, but is more costly, than another one. Additionally, in case a response is excluded from the response set, a new one has to be found afterwards.

To improve the scope of decision making to more than one goal, timing aspects between selected responses were considered by [104]. Therefore, they propose the *Intrusion Detection Alert management and Intrusion Response System (IDAM& IRS)* that selects appropriate responses based on *Hierarchical Task Network Planning (HTN)*. The administrator has to specify a certain response goal, meaning the goal of the automated intrusion response. Possible goals are for example, analyze the attack, maximize data integrity, or minimize costs. Additionally, the goal task needed for the HTN planner, consists of the intrusion scenario, a response strategy and response key points. Each response goal has a specific response strategy. Additionally, they define possible response key points, also called subtasks, like notification, record intrusion, or backup. All defined subtasks are very high-level and generic. For each response strategy, the subtasks have to be determined. This process is not further specified, but it seems that this linking has to be done manually. The main goal of the proposed approach is to determine the point in time for the execution of the responses automatically.

Decisions for Automation or Manual Intervention This category of related work focuses on deciding whether or not a security incident is handled automatically or manually by an administrator, rather than selecting responses for automated intrusion response.

In [12] the *Automatic or Administrator Response (AOAR) algorithm* is presented. Response selection is defined as a resource allocation problem. The interaction between an IRS and the attacker is modeled as a non-cooperative non-zero sum game. The proposed system decides whether or not, an alert describing a detected attack is forwarded to the administrator or not. If the alert is not forwarded to the administrator, the alert is handled by the automated IRS. They take the cost of the security incident into account, as well as the costs of possible responses. Additionally, they cope with the uncertainty of IDSes during the detection of a security incident. The proposed approach only covers the decision whether or not a security incident is handled automatically or by the administrator.

In [172] a queuing theoretic approach is presented to determine costs of active and passive responses. They take the IDS capability into account to either be strict or loose within the decisions to raise an alert. They assume that costs appear during the ongoing security incident. They propose a model how to calculate the costs of active and passive responses. Details are given in Section 6.1. They discovered under which conditions an active response should be preferred over a passive response. Therefore, they take the IDS configuration into account and consider false positive and false negative alerts. They neither consider a set of possible responses having dependencies with each other nor an attack consisting of multiple sub-attacks to be covered.

Domain Specific Approaches Approaches for particular domains or to antagonize specific attacks were presented: [76] considers requirements of relational database systems; Routegard [61] focuses on mobile ad-hoc networks and [164] introduces cost-sensitive response selection for this domain; [144] focuses on counteracting DOS attacks.

6.6.5 Summary, Comparison and Conclusion

The response selection strategies described in Subsections 6.6.2, 6.6.3 and 6.6.4 are compared against the requirements stated in Subsection 6.2.1. The related work stated in Subsection 6.6.1 is not included as the Requirement R1 requires the response selection to be cost-sensitive. The overview of the comparison is given in Table 6.10.

The related work within the field of response selection can be categorized as follows:

- Rule-based approaches (static mappings or ECA rules) [24, 29, 76, 86, 124]
- Cost-sensitive approaches [8, 57, 79, 83, 84, 110, 129, 141, 145, 146, 147, 151]
- Pre- or post-processing-based approaches [55, 58, 148]
- Decide whether or not to respond in an automated manner [12, 172]

Rule-based approaches simply map a single attack classification to a single response. Basic cost-sensitive approaches try to find the most effective response for a single attack classification. More advanced cost-sensitive approaches go beyond the basic selection strategies by combining them with other methods, like graphs or game-theory but they do not overcome the issue that only a single response is selected. Those classes do not consider a holistic response plan for multiple security incident. Pre- or post-processing-based approaches evaluate existing response plans or try to find a suitable response plan by iterating over the response plan multiple times. In case a huge number of entities in the network or a huge number of responses, predetermining response plans is not suitable. Instead of recalculating a non-working response plan, e.g. in case of conflicts, the proposed approach uses all available knowledge in advance to directly calculating the response plan.

The last category does not provide responses, but only the decision whether or not an automated response should be executed.

Tab. 6.10: Comparison of Related Work Based on the Requirements Stated in Subsection 6.2.1

Approach	R1 – Cost-sensitive	R2 – Comprehensive View	R3 – Conflict-free	RF4 – Preparation Actions	R5 – Synergy Effect	RF6 – Flexible Assessment
RORI-based [24, 57, 58]	✓	✗	✗	✗	✗	✗
Cost Balancing [141, 147]	✓	✗	✗	✗	✗	✗
Response Configuration [151]	✓	✓	✗	✗	✗	✓
ARB [8]	✓	✓	✗	✗	✗	✗
REASSESS [110]	✓	✗	✗	✗	✗	✗
Dependency Graph [83, 84]	✓	✗	✗	✗	✗	✓
Pre-Conditions [79]	✓	✗	✗	✗	✗	✓
RRE [181]	✓	✗	✗	✗	✗	✓
POMDP [174, 175, 177]	✓	✗	✗	✗	✗	✓
Orcef [129]	✓	✗	✗	✗	✗	✗
Kinesis [148]	✓	✗	✓	✗	✗	✓
IDAM& IRS [104]	✗	✗	✓	✓	✗	✗
Resource Allocation Problem [12]	✓	✗	✗	✗	✗	✓
Queuing Theory[172]	✓	✗	✗	✗	✗	✓

Summing up the related work, we can come to the conclusion that most response selection strategies only select a single response for a security incident instead of providing a holistic and structured response plan. Solutions that provide such response plans rely on pre- or post-processing that is not suitable for a huge number of entities within the network or a huge number of possible responses.

6.7 Publication Reference

Parts of the content of this chapter are already published on FPS (The 9th International Symposium on Foundations & Practice of Security) 2016 (cf. [63]). The own contribution was to provide the underlying set-based model and the basic idea of using LP as selection mechanism and the implementation as well as the execution of the evaluation and the corresponding framework. The implementation of the used heuristics and the linkage to the CPLEX solver are also part of the own contribution. Using pre- and postconditions, introducing weights as well as embedding the response selection mechanisms into the overall IHS are not part of the publication.

7. RESPONSE EXECUTION AND PREPARATION

Response execution and preparation describes the task of combining the selected responses to a response plan that is deployed and executed on the target system afterwards. This includes the following subtasks:

- Identify executing entities.
- Identify preliminary conditions for each response to be executed.
- Identify the correct order of responses and their preparation actions to be executed.
- Deploy needed responses to appropriate executing entities if needed.
- Execute selected responses and their preparation actions using the generated response plan on the target system.
- Ensure that the entire response plan is executed correctly.

In the following the *GNUnet Parallel Largescale Management Tool (GPLMT)* [162] is presented as the execution framework to deploy and execute responses on the target system. The identified requirements for response execution as well as the proposed system design are examined in Section 7.1. As a major part the design, the response plan definition language is explained in more detail in Section 7.2. This includes the description of the language's elements and the automated generation of response plans build upon this language definition. The implementation of GPLMT is described in Section 7.3 and gives a detailed insight into the realization of GPLMT. Within our evaluation in Section 7.4, we compare our approach to the requirements stated in Subsection 7.1.1. The related work is examined and compared against the requirements in Section 7.5. As this chapter is mainly based on GPLMT that is already published (cf. [162]), a publication reference stating the own contribution is given in Section 7.6.

7.1 Analysis and System Design

Within this section the requirements for response execution are described in Subsection 7.1.1. Those requirements reflect the needed features of this module and the structure of the response plan. In Subsection 7.1.2 the system design following the previously identified requirements is examined.

7.1.1 Requirements

In order to fulfill the subtasks of response execution the following requirements have to be fulfilled. Additionally, those requirements arise from Requirement RF7 and Requirement RF8 of the overall system such that the overall Incident Handling System (IHS) is retroactive as recommended in [130]. In case a requirement was specified in literature, this is stated in the description of the requirement.

Requirement R1 – Sequential Execution: The *sequential* execution of responses has to be supported (cf. [130]). As some responses may have preconditions, reflected as additional actions to be taken, the definition of the execution order of determined responses and their preconditions is essential. Part of this requirement is to keep all required steps in the response plan synchronized.

Requirement R2 – Parallel Execution: The *parallel* execution of responses has to be supported (cf. [130]). As responding to a detected security incident is time critical, responses should be parallelized as much as possible. Executing suitable responses in parallel can save time and further damage is prevented as the attack is stopped earlier.

Requirement R3 – Error Handling : Reacting to *failure situations* during the execution of the response plan is essential. In case the target system falls into an inconsistent or insecure state, some options have to be available to return the target system into a safe state. This means that after detecting a failure during the execution of the response plan fall-back and cleanup actions need to be triggered.

Requirement R4 – Modularity: Responses and executing entities have to be *specified independently* in order to ensure reusability of responses and to separate response definition from the underlying infrastructure definition. Both have to be combinable in a simplistic manner in order to ensure confirmability. This requirement is tightly coupled with an adequate addressing scheme suggested by [87] such that executing entities are well defined and controllable.

Requirement R5 – Flexible Parameterization: To ensure reusablility and flexibility within the response description, responses needs to be parameterizable in a flexible manner. This is also recommended by the authors in [87].

Requirement R6 – Flexible Deployment: In order to execute responses, the appropriate implementation has to be available on the executing entity. This can be done in advance by deploying all responses to all available entities. Consequently, changing a response implementation will result in redeploying the changed response on all entities leading to a high management effort. Therefore, a flexible way of deploying responses on-demand is needed. An automated deployment is recommended in [110].

7.1.2 System Design

The designed architecture of GPLMT is shown in Figure 7.1. The main parts of this architecture are the *central controller* and the maintained *data*, namely *targets, tasklists,* and *plans*. Both are described in the following.

GPLMT is designed as a stand-alone tool running on the so-called *GPLMT controller*. The central GPLMT controller is responsible for orchestrating the whole execution of the response plan, i.e. scheduling responses, so-called *tasks*, on the executing entity of the target system. Executing entities are called *nodes* in the following. GPLMT manages a connection from the controller to each single node. GPLMT does not require any original services on the nodes, but relies on a communication channel available during the execution of the response plan like Secure Shell (SSH) and possibly other protocols in the future, to connect to the node and control the execution of the response plan.

A response plan is given in GPLMT's own high-level description language that is based on the `Extensible Markup Language (XML)` (see Section 7.2). This response plan description has to be generated based on previously selected responses such that it can be passed to the GPLMT controller. To generate the response plans, the executing entities have to be specified as *targets* and responses have to be specified as *tasklists*. Separating this information has the advantage that responses can be defined independently from the

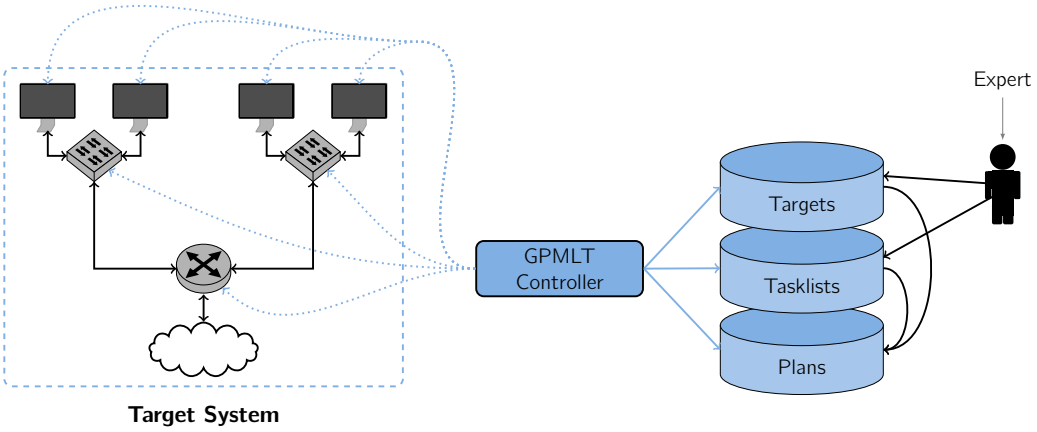


Fig. 7.1: GPLMT's Architecture – Overview

node and the underlying target system they are executed on (cf. Requirement R4). Both, the response and target definition, has to be done by the system administrator having knowledge about the target system. As the implementation of a response is centrally stored on the GPLMT controller as tasklist, changing and adding responses can be done with low management effort and allows flexible deployment (cf. Requirement R6). Build upon this information the response plan for a concrete security incident can be generated and executed afterwards.

First, the response implementations, given as tasklists, can be deployed to the executing nodes (targets). Afterwards, the controller schedules blockwhise parallel tasks until a synchronization barrier appears within the description that force the execution to wait until previous tasks are finished. After overcoming this barrier subsequent parallel tasks can be executed. Such a block of tasklists is called *execution block*.

Within the description *teardowns* can be specified that are scheduled after the whole response plan is finished. Those teardowns can be used for clean-up or reporting tasks. A teardown definition can be done within an arbitrary location of an execution block, as those tasks are appended to the scheduler at first place. The response plan is finished when all execution blocks are executed and all teardowns are finished.

In large infrastructures with many nodes, GPLMT will open a large number of connections. SSH in particularly is resource-intense. The SSH connection setup is computationally expensive due to cryptography and may overload a low-powered controller or the physical server hosting a lot of Virtual Machines (VM). A high rate of connection attempts may stress Intrusion Detection Systems (IDS), and may trigger IDS alerts for alleged SSH scanning.

GPLMT offers two solutions to limit its resource usage: *connection re-use* and *rate limiting* of connection attempts. GPLMT will tunnel all commands to the same node through a single control connection, but will still try to reconnect when the connection is lost. GPLMT optionally delays connection attempts, including reconnects, to not exceed a configurable number of attempts per interval. This resource limitation option is especially useful, if the target system is under attack and resources are already working to capacity.

7.2 Response Plan Description Language

The *response plan* in GPLMT is given as `XML` compound of three main parts.

Targets define the executing entity (see Subsection 7.2.1).

Tasklists define the response to be executed (see Subsection 7.2.2).

Steps define the *response plan* combining targets and tasks in the correct control flow (see Subsection 7.2.3).

The remaining XML description is given in Listing 7.1. The specified elements, namely *targets* (Line 4), *tasklists* (Line 5), and *steps* (Line 6), are mandatory for each response plan specification. The targets and tasklists can be included (see Line 3) into the response plan and have not to be defined for each response plan. Therefore, the definition of the file and the prefix is mandatory. Within the steps definition, included targets and tasklists can be accessed using the prefix and the name of the appropriate object.

Listings 7.1: Basic Structure of the Response Plan with GPLMT

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <experiment>
3    <include file="..." prefix="..." />
4    <targets> ... </targets>
5    <tasklists> ... </tasklists>
6    <steps> ... </steps>
7 </experiment>
```

7.2.1 Targets

A `target` element names a member node, and specifies how to access the node using node specific configurations. The target defines the executing entity of a response and can be derived from the infrastructure information of the target system. This may be the victim of an attack, in case a self-healing response is executed, a network element responsible for different network operations like a router capable of executing network-wide responses, or a dedicated execution element within the target system.

The following types of targets are currently supported:

`local` specifies executions on the GPLMT controller itself.

`ssh` states, that the nodes can be accessed using SSH. The child elements `username` and `password` may provide credentials. Authentication is ensured through pre-distributed public keys without pass phrase.

`planetlab` specifies a PlanetLab node and accepts the `PlanetLab-API-URL`, the `slice`, and the `user name` as attributes. This supports experimentation setups, GPLMT can be used for, additionally.

`group` specifies a nested target definition, creating a set of nodes (and other groups) addressable as a single target. This ensures a higher reusability of target definitions.

To support parameterization per target, each target definition can contain multiple `export-env` elements, which declare an environment variable to be exported. The value

of this variable is then available to all tasklists executed on the target. In case a response needs parameters those can be made available to the executing entity using the export functionality. A typical example of parameters used for responses may be IP-addresses or host names. This kind of parameterization is equivalent to a *global variable* that can not change during the execution of the response plan. *Local variables* have to be specified within the steps definition (see Subsection 7.2.3).

In Listing 7.2 an example for the local target (Line 3), an *ssh target* (Line 6) and a *group target* (Line 10) is given. The local target utilizes the `export-env` functionality to set the global variable `var` to the value `value-of-var` for the whole execution of the response plan. The *ssh target* does not require a password definition, as long as corresponding keys are deployed on the node. Using the *group target* can be done by referencing already existing targets, as shown in the example. Alternatively, the group definition can contain the specification of new targets that can also be referenced in later group definitions or includes.

Listings 7.2: Example for Local, SSH and Group Target.

```
1  ...
2     <targets>
3        <target name="my-local-node" type="local">
4           <export-env var="value-of-var" />
5        </target>
6        <target name="my-ssh-node" type="ssh">
7           <user>exampleuser</user>
8           <host>node1.example.com</host>
9        </target>
10       <target name="my-group-node" type="group">
11          <target ref="my-local" type="local" />
12          <target ref="my-ssh-node" />
13          <target name="second-ssh-node" type="ssh">
14             <user>my-user</user>
15             <host>node2.example.com</host>
16          </target>
17       </target>
18    </targets>
19 ...
```

7.2.2 Tasklists

The `tasklist` element binds a list of *tasks* to a name. The `tasklist` element is used to represent the concrete response implementation that can be used. For example, a response to block certain ports may have different implementations with respect to the underlying operating system, e.g. different tasklists for blocking ports with OpenVSwitch or IP-tables may exists, or depending on the execution target, e.g. shutting down a service will also result in a blocked port.

A `task` element contains one or more of the following predefined commands:

`get` and `put` are used to exchange files between the GPLMT controller and the targets. Those elements are used for the deployment of responses as needed implementations can be transferred from the controller maintaining the tasklists representing

responses to the executing entities. Additionally, they can be used to collect monitoring results from the nodes.

run accepts an arbitrary command to be executed. The commands to be executed have to match the underlying platform they are executed on. When a target defines additional environment variables, those are passed to the command using the `export-env` element.

par and seq elements contain nested lists of tasks. `seq` will run those tasks in sequence, whereas `par` will immediately start all tasks in parallel. This allows a fine-grained definition of the execution control flow of the response plan and the responses itself. The first sub-element of a `tasklist` has to be either `par` or `seq`.

call is used to reference a tasklist to be executed. A tasklist can be predefined in the tasklist section of the current response plan or is included using a separate file. This allows to better re-use existing tasklists and prevents duplicate code.

The `tasklist` element accepts the optional attributes `cleanup`, `timeout`, and `error`, controlling the tasklist's behavior in case of an error condition. `cleanup` references another tasklist to be executed after the current tasklist, even if the current tasklist aborts due to an error. This can be used to kill zombie processes and delete temporary files or to save intermediate results. `timeout` specifies the maximum amount of time the tasklist is allowed to execute before it is aborted. This guarantees progress in case a command loops infinitely or dead-locks. `on-error` determines how GPLMT continues when a task fails. The following failure modes are available:

abort-tasklist aborts the current *tasklist* and continues with the tasklist specified by the surrounding context or the next tasklist that is scheduled.

abort-step aborts the current *step* and continues with the next step. Steps are explained in Subsection 7.2.3.

panic aborts the whole execution of the response plan.

Those failure modes allow to react on errors occurring due to the runtime of a response plan. As the error correction depends on a concrete implementation of a response those mechanisms are realized within the tasklist definition.

The parameterization of a single tasklist is done by setting appropriate environment variables on the executing entity. This can be done in different ways.

- *Global variables* can be set by using the `export-env` element within the `target` element (see Subsection 7.2.1) and are available during the whole execution of the response plan.
- *Local variables* can be set by using the `export-env` element within the `step` or `repeat` element (see Subsection 7.2.3) and are available for the step they are defined for. Local variables overwrite global variables.

This allows a flexible parameterization of all taskslist representing a dedicated response to be executed and fulfills Requirement R5.

In Listing 7.3 an example tasklist is given. Each tasklist definition has to start either with the `par` or `seq` definition to explicitly define whether the execution is parallel or sequential. First, the script executing the response is loaded to the target (see Line 5).

Afterwards, the script is executed on the target using the `run` command (Line 9). Instead of writing a script executing the response, a tasklist can be used in case the response can be implemented with available system commands utilized by the `run` command.

Listings 7.3: Example Tasklist Utilizing `put` and `run` Commands.

```
1  ...
2     <tasklists>
3        <tasklist name="my-tasklist1">
4           <seq>
5              <put>
6                 <source>my-response-file.sh</source>
7                 <destination>response/my-response-file.sh↩
                     </destination>
8              </put>
9              <run>.response/my-response-file.sh</run>
10          </seq>
11       <tasklist>
12    </tasklists>
13 ...
```

For each response available to the system a tasklist has to be specified that can be executed on an appropriate target. As tasklists and targets are specified independently, and needed scripts or files can be transferred to the target, a tasklist has not to be adapted to every single target, but can be executed on every suitable target the controller can connect to and has the appropriate access permissions. This allows an easy deployment and execution of responses across the network without preparing each single network entity and equip them with different response implementations that may are needed in the future. Code needed to execute a certain response can be updated and modified on the GPLMT controller and can then be distributed to the target executing a certain response on demand.

7.2.3 Steps

The language requires exactly one `steps` element. It may contain multiple `step`, `synchronize`, `register-teardown`, and `repeat` elements.

The `step` element determines which tasklists run on which target. A `start time` and a `stop time` element can be added to schedule a task for later execution. Times are either relative to the start of the execution of the whole response plan or absolute wall clock times, allowing to defer a step until night-time when resources are available. Thus, `step` elements form the basic building block for orchestrating the response plan. Additionally, a step element can contain the `export-env` element as defined for targets. This is used to define local variables for each step.

Consecutive `step` elements run in parallel. A `synchronize` element represents a synchronization barrier and execution can only continue after currently running steps are finished. The `synchronize` element can further be specified by targets to wait for.

The `register-teardown` element references a tasklist by name that is executed when `steps` finishes. This tasklist is always executed, even if errors lead to the abortion of the response plan. The registered tasklist is intended to contain cleanup tasks and to transfer logging or monitoring data to the controller. The teardown only needs to be registered within the same execution block the `step` that allocates the corresponding resources is issued.

GPLMT's response plan definition language offers basic loops within the `steps` element: The `repeat` element loops over the enclosed steps until one of the following conditions is satisfied:

- A given number of iterations (`iterations`) are executed.
- A given amount of time has passed (`during`).
- A given point in time has passed (`until`).
- A given list of environment variables (`listing`) is iterated.

These are deliberately simple conditions that only allow for decidable loops, so it can be easily verified by manual inspection (or programmatically) whether a loop terminates. The loops do not implicitly wait after the first round of execution. This means, all steps in the loop are executed in parallel if no `synchronize` element is used at the end of the loop or directly after the `step` definition. Using a `synchronize` element at the end of the loop will enforce that a new round in the loop is started after all steps within the execution block before the synchronization are finished. This allows a higher flexibility in generating different kind of loops and a more flexible control flow.

In Listing 7.4 an example definition for steps is given. The tasklists to be executed can be bound to a single target (see Line 3) or to multiple targets (see Line 7). In the first execution block from the start of the response plan to the first `synchronize` command a teardown is registered in Line 5. This `teardown` is executed at the end of the response plan and can be used for clean up actions needed for the whole response plan. The synchronize command can be used as barrier for all targets included in the response plan as shown in Line 8 or for a subset of targets (see Line 6). The first `synchronize` command forces the system to wait until `node1` has finished its execution, but `node2` is not influenced. That means the execution of `node2` may not be finished when the second execution block is entered.

Listings 7.4: Example Steps Definition Utilizing `synchronize` and `teardown` Commands.

