FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Anomaly Detection Through Massive Event Correlation in ICT Networks

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Ivo Friedberg
Matrikelnummer 0725830

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: DDr. Florian Skopik

Wien, 22.04.2014

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Anomaly Detection Through Massive Event Correlation in ICT Networks

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Ivo Friedberg
Registration Number 0725830

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance: DDr. Florian Skopik

Vienna, 22.04.2014

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Ivo Friedberg
Sommerergasse 2A, 1130 Vienna

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

I would like to thank everyone who supported me during the work on this thesis.

First, I would like to thank Florian Skopik. Not only for giving me the opportunity to work on this project, but also for his endless support during the course of this work and beyond.

I further want to thank Prof. Wolfgang Kastner for his constant and very productive feedback on this thesis work, as well as for his non-bureaucratic but result driven attitude.

My thanks also go to Giuseppe Settanni for his time encouraging me and pushing me during the tiring evaluation phase, and to all the other colleagues at the Austrian Institute of Technology, for the great and productive work environment. I was very lucky to be a part of your team.

Finally, I also want to thank my family for their support and power of endurance during the whole time of my studies.

Abstract

As Information and Communication Technology (ICT) networks and their complexity evolved, so did the goals and the technical processes of attacks. Recent security incidents show that current security mechanisms are often not sufficient to prohibit targeted attacks. If an attack on a system is successful timely detection is critical to mitigate its impact. With the increasing use of common Internet protocols in connection with Supervisory Control and Data Acquisition (SCADA) systems, industrial networks are exposed to the same threats as corporate networks. This work proposes a novel anomaly detection approach, based on the timely correlation and analysis of log-files from various sources in a monitored network. The framework builds a system model that describes the normal behaviour of the different components in the monitored network. It does not rely on any information about syntax or semantics of the processed log-lines. Instead, the model is generated based on the processed information and constantly evolves while the system is monitoring the network. Using data from a controlled ICT network, this thesis shows that the generated model distinguishes meaningful subsets of log-files, and is able to model complex implications between different network components. An evaluation based on semi-synthetic log-data demonstrates the application of the approach in common ICT networks. Additionally, real-world data from a utility provider is used to demonstrate the system's application in the domain of SCADA systems.

Kurzfassung

Mit der Entwicklung und dem verbreiteten Einsatz von Informations- und Kommunikationsnetzwerken stieg auch der Umfang und die Komplexität von Attacken in diesen Netzwerken. Jüngste Vorfälle zeigen, dass aktuelle Sicherheitssysteme nicht ausreichen, um gezielte und gut vorbereitete Angriffe zu verhindern. War eine Attacke erst erfolgreich, ist eine zeitnahe Erkennung wichtig um die Auswirkungen einzudämmen. Mit der verbreiteten Anwendung von Internet Protokollen im Bereich von „Supervisory and Data Acquisistion“ (SCADA) Systemen sind industrielle Netzwerke oft schon jetzt den selben Bedrohungen ausgesetzt wie herkömmliche, vernetzte Systeme. Diese Arbeit präsentiert einen neuartigen Anomalieerkennungsansatz. Dieser basiert auf der zeitlichen Korrelaion von Log-Datein verschiedener Systeme in einem überwachten Netzwerk. Das System generiert ein Modell, welches das normale Verhalten verscheidener Komponenten im überwachten System beschreibt. Dabei verlässt sich das System nicht auf vordefinierte Regeln und benötigt auch kein Wissen über Syntax oder Semantik der Log-Zeilen, die es verarbeiten muss. Stattdessen wird das Model auf Basis der verarbeiteten Zeilen erstellt und fortlaufend weiterentwickelt, solange das Netzwerk überwacht wird. Ein vollständig kontrolliertes Netzwerk wird verwendet, um Testdaten zu generieren, die frei von Anomalien sind. Anhand dieser Testdaten wird gezeigt, dass das generierte Model in der Lage ist, aussagekräftige Teilmengen der verarbeiteten Zeilen zu unterscheiden. Diese Teilmengen werden weiters verwendet, um zu zeigen, dass auch Implikationen zwischen Ereignissen verschiedener, verteilter Komponenten erkannt werden. Basierend auf einem semi-synthetischen Datensatz, in dem Anomalien eingebaut sind, wird die Anwendbarkeit des Ansatzes in herkömmlichen IT-Netzwerken demonstriert. Weiters wird anhand eines Datensatzes aus dem Produktivsystem eines österreichischen Energieanbieters die Eignung im SCADA Bereich gezeigt.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Global connectivity is the core principle of our information age. From an information provision point of view, distances seem to shrink since information can be immediately accessed from all over the world. We are used to, and highly dependent on, information and communication services, but they increasingly motivate a certain criminal potential. As ICT networks and their complexity have evolved in recent years, so have the goals and the technical progress of attacks. The motivation for attacks has changed from immediate monetary gain to stealing proprietary information or personal details [22].

Since the emergence of the first ICT networks significant effort went into securing critical assets. Currently, one can choose from a variety of security solutions that target different attack schemes at different levels in the network: Firewalls that filter the network traffic at border crossings between sub-networks, scanners, checking binaries and executables for suspicious behaviour or Intrusion Detection Systems (IDSs) that monitor events all over the network and check them against predefined rules for anomalies. Additionally, most companies have various guidelines and processes in place to decrease the chance of human failure.

However, recent attacks like Operation Aurora [14] or Operation Shady RAT [1] demonstrate that the current security mechanisms are insufficient to prevent unique sophisticated and tailored attacks, also known as Advanced Persistent Threats (APT). Furthermore, these advanced attacks raise the question if it is even possible to prevent intrusions with reasonable certainty [1]. More likely is a scenario where sophisticated attacks are accepted to some degree. A new systematic approach is to know about an intrusion as fast as possible to start efficient countermeasures. IDSs aim at detecting intrusions, but again recent events showed their insufficiency when it comes to detecting APTs.

The goal of this thesis is to evaluate and improve a novel intrusion detection algorithm [19] [18]. The algorithm aims at extending the functionality of existing IDSs to enable a timely detection of novel attacks to a system, so the victim is able to address them in an effective way.

1.2 Problem Statement

Timely detection of intrusions is one major goal when securing critical systems in the future. As current security solutions are often not sufficient to prevent sophisticated and tailored attacks, novel approaches are required. In modern ICT systems many unnoticed relationships between different applications or components exist. Users that use the same passwords on multiple services, or firewall rules that are evaluated prior to a webserver request are examples of obvious relations. Looking at kernel level actions one can easily think of more subtle relationships between actions taken on different components in a network. We see many of these neglected relationships as the weak spot abused by attackers to compromise systems. However, we argue that it is not possible to exploit a system without violating some of these implicit implications between events in a system. We further argue that significant violations of these implications can be monitored and used to detect intrusions in a way that is hard – if not impossible – to circumvent by an attacker.

This thesis presents a novel intrusion detection algorithm. It exploits system-inherent event relationships gathered from log files of different sources in the network, in order to detect anomalies and intrusions in the infrastructure. The following hypotheses will be evaluated during the course of this work:

i It is possible to generate a meaningful system model that describes relationships between distributed components in a network based on the log information that is generated at each component. The generation process does not rely on pre-defined knowledge about syntax and semantics of the processed log information.

ii This system model can be used to detect anomalous behaviour in the monitored network promptly.

iii Detection works especially well in industrial networks due to the well defined communication channels and the well structured messages that are sent.

1.3 Methodology and Structure

The contributions of this thesis are as follows:

- A formal model definition of a novel anomaly detection approach based on log-line analysis is given. The approach correlates events that are generated by detailed log-line analysis without prior knowledge of syntax or semantics of the processed log-lines.
- An existing prototype implementation of basic concepts of the proposed approach is extended. New features that are implemented are:
  - A balancing algorithm that ensures even distribution of extracted knowledge from the input dataset over the log-lines of that set.
  - Aging functionality that enables automatically generated information which is deprecated or invalid to be deleted again.

  - Statistical analysis of hypotheses that were generated to model normal system behaviour. This analysis fulfils two goals: first, it enables a decision whether a hypothesis describes normal behaviour or not; second, it is used to detect anomalies in the system behaviour based on the set of proven hypotheses (modelled as rules).

- A sophisticated evaluation of the extended prototype is performed. This evaluation includes the following aspects:
  - After identifying the critical system parameters, optimal settings are discussed for each of these parameters.
  - A qualitative analysis is applied to evaluate the system model that is generated from the processed input. This evaluation includes the coverage of the input file by the model as well as the quality of the generated rules.
  - An evaluation is performed about the ability of the approach to detect anomalies in the system behaviour, resulting from authentic attack scenarios.

The remainder of this thesis is structured as follows: Section 2.1 provides details about SCADA systems, as well as an overview of current research in SCADA security. In order to show the broader applicability (in addition to Web-based systems) of the new approach, one focus of the evaluation lies on the system's applicability in the domain of Supervisory Control and Data Acquisition (SCADA) systems. Section 2.2 gives an overview on state of the art anomaly detection research, while Sect. 2.3 highlights current trends in the field of intrusion detection systems. Section 2.4 closes the chapter with information about credible means of evaluation and data acquisition.

Chapter 3 provides a formal definition of the novel anomaly detection approach. After an introduction, Sect. 3.2 formalises the core functionality of the approach, while Sect. 3.3 denotes the means of model extraction.

After the formal definition, Ch. 4 describes the prototype implementation in detail.

Ch. 5 describes the scenarios used to generate the datasets used in the evaluations. Valid evaluation is far from trivial. Commonly accepted datasets are nearly impossible to get, and reproducible test environments are similarly rare. In order to achieve a useful evaluation, we introduce a method for semi-synthetic dataset generation in Sect. 5.1 before describing a sample structure of a smart grid setup that isused for a proof of concept evaluation in the SCADA domain.

Ch. 6 states the performed evaluations of the prototype.

After the presentation and discussion of the evaluation results, Ch. 7 gives a conclusion and an outlook to finalise this thesis.

CHAPTER 2

Related Work

2.1 SCADA Systems

2.1.1 Introduction

In industrial settings (e.g., power grid management), automation of substations is essential. Three aspects have to be considered (see also [20]):

i *Data Acquisition* describes the task that collects all relevant data in the automated system. This data is acquired from different types of sensors throughout the system and can contain either analog or digital signals. The collected data can be used locally to fulfil monitoring tasks or it can be transmitted to central databases for later use by operators or analysts.

ii *Supervision* happens through processes or personnel that monitor status and condition of the different subsystems. Supervision happens based on the data acquired.

iii *Control* describes sending commands to different subsystems to manage their states.

A common approach to manage these aspects are supervisory control and data acquisition (SCADA) systems. Figure 2.1 gives a schematic overview of a traditional SCADA system. A critical infrastructure consists of a multitude of sensors and actuators. The *data acquisition* collects the data of the sensors; the *control* sends commands to the actuators to influence the state of a certain component in the system. A *remote terminal unit (RTU)* manages the distribution of messages between the *process network* and the different sensors and actuators. An RTU can also preprocess data locally. Therefore local data can automatically be exploited to trigger control commands. The process network collects data from different, geographically separated, sites of the system. It is also used to distribute the control commands to the right subsystems. A *central system* stores all data from different subsystems and provides the data for a controller or analyst. A central control room gives the operators the possibility to monitor the system status and trigger control commands.

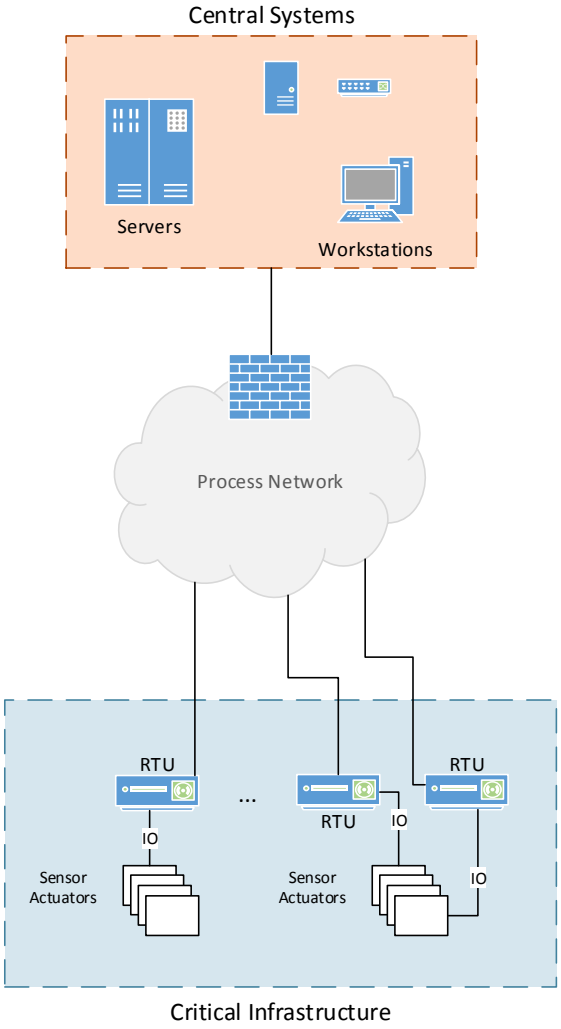


Figure 2.1: Overview of a traditional SCADA system.

2.1.2 Protocols

SCADA systems are not a novel approach to subsystem automation. Several protocols were developed over the years. Kanabar [12] identifies the three most popular protocols, out of a smart grid's perspective, to be the following:

IEC 60870-5 is a protocol standard for telecontrol, especially between two substations. The standard follows the Enhanced Performance Architecture (EPA) at ISO layer 3. It uses asynchronous serial telecontrol channel interfaces and is suitable for multiple configurations, e.g.,point-to-point or star. IEC 60870-5 defines:

- operating conditions
- electrical interfaces

- performance requirements
- data transmission protocols

DNP3 is a protocol developed in 1993 to achieve open and standard-based interoperability between:

- Remote Terminal Units (RTUs)
- Intelligent Electronic Devices (IEDs)
- Master Stations

It follows the EPA on OSI level 3 (similar to the IEC 60870-5 standard) and is also designed for telecontrol applications.

IEC 61850-90-1 focuses on the station bus. In ten parts the standard series defines:

- general and functional requirements of the substation communication system
- an XML-based substation configuration language
- common data and service models
- mappings from different data objects into manufacturing message specification (MMS)

2.1.3 Security Considerations

Supervisory Control And Data Acquisition (SCADA) systems get more and more connected to corporate networks and ultimately the Internet (see Fig. 2.2). This is done in the hope of increasing productivity. But early security considerations in SCADA systems are not sufficient to tackle the threats opposed by the Internet. Pires [15] gives an overview over the risks introduced by the interconnection of SCADA networks and corporate networks.

Early security considerations resulted in concepts like [15]:

- using a common password for user authentication
- considering protocols secure due to being proprietary
- transferring data in plain text

Those concepts hardly seem sufficient for corporate IT networks connected to the Internet. On the other hand existing security solutions for corporate IT systems like firewalls or intrusion detection systems are not applicable to SCADA networks without further consideration. SCADA networks have special demands. Firewall rules have to be adapted. Controllers and other embedded devices force resource restrictions upon security solutions. Furthermore control systems have a demand for 100 percent uptime and little room for latency [16]. All those constraints make it hard to use common solutions like e.g. encryption protocols.

A lot of work has already been done considering security aspects of SCADA system given the new context. Different security challenges in SCADA networks are pointed out by Igure [11]

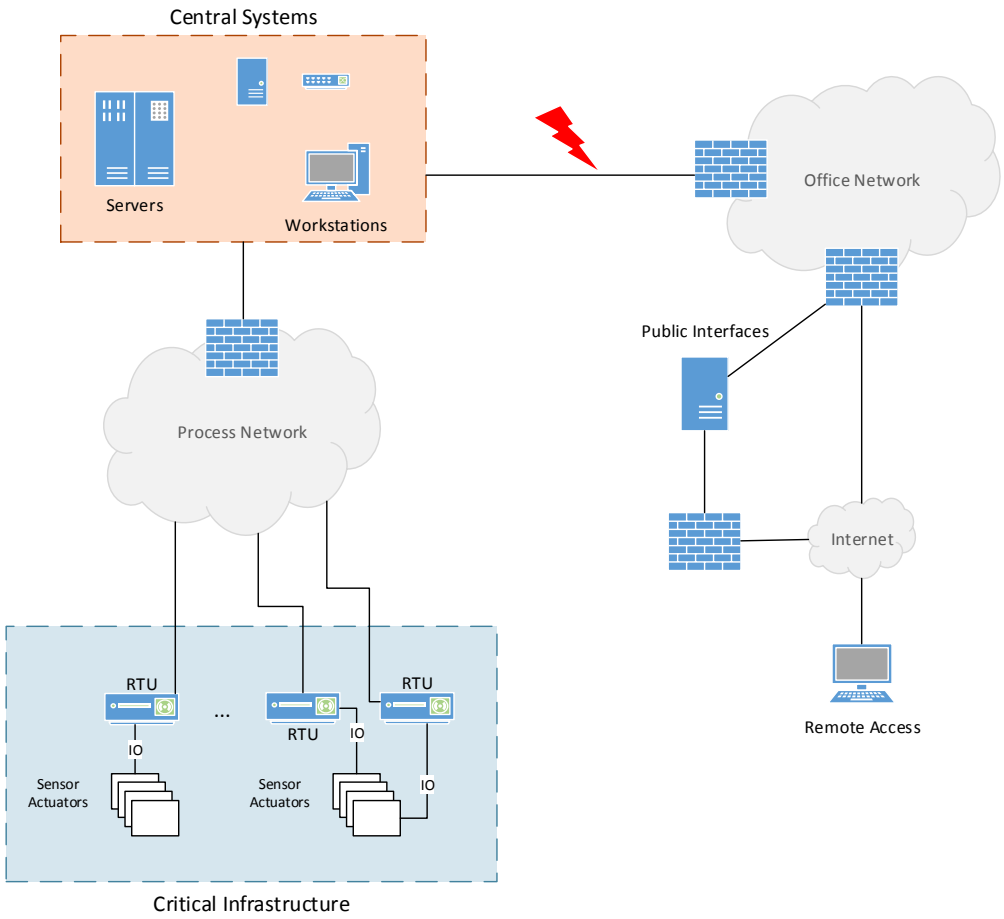


Figure 2.2: Overview of a traditional SCADA system that is connected to a corporate network.

including access control, firewalls and intrusion detection systems, protocol vulnerability assessment, cryptography and key management, device and operating system security as well as security management. Ten [24] proposes a framework to systematically evaluate vulnerabilities of a SCADA system. Onada [16] presents Support Vector Machine and Support Vector Data Description methods to classify events in the network. The authors point out, that common IDS solutions in IT systems are using signature based approaches but the lack of recorded cyber-attacks on SCADA systems makes those approaches hard to configure. Svendsen [21] gives an overview on modelling and detecting anomalies in SCADA networks. The work focuses on the fact that domain-specific knowledge can on one hand reduce error rates of anomaly detection algorithms. On the other hand this knowledge makes it easier for attackers to mask their malicious behaviour. Furthermore various approaches on anomaly detection and intrusion detection systems in SCADA networks were presented. Examples can be found in [3, 4, 6–8, 10, 21]

2.2 Anomaly Detection

Anomaly detection is an actively researched field in many domains. Chandola [9] identified the application domains as intrusion detection, fraud detection, medical and public health anomaly detection, image processing, anomaly detection in text data and anomaly detection in sensor networks apart from other not so prominent domains. In each domain we find different problems to solve, as well as different sets of data. In order to detect anomalies, a system has to use training data to define a ground truth about what is to be considered normal behaviour of a system [9,30]. The training data will be analysed to establish a notion about normality further used to mark patterns (also known as events or instances) in the data as normal or abnormal. Furthermore, it is important to note that systems evolve i.e., systems have to periodically reconstruct their notion of normality to adapt to this changes [30].

Chandola [9] distinguishes three kinds of anomalies.

Point Anomalies If a single event can be considered anomalous given the notion of normality we call it point anomaly.

Contextual Anomalies An event can be considered anomalous in respect to a given context. This contextual evaluation has to be encoded in the formulation of the problem. We can then deduce an anomaly given the events' behavioural attributes in its context. The same attributes might not be considered anomalous in another context.

Collective Anomalies If a series of events is considered anomalous we call it collective anomaly. Each event on its own in some other place in the stream might not be considered an anomaly. But the collective relation between them makes them anomalous.

Thottan [25] describes two main categories of anomalies in ICT networks. The first category describes anomalies due to system failures. The second category consists of security related problems resulting in anomalies.

Various classifications of anomaly detection approaches were taken by [9, 25, 29, 30] just to name a few. The broadest classification by Chandola [9] distinguishes six classes with various subclasses from all of the before described domains:

Classification Based Anomaly Detection Techniques try to classify every data instance as either normal or abnormal. Given a labelled training data set it generates a classifier later used to generate a label for every instance. Depending on the provided labels we can distinguish between *multi-class* (normal data has different labels) and *one-class* (either normal or abnormal as label) techniques. The general assumption of this approach says that it is possible to learn a classifier distinguishing between normal and abnormal data instances. Following techniques fall in this category:

- *Neural Networks-Based*
- *Bayesian Networks-Based*
- *Support Vector Machines-Based*
- *Rule-Based*

Nearest Neighbour-Based Anomaly Detection Techniques work on the general assumption that "normal data instances occur in dense neighbourhoods, while anomalies occur far from their closest neighbours" [9]. A metric has to be found serving as distance or similarity measure. This metric can then be used to generate an anomaly score for each data instance by two different means:

- *Using Distance to k^{th} Nearest Neighbour:* The anomaly score is the distance to its k^{th} nearest neighbour.
- *Using Relative Density:* The density of an instance's neighbourhood works as anomaly score where low density means that the instance is more likely an anomaly.

Clustering-Based Anomaly Detection Techniques try to generate clusters of similar data. The assumption here is that anomalies do not belong to any cluster, are far from the centre of the nearest cluster or occur only in a small cluster with low density while normal instances are near to the centre of a dense cluster. Then again an anomaly score can be used together with a threshold to identify anomalies.

Statistical Anomaly Detection Techniques assume the input data follows a stochastic model. This model usually describes normal behaviour. For each instance a statistical inference test decides if the data instance belongs to the model, hence is normal, or not. Two techniques can be observed.

- *Parametric Techniques* generally assume to know the underlying distribution and estimate its parameter from the training data.
- *Non-Parametric Techniques* do not assume to know the underlying distribution but try to learn it from the training data.

Information Theoretic Anomaly Detection Techniques assume that anomalies can be detected because they result in irregularities in the information content of the data set. Therefore, these techniques try to generate a model of normality about the information in the data in order to detect inconsistencies.

Spectral Anomaly Detection Techniques try to map the data space to a smaller subspace. They assume that there exists a subspace where normal data and anomalies can easily be identified and that there exists a transformation to get the data into this subspace.

Zhang [30] and Thottan [25] distinguish anomaly detection approaches with focus on ICT networks. They come up with subsets of the already described approaches before with [30] using *anomaly detection using statistics*, *anomaly detection using classifier*, *anomaly detection using machine learning* and *anomaly detection using finite state machines*. Thottan [25] distinguishes between *rule-based approaches*, *finite state machines*, *pattern matching* and *statistical analysis*.

Various challenges in anomaly detection are identified by [9]. The most prominent is of course to identify a complete notion of normality. This is made even harder by the fact, that there is only a fuzzy border between normal and abnormal behaviour. We further already mentioned the fact, that normal behaviour is evolving. When anomaly detection is used to detect

malicious activity it has to be pointed out, that an attacker disguises his actions by making them look normal. In some cases it might be possible he tampers the testing data or the system. Connected with the problem of getting a complete notion of normality is also the problem of getting representative, labelled training data and to differentiate noise from actual anomalies [5].

The number of papers describing novel or optimised mechanisms of anomaly detection in different contexts is far from countable and ever increasing. [13, 25, 28, 31, 32] are just some examples to show the diversity in the approaches. A good overview on anomaly detection in the domain of intrusion detection can be found in [29].

2.3 Intrusion Detection Systems

Nowadays various systems are in place in a corporate ICT network to ensure the three properties confidentiality, availability and integrity known as the security triangle. Any action attempting a violation of any of those properties can be seen as an intrusion [29]. Intrusion Detection Systems (IDSs) aim at detecting those intrusions to take actions from triggering warnings to actively preventing the attacker from causing further harm.

Literature as Yu [29] or Sabahi [17] classifies IDSs by different means. One way is to look at the data sources analysed or the scope the IDS looks at. Here we can differ between host based, network based and hybrid approaches. While host based approaches focus on the events on one single host to detect suspicious behaviour, network based approaches look at parts of networks and analyze the traffic and protocol data to detect intrusions [17, 29]. Sabahi [17] further classifies Hybrid approaches that use host and network data simultaneously as well as network behaviour analysis approaches monitoring traffic flows.

Another way to classify IDSs is to look at the types of intrusions they try to detect. We can differentiate between misuse and anomaly intrusion detection [2, 17, 29]. Misuse detection systems try to detect intrusions by matching events in the monitored domain against defined security policies. They can further be classified as signature based, rule based or based on state transitions [17]. Anomaly intrusion detection detects deviations of the events in the monitored domain from an as normal behaviour defined base line [29]. Here further differentiation can be taken between statistical based, distance based, rule based, profile based and model based methods [17]. Yu [29] has a different notion of differentiation regarding anomaly intrusion systems. Here the categories are statistical techniques, machine learning techniques, neural network techniques, data mining techniques and computer immunology techniques. Each category is supported by various projects.

2.4 Testing and Credible Evaluation

As shown by studies over the year [26, 27], empirical evaluation does not get the attention it deserves in research work in the field of computer science. This state did not change between 1995 and 2009 and there is no reason to assume a change in the years between 2009 and 2013. This poses a threat to the research community in the field of computer science since the validity and reliability of the studies have to be questioned [23]. The problematic condition can also be observed in the field of anomaly detection and intrusion detection systems [23].

But even given empirical evaluations, making meaning out of them or comparing different evaluations is far from trivial. A survey about experimental practices in the field of anomaly-based intrusion detection, taking 276 published studies in this area into account can be found in [23]. They identify two main problems regarding significant and comparable evaluation.

Test Data

One big problem is the lack of normalised, labelled test data from productive networks. Labelled data-sets can be found by the Defense Advanced Research Projects Agency (DARPA) from the years 1998, 1999 and 2000[1]. Another option is the knowledge and data mining (KDD) set which is derived from the DARPA set from 1998 . Due to their age those data sets cannot represent modern network traffic since network protocols and network topologies are constantly changing [5]. Another source is the MAVI traffic archive providing packet dumps from different measurement points. This lack of state-of-the-art test data results in researchers using data from university networks which can't be published due to privacy issues [5]. Of course it is then hard to compare results. Synthetic test data introduces the problem of proving completeness of an approach or its performance in a real-world set-up.

Metrics

Two common metrics to assess intrusion detection methods are established [23]. Behaviour can be classified as: 1) *true-positives*, describing attack instances detected as abnormal; 2) *false-positives* as events that got incorrectly classified abnormal; 3) *true-negatives* which are correctly ignored normal events; 4) *false-negatives* which are abnormal events classified as normal. Given this classification the common metrics are:

Detection Rate: ratio between detected attacks and occurred attacks

False-Positive Rate: ratio between incorrectly as anomaly classified events and normal events.

Those two metrics are often displayed together in a receiver operator characteristic (ROC) curve [23].

2.5 Summary

When the first industrial ICT networks came to existence they where isolated networks. There was no need for sophisticated security mechanisms on the communication layers and therefore most protocols do not inherently consider security issues. Due to modern trends, these networks get connected to corporate networks and ultimately the Internet what exposes them to the same threats common ICT infrastructures are exposed to.

IDSs aim at detecting violations regarding confidentiality, integrity or availability of a system. If an violation (i.e., an intrusion) is detected, different actions – from issuing warnings to actively preventing further harm – can be taken. IDSs can combine multiple intrusion detection

[1] see http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/

methods. One method is anomaly detection which can further be classified by the analysed data sources or by the way the data is analysed.

This thesis proposes a novel anomaly detection approach that aims at extending existing IDSs. It is a hybrid IDS approach that analyses host based data from various components in a network in order to make assumptions about relations between the components on the network level. Therefore, the approach can be considered as an *Information Theoretic Anomaly Detection Technique*. From the way the data sources are analysed and evaluated it is also a *Statistical Anomaly Detection Technique* with a parametric characteristic.

CHAPTER 3

Model Definition

The following chapter gives a detailed model definition of the before mentioned anomaly detection approach. It gives a short introduction in Sect. 3.1 followed by a detailed formal definition separated into two parts. Section 3.2 describes the core functionality of the approach while Sect. 3.3 defines algorithms to extract information used to build the system model.

3.1 Introduction

Most modern ICT components or services produce logging data to report events, internal state changes, and committed actions. Log files are a valuable source to establish situational awareness about the current status of ICT networks and to reproduce activities in the past; therefore they are human-readable. For protocols and applications with high market share, parsers exist to extract the key information from the lines. An anomaly detection system uses pre-defined rules to evaluate the extracted information and to detect attacks or to visualise the general system status to a human administrator. Section 2.3 gave an overview about the functionality and different types of IDS systems. These systems exist and they are widely and successfully used. But this approach brings two drawbacks:

i Pre-defined rules are often insufficient to detect unique or tailored attacks. Those rules are commonly known and, given enough effort is invested, can be tricked by tampering with the generated log lines.

ii For applications with low market share no sufficient parsers and rule sets are available[1]. General checks about the severity of messages can only provide a limited compensation for the lack of a sufficient rule set that encodes application specific operation sequences.

[1] One good example for such protocols and applications can be found in the SCADA domain as described in Sect. 2.1

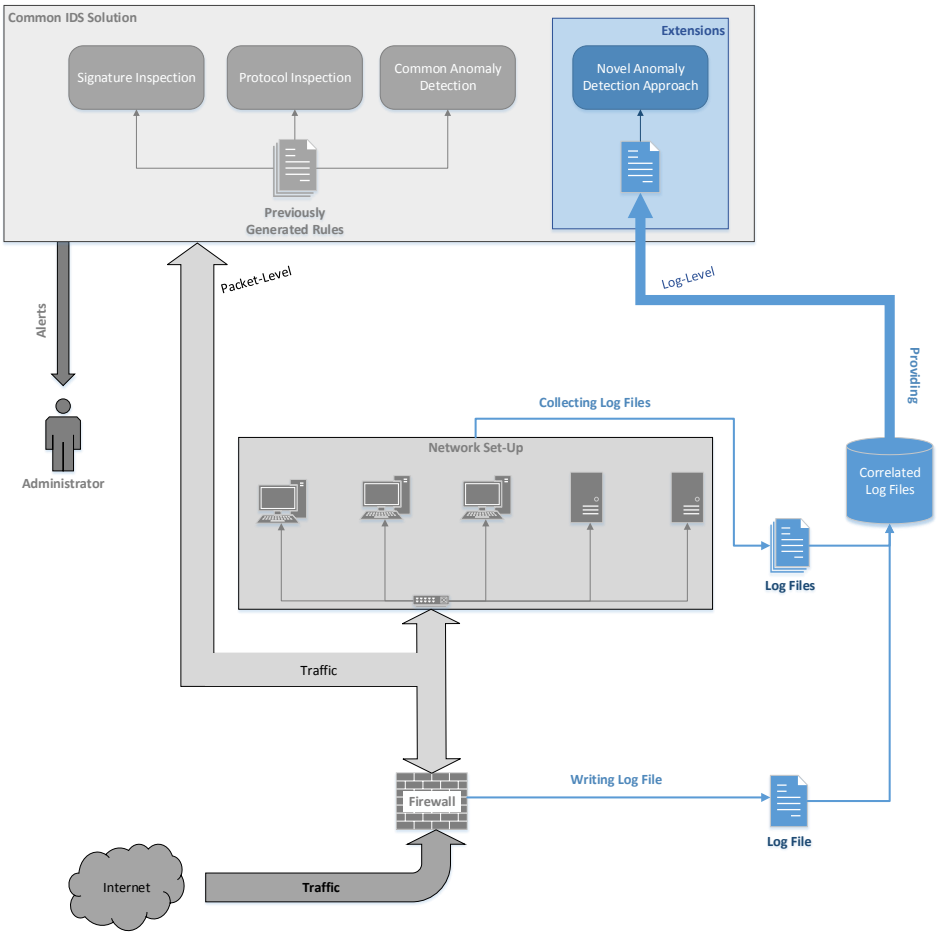


Figure 3.1: Approach positioning.

System administrators need to tackle those inadequacies, in order to increase security. But a system that is designed to replace all existing security mechanisms is no solution. Established ICT security solutions fulfil their purpose reasonably well for common security threats. As motivated in Ch. 1 they just lack the capability to tackle unique, tailored attacks because of their predictive structure.

As shown in Fig. 3.1 the proposed approach is therefore designed to extend common security mechanisms – especially „packet-level“ IDS systems – to improve their results. It further integrates seamlessly into the administrator's work-flow. Instead of replacing existing solutions or of establishing an additional security system, the approach aims at acting as an additional source for alerts in existing monitoring infrastructures. Similar to some existing mechanisms the proposed approach exploits log files. However it does not rely on existing knowledge about the syntax and the semantics of the lines. On the contrary, the system is constructing a model while processing the input. The system's model M is built by the following items (the model is described in more detail in Sect. 3.2):

Search-Patterns (P): Patterns are random substrings of the processed lines. Patterns are used to categorise information contained in a log-line.

Event Classes (C): Event classes classify log-lines using the set of known patterns. Each line can be classified by multiple event classes.

Hypothesis (H): Hypotheses describe possible implications[2] between log-lines based on their classification.

Rules (R): A rule is a proven hypothesis. This proof is given if the implication that is described by the hypothesis holds in a significant time of evaluations. The system evaluates rules continuously to detect anomalous behaviour in the system.

This approach tackles both above mentioned shortcomings; it also provides additional benefits:

i The model is generated following the real relationships of components in the monitored environment. The detected relationships involve expected relationships (e.g. firewall rules being evaluated before a request is transmitted to a secured server) but are not limited to those.

ii The rules in the model are automatically generated and unique for each monitored environment. A potential attacker cannot easily tailor an attack to prevent rules from failing; the attack, after all, alters the system behaviour.

iii Syntax and semantics of the log lines are widely irrelevant, since the model is tailored to the log input.

iv Sharing information about attacks with state actors, competitors or other third parties is often forbidden by companies, although information sharing is a requirement to monitor critical infrastructures on a national level. The reasons, to forbid information sharing, vary from legal obligations (e.g. privacy issues) to fear of the loss of customer's trust. As described later, the proposed approach handles log data, once classified, in a way, that privacy is not compromised. It is therefore possible, to transmit the abstract form of log-lines to a central (e.g. state controlled) instance; additionally to analysing it locally.

Intrusion Detection Systems only fulfil monitoring tasks. Actions to emit detected anomalies are not taken – humans still have to interpret a detected anomaly. Manual intervention is also required to determine the actions to take, in order to prevent further harm to the monitored ICT system.

Figure 3.2 provides a conceptional overview of the described approach's functionality, as first published partly in [19]. Using this visualisation, the approach is evaluated in two dimensions. On a horizontal dimension, the *Evaluation Stack* (visualised by the squared elements on the left side) is distinguished from the *Refinement Branches* (pictured by the elliptic elements on the right side). The *Evaluation Stack* describes the tasks performed to prepare and analyse the input and to detect anomalies. The approach performs all operations iteratively. The *Refinement*

[2] Implication in a logical sense as $A \rightarrow B \equiv \neg A \vee B$

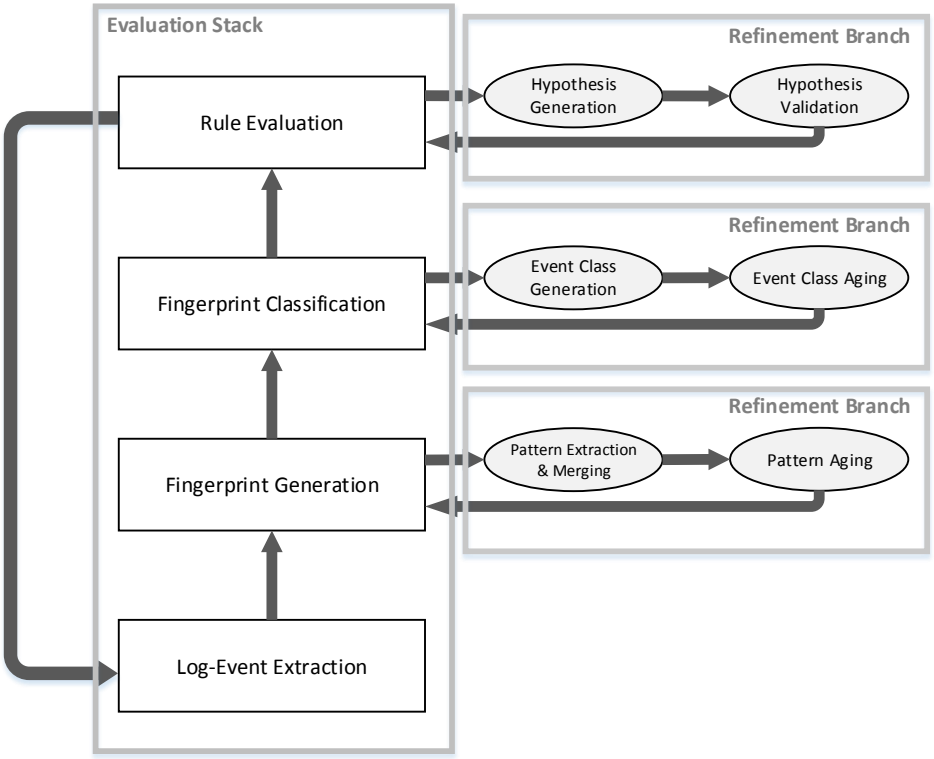


Figure 3.2: Conceptual overview.

Branches on the other hand, are triggered by the *Evaluation Stack's* elements; they evaluate and optimise the system model continuously. Figure 3.2 shows that refinement works in two steps. First, new knowledge is extracted from the currently processed line. Afterwards, the refinement process evaluates the knowledge and deletes deprecated or redundant information. The updated information is then available in the next iteration of the *Evaluation Stack*.

The system model M (see Equation 3.1) is defined as a tuple built from the sets of: known search-patterns $\mathbb{P}$, known event classes $\mathbb{C}$, known hypothesis $\mathbb{H}$ and known rules $\mathbb{R}$.

$$M = \langle \mathbb{P}, \mathbb{C}, \mathbb{H}, \mathbb{R} \rangle \tag{3.1}$$

3.2 Evaluation Stack

The *Evaluation Stack* performs tasks in sequential order. The tasks describe the general functionality of the system regarding input analyses and anomaly deteection:

3.2.1 Log-Event Extraction

A basic unit of logging information, e.g., one line for line-based logging, one binary log data record or one XML-element, is called a log-atom L_a; an L_a consists of a series of single symbols s (Eq. 3.2).

$$L_a = s_1 \dots s_n \tag{3.2}$$

Further a log event L_e (Eq. 3.3) is the association of a log-atom L_a with a timestamp t; L_e describes when L_a has been created.

$$L_e = \langle L_a, t \rangle \tag{3.3}$$

The first task collects the logging information from various distributed sources in the monitored network and emits them, one by one, to the next state in the *Evaluation Stack*. Assuming a complete and correctly sorted stream, atoms are emitted individually and timely sorted to the next task.

3.2.2 Fingerprint Generation

The next task vectorises each log-atom using $\mathbb{P}$. A search pattern P (Eq. 3.4) is a substring (an *n-gram*) of a log-atom L_a.

$$P = s_{1+i} \dots s_{m+i},\ where\ 0 \leq i\ and\ m + i \leq n \tag{3.4}$$

The vectorisation process transforms a log-atom L_a into an n-dimensional pattern vector – the so-called fingerprint $\vec{F}$ (Eq. 3.5). The information reported by L_a is encoded using the set of currently existing search patterns $\mathbb{P}$. The occurrence of each search pattern P, as a substring in L_a, is encoded with a bit $p_i \in \{0, 1\}$ in $\vec{F}$ (see Tab. 3.1 for an example).

$$\vec{F} = p_1 \dots p_n,\ where\ p_i \in \{0, 1\} \tag{3.5}$$

Fingerprinting L_a reduces the amount of data the next steps need to process – and speeds up the overall anomaly detection – significantly. As argued in Sect. 3.1, $\vec{F}$ can also be used to transmit log-data for external analysis; privacy issues do not require special consideration. Without $\mathbb{P}$ there is no way to extract sensitive data. Once L_a is vectorised, further analysis is performed solely on $\vec{F}$. Information encoded in L_a, that cannot be encoded by any combinations of $\{\forall P \in \mathbb{P}\}$, is inevitably lost for further steps in the *Evaluation Stack*.

```
1 service-3.v3ls1316.d03.arc.local apache: 2227 169.254.0.3:80 "mantis-3.v3ls1316.d03.arc
     .local" "mantis-3.v3ls1316.d03.arc.local" 169.254.0.2 - - [12/Feb/2014:13:30:16
     +0000] "GET␣/mantis/login_page.php␣HTTP/1.1" 200 1343 "-" "Mozilla/5.0␣(X11;␣Linux
     ␣i686)␣AppleWebKit/537.36␣(KHTML,␣like␣Gecko)␣Chrome/29.0.1547.65␣Safari/537.36"
```

Listing 3.1: Apache log excerpt from test environment.

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
Patterns $\mathbb{P}'$:	GET	POST	[12/Feb/2014:13:30:15 +0000]	v3ls13	s1316.d0	ice-4.v3	apache:	login_page.php	mysql-n
Fingerprint:	1	0	1	1	1	0	1	1	0

Table 3.1: Example of a fingerprint. Consider the sample log-line in Lst. 3.1 from an Apache server running Mantis[3] in our test environment. This table shows an example set of patterns $\mathbb{P}'$ and how a fingerprint of the line in Lst. 3.1 would look like.

3.2.3 Log-Atom Classification

Once an L_a is vectorised and $\vec{F}$ is generated the fingerprint (and therefore the underlying L_a) gets classified. Log event classification is the process that determines $\mathbb{C}_{\mathbb{L}_\supset}$ – the set of all event classes C a log-atom L_a belongs to (see Eq. 3.6). One L_a can belong to a multitude of classes, e.g., a log-atom might be an 'incoming connection event', an 'ssh service event' and an 'IP-zone X service event' at the same time. Each event class, that L_a belongs to, encodes a specific type of information that was originally encoded in L_a. Notice that L_a is classified, not L_e because the categorization is timestamp-independent.

$$\mathbb{C}_{L_a} = \{C | L_a \in C\} \tag{3.6}$$

A log-atom might also belong to no class at all. In this case L_a is discarded and not used for further evaluation. Information encoded in L_a is lost, since it could not be mapped to any existing event class C.

An event class C (Eq. 3.7) is defined as the combination of a mask $\vec{C_m}$ and a value $\vec{C_v}$.

$$C = \left\langle \vec{C_m}, \vec{C_v} \right\rangle \tag{3.7}$$

The event mask $\vec{C_m}$ acts as a filter and decides, what search patterns are considered relevant for the classification in the respective class (see Eq. 3.8). The value $\vec{C_v}$ decides for all relevant search patterns, if they are enforced on $\vec{F}$ or prohibited from being part of $\vec{F}$, for L_a to be classified as C (see Eq. 3.9). Note that $\vec{C_v}$ does only enforce or prohibit search patterns that are considered relevant by $\vec{C_m}$ (see Tab. 3.2 for an example).

$$\vec{C_m} = p_1 \dots p_n \text{ where } p_i = \begin{cases} 1 & \text{if } P \text{ at } i \text{ is relevant} \\ 0 & \text{if } P \text{ at } i \text{ is irrelevant} \end{cases} \tag{3.8}$$

$$\vec{C_v} = p_1 \dots p_n \text{ where } p_i = \begin{cases} 1 & \text{if } P \text{ at } i \text{ is enforced} \\ 0 & \text{if } P \text{ at } i \text{ is prohibited or irrelevant} \end{cases} \tag{3.9}$$

Each generated fingerprint $\vec{F}$ is classified by C if the condition in Eq. 3.10 holds. One fingerprint can be classified by one or more event classes.

$$\vec{C_v} = \vec{F} \wedge \vec{C_m} \tag{3.10}$$

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
Patterns $\mathbb{P}'$:	**GET**	POST	**[12/Feb/2014:13:30:15 +0000]**	v3ls13	s1316.d0	*ice-4.v3*	**apache:**	login_page.php	mysql-n
Fingerprint:	1	0	1	1	1	0	1	1	0
$\vec{C_m}$	1	0	1	0	0	1	1	0	0
$\vec{C_v}$	1	0	1	0	0	0	1	0	0

Table 3.2: Example of an event class that classifies the sample log-line in Lst. 3.1. While the bold patterns are enforced by C (i.e., they have to occur in L_a for it to be classified by C), italic patterns are prohibited by C (i.e., they must not occur in L_a for L_a to be classified by C). Other patterns are considered irrelevant by C.

Example: After classifying L_a the approach generates an event E^C (Eq. 3.11) for every event class $C \in \mathbb{C}_{L_a}$. An event E^C carries the information that a log-event L_e at time t got classified by C. These events are further investigated by the anomaly detection system.

$$E^C = \langle t, C \rangle \tag{3.11}$$

The idea is, to apply a basic concept of human reasoning. When a human analyses log-lines s/he scans each line and makes implications on the line's meaning from the line's content. Consider again the sample log-line in Lst. 3.1 from an Apache server running Mantis in our test environment.

One can understand certain aspects of the log-line, without detailed knowledge about an Apache log-line's syntax. One identifiable substring is *apache:* that tells us that this log was printed by an Apache server. We can further identify a timestamp with substring *[12/Feb/2014:13:30:16 +0000]* or, given some knowledge about HTTP commands, identify *GET* as one of those. We are able to extract a considerable amount of information without a lot of prior knowledge on the syntax and the semantics. We know the triggering application as well as the time and the request.

Of course, the substrings used in this basic sample are chosen intelligently and not completely random (as it is done by the system). But this simple sample is supposed to prove something different: Replacing one of the substrings (e.g. *GET* with *POST*) changes the meaning of the line fundamentally. This is the concept applied by classification. The meaning of the line – *„There was a GET-request on an apache server at time t"* – is inseparably connected with the fact, that the substrings *apache:*, *[12/Feb/2014:13:30:16 +0000]* and *GET* can be found in the respective line. On the other hand, those three substrings are insufficient to fully cover the information encoded in the line. We know that something was requested. But we don't know what was requested, who requested it or from which server it was requested from.

The translation of the example above, into the formal structure of the approach, looks as follows: An event class C has to encode the knowledge given by the three selected patterns from above. Without loss of generality we can say, that $P_1 =$ *GET* and $P_7 =$ *apache:*. C encodes the information carried by P_1 and P_7 if the following two conditions hold: C_m has to consider those two patterns as relevant and C_v has to enforce them. In a more formal way that means: p_1 and p_7 in $\vec{C_m}$ and $\vec{C_v}$ have to be 1 and $p_i = 0$ for $i \notin \{1, 7\}$. C can also encode more information. C_m would have to consider more patterns as relevant (formal: $\exists p_i = 1$ in $\vec{C_m}$ for $i \notin \{1, 7\}$). If p_i is 1 in $\vec{C_m}$ but 0 in $\vec{C_v}$ the represented patterns are prohibited. C might then not classify

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
Patterns $\mathbb{P}'$:	**GET**	POST	[12/Feb/2014:13:30:15 +0000]	v3ls13	s1316.d0	*ice-4.v3*	**apache:**	*login_page.php*	mysql-n
$\vec{C_m}$	1	0	0	0	0	1	1	1	0
$\vec{C_v}$	1	0	0	0	0	0	1	0	0

Table 3.3: Example of an event class that prohibits */mantis/login_page.php*. The line in Lst. 3.1 is not classified by this event class C but all apache *GET*-requests on other pages are classified by C.

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
Patterns $\mathbb{P}'$:	**GET**	POST	[12/Feb/2014:13:30:15 +0000]	v3ls13	s1316.d0	*ice-4.v3*	**apache:**	**login_page.php**	mysql-n
$\vec{C_m}$	1	0	0	0	0	1	1	1	0
$\vec{C_v}$	1	0	0	0	0	0	1	1	0

Table 3.4: Example of an event class that enforces */mantis/login_page.php*. The line in Lst. 3.1 is classified by this event class C but only apache *GET*-requests on this pages are classified by C.

GET-requests on a specific page (see Tab. 3.3 for an example). If p_i is also 1 in $\vec{C_v}$, a certain page (e.g. */mantis/login_page.php*) can be enforced on L_a, for L_a to be classified by C (see Tab. 3.4 for an example). In general a prohibited pattern carries less information than an enforced one.

3.2.4 Rule Evaluation

The last task of the *Evaluation Stack* changes the focus from single log-events to the relations between them. A hypothesis H (Eq. 3.12) is a non-validated correlation rule between events of two different event classes. The relation $\rightarrow$ is the logical consequence operator as specified by the truth table in Tab. 3.5. Although defined by two event classes, a hypothesis is used to evaluate the events that were triggered by the processed log-events. One evaluation of a hypothesis is therefore written as the test if $E^{C_{cond}} \rightarrow E^{C_{impl}}$ holds in t_w.

Condition	Implication	Result
$E^{C_{cond}}$	$E^{C_{impl}}$	$true$
$E^{C_{cond}}$	$\neg E^{C_{impl}}$	$false$
$\neg E^{C_{cond}}$	$E^{C_{impl}}$	$true$
$\neg E^{C_{cond}}$	$\neg E^{C_{impl}}$	$true$

Table 3.5: Truth table of the $\rightarrow$-relation.

The time window t_w describes the time span (relative to t at which L_e that triggered $E^{C_{cond}}$ occurred), in which the implication has to hold. The system automatically creates such correlation hypotheses, and subsequently tests them, to learn about event dependencies (see later). Notice that $t_w > 0$ does not hold in general. Some hypothesis make assumptions about events that have to have occurred before other events. t_w is fixed at generation time of H.

$$H = \langle C_{cond}, C_{impl}, \rightarrow, t_w \rangle \tag{3.12}$$

The system evaluates the stream of events against the hypotheses in $\mathbb{H}$ continuously. This evaluation process foresees one event queue Q_i for every existing hypothesis H_i. All queues Q_i listen to events relevant to the respective hypothesis; they add $E^{C_i}|_{C=C_{cond} \vee C=C_{impl}}$. A periodic evaluation process acts on the entries in the respective Q_i; its actions are described in Tab. 3.6. The evaluation is performed until none of the described cases hold. Then there are no more evaluations to be done in the current state. The rule is completely evaluated – until a new event is received by Q_i.

occurence	evaluation result
$E_1 \wedge \neg E_2$	Given t_w has passed the rule evaluates to **false** and a negative evaluation is stored. E_1 will be deleted.
$E_1 \wedge E_2$	Given both events occurred within t_w the rule evaluates to **true** and a positive evaluation is stored. E_1 and E_2 will be deleted.
$\neg E_1$	All occurrences of E_2 that lack an E_1 are deleted without an evaluation result being returned.

Table 3.6: Possible evaluation results of a hypothesis.

The result of an evaluation of Q_i – one evaluation – is called e (see Eq. 3.13). One evaluation stores the result res (true or false) of the evaluation, the hypothesis H_i with respect to which the evaluation was performed and the position pos that e has in the stream of evaluations.

$$e = \langle res, H, pos \rangle \tag{3.13}$$

Every hypothesis H_i stores evaluations in a stream S^{H_i} (see Eq. 3.14). The position of an evaluation is defined as its position in this stream; pos is incremented for all evaluations in S^{H_i}, whenever a new evaluation is stored. The most recent evaluation has always $pos = 0$.

$$S^{H_i} = \{e \mid H = H_i\} \tag{3.14}$$

A slot Sl is a filter, that can be applied on evaluation streams. The operation $size(Sl)$ returns a natural number, specifying the size of the slot. Applying a slot on an evaluation stream returns the newest $size(Sl)$ evaluations in that stream (see Eq. 3.15). The system generates k slots at initialization time with $size(Sl_i) < size(Sl_{i+1})$.

$$Sl(S^H) = \{e | e \in S^H \wedge pos < size(Sl_i)\} \tag{3.15}$$

Example: Consider the event class described in Tab. 3.4 as C_1 and the event class described in Tab. 3.3 as C_2. We can now construct a simple hypothesis as in Eq. 3.16. This hypothesis would state that after an apache *GET*-request was issued on *login_page.php* there is also an apache *GET*-request on another page within at most 10 seconds. Table 3.7 shows the status of Q_1 at different timestamps. Table 3.8 shows how different evaluation results affect the evalution stream S^{H_1} and how different slots access different evaluations.

$$H_1 = \langle C_1, C_2, +10s \rangle \tag{3.16}$$

Time	Q_1		
10:30:40	$\langle 10:30:35, C_1\rangle$		
10:30:42	$\langle 10:30:41, C_2\rangle$	$\langle 10:30:35, C_1\rangle$	
10:30:50	$\langle 10:30:45, C_1\rangle$		
10:30:57	$\langle 10:30:57, C_2\rangle$	$\langle 10:30:45, C_1\rangle$	
10:30:59			

Table 3.7: Sample status of the event queue Q_1.

Slot 3	Buffer	Sl_3									
Slot 2	Sl_2										
Slot 1	Sl_1										
10:30:40	1	1	0	1	1	1	1	1	1	1	1	1
10:30:50	1	1	1	0	1	1	1	1	1	1	1	1
10:30:59	0	1	1	1	0	1	1	1	1	1	1	1
pos	1	2	3	4	5	6	7	8	9	10	11	12

Table 3.8: Sample development of the evaluation stream S^{H_1}. In this sample we assume three slots ($k = 3$) with $size(Sl_1) = 2, size(Sl_2) = 5$ and $size(Sl_3) = 10$.

Considering the position of evaluations as timely ordered, the evaluation's results produce a binary stream. The continuous evaluation process can then be interpreted as a Bernoulli process; a discrete-time stochastic process that takes only two values. This Bernoulli process is used to decide about the stability of a hypothesis; a stable hypothesis is transferred into a rule R. The stability function in Eq. 3.17 implements a left-sided binomial test; $isStable(H)$ analyses the current evaluation stream against a pre-defined stochastic distribution. The distribution, against which the stability test is performed, is described by a statistical hypothesis p_0[4]. A binomial test evaluates, if a given sample supports a pre-defined distribution. $B(i \mid p_0, n)$ returns the probability that i out of n evaluations are positive, given that p_0 is the assumed chance of an evaluation being positive. A significance level α is the threshold below which the tested sample (and with it the evaluated hypothesis H) are refused[5]. Let further be $\{e_t^H\}$ the set of evaluations belonging to hypothesis H with $res = true$. The sample of evaluations, used for the stability evaluation of H, is $Sl_k(S^H)$; the biggest slot defined.

$$isStable(H) = \sum_{i=0}^{|\{e_t^H \in Sl_k(S^H)|} B(i|p_0, size(Sl_k) \geq \alpha \tag{3.17}$$

If $isStable(H)$ evaluates to *true* the tested hypothesis H is considered a stable rule R and becomes part of the set of rules $\mathbb{R}$ (see Eq. 3.18). Note that $\mathbb{R} \subset \mathbb{H}$.

$$\mathbb{R} = \{H \mid isStable(H)\} \tag{3.18}$$

[4] p_0 in this case relates to a hypothesis about the statistical distribution of the sample and is not comparable to a implication hypothesis as defined in Eq 3.12.

[5] n, H_0 and α are part of the initial configuration and will be discussed in more detail in Sect. 6.1

Rules are considered currently accepted hypotheses; they undergo the same periodical evaluations as all hypotheses. Additionally, their evaluation stream is analysed for anomalies. As mentioned before, the stability function $isStable(H)$ uses the biggest slot Sl_k to decide if a hypothesis is part of $\mathbb{R}$. Testing a rule $R \in \mathbb{R}$ for anomalies uses the same left-sided binomial test, as proving stability does. Only the parameters change (see Eq. 3.20). Let $\{e_t^R \in Sl_k(S^R)\}$ be the set of all positive evaluations, of a given rule, passing the filter applied by Sl_k. The test tries to verify a statistical hypothesis[6] p_0, that is now given by the ratio of positive evaluations in Sl_k to total evaluations in Sl_k (see Eq. 3.19). An anomaly significance α_a specifies the threshold below which a sample is considered anomalous. The anomaly analysis is performed for all slots Sl_i with $i < k$.

$$p_0 = \frac{|\{e_t^R \in Sl_k(S^R)\}|}{size(Sl_k)} \tag{3.19}$$

$$isAnomalous(R) = \bigvee_{j=0}^{k-1} \left(\sum_{i=0}^{|\{e_t^R \in Sl_j\}|} B(i|p_0, size(s_j)) \leq \alpha_a \right) \tag{3.20}$$

An anomaly A (see Eq. 3.21) is a highly significant deviation of the sample of evaluations that is given by applying slot Sl_j on the stream of evaluations S^R (i.e., $Sl_j(S^R)$), with respect to the expected distribution given by the evaluations in $Sl_k(S^H)$. A consists of a probability $p = 1 - \sum_{i=0}^{|\{e_t^R \in Sl_j\}|} B(i|p_0, size(Sl_j))$, a rule R, on which the anomaly was detected, and a time t, at which the anomaly was detected.

$$A = \langle p, R, t \rangle \tag{3.21}$$

Example: Taking into account the last snapshot of S^{H_1} in Tab. 3.8, we can calculate p_0 by counting the number of positive evaluations that are considered by Sl_3 ($|\{e_t^R \in Sl_k(S^R)\}| = 9$). The size of Sl_3 is 10 and the resulting p_0 is then calculated based on Eq. 3.19 as 0.9.

Table 3.9 describes all possible values for Sl_1 and Sl_2 that can be calculated by Eq. 3.20. Let's assume that $\alpha_a = 0.01$. In this case:

$$isAnomalous(H_1) = 0.19 \leq 0.1 \vee 0.08146 \leq 0.1 = false$$

3.3 Refinement Branches

The analysis of system behaviour, as described in Sect. 3.2, is highly dependent on knowledge, represented by the system model M. Since the proposed system does not rely on any pre-defined knowledge, M has to be built while analysing the input. A first step generates knowledge about processed input; this generated knowledge builds M. The lack of information about the meaning of good (meaningful) knowledge, justifies the need to evaluate M and delete deprecated or

[6]Note that this statistical hypothesis is not comparable to an hypothesis $H \in \mathbb{H}$. It describes the expected probability that the next evaluation is positive and is tested by statistical means to decide if the different evaluations still support this probability or if a significant change is measured (i.e., an anomaly is detected).

$\lvert\{e_t^R \in Sl_k(S^R)\}\rvert$:	0	1	2	3	4	5
Sl_1	0.01	**0.19**	1			
Sl_2	$1.0e^{-5}$	$4.6e^{-4}$	$8.56e^{-3}$	$\mathbf{0.08146}$	0.40951	1

Table 3.9: This table describes the values of a Cumulative Distribution Function (CDF) based on the slots from the last timeslot in Tab. 3.8. All possible values are calculated; the bold values are the ones valid for the values in Tab. 3.8

.

redundant information. The following subsection will go into more detail about how knowledge is generated and refined for search patterns $\mathbb{P}$, event classes $\mathbb{C}$, hypothesis $\mathbb{H}$ and rules $\mathbb{R}$.

3.3.1 Search Pattern Refinement

Generation

Search patterns $\mathbb{P}$ (see Eq. 3.4 for a single search pattern) are created by the system based on currently processed log-atoms L_a. Since search patterns are the only basis for vectorisation of L_a, it is essential, that all log-atoms are matched by multiple search patterns. Only the combination of search patterns matching L_a and search patterns not matching L_a gives meaning to an event E^C triggered after classifying L_e. There is information encoded in the fact that $\nexists P$ matching L_a. But information only gathered by the absence of search patterns has little entropy[7]. It is therefore important to get a sufficient coverage of occurring log-events with search patterns.

To achieve this the system generates new patterns with Alg. 3.1. Patterns are generated for:

i an uncovered L_a (e.g., when new log sources are being connected).

ii a well-covered L_a (in order to refine the knowledge; happens less frequently).

Generation of new search patterns P is balanced by generating new patterns more frequently for uncovered or weakly covered log-atoms.[8]. The system applies a simple but effective token bucket algorithm. Processing an L_e increases the number of tokens in a bucket. The system generates a new search pattern, if there are enough tokens in the bucket. An important configuration parameter is the price of a pattern ($baseCost$). This algorithm is an easy way to overcome the *pattern balancing problem*. The pattern balancing problem deals with the fact that rarely occurring log-atoms are not properly indexed by the search patterns curently in $\mathbb{P}$, while frequently occurring log lines are indexed (i.e., covered with patterns) very well. But especially the rarely occurring log-atoms are the most interesting ones; they represent exceptional events. To overcome this problem, the generation of a search pattern from rare log-atoms is cheaper, i.e., less tokens are withdrawn from the bucket when a search pattern for a rare log-atom is created. This allows the creation of several patterns for one rare L_a. Buying a pattern for a well covered type of log atom on the other hand is expensive, but still affordable from time to time. Not preventing

[7]Entropy describes how much new information is carried by the result of a random event.

[8]Coverage in this sense means the number of already collected search patterns matching an L_a

the generation of search patterns for well covered log-atoms is important to be able to generate more restrictive event classes. Too general event classes result in a high number of trivial rules that cloud the system model.

Data: L_a, $baseCost$
Result: P
$bucketSum$++
$matchingCount \leftarrow$ sum_matching_search_patterns(L_a)
$cost \leftarrow baseCost * 2^{matchingCount}$
if $bucketSum \geq cost$ **then**
    $candidate$ = get_random_substring()
    if $\exists P \equiv$ *substring* **then**
        $P \leftarrow substring$
        $bucketSum$ = $bucketSum$ - $cost$
    end
end

Algorithm 3.1: Creation of a new search pattern P.

Example: Let's assume $baseCost = 10$ and L_a is the log-line from Lst. 3.1. Lets further assume that $\mathbb{P}'$ is again given by the set of patterns used in Tab. 3.1. Based on $\vec{F}_{L_a}$ the $matchCount$ is 6 and the $cost$ would be calculated as: $baseCost = 10 + 2^6 = 74$. So if $bucketSum \geq 74$ a new pattern will be generated on L_a. Otherwise no pattern gets generated.

Merging

Section 3.2 described that search patterns P are used to classify log-atoms L_a by event classes C. Each L_a can be classified by multiple event classes and therefore triggers a multitude of events E. These events are then used in hypotheses and rules to analyse implications between event classes.

Generating a hypothesis $H = \langle C_1, C_1, \rightarrow, t_w \rangle$ is not useful. Evaluations of H are trivially true, for every occurrence of E^{C_1}; H generates no information. But this H increases the computational complexity. The hypothesis will become stable and statistical evaluations have to be performed for every occurrence of E^{C_1}.

Preventing this case is trivial; simply do not allow hypothesis to be implications between between the same event class. But given the random nature of search patterns the situation is more complex. Let's assume two search patterns P_1 and P_2. Let's further assume the relation in Eq. 3.22 holds.

$$\text{iff } P_1 \in L_a \rightarrow P_2 \in L_a, \forall L_a \tag{3.22}$$

The system can generate two event classes C_1 and C_2. Let p_1 be the bit in $\vec{C_m}$ and $\vec{C_v}$ stating the relevance or enforcement of P_1 and let p_2 be the same for P_2. Without loss of generality we can generate C_1 and C_2 in a way that C_1 is equivalent to C_2 except for p_1 and p_2. While P_1 is

enforced in C_1 it is irrelevant in C_2. And while P_2 is enforced in C_2 it is irrelevant in C_1. Given this case Eq. 3.23 holds:

$$\{E|E \in C_1\} \equiv \{E|E \in C_2\} \tag{3.23}$$

The information carried by C_1 is exactly the same as the one carried by C_2. From an information theoretical point of few they are identical. But on implementation level they are not; they are considering different search patterns. The correlation given in Eq. 3.22 is not known to the system and cannot be exploited. Let's assume a third event class C_3 in our model. Let's further assume that there is a valid rule:

$$R_1 = \langle C_1, C_3, \rightarrow, t_w \rangle \tag{3.24}$$

The system can also extract the following rules, and would accept them as stable rules:

$$R_1 = \langle C_2, C_3, \rightarrow, t_w \rangle \tag{3.25}$$

$$R_2 = \langle C_1, C_2, \rightarrow, t_w \rangle \tag{3.26}$$

$$R_3 = \langle C_2, C_1, \rightarrow, t_w \rangle \tag{3.27}$$

These rules will not add any additional information to the model. But they will consume computing power. Rule R_2 will further trigger the same anomalies as rule R_1. That will make it harder for an administrator to monitor the system status. In a system status with a general set of event classes $\mathbb{C}$ with size n one additional event class equivalent to an existing event class would mean $2 * n$ new potential hypotheses. One new event class increases the search space, in which the system tries to identify rules, by $2 * n$. But in our case the increased space does not hold any additional possibilities as the knowledge in M stays the same.

Given the assumptions made in the design of the system (namely no use of pre-defined knowledge about the analysed dataset) it is not possible to formally decide which search patterns should be generated or which search patterns should be combined in a new event class. Instead the decision process is completely random.

To limit the risk of redundant event classes the system approximates the existence of the case given in Eq. 3.22 between two search patterns. A merge candidate P' is a new search pattern, that is created by merging two existing search patterns. Table 3.10 covers the conditions to identify merge candidates. Note that merge candidates are only generated right after a new search pattern is generated. Note further, that in every case given in Tab. 3.10 the existing search pattern P and the new search pattern P_{new} can switch roles.

Merging search patterns is a critical task. Existing search patterns are often already relevant for event classes. Merging patterns can have second level effects on event classes – effects on the information encoded in an event class. Proving that the condition in Eq. 3.22 holds would ensure that a merge has no effects on the information encoded in the event classes. But this proof is not possible without complete knowledge about syntax and semantics of the input data. The system has to substitute this proof by approximation. Algorithm 3.3 defines this approximation.

Description	Formular
Log-Atom	$L_a = s_1 \dots s_n$
Existing Pattern $\in L_a$	$P = s_o \dots s_p$ with $1 < o < p < n$
New Pattern generated from L_a	$P_{new} = s_q \dots s_r$ with $q < r$
Case 1: The new pattern is a substring of an existing pattern	**Condititons:** $q > o \wedge r < p$
	Candidate: $P' = P$
Case 2: New pattern and existing pattern are overlapping	**Condititons:** $o < q \leq p \wedge r > p$
	Candidate: $P' = s_o \dots s_r$
Case 3: New pattern can be found in L_a right next to an existing pattern	**Condititons:** $q = p + 1$
	Candidate: $P' = s_o \dots s_r$
Case 4: New pattern and existing pattern are just separated by a space	**Condititons:** $q = p + 2 \wedge s_{p+1} = \text{' '}$
	Candidate: $P' = s_o \dots s_r$

Table 3.10: Definition of cases in which search patterns can get merged.

Given a specified threshold[9] the system evaluates the number of times a substitution of P_1 and P_2 by their merge candidate P' would be valid[10]. A merge is considered iff the condition in Eq. 3.28 holds:

$$P_1 \in L_e \leftrightarrow P_2 \in L_e \ , \ \forall L_e \text{ with } t < now \tag{3.28}$$

The system further has to ensure that the proposed merge candidate is a valid substitution (i.e.: would be generated again if the generation process was performed on the current L_e). If one of these conditions fails at any time during the evaluation P_1 and P_2 will not get merged. This is also the reason why it is sufficient to generate new merge candidates only with the generation of new patterns (as seen in Tab. 3.10).

Algorithm 3.2 shows the generation process of new merge candidates. The method *findMergeCandidate(*P_1, P_2*)* returns a new substring describing the possible merge candidate of P_1 and P_2.