```
1  ...
2    <steps>
3       <step tasklist="tasklist1" targets="node1" />
4       <step tasklist="tasklist2" targets="node2" />
5          <register-teardown tasklist="teardown-tasklist1"↩
               targets="node1" />
6       <synchronize targets="node1" />
7       <step tasklist="tasklist3" targets="node1 node2" />
8       <synchronize />
9       <step tasklist="tasklist4" targets="node3" />
10   </steps>
11 ...
```

As the target and tasklist definition has to be provided in advance to the system and is decoupled from the beg control flow, the steps definition is the only part that has to be generated on demand per security incident. Details about the automated generation of the response plan is given in Subsection 7.2.4.

7.2.4 Automated Generation of Response Plans

In order to provide automated intrusion response a response plan adjusted to the security incident to handle the security incident is needed. The building blocks are the `targets`, the response has to be executed on, and the `tasklist`, describing the response itself. Using this information an appropriate response plan can be generated that ensures the correct sequence of execution with respect to preconditions of a response.

To generate a response plan the following two steps are required:

1. Evaluate the correct sequence of execution based on precondition relations (see Algorithm 3) by extracting the needed information from the underlying blackboard.

2. Use the previously generated dependency graph to generate the `steps` definition (see Algorithm 4) and produce a valid GPLMT description.

The correct execution order is determined in Algorithm 3 and a dependency graph is generated. Each vertex within the dependency graph has an `ID` reflecting the identifier of the tasklist to be executed. Each tasklist is identified by its name. Additionally, a vertex has an `Executor` property reflecting the identifier of the executing entity. The executing entity is identified by the name of the target. The preconditions of each tasklist, representing a response, are stored on the underlying blackboard and are reflected by the `responseispreconditionofresponse` relation between responses. This information is extracted and an internal dependency graph is generated for further processing.

Algorithm 3 Generate Dependency Graph

```
1:  global G                                                    ▷ graph of responses
2:  procedure getPreconditionsOfElem(id, v)
3:      preconditions ← getImplementationPreconditionsWithExecutor(id) ▷ returns a
    list of IDs and executors
4:      for all precondition in preconditions do
5:          if precondition.ID in G then
6:              vPre = G.getVertex(precondition.ID)
7:          else
8:              vPre = G.addVertex(precondition.ID)
9:              vPre.Executor = precondition.Executor
10:         relation = G.addEdge(vPre, v)          ▷ an edge in the graph reflects the
    precondition relation
11:         getPreconditionsOfElem(precondition.ID, vPre)                ▷ recursive call
```

Generating a valid GPLMT description is done in a second step. Therefore, a template can be used that includes the static parts of the description file, namely, the includes to all available `target` and `tasklist` definitions and empty `target` and `tasklist` elements. The generated part of GPLMT is limited to the steps definition that describes the response plan for the current handled security incident. The previously generated dependency graph is traversed to find all vertexes that do not have an incoming edge, as this means that those responses can be executed without prior responses. Those responses can be combined into an execution block that is separated by a synchronization barrier (`synchronize` statement) from the rest of the response plan. As all those vertexes are going to be deleted, this step is repeated until the dependency graph is empty. Lastly, the generated execution blocks can be added to the GPLMT description file in the `steps` element.

Algorithm 4 Generate Steps Definition

```
1: procedure createResponsePlan(G)
2:     responsePlanTemplate   ▷ template to be used with missing steps definition
3:     stepsTemplate   ▷ template to be used with missing target and task definition
4:     while G ≠ empty do
5:         vertexToDelete = []
6:         for all v in G do                                   ▷ traverse over all vertexes
7:             if v.inDegree() = 0 then                        ▷ vertex has no incoming edges
8:                 vertexToDelete.append(v)
9:                 task = v.ID
10:                target = v.Executor
11:                stepsToDo.append(stepsTemplate(task, target))      ▷ generate steps list
12:        steps.append(stepsToDo)                  ▷ append steps list to steps definition
13:        steps.append(synchronize)                          ▷ add synchronization barrier
14:        for all v in vertexToDelete do   ▷ delete vertexes including connected edges
15:            G.deleteVertex()
```

7.3 Implementation

GPLMT is the interpreter of the response plan written in Python 3. The response plan, given in `XML` format, is first validated against the schema definition given as `rng` file (Regular Language Description for XML New Generation (RELAX NG)). Therefore, the Python library `lxml` is utilized. RELAX NG is an `XML` schema language used to define `XML` documents. The schema language provided by RELAX NG is more compact than XML Schema (XSD).

If the schema validation is successful, the response plan is processed block-wise. An *execution block* is defined as all `steps` definitions above a `synchronize` statement or the end of the response plan. Each execution block is read and all contained elements are prepared to be scheduled. Tear-downs are registered before an execution blocked is executed to ensure that all tear-downs are registered before the steps they belong to.

In case an execution block contains a loop, a new *execution context* is created. This ensures that `synchronize` statements inside the loop are not influenced by `synchronize` statements outside the loop. A loop simply schedules steps inside the loop until the loop condition is reached. As an explicit `synchronize` statement is available, the implementation does not provide implicit synchronization after a loop iteration.

For each `tasklist` a *tasklist environment* is provided. The tasklist environment carries the cleanup-tasks that are executed at the end of the tasklist's execution.

For each `task` a *variable environment* is provided. The variable environment carries all environment variables that are set when a task is executed. This includes variables, that are set for a specific target using the `export-env` statement and variables used in the `listings-loop` or `step` definition. As those loops may be nested, the variable environment is composed of all variables defined until the task is finally executed. Later definitions overwrite earlier definitions. That means that in case a variable is defined as global variable and as local variable, the local definition is preferred.

Each task is executed in a single `SSH` connection. Notably, GPLMT wraps OpenSSH, so all features of OpenSSH are available via a local OpenSSH configuration file. GPLMT directly uses OpenSSH's *control master* feature to re-use connections to the same node.

To copy files from the GPLMT controller to the nodes controlled with GPLMT, `put` and `get` is implemented. Therefore, native `Secure Copy (SCP)` is utilized.

7.4 Evaluation

Our approach provides sequential execution of responses (cf. Requirement R1). Within the tasklist definition the `seq` command explicitly executes single commands within the tasklist in a sequential order. Within the steps definition single steps can be executed in sequence by utilizing the `synchronize` command.

Our approach provides parallel execution of responses (cf. Requirement R2). Within the tasklist definition the `par` command explicitly executes single commands within the tasklist in parallel. Within the steps definition single steps are executed in parallel implicitly within an execution block.

Our approach provides error handling (cf. Requirement R3). Each `tasklist` command can be used with the `on-error` declaration. If the tasklist fails, multiple options can be used for error handling. Hereby, we provide abort the tasklist, the whole step or the whole response plan. Additionally, clean-up tasklists can be specified.

Our approach decouples responses from targets the responses are executed on (cf. Requirement R4). To provide better modularity, we distinguish between target and tasklist definitions. The description of the infrastructure information is given within the `target` elements, while potential responses are reflected in the `tasklist` elements. This allows defining both independently and no hard coding of available responses to concrete executing entities is needed.

Our approach provides parameterization of responses (cf. Requirement R5). We support local as well as global variables for steps and tasklists. Global variables are specified within the `target` element. Local variables are specified within the `step` element.

Our approach provides a flexible deployment of responses (cf. Requirement R6). This can be achieved by providing a central response repository stored on the GPLMT controller. Responses, e.g. represented as scripts, can be deployed on the nodes within the target system using the `put` command. If a response is not needed on a node anymore, it can be erased. Changes within the response repository can be done centrally and new responses can be deployed on the nodes.

7.5 Related Work

In this section related approaches with respect to response execution are examined. Concrete approaches within the domain of Intrusion Response Systems (IRS) are limited, as most often only a high-level definition of the execution itself is given (see [29, 37, 38, 39, 51, 71, 113, 115, 166, 168]). Most often, the proposed systems do not specify a specific response plan language or give insights in the structure of the response plan. Related work in the field of IRSes is further examined in Subsection 7.5.1. Another field of interest are frameworks or solutions focusing on the automated configuration of networks or network components. Related work within this field is covered in Subsection 7.5.2. Additionally, structured execution of experiments within different testbed solutions is covered in Subsection 7.5.3 as an experiment requires a structured execution as response plans do. The proposed related work is compared against the requirements listed in Subsection 7.1.1 in the last subsection (see Subsection 7.5.4).

7.5.1 Response Execution in IRSes

In [87] the problem of initiating responses within a distributed system is addressed. The authors state the need of an appropriate communication protocol to distribute a response to different locations in the network. Their focus in this paper are mobile ad-hoc networks (MANET). They propose the `Intrusion Response Message Exchange Format (IRMEF)` extending `IDMEF` to spread messages of responses to execute within the system. This format contains amongst others the time the response has to be executed and the targets that have to apply these response. A locally installed agent will receive this message. In case it's address is listed in the targets definition it will execute the response at the specified time. The proposed communication protocol between those agents and a console is the Simple Network Management Protocol (SNMP) in Version 3. This scheme does not allow to schedule more actions in an appropriate manner. For example no sequential or otherwise timed actions or the interconnection of multiple actions are possible. The specification of the control flow in more detail is not possible.

In [55, 58] they provide a selection mechanism for existing response plans. Within their work they do not specify the response plan itself, nor give a definition language for a structured response plan. They execution part is completely left aside within this work.

Some related work within the field of IRSes only provide a single response to be executed, such that no coordination between multiple responses is needed, e.g. see [24, 123, 124]. As they cannot cover a security incident effecting multiple network entities and consisting of multiple attacks, they are no suitable IHS.

7.5.2 Network Management and Configuration Solutions

Responses that are executed on the target system may be reconfigurations of hosts or network elements. Within this subsection we examine possibilities how a network can be configured automatically. Therefore, we first examine tools that are used for configuration management and orchestration. Afterwards, we examine dedicated protocols that can be used for network configuration. The goal of this analysis is to find suitable foundations for the response execution module.

Ansible [1] is an open source IT configuration, deployment and management tool. It is designed to be minimal and does not rely on an agent or client that is installed on the target machine that is controlled. Machines are managed by default using SSH or Windows Remote Management (WinRM) as they exist natively on those platforms. To manage the configuration of components, so-called *playbooks* are utilized. The playbooks are written in YAML and define parts that are automated. A playbook is consisting of mulitple *plays* defining the automation for a set of hosts, called *inventory*. Each play consist of multiple *tasks* targeting multiple hosts in the inventory.

Within a playbook, tasks are executed in order, and within a play all hosts will execute the same directives. If the execution fails, the playbook has to be adapted and needs to be rerun. If blocks are used, error handling strategies can be applied. Ansible provides multiple different loops that can be used within a playbook. Ansible's main purpose is to provide a reconfiguration of machines, other responses have to be included by writing own modules for Ansible.

Puppet [2] relies on a pull model. For each machine that has to be controlled with Puppet an agent on the host is required to be installed. The agent pulls configuration files from a central server and determines changes between the host's state and the

[1] https://www.ansible.com/
[2] https://puppet.com/

configuration specified. The Puppet language is declaring *resources* that can be organized in *classes*. A resources is typically a single file or package, while a class describes a service or application. *Node classifications* are used to apply the classes to and group the *nodes*. As the configuration is rolled out to the clients, a synchronization between the clients can only be done by regulating the polling interval of the clients.

In addition to the configuration management tools, standardized network protocols for management purposes are available, e.g. *Simple Network Management Protocol (SNMP)* and *Network Configuration Protocol (NETCONF)*.

SNMP [22] is mainly used for extracting information about network components, like routing or device information. Therefore, an extensible information model named *Management Information Base (MIB)* is used. Available information is structured hierarchically and can be extended by vendors, whereas some standard MIBs are defined that have to be offered at least. Each managed device has to run a SNMP client. For reconfiguration of specified MIBs, the `SET` command can be used. To use SNMP as a response execution module, the surrounding environment has to be implemented. A response execution module can use SNMP as a configuration possibility, but SNMP itself does not provide any built-in functionality for describing the control flow of an execution.

Due to the limitations of SNMP regarding the configuration capabilities of components, NETCONF [45] has been introduced. NETCONF operates on top of Remote Procedure Call (RPC) and uses XML for encoding of data and messages. NETCONF consist of four layers, the *content layer* holds configuration data, the *operations layer* defines who to read and write this data, the *message layer* provides RPC and the *secure transport layer* provides security and reliability. NETCONF can be used to reconfigure single devices and network elements like routers. It does not provide a framework for synchronized execution of configurations, but can be integrated, as well as SNMP, into the proposed response execution module in order to implement tasklists.

7.5.3 Control Flow Definition for Experiments

Various different tools exist to manage and control network experiments. A rather extensive list can be found on the PlanetLab website [3]. [70] provides a comprehensive analysis with respect to quality and usability of such tools, finding most of them not usable or suitable to be used with respect to today's network experiments. Many of these tools are outdated and not available anymore (Plush, Nebula, Plman, AppManager) or were not even made publicly available at all (PLACS). Some of these tools provide rather basic functionality to invoke commands on remote nodes (pssh, pshell, vxargs) not supporting error conditions and error handling as well as orchestrating nodes to perform complex and synchronized operations. In the following we examine different testbed experiment tools and frameworks in order to investigate possibilities if those systems can be used as response execution module.

The *Stork project* [4] provides a deployment tool for PlanetLab nodes including configuration. It is recommended to install Stork on the nodes that are controlled. For response execution within an existing network, agent software might not always be possible, as routers or switches may be reconfigured as well. Additionally, policies within the network may forbid the installation of agent software. Stork is mainly used for package management and distribution on a large scale of nodes. This tool lacks fine-grained execution control to setup more complex execution flows that are needed in order to be useful for

[3] https://www.planet-lab.org/tools
[4] http://www.cs.arizona.edu/stork/

response execution.

Gush (GENI User Shell) [1] claims to be an execution management framework for the GENI testbed. Gush provides extensive methods to define resources but is limited regarding control flow aspects. Parallel or sequential execution is not possible in a straight forward manner. Gush is mainly used to define the underlying testbed itself. The control flow definitions to execute complex execution flows are left to the user as one scripts have to be written. In addition, Gush is not longer supported [5].

Experimentation frameworks like *NEPI* [6] [114] provides a scripting interface to known testbeds, like OMF, PlanetLab or NS-3. Additionally, Linux hosts and OpenFlow can be included. The main focus of this framework is the interoperability of those resources. NEPI requires the user to do rather complex adaptations in the source code to extend it with new functionalities and add support for new platforms. An experiment or a response plan is a user-defined script, where commands can be executed on resources or monitoring can be activated. To structure the control flow pure Python has to be used. A simplistic automated generation of response plans is not possible.

Approaches like *OMF* [117] focus on the management and operation of network testbed infrastructures and federation between infrastructures but do not focus on the experiment execution itself. The provide a OMF Experiment Description Language (OEDL) to specify experiments. This language allows to specify resources within the testbed, properties, applications, groups and own scripts. They do not provide an easy way of defining the control flow, but using events than can be used to manage the control flow.

The *COCOMA framework* [7] [116] focuses on providing an experimentation framework for cloud based services to control and execute tests for cloud-based services in a controlled and reproducible manner and to study resource consumption of such services. Within this framework, the testbed description itself is tightly coupled to the control flow of the experiment. The framework provides mechanisms to create, monitor and control contentious and malicious system workload.

In [54] the authors propose an emulated testbed for the domain of cyber-physical systems. This work focuses more on the testbed implementation and less on the execution of experiments.

7.5.4 Summary, Comparison and Conclusion

The requirement stated in Subsection 7.1.1 are mapped to the presented approaches above. The results are summarized in the following Table 7.1.

The response execution capability of the related work in the field of IRSes is limited. Most of them only describe a generic and high-level behavior of executing responses. Synchronization between multiple responses or a structured response plan are not covered within this related work.

[5] http://gush.cs.williams.edu/trac/gush
[6] http://nepi.inria.fr/
[7] http://www.bonfire-project.eu/services/cocoma

Tab. 7.1: Comparison of Related Work Based on the Requirements Stated in Subsection 7.1.1

Approach	R1 – Sequential Execution	R2 – Parallel Execution	R3 – Error Handling	R4 – Modularity	R4 – Flexible Parameterization	R5 – Flexible Deployment
IRMEF [87]	✗	✗	✗	✓	✓	✗
Ansible	✓	✗	✓	✓	✗	✓
Puppet	✗	✗	✓	✓	✗	✓
SNMP	✗	✗	✗	✓	✗	✗
NETCONF	✗	✗	✗	✓	✗	✗
Stork	✗	✗	✗	✓	✗	✓
Gush [1]	✗	✗	✗	✓	✗	✗
Nepi [114]	✓	✓	✓	✓	✗	✗
OMF [117]	✗	✗	✗	✓	✓	✗
COCOMA [116]	✗	✗	✗	✗	✗	✗

Protocols or frameworks for configuration, deployment and network management tasks provide a solid basis for automated intrusion response. They can be integrated into a response execution module, e.g. implementing a dedicated tasklist within our approach. As out-of-the-box solutions they are not well suited, because they do not provide enough flexibility to describe the desired control flow of the response plan. Those approaches can be used to directly trigger reconfigurations on single machines or a huge number of machines treated in the same manner. Nevertheless, tools like Ansible and Puppet are most close to be useful as response execution module.

Within the domain of testbed experiment control, none of the examined solutions provide a structured and flexible response plan. Most often, the execution of an experiment is tightly coupled with the definition of the testbed infrastructure itself. This makes it not possible to just use the experiment definition as response plan and deploy it on the target system. Additionally, the definition of the control flow is limited and left to the user by writing own scripts or code.

7.6 Publication Reference

GPLMT is already published on PAM (Passive and Active Measurement Conference) 2016 in the context of executing experiments on various testbeds (cf. [162]). Within this work the own contribution encompasses fundamental ideas of the design of the description language for response plans and implementation enhancements within the GPLMT framework. Embedding GPLMT into the context of incident handling was not part of the publication as GPLMT was used for experimentation control flow management.

8. IMPLEMENTATION AND EVALUATION

In this chapter we cover the implementation of our proposed Incident Handling System (IHS) and evaluate the overall system that is proposed. In Section 8.1 we describe the implementation of the overall IHS. This includes the blackboard implementation, as well as the implementation of our information model from Section 3.3. All implemented modules are discussed and linked to previously described module implementations given in Chapter 4 to 7. The evaluation given in Section 8.2, covers the mapping to the requirements an IHS has to fulfill as discussed in Section 3.1, an analysis of an appropriate blackboard, and a closer look on the intrusion response capabilities of the system. Additionally, we provide considerations on security aspects as well as an evaluation of our use case described in Subsection 4.1.3 and 5.2.3. As parts of this system are already published in [62] in the last section a publication reference is given to determine the own contributions to this system (see Section 8.3).

8.1 Implementation

As a proof of concept and basis for our evaluation, we provide a prototype implementation in Python 3, available on GitHub[1]. Our implementation requires an underlying database system as blackboard to hold and persist information. The backend implementation is examined in Subsection 8.1.1. This encompasses utilized databases including the needed communication from and to the database. A description of how the information model of our system described in Section 3.3 is implemented is given in Subsection 8.1.2. This includes interfacing with the underlying databases and the setup process. The main components of the proposed IHS structured into interfaces, modules and controllers are examined in more detail in Subsection 8.1.3. This includes a description how previously examined modules are integrated.

8.1.1 Backend Implementation

First, we present the supported databases, namely *PostgreSQL* and *OrientDB* in Subsubsection 8.1.1.1. Afterwards, we examine the main important point when using a database as blackboard, namely the notification mechanisms to provide notifications on changes on the database, in Subsubsection 8.1.1.2.

8.1.1.1 Supported Databases

We support two different types of databases, namely object-relational and graph-based databases. As the structure of the blackboard is graph-based, a graph database is naturally highly suited. We chose OrientDB[2] as it provides hooks and triggers. OrientDB supports object-oriented concepts by providing classes for vertexes and edges that can be extended

[1] https://github.com/Egomania/BlackboardIDRS
[2] http://orientdb.com/orientdb/

for the own usage using a schema definition. The query language used for OrientDB is a dialect of the Structured Query Language (SQL) that provides some additions to work with graphs but mainly focus on the standard. OrientDB claims to provide Atomicity, Consistency, Isolation and Durability (ACID) properties. OrientDB can be used as distributed database and provides linear scalability. Additionally, OrientDB provides complex data types, like `DATE` and `DATETIME`. To interface with OrientDB, pyorient[3] is used.

As an alternative, we implement our blackboard based on PostgreSQL[4], a widely used object-relational database. PostgreSQL is compliant to the ANSI-SQL:2008 standard and supports full subqueries, read-committed and transaction level isolation. Data integrity features like primary and foreign keys, cascading updates or deletions, check and unique constraints are supported. PostgreSQL is full ACID compliant. To interface with PostgreSQL, psycopg2[5] is used.

8.1.1.2 Notification Mechanism

The main important point from the backend point of view is to provide a channel that can be used for notifications of changes on the database content that represent the blackboard. Information about the change as well as the change itself has to be propagated to the controller or the modules of the IHS. A pull-based mechanism, where all modules regularly query for information changes, is not suitable as additional load on the database would be produced and the query window would allow an attacker to misuse this blind spot. To enable change notification from the blackboard, the underlying database has to support a notification or trigger mechanism.

OrientDB For OrientDB a *native Java Hook* is utilized triggering after a new node or edge is added, updated or deleted. OrientDB provides different built in methods: `onRecordAfterUpdate`, `onRecordAfterCreate` and `onRecordAfterDelete`. Those methods are combined within the provided hook. OrientDB calls those methods with an `ODocument` value containing the original as well as the new values of the node or edge. This information together with the operation (update, insert, or delete) and the ID of the node or edge are sent to the controller.

To reduce the amount of information pushed from the database to the controller, this hook is configurable to decide which nodes and edges are needed. Within the OrientDB server configuration the needed tables can be specified and changes are available after restarting the server.

The database sends the change information on updates, inserts and deletes using a `POST` method to push the information to the controller. As OrientDB does not support after-commit hooks directly, this feature has to be implemented within the modules. The modules have to actively query the database to wait for the commit.

PostgreSQL In case of PostgreSQL, the built-in `NOTIFY` command is utilized. This command sends a notification including a payload to all clients previously executed the `LISTEN` command. Hereby, the interprocess communication is possible for all processes accessing the same database and the same channel. Access restrictions on those channels are not possible. `NOTIFY` can be coupled with transactions, such that a notification is only send after a commit.

[3] https://github.com/mogui/pyorient
[4] https://www.postgresql.org/
[5] http://initd.org/psycopg/

An *after-commit hook* triggers a function preparing the information to be sent over the notification channel. For each specified table three different triggers for updating, deletion and insertion are created using the PostgreSQL `CREATE TRIGGER AFTER` command. Those triggers call for each row that has been changed the `table_update_notify` function. This after-commit hook can be generated on startup of the IHS in contrast to the OrientDB Hook, that has to be updated manually.

Within the `table_update_notify` function the information is prepared using built-in functionality to generate `json` files at `pg_notify` call. PostgreSQL provides variables to extract the latest operation that triggered the function. Additionally, the original and new fields of each row can be extracted using PostgreSQL build-in functionality. After preparing all information including old and new data, the ID of the changed node or edge and the operation it self, the payload can be send over the appropriate channel. The controller on the other side only needs a connection to the database to listen on the channel the notifications are sent to.

8.1.2 Information Model Representation and Setup

First, we introduce how the information model from Section 3.3 is mapped to the implementation in Subsubsection 8.1.2.1. Afterwards, we examine how to load infrastructure information in Subsubsection 8.1.2.2 and policy information in Subsubsection 8.1.2.3. The infrastructure configuration is decoupled from the policy configuration such that both can be exchanged independently.

8.1.2.1 Representing the Information Model

To represent our information model Python classes are utilized. Each information element (node) and relation (edge) is represented as a class. Those classes are translated automatically to the database scheme for both database systems. This includes setting up tables respectively classes, constraints if needed, creating an index on marked fields and the deletion of unneeded elements. Listing 8.1 shows the structure of those classes.

Listings 8.1: Node Class Example – `Service` Node