Data: P_{new}, $\{P|P \in L_a\}$
Result: $\{P'\}$
1 **foreach** $P \in L_a$ *as* P_{known} **do**
2 $P' \leftarrow$ findMergeCandidate(P_{known}, $Pnew$)
3 **if** $P' \neq$ *null* **then**
4 storeMergeCandidate(P',P_{new},P_{known})
5 **end**
6 **end**

Algorithm 3.2: Find merge candidates for new search pattern P_{new}.

Algorithm 3.3 describes the evaluation process of a merge candidate P' before the merge is performed. The duration of the evaluation is limited by the pre-set threshold ϑ. ϑ speci-

[9]This threshold is defined manually in the configuration file of the system

[10]The substitution is considered valid if both search patterns have the same merge candidate on the current L_e as on all L_e before where they mached.

fies the number of evaluations P' has to be proven a valid merge candidate before the system performs the merge. Each merge candidate is a combination of exactly two search patterns. *getOtherPattern(P, P')* is used, given one search pattern P and a merge candidate P', to retrieve the other search pattern used to generate P'. The method *increaseCounter(P')* is used to mark one positive evaluation of the merge candidate. If the counter reaches ϑ the merge is performed. By removing a pattern we change the meaning of p_i in all fingerprints $\vec{F}$. This change will be automatically adopted by new fingerprints and event classes. But it has to be propagated to masks ($\vec{C_m}$) and values ($\vec{C_v}$) in all event classes in $\mathbb{C}$. After a successful merge the merge candidate P' will replace one of the two merged search patterns while the other one will be deleted. Method *updateEventClasses()* performs this task. Table 3.11 shows the required changes on $\vec{C_m}$ and $\vec{C_v}$ in each event class to mitigate second level effects (see Tab. 3.12 for an example).

```
   Data: L_a, ϑ
 1 foreach P ∈ L_a do
 2     foreach stored merge candidate of P as P' do
 3         P_o ← getOtherPattern(P', P)
 4         if P_o ∈ L_a then
 5             if P' = findMergeCandidate(P, P_o) then
 6                 increaseCounter(P')
 7                 if counter(P') ≥ ϑ then
 8                     merge(P, P_o)
 9                     updateEventClasses()
10                 end
11             else
12                 deleteMergeCandidate(P')
13             end
14         else
15             deleteMergeCandidate(P')
16         end
17     end
18 end
```

Algorithm 3.3: Evaluation of merge candidates for $P \in L_a$.

Aging

Search patterns represent the substrings of a line that are known to the system as part of M. Because search pattern generation happens completely random not all generated search patterns carry useful information. Figure 3.3 categorises different types of search patterns. Most information is carried by *reoccurring* substrings:

Localisers can give information about source and destination of a logged event. They can be used to identify involved parties.

	p_1	p_2	p'_1	Description
$\vec{C_m}$	1	1	1	Both patterns are enforced. Since the merged pattern covers both C has to
$\vec{C_v}$	1	1	1	enforce p'_1 to classify the same log-atoms as before the merge.
$\vec{C_m}$	1	1	-	This event class should get deleted. Since 3.22 holds: $\nexists L_a : \neg P_1 \in L_a \wedge P_2 \in$
$\vec{C_v}$	0	1	-	L_a.
$\vec{C_m}$	0	1	1	P_1 is ignored. Since Eq. 3.22 holds $P_1 \in L_a$ has to hold anyway for L_a to be
$\vec{C_v}$	0	1	1	in C.
$\vec{C_m}$	1	1	-	This event class should get deleted since Eq. 3.22 holds: $\nexists L_a : P_1 \in L_a \wedge$
$\vec{C_v}$	1	0	-	$\neg P_2 \in L_a$.
$\vec{C_m}$	1	1	1	Both search patterns are prohibited. The same has to be true for there merged
$\vec{C_v}$	0	0	0	successor.
$\vec{C_m}$	0	1	1	One search pattern is ignored, so the other search pattern determines the status
$\vec{C_v}$	0	0	0	of the merged one.
$\vec{C_m}$	1	0	1	One search pattern is ignored, so the other search pattern determines the status
$\vec{C_v}$	1	0	1	of the merged one.
$\vec{C_m}$	1	0	1	One search pattern is ignored, so the other search pattern determines the status
$\vec{C_v}$	0	0	0	of the merged one.
$\vec{C_m}$	0	0	0	Both search patterns are ignored. The same has to be true for the merged search
$\vec{C_v}$	0	0	0	pattern.

Table 3.11: Changes performed on event classes after pattern merge.

Keys are used to set information carried in a log-atom into a specific context (e.g., *value* = . . .).

Values contain information about the action reported by a given log-event. These values can be reoccurring (if they are part of an enumeration or a well defined set of possible values) or infrequent if they represent measurement results from a sensor.

Search patterns carrying most information are substrings from those reoccurring cases. Figure 3.3 shows that not all substrings of log-information fit into reoccurring categories. Arbitrary sequences, unique measurement values, punctuation marks or syntax dependent special characters provide less useful search patterns. Changes to the monitored system's architecture can also render certain search patterns useless.

Aging is a periodic process, that identifies and removes search patterns from $\mathbb{P}$ that have no use for classifying log lines. This can have many reasons; two are tackled with aging:

i P reflects a unique substring from a previously processed log-event L_e (e.g. a session ID). It will not occur any more – or is at least extraordinary rare – in future log-events. P is therefore useless for classification.

ii P reflects a substring that does not occur any more due to changes in the monitored system (e.g. the network name of a server that was shut down).

		p_1	p_2	p_3	p_4	p_5	p_6	p_7
	Patterns $\mathbb{P}'$:	**GET**	POST	v3ls13	s1316.d0	*ice-3.v3*	**apache:**	*login_page.php*
Old	$\vec{C_m}$	1	0	0	1	1	1	1
	$\vec{C_v}$	1	0	0	1	0	1	0
Prep	$\vec{C_m}$	1	0	1	0	1	1	1
	$\vec{C_v}$	1	0	1	0	0	1	0
	Patterns $\mathbb{P}''$:	**GET**	POST	v3ls1316.d0	*ice-3.v3*	**apache:**	*login_page.php*	
New	$\vec{C_m}$	1	0	1	1	1	1	
	$\vec{C_v}$	1	0	1	0	1	0	

Table 3.12: Example of the effects of a pattern merge on an event class. The patterns P_3 and P_4 can get merged, since they are overlapping parts of the general network name that is part of every network address in our sample network. Before the merge P_4 is enforced by C while P_3 is ignored. During the preparation phase it is ensured that C is not dependent on P_4 any more since it should get deleted. Instead P_3 gets enforced and P_4 is ignored. After the merge $\mathbb{P}'$ transcended into $\mathbb{P}''$ after the old P_4 was deleted. $\vec{C_m}$ and $\vec{C_v}$ are adapted accordingly.

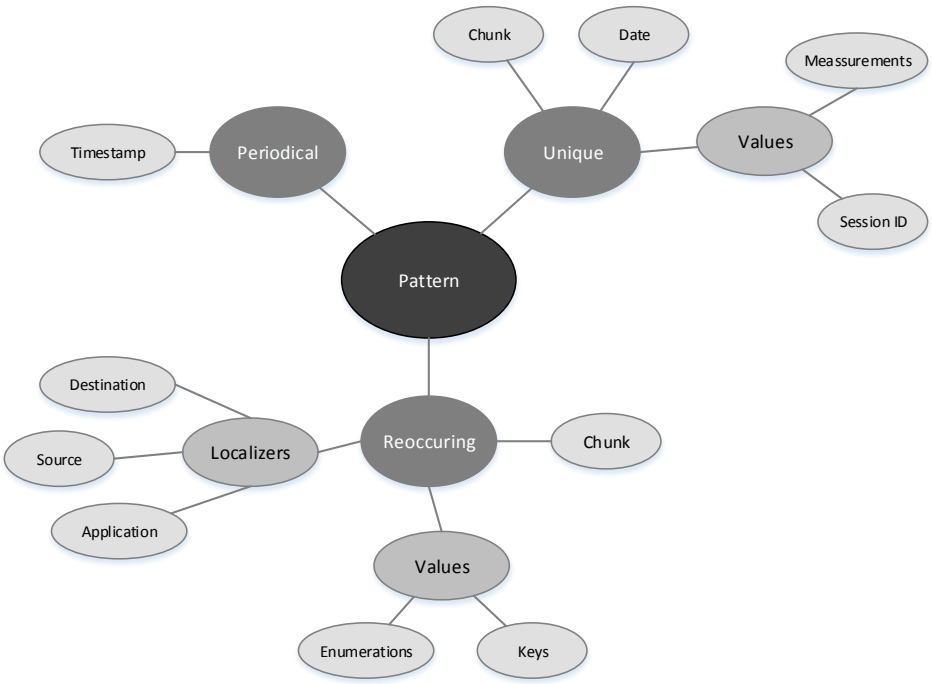


Figure 3.3: Mindmap about substring types.

Again – as in previous refinement steps – the system cannot prove behaviour of the input in the future. Knowing that any search pattern fulfils *(i)* and *(ii)* is not possible and approximations have to be applied.

The *Monitored System Behaviour Period* T is defined as the smallest time window, in which every periodic task, in the monitored system, occurs at least once[11]. In a corporate environment T will typically be set to one week. Given that there are daily and weekly backup processes, a

[11]This definition considers normal behaviour in the monitored system

period of one day would be too short. The weekly backup process would not occur in all periods. There can also be different workloads over the weekends than during week days, making a one day period not optimal. A SCADA set-up allows a much shorter period. Given a 24/7 operation mode and only non-periodic maintenance access, a period of one day is probably a good choice.

Given a certain period T, with respect to the monitored system, approximation is performed as follows: a search pattern should age out (i.e. get deleted) iff Eq. 3.29[12] and Eq. 3.30 hold:

$$\nexists L_e \mid (t_{now} - \rho_P * T < t_{L_e} < t_{now}) \wedge P \in L_e \tag{3.29}$$

$$\nexists C \mid \vec{C_m}(i) = 1 \text{ with } P_i \text{ being the aging candidate} \tag{3.30}$$

This approximation substitutes the proof, that a search pattern P is not occurring any more in future log-events. In the condition in Eq. 3.29 T is used to ensure that search patterns, that describe periodic log-events, do not get deleted during normal system behaviour. Search patterns, that get deleted because of aging, further cannot be substituted by other search patterns, carrying the same information. Second level effects on event classes, from deleting a search pattern because of aging, cannot be prevented by the measures described in Tab. 3.10; Equation 3.10 cannot be applied. Second level effects on event classes can only be eliminated by the condition in Eq. 3.30; ensuring that there are no event classes considering P as relevant. The corresponding position in $\vec{C_m}$ and $\vec{C_v}$ can be deleted without effects on C.

3.3.2 Event Class Refinement

Generation

The system defines event classes C automatically using a similar token bucket algorithm[13] as the one used when generating search patterns. Each log-atom L_a is characterised by its corresponding fingerprint $\vec{F}$. Balancing event class generation is important because any log-event L_e, that cannot be classified by an event class $C \in \mathbb{C}$, contains no information interpretable by the system. Consequently L_e will not be analysed by the system. The system ensures the following two cases in order to balance event class generation:

i A new event class gets generated for every $\vec{F}|_{\nexists\ C \in \mathbb{C} \text{ classifying } \vec{F}}$

ii New event classes might be generated for every $\vec{F}$. Generation is less probable the more existing event classes classify $\vec{F}$.

One property of event classes is generality. Event classes carry information because they classify a subset of the log-events described by the input. The entropy of an event E^C, marking

[12] ρ_P is a global configuration parameter that specifies the number of periods a search pattern P is kept in $\mathbb{P}$ without occurring in any L_e

[13] Here, tokens (a kind of virtual credits) are generated over time and put into a basket. If there are enough tokens in the bucket, a new class is generated and the required amount of tokens removed from the bucket.

the classification of L_e by C, is inverse proportional to the percentage of log-events in a finite input set classifiable by C (see Eq. 3.31).

$$Entropy(E_C) = \frac{|\{L_e\}|}{|\{L_e \mid C \text{ classifies } L_e\}| - 1} \tag{3.31}$$

Trivial event classes, classifying all possible log-events, carry no information about a specific log-event. An event class generation process has to ensure a certain level of entropy. Table 3.13 establishes three parameters used to ensure proper levels of entropy on newly generated event classes.

μ_e	**Enforced search patterns:** This parameter defines a minimum of enforced bits in a new event class.
ϕ_e	**Percentage of matching search patterns that will be enforced:** This parameter defines a percentage of the search patterns matching L_e. This is then the number of search patterns that should be enforced by a new event class.
ϕ_p	**Percentage of not matching search patterns that will be prohibited:** This parameter defines a percentage of the search patterns not matching L_e. This is then the number of search patterns that should be prohibited by a new event class.

Table 3.13: Configuration parameters used to ensure entropy on newly generated event classes.

Algorithm 3.4 shows the class creation process. Each event class is generated based on the fingerprint $\vec{F}$ of the currently processed log-event. The method `numberOfClassifyingEventClasses(`$\vec{F}$`)` returns the number of event classes $C \in \mathbb{C}$ that already classify $\vec{F}$. Once the algorithm decided if a new event class can be generated based on $\vec{F}$, and how many search patterns have to be considered, enforced and prohibited, the methods `enforceNRandomPatterns(`$number\_enforced\_bits$, $\vec{C_m}$, $\vec{C_v}$, $\vec{F}$`)` and `prohibitNRandomPatterns(`$number\_prohibited\_bits$, $\vec{C_m}$, $\vec{C_v}$, $\vec{F}$`)` set the bits in $\vec{C_m}$ and $\vec{C_v}$ accordingly. Table 3.14 gives an example.

Aging

Event classes store information about the composition of log-atoms from search patterns $P \in \mathbb{P}$. The set of known event classes $\mathbb{C}$ also represents the search space used for creating new hypotheses in the search for rules. The monitored system can dynamically change. Certain event classes might not be classifying any log-events any more because of intended changes in the monitored system. These never triggered event classes increase the search space without increasing the set of hypotheses. As described in Sect. 3.2, hypotheses are generated based on the event classes classifying the currently processed log-events. Event classes that do not classify any log-events cannot be used to generate meaningful hypotheses. The aging of event classes is a periodic process on $\mathbb{C}$ that identifies and removes unused elements $C \in \mathbb{C}$.

Data: $\vec{F}$, μ_e, ϕ_p, ϕ_e
Result: $\vec{C_v}$, $\vec{C_m}$

```
bucketSum++
matchingCount ← numberOfClassifyingEventClasses(F⃗)
cost ← baseCost + (matchingCount * balancingCost)
if bucketSum ≥ cost then
    matching_bits ← {p_i | p_i ∈ F⃗ ∧ p_i = 1}
    number_enforced_bits ← φ_e * |matching_bits|
    not_matching_bits ← {p_i | p_i ∈ F⃗ ∧ p_i = 0}
    number_prohibited_bits ← φ_p * |not_matching_bits|
    if |matching_bits| < μ_e then
        abort
    end
    if number_enforced_bits < μ_e then
        number_enforced_bits ← μ_e
    end
    enforceNRandomPatterns(number_enforced_bits, C⃗_m, C⃗_v, F⃗)
    prohibitNRandomPatterns(number_prohibited_bits, C⃗_m, C⃗_v, F⃗)
end
```

Algorithm 3.4: Creation of a new class C.

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
Patterns $\mathbb{P}'$:	**GET**	*POST*	[12/Feb/2014:13:30:15 +0000]	v3ls13	s1316.d0	ice-4.v3	**apache:**	**login_page.php**	mysql-n
Fingerprint:	1	0	1	1	1	0	1	1	0
$\vec{C_m}$	1	1	0	0	0	0	1	1	0
$\vec{C_v}$	1	0	0	0	0	0	1	1	0

Table 3.14: Example of a newly generated event class based on $\mu_e = 2$, $phi_e = 0.5$ and $phi_p = 0.3$ and the fingerprint from Tab. 3.1.

An event class gets deleted if the conditions in Eq. 3.32 and in Eq. 3.33 hold:

$$\nexists L_e \mid (t_{now} - \rho_C * T < t_{L_e} < t_{now}) \wedge C \text{ classifies } L_e \tag{3.32}$$

$$\nexists H \in \mathbb{H} \mid C \text{ relevant for } H \tag{3.33}$$

The concept is similar to the concept of the aging process on $\mathbb{P}$. Hypotheses depend on certain event classes. The condition in Eq. 3.33 ensures that the aging process only deletes event classes without dependent hypotheses. Event classes with dependent hypotheses, cannot be deleted without deleting the dependent hypotheses too. There is no way to find a substitution for C without possibly altering the information encoded in a hypothesis. Furthermore hypotheses model relations between event classes. The lack of log-events being classified by an event class C does not automatically mean that C is deprecated. Anomalous behaviour would also result in missing log-events. The aging process therefore has to take $\mathbb{H}$ into account. Given that the

condition in Eq. 3.33 holds, the condition in Eq. 3.32 specifies the threshold for deleting an event class. The system uses the already introduced period T to ensure that periodic tasks occur at least once before deleting an event class. The constant configuration parameter ρ_C specifies the accuracy of the approximation.

If both conditions hold, the system assumes that there will not be a log-event in the future that can be classified by C and C will be deleted. If this assumption was wrong the dynamic generation process can potentially generate a new event class C' that classifies $\{L_e \mid C \text{ classifies } L_e\}$.

3.3.3 Hypothesis and Rule Refinement

Generation

The system generates hypotheses based on $\mathbb{C}_{L_a}$ (see Eq. 3.6) – the set of event classes that classify the currently processed log-atom. Algorithm 3.5 describes the process in detail. $\mathbb{H}_D$ is the set of hypotheses that were discarded by the aging process. The generation process applies the same bucket algorithm as the one used for balancing the generation of search patterns and the generation of event classes, for balancing the generation of hypotheses. The algorithm increases the bucket count for every processed log-atom and ensures that the cost for a new hypothesis, calculated based on the current log-event, can be served by the bucket. The method *getRendomElement(*$\mathbb{S}$*et)* returns a random element from the supplied set. Method *getRandomTimeWindow()* returns a random time window t_w that is enforced on the implication encoded in H_{new}.

The evaluation of hypotheses is the most time consuming task performed by the system. The system is therefore designed to prohibit the re-generation of:

i hypotheses that are currently in $\mathbb{H}$

ii hypotheses that can be substituted by a hypothesis that is currently in $\mathbb{H}$

iii hypotheses that were already discarded by the aging process (as well as substitutes of those)

Table 3.15 and Tab. 3.16 describe the conditions under which a generation of H_{new} is prohibited. These are also the conditions tested by the method *isSubstitutedBy(*H_{new}*,* H_{old}*)*.

Stability

The periodic generation process only generates hypotheses and cannot extend the set of rules directly. The set of rules $\mathbb{R}$ is a subset of the set of hypotheses $\mathbb{H}$. A hypothesis is considered to be a rule if $isStable(H)$ (see Eq. 3.17) evaluates to true. Figure 3.4 visualises the Probability Density Function (PDF) of some sample binomial distributions. While n defines the size of the sample, p specifies the probability of one evaluation of H to be true. Figure 3.5 shows the Cumulative Distribution Function (CDF) for the same distributions. A PDF states the probability for every number of positive elements in the sample; a CDF shows the probability for every number of positive elements in the sample, that at least that many elements are positive. The CDF is used to determine the threshold of the hypothesis' stability. Figure 3.6 shows how the

Data: $\mathbb{C}_{L_a}$, $\mathbb{C}$, $\mathbb{H}$, $\mathbb{H}_D$
Result: H_{new}

$bucketSum$++
$matchingCount \leftarrow |\mathbb{C}_{L_a}|$
$cost \leftarrow baseCost + (matchingCount * balancingCost)$
if $bucketSum \geq cost$ **then**
  $C_1 \leftarrow$ getRandomElement($\mathbb{C}_{L_a}$)
  $C_2 \leftarrow$ getRandomElement($\mathbb{C}$)
  $t_w \leftarrow$ getRandomTimeWindow()
  $H_{new} \leftarrow \langle C_1, C_2, \rightarrow, t_w \rangle$
  foreach $H_{old} \in \mathbb{H} \cup \mathbb{H}_D$ **do**
    if *isSubstitutedBy(H_{new}, H_{old})* **then**
      abort
    end
  end
  $bucketSum = bucketSum - cost$
end

Algorithm 3.5: Creation of a new hypothesis H.

$H_{old} \in \mathbb{H}$	H_{new}	Description
$\langle C_1, C_2, \rightarrow, t_1 \rangle$	$\langle C_1, C_2, \rightarrow, t_1 \rangle$	$\mathbb{H}$ already contains the exact same hypothesis. Generating a new hypothesis encoding the same information does not extend the model.
$\langle C_1, C_2, \rightarrow, t_1 \rangle$	$\langle C_1, C_2, \rightarrow, t_2 \rangle$	H_{new} is a more generic hypothesis than the existing H_{old}. As long as the aging process does not decline H_{old} a more generic rule does not extend the model.
$\langle C_1, C_2, \rightarrow, -t_1 \rangle$	$\langle C_1, C_2, \rightarrow, -t_1 \rangle$	$\mathbb{H}$ already contains the exact same hypothesis. Generating a new hypothesis encoding the same information does not extend the model.
$\langle C_1, C_2, \rightarrow, -t_1 \rangle$	$\langle C_1, C_2, \rightarrow, -t_2 \rangle$	H_{new} is a more generic hypothesis than the existing H_{old}. As long as the aging process does not decline H_{old} a more generic rule does not extend the model.

Table 3.15: Conditions prohibiting generation of H_{new} based on $\mathbb{H}$. Note that: $0 < t_1 < t_2$.

stability evaluation uses the significance value α to define the stability threshold. Equation 3.34 shows the formula to calculate a PDF; Equation 3.35 shows the calculation of a CDF.

$$PDF(x) = \binom{n}{x} p^x (1-p)^{n-x} \tag{3.34}$$

$$CDF(x) = \sum_{k=0}^{x} \left(\binom{n}{k} p^k (1-p)^{n-k} \right) \tag{3.35}$$

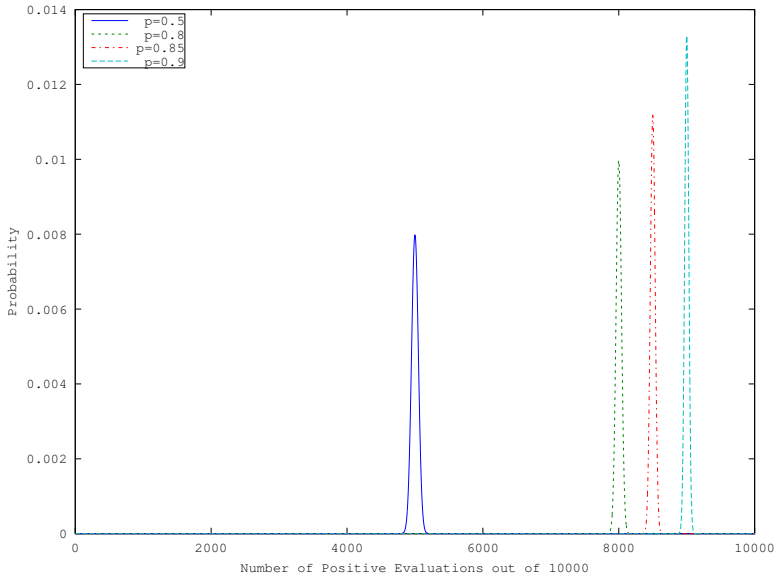


Figure 3.4: Samples of Probability Distribution Functions with $n = 10\,000$.

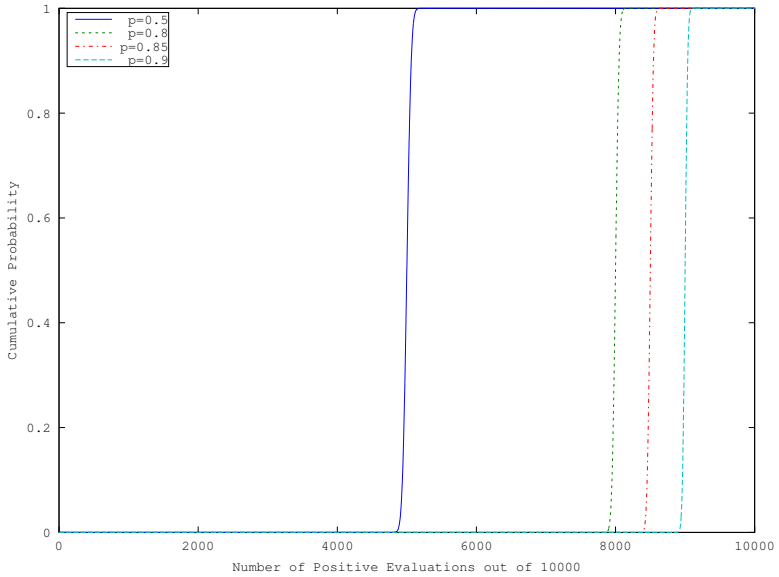


Figure 3.5: Samples of Cumulative Probability Functions with $n = 10\,000$.

$H_{old} \in \mathbb{H}_D$	H_{new}	Description
$\langle C_1, C_2, \rightarrow, t_1 \rangle$	$\langle C_1, C_2, \rightarrow, t_1 \rangle$	$\mathbb{H}_D$ already contains the exact same hypothesis. This hypothesis got already declined by the aging process. Evaluating the same hypothesis again will have the same result.
$\langle C_1, C_2, \rightarrow, -t_1 \rangle$	$\langle C_1, C_2, \rightarrow, -t_1 \rangle$	$\mathbb{H}_D$ already contains the exact same hypothesis. This hypothesis got already declined by the aging process. Evaluating the same hypothesis again will have the same result.

Table 3.16: Conditions prohibiting generation of H_{new} based on $\mathbb{H}_D$. Note that: $0 < t_1 < t_2$.

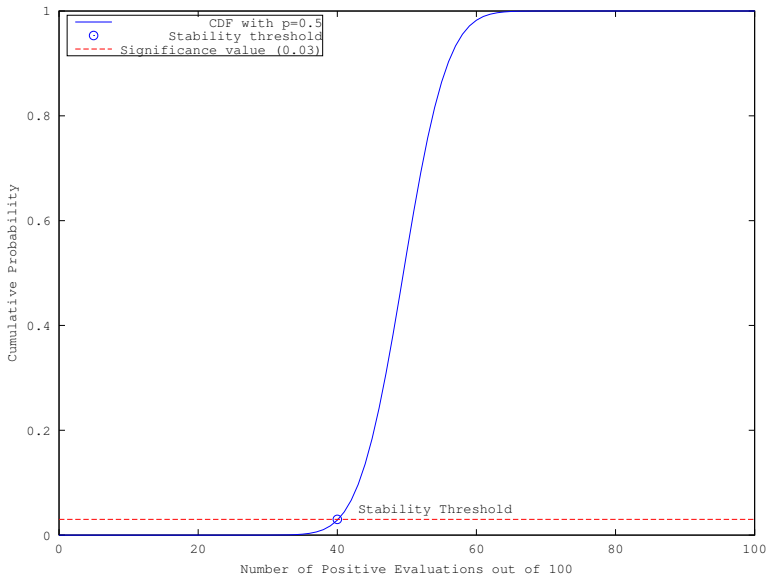


Figure 3.6: Calculations of Stability Threshold given $\alpha = 0.03$ and $n = 100$.

An analysis of Fig. 3.4 and Fig. 3.5 also shows, that the expected value ($\mu = n * p$) moves to the right, as p increases. This movement also results in denser probability distributions. The probability of μ is increasing with an increasing value of p. Setting $p = 1$ would result in an extreme situation described by Eq. 3.36.

$$PDF(x) = \begin{cases} 1 & \text{if } x = n \\ 0 & \text{else} \end{cases} \tag{3.36}$$

Equation 3.17 describes – and Fig. 3.6 visualises – how stability of a hypothesis is evaluated. But the evaluation of stability of a hypothesis is only performed once a reliable number of evaluations e (see Eq. 3.13) can be found in S^H (see Eq. 3.14). Some input has to be processed before stability of a hypothesis can be decided. The stability evaluation is performed on $Sl_k(S^H)$

(see Eq. 3.15). The use of statistical means every time stability gets evaluated – in contrast to just calculating a fixed threshold once, for a sample set of $size(Sl_k)$ – makes it possible to decide on stability, even if $|S^H| << size(Sl_k)$. Therefore, the decision can be made shorter after the generation of a new hypothesis.

Aging

The aging process for hypotheses is again a periodic process, that checks that one of two conditions holds before deleting a hypothesis. The condition described by Eq. 3.37 states, that the aging process deletes unstable hypotheses. Hypotheses are also deleted if they cannot be evaluated over a long time period. The lack of evaluations means, that the processed input triggers no condition events. The behaviour described by the rule is therefore not represented in the processed log-files. The event class, that describes the condition events, does not classify any log-atoms and should have been deleted by the event class aging process before H got generated (see Eq. 3.32). But now that $H \in \mathbb{H}$ the condition in Eq. 3.33 prohibits the deletion of the condition event class. H can never be evaluated, and therefore it has to be deleted by the aging process. Equation 3.38 describes this condition. The aging process again uses the period T as estimation and a constant parameter $\rho_H \geq 1$ as a multiplier to ensure the occurrence of periodic log-events before deletion. The function $time(e)$ returns the timestamp of a given evaluation e. As described in Eq. 3.14 e_0 is the newest evaluation in any S_H.

$$isStable(H) = \text{ false} \tag{3.37}$$

$$time(e_0 \in S^H) < now - \rho_H * T \tag{3.38}$$

In contrast to the aging processes for search patterns and event classes, the aging process of hypotheses is applied if any of the before mentioned conditions holds. There are no atoms in M that H depends on. Equation 3.37 also shows, that aging can only be applied on hypotheses, never on rules since $\{\nexists R \in \mathbb{R} \mid isStable(R) = \text{ false}\}$.

CHAPTER 4

Implementation

This chapter describes the concrete prototype implementation of the approach described in Ch. 3. After a short introduction in Sect. 4.1 follows a description of the configurable parameters in Sect. 4.2. Section 4.3 gives a detailed insight into the implementation of different core functionalities such as atom generation or hypotheses evaluation.

4.1 Introduction

The system described in the previous chapter (see Ch. 3) is implemented as a prototype using the Java[1] programming language. Figure 4.1 shows the already known conceptual overview of the system, as well as the implementing Java class for every task. Not all functionalities of a specific task described in Sect. 3.1 are implemented solely in this class. For example: The decision, if two patterns should get merged or not, is taken in `cais.atomhandler.balancer.PatternBalancer`. The handling of the effects of the deletion of a pattern on event classes (see Tab. 3.10), is only triggered by the `PatternBalancer` class but it is realised in the `cais.atomhandler.balancer.EventClassificationBalancer`.

The Java classes in Fig. 4.1 control the correct workflow of the system. The tasks are performed strictly sequential. A Publish-Subscribe Pattern[2] enables the classes that control a task to register for updates about new log-events at their predecessor. The two Java interfaces `cais.atomhandler.listener.IAtomListener` and `cais.atomhandler.listener.IAtomSource` implement the described Publish-Subscribe Pattern (see Fig. 4.2).

[1] Java is a popular, object-oriented programming language. See http://www.java.com/ for details.

[2] In this pattern, entities acting as publishers send information to all registered subscribers. Two interfaces are required: The publisher provides an interface for entities to register and unregister as subscribers. Whenever the publisher has information to transmit, it sends this information to all currently registered subscribers. The subscribers therefore have to provide a unified interface to receive the information from the publisher.

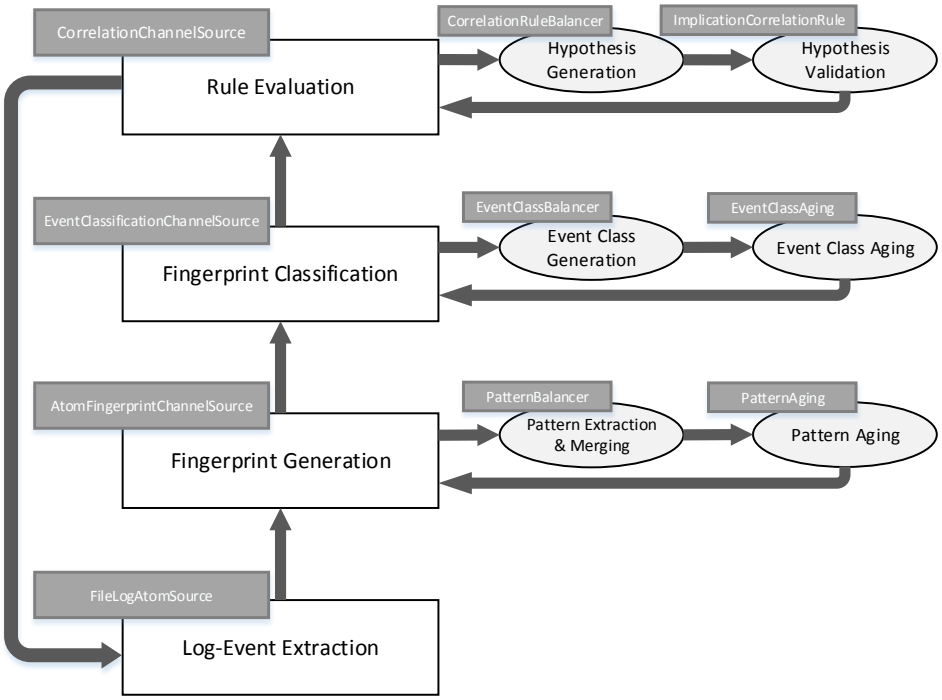


Figure 4.1: Conceptual overview including implementing Java classes.

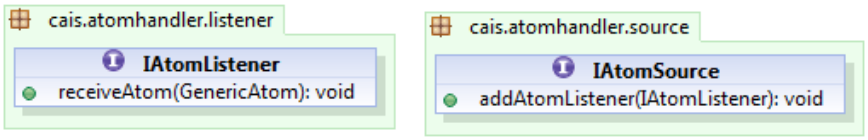


Figure 4.2: Interfaces for the publish-subscribe pattern.