```
1  class service(node):
2     cluster_id = None
3     mapper = {'name': 'STRING'}
4     __slots__ = list(mapper.keys())
5     index = ['name']
6     def __init__ (self, name=None, rid=None, client=False, ↩
         batch=False):
7        if not client:
8           client = self.client
9        node.__init__(self, rid)
10       if self.rid == None:
11          self.createOrGet(client, batch)
12       else:
13          self.getByRid(client, batch)
```

In Line 1 the inheritance of each class is indicated. The `node` class provides the following functions:

getByRid: Queries the table corresponding to the class name and the given identifier and returns the corresponding instance with all attributes set in the database.

createOrGet: Returns an instance with the given identifier based on the available attributes. If no applicable database entry exists, a new entry is created.

update: Updates all values given in the instance into the underlying database.

The `cluster_id` in Line 1 is relevant for OrientDB, as this value determines the place where the records are stored. This value is set during startup. In Line 3 the attributes of the class are defined. The attributes are named and the data type is specified. If an attribute is defined with a leading underscore (_) this attribute is not used for searching an entry in the database using the `createOrGet` method. Line 4 suppresses that attributes can be added dynamically and space for all specified objects is reserved. This allows listing the attributes within an instance of the object. In Line 5 a list of dedicated fields, an index has to be created on, can be specified. The identifier (`RID`) of each instance is provided with an index automatically. The `node` class is equipped with a separate connection to the database. If no other connection is specified, this connection is used (see Line 8). To create or get a database entry from an instance, the `createOrGet` method (see Line 11) or the `getByRid` (see Line 13) method is used dependent on the delivered function arguments.

The relations between nodes are defined in the `edge` class. An example is shown in Listing 8.2. Additionally, to the `node` class the foreign keys have to be specified for PostgreSQL (see Line 5).

Listings 8.2: Edge Class Example – `serviceusesip` Edge

```
1  class serviceusesip(edge):
2      cluster_id = None
3      mapper = {'name': 'STRING', 'port': 'INTEGER'}
4      __slots__ = list(mapper.keys())
5      psql = {'fromnode': 'service', 'tonode': 'ip'}
6
7      def __init__ (self, fromNode, toNode, port, client=False, ↩
           batch=False):
8          if not client:
9              client = self.client
10         edge.__init__(self, fromNode, toNode)
11         self.port = port
12         self.createOrGet(client, batch)
```

8.1.2.2 Infrastructure Configuration

The infrastructure configuration provides the system with the needed infrastructure information on startup. Continuously monitoring and scanning the underlying target system may lead to changes on the blackboard such that the original infrastructure configuration gets outdated. The infrastructure configuration is given in `JavaScript Object Notation (JSON)` and contains the following elements: `Template` nodes, `Server` nodes, `Network` nodes, `Device` nodes, and `Service` nodes. Each element to specify is given as list including the needed attributes. An example how to specify a single device

is shown in Listing 8.3. Each device needs to be specified with a name and a corresponding template (see Lines 4 to 5). Additionally, a list of interfaces can be specified (see Lines 6 to 11). This specification includes the `l2 Network` and `l3 Network` nodes. The `l3 Network` definition has to correspond with specified networks in the corresponding section.

Listings 8.3: Infrastructure Configuration – `Device` Node

```
1 ...
2 {"device":
3       {
4       "name": "r1",
5       "template" : "router",
6       "interfaces": [{
7             "interface":
8                   {"order": 1, "l2": "r1_eth0", "mac": "↩
                       1E.00.00.00.00.01", "l3": "productive↩
                       ", "ip": "192.10.0.1"}},{
9             "interface":
10                  {"order": 2, "l2": "r1_eth1", "mac": "↩
                       2E.00.00.00.00.01", "l3": "backbone",↩
                        "ip": "192.20.0.1"}}
11      ]
12      }
13 },
14 ...
```

If an infrastructure configuration is given, the whole configuration file is read on startup of the IHS and is transfered to the underlying database. Hereby, it is configurable if the current configuration is deleted before the new configuration is added or not.

Additionally, a stub for GPLMT target definitions can be generated optionally from the infrastructure configuration. This includes setting up the targets description file and list all nodes with the given names and host descriptions. An automated key deployment is not included within this generation.

8.1.2.3 Policy Configuration

The policy configuration provides available responses, as well as their mapping to possible attacks using their consequences. `Attack` and `Consequence` nodes are simply a list of known values, whereas attacks provide a mapping to the consequences that can occur in case they are successful. `Attack` nodes specify all known attacks the system can cope with and are used for the implicit normalization of incoming alert classifications. A `Response` node is defined as follows, (see Listing 8.4).

Each response is identified uniquely by its name (see Line 4). Each response can be used on a certain group of targets in the target system (see Line 5). The mapping between responses and potential consequences the response can mitigate is done in Line 6. A list of implementations (see Lines 9 to 17) defines the device the response is available on.

If a policy configuration is given, the whole configuration file is read on startup of the IHS and is stored in the underlying database. Hereby, it is configurable if the current configuration is deleted before the new configuration is added or the new configuration extends the existing one.

Listings 8.4: Policy Configuration – `Response` Node

```
1 ...
2 {"response":
3       {
4       "name": "RateLimiting",
5       "target": ["host", "service"],
6       "responsemitigatesconsequences": ["ad"],
7       "preconditions": [],
8       "conflicts": []
9       "implementations":[{
10            "implementation": {
11                  "name": "simpleRateLimiter",
12                  "deployedOn": ["h1"],
13                  "executor": "r1",
14                  "metrics": {"cost": 0.5}
15            }}
16      ]
17      }
18 },
19 ...
```

Additionally, a stub for GPLMT tasklists definitions can be generated optionally from the policy configuration. This includes setting up the taskslist description file and list all responses with the given names.

8.1.3 Interfaces, Modules and Controller

The main part of the implementation is divided into

Interfaces Interfaces are used to bind external monitoring system, like Intrusion Detection Systems (IDS) or monitoring systems, gathering infrastructure information and alerts (see Subsubsection 8.1.3.1).

Modules Modules represent the knowledge sources mapping the functionality of all tasks within the steps of the incident handling process (see Subsubsection 8.1.3.2).

Controller The controllers manage the access from an to the underlying database representing the blackboard (see Subsubsection 8.1.3.3).

8.1.3.1 Interfaces

To interface with the system, a *basic insert* is provided. This Python class maps incoming alerts to the underlying database. Therefore, the `Alert` class is defined, consisting of the `message ID`, `source`, `target`, `classification` and `timestamp` information that can be extracted from `IDMEF` messages. To be compliant with the *Intrusion Detection Message Exchange Format (IDMEF)* standard, `users` and `services` can be included as they can occur within the target definition of an `IDMEF` message.

For each database that is supported, an `insert` function has to be defined for the basic insert. This function expects an instance of the `Alert` class and an open database connection. This function creates the new alert entry according to the information stored in the `Alert` class. Additionally, an alert context is created or re-used with matching

source, target and classification definition. The created `Alert` node is linked to the `Alert Context` node. In case a new alert context has to be created, the `Alert Context` node is linked to the source, target and attack elements.

Additionally, a *simulator interface* and a *REST API* interface are implemented utilizing the basic insert. The simulator reads a number of given `XML` files containing `IDMEF` messages. Those messages are parsed, missing attacks, sources and targets are inserted into the database, and for each alert an instance of the `Alert` class is created. Those instances are handed over to the basic insert one after another.

The simulator observes all messages that are passed from the controller to the modules. Therefore, the simulator registers itself to all available queues. The simulator stops the experiment when no longer messages occur. Relevant statistics are saved and the next file is processed.

The *REST API interface* starts a web server and listens on `PUT Requests` containing `IDMEF` messages. Those messages are parsed, the instances of the `Alert` class are created and handed over to the basic insert.

8.1.3.2 Modules

After successfully deploying the schema, the modules are started within the main process of the IHS. Each module runs in a separate process to support multiprocessing. This allows the distribution of single modules across the network.

As a monitoring or IDS module, we use our anomaly-based IDS presented in Chapter 4. Our IDS can interface with the REST API described in Subsection 8.1.3.1.

Our implementation includes for both databases an aggregator, a correlator, and two priorisation modules for alert processing. As this is only a proof-of-concept implementation, the functionality of these modules is limited. The aggregator fuses alerts to an `Alert Context` node based on equal sources, targets or attack classifications. The correlator supports checking for attack paths by searching for attacks launched by sources that were previously a target of another attack. The first priorisation module randomly allocates a priority to an `Alert` node. The second propagates priority information through connected `Alert Context` nodes.

For intrusion response, we integrate the following components in form of separate modules:

Response Identification Module This module implements the response identification algorithms as discussed in Chapter 5.

Response Selection Module This module implements the response selection task as discussed in Chapter 6. Hereby, we provide a module that extracts the required information from the database, passes this information to a solver, generating and solving the problem, and transfers back the solver's output to the underlying blackboard. The information gathered from the blackboard is: the affected entities like `Device`, `User` or `Service` nodes and not yet selected `Implementation` nodes connected to the current `Bundle` node including the executor of the response implementation, conflicting responses and preconditions, as well as metrics and damage information.

Response Execution Module This module implements the response execution task discussed in Chapter 7. Hereby, we provide a module extracting required information to generate the descriptions for GPLMT. The generated GNUnet Parallel Largescale Management Tool (GPLMT) file is stored and can be executed by an administrator.

Additionally, we implemented a response evaluation module. This module is triggered by a `Bundle` node with a `Ready` attribute set to `TRUE`. The response evaluation module collects all `Implementation` nodes that are going to be executed and extracts their `Metric` nodes. After everything is prepared, the response evaluation module sets the `Prepared` attribute to `TRUE` and waits for the response execution module to set the `Executing` attribute to `TRUE`. This will trigger the response execution module to automatically trigger the generated response plan. For simulation reasons, the response execution module simply waits a predefined time span instead of actually executing the generated GPLMT scripts and the response evaluation module calculates randomly distributed values between 0 and 1 to update the `Metric` nodes. The response evaluation module sets the `Executed` attribute of the edge between the `Bundle` and the `Implementation` nodes to `TRUE`.

8.1.3.3 Controller

To receive notifications about changes on the blackboard, two different controllers – one for each database type – are implemented. Those controllers wait for incoming notifications from the database and notify modules subscribed for the particular information. For this purpose, Python's built-in queues are utilized. All modules are triggered to start their task by receiving notifications from the controller.

The subscription of the modules is done on startup of the system. For each module and the information element, the module listens to, a separate multiprocessing queue is initialized. The controller receives a dictionary including the information element as key and a list of queues a notification for this element has to be sent to. Initializing a queue for each module ensures that each module will receive the notification and no information gets lost.

8.2 Evaluation

In this section the overall IHS is evaluated. First, we evaluate that the proposed IHS fulfills all functional and non-functional requirements for an IHS stated in Section 3.1 (see Subsection 8.2.1). Afterwards, we provide a qualitative analysis of the access behavior of the modules of the IHS in order to show that those modules can act conflict-free on the proposed information model in Subsection 8.2.2. In Subsection 8.2.3, we evaluate which database system is a suitable blackboard for our IHS. As the main focus of this thesis are intrusion response capabilities we evaluate the interaction of the related modules supplementary in Subsection 8.2.4. The use case explained in Subsection 4.1.3 and 5.2.3 is evaluated in more detail in Subsection 8.2.5. In Subsection 8.2.6, we describe a thread analysis of the system including possible attack vectors on the proposed IHS.

8.2.1 Requirement Alignment

The proposed IHS supports multiple IDSes (cf. RF1). Each IDS generating alerts using the IDMEF can be used directly with the proposed IHS (see Subsubsection 8.1.3.1). IDSes providing other formats to exchange information can be integrated by providing an additional interface to proposed IHS (see Subsection 3.4.1).

The proposed IHS supports alert processing (cf. RF2). The needed information elements for alert processing are provided within our information model (see Section 3.3).

Additionally, we provide implementations for the following alert processing tasks: prioritization, aggregation, normalization and correlation (see Subsection 3.4.2 and Subsubsection 8.1.3.2). To support more comprehensive alert processing tasks, additional modules can be implemented and added directly to the proposed IHS.

The proposed IHS provides flexible triggers for intrusion response (cf. RF3). In Subsection 5.2.2 we provide possible triggers for automated intrusion response that are derived from our information model explained in Section 3.3. They can be used as standalone triggers or in combination and provide high flexibility to adapt to different use cases.

The proposed IHS supports response identification (cf. RF4). The response identification module is described in Chapter 5. Multiple responses are examined and can be implemented using GPLMT tasklist definitions (see Subsection 7.2.2) to be available for the proposed IHS. Additionally, attacks and consequences are combined with possible responses for the given use case of this thesis (see Subsection 5.2.3).

The proposed IHS is cost-sensitive (cf. RF5). The response selection proposed for the IHS relies on assessing the costs of a response as well as the costs of the potential damage a security incident causes (see Chapter 6). The proposed approach balances those costs instead of using simplistic mappings between attacks and responses.

The proposed IHS selects the optimal set of responses to counteract the security incident (cf. RF6). The response selection module is implemented using Mixed Integer Linear Programming (MILP) and optimizes the set of candidate responses with respect to different use case specific metrics (see Section 6.3). The solution is, therefore, optimal in terms of the chosen metrics.

The proposed IHS provides response plans for coordinated response execution (cf. RF7). In Chapter 7 GPLMT as response execution framework is introduced. GPLMT's definition language provides coordinated execution by specifying the control flow of the response plan (see Section 7.2). GPLMT allows to specify parallel and sequential execution to reflect pre- and postconditions of responses. Additionally, error behavior can be specified.

The proposed IHS provides response execution capabilities (cf. RF8). As the response execution module is implemented using GPLMT the specified responses can be executed directly and in a coordinated manner. The GPLMT controller, therefore, connects to the executing network entities and executes the defined responses in form of tasklists (see Subsection 7.1.2).

The proposed IHS is completely automated (cf. RNF1). All modules of the proposed IHS are completely automated. The response execution module generates the GPLMT script that can be used for automated intrusion response. At this point human interaction is possible, but not required as the script can be executed automatically.

The proposed IHS is adaptable (cf. RNF2). Due to continuous execution of the incident handling process the response plan is adapted and regenerated if the security incident could not be solved with the given response plan (cf. RNF5).

The proposed IHS provides a comprehensive view (cf. RNF3). Instead of solving single intrusions determined by alerts, the proposed IHS enables alert processing to find the comprehensive security incident (see Subsection 3.2.3). The response identification module combines `Alert Context` nodes that can be covered together as they target the same network entities (see Subsection 5.2.1). The response selection module optimizes the set of candidate responses with respect to the security incident as a whole and takes synergy effects into account (see Section 6.3).

The proposed IHS is generic (cf. RNF4). The system design is not specific to a certain use case. The single modules can be used independently from the given instances of the information element that are available to the system. In order to use the proposed IHS

for a specific use case the system has to be configured appropriately by expert knowledge. This expert knowledge includes the description of the infrastructure and the policy to be used (see Subsection 8.1.2).

The proposed IHS provides continuous incident handling (cf. RNF5) in a just-in-time manner. The response identification module continuously adds new `Alert Context` nodes to the `Bundle` node used for intrusion response (see Subsection 5.2.1). This includes adding new `Implementation` nodes used as candidate responses. The response selection component only includes targets that belong to an unsolved `Alert Context` node within the bundle (see Subsection 8.1.3). The response evaluation module ensures, that only `Alert Context` nodes are marked as solved that are targeted by a response implementation (see Subsection 8.1.3). The interaction of those modules ensures continuous incident handling.

The proposed IHS provides active responses to counteract a detected security incident while it is on-going (cf. RNF6). As soon as a security incident can be identified, the proposed IHS will be triggered to start automated intrusion response. The proposed responses are active instead of passive and are not restricted to recovery related responses (see Subsection 5.2.3).

The proposed IHS is collaborative (cf. RNF7). The proposed IHS is based on the Blackboard Pattern (see Section 3.2). The needed functionality of the incident handling process is split up into single modules fulfilling the incident handling in cooperation by fulfilling their own subtask. The interaction of those modules allows to implement a complex and comprehensive tasks like incident handling.

The proposed IHS is designed to be modular (cf. RNF8). The functionality of the incident handling process is split into single tasks (see Section 3.4). For each single tasks different modules can be implemented, exchanged, and used as long as they follow the given information model examined in Section 3.3.

8.2.2 Qualitative Analysis

The access behavior of the modules is most essential for the Blackboard Pattern as data consistency has to be ensured. The blackboard has been designed in a way that most of the writing behavior is additive and only in some cases updating existing information elements is necessary. This approach facilitates the access control and reduces the amount of potential states where inconsistencies can appear.

Infrastructure nodes are updated or inserted by corresponding modules responsible for scanning and generating the infrastructure data. In case of static data (e.g. `IP`, `Device` or `Interface` nodes) nodes and relations are updated infrequently. Consequently, the access to those nodes is rare. Additionally, the modules are separated such that only one module will update information, while others will only read information.

The same applies to `Attack`, `Response` and `Implementation` nodes. The nodes' information is only updated in case of a policy refinement; otherwise their information is only read as input for other modules.

In case of `Alert` and `Alert Context` nodes we designed an additive behavior for updates or inserts. If a new alert is raised by an IDS, a new `Alert` node is inserted. Aggregation and correlation only add `Alert Context` nodes or relations between `Alert` and `Alert Context` nodes respectively. The prioritization module updates (re-prioritization) or adds initial priority information. As only a single module is allowed to change this information and others are only permitted to read, no inconsistencies can occur.

One crucial action on the blackboard is the deletion of outdated information. This operation is done on a large amount of nodes and relations. In case an alert processing module wants to access this data during deletion, inconsistencies may occur. The decision of deleting information is done by a separate module – the garbage collector. This allows defining fine-grained strategies for removing information. Additionally, this garbage collector can store outdated information on long-term storage.

8.2.3 Blackboard Analysis

Within this quantitative analysis, we investigate a suitable database that can be used as a blackboard for our proposed IHS. First, the test setup is introduced in Subsubsection 8.2.3.1. In order to evaluate our approach different datasets representing typical but challenging use cases are identified (see Subsubsection 8.2.3.2). Subsubsection 8.2.3.3 finally shows the results of the evaluation for different test cases.

8.2.3.1 Evaluation Methodology

The usefulness of alert processing has already been proven [156] with common datasets. Hence we provide a simulation to evaluate our system under typical use cases providing a challenge to the approach. For this purpose, we generate different datasets containing `IDMEF` messages with specific and challenging properties (see Subsubsection 8.2.3.1). Needed infrastructure information is provided by preprocessing those datasets.

Those datasets are read by our *simulator module* transferring the single alerts one by one into the underlying database. After insertion, the simulator waits until all modules finished their processing and stores relevant statistics. All implemented modules are executing their specific tasks as describes previously. Each dataset was generated in different variants with sizes from 1000 to 5000 alerts using a step size of 500. As the measurements are stable (see Subsubsection 8.2.3.3), each test was done 5 times.

All variants of each dataset were evaluated under the following three different test cases.

Test Case 1 (t_1): Insert Nodes – For this test case the alerts are pushed into the blackboard. This includes adding an `Alert` was well as an `Alert Context` node if needed. Additionally, relations between the `Alert Context` node and the `IP` and `Attack` nodes are inserted. As first only the insertion behavior of our system is tested, no further analysis steps are performed.

Test Case 2 (t_2): Prioritize Nodes – This test case includes the prioritization modules, to investigate the update behavior of our system. Alerts are added as describes above, but beyond alert and context information is prioritized. Within this test case the update behavior of our system is evaluated.

Test Case 3 (t_3): Combining Nodes – For this test case all implemented alert processing modules are activated to investigate the analysis behavior of our IHS. The information is pushed into the blackboard as described above and all alert processing modules are working simultaneously on the data. Within this test case a huge amount of information is requested from the blackboard in order to perform aggregation and correlation.

We executed the evaluation using a workstation equipped with an Intel Xeon E3-1275 quad core CPU with active Hyper-Threading running at 3.5 GHz and 16 GB of RAM and

a SSD. The operating system was Ubuntu 15.10 64-Bit with Python 3.4.3, pyorient 1.5.2 and psycopg2 2.6.1.

8.2.3.2 Generated Datasets

To describe the properties of our generated datasets, we use the following notation. The number of unique `Alert Context` nodes (*#unique*) describes the number of `Alert Context` (*#context*) nodes not having a context as superset. The number of possibilities to aggregate same source ($same_s$), target ($same_t$) and attack classification ($same_c$) and the number of attack paths (*paths*) are given. For each dataset a database is created storing used attacks and IP-addresses. The generated datasets are available on GitHub[6].

DOS Dataset The DOS dataset simulates a distributed Denial of Service (DOS) attack with a huge number of sources, i.e., attackers, to a small number of targets, i.e., victims. The most challenging aspect of this dataset is to build the structure of resulting `Alert Context` nodes pointing to a huge number of sources. This results in many relations between the nodes. We used two different attacks, namely, a port scan as preparation attack and a DOS attack against the target system resulting in two `Alert Context` nodes for an aggregation of attack classifications ($same_c = 2$) for all variants of this datasets. For this dataset the following with respect to same target and attack path applies to all variants: $same_t = 18$ and $paths = 0$. 2018 IP-addresses are stored in the database. In the following Table 8.1 not yet described properties of the dataset are listed.

Tab. 8.1: Properties of the DOS Dataset

Setting	D1	D2	D3	D4	D5	D6	D7	D8	D9
#alerts	1000	1500	2000	2500	3000	3500	4000	4500	5000
#context	1198	1834	2497	3175	3828	4455	5089	5694	6273
#unique	207	351	534	717	895	1046	1196	1328	1435
$\#same_s$	187	331	514	697	875	1026	1176	1308	1415

Flooding Dataset Within this dataset, a small number of sources execute three different attacks on a small number of targets. The attacks are port scan, DOS and buffer-overflow, so $same_c = 3$. The challenge of this dataset is frequent use of the same properties, namely source, target and attack classification. This occurs when multiple IDSes detect the same attack at once or a single IDS reports the intrusion multiple times.

This dataset did not contain any attack path, so $paths = 0$. 35 IP-addresses are stored in the database. For this dataset the following with respect to same target and source applies to all variants: $same_t = 32$ and $same_s = 3$. Consequently, as the $same_c$ is also constant for all variants, the number of unique context nodes is 38 for all dataset variants. In the following Table 8.2 not yet described properties of the dataset are listed.

Attack Path Dataset This dataset simulates a security incident spreading across the network. First, a small number of sources attacks a huge number of targets to compromises them. The compromised entities finally launch additional attacks. The challenge of this dataset is identifying a spreading intrusion. To detect this attack huge portions

[6] https://github.com/Egomania/BlackboardIDRS

Tab. 8.2: Properties of the Flooding Dataset

Setting	D1	D2	D3	D4	D5	D6	D7	D8	D9
#alerts	1000	1500	2000	2500	3000	3500	4000	4500	5000
#context	315	325	326	326	326	326	326	326	326

of the database have to be investigated by the correlator. The following Table 8.3 lists the dataset's properties. The launched attacks are port scans and buffer overflows, while buffer overflows will result in the take-over of the target that is than the source of a new attack, so $same_c = 2$. The initial number of attackers is three, so $paths = 3$. 2003 IP-addresses are stored in the database.

Tab. 8.3: Properties of the Attack Path Dataset

Setting	D1	D2	D3	D4	D5	D6	D7	D8	D9
#alerts	1000	1500	2000	2500	3000	3500	4000	4500	5000
#context	1378	2124	2883	3727	4516	5271	5901	5894	5797
#unique	379	626	883	1228	1516	1773	1965	1978	1939
$\#same_s$	260	372	489	634	745	844	985	990	963
$\#same_t$	114	249	389	589	766	924	975	983	971

8.2.3.3 Evaluation Results

Within the evaluation different aspects are of interest. First, we will have a look on the stability of our measurements by examining the standard deviation of our measurements. Second, we will examine the alert per second rate of our test cases for all datasets. Lastly, we will examine the new nodes per second rate as every dataset differs in terms of new `Alert` and `Alert Context` nodes to be inserted.

Measurement Stability In Table 8.4 the average standard deviation of all measurements for both backends given in seconds (s) is shown. The table denotes the dataset [7] as well as the test case (t_1 to t_3).

Tab. 8.4: Average Standard Deviation

Dataset	d_{t_1}	d_{t_2}	d_{t_3}	ap_{t_1}	ap_{t_2}	ap_{t_3}	f_{t_1}	f_{t_2}	f_{t_3}
PostgreSQL	0.15	0.34	4.11	0.09	0.45	3.82	0.05	0.05	0.09
OrientDB	1.02	1.48	15.79	1.06	1.59	205.53	0.74	0.99	0.84

The table shows that generally PostgreSQL is more stable than OrientDB, as the PostgreSQL measurements show small deviations. With raising computational effort of the test cases, the average standard deviation is increasing.

[7] dos = d, attack path = ap, flooding = f

Alerts per Second Rate The number of alerts that can be processed per second (alert/s) is shown in Table 8.5. Hereby, the minimum (*min*), maximum (*max*) and average (*avg*) values for PostgreSQL (p) and OrientDB (o) are given. The table denotes the dataset [7] as well as the test case (t_1 to t_3).

Tab. 8.5: Alerts per Second Rate (alert/s) for PostgreSQL p and OrientDB o

Dataset	p_{min}	p_{max}	p_{avg}	o_{min}	o_{max}	o_{avg}
d_{t_1}	287.09	354.72	320.75	11.4	19.72	14.73
d_{t_2}	228.61	307.27	257.8	8.4	16.24	11.55
d_{t_3}	64.97	125.44	86.15	1.37	6.75	3.12
ap_{t_1}	299.4	355.76	324.76	12.5	19.35	15.13
ap_{t_2}	230.36	287.86	250.71	8.91	16.23	11.62
ap_{t_3}	30.80	85.12	49.59	0.51	3.01	1.1
f_{t_1}	370.32	396.63	384.58	37.88	50.87	44.77
f_{t_2}	318.1	330.31	325.04	15.4	35.29	23.38
f_{t_3}	281.78	293.31	287.73	14.13	18.00	16.97