4.1.1 Execution Flow

The execution order of the tasks in the *Refinement Branches* is not ensured by the implementation. All tasks in a *Refinement Branch* are registered directly at the triggering task of the *Evaluation Stack*. The uncertainty in the execution order does not alter the algorithm results, since the execution order is irrelevant as long as all tasks are performed for one log-event, before the next log-event gets processed.

The information submitted by classes implementing the `IAtomSource` interface is wrapped in `cais.atom.LineFingerprintAtom`. Figure 4.3 shows the general structure of this class. The information encoded in the different fields is as follows:

data: Object This field stores the fingerprint $\vec{F}$ of the current log-event. $\vec{F}$ is implemented as an array of integer values. Each bit of every integer encodes the match (1) or mismatch

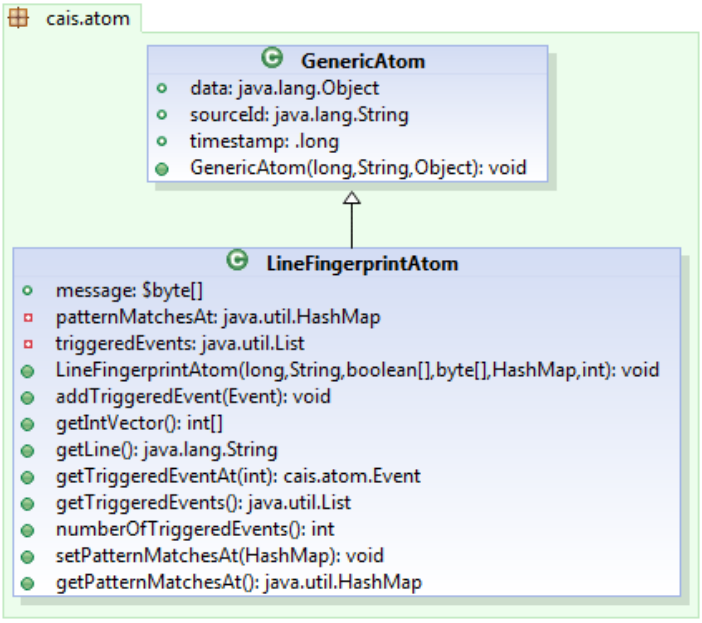


Figure 4.3: Class diagram of LogFingerprintAtom.java.

(0) of one pattern on the wrapped log-line.

timestamp: long The timestamp that was extracted from the log-line in the first task of the *Evaluation Stack*.

message: byte[ ] The line (excluding the timestamp that was extracted) that is wrapped by the object.

patternMatchesAt: HashMap<Pattern, int> This `HashMap` stores, for every search pattern P that matches the line, the position of the search pattern in the respective line. The case that one pattern can occur multiple times in one log-line is omitted. The system considers the first occurrence.

triggeredEvents: List<Event> This list stores all events E that were triggered by the currently processed log-event.

During runtime of the system, these fields are continuously filled when the respective information is available. Figure 4.4 shows the general information flow in the system (from bottom to top with respect to the *Evaluation Stack*). At every level the box visualising `LineFingerprintAtom.java` shows, what fields are set at this level and what fields are used. Furthermore, tilted boxes indicate the fields used by classes implementing or managing refinement tasks. The bent arrows encode this information flow. The straight arrows describe the general execution

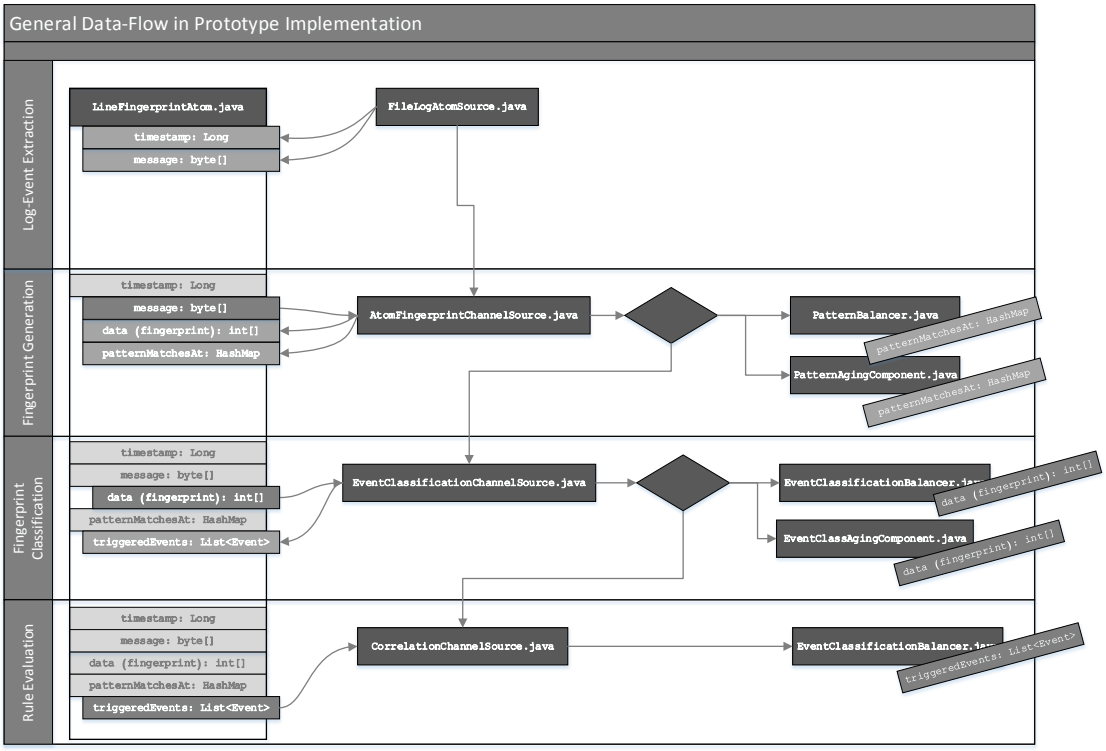


Figure 4.4: Flow diagram of general components picturing the use of LineFingerprintAtom at all stages.

flow. The order, in which the next tasks are called, is not fixed before runtime. Every straight arrow indicates one call of `receiveAtom(GenericAtom)` that passes one `LineFingerprintAtom` object, encoding the log-event that is currently processed by the system.

4.1.2 Atoms

As seen in Sect. 3 the system highly relies on the quality of the information in the system model M. An atom is a search pattern, an event class, a hypothesis, an event or an anomaly. Figure 4.5 shows the class structure of the main atoms: search patterns P (`cais.atom.Pattern`), event classes C (`cais.atom.EventClass`) and hypotheses H (`cais.rule.ImplicationCorrelationRule`). Some general functionality, that each atom has to provide, is extracted into abstract classes and interfaces for maintenance reasons. Each atom is a `cais.atom.AbstractUpdatableObject`. This abstract class ensures a unique id over all atoms. Each atom is further a `cais.atom.AbstractAgingObject`, providing functionality to manage the aging conditions directly at the atom itself. The implementation further handles events E^C and anomalies as separate classes in the atom package. The `cais.atom.Event` class describes events; the `cais.atom.Anomaly` class describes anomalies. They do not belong to the same inheritance tree as patterns, event classes and hypotheses (see Fig. 4.5) because they are neither

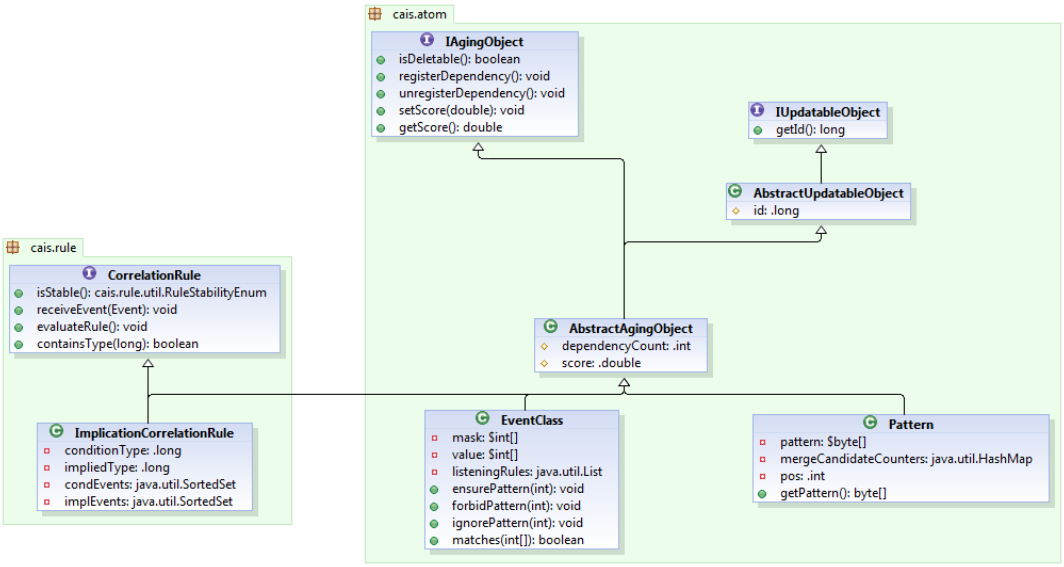


Figure 4.5: Class structure of atoms building M.

subject to any aging functionality nor managed by any registry. Instead events and anomalies are managed by the hypotheses they belong to.

4.1.3 Package Structure

The complete package structure is visualised by Fig. 4.6. The following list gives a short overview about the purpose of each package:

cais The root package contains the main class `cais.FingerPrintDemo` and hosts all other packages.

cais.atom The `atom` package hosts all classes that represent atoms or provide functionality for these atoms.

cais.atomhandler This package houses all classes that perform tasks on atoms. These can either be tasks from the *Evaluation Stack* or tasks from the *Refinement Branches*. The following subpackages cluster different types of tasks by the stage at which they are performed at.

cais.atomhandler.source This package contains the interface description for *Publishers* as well as all classes acting solely as *Publishers*.

cais.atomhandler.listener This package contains the *Subscriber's* interface.

cais.atomhandler.registries Registries are classes that are instantiated only once during runtime of the system (so called *Singleton* classes). They act as a in-code database for different types of atoms. Registries handle generation, deletion and maintenance of the respective atoms and provide access to the stored atoms by different means.

cais.atomhandler.intermediate A class is considered an intermediate, if it implements the `IAtomListener` interface as well as the `IAtomSource` interface. These classes are *Subscriber* as well as *Publisher*.

cais.atomhandler.balancer Classes in this package manage atom generation.

cais.atomhandler.aging Classes in this package manage aging of atoms.

cais.configuration This package contains utilities handling global configurations.

cais.rule This package contains all classes implementing the functionality of hypotheses and rules.

cais.rule.util This package contains all classes that are utilised to implement rule functionality.

cais.util This package contains global utility classes.

cais.db All classes in this package are used to persist knowledge generated by the system in an SQL-database.

4.2 Parameters

During runtime, the proposed system extracts information from log-lines and analyses the input at the same time using this information. To achieve this, the system requires some pre-defined parameters. These parameters define thresholds, initialization values or settings from external tools (e.g. credentials for database access). The choice of these parameters is critical but most of the parameters do not depend on the architecture of the monitored system. One goal of the system is small configuration effort at set-up time. The default values of the parameters will mostly be sufficient, but parametrization gives the freedom to adapt the system to more complex needs. Table 4.1 gives an overview on the existing parameters and their functionality. The most critical parameter in this configuration is the *Monitored System Behaviour Period* T. The implementation does not use the timestamps of processed log-lines to measure the time passed. Decisions involving time (e.g. aging or set-up phases) are taken based on the number of processed log-lines[3]. The system measures T by counting the number of log-lines processed. The number of lines that occur in one period is a fixed value in the current implementation. Future version can adapt this number dynamically at runtime.

[3] An exception to this rule is the evaluation of hypotheses based on the events relevant for H. This decision is based on the exact time of a log-event.

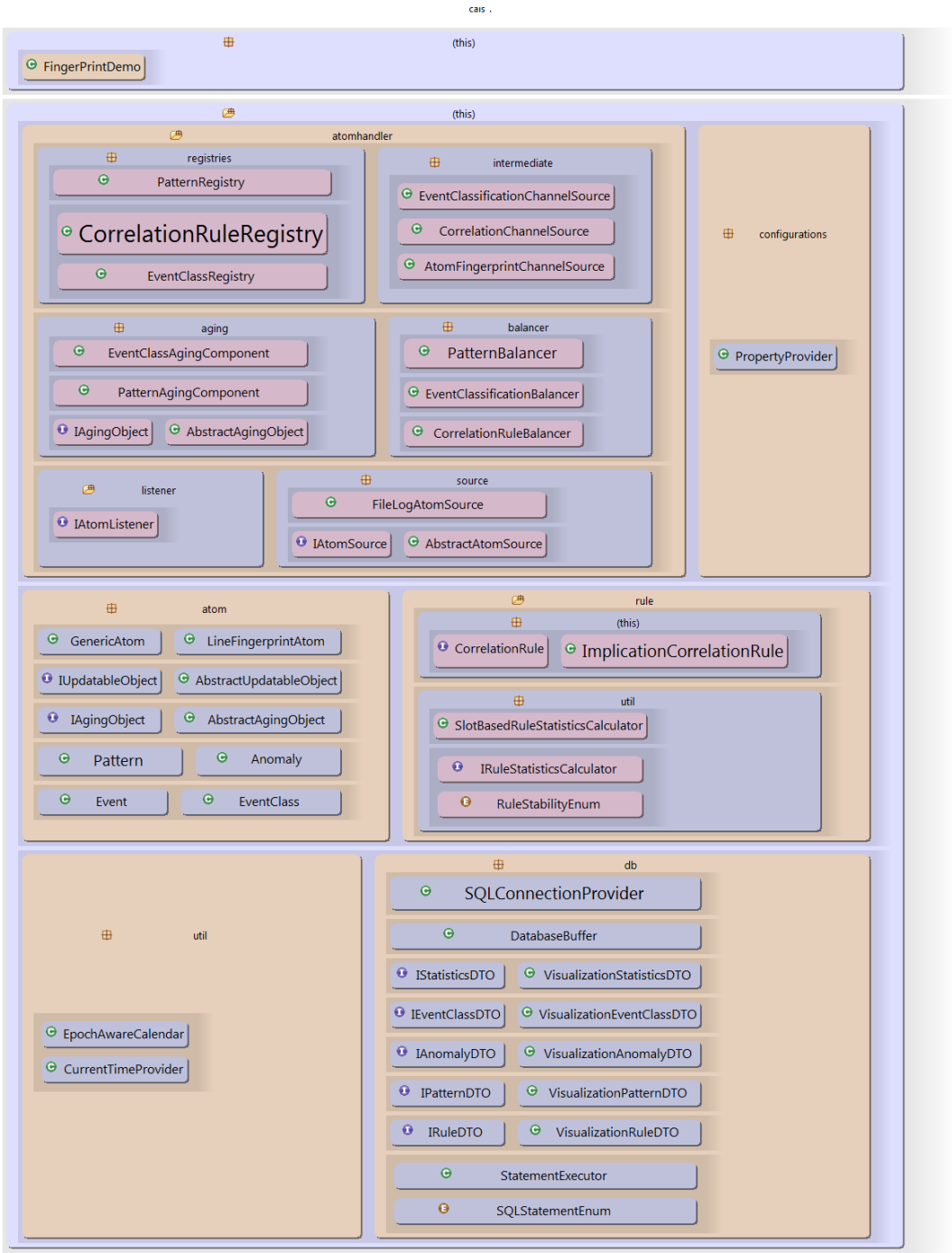


Figure 4.6: Package structure of the prototype implementation.

The approximation of time by the number of processed lines, makes the algorithm more syntax independent. The system is also more resistant to network delays: events that arrive with a delay do not cause the „system clock“ to run backwards.

Database Configuration	
`cais.database`	Enables persistent storage of the generated model M. The possible values are *true* or *false*.
`cais.database.*`	Possible parameters are: *driver*, *url*, *username* and *password*. They are used to define location and type of a database as well as the credentials to access it.
General Configuration	
`cais.logLinesPerPeriod`	The number of lines that are used to approximate one period T.
PatternBalancer Configuration: `cais.atomhandler.balancer.PatternBalancer.*`	
`baseCost`	Fixed price for a new search pattern P in the token balancing algorithm.
`patternCreationAttempts`	A new search pattern P is only generated, if $P \notin M$. If a randomly chosen substring is already part of the model it is not generated again. This parameter defines how often the system tries to find a new pattern on a line.
`minPatternLength`	The minimum length of a substring building a search pattern.
`maxPatternLength`	The maximum length of a substring building a search pattern.
EventClassificationBalancer Configuration: `cais.atomhandler.balancer.EventClassificationBalancer.*`	
`baseCost`	Fixed price for every new event class according to the balancing algorithm (see Alg. 3.4).
`triggeredEventCost`	Variable price for every new event class according to the balancing algorithm (see Alg. 3.4).
`prohibitedBitsPercentage`	Every event class is generated based on a log-line's fingerprint $\vec{F}$. A certain number of all patterns that cannot be found on L_a is prohibited in the event class. The prohibited patterns are randomly selected. This parameter specifies the percentage of all patterns that are not matching L_a that are selected. (see ϕ_p in Alg. 3.4).
`minEnforcedBitsPerMask`	The number of search patterns that have to be enforced by an event class for the event class to be generated (see μ_e in Alg. 3.4).

`enforcedBitsPercentage`	A certain number of all patterns that are found on L_a is enforced in the event class. The enforced patterns are randomly selected. This parameter specifies the percentage of all patterns that are matching L_a that are selected (see ϕ_e in Alg. 3.4).
`classCreationAttempts`	Number of times the system tries to randomly generate a new event class that is not yet part of $\mathbb{C}$.
CorrelationRuleBalancer Configuration: `cais.atomhandler.balancer.CorrelationRuleBalancer.*`	
`baseCost`	Fixed price for every new hypothesis according to the balancing algorithm (see Alg. 3.5).
`existingHypothesisCost`	Variable price for every new hypothesis according to the balancing algorithm (see Alg. 3.4).
`createAttempts`	Every hypothesis is generated based on the event classes classifying the current log-event L_e as well as the set of all known event classes $\mathbb{C}$. This parameter defines how often the system tries to randomly define a new hypothesis on one line.
Aging Configuration: `cais.atomhandler.aging.*`	
`numberOfInitialPeriods`	Once a search pattern or an event class is generated, this parameter defines the number of periods T it does not age out although no event class or hypothesis depend on it.
`numberOfPeriodsForUnusedPattern`	Number of periods T a search pattern does not age out, without depending event class (see ρ_P in Eq. 3.29).
`numberOfPeriodsForUnusedEventclass`	Number of periods T an event class does not age out, without depending hypothesis (see ρ_C in Eq. 3.32).
`numberOfPeriodsForUnevaluatedRule`	Number of periods a hypothesis does not age out, if it is not considered stable and no new evaluations are triggered (see ρ_H in Eq. 3.38).

Table 4.1: Configuration parameters in the prototype implementation.

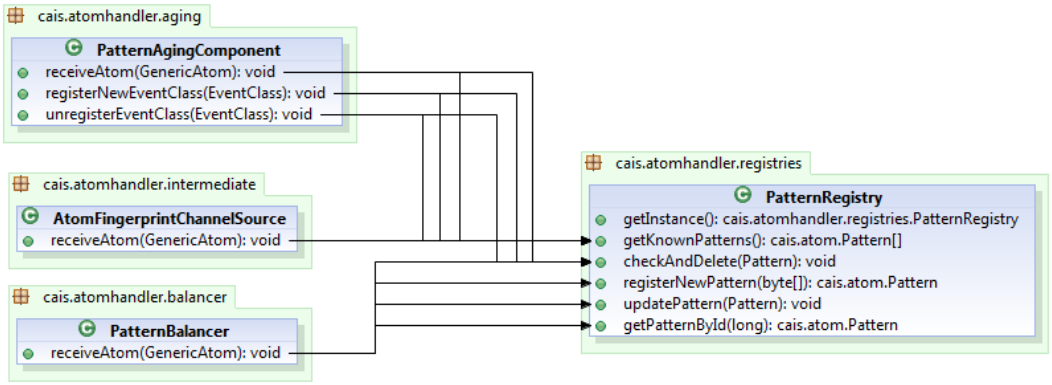


Figure 4.7: Class diagram and call overview of `PatternRegistry`.

4.3 Core Functionalities

4.3.1 Registries

The system model M (see Eq. 3.1) is managed by so called registries. Each registry manages one subset ($\mathbb{P}$, $\mathbb{C}$ or $\mathbb{H}$) of M[4]. They act as internal databases providing CRUD[5] operations for the atoms. The following figures show the registry classes as well as an overview of the calls made to the specific methods provided: Figure 4.7 depicts the `PatternRegistry`, Fig. 4.8 depicts the `EventClassRegistry` and Fig. 4.9 depicts the `CorrelationRuleRegistry`.

In general, the respective balancing components, aging components and channel sources access the registries most frequently.

Balancing-components generate new information based on the `LineAtomFingerprint` encoding the current log-event. This generation process requires information about the existing atoms, because duplicated atoms are prohibited. Additionally, after generating a new atom, this atom requires to be registered at the respective registry. Only after this registration the new atom is considered part of the system model M.

Aging-components handle the deletion of invalid or deprecated atoms. In order to make the decision if an atom should be deleted, the aging component has to keep track about dependencies on the atom in question and on the atom's internal status. Since the atoms are managed by the registry, the aging component simply requests the required information. If an atom has to be deleted the deletion process is triggered at the registry. That way a common knowledge of M over all components of the system is guaranteed.

Channel-sources manage the general workflow in the system. They manage the tasks described by the *Evaluation Stack*. They also get the required information directly from the reg-

[4]Since the set of rules in the system model $\mathbb{R}$ is a subset of the set of hypotheses $\mathbb{H}$ it is not managed by a separate registry. Instead $\mathbb{H}$ and $\mathbb{R}$ are handled by the same registry.

[5]CRUD is an acronym describing general database operations: Create, Read, Update and Delete

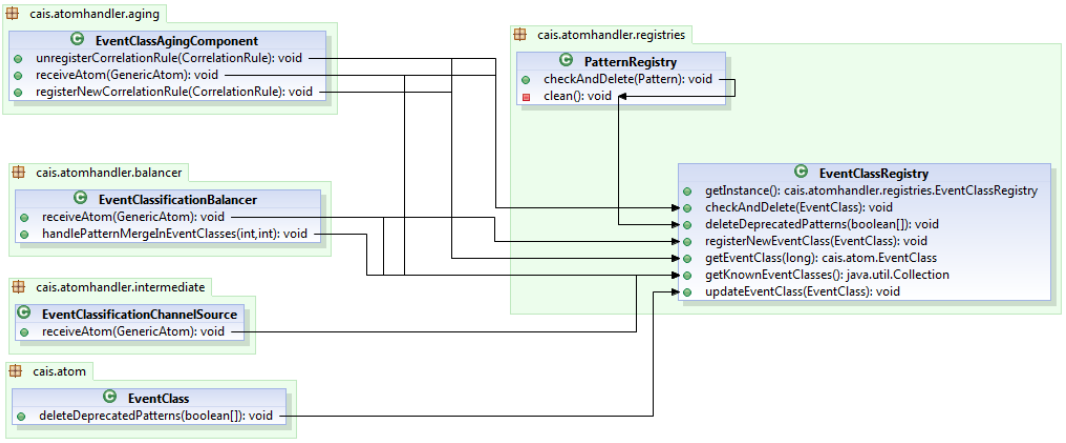


Figure 4.8: Class diagram and call overview of `EventClassRegistry`.

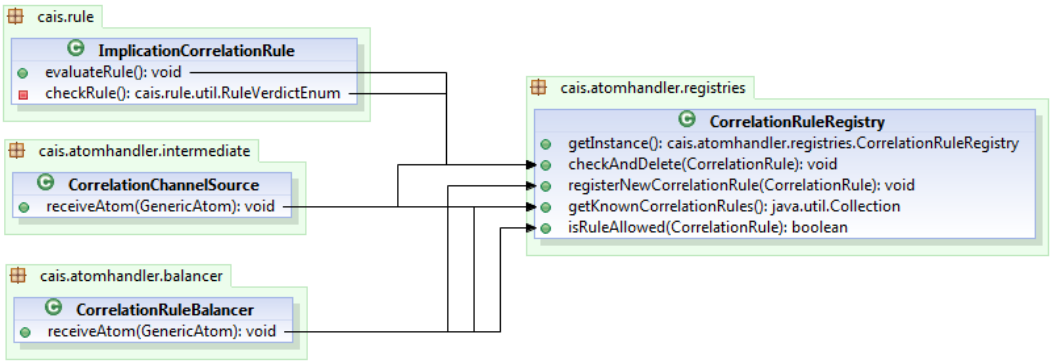


Figure 4.9: Class diagram and call overview of `CorrelationRuleRegistry`.

istries.

Although registries are used mainly by the components responsible for atoms they are available throughout the system and can be accessed by all components. One example can be seen in Fig. 4.8: The `PatternRegistry` accesses the `EventClassRegistry` in order to trigger the changes that have to be performed on $\vec{C_v}$ and $\vec{C_m}$ after a pattern got deleted or two patterns got merged (see Tab. 3.11).

4.3.2 Balancing Components

Balancing components handle the generation of atoms in M. There are three balancing components: one for search patterns $\mathbb{P}$, one for event classes $\mathbb{C}$ and one for hypotheses $\mathbb{H}$. Figure 4.10 shows a class diagram of the `PatternBalancer`. The information about the currently

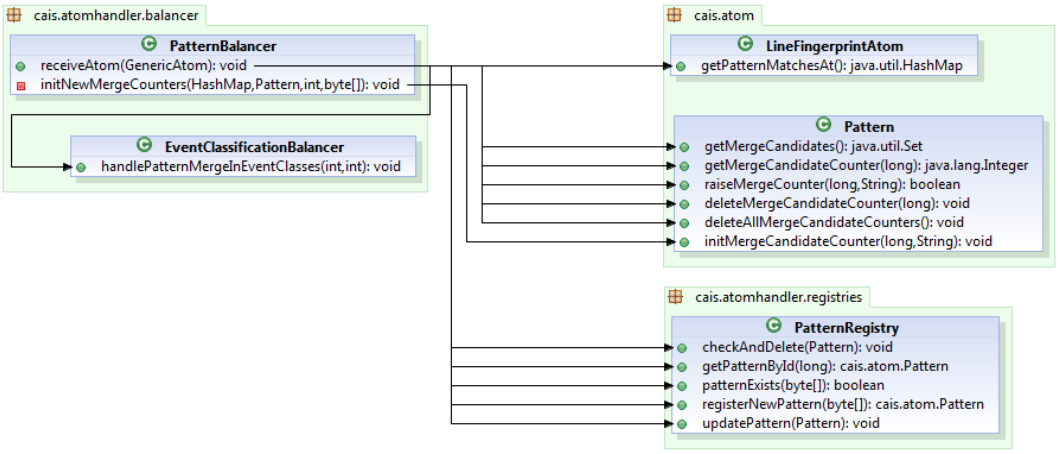


Figure 4.10: Class diagram showing the interactions triggered by the `PatternBalancer`.

processed log-line is retrieved from the `LineFingerprintAtom`. If the balancer decides to generate a new pattern it is registered at the `PatternRegistry`. The `PatternRegistry` is also used to get information about the current status of $\mathbb{P}$. The `PatternBalancer` further handles the merging of patterns. Therefore the `PatternBalancer` counts the lines in which two patterns have the same valid merge candidate. The verification that a merge is possible fails, if any line occurs, where the merge candidate can not be verified. If a merge is performed, the `EventClassificationBalancer` handles the effects on the event classes (see Tab. 3.11).

Figure 4.11 shows a sequence diagram of the search pattern generation process. If the `PatternBalancer` has enough tokens to generate a new pattern it first generates a candidate. The balancer then checks the candidate already exists in $\mathbb{P}$. If the candidate does exist already the `PatternBalancer` generates a new candidate and performs the check again. New candidates are generated for a certain number of times. If the `PatternBalancer` cannot find a new candidate it aborts the generation. If the pattern candidate does not exist yet, the `PatternBalancer` generates a new search pattern notifies the registry. Additionally the balancer checks every new pattern against all old patterns in $\mathbb{P}$ that match the current log-line, to find a new merge candidate. If a merge candidate is found it is initialised and gets evaluated in future log-lines.

Figure 4.12 shows a sequence diagram of the algorithm that handles the effects of pattern merges on event classes. It adapts $\vec{C_m}$ and $\vec{C_v}$ depending on the cases given in Tab. 3.11. These changes also influence the dependencies on different patterns involved in the merge.

For reasons of completeness Fig. 4.13 shows the class diagram of the `EventClassificationBalancer` while Fig. 4.14 shows the class diagram of the `CorrelationRuleBalancer`.

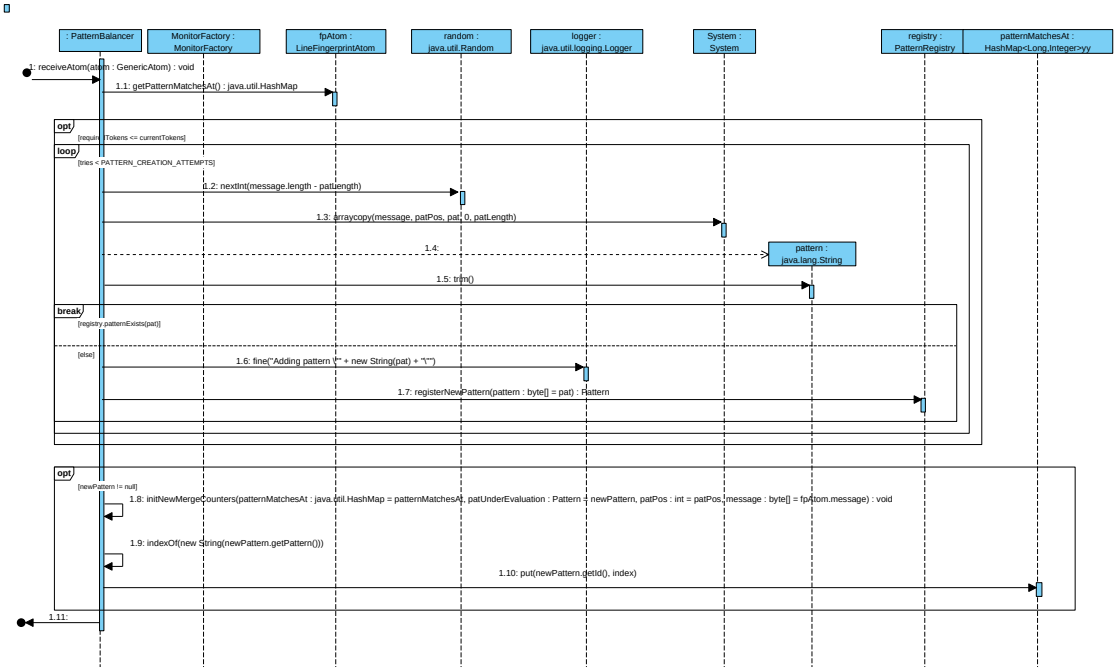


Figure 4.11: Sequence diagram showing the balancing algorithm on the sample of patterns.

4.3.3 Aging Components

Not all generated atoms are useful to the system. Invalid or deprecated atoms should get deleted automatically as soon as a deletion is possible without affecting other atoms. Aging components fulfil the task of cleaning up the system model M. Figure 4.15 shows a class diagram of the aging components. The diagram also shows how registries and aging components affect each other. The aging components just manage the internal state of the atoms that reflects if an atom should be deleted. The actual deletion process is triggered. Deletion is then performed by the registry if all conditions hold.

Note that deletion of atoms can not only be triggered by the aging component. One example would be the merging of two search patterns. In that case one pattern will be extended and the other one will be deleted. This procedure is valid, since the extended pattern substitutes the deprecated one.

Figure 4.16 gives an overview about the method calls involved in the periodic aging process, using the example of event class aging. For every processed log-line, a loop over all event classes in $\mathbb{C}$ checks, if a certain event class is part of a hypothesis. This check is passed if the condition in Eq. 3.33 holds. A score is used to count the number of times that the described check fails. This score measures the number of lines that can be processed before C gets deleted, given that C is not used during this time span. Whenever the score is reduced, the registry is triggered to check all deletion conditions (see Eq. 3.32 and Eq. 3.33). The registry is also the component that performs the actual deletion.