First, it can be observed that the number of alerts that can be processed per second (alert/s) is decreasing with a higher number of activated modules. While insertion is quite fast using PostgreSQL, prioritizing nodes is noticeable. More comprehensive analysis like aggregation and correlation leads to drop down the throughput.

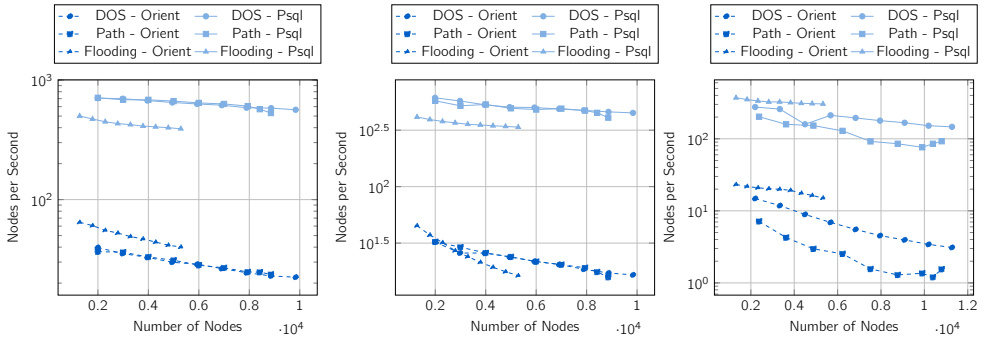
Secondly, it is obvious that the processing rates between OrientDB and PostgreSQL significantly differ. While PostgreSQL is still able to process alerts within an adequate time, OrientDB is not suitable for this use case. We assume, that's because OrientDB is actively queried to wait for the committed results, as OrientDB only provides an after-create hook in contrast to PostgreSQL. Additionally, for OrientDB the notifications are sent via `Http Post` while for PostgreSQL inbound channels can be used.

Additionally, it is shown that the processing rates differ between the datasets. While the Flooding dataset has a good throughput, the Path and DOS dataset seam to cause performance issues. That's because of their structure leading to a higher number of `Alert Context` nodes to be insert.

For Test Case t_1 the number of alerts processed per second (alert/s) slightly drops for the all dataset. For Test Case t_2 and t_3 the number of alerts processed per second slightly drops for the Flooding dataset while the rate drops significantly with a rising number of alerts for the other two datasets.

New Nodes per Second Rate As the datasets results in a different number of new `Alert Context` nodes that have to be added, we investigate the rate of new nodes added to the database per second (node/s). Hereby, new nodes are the sum of `Alert` and `Alert Context` nodes to be added. The results for this part of the evaluation are shown in Figure 8.1a. Whereat, the x-axis show the number of new nodes and the y-axis the evaluated rate in log scale.

This representation shows again, that OrientDB cannot combat with PostgreSQL within our application scenario as blackboard for our IHS. Additionally, the drop down of the throughput with a rising number of alerts is clearly visible through all datasets and test cases. This representation shows that the number of new nodes within the Flooding dataset is significantly lower than for the DOS and Path dataset, as those are structured such that a high number of `Alert Context` nodes are produced.



(a) Test Case t_1 – Insertion

(b) Test Case t_2 – Updating

(c) Test Case t_3 – Reading

Fig. 8.1: Evaluation of Different Datasets and Test Cases Using PostgreSQL and OrientDB

Comparing Figure 8.1a and Figure 8.1b a drop down of the throughput is noticeable. The prioritization of alerts produces therefore, a visible impact on the system. In both figures the Flooding dataset seams to perform worse than the DOS and Path dataset. This is because most of the alerts to insert are unique in terms of source, target and attack classification. This results in a low number of additionally added `Alert Context` nodes, but the checks for uniqueness is the same effort as in the other two datasets.

Comparing Figure 8.1b and Figure 8.1c another throughput drop is noticeable. The additionally added modules produce the main effort within the system. The Flooding dataset is now performing better than the other two datasets as the checks for new `Alert Context` nodes take more time in the DOS and Path dataset as, for example the number of IPs to check is higher in those two datasets.

Summary and Conclusion This evaluation shows that the PostgreSQL database is more suitable as a blackboard for our IHS than the OrientDB database. All needed operations – insertion of information elements, updates, and querying information from the blackboard – can be done more efficiently using a common relational database instead of the graph database. For further investigations on intrusion response capabilities we will focus on the PostgreSQL as this database promises better performance results.

8.2.4 Intrusion Response Capabilities Analysis

Based on our results from Subsection 8.2.3, we further investigate in this subsection the intrusion response capabilities of our holistic IHS based on the PotstgreSQL database as blackboard. Therefore, we first introduce our evaluation methodology covering the test cases in Subsubsection 8.2.4.1. For this part of the evaluation, an infrastructure, as well as a policy configuration are used as described in Subsubsection 8.1.2.2 and 8.1.2.3. Both are examined and described in more detail in Subsubsection 8.2.4.2. Based on those configurations, different datasets are generated that are explained in Subsection 8.2.4.3. The results of the evaluation are presented in Subsubsection 8.2.4.4 covering the stability of the measurements and the performance outcomes.

8.2.4.1 Evaluation Methodology

As in the previous section, datasets in different variants containing `IDMEF` messages are generated and processed by the simulator module. All variants of each dataset were evaluated under the following different test cases, whereas more and more subsequent modules of intrusion response are activated.

Test Case 1 (t_1): Alert Processing Capabilities – This test case equals Test Case 3 (t_3) in Subsection 8.2.3. The following alert processing modules are acitvated within this test case: the aggregation module and both prioritization modules. As we do not have attacks paths within the dataset, the correlator is not used. This test case represents the baseline for comparisons with the results shown in Subsection 8.2.3.

Test Case 2 (t_2): Response Identification Module – In this test case the response identification module is activated additionally. Possible candidate responses are gained from the database and the `Issue` as well as the `Bundle` nodes are generated. The response identification module runs with a delay of 10s to wait for a stable `Alert Context` node. The trigger rule for the response identification module is set to `Alert Context` nodes of the *sameClassification* class such that only aggregated alerts are taken into account that means no `Alert Context` nodes produced by the *initial insert* are considered.

Test Case 3 (t_3): Response Selection Module – In this test case the response selection module is activated additionally. As solver for response selection the *Cheapest-First* heuristic is used because all Python-bindings for the CPLEX or GLPK solver produce unnecessary overhead during the problem generation.

Test Case 4 (t_4): Response Execution Module – In this test case the response execution module is activated additionally. This module is responsible for preparing the response plan using GPLMT scripts. Those scripts are generated and stored for later execution. The execution itself, is for simulation purposes, a time delay of 1s.

Test Case 5 (t_5): Response Evaluation Module – In this test case the response evaluation module is activated additionally. All metrics that need to be updated are assigned randomly, using Python's built-in random number generator (`random`). Those values are assigned uniformed distributed within the interval $[0, 1]$.

We executed the evaluation using a workstation equipped with an Intel Xeon E3-1275 quad core CPU with active Hyper-Threading running at 3.5 GHz and 16 GB of RAM and a SSD. The operating system was Ubuntu 15.10 64-Bit with Python 3.4.3 and psycopg2 2.6.1.

8.2.4.2 Generated Policy and Infrastructure

To evaluate the response identification capabilities, an infrastructure and a policy configuration are generated. All test cases described in this section are using those configurations. Therefore, we shortly describe the generated infrastructure and the corresponding policy. All important data is listed in Table 8.6.

Our generated infrastructure consists of 5 networks: 3 productive networks hosting 100 hosts each, a service network hosting 7 devices running 10 services and a backbone network to connect those single networks. Each network has its own dedicated router. Communication between those networks crosses the backbone network. A single IDS

Tab. 8.6: Basic Information on the Generated Infrastructure and Used Policy

Property	Value	Property	Value
Number of IPs	345	Number of MACs	345
Number of interfaces	345	Number of services	10
Number of networks	5	Number of users	100
Number of devices	312	Number of routers	4
Number of IDSes	1	Number of hosts	300
Number of service hosts	7		
Number of attacks	100	Number of consequences	300
Number of responses	1511	Number of implementations	5111

is monitoring each network, that means it has an interface to each dedicated network. Within the network 100 users are using the services and are logged on arbitrary devices.

We defined 100 different attacks known to the IHS. We defined 300 consequences, whereas an attack is mapped to a subset of those consequences. Additionally, we generated 1511 responses and 5111 implementations. Each implementation is assigned to a response, whereas a response can have multiple, but at least one implementation.

8.2.4.3 Generated Datasets

To describe the properties of our generated datasets, we use the notation described previously: The number of unique `Alert Context` nodes (*#unique*) describes the number of `Alert Context` (*#context*) nodes not having a context as superset. The number of possibilities to aggregate same source ($same_s$), target ($same_t$) and attack classification ($same_c$) are given. The purpose and generation methods for those dataset are equal to the description given in Subsubsection 8.2.3.2.

DOS Dataset We used 10 different attacks for preparation attacks and the DOS attack, resulting in 10 `Alert Context` nodes for an aggregation of attack classifications ($same_c = 10$) for all variants of this datasets. For this dataset the following with respect to same target applies to all variants: $same_t = 7$. The 7 targets of the DOS attack are the service hosts within the generated infrastructure. 1345 IP-addresses are used from the database. 345 IP-addresses are the IPs of the own infrastructure, while the DOS attack is performed by 1000 external IPs. In the following Table 8.7 the not yet described properties of the dataset are listed.

Tab. 8.7: Properties of the DOS Dataset

Setting	D1	D2	D3	D4	D5	D6	D7	D8	D9
#alerts	1000	1500	2000	2500	3000	3500	4000	4500	5000
#context	1273	1930	2586	3206	3756	4278	4799	5291	5768
#unique	275	451	621	738	817	872	926	951	973
$\#same_s$	258	434	604	721	800	855	909	934	956
#relations	8892	13452	18090	22802	27200	31524	35863	40146	44334

As only a limited number of attacks is performed within this dataset, the number of responses and implementations is limited. The number of responses to investigate because of matching consequences is limited to 155. Potentially deployed are only 1195 responses. Implementations usable because of matching consequences are 1289. Potentially, 5075 are deployed. This limits the effort the response identification module has to afford during the search for candidate responses.

During the test cases 10 `Bundle` nodes are calculated for the DOS dataset. This will result in 10 response plans given as GPLMT descriptions stored on disk. Those bundles include 370 possible responses, as they were identified by the response identification module. The number of identified responses is equal for all test cases as the number of targets is limited a low number (7) and the response identification module will compose the same alert context hierarchy for each `Bundle` node. In average 46.5 responses are selected by the response selection module. As the response evaluation module changes the metrics used for the response selection module the number of selected responses differ for each run. This indicates that using the response evaluation module, the metrics of those selected responses are updated with new values, resulting in additional work load, during response evaluation. The number of relations (*#relations*) indicates the number of edges added during processing.

Flooding Dataset For the Flooding dataset we utilized 100 attacks, so $same_c = 100$. 1045 IP-addresses are used from the database. 345 IP-addresses are the IPs of the own infrastructure, while the attacks are performed by 700 external IPs. Potentially, 300 hosts of the generated infrastructure could be a victim of an attack. In the following Table 8.8 the not yet described properties of the dataset are listed.

Tab. 8.8: Properties of the Flooding Dataset

Setting	D1	D2	D3	D4	D5	D6	D7	D8	D9
#alerts	1000	1500	2000	2500	3000	3500	4000	4500	5000
#context	1686	2340	2947	3516	4052	4567	5086	5588	6097
#unique	686	840	947	1017	1052	1067	1086	1089	1097
$\#same_s$	294	442	547	617	652	667	686	689	697
$\#same_t$	292	298	300	300	300	300	300	300	300
#identified	556	705	850	876	889	846	974	871	938
#selected	443	567	695	706	703	686	776	713	768
#relations	9549	14250	18829	23541	28109	32637	37173	41672	46195

As all available attacks are used in this dataset, the response identification module has to investigate all responses and their implementations available in the database.

During the test cases 100 `Bundle` nodes are calculated for the Flooding dataset. This will result in 100 response plans given as GPLMT descriptions stored on disk. Those bundles include possible responses (*#identified*) identified by the response identification module as listed in Table 8.8. The number of identified responses alters because of different reasons. The more alerts are included within the test runs, the more possible responses can be associated with the `Bundle` node. Additionally, the response identification module fuses possible `Alert Context` nodes with respect to the target. As this fusion is time triggered due to the waiting time for a stable `Alert Context` node to use as `Issue` the result is not deterministic and different combinations can be possible.

The average of selected values (*#selected*) per test run are shown in Table 8.8.

As the response evaluation module changes the metrics used for the response selection module the number of selected responses differ for each run. Additionally, due to the different number of identified responses the number of selected responses alters as well. This indicates that using the response evaluation module, the metrics of those selected responses are updated with new values, resulting in additional work load, during response evaluation. The number of relations (*#relations*) indicates the number of edges added during processing.

8.2.4.4 Evaluation Results

Within the evaluation different aspects are of interest. First, we will have a look on the stability of our measurements by examining the standard deviation of our measurements. Second, we will examine the execution time and the alert per second rate of our test cases for all datasets and their variants. Lastly, we will examine the new nodes and entities per second rate as every dataset differs in terms of new `Alert`, `Alert Context`, and `Bundle` nodes to be inserted.

Measurement Stability In Table 8.9 the average standard deviation of all measurements in percent (%) of the average value accurate to 2 decimal places is shown. The table denotes the dataset [8] as well as the test case (t_1 to t_5).

Tab. 8.9: Standard Deviation for all Test Cases (t_1 to t_5) and Datasets (d) [8] in Percent (%) of the Average Value

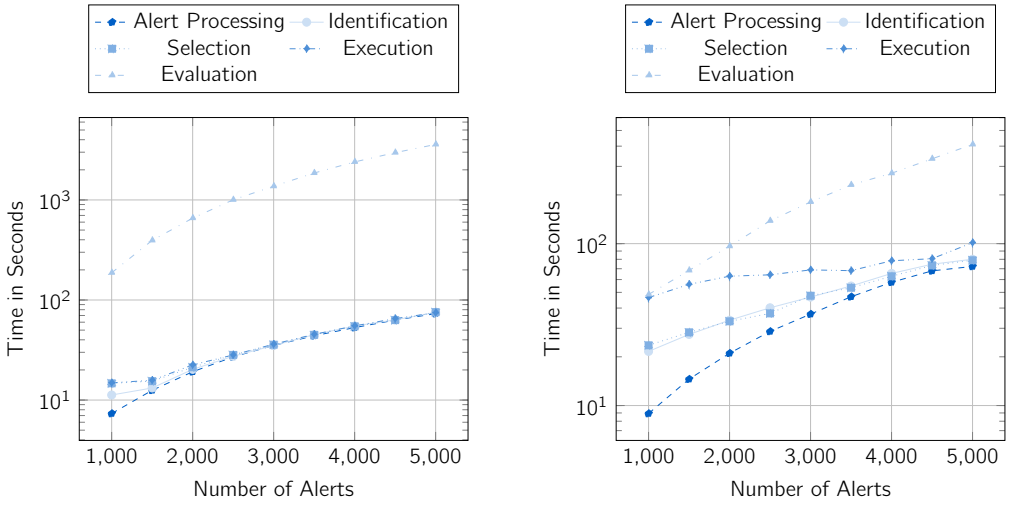
#alerts	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**	**5000**
t_1 for d_d	1.35	0.55	2.24	2.17	0.82	0.54	1.42	2.35	1.80
t_2 for d_d	0.98	2.71	3.81	2.18	2.22	3.42	1.70	0.97	2.27
t_3 for d_d	0.92	1.82	3.92	1.59	1.93	1.89	1.50	1.16	1.56
t_4 for d_d	0.55	3.36	3.28	1.12	1.99	1.53	2.06	2.44	1.00
t_5 for d_d	0.07	0.92	3.31	1.42	1.05	0.77	1.12	0.58	2.48
t_1 for d_f	0.67	2.45	1.32	2.60	1.39	3.73	3.60	2.65	0.96
t_2 for d_f	1.28	1.70	2.70	2.73	2.69	1.18	2.91	2.21	1.43
t_3 for d_f	1.96	0.80	1.22	2.28	4.43	1.65	1.52	1.15	0.98
t_4 for d_f	1.58	1.46	1.39	1.39	1.26	2.79	1.78	1.95	3.16
t_5 for d_f	1.81	0.43	1.11	1.62	0.43	1.51	3.27	2.54	1.61

All deviations are below 5 % of the average value of each test run. Only a single deviation is higher than 4 % and 10 additional deviation are above 3 %. This indicates that the measurements are stable and the results are reliable.

Execution Time First, we examine the raw execution time for both datasets. The results are shown in Figure 8.2, whereas the DOS dataset is determined in Figure 8.2a and the Flooding dataset is determined in Figure 8.2b. The x-axis shows the number of alerts within the dataset. The y-axis shows the execution time needed to process the dataset in average given in seconds (s). The y-axis is given in log-scale with basis 10.

Both figures show that the execution time increases with the number of alerts and the number of modules activated as expected. In both datasets the response identification

[8] dos = d_d, flooding = d_f



(a) DOS Dataset **(b)** Flooding Dataset

Fig. 8.2: Execution Time of Different Datasets and Test Cases

module shows a moderate impact on the execution time. The response selection module shows only a minimal impact on the execution time. As the response selection module is already evaluated in detail, those results are as expected. The evaluation of the response selection module itself shows very low execution times (see Section 6.5) compared with the execution time of the alert processing (see Subsection 8.2.3). The response execution module shows a moderate impact on the execution time, even if the GPLMT description files are stored on disk within this module.

The main impact is produced by the response evaluation module. This module has to:

- Update all metrics connected to the selected responses,
- Update all executed responses by setting the `Executed` attributed within the relation between the `Bundle` and `Implementation` node, and
- Traverse the alert context hierarchy starting from the `Issue` related to the `Bundle` node and set all connected `Alert Context` nodes to `Solved`.

Comparing the execution time within the DOS and Flooding dataset, the main impact is to traverse the alert context hierarchy. Within the DOS dataset the number of selected responses is lower than in the Flooding dataset. Therefore, updating metrics and the `Executed` attribute can not be the main influencing factor. Within the DOS dataset the alert context hierarchy differs compared to the hierarchy in the Flooding dataset. The Flooding dataset has much more `Bundle` nodes than the DOS dataset, as they are based on `Alert Context` nodes aggregating alerts with the same attack classification.

Within the DOS dataset, all alerts are combined within a smaller number of `Alert Context` nodes of the same classification resulting in more relations between this node and the `Alert Context` nodes from the basic insert. This case shows the shortcomings of a relational database in contrast to a graph-based database, as the hierarchy has to be reconstructed for each `Issue` using recursive queries. Those recursive queries are more expensive in case of the DOS dataset as more `Alert Context` nodes are included. Additionally, this shows the importance of an adequate garbage collector transferring out-dated information to a long term storage and deleting outdated information from the blackboard in order to reduce the amount of data to be analyzed and processed. A garbage

collector removing solved `Alert Context` nodes and connected information elements can reduce the amount of information needed to calculate the alert context hierarchy.

Additionally, the response evaluation module is implemented for simulations purposes only and therefore uses threads to realize parallel access on the database for different bundles to evaluate. Python's threading module does not provide true threads in terms of concurrency. Therefore, it might be that the recursive query is blocking the execution of additional threads within the response evaluation module leading to nearly sequential execution for all recursive queries that need to be executed.

Alerts per Second Rate Next, we examine the rate for alerts that can be processed. In Table 8.10 the number of alerts that can be processed per second (alert/s) are shown. The table denotes the dataset [8] as well as the test case (t_1 to t_5).

Tab. 8.10: Alert per Second for all Test Cases (t_1 to t_5) and Datasets (d) [8]

#alerts	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**	**5000**
t_1 for d_d	136	120	104	93	85	80	76	72	68
t_2 for d_d	89	114	99	91	86	78	73	71	67
t_3 for d_d	68	97	94	88	84	78	73	71	66
t_4 for d_d	67	95	89	89	83	77	73	69	67
t_5 for d_d	5	4	3	2	2	2	2	2	1
t_1 for d_f	112	103	95	87	82	74	69	66	69
t_2 for d_f	46	55	59	62	64	64	61	60	62
t_3 for d_f	42	53	60	67	63	66	64	61	63
t_4 for d_f	22	27	32	39	44	51	51	56	49
t_5 for d_f	21	22	21	18	17	15	15	13	12

The table shows that the rate drops the more modules are activated for each instance of the dataset. The main impact can be determined for the response evaluation module. This evaluation shows that for this specific module, a more efficient implementation is needed, as the rate drops to a level the system can not react quickly enough. The reasons for this rate drop are already examined in the previous paragraph.

In some cases the rate decreases for specific datasets containing more alerts. This is because additional alerts may do not cause any additional effort to the system to process those alerts. For example, alerts, that are plain duplicates of previous ones, need no complex treatment. Within the Flooding dataset, where the explained behavior can be observed in multiple instances of the dataset, multiple duplicates are contained. The duplicates will simply be added to an appropriate `Alert Context` node and no further processing within subsequent modules is required.

New Entities per Second Rate Within this evaluation we compare all our measurements based on the entities that are processed. Hereby, the following entities can be examined:

- New `Alert Context` nodes per second, this includes the `Issue` generated by the response identification module (see Figure 8.3),
- New nodes per second, this includes `Alert` nodes, `Alert Context` nodes and `Bundle` nodes (see Table 8.11), and

- New entities per second, this includes new nodes and relations between those nodes (see Table 8.11).

The relations of interest between the nodes are relations from the `Alert Context` node to targets (service, host, user, or device), `Source`, `Attack`, `Alert`, and other `Alert Context` nodes, for Test Case t_1 and, additionally, between `Bundle` and `Alert Context` and `Implementation` nodes for the other test cases. The number of relations are listed in Table 8.7 and 8.8.

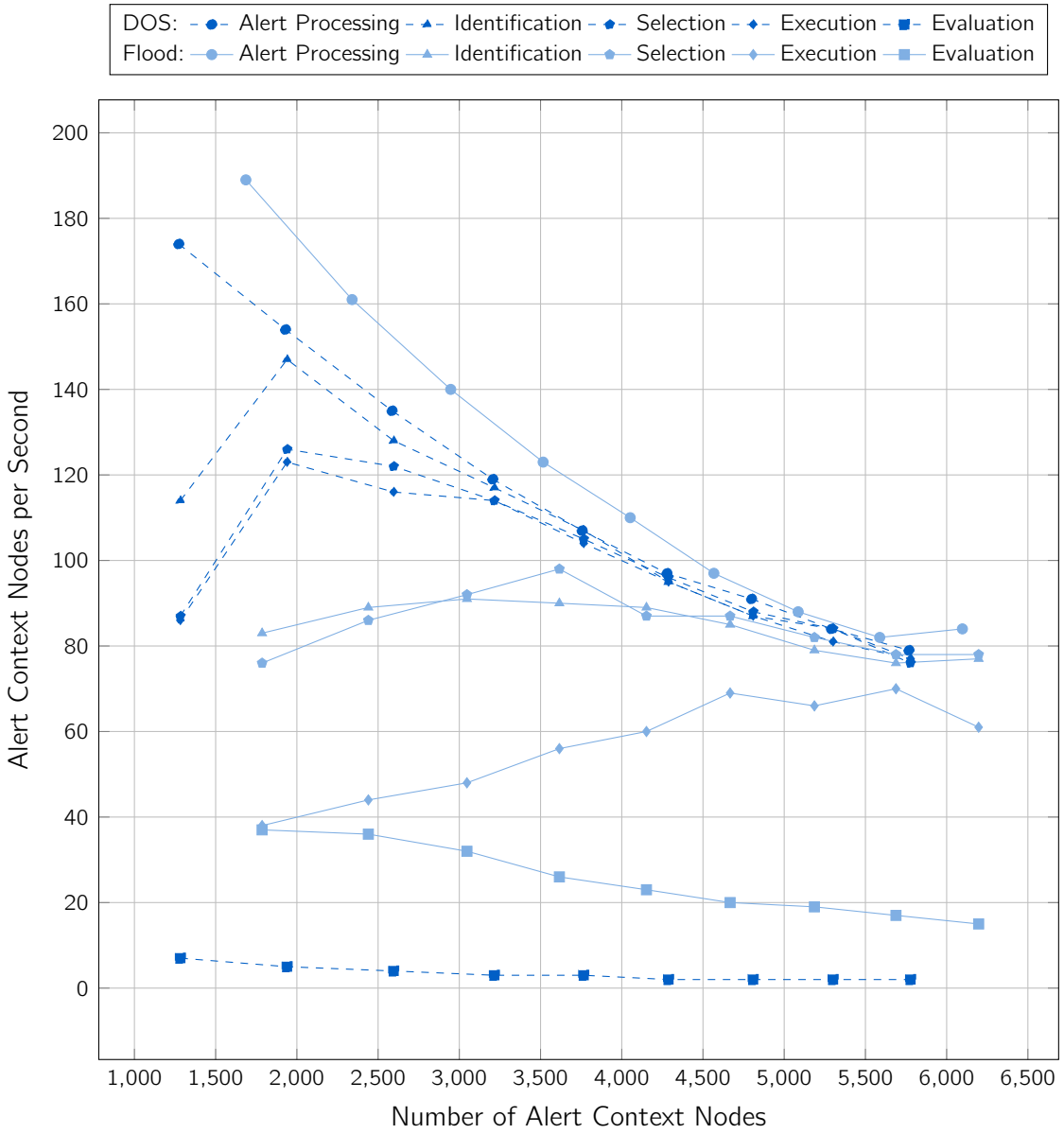


Fig. 8.3: `Alert Context` Nodes per Second Rate for all Test Cases of the DOS and Flooding Dataset

First, we examine the number of `Alert Context` nodes that can be processed per second (node/s). Therefore, all our measurements are shown within one single plot in Figure 8.3. The x-axis shows the number of `Alert Context` nodes generated due to processing the dataset. The y-axis shows the execution time needed to process the dataset

in average given in seconds (s). Within this plot the dominating impact of the response evaluation module for both datasets is obvious. For the DOS dataset all other modules are close together. Within the Flooding dataset, the impact of the response evaluation module is less than in the DOS dataset.

Additional differences of the impact of the modules within those datasets can be observed. The alert processing modules are a bit faster for the Flooding dataset, as already shown in Subsection 8.2.3. The response identification, response selection, and the response execution module are a bit faster on the DOS dataset, because the number of `Bundle` nodes to be processed is lower.

Examining the rate conducted of the new nodes or entities results, as expected, in an equivalent graph as depicted in Figure 8.3 only showing the `Alert Context` nodes. Therefore, those calculations are just given in Table 8.11 to show the much higher rates that can be achieved.

Tab. 8.11: New Nodes (First Part) and New Entities (Second Part) per Second Rate for all Test Cases (t_1 to t_5) and Datasets (d) [8]

#alerts	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**	**5000**
t_1 for d_d	310	274	239	211	192	177	166	156	147
t_2 for d_d	204	261	227	209	193	173	160	155	143
t_3 for d_d	156	223	217	203	189	174	162	156	143
t_4 for d_d	154	219	205	203	187	172	160	151	144
t_5 for d_d	12	9	7	6	5	4	4	3	3
t_1 for d_f	301	264	235	210	192	172	157	149	153
t_2 for d_f	134	147	153	155	155	151	142	138	141
t_3 for d_f	123	143	155	168	152	155	148	140	143
t_4 for d_f	62	72	82	97	105	121	118	127	111
t_5 for d_f	59	59	53	45	40	36	34	31	28
t_1 for d_d	1522	1349	1182	1055	966	896	843	793	754
t_2 for d_d	1029	1309	1139	1055	979	881	819	795	738
t_3 for d_d	787	1118	1086	1023	961	886	826	800	734
t_4 for d_d	778	1097	1027	1025	950	875	817	772	743
t_5 for d_d	62	44	35	29	25	21	19	17	15
t_1 for d_f	1382	1250	1135	1034	961	869	803	764	794
t_2 for d_f	606	695	741	765	778	764	726	709	731
t_3 for d_f	556	674	752	829	762	784	756	722	741
t_4 for d_f	282	341	396	479	528	614	606	656	576
t_5 for d_f	270	280	259	222	201	181	174	158	143

The results in Table 8.11 show the impact of the additional nodes and entities (nodes and relations). As the number of relations is enormous, the rate calculated on basis of the new entities inserted during the test runs is significantly higher compared to the rates calculated on alert or alert context basis.

Roughly a share of 30% (average 30,832%, standard deviation 1,23%) of all relations, are relations between different `Alert Context` nodes. This shows how highly connected the `Alert Context` nodes are. For modules using recursive queries, those interdependencies have a high impact, as those dependencies have to be resolved during runtime and are recalculated for each node the hierarchy is required.

Summary and Conclusion Within this section we have shown the interoperability of our proposed incident handling modules. The response identification, response selection, and response execution module examined in more detail in previous sections are usable for a comprehensive IHS. The response evaluation module, used for simulation purposes and to show how to close the loop within the proposed IHS, needs improvements to evidently increase its performance.

8.2.5 Use Case Analysis

Additionally, we investigate the use case described in Subsection 4.1.3 and 5.2.3 in more detail. The goal is to investigate the applicability of our proposed IHS for a real world scenario. For this analysis we use the same test cases (t_1 to t_4) as described previously in Subsection 8.2.4. As we already investigated the shortcomings of the response evaluation module, we concentrate on the other modules within this part of the evaluation. Therefore, we focus on a more detailed evaluation of the possible trigger mechanisms of the response identification module and possible impacts the triggers have. The used infrastructure and policy as well as the generated datasets for our use case analysis are examined in Subsubsection 8.2.5.1. The results of the evaluation are presented in Subsubsection 8.2.5.2 covering the stability of the measurements and the performance outcomes.

8.2.5.1 Generated Policy, Infrastructure and Dataset

Infrastructure Description The simplified infrastructure of the airbus data cabin network is shown in Subsection 4.1.3. For this analysis we used 16 services running separately in a Virtual Machine (VM). Those services are located in a dedicated network. In a second network the single devices (VMs) are located. Each of the 200 VMs is equipped with one interface and one IP address. The VMs run on a single machine that can be used as executor for certain responses. Additionally, a router connects the network the hosts are located in and the service network. The IDS is connected to both networks.

Policy Description The attacks, consequences, and responses are used as defined in Subsection 5.2.3. To suspend or slow down multiple services or devices the VM server can be utilized. Additionally, the VM server can limit the resources of each VM. The system can be rebooted by utilizing the VM server or by directly utilizing the corresponding VM. Privileges can be denied by each service itself or by deploying additional rules on the router. Systems can be blocked or traffic can be filtered by using the router of the system.

Summary All important data regarding the infrastructure and the used policy is listed in Table 8.12. Both are imported into the IHS using the `Extensible Markup Language (XML)` description given in Subsection 8.1.2.

Dataset As in previous sections, different instances of a dataset containing a varying number of `IDMEF` messages are generated. The smallest file contains 1000 alerts, the biggest contains 5000 alerts, the step size is 500 alerts. From the devices available within the infrastructure 5 devices are picked to act as internal attackers. All other devices and services can be potential victims of those attackers. The alerts were generated by simply choosing an attacker as source, an arbitrary device or service as victim and an attack. The distribution for the selection was equally distributed.

Tab. 8.12: Basic Information on the Generated Infrastructure and Used Policy

Property	Value	Property	Value
Number of IPs	221	Number of MACs	221
Number of interfaces	221	Number of services	15
Number of networks	2	Number of users	0
Number of devices	218	Number of routers	1
Number of IDSes	1	Number of hosts	200
Number of service hosts	16		
Number of attacks	7	Number of consequences	5
Number of responses	7	Number of implementations	237

We used all 7 possible attacks, resulting in 7 `Alert Context` nodes for an aggregation of attack classifications ($same_c = 7$) for all variants of this datasets. For this dataset the following with respect to same source applies to all variants: $same_s = 5$. In the following Table 8.13 the not yet described properties of the dataset are listed.

Tab. 8.13: Properties of the Use Case Dataset

Setting	D1	D2	D3	D4	D5	D6	D7	D8	D9
$\#alerts$	1000	1500	2000	2500	3000	3500	4000	4500	5000
$\#context$	1168	1588	2007	2360	2739	3063	3388	3621	3939
$\#unique$	231	233	233	233	233	233	233	233	233
$\#same_t$	219	221	221	221	221	221	221	221	221
$\#identified_C$	561	705	740	735	749	731	762	749	747
$\#selected_C$	528	692	706	691	728	700	665	719	709
$\#identified_S$	692	883	899	922	864	886	861	885	917
$\#selected_S$	622	799	813	833	788	802	783	803	832
$\#identified_T$	1149	1248	1269	1277	1272	1275	1286	1292	1289
$\#selected_T$	213	215	215	215	215	215	215	215	215

During the test cases 7 `Bundle` nodes are calculated for the dataset as the same classification was used as trigger. This will result in 7 response plans generated as GPLMT descriptions stored on disk. Those bundles include possible responses, ($\#identified_C$) identified by the response identification module using alert contexts aggregated with the same attack classification as trigger and selected ($\#selected_C$) by the response selection module, as listed in Table 8.13. The respective values using same targets ($\#identified_T$ and $\#selected_T$) and the same source ($\#identified_S$ and $\#selected_S$) as trigger conditions are listed as well. The number of identified and selected responses alters because of the same reasons as explained in Subsubsection 8.2.4.3.

8.2.5.2 Evaluation Results

Within the evaluation different aspects are of interest. First, we will have a look on the stability of our measurements by examining the standard deviation of our measurements. Second, we will examine the execution time and the alert per second rate of our test cases for all variants of all datasets. Lastly, we will examine the new nodes and entities

per second rate as every dataset differs in terms of additional nodes to be inserted.

Measurement Stability In Table 8.14 the average standard deviation of all measurements in percent (%) of the average value accurate to 2 decimal places is shown. All measurements are done 5 times for this evaluation. The table denotes the test case (t_1 to t_4) and the different trigger settings [9]. As for Test Case t_1 the response identification module is not activated this measurement has not assigned a trigger setting. For all other trigger settings each single test case is listed.

Tab. 8.14: Standard Deviation for all Test Cases (t_1 to t_4) and Trigger Settings [9] in Percent (%) of the Average Value

#alerts	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**	**5000**
t_1	0.67	0.54	0.32	0.64	0.47	1.23	1.74	0.32	2.36
t_2 for $same_C$	0.43	3.11	3.66	0.46	0.57	0.76	1.03	0.77	0.52
t_3 for $same_C$	0.46	0.82	1.75	2.82	2.12	0.91	0.87	0.93	0.20
t_4 for $same_C$	0.45	1.98	2.05	1.68	0.59	0.42	0.76	0.58	0.65
t_2 for $same_S$	0.98	1.98	3.34	4.18	0.65	0.73	1.00	0.33	0.67
t_3 for $same_S$	1.03	1.06	2.50	2.56	0.35	0.72	1.71	0.48	0.60
t_4 for $same_S$	0.71	1.43	2.77	3.52	0.49	0.44	0.52	0.32	0.66
t_2 for $same_T$	1.62	2.35	2.68	2.09	1.52	3.57	3.67	4.07	1.54
t_3 for $same_T$	1.35	2.85	1.59	1.81	3.60	2.48	3.76	1.56	3.81
t_4 for $same_T$	0.58	2.01	0.73	1.11	1.07	1.36	2.85	0.62	3.83

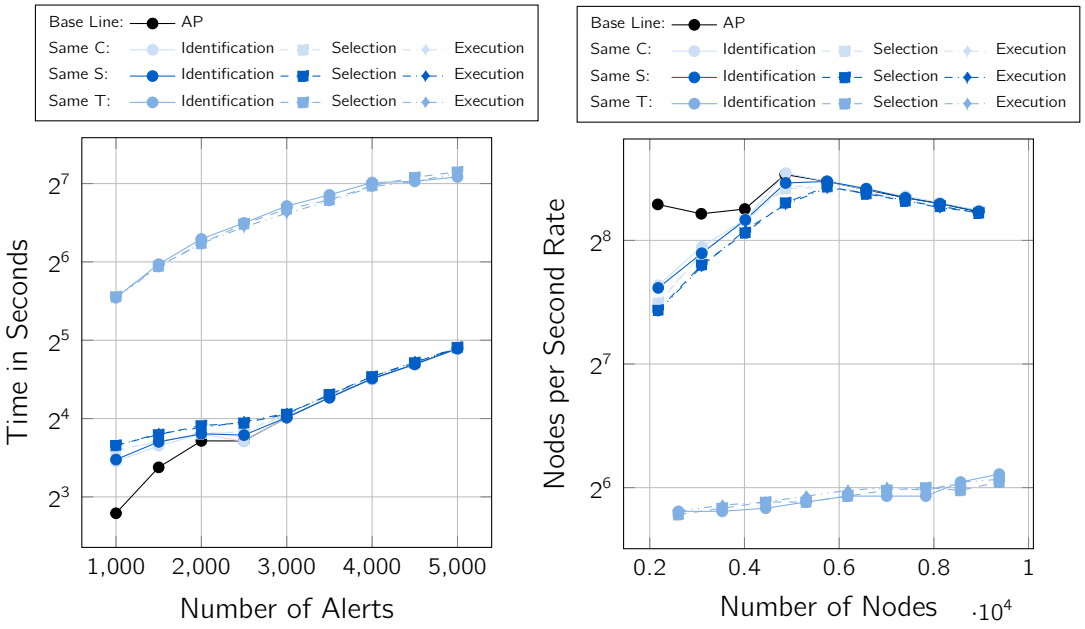
All deviations are below 5% of the average value of each test run. Only two deviations are slightly above 4%, and 10 additional deviations are above 3%. This indicates that the measurements are stable and the results are reliable.

Execution Time First, we examine the raw execution time for the use case scenario. The results are shown in Figure 8.4a. The x-axis shows the number of alerts within the dataset. The y-axis shows the execution time needed to process the dataset in average given in seconds (s). The y-axis is given in log-scale with basis 2.

As Test Case t_1 is the same for each trigger setting, this test case is plotted as base line. For each trigger setting, the response identification, response selection, and response execution modules are activated subsequently. For the Trigger Settings $same_C$ and $same_S$ the number of `Bundle` nodes identified and added are nearly equal ($same_c = 7$ and $same_s = 5$). Therefore, the measurements are close together and do not show a high additional impact compared to the base line. As evaluated before, the additional impact of the response identification, response selection, and response execution modules compared to each other is very low.

Using the Trigger Setting $same_T$ the workload for the system is much more higher. As 219 respectively 221 different targets appear within the variants of the dataset. For each of those aggregated `Alert Context` nodes, a `Bundle` node is created and processed by the response identification module. This results in a higher execution time and shows the importance of an appropriate trigger condition to have a low number of concurrent bundles to be processed.

[9] same source = $same_S$, same target = $same_T$, same attack classification = $same_C$



(a) Execution Time **(b)** Nodes per Second Rate (node/s)

Fig. 8.4: Execution Time and Nodes per Second Rate for all Test Cases and Trigger Settings [9] for the Use Case Scenario

Using the given trigger conditions each alert context is solved, as each alert is part of those `Alert Context` nodes. Using the same source or attack classification results in an `Issue` node fusing much more `Alert Context` nodes compared to fusing based on the same target. Relying on `Issue` nodes that are generated based on a stable `Alert Context` node and capable of fusing additional parts into the `Issue` node are vital for the systems applicability.

The evaluation shows, that in this case only the trigger condition is the determining factor for the additional impact on the execution time. The response selection and response execution module have a low impact on the execution time, even when the trigger condition is not optimal, although this implies additional work load for both modules, as they are triggered more often.

Alerts per Second Rate Next, we examine the rate for alerts that can be processed. In Table 8.15 the number of alerts that can be processed per second (alert/s) are shown. The table denotes the test case (t_1 to t_4) and the different trigger settings [9].

The rates for the trigger using the same attack classification and the same source are nearly equal as the effort for the system is comparable. Comparing both rates with the last trigger setting using the same target, shows that this rate is lower, even if the outcome, all alerts are solved, is the same. This shows again, the importance of an appropriate trigger condition.

Comparing the results in this evaluation with the previous simulations, shows that the rates for the use case are much better. That is because the number of responses corresponding implementations and consequences is lower. The simulations use an extensive policy to show the applicability for larger systems. The use case of the data cabin network has less response options and is more restrictive than the policy used in the simulations.

As seen before the rates rise with a higher number of alerts and slightly drop for the

Tab. 8.15: Alert per Second Rate (alert/s) for all Test Cases (t) and Trigger Settings [9]

#alerts	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**	**5000**
t_1	145	144	152	191	186	181	176	174	168
t_2 for $same_C$	91	119	143	191	186	181	177	175	169
t_3 for $same_C$	83	115	143	176	179	178	173	171	168
t_4 for $same_C$	83	115	142	178	180	179	173	170	166
t_2 for $same_S$	90	115	143	181	186	182	176	174	168
t_3 for $same_S$	79	108	133	162	181	177	172	172	166
t_4 for $same_S$	79	107	136	161	179	178	173	170	168
t_2 for $same_T$	21	24	26	28	29	30	31	34	37
t_3 for $same_T$	21	24	27	28	29	32	32	33	35
t_4 for $same_T$	21	25	27	29	30	32	32	35	36

last datasets. This is because the additional effort for more alerts is lower with a rising number of alerts. Additional `Alert` nodes inserted into the system will only lead to additional edges, and no additional nodes, e.g. alerts are only added to an existing alert context during aggregation.

New Nodes and Entities per Second Rate Within this evaluation we compare all our measurements based on the entities that can be processed. As already shown before, the rate conducted of the new nodes or entities results, as expected, in an equivalent graph as if only showing the `Alert Context` nodes, we show only the respective graph of new nodes. The results are shown in Figure 8.4b. The x-axis shows the number of nodes inserted during processing the dataset. The y-axis shows the nodes per second rate needed to process the dataset in average given in seconds (node/s). The y-axis is given in log-scale with basis 2. As the representation of the new nodes and the new entities nearly look the same, we limit ourselves to show the behavior for new nodes.

Within Figure 8.4b we see the opposite behavior compared to the execution time. The number of nodes on the x-axis is slightly different for all trigger settings and the base line. This is because the number of `Bundle` nodes differ through the settings. Within the base line, no bundles are present, the trigger setting based on the same attack classification and same source are nearly equal. The last trigger setting based on the same target shows a shift to the right on the x-axis as noticeably more `Bundle` nodes are generated.

Summary and Conclusion Within the evaluation of the use case, we are able to show the applicability of the proposed IHS for the given use case of the data cabin network of an aircraft. Additionally, we investigate the importance of the trigger mechanisms within the response identification module by showing the impact of amendable trigger settings leading to a higher complexity and more concurrent response execution processes. Moreover, we validated the minor impact of the response selection and response execution module even with non-optimal settings for the trigger conditions in the response identification module.

8.2.6 Security and Threat Analysis

In this subsection possible attack vectors on the proposed IHS are examined and strategies to cope with those attacks are discussed. We consider the following attack vectors for our IHS:

- DOS attacks on the IHS are covered in Subsubsection 8.2.6.1
- Compromised or impersonated components are covered in Subsubsection 8.2.6.2
- Data leakage on the blackboard is covered in Subsubsection 8.2.6.3
- Data loss on the blackboard is covered in Subsubsection 8.2.6.4

8.2.6.1 DOS Attacks

A typical challenge for IDSes are DOS attacks flooding the system with alerts and leading to critical performance decreases. In order to secure the proposed IHS from DOS attacks, different countermeasures can be applied.

First, the interfaces responsible for insertion of information can be enhanced by applying *prefiltering* of incoming information. This mainly targets interfaces for IDSes inserting new `Alert` and `Alert Context` nodes. Single IDSes can be *rate limited* in order to ensure that a compromised IDS cannot flood the proposed IHS. In case of corrupted monitoring modules, those can be rate limited as well.

Additionally, *multiple instances* of the interfaces or modules can be used in parallel. Multiple interface instances can balance the load occuring during the insertion of alerts raised by IDSes. Multiple instances of the modules can balance the load of the system during processing. In case of using multiple module instances in parallel the controller can be responsible for distributing the load between the single modules.

Another option for splitting the load between modules of the alert processing step is to split the rules they process. An aggregation or correlation module is equipped with a number of different aggregation or correlation rules. For example, the aggregation module applies rules to aggregate `Alert Context` nodes with the same target and `Alert Context` nodes with the same source, each of those two rules can be applied by two instance of the aggregation module.

Additionally, the controller itself can be designed to be distributed. Multiple instances of the controller can be achieved by separating the responsibilities of the controller. This means that each controller instance is responsible for a certain distinct subset of information elements and notifies modules listening to those information elements.

8.2.6.2 Compromised or Impersonated Components

In case a single module or interface of the proposed IHS is compromised wrong information can be propagated on the blackboard. This can lead to wrong decisions within the incident handling process. Additionally, compromised modules or interfaces can simply not forward information to the blackboard. Consequently, the security incident cannot be detected. Additionally, an attacker may tries to impersonate a certain module in order to disturb the IHS. The attacker subscribes for information elements stored on the blackboard and gets notified on changes. The attacker can than actively disturb the execution by adding or deleting certain information elements stored on the blackboard.

To prevent those attacks the modules and interfaces needs to be *authenticated* by the controller, e.g. using *remote attestation*. Before a module or interface can subscribe

to certain information elements, the modules have to prove their authenticity, e.g. using cryptographic keys. The controller can than set the appropriate access permissions for the module or interface. This allows to restrict the access of single modules to information elements they need for their execution by limiting their access to, e.g. certain tables. Using authentication mechanisms ensures that only legitimate modules or interfaces can connect to the blackboard. In case signs of a compromise of a module or interface appear, the controller can revoke the access rights of the respecting module or interface. The compromised module is than not able to access the blackboard until the controller allows access again. Therefore, the prove of authenticity has to be done repeatedly.

In order to cope with a compromised controller, the following strategy can be applied. Multiple controller instances are used. Those controller instances monitor and observe each other. In case a compromised controller is detected, a *consensus protocol* can be used to determine whether a suspicious controller is compromised or not. If the critical mass of controllers agree on the compromise of the suspicious controller they can conjointly exclude this controller by revoking its access rights on the blackboard.

8.2.6.3 Data Leakage on the Blackboard

Each module or interface can potentially read and write all information elements stored on the blackboard. Due to an intrusion that compromises a module or interface or due to a failure, information not needed for the execution of a module or interface can leak. This can be critical as the network topology is stored on the blackboard. For example, the users that are logged on certain devices are stored on the blackboard, that allows to track the single users.

To encounter this problem the *principle of least privileges* has to be enforced. Hereby, privileges are interpreted as access rights to certain information elements, attributes the information elements have and relations between information elements. This access of modules and interface has to be restricted. This can be done by limiting the access of a certain module or interface directly on the blackboard. The underlying database has to support multi-user access in order to equip each module with unique credentials. With respect to the modules or interfaces needs, the access rights to the information elements has to be restricted to those needed for the module's execution.

If a module or interface needs a combination of different information elements connected through edges the blackboard has to provide *views* encapsulating this combined information element. For example, a module needs the relation between `IP` nodes and `Device` node, it would need to follow the edges from the `IP` to the `MAC` node and further to the `Device` node. The MAC address is not relevant to the module but needs to be traversed. The blackboard can than provide a view containing the mapping of IP and device without information about the MAC address.

8.2.6.4 Data Loss on the Blackboard

Due to attacks on the blackboard itself, information stored on the blackboard may get lost. For example, the underlying database or the device the database runs on could be compromised and damaged.

To counteract possible data loss of the blackboard a *distributable database* can be used as underlying database system. The data can than be distributed across different multiple physical locations that are independent from each other such that in case of an attack only a single instance is affected. For synchronization between the multiple instances of the database replication or duplication strategies can be used to keep a concurrent

state of all instances. In case of data loss of single instances the affected database can be restored using the remaining working instances of the database. Additionally, the distributed database increases the resilience of the proposed IHS.

8.3 Publication Reference

Parts of the content of this chapter are already published on WISCS (Workshop on Information Sharing and Collaborative Security) 2016 held in cooperation with NOMS (cf. [62]). The own contribution of this paper are the design and the implementation of the proposed IHS. The design and realization of the evaluation are also part of the own contribution. This chapter includes a more detailed view on the implementation and evaluation than the paper. The evaluation of the response capabilities and the use case evaluation was not part of the paper.

9. CONCLUSIONS

Incident handling, consisting of intrusion detection, alert processing and intrusion response, increases the resilience of IT systems and ensures service continuity even during an ongoing attack. This thesis addresses the following central research objective:

How to increase security of computer networks by integrating automated intrusion response into a holistic Incident Handling System (IHS)?

This overall research objective can be divided further into research questions that are answered in this thesis. In the following we will cover the contributions to the single research questions achieved in this thesis and highlight the key findings ascertain in this thesis. Afterwards, we discuss possible future work and research directions.

9.1 Contributions to Research Questions and Key Findings

Research Question RQ1 – What are requirements for an IHS? We identify functional and non-functional requirements for a holistic IHS from analyzing existing IHSes and collecting explicitly stated requirements from literature. Our analysis of the related work shows that non of the existing approaches is able to fulfill all identified requirements. The main issues are spotted in the field of triggering responses in a flexible manner and structured execution of responses. Additionally, the selection mechanisms are not usable for multiple responses.

Research Question RQ2 – How to integrate all single steps into an IHS in a continuous manner? To answer this research question, we analyze possible execution models for incident handling. We identify shortcomings of sequential execution and approaches based on Complex Event Processing (CEP) or agents. Additionally, we examined the advantages of systems based on an information sharing component and cover limitations of existing approaches. Based on this findings we propose a comprehensive and collaborative IHS based on the Blackboard Pattern using an information sharing component and an appropriate execution model. We disassembled the incident handling process into more manageable modules solving subproblems of the incident handling process and providing a holistic solution due to collaboration. Those modules are designed to be free of interference and conflicts to enhance collaboration between these modules. Within our overall evaluation we are able to show the interoperability of our proposed module and the applicability of our comprehensive IHS.

Research Question RQ3 – What information elements are required to support all steps of incident handling? Using the Blackboard Pattern requires an underlying information model in order to enable the modules to collaborate. Therefore, we analyze the needed steps of incident handling, namely intrusion detection, alert processing, and

intrusion response, and identify required information elements. We propose a novel information model covering all needed information elements using a graph-based representation to provide the needed connections between information elements and enable information sharing between the modules.

Research Question RQ4 – How to reliably identify attacks in a stream of packets? Within this research question we investigate that the reliability of an Intrusion Detection System (IDS) is crucial for subsequent intrusion response. Automated intrusion response can deal with detected attacks as well as misconfigurations, but is partially useful in case of false alerts. Therefore, we propose an anomaly-based IDS using a rule-based specification of the protocol behavior. Deviations from the specified behavior are detected as attacks and will be handled by subsequent intrusion response. We propose a novel IDS based on CEP and provide a rule set for the SOME/IP protocol that is used in the data cabin network acting as use case of this thesis. Additionally, our approach is capable of identifying attacks within a stream of packets as soon as the attack is unambiguous. The proposes IDS allows to reliably detect attacks just-in-time within a stream of packets.