A very similar sequence of operations can be observed in the pattern aging component. This

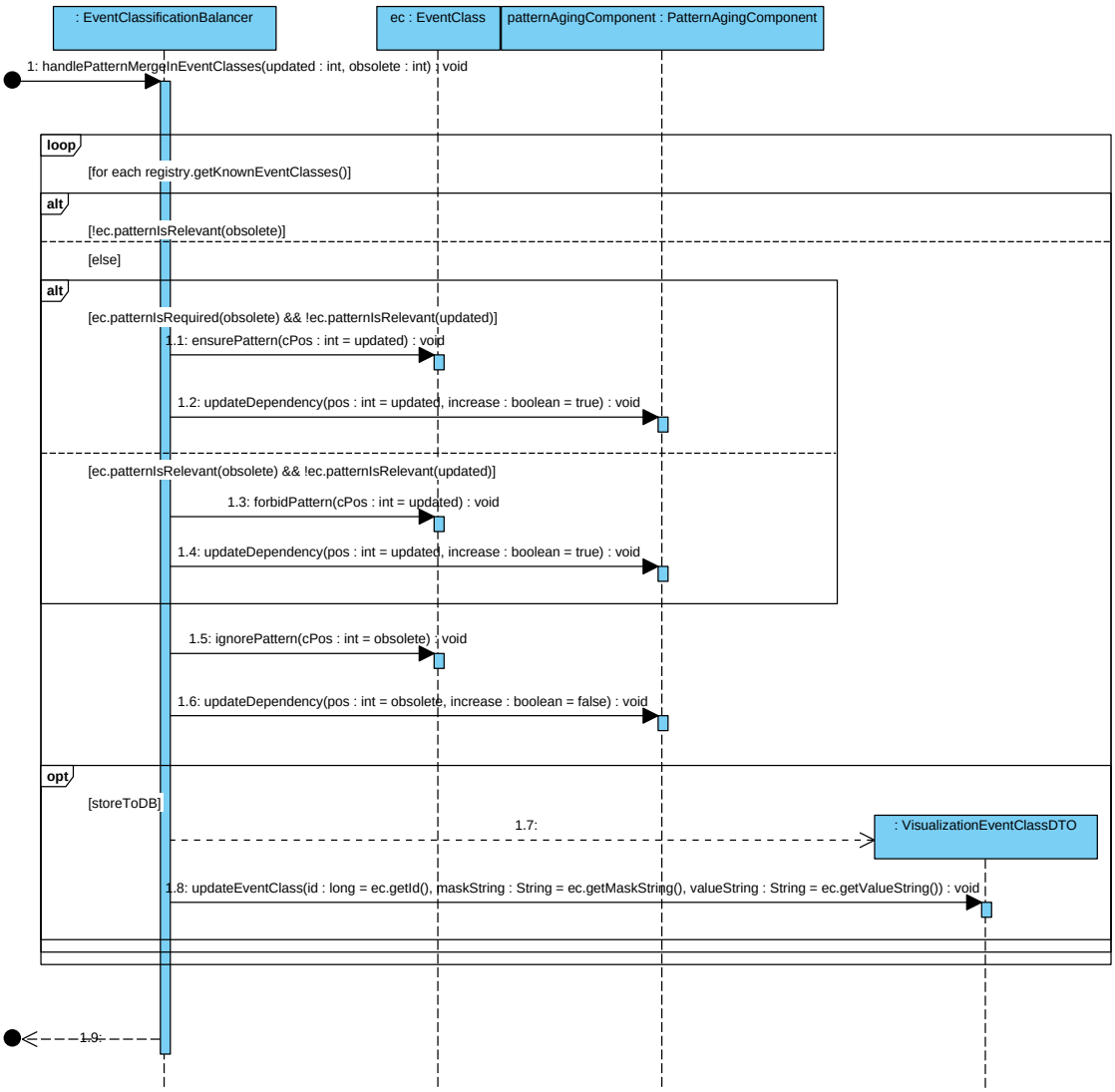


Figure 4.12: Sequence diagram showing the handling of merge effects by the `EventClassificationBalancer`.

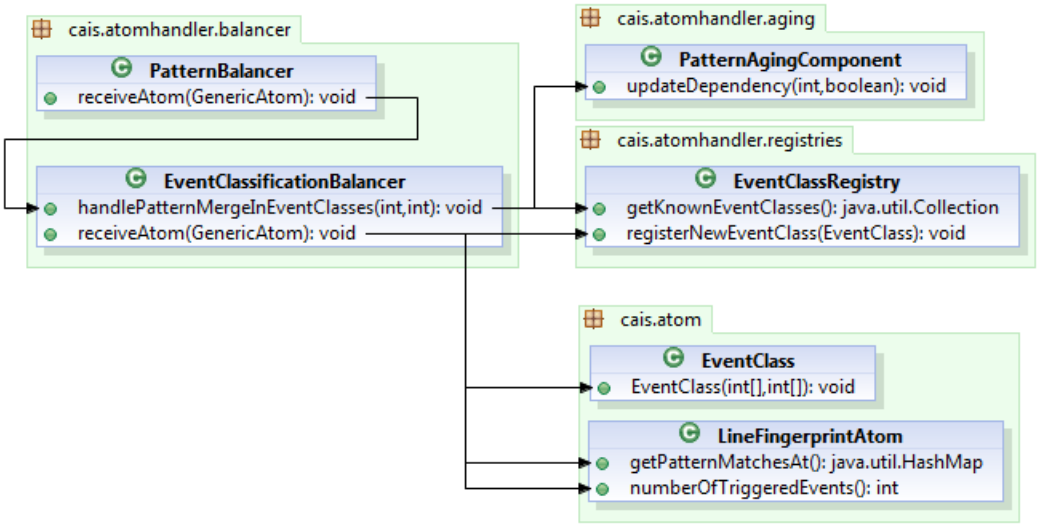


Figure 4.13: Class diagram showing the interactions triggered by the `EventClassificationBalancer`.

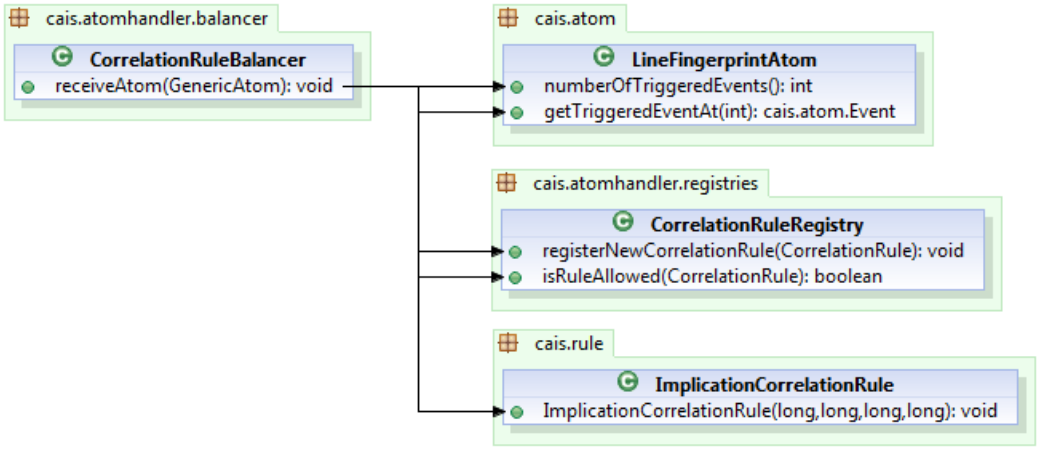


Figure 4.14: Class diagram showing the interactions triggered by the `CorrelationRuleBalancer`.

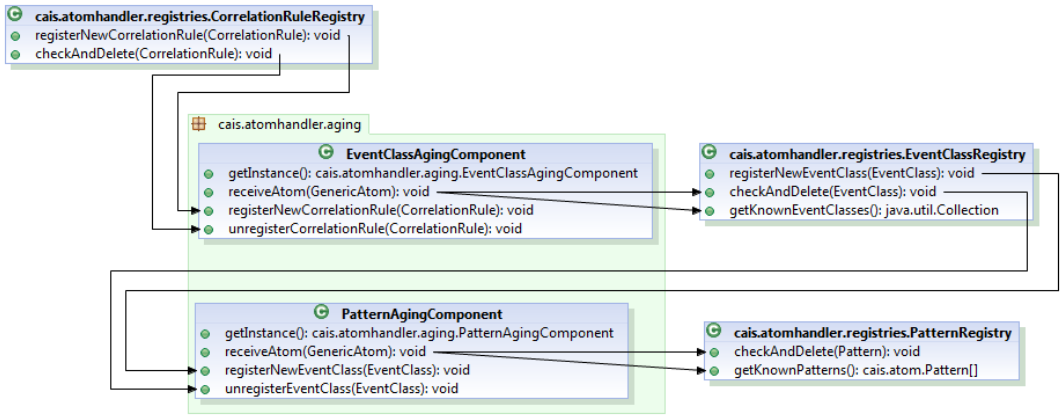


Figure 4.15: Class diagram showing the interactions between aging components and registries.

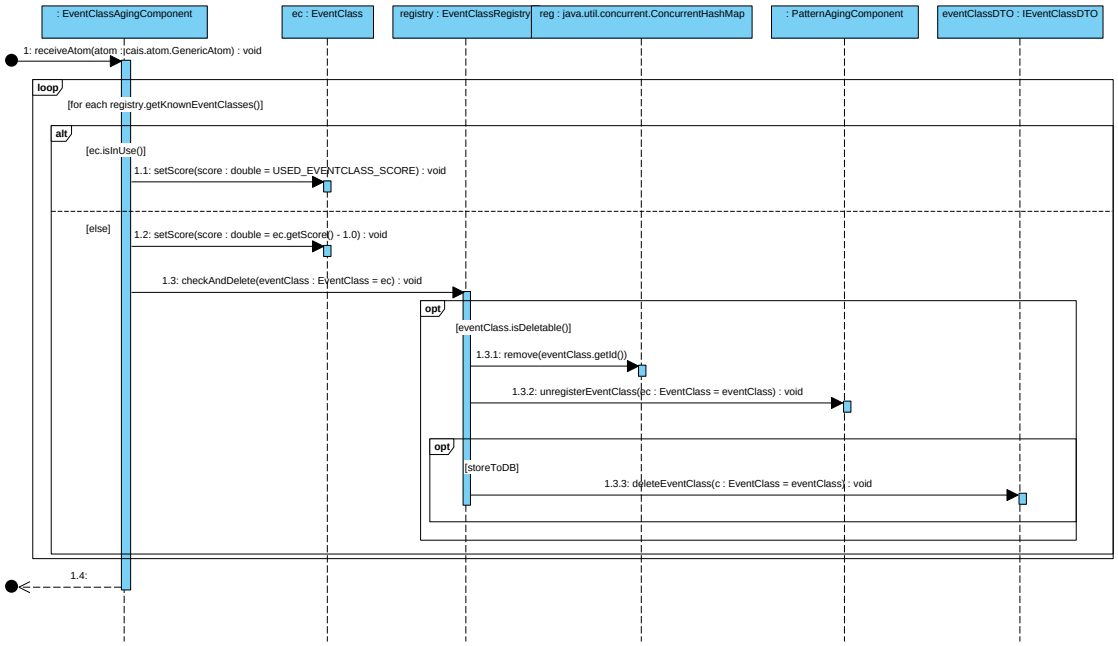


Figure 4.16: Sequence diagram visualising aging on the sample of event classes.

sequence is not explicitly shown here for simplicity reasons.

4.3.4 Hypothesis Evaluation

In the current implementation, hypotheses and rules can only be distinguished by their internal state of stability. The `RuleStabilityEnum` describes the possible stability states. The possible states are:

STABLE The `CorrelationRule` is $\in \mathbb{R}$.

UNSTABLE The `CorrelationRule` will be deleted.

UNDECIDABLE There are not enough evaluations of the `CorrelationRule` yet to decide about the stability. The `CorrelationRule` will be further evaluated and is considered $\in \mathbb{H} \backslash \mathbb{R}$

Every `CorrelationRule` stores its evaluations in various slots (see Eq. 3.15). The number and the size of these slots are given at start-up time (see Tab. 4.1) and they are generated by a `SlotFactory` for every newly generated hypothesis. Figure 4.17 shows a class diagram of the `SlotFactory` as well as the types of slots. Every new evaluation of the hypothesis is pushed to all slots. As described in Sect. 3.3 the last and biggest slots have a special meaning. It is used to describe the distribution that is assumed for the hypothesis. It is important, that evaluations that indicate an anomaly are not considered by this last slot. It therefore buffers elements before considering them. If there is an anomaly detected in the smaller slots, the buffered evaluations are dropped. Therefore, the distribution of positive and negative evaluations in the last slot is only updated with evaluations from stable phases.

Slots also store, if their evaluations contain an anomaly. Since every slot can detect an anomaly it probable that the same anomaly is detected several several times by different slots. This is prohibited by ensuring that a rule can only detect a new anomaly, if all slots were anomaly free before. Otherwise the already detected anomaly is considered ongoing.

Figure 4.18 shows a class diagram visualising the relationships between `CorrelationRules` and the `IRuleStatisticsCalculators`. Statistical evaluations over the slots are performed by the `IRuleStatisticsCalculator` implementation. This involves checks on the stability status of the hypothesis as well as checks about anomalies on rules.

The implementation has no aging component for rules. This functionality is implemented at two different locations in the code. The `CorrelationRuleChannelSource` handles aging due to a lack of evaluations (see Eq. 3.38). Figure 4.19 shows the call sequence in this case. All rules get evaluated, if new relevant events for a rule got triggered by the current line. If a hypothesis is unstable and no evaluations can be performed over a long period of time, the hypothesis is considered invalid and gets deleted.

Figure 4.20 shows a sequence diagram of the `evaluateRule()` method. In this method all events, that were not yet processed, get analysed and the rule evaluations get pushed into the slots. After storing the statistical analysis results into a persistent database (if this feature is activated), the method evaluates the stability of the rule. Hypotheses are not further considered, while unstable rules get deleted (see Eq. 3.37). For stable rules the method checks, if any slot

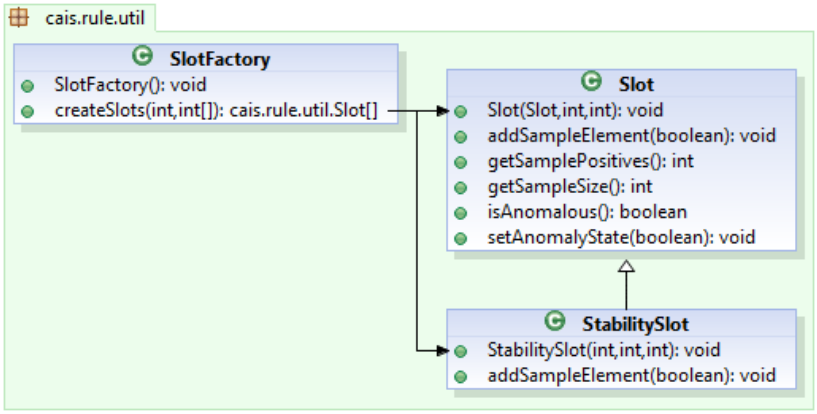


Figure 4.17: Class diagram showing slot types and the `SlotFactory`.

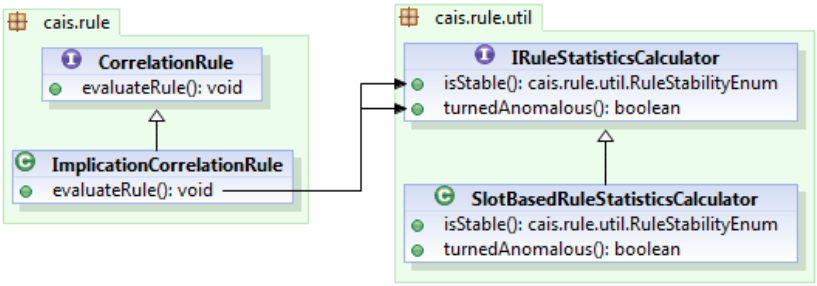


Figure 4.18: Class diagram showing the relation between `CorrelationRules` and the statistical utility class `IRuleStatisticsCalculator`.

turned anomalous after the last evaluation. If none was anomalous before, the method triggers an anomaly and stores it into the database. Otherwise the status does not change and the evaluation is finished.

4.3.5 Persistent Storage

The prototype implementation provides a persistent storage utility. A PostgreSQL[6] database can be used to store the system model as well as statistical measurements of the hypotheses generated by the system. This persistent storage acts as an interface for external applications such as visualization tools or evaluation scripts (see Ch. 6). Figure 4.21 shows an Entity-Relationship diagram, visualizing the database used to store the happenings in the system. The

[6]PostgreSQL is an open source, object-relational database system. See http://www.postgresql.org/ for more details.

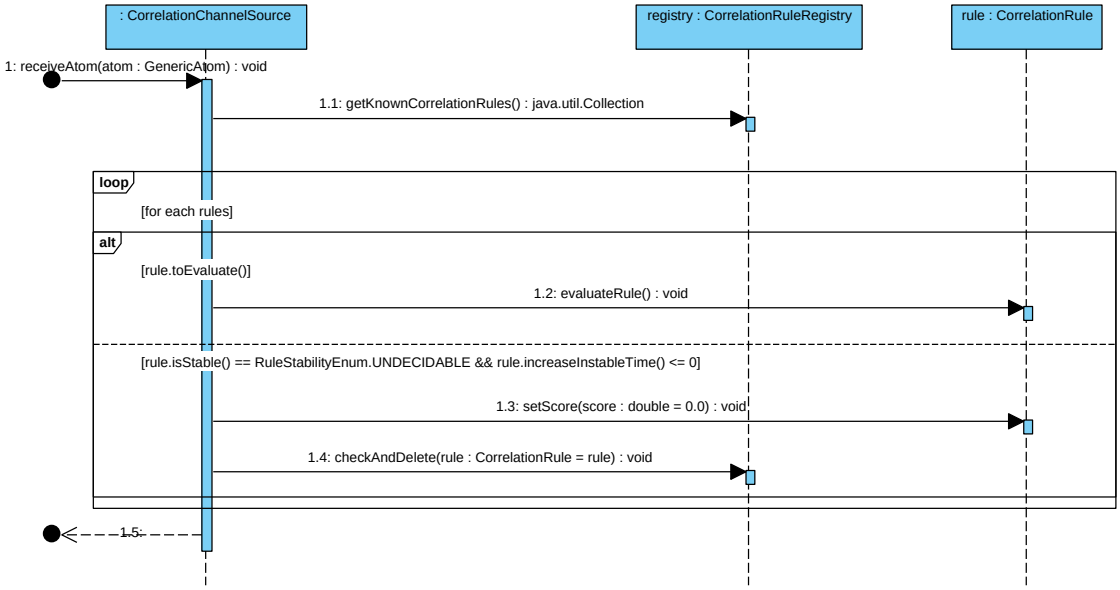


Figure 4.19: Sequence diagram visualizing rule aging in the `CorrelationRuleChannelSource`.

current implementation of the persistent storage utility does not support various numbers of slots. If persistent storage is activated, the number of slots used to statistically evaluate rules is fixed to four (one being the `StabilitySlot`).

The use of persistent storage is not mandatory. Anomalies can also be extracted by looking into the log-files produced by the system.

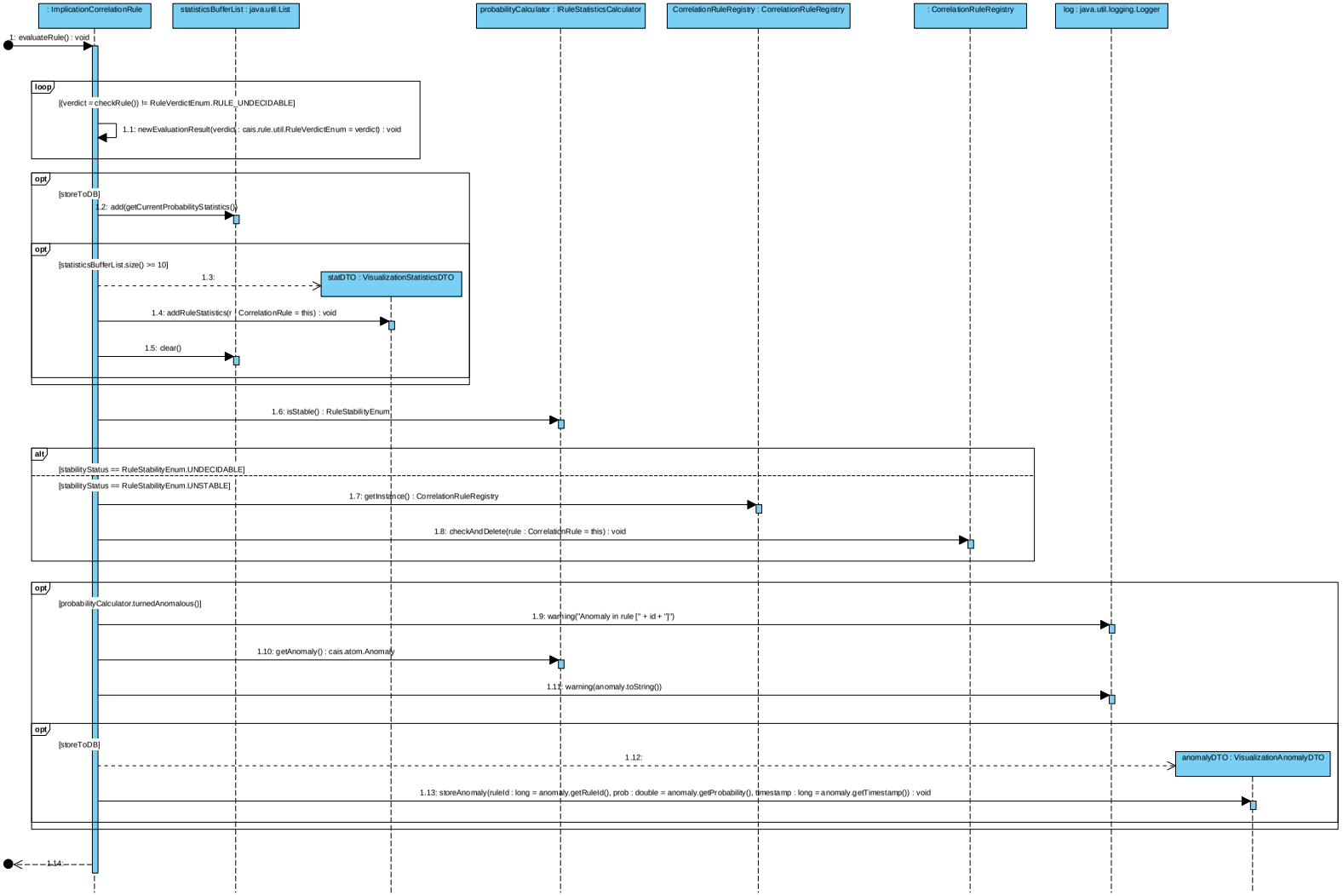


Figure 4.20: Sequence diagram visualizing rule aging in the `CorrelationRuleChannelSource`.

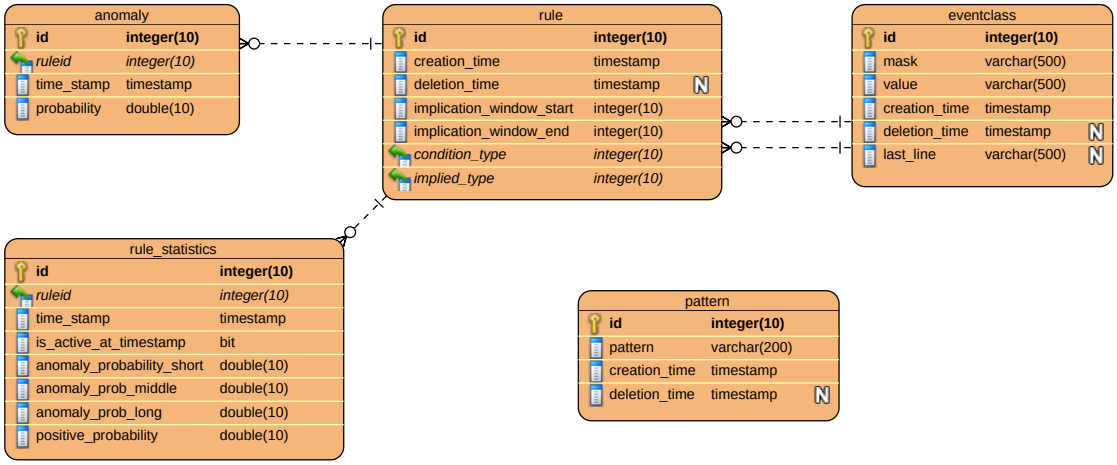


Figure 4.21: Entity relationship diagram of the database used to store the system model and measurement values.

CHAPTER 5

Test Environments

This chapter describes the different test environments. The datasets that are generated by the described environments are then used in Ch. 6 for detailed evaluations of the prototype implementation from Ch. 4.

5.1 ICT Environment

5.1.1 Data Generation Approach

A accurate evaluation relies highly on the quality of the data-set used to evaluate a system. Table 5.1 lists requirements for test data. A datageneration approach has to make tradeoffs, since not all of these requirements can be fulfilled in an optimal way at the same time.

Especially in the early evaluation stages, datasets from productive systems are hardly usable. The complexity is naturally very high and it is hardly possible to analyse these datasets prior to evaluation. But without an analysis of the dataset, an evaluation is not possible; the evaluator would not know what to look for. One solution to this problem can be the use of publicly available, annotated datasets (see Sect 2.4), but such datasets are hard to come by and often have limitations of their own. The usage of private datasets on the othe hand, hampers the credibility Often such datasets cannot be published due to privacy and security reasons, what makes the results non-reproducible. Completely synthetic datasets have a reduced complexity and are generated based on information about controlled environments. They seem good choices for early evaluations but they can only provide a very simplified view on system behaviour. It is hard to generate realistic datasets with different complexity, solely by synthetic means.

The approach taken in this thesis is semi-synthetic; a hybrid approach between synthetic data generation and collection of productive log-data: A virtual ICT network is stimulated by virtual users. These virtual users are implemented by scripts and are executed on separated virtual machines. They simulate realistic user behaviour in terms of service interaction properties. The data for later evaluation is recorded from real systems in the virtual setting. The result is very similar to data generated by a similar productive setting but without possible noise coming from

data property	*explanation*
large size	Scale and amount of the generated data is comparable to data generated by productive networks.
realistic	The data is comparable to data generated by productive networks in terms of distribution, variability, random noise, complexity etc.
analysable	Either the input is completely controlled or the data is completely annotated. Otherwise the results of the system under test cannot be verified.
easy to generate	Data for different test setups can be generated in a time efficient way.
easy to adapt	Coverage of various test conditions and purposes has to be achievable.
configurable complexity	Data with different complexity can be generated to cover basic scenario tests as well as detailed evaluations.
extensible generation	The generated data can adapt to new trends in the monitored systems.

Table 5.1: Required data properties.

data origin	*advantage*	*disadvantage*
synthetic	easy to (re-)produce, has desired properties, no unknown properties	no realistic 'noise' mostly simplified situations
real	realistic test basis	bad scalability (user input, varying scenarios), privacy issues, attack on own system required
semi-synthetic	more realistic than synthetic data, easier to produce than real data	simplified and biased if an insufficient synthetic user model is applied

Table 5.2: Test data origins and their pros and cons.

malicious users or users with erroneous behaviour. Table 5.2 gives a brief overview of the three possible types of test data generation.

Figure 5.1 illustrates of the implemented test-data generator. It operates in a distributed virtual set-up on three different layers.

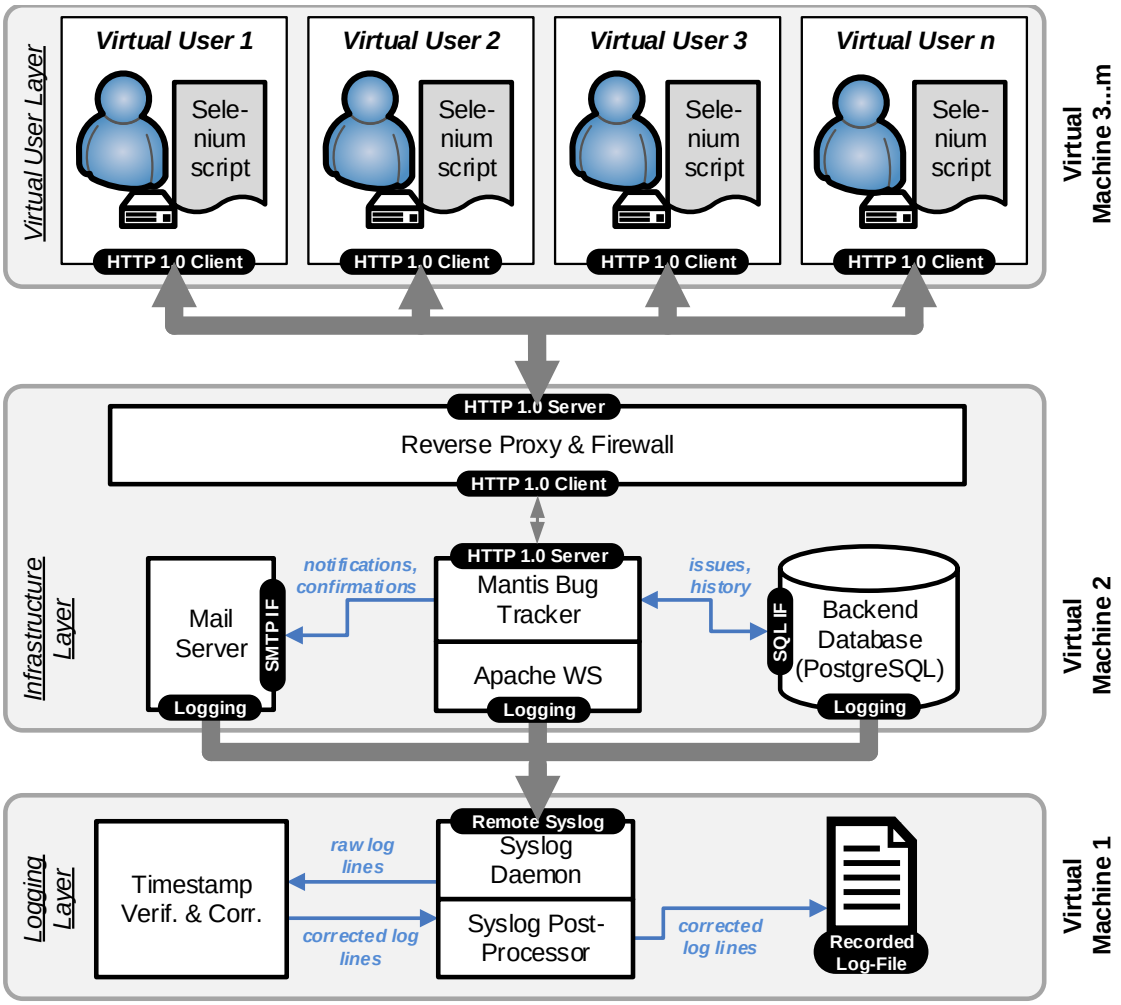


Figure 5.1: Overview of the semi-synthetic log-file generation approach.

Logging Layer

On the bottom layer, log-lines from various sources in the *Infrastructure Layer* get aggregated. A remote syslog-daemon[1], on a virtual machine, collects all log-events from registered systems. A *Timestamp Verification and Correction Unit* then checks the lines, before they are redirected into one central log-file. The *Timestamp Verification and Correction Unit* checks the log-events for correct timely order and reorders the lines if needed. The overall result of the data generator is one central log-file that contains all log-events, from all registered components in the *Infrastructure Layer*, in a timely sorted manner. The evaluation is performed on this recorded log-file. That way, evaluation results are reproducible and a more time effective evaluation is possible.

[1]Syslog is a standardised protocol for log message transmission defined in RFC 5424. See http://tools.ietf.org/html/rfc5424 for more details.

Infrastructure Layer

To generate realistic data, the *Infrastructure Layer* contains a virtual ICT network. For the evaluations in Ch. 6, the network consists of four services that are wrapped in one virtual machine. A reverse-proxy makes these services available; a firewall filters all access coming over the reverse-proxy. The following components build the network infrastructure:

Mantis Instances: One or more Mantis[2] instances get deployed on an Apache Webserver[3] and allow multiple users to manage bugs and tickets that are persistently stored in a MySQL[4] database. Mantis further sends notifications about assignments via the mail server to the mail addresses provided by the users. Different test scenarios use a different number of Mantis instances. The complexity of the generated data can be increased by increasing the number of instances as well as by increasing the number of virtual users.

MySQL Database: This database server is used by all deployed Mantis instances. Separated databases are used for different Mantis instances but they are all based on one database server.

Mail Server: The mail server is used by all Mantis instances to notify the users about assignments of tasks or newly generated tickets. Similar to the database server, the mail server is used by all deployed Mantis instances simultaneously.

This network could be arbitrarily extended by additional services. For our evaluations the complexity of the described layout is sufficient.

Virtual Users Layer

On the top layer a configurable number of virtual users are deployed that simulate user interaction with a virtual network. Selenium[5], a browser automation tool, simulates the virtual users based on a Selenium script. Every virtual user repeatedly executes various use cases; in random order and for a given time. Between the actions in the browser, realistic timeouts are inserted to simulate realistic speed of user actions. Each use-case has a certain probability to be executed once a user finished its last use-case. This ensures that actions, that are performed by real users more frequently, are also performed more frequently by the virtual users. Figure 5.2 shows a flow diagram of a virtual user's behaviour. Figure 5.3 refines the diagram by specifying the *Update Bug Report* subprocess.

[2]Mantis is a web-based, open source issue tracking system. See http://www.mantisbt.org/ for more details

[3]The Apache HTTP Server is an open source HTTP server. See http://httpd.apache.org/ for more details.

[4]MySQL is an open source, relational database management system. See http://www.mysql.com/ for more details.

[5]Selenium is a web browser automation tool. Originally developed for web application testing, its functionality is not limited to that. See http://docs.seleniumhq.org/ for more details.