Research Question RQ5 – How to determine when to react? Based on our information model, we derived possible trigger conditions to initiate automated intrusion response. Those triggers can be calculated based on the collected information and are combinable in a flexible manner in order to cover multiple different use cases. In our evaluation of the overall system, we show that the selection of an appropriate trigger mechanisms is crucial for the efficiency of the overall IHS.

Research Question RQ6 – What are suitable response options and how to identify them? We identified and classified possible responses an holistic IHS can support due to a literature search. We provide a classification scheme based on available taxonomies that eases the identification and selection of responses for specific use cases and environments. We utilized this scheme for our use case, the data cabin network within an airplane, to map possible attacks, our IDS is able to detect, to applicable responses with respect to the requirements of the underlying use case. In our implementation we provide a response identification module that is capable of identifying effective and available responses based on the classification and target of a security incident.

Research Question RQ7 – How to assess and select an optimal set of responses from multiple response options? Given a list of suitable responses, the optimal set of responses to execute on the target system needs to be selected. We identified relations between responses including or excluding other responses from the response plan. To model the response selection process we propose a set-based representation of the response selection mechanism that can be transferred to a linear optimization problem. The resulting Mixed Integer Linear Programming (MILP) definition can be solved and the optimal set is deduced. We provide an implementation using GLPK and CPLEX that are compared with respect to the solution quality and performance with two heuristics. We are able to show that the solution quality can be improved by using our MILP-based approach while the computational effort is still acceptable. Compared to existing approaches towards response selection, we are able to determine a comprehensive response plan considering relations between responses instead of only selecting a single response for a dedicated alert.

Research Question RQ8 – How to deploy and execute the selected set of responses within the target system? To answer this research question we propose a response plan description language to allow a structured execution of selected responses. We provide an execution framework, called GPLMT, that can be used as central response repository, allows a flexible deployment of responses and can execute a structured response plan given in our description language. We integrated this tool into our overall IHS and provide an automated generation of response plans based on selected responses.

9.2 Future Work and Further Research Directions

Potential future work and further research directions are possible in different areas. In the following, we examine those areas and spot possible solutions to the identified issues.

Security Related Issues Our evaluation of the overall IHS shows that the security of the proposed system can be improved. Here, a more detailed analysis of possible attack vectors is required in order to identify possible issues. Afterwards, additional security measures to protect the IHS itself have to be integrated into the system. Especially, authentication and identification mechanisms have to be developed in order to protect the blackboard storing all the required information and to cope with malicious or misconfigured modules. As may be privacy-related information is stored on the blackboard, the principle of least privileges has to be enforced for modules working on the blackboard. Here, an access strategy has to be elaborated and deployed on the blackboard. Additionally, the communication between the modules and the blackboard needs to be secured appropriately. Secure and encrypted channels are required to cope with interception and eavesdropping attempts.

Performance Related Issues With respect to performance some additional measures can be integrated into the proposed IHS to improve the overall performance. First, the response evaluation module has to be improved as the evaluation had shown shortcomings of this module. Additionally, another backend may be more appropriate. Within our evaluation we show that OrientDB is not suitable as graph-based database, but other graph-based databases are available that may be more suited than OrientDB. In this field a more deeper analysis needs to be done. Additionally, all modules do not use caches that may further improve the performance. Closely related are optimizations with respect to the querying behavior. The current implementation does not support bulk queries but inserts or updates each information element within a single query. Of course this is not optimal and provides space for improvements. A last option for improving the performance is to exchange the programming language and using a more performant programming language than Python.

Distribution Related Issues The proposed system was only tested on a single machine. The design allows to distribute the underlying blackboard as well as the modules across the network. Therefore, an appropriate communication channel is required. An evaluation with respect to performance would be needed for the distributed variant of the proposed IHS. Additionally, the design of a distributed controller can enhance the overall IHS. Methods to coordinate all distributed components need to be developed and further evaluated.

Feature Related Issues The functionality of our proposed controller is limited. Within this directions improvements are possible. Our controller does not support symmetric

publish-subscribe and cannot cope with a more complex control plan. Additionally, security related features could be added into the controller. Our implementation does not provide a garbage collector. The design, implementation and evaluation of a suitable garbage collector is also part of future work. Within the field of alert processing our implementation is limited. Here, additional modules can be integrated and tested. The same applies within the field of monitoring and IDSes. Within this fields we see space for improvements as those topics are out of scope of this thesis.

ABBREVIATIONS

ACID Atomicity, Consistency, Isolation and Durability. 138

AGI Airbus Group Innovations. v, 47, 50, 53, 61, 66, 81

BMBF German Federal Ministry of Education and Research. v

CEP Complex Event Processing. vii, ix, 4, 5, 37, 38, 47, 52–54, 60, 63–66, 169, 170

CIA Confidentiality, Integrity and Availability. 7, 90, 91

CIDF Common Intrusion Detection Framework. 7, 10

CPLEX IBM ILOG CPLEX Optimization Studio. 106, 108–112, 120, 152, 170

DecADe Decentralized Anomaly Detection. v

DOS Denial of Service. 29, 31, 42, 50, 59, 64, 71, 82, 97, 119, 148, 153, 165

ECA Event Condition Action. 14, 42, 115, 118, 181

EPL Event Processing Language. 54–57, 60

GLPK GNU Linear Programming Kit. 106, 108–112, 152, 170

GPLMT GNUnet Parallel Largescale Management Tool. 121–131, 135, 141–145, 152, 154, 156, 161, 171, 177, 181

HIDS Host-based Intrusion Detection System. 8, 9, 17, 20

I(D)RS Intrusion (Detection and) Response System. 7, 39

IDMEF Intrusion Detection Message Exchange Format. 7, 8, 10, 24, 28, 29, 47, 115, 142–144, 147, 152, 160

IDS Intrusion Detection System. vii, ix, 1, 3, 4, 7–10, 13, 17, 20, 21, 23, 26, 28–32, 38–40, 42–44, 46, 47, 50–52, 60, 63–66, 79, 80, 90–94, 117, 119, 123, 142–144, 148, 152, 153, 160, 161, 165, 170, 172, 177

IHS Incident Handling System. vii, ix, 2, 4, 5, 8, 17–24, 29, 31–33, 39, 42, 43, 45, 60, 67, 84, 85, 90, 120, 121, 132, 137–139, 141, 143–147, 150, 151, 153, 160, 164, 165, 167, 169–171, 177

ILP Integer Linear Programming. 96

IP Internet Protocol. 53

IPS Intrusion Prevention System. 7, 11

IRMEF Intrusion Response Message Exchange Format. 132, 135

IRS Intrusion Response System. 1, 7, 11, 13, 14, 17, 37, 39, 41–45, 68, 93, 97, 98, 114, 117, 119, 131, 132, 134

JSON JavaScript Object Notation. 140

LP Linear Programming. 87, 95–97, 101, 106, 120

MILP Mixed Integer Linear Programming. vii, ix, 4, 5, 87, 96, 98, 101, 102, 105–107, 112, 145, 170

NETCONF Network Configuration Protocol. 133, 135

NIDS Network-based Intrusion Detection System. 5, 9, 17, 20, 50, 52, 64

OrBAC Organization Based Access Control. 28, 40, 41, 115

RELAX NG Regular Language Description for XML New Generation. 130

RPC Remote Procedure Call. 47, 48, 133

SCP Secure Copy. 131

SIEM Security Information and Event Management. 29, 30

SNMP Simple Network Management Protocol. 132, 133, 135

SOME/IP Scalable service-Oriented MiddlewarE over IP. vii, ix, 47–50, 52–57, 60, 61, 64, 66, 81, 82, 170, 177, 179

SQL Structured Query Language. 54, 60, 61, 138

SSH Secure Shell. 122–124, 130, 132

SURF Systemic Security for Critical Infrastructures. v

SWRL Semantic Web Rule Language. 115

TCP Transmission Control Protocol. 47, 53

UDP User Datagram Protocol. 47, 53

VM Virtual Machine. 26, 68, 69, 81–84, 89, 123, 160

WinRM Windows Remote Management. 132

XML Extensible Markup Language. 10, 55, 122, 124, 130, 133, 143, 160

XSD XML Schema. 130

GLOSSARY

Alert Processing

Processing incoming raw alerts using different methods like aggregation, correlation, and root cause analysis. vii, 2–5, 8, 10, 11, 15, 17, 19–24, 26, 30–33, 37, 38, 40, 41, 43, 45, 46, 85, 143–145, 147, 152, 159, 165, 169, 172

Blackboard Pattern

Software pattern with global information sharing component (knowledge base, blackboard) commonly updated by modules (knowledge sources). vii, ix, 17, 20–22, 30, 146, 169, 177

Candidate Responses

Set of responses applicable to the security incident and deployed on the victim (target) of an attack. 15, 18, 23, 32–34, 67, 68, 75, 86, 87, 92, 98, 103, 116–118, 145, 146, 152, 154

Incident Handling

Comprehensive process consisting of the three main steps intrusion detection, alert processing and intrusion response. vii, 1–3, 5, 8, 11, 17–20, 22, 24, 29–31, 37–39, 43–45, 135, 142, 145, 146, 160, 165, 169

Information Element

A single node within the information model representing a single concept. 2, 3, 23, 24, 26, 29, 46, 79, 85, 97, 139, 144–146, 151, 157, 165, 166, 169–171

Infrastructure Information

Part of the information model describing the underlying target system. 23, 24, 26, 28–35, 68, 79, 97, 101, 102, 124, 131, 142, 147

Intrusion Detection

Identifying attacks against the target system by monitoring and analyzing the target system and generating security events (alerts) in case of a detected attack. vii, 1, 2, 4, 5, 8, 9, 11, 19, 20, 22, 24, 26, 28, 37, 39, 40, 42, 44, 45, 47, 115, 117, 169

Intrusion Prevention

Hardening and securing the target system such that security incidents are prevented to happen. 1

Intrusion Response

Counteracting a detected intrusion by triggering appropriate responses. vii, 1–5, 7, 8, 13, 15, 17–24, 26–30, 32, 33, 37, 39, 40, 44–46, 51, 52, 60, 67, 68, 75–81, 85, 92, 97, 115, 118, 129, 135, 137, 143–146, 151, 152, 169, 170

Response Assessment

Evaluating the costs of a single or set of responses to enable response selection and to compare the response costs against the costs of the security incident. 15, 20, 23, 34, 39, 43, 87, 88, 90, 92, 94, 95

Response Evaluation

During response execution metrics required for response selection and response assessment are collected. 23, 34–36, 144, 146, 152, 154–157, 159, 160, 171

Response Execution

Executing responses on the target system following a predefined response plan. 4, 5, 14, 33–36, 40, 41, 43, 46, 86, 101, 121, 131–135, 143–145, 152, 156, 159, 160, 162–164

Response Identification

Identifying suitable responses applicable to the security incident and deployed on the victim (target) of an attack as candidate responses. 4, 33–36, 67, 68, 75, 76, 78, 84–86, 102, 143, 145, 146, 152, 154, 155, 157, 159–162, 164, 170

Response Plan

Structured execution plan for responses including the control flow and response interdependencies. vii, 4, 5, 15, 18, 19, 33–36, 40, 41, 46, 95, 99, 107, 118–132, 134, 135, 144, 145, 152, 154, 161, 170, 171

Response Selection

Identifying the optimal combination of responses from a set of candidate responses. vii, 4, 5, 14, 20, 27, 33–36, 39–43, 87, 92, 94, 95, 101, 102, 106, 107, 114–117, 119, 120, 143, 145, 146, 152, 154–156, 159–164, 170

Security Incident

A detected attack consisting of one or multiple security events. vii, 1–3, 7, 8, 10–15, 17–20, 23, 26–29, 33–36, 40–44, 46, 52, 53, 60, 67, 68, 70, 75, 76, 78–81, 86–95, 97–103, 106, 107, 114–120, 122, 123, 128, 129, 132, 145, 146, 148, 165, 170

Target System

The underlying infrastructure that is monitored and protected. vii, 1–3, 7–9, 11, 13, 15, 17–20, 23, 27–29, 31, 33, 34, 39, 42–44, 50, 60, 67–70, 75, 78, 79, 81, 82, 84, 86–88, 95, 98, 99, 101, 107, 116, 121–124, 131, 132, 135, 140, 141, 148, 170, 171

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

LIST OF ALGORITHMS

LIST OF EQUATIONS

Bibliography

[1] Albrecht, J., and Huang, D. Y. Managing Distributed Applications using Gush. In *Testbeds and Research Infrastructures. Development of Networks and Communities*, T. Magedanz, A. Gavras, N. Thanh, and J. Chase, Eds., vol. 46 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. Springer Berlin Heidelberg, 2011, pp. 401–411.

[2] Aniello, L., Lodi, G., and Baldoni, R. Inter-domain Stealthy Port Scan Detection Through Complex Event Processing. In *Proceedings of the 13th European Workshop on Dependable Computing (EWDC '11)* (New York, NY, USA, 2011), ACM, pp. 67–72.

[3] Aniello, L., Luna, G. A., Lodi, G., and Baldoni, R. A collaborative event processing system for protection of critical infrastructures from cyber attacks. In *Proceesding of 30th International Conference on Computer Safety, Reliability, and Security (SAFECOMP)* (Berlin, Heidelberg, 2011), Springer Berlin Heidelberg, pp. 310–323.

[4] Anuar, N., Papadaki, M., Furnell, S., and Clarke, N. An investigation and survey of response options for Intrusion Response Systems (IRSs). In *Information Security for South Africa (ISSA)* (2010), pp. 1–8.

[5] Anuar, N. B., Papadaki, M., Furnell, S., and Clarke, N. An investigation and survey of response options for Intrusion Response Systems (IRSs). In *Information Security for South Africa (ISSA)* (2010), IEEE, pp. 1–8.

[6] Atighetchi, M., Pal, P., Webber, F., Schantz, R., Jones, C., and Loyall, J. Adaptive Cyberdefense for Survival and Intrusion Tolerance. *IEEE Internet Computing 8*, 6 (2004), 25–33.

[7] AUTOSAR. Specification of SOME/IP Transformer. Tech. Rep. AUTOSAR Release 4.2.2 UID 660, Standard, AUTomotive Open System ARchitecture, 2015.

[8] Balepin, I., Maltsev, S., Rowe, J., and Levitt, K. Using Specification-Based Intrusion Detection for Automated Response. In *Recent Advances in Intrusion Detection*, G. Vigna, C. Kruegel, and E. Jonsson, Eds., vol. 2820 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003, pp. 136–154.

[9] Bhandari, P., and Gujral, M. S. Ontology based approach for perception of network security state. In *Recent Advances in Engineering and Computational Sciences (RAECS)* (March 2014), pp. 1–6.

[10] Bhargavi, R., Vaidehi, V., Bhuvaneswari, P. T. V., Balamurali, P., and Chandra, G. Complex Event Processing for Object Tracking in Wireless Sensor Networks. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)* (August 2010), vol. 3, pp. 211–214.

[11] Birkholz, H., Sieverdingbeck, I., Sohr, K., and Bormann, C. IO: An Interconnected Asset Ontology in Support of Risk Management Processes. In *ARES* (2012), IEEE Computer Society, pp. 534–541.

[12] Bloem, M., Alpcan, T., and Basar, T. Intrusion Response as a Resource Allocation Problem. In *45th IEEE Conference on Decision and Control* (December 2006).

[13] Boyd, J. A Discourse on Winning and Losing. *Unpublished set of briefing slides available at Air University Library, Maxwell, AFB, Alabama* (May 1987).

[14] Briesemeister, L., Cheung, S., Lindqvist, U., and Valdes, A. Detection, correlation, and visualization of attacks against critical infrastructure systems. In *8th Annual International Conference on Privacy Security and Trust (PST)* (August 2010), pp. 15–22.

[15] Bro. https://www.bro.org/.

[16] Buchmann, A., and Koldehofe, B. Complex event processing. *IT-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik 51*, 5 (2009), 241–242.

[17] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., Sommerlad, P., and Stal, M. Pattern-oriented software architecture, volume 1: A system of patterns, 1996.

[18] Butler, J. M. Need for Speed: Streamlining Response and Reaction to Attacks. A sans whitepaper, infosec reading room, SANS Institute, May 2013.

[19] Carey, N., Clark, A., and Mohay, G. IDS Interoperability and Correlation Using IDMEF and Commodity Systems. In *Information and Communications Security*, R. Deng, F. Bao, J. Zhou, and S. Qing, Eds., vol. 2513 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2002, pp. 252–264.

[20] Carver, C. A., Hill, J. M., and Pooch, U. W. Limiting uncertainty in intrusion response. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security* (2001), pp. 5–6.

[21] Carver Jr, C. A. *Adaptive agent-based intrusion response*. PhD thesis, Texas A and M University, 2001.

[22] Case, J., Fedor, M., Schoffstall, M., and Davin, J. A Simple Network Management Protocol (SNMP). RFC 1157, RFC Editor, May 1990.

[23] Cichonski, P., Millar, T., Grance, T., and Scarfone, K. SP 800-61 Rev. 2. Computer Security Incident Handling Guide. Tech. rep., National Institute of Standards & Technology (NIST), Gaithersburg, MD, United States, 2012.

[24] Costante, E., Fauri, D., Etalle, S., Hartog, J. D., and Zannone, N. A Hybrid Framework for Data Loss Prevention and Detection. In *2016 IEEE Security and Privacy Workshops (SPW)* (May 2016), pp. 324–333.

[25] Cugola, G., and Margara, A. Processing Flows of Information: From Data Stream to Complex Event Processing. *ACM Computer Surveys 44*, 3 (June 2012), 15:1–15:62.

[26] Cuppens, F. Managing alerts in a multi-intrusion detection environment. In *Proceedings 17th Annual Computer Security Applications Conference (ACSAC)* (December 2001), pp. 22–31.

[27] Cuppens, F., and Miege, A. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of IEEE Symposium on Security and Privacy* (2002), pp. 202–215.

[28] Cuppens, F., and Miège, A. Alert Correlation in a Cooperative Intrusion Detection Framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy (SP '02)* (Washington, DC, USA, 2002), IEEE Computer Society.

[29] Cuppens-Boulahia, N., Cuppens, F., de Vergara, J., Vazquez, E., Guerra, J., and Debar, H. An ontology-based approach to react to network attacks. In *3rd International Conference on Risks and Security of Internet and Systems (CRiSIS '08)* (October 2008), pp. 27–35.

[30] Dain, O., and Cunningham, R. K. Fusing a Heterogeneous Alert Stream into Scenarios. In *In Proceedings of the 2001 ACM Workshop on Data Mining for Security Applications* (2001), pp. 1–13.

[31] Dantzig, G. *Linear programming and extensions*. Princeton Univ. Press, Aug. 1963.

[32] Dass, M., Cannady, J., and Potter, W. D. A blackboard-based learning intrusion detection system: a new approach. In *Developments in Applied Artificial Intelligence*. Springer, 2003, pp. 385–390.

[33] Dass, M., Cannady, J., and Potter, W. D. LIDS: A learning intrusion detection system. In *FLAIRS Conference* (2003), pp. 12–16.

[34] Dave, B. T., and Jimit Mahadevia, S. Application Profiling based on Attack Alert Aggregation. *Global Journal of Computer Science and Technology 13*, 16 (2014).

[35] Debar, H., Curry, D., and Feinstein, B. The Intrusion Detection Message Exchange Format (IDMEF). RFC 4765, RFC Editor, March 2007.

[36] Debar, H., Dacier, M., and Wespi, A. Towards a Taxonomy of Intrusion-detection Systems. *Computer Networks 31*, 9 (April 1999), 805–822.

[37] Debar, H., Thomas, Y., , Cuppens, F., and Boulahia-Cuppens, N. Response: bridging the link between intrusion detection alerts and security policies. *Intrusion Detection Systems 3*, 3 (2008), 195–210.

[38] Debar, H., Thomas, Y., Boulahia-Cuppens, N., and Cuppens, F. Using Contextual Security Policies for Threat Response. In *Proceedings of the 3rd International Conference Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, R. Büschkes and P. Laskov, Eds. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 109–128.

[39] Debar, H., Thomas, Y., Cuppens, F., and Cuppens-Boulahia, N. Enabling automated threat response through the use of a dynamic security policy. *Journal in Computer Virology 3*, 3 (2007), 195–210.

[40] Debar, H., and Wespi, A. Aggregation and Correlation of Intrusion-Detection Alerts. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID '00)* (London, UK, UK, 2001), Springer-Verlag, pp. 85–103.

[41] Eckert, M., and Bry, F. Complex Event Processing (CEP). *Informatik-Spektrum 32*, 2 (2009), 163–167.

[42] Eckert, M., and Bry, F. Complex event processing (CEP). *Informatik-Spektrum 32*, 2 (2009), 163–167.

[43] Elshoush, H. An innovative framework for collaborative intrusion alert correlation. In *Science and Information Conference (SAI)* (August 2014), pp. 607–614.

[44] Elshoush, H. T., and Osman, I. M. An Improved Framework for Intrusion Alert Correlation. *Lecture Notes in Engineering and Computer Science 2197*, 1 (2012), 518–523.

[45] Enns, R., Bjorklund, M., Schoenwaelder, J., and Bierman, A. Network Configuration Protocol (NETCONF). Tech. Rep. 6241, Internet Engineering Task Force, June 2011. Updated by RFC 7803.

[46] EsperTech. Complex Event Processing and Event Series Analysis platform, 2015. http://www.espertech.com/esper/.

[47] EsperTech. Esper Reference. Version 5.2.0, 2015. http://www.espertech.com/esper/release-5.2.0/esper-reference/html/index.html.

[48] Fawaz, A., Berthier, R., and Sanders, W. Cost modeling of response actions for automated response and recovery in AMI. In *IEEE Third International Conference on Smart Grid Communications (SmartGridComm)* (November 2012), pp. 348–353.

[49] Ficco, M., and Romano, L. A Generic Intrusion Detection and Diagnoser System Based on Complex Event Processing. In *2011 First International Conference on Data Compression, Communications and Processing* (June 2011), pp. 275–284.

[50] Foo, B., Glause, M. W., Howard, G. M., Wu, Y.-S., Bagchi, S., and Spafford, E. H. Intrusion response systems: a survey. In *Information Assurance: Dependability and Security in Networked Systems*. Spafford EH (ed.). Morgan Kaufmann Publishers: Burlington, MA, 2008, pp. 377–412.

[51] Foo, B., Wu, Y.-S., Mao, Y.-C., Bagchi, S., and Spafford, E. ADEPTS: adaptive intrusion response using attack graphs in an e-commerce environment. In *2005 International Conference on Dependable Systems and Networks (DSN'05)* (2005), IEEE, pp. 508–517.

[52] Gad, R., Kappes, M., Boubeta-Puig, J., and Medina-Bulo, I. Employing the CEP paradigm for network analysis and surveillance. In *Proceedings of the 9th Advanced International Conference on Telecommunications* (June 2013), pp. 204–210.

[53] Gehani, A., and Kedem, G. RheoStat: Real-Time Risk Management. In *Recent Advances in Intrusion Detection: Proceedings of the 7th International Symposium (RAID04)*, E. Jonsson, A. Valdes, and M. Almgren, Eds. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 296–314.

[54] Genge, B., Siaterlis, C., Fovino, I. N., and Masera, M. A cyber-physical experimentation environment for the security analysis of networked industrial control systems. *Computers & Electrical Engineering 38*, 5 (2012), 1146 – 1161.

[55] Gonzalez-Granadillo, G., Alvarez, E., Motzek, A., Merialdo, M., Garcia-Alfaro, J., and Debar, H. Towards an Automated and Dynamic Risk Management Response System. In *Secure IT Systems: 21st Nordic Conference, NordSec 2016, Oulu, Finland, November 2-4, 2016. Proceedings* (Cham, 2016), B. B. Brumley and J. Röning, Eds., Springer International Publishing, pp. 37–53.

[56] Gonzalez Granadillo, G., Ben Mustapha, Y., Hachem, N., and Debar, H. An ontology-driven approach to model SIEM information and operations using the SWRL formalism. *International Journal of Electronic Security and Digital Forensics 7 4*, 2-3 (2012), 104–123.

[57] Gonzalez Granadillo, G., Débar, H., Jacob, G., Gaber, C., and Achemlal, M. Individual Countermeasure Selection Based on the Return On Response Investment Index. In *Computer Network Security*, I. Kotenko and V. Skormin, Eds., vol. 7531 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 156–170.

[58] Granadillo, G. G., Motzek, A., Garcia-Alfaro, J., and Debar, H. Selection of Mitigation Actions Based on Financial and Operational Impact Assessments. In *2016 11th International Conference on Availability, Reliability and Security (ARES)* (Aug 2016), pp. 137–146.

[59] Gresty, D., Shi, Q., and Merabti, M. Requirements for a general framework for response to distributed denial-of-service. In *Proceedings 17th Annual Computer Security Applications Conference (ACSAC)* (December 2001), pp. 422–429.