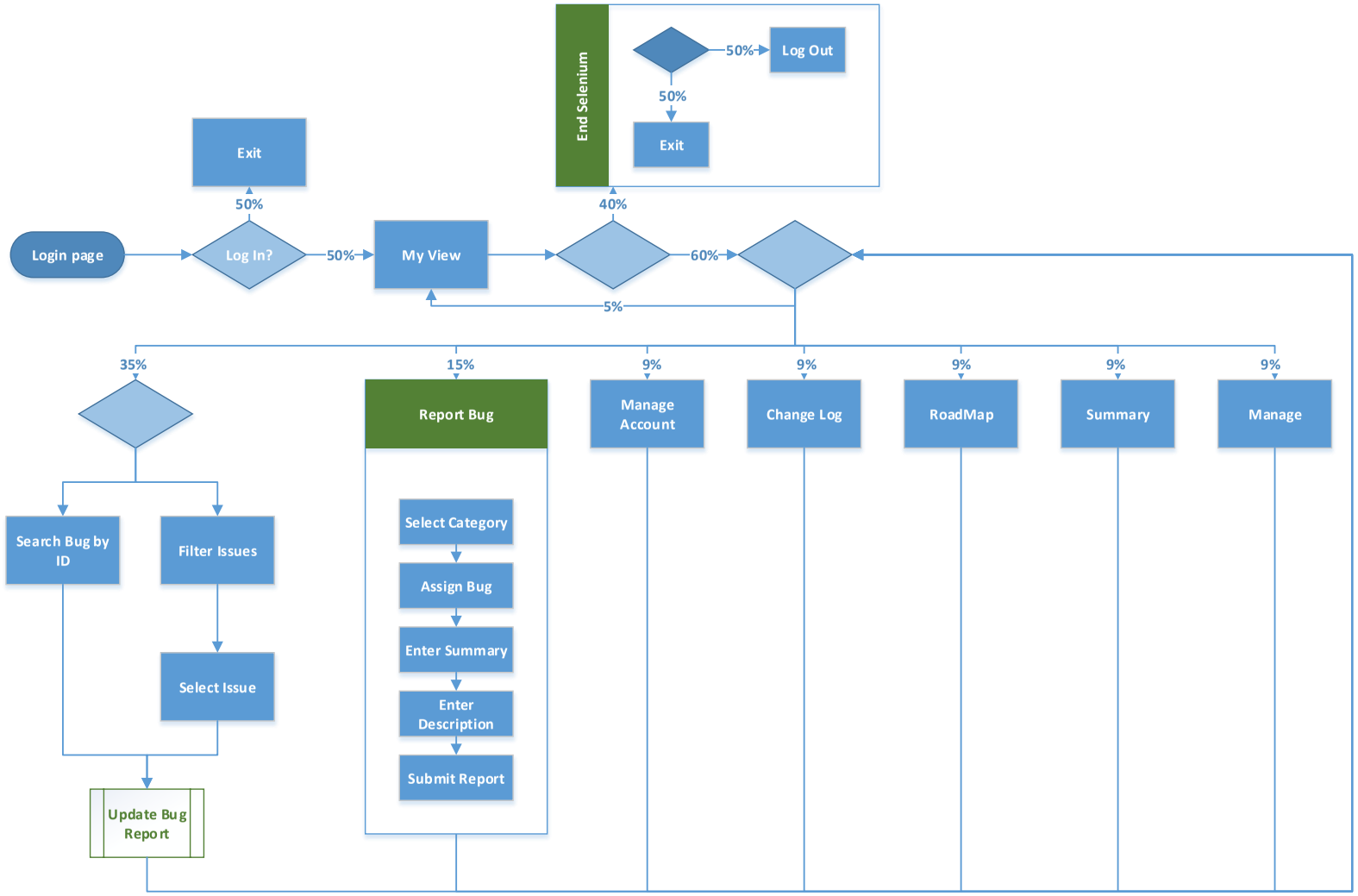


Figure 5.2: A flow diagram of the actions the virtual user can take and their probabilities.

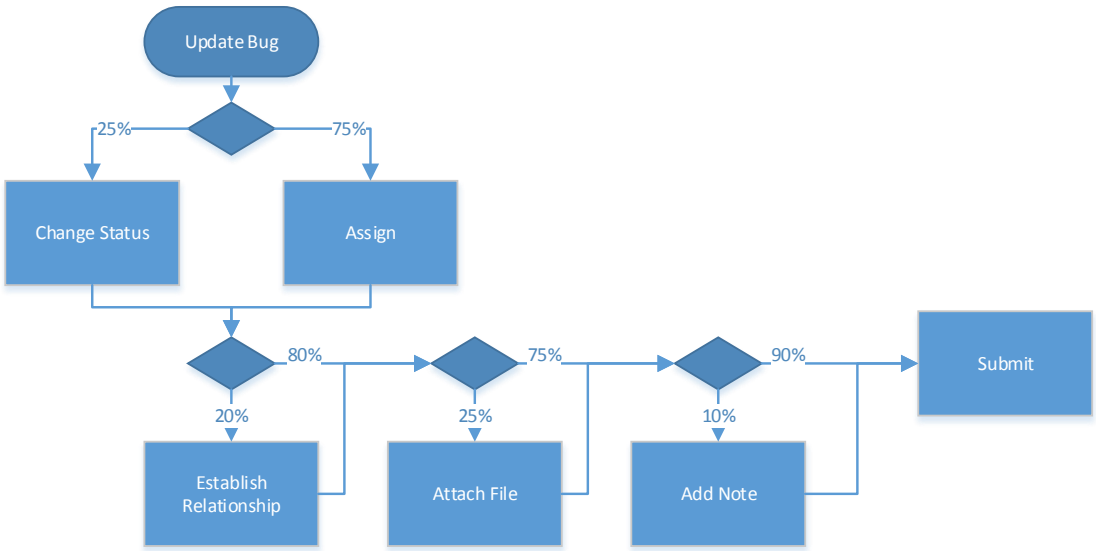


Figure 5.3: Flow diagram of the *Update Bug Report* process from the flow diagram in Fig. 5.2.

5.1.2 Data Generation Approach Evaluation

The behaviour of virtual users is based on an analysis of user behaviour in a productive Mantis system over 3 months. Figure 5.2 describes the results of this evaluation by denoting the probability of each branch at every decision point in the diagram. The diagram gives an overview of average user behaviour. Of course these probabilities might vary depending on the user's role in the team (e.g. developer, software tester, project manager, etc.). Based on the results of this analysis the virtual user decides its next actions.

The evaluation of the semi-synthetic approach was performed with 5 and 25 virtual users. Each case was performed 5 times in order to reach an averaging effect. This is important since the approach is non-deterministic. Each run recorded 15 minutes of log-data. Table 5.3 shows the number of lines generated by the different components in the network. The database server generates well over 90% of all log-lines. This has two reasons:

i Each request to the webserver triggers multiple SQL commands in the backend. The fact that virtual users only interact directly with the webserver, automatically results in a higher number of database log-lines.

ii The log-output produced by MySQL for one SQL command is split over multiple log-lines (see Lst. 5.1). A post-processor could be used to parse and merge the SQL commands into one line. But this would alter the original output and the approach would lose its general applicability.

```
1 Feb 12 13:30:16 database-0.v3ls1316.d03.arc.local mysql-normal #011#01158177 Query#011
      SELECT DISTINCT p.id, p.name, ph.parent_id
2 Feb 12 13:30:16 database-0.v3ls1316.d03.arc.local mysql-normal #011#011#011#011#011
      FROM mantis_project_table p
```

	5 users		25 users	
Log Source	*min.*	*max.*	*min.*	*max.*
Web Server	926	1029	3063	4308
Database	72150	78589	240889	305068
Reverse proxy	916	1018	3022	4217
Total log lines	74072	80521	247231	313593
avg.lines/minute	4938	5368	16482	20906

Table 5.3: Log data production performance during a 15 minutes run.

```
3 Feb 12 13:30:16 database-0.v3ls1316.d03.arc.local mysql-normal #011#011#011#011#011
      LEFT JOIN mantis_project_hierarchy_table ph
4 Feb 12 13:30:16 database-0.v3ls1316.d03.arc.local mysql-normal #011#011#011#011#011
      ON ph.child_id = p.id
5 Feb 12 13:30:16 database-0.v3ls1316.d03.arc.local mysql-normal #011#011#011#011#011
      WHERE p.enabled = 1 AND
6 Feb 12 13:30:16 database-0.v3ls1316.d03.arc.local mysql-normal #011#011#011#011#011
      #011 ph.parent_id IS NOT NULL
7 Feb 12 13:30:16 database-0.v3ls1316.d03.arc.local mysql-normal #011#011#011#011#011
      ORDER BY p.name
```

Listing 5.1: MySQL log excerpt from test environment.

Quality Analysis

Real Mantis Usage Reference Data set. To rate the types and the frequency of virtual user actions, the log-file of a Web server, running a Mantis instance used by real users in a software development team[6], was analysed. The analysed log-file contained the actions of 25 users over the time period of one working day. The file contained a total of 492 832 (Apache) web server requests composed of 291 different types. The distribution of the total number of requests among the different types is strongly exponential: the 10 most often issued request types account for half of all requests (approximately 250 000). Half of the request types, on the other hand, occurred less than 30 times.

Semi-Synthetic Data Set. The dataset generated by the semi-synthetic approach with 25 virtual users over 15 minutes contains between 3 063 and 4308 web server requests depending on the run. The dataset distinguishes 88 types of web server requests. The missing request types in, comparison to the real dataset, can be explained by the fact, that not all options available to a user are really implemented in the virtual user script. The exponential distribution of the request types throughout the real dataset suggests, that support of the most common operations is sufficient for a credible dataset. The semi-synthetic approach could generate much more web server requests in the same time, but it has the goal to achieve normal user behaviour. Therefore waiting times are inserted between opening a page and submitting a formular. Furthermore there are cases where a normal user opens a page, but waits several minutes before interacting. If those real-world cases are not an issue for the system that should be evaluated, the waiting times can also be reduced or removed to achieve a faster generation of web server requests.

[6]This was an anonymised, Ausstrian Institute of Technologs (AIT) internal dataset

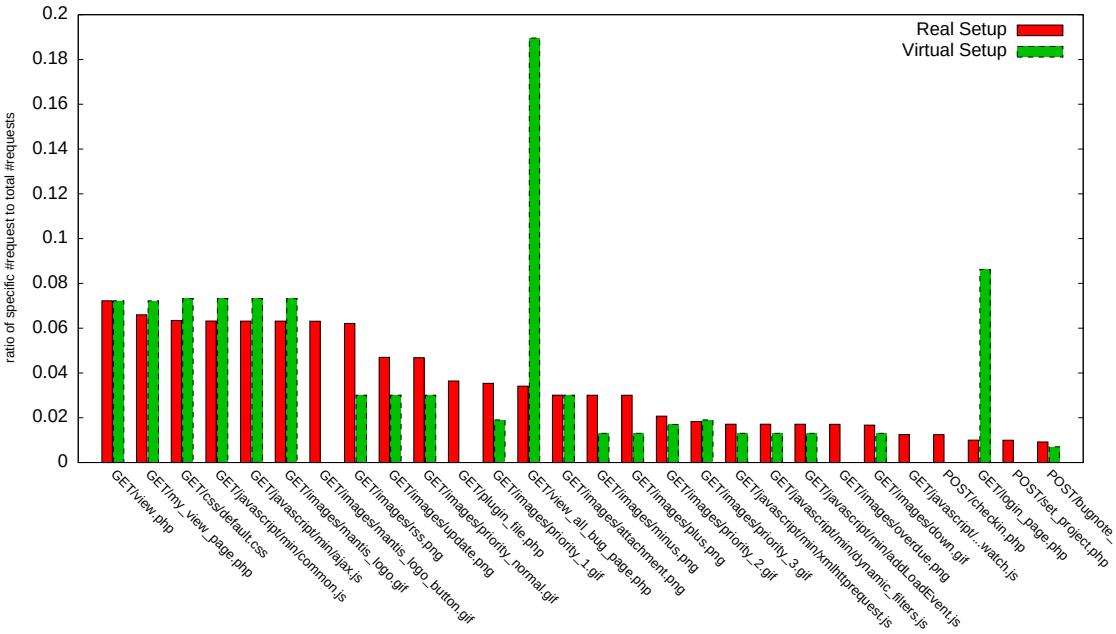


Figure 5.4: Distribution of user requests extracted from a data set from a real productive environment compared to request occurring in the simulation environment with virtual users. Notice: some user actions are not implemented in the virtual user model (e.g., `GET/plugin_file.php`). Moreover, the two outliers result from the fact that the simulated user does not use cookies to keep logged-in and does not use bookmarks to go to specific views directly, thus requests `GET/view_all_bug_page.php` much more frequently.

Comparison. Figure 5.4 shows the relative request rate of the most common request types for the real dataset as well as for the semi-synthetically generated dataset. Already the simple implementation that supports only 88 out of 291 request types achieves a realistic load on the system. A 100% accuracy is not realistic since neither the real user nor the virtual user scripts act deterministic. The goal of the semi-synthetic approach is to create realistic load and background noise on the system. When an attack is then injected into the system, the complexity to detect the attack is similar to the complexity in a real system.

Two significant outliers have to be considered separately. The request of `view_all_bug_page.php` is the starting point for all actions the virtual user performs. It is therefore called much more frequently than in a real system (users might use direct links or bookmarks). Furthermore virtual users do not use cookies to keep logged in. The login page is also requested significantly more often.

5.1.3 Datasets

The following evaluations in Ch. 6 use three datasets that were generated with the semi-synthetic setting described in Sect. 5.1.1. The first two datasets were recorded on a clean system. They will be used in the further evaluations of system parameters and performance when generating

	2U8h	4U12h
Recorded Time	8 Hours	12 Hours 30 Minutes
Simulated Users	2	4
Mantis Instances	1	2

Table 5.4: Differences between recorded datasets.

the system model M. During the generation of the third dataset different anomalies got injected in the system at different timestamps during recording. It will be used to analyse the system's performance in detecting anomalies. The following sections will describe the datasets in more detail.

Clean Datasets

Two datasets were recorded on a clean system. An evaluation of the generated system model requires a clean dataset. Anomalies in these datasets would lead to biased results because the effects of anomalies on the extracted figures cannot be calculated at this evaluation stage. Additionally, a solid evaluation has to show that the analysed results are not over-fitting the analysed dataset. The choice of two different datasets gives the evaluator the means to prove consistent results over various systems. Table 5.4 shows the differences between the two datasets.

One can see that dataset *4U12h* is in all aspects more complex than dataset *2U8h*. Not only the recording time, but also the number of simulated users and the number of simulated Mantis instances (all operating on one database server) is higher.

Anomalous Dataset

An additional dataset was recorded while anomalies were injected in the monitored system at different time periods. The configurations of the monitored system were equal to the ones taken when recording dataset *4U12h*. It contains 12 hours and 30 minutes of log information. During that time 4 virtual users used two different Mantis instances that operated on one database server. The dataset consists of two main parts:

i **Training Phase:** The first 7 hours of the dataset are recorded on a clean system. In this first phase no anomalies are injected. The data are used by the algorithm to build a clean system model M.

ii **Attack Phase:** After the *Training Phase*, two types of anomalies are injected several times in different time slots. One time slot is 30 minutes long and for each type of anomaly, four different slots are generated. Figure 5.5 shows a timeline of the anomalous phase.

Two different types of anomalies are injected in this dataset:

A1 At each injection point in a time slot *A1* in Fig. 5.5 a malicious script dumps all databases on the central database server. This dump is not performed remotely but directly on the

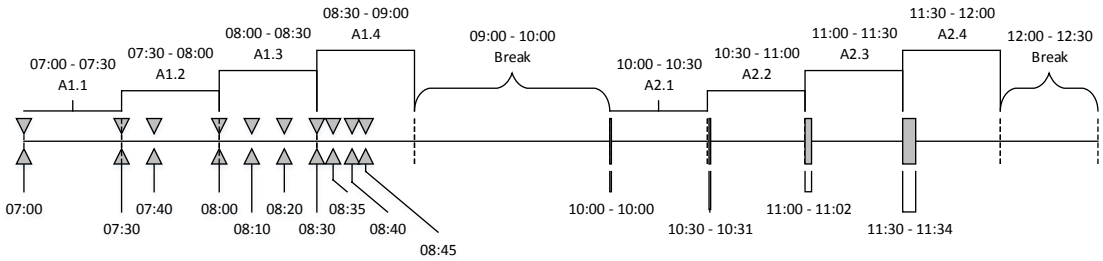


Figure 5.5: Timeline of the *Anomalous Phase* in the anomalous dataset

host of the database server. After dumping the database, the script uses the mail server to send the dump files to an external network.

A2 At each injection point in a time slot *A2* in Fig. 5.5, a malicious script disables the logging facility on the central database server, for a certain time period. This might be done in order to hide the attacker's tracks while tampering the database. In contrast to the first type of anomalous time slots, the simulated attacker increases the number of anomalies in the monitored system by extending the time span the logging facility gets turned off, rather than performing the attack more often. The time periods, in which logging is turned off, range from 30 seconds to 4 minutes; each time slot doubles the anomalous time span of the previous one.

The described injections are representative. Especially when talking about advanced persistent threats, the attack is often coming from inside the network. Sophisticated attacks often happened beforehand to enumerate and footprint various users. The actual harm is then done silently when extracting further sensitive information. Disabling or tampering logging facilities in order to cover up malicious actions is also a common approach.

5.2 SCADA Environment

One dataset is provided by an Austrian utility provider's SCADA infrastructure. The original dataset was recorded during one hour on a branch of a real network infrastructure from the utility provider's SCADA system that is coupled to the corporate LAN. The same dataset was already used in a previous evaluation of an early version of the approach in [19]. The monitored infrastructure consists of:

1. A firewall that records and filters incoming connections.
2. A switch that forwards traffic to and from the SCADA system.
3. A SCADA system that issues switching commands and provides measurement values on request.

Listing 5.2 provides two excerpts from the set of firewall logs in the dataset. The first line in the listing describes the semantics of the different fields. Notice that some information had to be removed or obfuscated due to confidentiality reasons. The firewall accepts and drops requests from the corporate LAN (and therefore also from the outside world).

```
1 "Number" "Date" "Time" "Interface" "Origin" "Type" "Action" "Service" "Source␣Port" "
      Source" "Destination" "Protocol" "Rule" "Rule␣Name" "Current␣Rule␣Number" "User" "
      Information" "Product" "Source␣Machine␣Name" "Source␣User␣Name"
2 "5487639" "6May2013" "10:59:58" "eth3.842" "mntfwp33" "Log" "Accept" "ntp-udp" "ntp-udp
      " "[removed-url].at" "[removed-url].at" "udp" "834" "-->␣RemoteNet␣to␣Sub" "834-
      MNT_ON_PX_RN_Global" "" "service_id:␣ntp-udp" "VPN-1␣Power/UTM" """"
3 "5515746" "6May2013" "11:02:22" "eth4.1151" "mntfwp33" "Log" "Accept" "cust-tcp
      -3505-3506-iec104" "52094" "[removed-url].at" "[removed-url].at" "tcp" "839" "-->␣
      remnet␣IEC␣104␣to␣VX" "839-MNT_ON_PX_RN_Global" "" "service_id:␣cust-tcp
      -3505-3506-iec104" "VPN-1␣Power/UTM" "" ""
```

Listing 5.2: Firewall log (excerpt from 221 lines).

Listing 5.3 shows two sample SCADA logs. Both lines store that measurement values from the system under supervision by the SCADA system got transferred to the requester. The transfer uses a connection that was previously established via the firewall.

```
1 Tele000592/06.05.2013 11:01:20,12/In /Source=4123/Len=21 Measured float/36 Cause=3()
      Number=1 Common=27/16 floating point Info/Obje=12/17/66 Val=5.97 QDS=0x00 Date/
      Time=06.05.2013/11:01:20,042 - IV=0 DST=1
2 Tele000593/06.05.2013 11:01:21,17/In /Source=4123/Len=21 Measured float/36 Cause=3()
      Number=1 Common=27/16 floating point Info/Obje=12/15/66 Val=99.65 QDS=0x00 Date/
      Time=06.05.2013/11:01:20,780 - IV=0 DST=1
```

Listing 5.3: SCADA log (excerpt from 2854 lines).

The log level of the switch is at a too restrictive level to produce log-events that are useful to the evaluation. The switch only logs complete failures e.g, if it is shut down.

Limitations. The monitored systems do not generate a high amount of log-events while they are in a normal state. The collected log file contains a total of 3 478 log-lines that are collected during one hour. This number of log-lines is not sufficient for an evaluation of the proposed approach. In order to achieve a reasonable number of lines, the existing hour is duplicated and the log-file is extended to represent recordings of 10 hours of recorded data. This duplication results in a highly periodic dataset. But given the very stable nature of the log-lines the approach is applicable.

The resulting dataset contains 30 310 log-lines and simulates recordings of 10 hours.

Injected Anomalies

Three anomalous datasets are generated from the basic SCADA dataset. Due to the limitations of the dataset, we generate one separate file for every injected anomaly.

Anomalous Firewall Activity. The first injected anomaly consists of two lines that indicate connection attempts from the corporate LAN that were accepted by the firewall. In a time range of about 10 seconds after the connections were accepted, no measurement values are sent by the SCADA system.

This anomaly can either describe a malicious component in the corporate LAN that tampers with the SCADA system in an unexpected way. But it could also mean that the SCADA system is not responsive.

Invalid Values. During one minute all values that are transmitted by any sensor are 0.0 instead of the really measured values. All measurement points appear in the log-files and only the results are altered.

Like in the previous anomalies the reason can be an attacker who tampers, disables or overloads one sensor or it can be a sensor failure.

CHAPTER 6

Evaluation

The following chapter discusses the evaluation of the approach, performed on the dataset described in Ch. 5. A detailed evaluation of the model generation is given in Sect. 6.1 - 6.3. The evaluation starts with finding a solid setting of the configurable system parameters (see Sect. 4.2) in Sect. 6.1. Section 6.2 discusses the quality of the generated search patterns and event classes based on the input sets. Event classes are further evaluated regarding clusters of lines found in the input sets, which are also used in Sect. 6.3 to analyse the quality and expressiveness of rules.

Section 6.4 and Sect 6.5 discuss the core functionality of the system: the ability to detect meaningful anomalies. This is done in two application domains: Section 6.4 discusses the system's performance in the information system domain; Sect. 6.5 discusses the performance in the domain of SCADA systems.

6.1 Parameter Evaluation

As a first step this chapter evaluates the different parameters in the prototype implementation. One goal is to define a stable configuration of the system. Another goal is to get a practical impression about the influences different parameters have on the system model M and on the results produced by the system. Prior to the detailed evaluation a *Monitored System Behaviour Period* T has to be defined (see Sect 4.2). For the given datasets, a period of 20 minutes is chosen. Although the datasets are based on an ICT infrastructure, the recorded setting did not contain any backup facilities or periodic tasks except the simulated user input. Based on the evaluation of the data generation approach (see Sect. 5.1.1), the actions of the simulated users reach a request distribution, similar to a productive setting after 15 minutes. Since the *2U8h* dataset does only record a virtual system that is stimulated by two users, the period is extended with a buffer of 5 minutes, to be sure that the request distribution is comparable to the one in a real system.

The system parameters can be divided into three categories:

i *Utility parameters* that do not influence the direct result of the system. Examples are the parameters defining how the system can access the database to persist execution results.

ii *Input independent parameters* that have an optimal setting which is not influenced by the structure and complexity of the analysed dataset.

iii *Input dependent parameters* that have a different optimal setting depending on the structure and size of the analysed dataset.

Several qualitative and quantitative evaluation steps preceded the following categorization of parameters. Table 6.1 shows all parameters categorised as independent of the analysed input file; Table 6.2 shows all parameters with an optimal setting that depends on the analysed network. The following evaluation will focus on the settings of dependent parameters. Starting with a base configuration, each parameter is altered separately to analyse its effects on the resulting system model. Table 6.1 and Tab. 6.2 state the values of each parameter in the base configuration. This base configuration was determined in preceding iterations of evaluations of the same type. We do not go into detail about earlier iterations at this point. The results of the approach highly rely on a reasonable setting for all parameters. In early iterations several parameters were not chosen in a valid range which resulted in biased evaluation results. It was only possible after multiple iterations to get to a state where trends for each parameter could be detected. Additionally listing or visualising all iterations would exceed the scope of this chapter.

Starting from the described base configuration all parameters in Tab. 6.2 get changed to various values separately. This evaluation is performed on both clean datasets from Sect. 5.1.3 (namely *2U8h* and *4U12h*) in order to prohibit results that overfit one dataset. The figures extracted at the end of the execution can be seen as one arbitrary state of the system model during a continuing analysis. Higher absolute numbers in the results generated with the *4U12h* dataset, compared to the *2U8h* dataset are a result of the increased complexity of the recorded infrastructure rather than a result of a longer recording time. Each evaluation run will be evaluated based on six metrics:

Number of Patterns: Describes the number of search patterns in M (also: $|\mathbb{P}|$), at the end of the execution.

Number of Event Classes: Describes the number of event classes in M (also: $|\mathbb{C}|$), at the end of the execution.

Number of Rules: Describes the number of rules in M (also: $|\mathbb{R}|$), at the end of the execution.

% of Rules in $\mathbb{H}$: Describes the ratio between stable rules $\mathbb{R}$ and undecidable hypotheses $\mathbb{H}\backslash\mathbb{R}$ in M (also $\frac{|\mathbb{R}|}{|(\mathbb{H}\backslash\mathbb{R})|}$).

Number of Anomalous Rules: The number of rules $\mathbb{R}$ that detected any anomaly during the execution.[1]

[1] This is the only metric that is not completely independent of the execution time. A longer execution time would result in a higher number of anomalous rules since the chance of false positives increases if the analysed timespan increases. This fact is not considered, since this metric describes a total number and not a ratio depending on the time.

PatternBalancer Configuration: `cais.atomhandler.balancer.PatternBalancer.*`		
Parameter	*Setting*	*Description*
`minPatternLength`	3	The selection of search patterns is not based on separators (such as spaces or punctuation marks) but performed completely random. Therefore the length of the search patterns is considered independent of the input structure.
`maxPatternLength`	12	

EventClassificationBalancer Configuration: `cais.atomhandler.balancer.EventClassificationBalancer.*`		
Parameter	*Setting*	*Description*
`minEnforcedBitsPerMask`	3	The minimum of enforced bits is set to prohibit event classes that are too generic from being generated. The generality of an event class is input independent.
`classCreationAttempts`	2	Additional generation tries are just there to prevent the chance that an existing event class gets chosen again and is prevented from being generated.

Aging Configuration: `cais.atomhandler.aging.*`		
Parameter	*Setting*	*Description*
`numberOfInitialPeriods`	3	These aging parameters are all based on the chosen period. The choice of the period is the input dependent choice; the number of periods waited, before aging is enforced, is independent of the input.
`numberOfPeriodsForUnusedPattern`	3	
`numberOfPeriodsForUnusedEventclass`	3	
`numberOfPeriodsForUnevaluatedRule`	3	

Table 6.1: Parameters with optimal settings, independent of the input dataset.

General Configuration	
Parameter	*Setting*
`cais.logLinesPerPeriod`	30 000
PatternBalancer Configuration: `cais.atomhandler.balancer.PatternBalancer.*`	
Parameter	*Setting*
`baseCost`	1
EventClassificationBalancer Configuration: `cais.atomhandler.balancer.EventClassificationBalancer.*`	
Parameter	*Setting*
`baseCost`	1
`triggeredEventCost`	30
`prohibitedBitsPercentage`	50%
`enforcedBitsPercentage`	40%
CorrelationRuleBalancer Configuration: `cais.atomhandler.balancer.CorrelationRuleBalancer.*`	
Parameter	*Setting*
`baseCost`	30
`existingHypothesisCost`	300

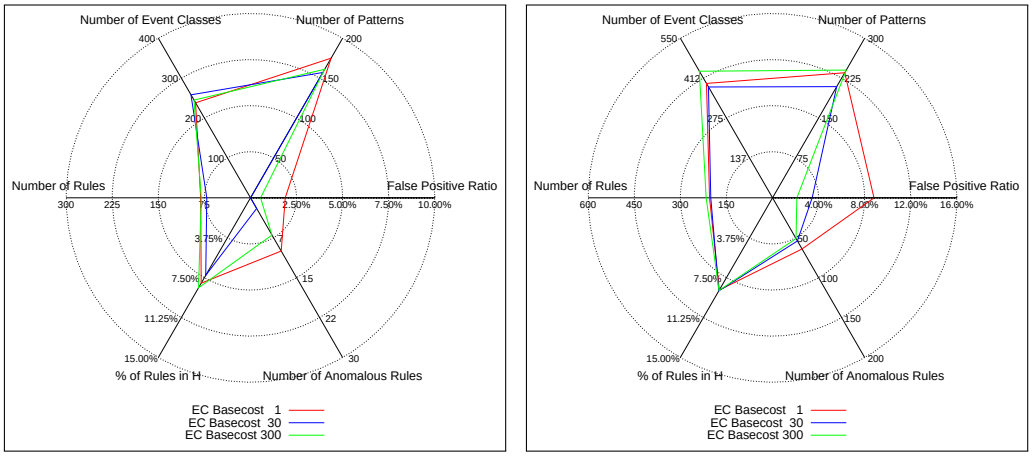
Table 6.2: Parameters with different optimal settings that depend on complexity and structure of the analysed network as well as the duration of the analysis.

False Positive Rate: Every rule is evaluated multiple times, during the runtime of the algorithm. Since we know that we are operating on a clean dataset, we expect no anomalies to be detected. We define a false positive as an evaluation of a rule that results: (i) in the rule entering an anomalous state, or (ii) in extending the anomalous state a rule is currently in. Thus not every negative evaluation of a rule is considered a false positive. This would be wrong since the system is designed to accept unique negative evaluations as accepted input. At the same time it would be wrong to consider every detected anomaly as a false positive. The problem here lies in the definition of true negatives. The only way to define true negatives is as: *Every evaluation of a rule, that did not trigger an anomaly*. This leads to the above definition of false positives.

Based on these metrics the effects of each parameter on the system are analysed. In the following graphs the red line always describes the results produced by using the base configuration; the other lines describe results produced by altering the parameter under evaluation. The possible values for the various cost-parameters are not randomly chosen but set as a fraction of the period T. The lowest possible value is 1 which is considered independent of T. The other values are $T * 1E^{-3}$ and $T * 1E^{-2}$.

6.1.1 Single Parameter Evaluation

Event Class Base Cost. The first parameter evaluated is the base cost of a new event class. In the base configuration the value is set to 1. That way, the algorithm is able to generate an event class whenever a log-event is not yet covered by any event class. Otherwise such log-events cannot be considered by the rule evaluations. This would mean, that the algorithm is blind regarding these log-events.

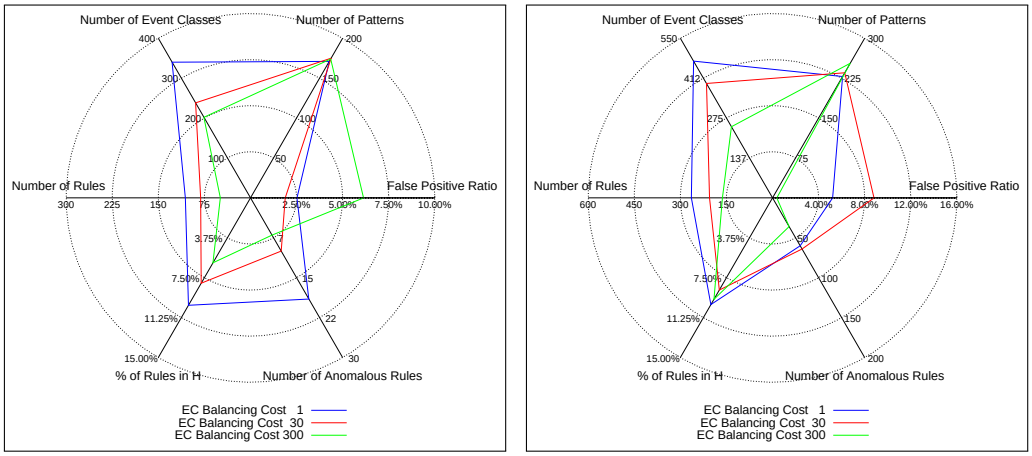


(a) 2U8h (b) 4U12h

Figure 6.1: Radar plots when changing base cost for generating a new event class.

Looking at the metrics visualised in Fig. 6.1 we can see that an increased base cost for event classes results in a trend towards a lower false positive rate. Note that there is no significant impact on the *Number of Event Classes*. The evaluation of the event classes performed in Sect. 6.2 will show that all log-lines are well covered by event classes. The main cost for a new event class that gets generated, is a result of the balancing cost of the event classes. Increasing the base cost of event classes only shifts the point at which $\mathbb{C}$ is saturated, to a later point during execution. We can further see, that the value, measured for the *False Positive Rate* with a base cost of 30 in the *2U8h* dataset, seems to be over-fitted. A trend towards 0 cannot be supported by the *4U12h* dataset. Changing the value of the event class base cost to 30 is one change we should remember for further evaluations.

Event Class Balancing Cost. The next parameter that gets evaluated is the balancing cost for a new event class. The base configuration sets a value of 30 since the number of event classes defines the size of the search space for new rules. It is therefore important to have a sufficient number of event classes so that meaningful rules get generated. At the same time it is important that the generation of event classes is not too excessive. A search space that is too big would result in rules describing the same relations by slightly different means. It also makes it harder to find the hypotheses that have the potential to get stable. Figure 6.2 shows the results of the evaluation.



(a) 2U8h (b) 4U12h

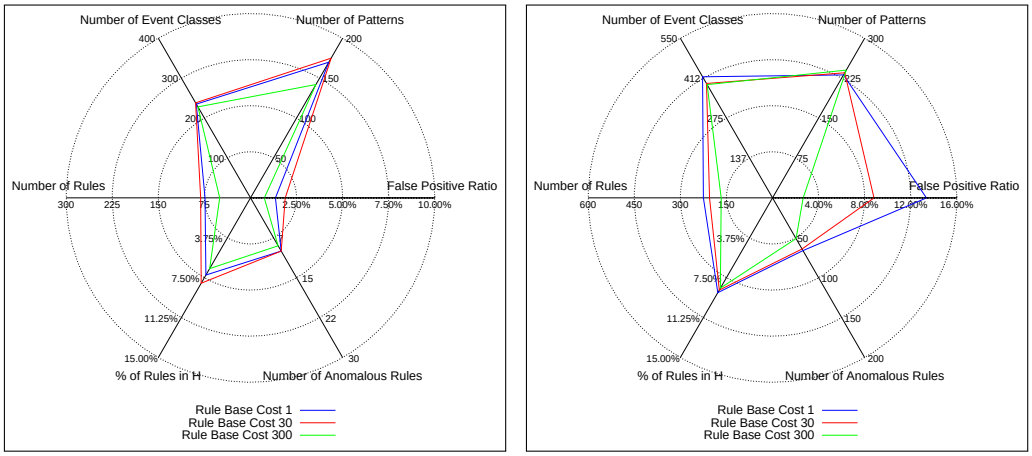
Figure 6.2: Radar plots when changing balancing cost when generating a new event class.

As expected there is an indirect proportional relation between the value set for the balancing cost of event classes and the *Number of Event Classes*. A second trend that can be shown is, that the number of event classes does not affect the number of patterns. An effect could have been expected due to the aging condition in Eq. 3.33. This relation is not supported by the data. The balancing cost of event classes further has significant effects on all metrics regarding rules, but no real trend can be extracted. A change of the balancing cost from 30 to 1 could get evaluated further, since a trend towards a higher number of rules can be shown.

Rule Base Cost. The same cost parameters from event classes are also evaluated for hypotheses. First we evaluate the hypotheses base cost. The base configuration sets a value of 30 for the hypotheses base cost. Hypotheses are the central resource to analyze the monitored system in order to detect anomalies. But they are also the most „expensive“ atoms to generate. A lower base cost might result in multiple hypotheses being generated, for event classes that describe the same class of lines, if these lines occur very frequently. On the other hand the hypotheses base cost should be kept in a reasonable range, so every event class is considered by hypotheses. Figure 6.3 shows the results of the evaluations.

As seen before when analysing the base cost of event classes, the hypotheses base cost has limited effect on the *Number of Rules*. There is a small trend towards a decreasing number of rules with an increasing base cost, but it is not significant. The trend towards a higher false positive rate that is suggested by Fig. 6.3b is not supported by Fig. 6.3a. Other effects are very limited.

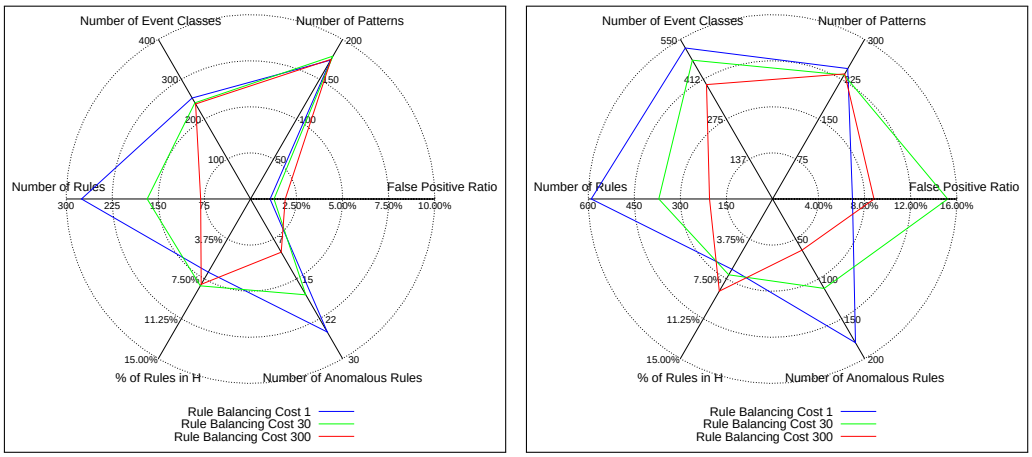
Hypotheses Balancing Cost. The hypotheses balancing cost parameter is probably the most influential parameter on all metrics regarding rules. This includes metrics that describe the internal system model, as well as metrics that describe the final output (i.e. detected anomalies).



(a) 2U8h (b) 4U12h

Figure 6.3: Radar plots when changing the hypothesis base cost.

The base configuration is very restrictive and fixes the hypothesis balancing cost at the highest value: 300. Figure 6.4 shows the results of the evaluations.



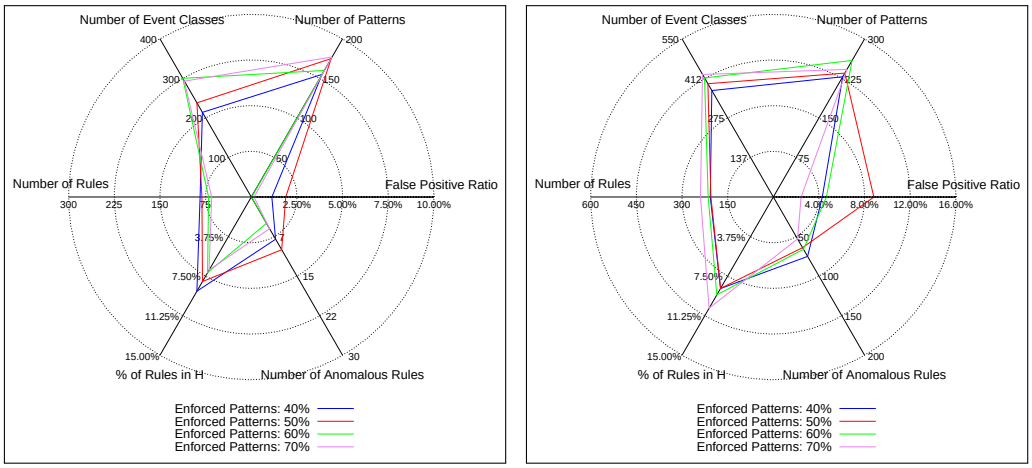
(a) 2U8h (b) 4U12h

Figure 6.4: Radar plots when changing hypotheses balancing cost.

Two trends are very obvious: First there is again a strong indirect proportional relation between the hypotheses balancing cost and the *Number of Rules*. This is interesting, because the parameter can only directly influence the number of generated hypotheses but not the number of hypotheses that get stable. The second trend is that as the *Number of Anomalous Rules* increases, so does the *Number of Rules*. Both trends result in two observations: The first observation is that a better coverage of the search space, at the cost of less effective balancing, does not

increase the relative number of discovered information (i.e. the percentage of rules $\frac{\mathbb{R}}{\mathbb{H}}$) The second observation is that, as the number of rules increases, so does the number of rules that turn anomalous. Both observations lead to the conclusion that increased search space coverage, at the cost of less effective balancing, does not generate genuine new information, but only reproduces information already part of the system model M.

Enforced Pattern Percentage. Generality of event classes is highly dependent on the percentage of possible search patterns that are enforced in a newly generated event class. Possible search patterns are all patterns that match the log-line that is used to generate the new event class. In the base configuration this value is set to 50%. This is a reasonable trade off between specificity of the new event class and the chance of a later generated event class to find another unique set of enforced patterns. Figure 6.5 shows the results of the evaluations.



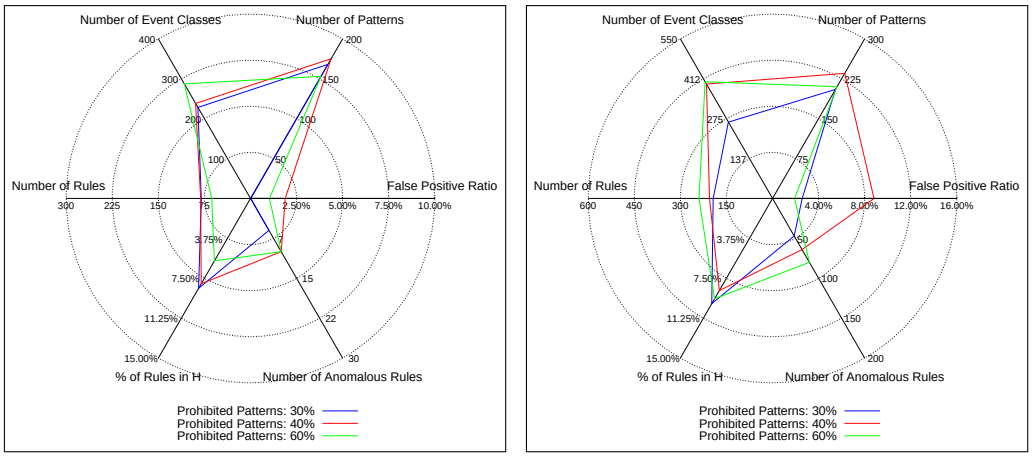
(a) 2U8h (b) 4U12h

Figure 6.5: Radar plots when changing the percentage of enforced patterns in a new event class.

The evaluations show no significant trends when this parameter is altered. A small, but not significant, trend towards a higher number of event classes can be found when increasing the percentage of enforced patterns. Otherwise the metrics stay more or less stable through out the different evaluations. It is a sufficient trade off to fix the percentage of enforced patterns at 50% but more specificity of event classes does not hamper the results.

Prohibited Pattern Percentage. Generality of event classes also depends on the percentage of patterns that are not matching the log-line used to generate the event class and are therefore prohibited. The base configuration sets a value of 40% for this parameter. It is infeasible to qualitatively analyse the effects of prohibited patterns on the class of lines described by the event class. It would include an analysis of all lines that are matched by the event class, only considering the enforced patterns. A high number of prohibited patterns might result in too

specific event classes. On the other hand, are prohibited patterns one mean to identify subclasses of the set of lines classified by enforced patterns. Figure 6.6 shows the results of the evaluations.



(a) 2U8h (b) 4U12h

Figure 6.6: Radar plots when changing the percentage of prohibited patterns in a new event class.

Clean trends are again hard to extract, but a choice of 30% of prohibited patterns seems optimal with respect to the *Number of Anomalous Rules* in combination with the *False Positive Rate*. A trend regarding changes in the number of event classes cannot be supported by both datasets.

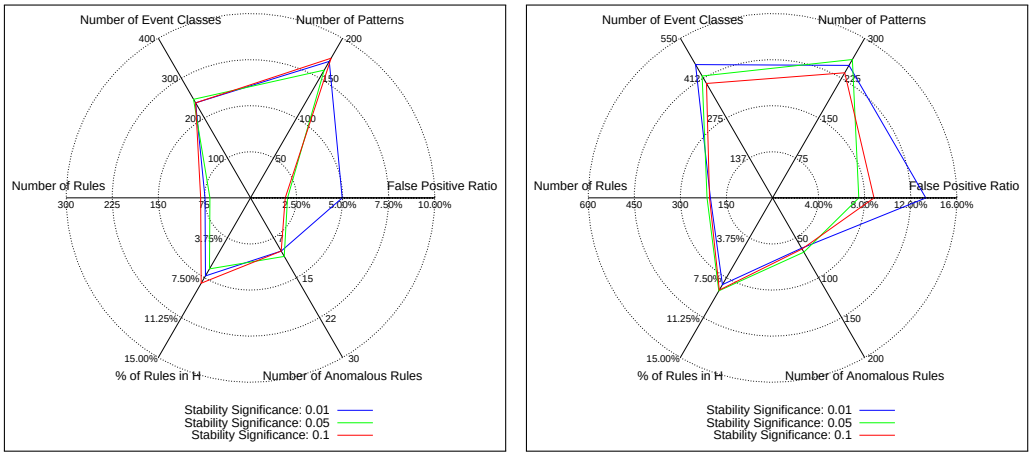
Stability Significance. The stability significance defines the significance of the stability test of hypotheses. It ensures that only highly reliable hypotheses get stable. The base configuration defines a significance value of 0.1. The evaluation should decide how a more restrictive approach affects the results.

The choice of the significance does neither influence the number of rules nor does it affect the percentage of hypotheses that get stable. This is a clean indication that a significance of 0.1, as chosen by the base configuration, is sufficient to restrict the stability decision.

6.1.2 Combined Parameter Evaluation

After evaluating each parameter separately, an optimal setting can be generated. Therefore, the combined configuration contains deviations from the base configuration regarding four different parameters:

i The base cost for event classes is increased from 1 to 30. The expectation is to see a decrease in the *False Positive Ratio* and in the *Number of Anomalous Rules*.



(a) 2U8h

(b) 4U12h

Figure 6.7: Radar plots when changing the significance of the stability check for hypotheses.

ii The base cost of hypotheses is decreased from 30 to 1. The expectation is an increase in the *Number of Rules* that is not caused by multiple rules describing the same relationships (which would be the result if the hypotheses balancing cost would be adapted instead).

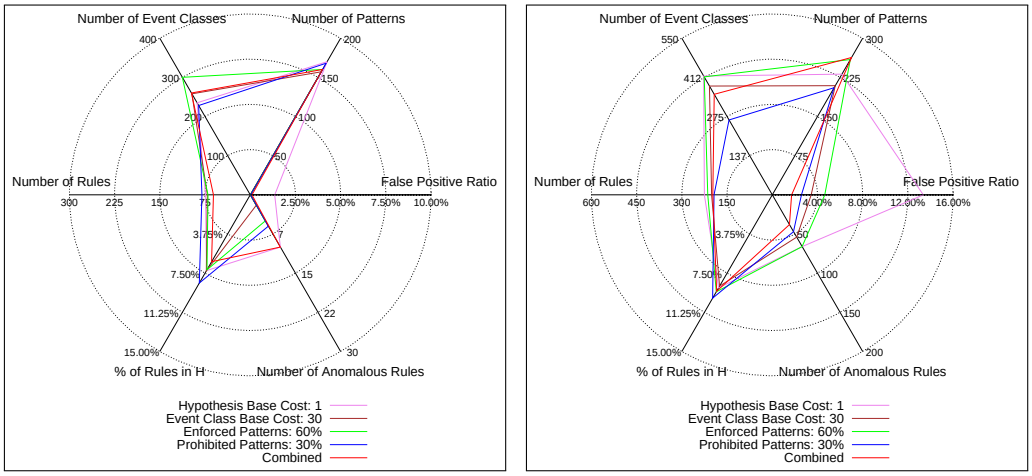
iii The percentage of enforced patterns is increased from 50% to 60%. The expectation is that more specific event classes are generated and that this results in a higher*Number of Event Classes*. Additionally we expect a lower *False Positive Rate* as well as a lower *Number of Anomalous Rules*.

iv The percentage of the prohibited patterns is decreased from 40% to 30%. The expectation is that a lower *Number of Anomalous Rules*, as well as a lower *False Positive Rate*, is generated. Additionally we expect that the *Number of Event Classes* does not decrease significantly.

Figure 6.8 shows the evaluated parameters, that fulfil most of the expectations we had from the combined configuration file. Statistical outliers, in configurations where only one parameter was adapted, can be removed by the combination of different changes. Figure 6.9 compares the combined configuration to the base configuration. The trend towards a lower *Number of Anomalous Rules* can be shown as well as the trend towards a lower *False Positive Rate*. Additionally the system model does not lose any quality. Further evaluations will now use this combined setting.

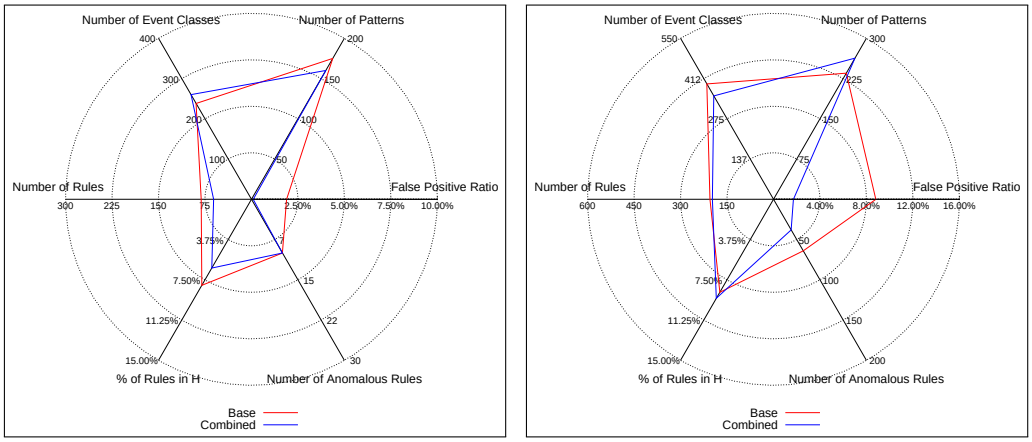
6.2 Event Class Evaluation

The evaluations in the previous section resulted in a near-optimal configuration for the approach. In order to evaluate the generated system model by qualitative means, this section gives a detailed look on the event classes that are generated on the *4U12h* dataset. Two different evaluations are performed: Section 6.2.1 evaluates the coverage of the lines in the input set by event



(a) 2U8h (b) 4U12h

Figure 6.8: Comparison of the combined configuration with the different single parameter adaptions.



(a) 2U8h (b) 4U12h

Figure 6.9: Comparison of the combined configuration with the base configuration.

classes. The result will show that the generated event classes build a meaningful system model. Section 6.2.2 on the other hand performs a cluster based evaluation of the event classes. The results of this evaluation will be used later to evaluate the quality of rules.

6.2.1 Line Based Event Class Evaluation

The anomaly detection approach uses event classes to classify incoming log-events. If a line can be classified by an event class, an event is triggered that can further be analysed by the set

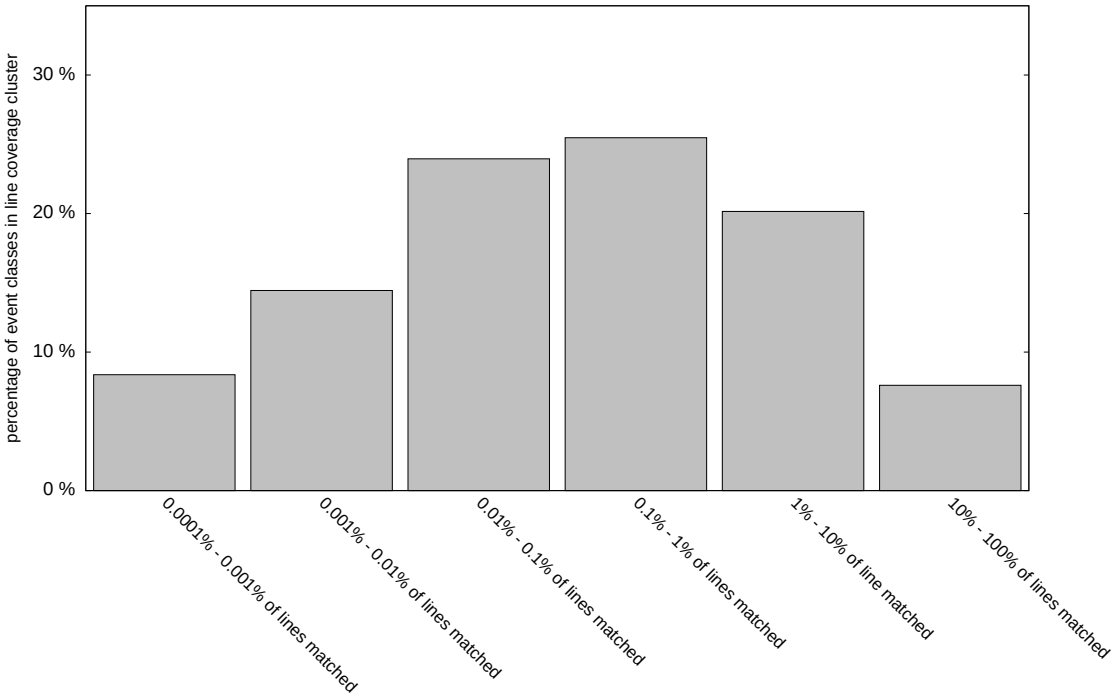


Figure 6.10: Cumulative distribution of event classes clustered by the percentage of lines in the dataset, that are classified by the event class.

of rules in $\mathbb{R}$. Following this approach, event classes have to categorise a subset of all possible incoming log-lines. It is important that the classification of a log-event by an event class C has a certain entropy (see Eq. 3.31). An event class that matches only one line in the whole processed dataset is as useless as an event class that matches all log-lines in the respective dataset.

Figure 6.10 shows a distribution of event classes, distinguished by the percentage of all lines in the processed input file that are classified by the respective event class. Each event class belongs to one of these line coverage clusters. This also means that there exists no event class in $\mathbb{C}$ that does not classify any line. The number of lines that an average event class in one cluster classifies, grows exponentially between two adjacent coverage clusters.

About 40% of the event classes in the system model can be considered highly specific; on average they classify not more than every thousandth line. About 15% of the event classes are even more specific and classify only every ten-thousandth log-line on average. 25% of the event classes can still be considered specific, because they match not more than 1% of all log-lines in the analysed dataset. Only about 8% of event classes have to be considered very generic. They match more than 10% of all lines in the dataset and could be considered trivial (but not useless as discussed in Sect. 6.4).

General statistics about log-line coverage by event classes are given in Tab. 6.3. These values show that the algorithm manages to generate a model that covers the log-input completely. This is not trivial, as aging might cause event classes, that solely classify a rarely occurring type of log-lines, to be deleted. On average, each log-line is covered by multiple event classes. Be-

Number of log-lines that are not classified by any event class:	0
Average number of event classes that classify one line:	12.79
Median number of event classes that classify one line:	13
Maximum number of event classes that classify one line:	17

Table 6.3: Log-line coverage statistics.

cause the average and the median are close to the maximum it is indisputable that the prototype produces a sufficient event class coverage of the log-line in the datasets.

6.2.2 Cluster Based Event Class Evaluation

Event classes describe a class of lines based on search patterns. An event class mask $\vec{C_m}$ and an event class value $\vec{C_v}$ describe each event class and define a set of enforced, and a set of prohibited patterns (see Ch. 3 for more details). It is possible that different event classes describe a very similar set of lines. This is not problematic for the system functionality, but an evaluation of the quality of $\mathbb{C}$ has to evaluate the degree of similarity between different event classes. One way to define the degree of similarity is to generate a dependency tree that describes, if the set of lines classified by one event class C_1 is a subset of the set of lines classified by another event class C_2. In that case C_1 would be a subclass of C_2.

The generation of a dependency tree based solely on the information of the event classes is not possible, because of the differentiation of enforced and prohibited patterns. A pattern P might not be prohibited in C_1 but it is prohibited in C_2 while C_1 is otherwise a subclass of C_2. P would prohibit the relation although, it might have no effect on the set of lines that are classified by C_1, given the context of the monitored system. This is the fact if P is not related to the set of lines described by C_1. It is therefore not possible to generate a valid dependency tree, solely based on the information encoded in the event classes because this information lacks knowledge about the system context. Instead slct[2] – a command line tool that derives clusters of log-lines from a given log-file – is used, to cluster the log-lines in the *4U12h* dataset. Based on these clusters, a tree can easily be generated that shows sub-cluster relationships. If it is further possible to relate event classes to clusters, the cluster tree will substitute the event class tree.

Figure 6.11 shows a limited view of the tree that is generated from the clusters of the *4U12h* dataset. The clusters are generated based on words[3]. Each '*' in the description is a place holder for one word. A '*' that occurs at the end of the descriptor can also substitute multiple words.

Event classes and clusters are not equivalent. Further, there is no distinct mapping between one event class and exactly one cluster but there are two schemas that can be used to generate an n-to-m mapping between event classes and clusters:

Descriptor Based: In this case, a fingerprint is generated for the descriptor of the cluster. Each event class that classifies the cluster descriptor is considered to describe the cluster. This

[2]The *Simple Log Clustering Tool* is a command line base tool that analyses log-files in order to derive clusters of log-lines. See http://ristov.users.sourceforge.net/slct/ for more details.

[3]A word in this case is a substring that is delimited from the rest of the string by space characters or special punctuation marks

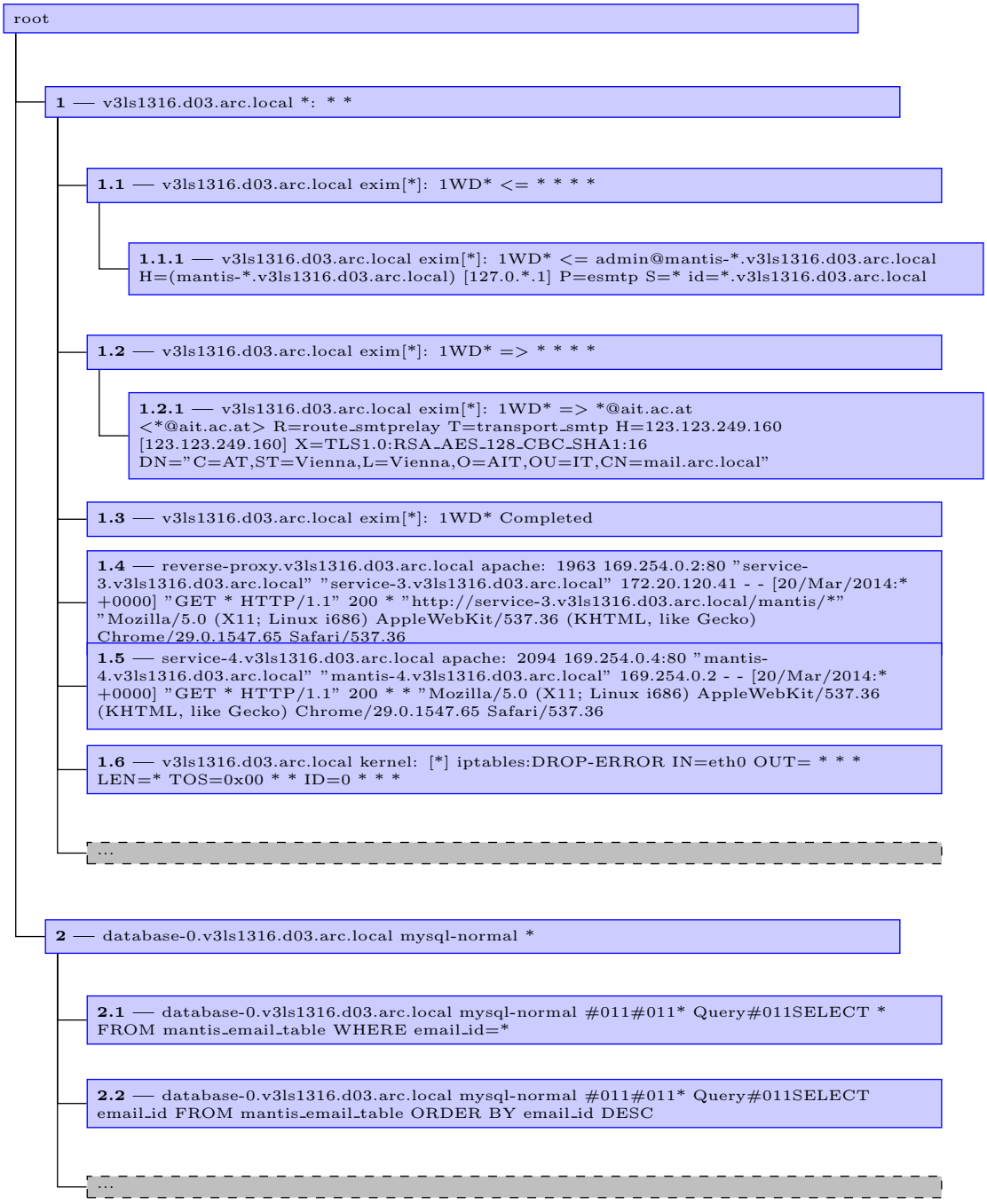


Figure 6.11: Sample of the cluster tree spanned by the clusters of the *4U12h* dataset.

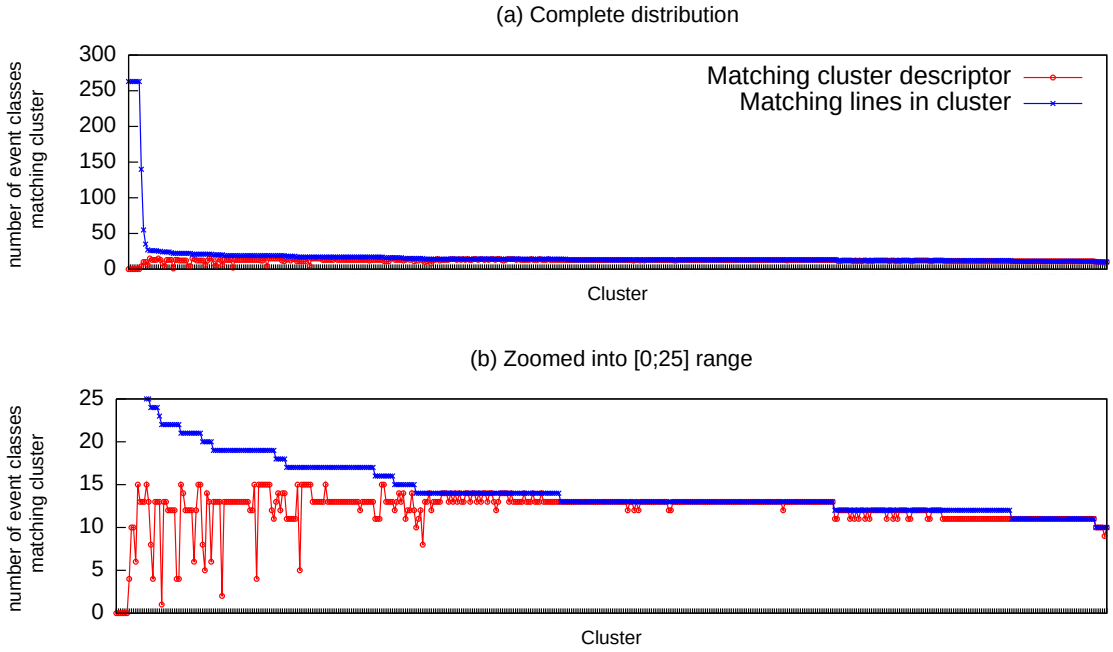


Figure 6.12: Number of event classes that are considered to be related to the generated clusters based on the two metrics.

approach has various limitations. The place holders cannot be expanded to all possible words. It cannot be proven that an enforced pattern cannot be found in every possible word for a place holder with respect to the dataset. At the same time a prohibited pattern could be found in the possible words that would forbid a relation. This approach can be too restrictive.

Line Based: For every cluster, we extract the set of lines from the dataset that are matched by the cluster. If an event class can match at least one line in the set of matched lines a relation between the cluster and the event class is assumed. This schema is more generic than the *Descriptor Based* approach.

Figure 6.12 shows, how many event classes relate to each cluster that was generated from the *4U12h* dataset. It shows that well over 90% of the clusters match less than 25 event classes. The assumption that the *Descriptor Based* schema is more restrictive can be proven. The assumption holds especially for very generic event classes. While the descriptor cannot match all for these event classes, the *Line Based* metric returns relations with well over 200 clusters.

Figure 6.13 shows for every event class in $\mathbb{C}$, how many clusters are related to it. The differences between the two schemas are much more prominent than in Fig. 6.12. While the restrictive *Descriptor Based* schema reults in multiple event classes without relating cluster, the use of the *Line Based* approach gives a reasonable distribution of relations. About half of all event classes are related to less than 10 clusters. About 80% of the event classes are related to less than 50 clusters.

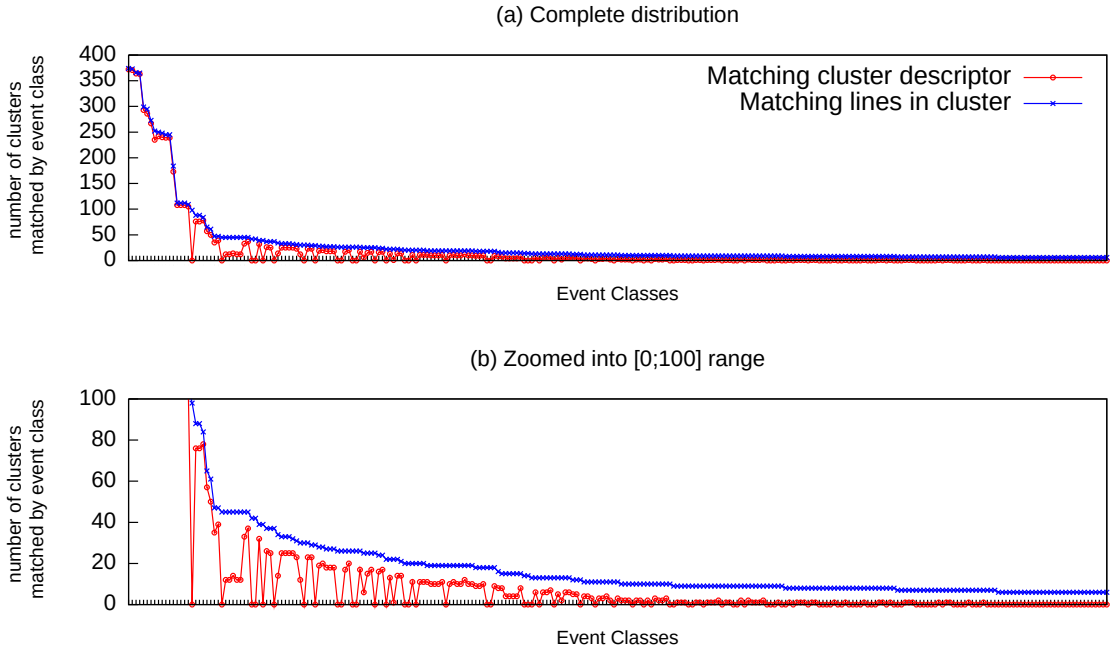


Figure 6.13: Number of clusters related to event classes in $\mathbb{C}$ based on two schemas.

The generated relations are not optimal because event classes and clusters are not intrinsically comparable. But in order to generate a hierarchy between event classes and to analyse the meaning of different rules on the other hand, the generated mappings are sufficient. The evaluations of rules in Sect. 6.3 are performed on the cluster tree that is generated from the 4U12h dataset. The evaluation process extends every cluster in the cluster tree with related event classes. It uses the *Line Based* schema to detect relations.

6.3 Rule Evaluation

Rules built the top level of the system model. In contrast to search patterns and event classes, which describe the meaning of single log-events, rules describe the relations between log-events of different types. Rules therefore model the normal behaviour of the system; they add a time component to the system model. A stable model has to be able to detect timely relations between log-events from different systems in the monitored network, as well as relations between events in one system. This section evaluates the set of rules generated when processing the *4U12h* dataset with the combined parameter setting.

Section 5.1.1 describes that about 90% of the lines in the 4U12h dataset are produced by the database. The reasons for this are described in the referenced section and we will not go into more detail at this point but this distribution is important for the set of generated rules. Section 6.2 showed that the balancing algorithms in place are sufficient to achieve an even distribution of event classes over the processed log-lines. This section evaluates the question, if the

same balancing algorithms are also sufficient to generate an even distribution of rules over the components in the monitored network.

As seen in Sect. 6.2.2, when spanning the cluster tree, the dataset distinguishes two main parts:

i Log-lines that are produced by the database.

ii Log lines that are not produced by the database.