[60] Guttman, B., and Roback, E. A. SP 800-12. An Introduction to Computer Security: The NIST Handbook. Tech. rep., National Institute of Standards & Technology, Gaithersburg, MD, United States, 1995.

[61] Hasswa, A., Zulkernine, M., and Hassanein, H. Routeguard: an intrusion detection and response system for mobile ad hoc networks. In *IEEE International Conference on Wireless And Mobile Computing, Networking And Communications (WiMob'2005)* (2005), vol. 3, pp. 336–343.

[62] Herold, N., Kinkelin, H., and Carle, G. Collaborative Incident Handling Based on the Blackboard-Pattern. In *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security* (New York, NY, USA, 2016), WISCS '16, ACM, pp. 25–34.

[63] Herold, N., Posselt, S. A., Hanka, O., and Carle, G. Anomaly detection for SOME/IP using complex event processing. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium* (April 2016), pp. 1221–1226.

[64] Herold, N., Wachs, M., Posselt, S.-A., and Carle, G. An Optimal Metric-Aware Response Selection Strategy for Intrusion Response Systems. In *Foundations and Practice of Security: 9th International Symposium, FPS 2016, Québec City, QC, Canada, October 24-25, 2016, Revised Selected Papers*, F. Cuppens, L. Wang, N. Cuppens-Boulahia, N. Tawbi, and J. Garcia-Alfaro, Eds. Springer International Publishing, Cham, 2017, pp. 68–84.

[65] Hooper, E. An Efficient and Intelligent Intrusion Detection and Response System using Virtual Private Networks, Firewalls and Packet Filters. *International Journal of Security and Its Applications, IJSIA 1*, 1 (July 2007), 25–34.

[66] Inayat, Z., Gani, A., Anuar, N. B., Khan, M. K., and Anwar, S. Intrusion response systems: Foundations, design, and challenges. *Journal of Network and Computer Applications 62* (2016), 53 – 74.

[67] Jahnke, M., Thul, C., and Martini, P. Graph based Metrics for Intrusion Response Measures in Computer Networks. In *32nd IEEE Conference on Local Computer Networks (LCN)* (October 2007), pp. 1035–1042.

[68] Jahnke, M., Thul, C., and Martini, P. Comparison and Improvement of Metrics for Selecting Intrusion Response Measures against DoS Attacks. In *Sicherheit* (2008), Citeseer, pp. 381–393.

[69] Jalal Baayer, B. R. New Cost-Sensitive Model for Intrusion Response Systems Minimizing False Positive. *IJMER - International Journal of Modern Engineering Research 2*, 5 (October 2012), 3473–3478.

[70] Jaros, M. Distribution and orchestration of network measurements on the planetlab testbed. Bachelor's thesis, Technische Universität München, Chair for Network Architectures and Services, 4 2015.

[71] Jr, C. C., Hill, J. M. D., Surdu, J. R., Member, J. R. S., Pooch, U. W., and Member, S. A Methodology for Using Intelligent Agents to provide Automated Intrusion Response. In *In Proceedings of the IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop* (2000), pp. 6–7.

[72] Julisch, K. Mining alarm clusters to improve alarm handling efficiency. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC 2001)* (2001), IEEE, pp. 12–21.

[73] Julisch, K. Clustering Intrusion Detection Alarms to Support Root Cause Analysis. *ACM Transactions on Information and System Security (TISSEC) 6*, 4 (November 2003), 443–471.

[74] Julisch, K., and Dacier, M. Mining Intrusion Detection Alarms for Actionable Knowledge. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '02)* (New York, NY, USA, 2002), ACM, pp. 366–375.

[75] Kaisler, S. *Software Paradigms*. Wiley, 2005.

[76] Kamra, A., and Bertino, E. Design and Implementation of an Intrusion Response System for Relational Databases. *IEEE Transactions on Knowledge and Data Engineering 23*, 6 (2011), 875–888.

[77] Kanoun, W., Cuppens-Boulahia, N., Cuppens, F., and Dubus, S. Risk-Aware Framework for Activating and Deactivating Policy-Based Response. In *4th International Conference on Network and System Security (NSS)* (September 2010), pp. 207–215.

[78] Kanoun, W., Cuppens-Boulahia, N., Cuppens, F., Dubus, S., and Martin, A. Success Likelihood of Ongoing Attacks for Intrusion Detection and Response Systems. In *International Conference on Computational Science and Engineering (CSE '09)* (August 2009), vol. 3, pp. 83–91.

[79] Kanoun, W., Cuppens-Boulahia, N., Cuppens, F., Dubus, S., and Martin, A. Intelligent response system to mitigate the success likelihood of ongoing attacks. In *6th International Conference on Information Assurance and Security (IAS)* (Aug 2010), pp. 99–105.

[80] Kanoun, W., Samarji, L., Cuppens-Boulahia, N., Dubus, S., and Cuppens, F. Towards a Temporal Response Taxonomy. In *Data Privacy Management and Autonomous Spontaneous Security*, R. Di Pietro, J. Herranz, E. Damiani, and R. State, Eds., vol. 7731 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013, pp. 318–331.

[81] Karp, R. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, R. Miller and J. Thatcher, Eds. Plenum Press, 1972, pp. 85–103.

[82] Khachiyan, L., Boros, E., Borys, K., Elbassioni, K., and Gurvich, V. Generating All Vertices of a Polyhedron Is Hard. *Discrete & Computational Geometry 39*, 1-3 (2008), 174–190.

[83] Kheir, N., Cuppens-Boulahia, N., Cuppens, F., and Debar, H. A Service Dependency Model for Cost-Sensitive Intrusion Response. In *Computer Security – ESORICS 2010*, D. Gritzalis, B. Preneel, and M. Theoharidou, Eds., vol. 6345 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010, pp. 626–642.

[84] Kheir, N., Debar, H., Cuppens-Boulahia, N., Cuppens, F., and Viinikka, J. Cost Evaluation for Intrusion Response Using Dependency Graphs. In *International Conference on Network and Service Security (N2S '09)* (June 2009), pp. 1–6.

[85] Khurana, H., Basney, J., Bakht, M., Freemon, M., Welch, V., and Butler, R. Palantir: A Framework for Collaborative Incident Response and Investigation. In *Proceedings of the 8th Symposium on Identity and Trust on the Internet (IDtrust '09)* (New York, NY, USA, 2009), ACM, pp. 38–51.

[86] Kim, H. K., Im, K. H., and Park, S. C. DSS for computer security incident response applying CBR and collaborative response. *Expert Systems with Applications 37*, 1 (2010), 852 – 870.

[87] Klein, G., Rogge, H., Schneider, F., Toelle, J., Jahnke, M., and Karsch, S. Response Initiation in Distributed Intrusion Response Systems for Tactical MANETs. In *European Conference on Computer Network Defense (EC2ND)* (October 2010), pp. 55–62.

[88] Klein, G., Tolle, J., and Martini, P. From detection to reaction - A holistic approach to cyber defense. In *Defense Science Research Conference and Expo (DSR)* (August 2011), pp. 1–4.

[89] Kotenko, I., and Stepashkin, M. Analyzing network security using malefactor action graphs. *International Journal of Computer Science and Network Security 6*, 6 (2006), 226–235.

[90] Kruegel, C., Valeur, F., and Vigna, G. *Intrusion Detection and Correlation - Challenges and Solutions*, vol. 14 of *Advances in Information Security*. Springer, 2005.

[91] Lalanda, P. Two complementary patterns to build multi-expert systems. In *Pattern Languages of Programs* (1997).

[92] Lee, S.-H., Lee, H.-H., and Noh, B.-N. A Rule-Based Intrusion Alert Correlation System for Integrated Security Management. In *Proceedings of the 4th International Conference on Computational Science (ICCS 2004)*, M. Bubak, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, Eds. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 365–372.

[93] Lee, W., Miller, M., Stolfo, S. J., Fan, W., and Zadok, E. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security 10* (2002), 2002.

[94] Li, W., and Tian, S. An ontology-based intrusion alerts correlation system. *Expert Systems with Applications 37*, 10 (2010), 7138 – 7146.

[95] Lippmann, R., Haines, J. W., Fried, D. J., Korba, J., and Das, K. Analysis and Results of the 1999 DARPA Off-Line Intrusion Detection Evaluation. In *Proceedings of the 3rd International Workshop on Recent Advances in Intrusion Detection (RAID '00)* (London, UK, UK, 2000), Springer-Verlag, pp. 162–182.

[96] Lodi, G., Aniello, L., Di Luna, G. A., and Baldoni, R. An Event-based Platform for Collaborative Threats Detection and Monitoring. *Information Systems 39* (January 2014), 175–195.

[97] Lowe, G. Towards a completeness result for model checking of security protocols. In *Proceedings of 11th IEEE Computer Security Foundations Workshop* (June 1998), pp. 96–105.

[98] Luckham, D. *The power of events*, vol. 204. Addison-Wesley Reading, 2002.

[99] Manganiello, F., Marchetti, M., and Colajanni, M. Multistep Attack Detection and Alert Correlation in Intrusion Detection Systems. In *Information Security and Assurance*, T.-h. Kim, H. Adeli, R. Robles, and M. Balitanas, Eds., vol. 200 of *Communications in Computer and Information Science*. Springer Berlin Heidelberg, 2011, pp. 101–110.

[100] Mansour, Y. An Early Malware Detection, Correlation, and Incident Response System with Case Studies. GIAC (GCIA) Gold Certification Version1.0, SANS Institute, January 2014.

[101] Mateos, V., Villagrá, V. A., Romero, F., and Berrocal, J. Definition of response metrics for an ontology-based Automated Intrusion Response Systems. *Computers & Electrical Engineering 38*, 5 (2012), 1102 – 1114.

[102] Miller, P., and Inoue, A. Collaborative Intrusion Detection System. In *22nd International Conference of the North American Fuzzy Information Processing Society (NAFIPS)* (July 2003), pp. 519–524.

[103] Mitropoulos, S., Patsos, D., and Douligeris, C. On Incident Handling and Response: A state-of-the-art approach. *Computers & Security 25*, 5 (2006), 351 – 370.

[104] Mu, C., and Li, Y. An intrusion response decision-making model based on hierarchical task network planning. *Expert Systems with Applications 37*, 3 (2010), 2465 – 2472.

[105] Mu, C. P., Li, X. J., Huang, H. K., and Tian, S. F. Online Risk Assessment of Intrusion Scenarios Using D-S Evidence Theory. In *Proceedings of the 13th European Symposium on Research in Computer Security (ESORICS08)*, S. Jajodia and J. Lopez, Eds. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 35–48.

[106] Ning, P., Cui, Y., Reeves, D. S., and Xu, D. Techniques and Tools for Analyzing Intrusion Alerts. *ACM Transactions on Information and System Security 7*, 2 (May 2004), 274–318.

[107] Nojiri, D., Rowe, J., and Levitt, K. Cooperative response strategies for large scale attack mitigation. In *Proceedings of DARPA Information Survivability Conference and Exposition* (April 2003), vol. 1, pp. 293–302.

[108] Obrst, L., Chase, P., and Markeloff, R. Developing an Ontology of the Cyber Security Domain. In *STIDS* (2012), P. C. G. da Costa and K. B. Laskey, Eds., vol. 966 of *CEUR Workshop Proceedings*, CEUR-WS.org, pp. 49–56.

[109] Ortega-Arjona, J. L., and Fernandez, E. B. The secure blackboard pattern. In *Proceedings of the 15th Conference on Pattern Languages of Programs* (2008), ACM, p. 22.

[110] Ossenbühl, S., Steinberger, J., and Baier, H. Towards Automated Incident Handling: How to Select an Appropriate Response against a Network-Based Attack? In *9th International Conference on IT Security Incident Management IT Forensics (IMF)* (May 2015), pp. 51–67.

[111] Peter, A. S., S, P., and Ekert, L. V. An Ontology for Network Security Attacks. In *In Proceedings of the 2nd Asian Applied Computing Conference (AACC'04)* (2004), Springer-Verlag, pp. 317–323.

[112] Porras, P. A., Fong, M. W., and Valdes, A. A Mission-Impact-Based Approach to INFOSEC Alarm Correlation. In *Recent Advances in Intrusion Detection: Proceedings of th 5th International Symposium (RAID 2002)*, A. Wespi, G. Vigna, and L. Deri, Eds. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 95–114.

[113] Porras, P. A., and Neumann, P. G. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *National Information Systems Security Conference* (October 1997).

[114] Quereilhac, A., Lacage, M., Freire, C., Turletti, T., and Dabbous, W. NEPI: An integration framework for Network Experimentation. In *19th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)* (September 2011), pp. 1–5.

[115] Ragsdale, D., Carver, C., Humphries, J., and Pooch, U. Adaptation techniques for intrusion detection and intrusion response systems. In *IEEE International Conference on Systems, Man, and Cybernetics* (2000), vol. 4, pp. 2344–2349.

[116] Ragusa, C., Robinson, P., and Svorobej, S. A Framework for Modeling and Execution of Infrastructure Contention Experiments. In *2nd Internation Workshop on Measurement-based Experimental Research, Methodology and Tools* (2013).

[117] Rakotoarivelo, T., Ott, M., Jourjon, G., and Seskar, I. OMF: A Control and Management Framework for Networking Testbeds. In *ACM Operating Systems Review (OSR)* (1 2010), pp. 54–59.

[118] Ryutov, T., Neuman, C., Dongho, K., and Li, Z. Integrated Access Control and Intrusion Detection for Web Servers. *IEEE Transactions on Parallel and Distributed Systems 14*, 9 (September 2003), 841–850.

[119] Ryutov, Tatyana and Neuman, Clifford and Kim, Dongho. Dynamic authorization and intrusion response in distributed systems. In *Proceedings of the DARPA Information Survivability Conference and Exposition* (2003), vol. 1, IEEE, pp. 50–61.

[120] Sadighian, A., Fernandez, J., Lemay, A., and Zargar, S. ONTIDS: A Highly Flexible Context-Aware and Ontology-Based Alert Correlation Framework. In *Foundations and Practice of Security*, J. L. Danger, M. Debbabi, J.-Y. Marion, J. Garcia-Alfaro, and N. Zincir Heywood, Eds., Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 161–177.

[121] Sadoddin, R., and Ghorbani, A. Alert Correlation Survey: Framework and Techniques. In *Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services (PST '06)* (New York, NY, USA, 2006), ACM, pp. 37:1–37:10.

[122] Scarfone, K., and Mell, P. Guide to Intrusion Detection and Prevention Systems (IDPS). Special publication 800-94, National Institute of Standards and Technology (NIST), February 2007.

[123] Schnackenberg, D., Djahandari, K., and Sterne, D. Infrastructure for intrusion detection and response. In *Proceedings DARPA Information Survivability Conference and Exposition (DISCEX '00)* (2000), vol. 2, pp. 3–11.

[124] Schnackengerg, D., Holliday, H., Smith, R., Djahandari, K., and Sterne, D. Cooperative Intrusion Traceback and Response Architecture (CITRA). In *Proceedings of DARPA Information Survivability Conference and Exposition II (DISCEX '01)* (2001), vol. 1, pp. 56–68.

[125] Sekar, R., Gupta, A., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H., and Zhou, S. Specification-based Anomaly Detection: A New Approach for Detecting Network Intrusions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS '02)* (New York, NY, USA, 2002), ACM, pp. 265–274.

[126] Shahrah, A. Y., Hossain, M. A., and Alghamdi, A. S. A collaboration architecture for distributed smart surveillance systems based on DoDAF 2.0. In *International Joint Conference on Computer Science and Software Engineering (JCSSE)* (May 2012), pp. 305–310.

[127] Shahrah, A. Y., Hossain, M. A., and Alghamdi, A. S. Alert-response for distributed surveillance: DODAF-based services and systems. In *16th International Conference on Advanced Communication Technology* (February 2014), pp. 949–954.

[128] Shameli-Sendi, A., Cheriet, M., and Hamou-Lhadj, A. Taxonomy of intrusion risk assessment and response system. *Computers & Security 45*, 0 (2014), 1 – 16.

[129] Shameli-Sendi, A., and Dagenais, M. ORCEF: online response cost evaluation framework for intrusion response system. *Journal of Network and Computer Applications 55* (2015), 89–107.

[130] Shameli-Sendi, A., Ezzati-jivan, N., Jabbarifar, M., and Dagenais, M. Intrusion Response Systems: Survey and Taxonomy. *IJCSNS International Journal of Computer Science and Network Security 12*, 1 (January 2012).

[131] Shirey, R. W. Internet Security Glossary. Internet RFC 2828, May 2000.

[132] Shittu, R., Healing, A., Ghanea-Hercock, R., Bloomfield, R., and Muttukrishnan, R. OutMet: A new metric for prioritising intrusion alerts using correlation and outlier analysis. In *IEEE 39th Conference on Local Computer Networks (LCN)* (September 2014), pp. 322–330.

[133] Silva, O., Garcia, A., and Lucena, C. The Reflective Blackboard Pattern: Architecting Large Multi-agent Systems. In *Software Engineering for Large-Scale Multi-Agent Systems*, A. Garcia, C. Lucena, F. Zambonelli, A. Omicini, and J. Castro, Eds., vol. 2603 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003, pp. 73–93.

[134] Snort. https://www.snort.org/.

[135] Somayaji, A., and Forrest, S. Automated Response Using System-call Delays. In *Proceedings of the 9th Conference on USENIX Security Symposium (SSYM'00)* (Berkeley, CA, USA, 2000), vol. 9, USENIX Association.

[136] SOME/IP Analyzer. https://github.com/Egomania/SOME-IP_Analyzer.

[137] SOME/IP Generator. https://github.com/Egomania/SOME-IP_Generator.

[138] Song, J., Cadar, C., and Pietzuch, P. SymbexNet: Testing Network Protocol Implementations with Symbolic Execution and Rule-Based Specifications. *IEEE Transactions on Software Engineering 40*, 7 (2014), 695–709.

[139] Song, J., Ma, T., Cadar, C., and Pietzuch, P. Rule-Based Verification of Network Protocol Implementations Using Symbolic Execution. In *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)* (July 2011), pp. 1–8.

[140] Sowa, A., and Fedtke, S. *Metriken - Der Schlüssel Zum Erfolgreichen Security und Compliance Monitoring: Design, Implementierung und Validierung in der Praxis*. Vieweg + Teubner Praxis. Vieweg+Teubner Verlag, 2011.

[141] Stakhanova, N., Basu, S., and Wong, J. A Cost-Sensitive Model for Preemptive Intrusion Response Systems. In *21st International Conference on Advanced Information Networking and Applications (AINA '07)* (May 2007), pp. 428–435.

[142] Stakhanova, N., Basu, S., and Wong, J. A Taxonomy of Intrusion Response Systems. *International Journal of Information and Computer Security 1*, 2 (January 2007), 169–184.

[143] Stallings, W. *Network Security Essentials: Applications and Standards*. William Stallings books on computer and data communications technology. Prentice Hall, 2007.

[144] Sterne, D., Djahandari, K., Wilson, B., Babson, B., Schnackenberg, D., Holliday, H., and Reid, T. Autonomic Response to Distributed Denial of Service Attacks. In *Recent Advances in Intrusion Detection*, W. Lee, L. Mé, and A. Wespi, Eds., vol. 2212 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2001, pp. 134–149.

[145] Strasburg, C., Stakhanova, N., Basu, S., and Wong, J. A Framework for Cost Sensitive Assessment of Intrusion Response Selection. In *33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC '09)* (2009), vol. 1, pp. 355–360.

[146] Strasburg, C., Stakhanova, N., Basu, S., and Wong, J. S. Intrusion Response Cost Assessment Methodology. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security (ASIACCS '09)* (New York, NY, USA, 2009), ACM, pp. 388–391.

[147] Strasburg, C. R., Stakhanova, N., Basu, S., and Wong, J. S. The methodology for evaluating response cost for intrusion response systems. Technical Report TR08-12, Iowa State University, 2008.

[148] Sultana, S., Midi, D., and Bertino, E. Kinesis: A Security Incident Response and Prevention System for Wireless Sensor Networks. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems (SenSys '14)* (New York, NY, USA, 2014), ACM, pp. 148–162.

[149] Tanachaiwiwat, S., Hwang, K., and Chen, Y. Adaptive intrusion response to minimize risk over multiple network attacks. *ACM Trans on Information and System Security 19* (2002), 1–30.

[150] Ten, C.-W., Manimaran, G., and Liu, C.-C. Cybersecurity for Critical Infrastructures: Attack and Defense Modeling. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 40*, 4 (July 2010), 853–865.

[151] Toth, T., and Kruegel, C. Evaluating the impact of automated intrusion response mechanisms. In *Proceedings of the 18th Annual Computer Security Applications Conference* (2002), pp. 301–310.

[152] Tung, B., Staniford-Chen, S., Porras, P., Kahn, C., Schnackenberg, D., Feiertag, R., and Stillman, M. The Common Intrusion Detection Framework - Data Formats. Internet-draft, Internet Engineering Task Force, March 1998.

[153] Undercoffer, J., Joshi, A., and Pinkston, J. Modeling Computer Attacks: An Ontology for Intrusion Detection. In *Recent Advances in Intrusion Detection*, G. Vigna, C. Kruegel, and E. Jonsson, Eds., vol. 2820 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003, pp. 113–135.

[154] Undercoffer, J., and Pinkston, J. Modeling computer attacks: a target-centric ontology for intrusion detection. In *Proceedings of the 2002 UMBC Center for Architectures for Data-Driven Information Processing Research Symposium* (2002).

[155] Valdes, A., and Skinner, K. Probabilistic Alert Correlation. In *Recent Advances in Intrusion Detection (RAID 2001)* (2001), no. 2212 in Lecture Notes in Computer Science, Springer-Verlag.

[156] Valeur, F., Vigna, G., Kruegel, C., and Kemmerer, R. Comprehensive approach to intrusion detection alert correlation. *IEEE Transactions on Dependable and Secure Computing 1*, 3 (July 2004), 146–169.

[157] van Heerden, R., Irwin, B., Burke, I. D., and Leenen, L. A Computer Network Attack Taxonomy and Ontology. *IJCWT 2*, 3 (2012), 12–25.

[158] van Heerden, R., Leenen, L., and Irwin, B. Automated classification of computer network attacks. In *International Conference on Adaptive Science and Technology (ICAST)* (Nov 2013), pp. 1–7.

[159] Varshovi, A., and Sadeghiyan, B. Ontological classification of network denial of service attacks: basis for a unified detection framework. *Scientia Iranica. Transaction D, Computer Science & Engineering, Electrical 17*, 2 (2010).

[160] Vorobiev, A., and Han, J. H. J. Security Attack Ontology for Web Services. In *2006 Semantics, Knowledge and Grid, Second International Conference on* (Nov 2006), pp. 42–42.

[161] Völker, L. SOME/IP – Die Middleware für Ethernet-basierte Kommunikation. *Hanser automotive networks* (November 2013).

[162] Wachs, M., Herold, N., Posselt, S.-A., Dold, F., and Carle, G. GPLMT: A Lightweight Experimentation and Testbed Management Framework. In *Passive and Active Measurement: 17th International Conference, PAM 2016, Heraklion, Greece, March 31 - April 1, 2016. Proceedings*, T. Karagiannis and X. Dimitropoulos, Eds. Springer International Publishing, Cham, 2016, pp. 165–176.

[163] Wagner, M. O. W. *Gefahrenerkennung in Konfigurationen verteilter Systeme*. Dissertation, Technische Universität München, München, 2016.

[164] Wang, S.-H., Tseng, C., Levitt, K., and Bishop, M. Cost-Sensitive Intrusion Responses for Mobile Ad Hoc Networks. In *Recent Advances in Intrusion Detection*, C. Kruegel, R. Lippmann, and A. Clark, Eds., vol. 4637 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2007, pp. 127–145.

[165] Wang, X., Reeves, D. S., and Felix, S. Tracing Based Active Intrusion Response. In *In Journal of Information Warefare* (2001).

[166] White, G., Fisch, E., and Pooch, U. Cooperating security managers: a peer-based intrusion detection system. *IEEE Network 10*, 1 (1996), 20–23.

[167] Wu, Y., and Liu, S. A Cost-Sensitive Method for Distributed Intrusion Response. In *12th International Conference on Computer Supported Cooperative Work in Design (CSCWD)* (2008), pp. 760–764.

[168] Wu, Y.-S., Foo, B., Mao, Y.-C., Bagchi, S., and Spafford, E. H. Automated Adaptive Intrusion Containment in Systems of Interacting Services. *Computer Networks 51*, 5 (April 2007), 1334–1360.

[169] Xiao, F., Jin, S., and Li, X. A novel data mining-based method for alert reduction and analysis. *Journal of networks 5*, 1 (January 2010), 88–97.

[170] Yongyong, S. Decision Algorithm for Multi-Agent Intelligent Decision Support System based on Blackboard. *Information Technology Journal 12*, 21 (2013), 6235–6240.

[171] Yu, D., and Frincke, D. Improving the quality of alerts and predicting intruder's next goal with Hidden Colored Petri-Net. *Computer Networks 51*, 3 (2007), 632–654.

[172] Yue, W. T., and Çakanyıldırım, M. A cost-based analysis of intrusion detection system configuration under active or passive response. *Decision Support Systems 50*, 1 (2010), 21 – 31.

[173] Zaghdoud, M., and Al-Kahtani, M. S. Contextual Fuzzy Cognitive Map for Intrusion Response System. *International Journal of Computer and Information Technology (IJCIT) 2*, 3 (May 2013), 471–478.

[174] Zan, X., Gao, F., Han, J., Liu, X., and Zhou, J. A hierarchical and factored POMDP based automated intrusion response framework. In *2nd International Conference on Software Technology and Engineering (ICSTE)* (October 2010), vol. 2.

[175] Zan, X., Gao, F., Han, J., Liu, X., and Zhou, J. NAIR: A novel automated intrusion response system based on decision making approach. In *IEEE International Conference on Information and Automation (ICIA)* (June 2010), pp. 543–548.

[176] Zang, T., chun Yun, X., and 0002, Y. Z. A Survey of Alert Fusion Techniques for Security Incident. In *WAIM* (2008), IEEE, pp. 475–481.

[177] Zhang, Z., Ho, P.-H., and He, L. Measuring IDS-estimated attack impacts for rational incident response: A decision theoretic approach. *Computers & Security 28*, 7 (2009), 605 – 614.

[178] Zhu, B., and Ghorbani, A. A. Alert correlation for extracting attack strategies. *International Journal of Network Security 3*, 3 (2006), 244–258.

[179] Zhu, B., and Ghorbani, A. A. Alert correlation for extracting attack strategies. *International Journal of Network Security 3*, 3 (2006), 244–258.

[180] Znaidi, W., Minier, M., and Babau, J.-P. An Ontology for Attacks in Wireless Sensor Networks. Research Report RR-6704, INRIA, 2008.

[181] Zonouz, S., Khurana, H., Sanders, W., and Yardley, T. RRE: A Game-Theoretic Intrusion Response and Recovery Engine. *IEEE Transactions on Parallel and Distributed Systems 25*, 2 (February 2014), 395–406.

ISBN 978-3-937201-59-7
DOI 10.2313/NET-2017-05-2

ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)