Category (ii) can further be divided into several sub-trees describing the log-lines generated by different systems in the network (e.g., the mail server). This section uses the cluster tree generated in Sect. 6.2.2, to classify rules by the system events that are connected by one rule. Therefore, every event class is assigned to exactly one cluster. Since there is no unique mapping between event classes and clusters, the most specific parent cluster, that is common to all clusters matched by the event class, is selected.

Using this mapping between event classes and clusters, each rule can now be seen to connect exactly two clusters; it can be described by a path in the cluster tree. This description is used to evaluate the quality of rules. The turnover-node is the most generic node (i.e., the node with the lowest distance to the root node) in the path described by a rule. It has a special meaning, because it can be used to identify the systems that are connected by the rule. If the turnover node of a rule is the root node, the rule connects any non-database service with the database or vice-versa. Using this schema Fig. 6.14 shows the distribution of rules by their turnover nodes. The first thing to note is that only three turnover nodes are distinguished. This is mainly caused by the line-based schema applied when mapping event classes to clusters. As shown in Fig. 6.13 every event class matches at least 5 clusters. The utilisation of the described method to generate a mapping from one event class to exactly one cluster, results in event classes that get mapped to very generic clusters. The figure shows that an even distribution of rules is achieved. As expected, the biggest number of rules connects components with the database or vice-versa. As expected, the balancing algorithms are able to keep the number of rules, that connect database internal log-events to a reasonable number so they do not flood the system model. The relatively high number of rules that connect different non-database services, can be explained by the less uniform structure of log-lines produced by different services, compared to the structure of database internal log-lines.

With the end of this section the evaluation of the system model is completed. We showed that the algorithms of the approach are sufficient to generate an extensive and balanced system model. We further proved hypothesis (i) from Sect. 1.2. The evaluations of the anomaly detection capabilities in Sect. 6.4 and Sect. 6.5 are based on this system model.

6.4 Anomaly Detection in common ICT networks

This section covers the evaluation of the anomaly detection capabilities of the prototype implementation, based on the anomalous dataset described in Sect. 5.1.3. After a short introduction into the used metrics, the evaluation will be split into two parts; one for each type of anomaly that was injected.

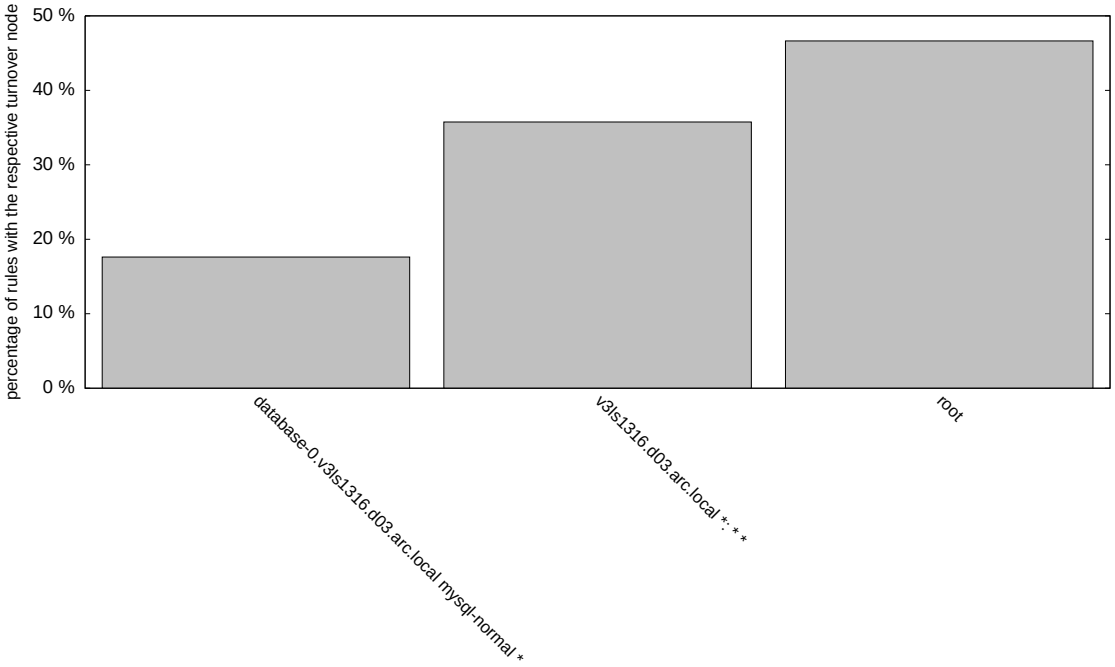


Figure 6.14: Distribution of rules classified by turnover nodes.

6.4.1 Metrics

False Positive Rate. The false positive rate describes the ratio between the number of events that were considered positives (in our case the number of detected anomalies) out of the number of events that should have been negatives (in our case events where no anomaly is expected). The denominator of Equation 6.1 is therefore the sum of false positives and true negatives (i.e., the number of events correctly considered normal).

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{6.1}$$

In order to calculate this rate a definition of false positives and true negatives regarding the system under evaluation is required. We define them as follows:

False Positive: Every evaluation of a rule that either: (i) results in an anomaly being detected by the rule, or (ii) extends the anomalous state of the rule, if the rule should not be anomalous, is considered a false positive.

True Negative: Every evaluation of a rule that does not fulfil the conditions of a false positive, given that the system is in a normal state at the time of the evaluation, is considered a true negative.

A definition of false positives as the number of anomalies that were detected incorrectly would seem better suited at first sight. But when we use this definition, the calculation of a false positive

rate is not possible any more, due to the lack of an equivalent definition for true negatives. Therefore calculation of the false positive rate needs to be based on single evaluations of rules.

For the proposed definitions, it is not possible to calculate the false positive rate in an anomalous slot. Instead for every anomalous time slot that is evaluated we calculate the false positive rate in the same time slot but in the clean *4U12h* dataset. Since complexity, number of stimulating virtual users and recording time are equal, the results are comparable. Although the actions of the virtual users are highly random, the evaluation does not rely on single actions of certain users but on the random noise that is generated by their interactions.

True Positive Rate. The true positive rate is the ratio between the number of events that were correctly classified as positives and the number of events that should have been classified as positives. Equation 6.2 shows the ratio. The denominator is the sum of the already described true positives and the false negatives (i.e., events that should have been considered anomalous but were not).

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{6.2}$$

Regarding the proposed system under evaluation true positives and false negatives are defined as follows:

True Positive: Given a time range Δt in which a rule should detect an anomaly, all evaluations of the rule in that time range are considered true positives, if there is at least one evaluation that either: (i) results in an anomaly being detected by the rule, or (ii) extends the anomalous state of the rule, and the cause of this state is related to the anomaly in the monitored system.

False Negative: Given the same time range Δt in which a rule should detect an anomaly, all evaluations of the rule are considered false negatives, if they cannot be considered true positives by its definition.

6.4.2 Illegal Database Dump

The first anomaly type that gets injected, is an illegal dump of all databases on the database server in the network. This evaluation considers four different time slots of 30 minutes. Each timeslot contains one more injected anomaly than the previous one (see also Sect. 5.1.3).

In order to calculate the TPR for every anomalous slot, we identify a set of rules which should trigger anomalies due to the injections. Two examples of these rules are analysed below.

Rule 849. The following rule describes the relation between a `SHOW TABLES` database command and a database connect request. The time frame in which this implication has to hold is -10 seconds. This means that at most 10 seconds before a `SHOW TABLES` command is logged, a connection to the database has to has happened. Table 6.4 shows the enforced and prohibited patterns for the condition as well as for the implied event. The last lines matched by these events are the following:

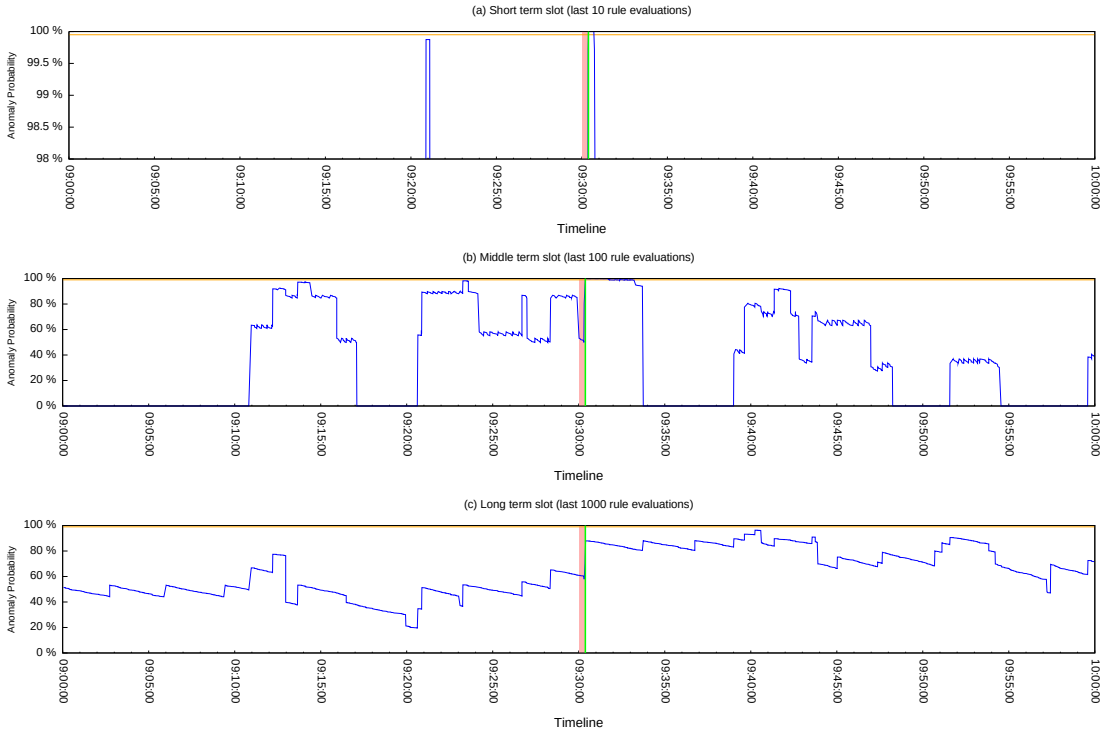


Figure 6.15: Overview of the slot stabilities in rule 849 during the injection slot A1.1.

Condition Event: `database-0.v3ls1316.d03.arc.localmysql-normal#011#01133179Query#011SHOW TABLES`

Implied Event: `database-0.v3ls1316.d03.arc.localmysql-normal#011#01133179Connect#011mantis_user_4@service-4.v3ls1316.d03.arc.localon`

This rule has to be considered very relevant for detecting the injected anomalies. During a database dump, the `SHOW TABLES` command is executed multiple times. In contrast to the regular system behaviour, all of these commands are only preceded by a single connect statement. The Mantis instances on the other hand, reconnect for every new user request. Thus every database dump should trigger an anomaly in rule 849.

Figure 6.15 shows the stability development of the rule's slots, before and during injection A1.1. The orange line in the smallest slot indicates the anomaly threshold. Only if the certainty for an anomaly is above this threshold, an anomaly gets triggered. The red area marks the duration of the injection; the time stamp, at which an anomaly is detected (exactly two seconds after the injection finished), is marked by the green line.

Figure 6.15 shows that there were some false evaluations prior to the injection, but they were not significant enough to be considered an anomaly. The anomaly probability only exceeds the threshold right after the injection is finished and an anomaly is detected. The anomaly is only

Rule 849: -10sec			
Condition Event Class		**Implied Event Class**	
Enforced	*Prohibited*	*Enforced*	*Prohibited*
normal #011#011	ocal rsys	rvi	gin_t
Query#01	0204	arc.local mysql-normal	4.0
arc.local mysql-normal	9c:25:67:0	database-0.v3ls1316.	tus_
y#011SH	4.0	v3ls1316.d03.arc.local	setta
database-0.v3ls1316.	tus_		[10/
v3ls1316.d03.arc.local	ROM ma		ff:ff:ff:ff
	ntis_p		t-minimal
	setta		mber n
	im[		b77
	rvi		ng//dat
	tis-4.		ef49e
	exim[		l exim
	t-minimal		like Gecko
	b77		ix monitori
	ng//dat		f:ff:
	] [client		b3385
	f:ff:		l: [120
	al) [127		0 Complet
	ind /va		ON b
	0001jB-6p C		n/projax/so
	im[66		6J
	n/projax/so		
	.local k		
	k0-Ow Compl		
	run-part		
	106		
	nit DB#0		
	BY b.id		
	]:		
	OR		
	Comp		
	1WY4Ud-000		
	apache: 210		
	_3@s		

Table 6.4: Detailed description of *Rule 849*.

detected by the short-term slot. It is not significant enough to trigger an anomaly in the middle- or long-term slot, but the effects on the anomaly probability are clearly visible.

Rule 286. The following rule describes the fact that a database event is always followed by a firewall entry within 10 seconds. This rule is not optimal since it does not show a relation enforced by the system behaviour. Instead the described relationship is caused by the regular input from various users. It is not enforced by the system that a firewall log-event follows a database request. But it is normal in the monitored system because virtual users act in bursts of actions. It is rarely the case that a user only sends one single request. Instead multiple consecutive requests can be expected. There are also time ranges where no users interact with the system. The behaviour that the database server answers requests while, in the same time range, no user acts is considered abnormal. But exactly this system behaviour is triggered by the injected database dump if it occurs in a time window of low user activity. Table 6.5 describes the rule in more detail. Additionally the last lines matched by the condition event class and the implied event class were as follows:

Condition Event: `database-0.v3ls1316.d03.arc.localmysql-normal#011#01133179Query#011UPDATE mantis_user_table`

Implied Event: `v3ls1316.d03.arc.localkernel: [165361.740449] iptables:ACCEPT-INFO IN=lo OUT= MAC=00:00:00:00:00:00:00:00:00:00:08:00 SRC=169.254.0.4 DST=169.254.1.0 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=19898 DF PROTO=TCP SPT=52372 DPT=3306 SEQ=1271940985 ACK=0 WINDOW=43690 RES=0x00 SYN URGP=0 OPT (0204FFD70402080A02760AC30000000001030306)`

Figure 6.16 shows the stability development of the rule's slots, before and during injection slot A1.1.

As expected the rule possesses limited stability. We can see that various failed evaluations prior to the injection set the middle-term slot into a somewhat anomalous stage, but the evaluations are not considered anomalous enough to be an anomaly. The anomaly probability in the short-term slot only exceeds the threshold at 9:30:25 (exactly two seconds after the injection is finished) and triggers an anomaly.

Overall Performance

After we decided on the set of rules that are considered relevant for detecting the injected anomalies, the FPR and the TPR for each anomalous time slot in the dataset can be calculated. Figure 6.17 shows the results in a scatter diagram. The metrics were calculated with the combined configuration and with the base configuration, to evaluate the effects that the performed parameter changes have on the results.

The injected anomalies are detected in every timeslot, but the detection rates of different rules are not very constant. Only about 40% of the rules in the identified rule set detected the anomaly timely with the combined configuration. But the approach did not perform constantly

Rule 286: +10sec			
Condition Event Class		**Implied Event Class**	
Enforced	*Prohibited*	*Enforced*	*Prohibited*
normal #011#011	n/projax/so	=0x00 TT	ocal rsys
Query#01	ocal rsys	=TCP SPT	2226 ACK
arc.local mysql-normal	ernel: [119	URGP=0	database-0.v3ls1316.
database-0.v3ls1316.	W=43690 RE	v3ls1316.d03.arc.local	y#011SH
v3ls1316.d03.arc.local	6 Query#	0204	9c:25:67:0
	3a:		arc.local mysql-normal
	62c5		setta
	[10/		8 +0000]
	rvi		rvi
	exim[		_smtp
	WY2hl-0		exim[
	it#01		it#01
	t-minimal		normal #011#011
	b77		cal exim[5
	ng//dat		t-minimal
	ef49e		b77
	l exim		al) [127
	] [client		ind /va
	ix monitori		0001jB-6p C
	l: [120		bles:D
	ON b		0001kM-7o
			DST=22

Table 6.5: Detailed description of *Rule 286*.

with either configuration. With each configuration it had difficulties to detect the anomalies in one of the anomalous slots. Overall, the base configuration had a higher detection rate (i.e., true positive rate) on average. But it has to be noted that the combined configuration was designed to reduce the false positive rate of the system. This tendency can be seen in the results and it is one reason for the lower true positive rate. The results in detecting the injected anomalies are not optimal but show that every anomaly was detected by at least one of the expected rules.

6.4.3 Database Logging Disabled

The second type of anomaly that is injected in the anomalous dataset is a time period where the logging functionality of the database server gets disabled. Four anomalous time slots get evaluated. The duration of each slot is 30 minutes and the duration of the injection differs. The injection times are: 30 seconds, 1 minute, 2 minutes and 4 minutes. As in the previous case, a set of rules is selected which are expected to detect the injected anomaly. Two examples are

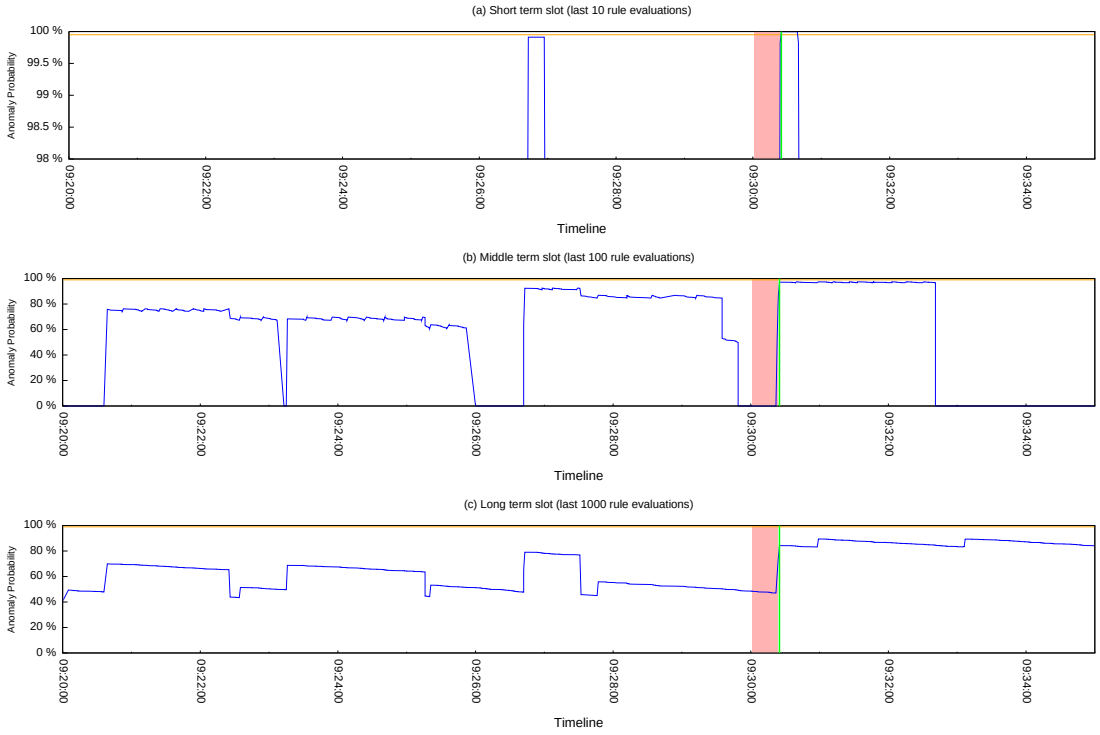


Figure 6.16: Overview of the slot stabilities in rule 286 during the injection slot A1.1.

given below.

Rule 708. This rule describes the fact that a firewall log-event is followed by a database connect event. This is a highly relevant rule that describes a very stable system behaviour: Every user request is handled by the firewall before it is passed on to the web-server. The web-server on the other hand queries the requested data from the database. If something turns off the logging facilities on the database server, an anomaly should be immediately detected by this rule. Table 6.6 gives more details about the rule while the last classified lines by the condition event class and the implied event class are as follows:

Condition Event: `v3ls1316.d03.arc.local kernel: [165361.682603] iptables:ACCEPT-INFO IN=eth0 OUT= MAC=00:50:56:9c:25:67:02:01:f4:01:00:30:08:00 SRC=172.20.120.49 DST=169.254.0.2 LEN=60 TOS=0x00 PREC=0x20 TTL=59 ID=26264 DF PROTO=TCP SPT=57999 DPT=80 SEQ=1135473877 ACK=0 WINDOW=14600 RES=0x00 SYN URGP=0 OPT (020405B40402080A012C924D0000000001030306)`

Implied Event: `database-0.v3ls1316.d03.arc.localmysql-normal #011#01133179 Connect#011mantis_user_4@service-4.v3ls1316.d03.arc.local on`

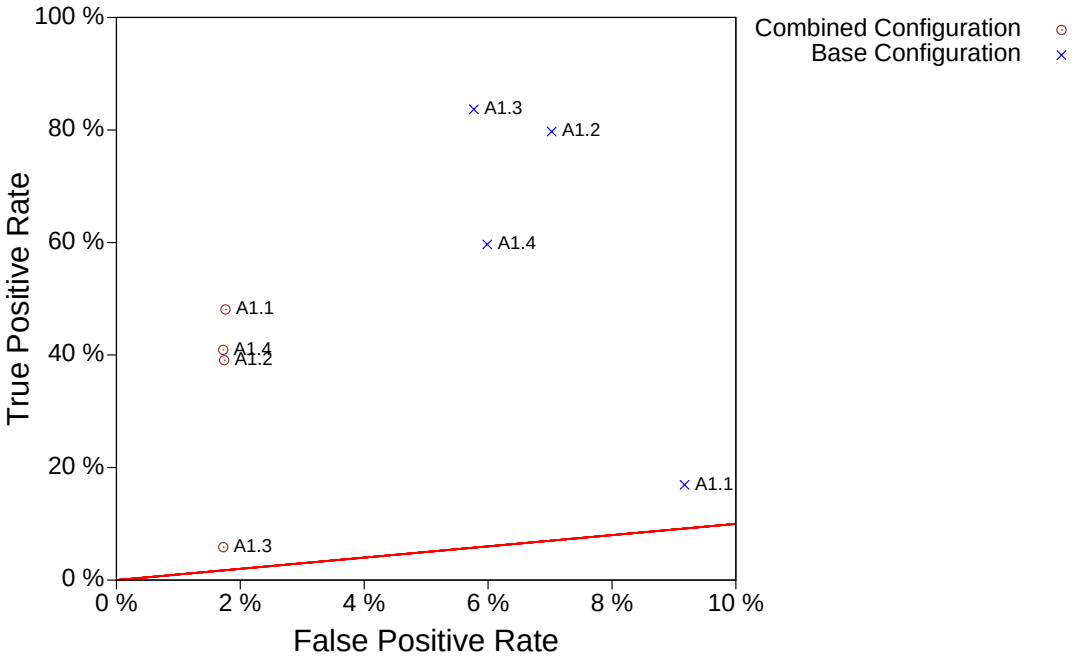


Figure 6.17: ROC scatter for the time slots containing injected database dumps calculated with the base configuration as well as the combined configuration.

Figure 6.18 shows the stability development of the rule's slots, before and during the injection of anomalies of the second type.

The rule and its slots behave exactly as expected. The time before the injection starts is completely regular. Only the long-term slot measured some disturbances which seem to be late effects of the anomalies injected in slot A1.4. But the rule is on a good way to a completely stable status. Exactly when the first anomaly starts, all three slots exceed the anomaly threshold. Already the first anomaly is significant enough for the long-term slot to turn anomalous. It further cannot get stable again until the next injection hits. The anomaly probabilities in the short-term and in the middle-term slots show, how the size of the different slots affect their sensibility towards anomalies. Since the long-term slot does not get into a normal state any more, no new anomalies can be triggered for the second, third and fourth injection. Anyway, the effects in the short- and middle-term slots confirm that every single injection would have been discovered by the rule.

Rule 804. This rule describes that every web-server request (it does not matter if it is recorded by the firewall or by the web-server directly) is followed by a database query. The motivation to put this rule into the rule set is similar to the motivation for rule 708. Table 6.7 shows detailed information about the rule and the last lines classified by the condition event class and the implied event class are as follows:

Condition Event: `v3ls1316.d03.arc.local kernel: [165361.682603] ip`

Rule 708: +1sec			
Condition Event Class		**Implied Event Class**	
Enforced	*Prohibited*	*Enforced*	*Prohibited*
T=1	8014]:	rvi	gin_t
=TCP SPT	ernel: [119	arc.local mysql-normal	4.0
9c:25:67:0	2226 ACK	database-0.v3ls1316.	tus_
v3ls1316.d03.arc.local	database-0.v3ls1316.	v3ls1316.d03.arc.local	setta
INFO	y#011SH		[10/
	antis/view		ff:ff:ff:ff
	ROM ma		t-minimal
	62c5		mber n
	Query#01		b77
	ntis_p		ng//dat
	.30		ef49e
	8 +0000]		l exim
	ff:ff:ff:ff		like Gecko
	WY2hl-0		ix monitori
	normal #011#011		f:ff:
	ocal chec		b3385
	like Gecko		l: [120
	] [client		0 Complet
	l: [120		ON b
	0001jB-6p C		n/projax/so
	im[66		6J
	bles:D		
	ON b		
	run-part		
	106		
	WHER		
	.v3ls131		
	0111571		
	]:		
	OR		
	iptables:D		
	(r		

Table 6.6: Detailed description of *Rule 708*.

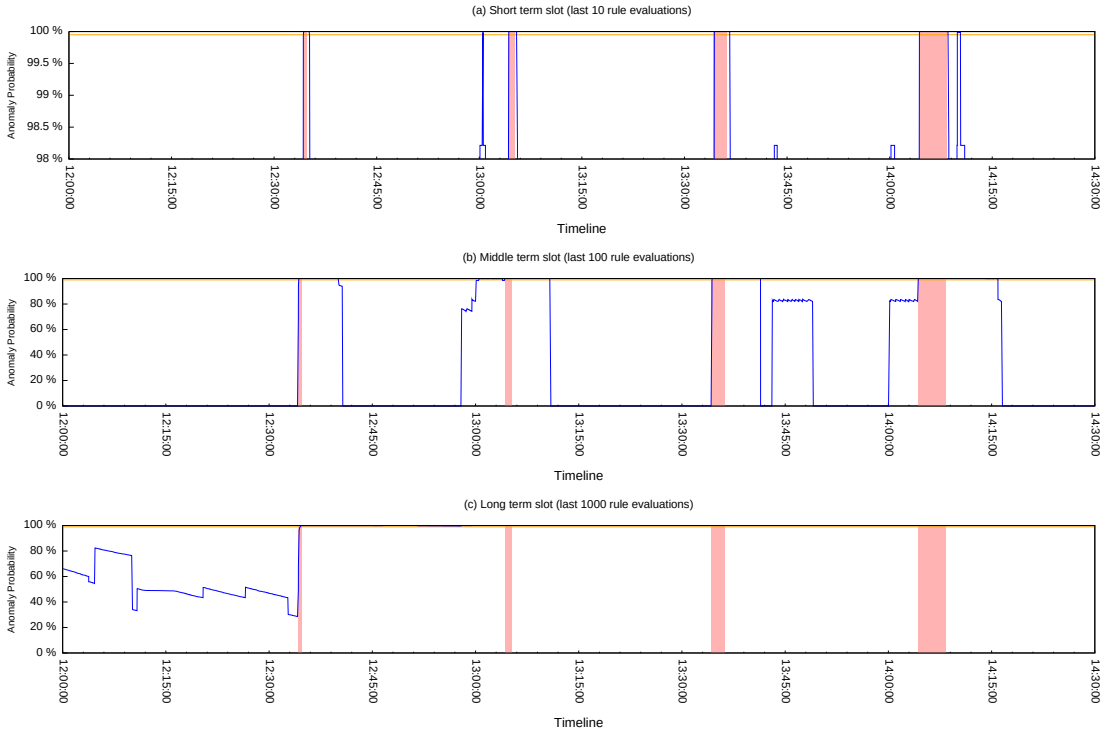


Figure 6.18: Overview of the slot stabilities in rule 708 during all four injection slots of the second anomaly type.

```
tables:ACCEPT-INFO IN=eth0 OUT= MAC=00:50:56:9c:25:67:02:01
:f4:01:00:30:08:00 SRC=172.20.120.49 DST=169.254.0.2 LEN=60
TOS=0x00 PREC=0x20 TTL=59 ID=26264 DF PROTO=TCP SPT=57999 D
PT=80 SEQ=1135473877 ACK=0 WINDOW=14600 RES=0x00 SYN URGP=0
OPT (020405B40402080A012C924D0000000001030306)
```

Implied Event: `database-0.v3ls1316.d03.arc.local mysql-normal #011 #01133179 Query#011UPDATE mantis_user_table`

Figure 6.19 shows the stability development of the rule's slots, before and during the injections of anomalies of the second type.

This rule proves even more stable than rule 708. Even the negative evaluations that where considered a false positive in rule 708 at 13:00:01 are not significant enough and the short-term slot probability does not exceed the threshold. The other results are similar to the results of rule 708. A stable detection rate can be derived from the collected data.

Overall Performance

After the selection of a set of rules that has to detect the injected anomaly, we calculate the false positive rate and the true positive rate of the evaluated time slots. Figure 6.20 shows the familiar

Rule 804: +1sec			
Condition Event Class		**Implied Event Class**	
Enforced	*Prohibited*	*Enforced*	*Prohibited*
T=1	0:1	Query#01	0204
.local k	W=43690 RE	arc.local mysql-normal	antis/view
URGP=0	gin_t	database-0.v3ls1316.	4.0
v3ls1316.d03.arc.local	antis/view	v3ls1316.d03.arc.local	tus_
INFO	tus_		ntis_p
	arc.local mysql-normal		.0.4.
	ROM ma		bug_page.p
	ntis_p		=0x00 TT
	=0x00 TT		_smtp
	8 +0000]		ff:ff:ff:ff
	_smtp		tis-4.
	cal exim[5		cal exim[5
	mber n		mber n
	ocal chec		b77
	ng//dat		ng//dat
	0 Complet		ef49e
	n/projax/so		0001jB-6p C
	6J		6J
	k0-Ow Compl		k0-Ow Compl
	tis-3.		tis-3.
	0001kM-7o		
	xim[		
	AND		
	106		
	nit DB#0		

Table 6.7: Detailed description of *Rule 804*.

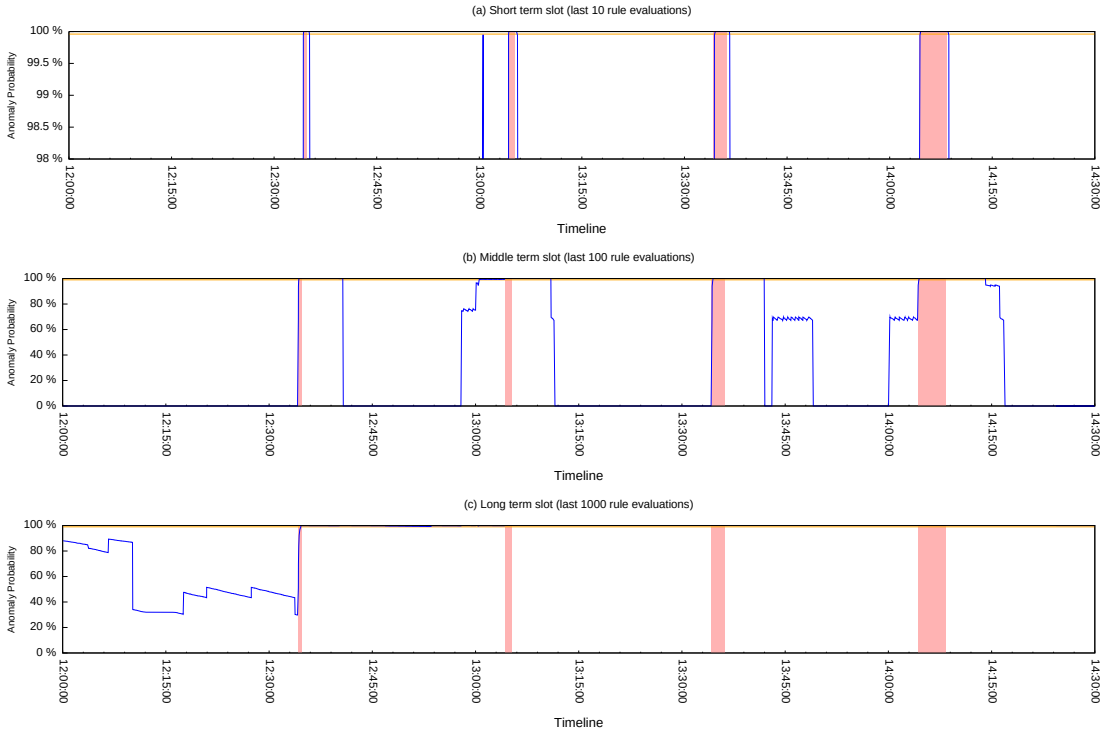


Figure 6.19: Overview of the slot stabilities in rule 804 during all injections of anomalies of the second anomaly type.

ROC scatter. It shows that the prototype implementation performs much better on the second anomaly type. There is a stable true positive rate above 80% with a reasonable false positive rate of around 3% using the combined dataset. Not all rules in $\mathbb{R}$ that we identified as rules that should detect the anomalies were constantly able to do so, but about 80% were. In the analysed slots the combined configuration also shows more stable results than the base configuration. The trend towards the lower false positive rate is again supported by the extracted data.

6.4.4 Conclusion

The detection capability of the prototype implementation regarding two types of anomalies was analysed in different time slots with different injection rates. The prototype detected all inserted anomalies but injected database dumps were not detected consequently by a single rule. Given the defined set of rules, every injected dump is detected by about 40% of the rules in the set. This guarantees detection of the injected anomalies, but a relatively high false positive rate of 3% clouds the results.

Tampering with logging capabilities on the other hand, was detected at a very stable rate. It was possible to prove the detection of every injected anomaly by one single rule. Also the results of the ROC metric looked very promising.

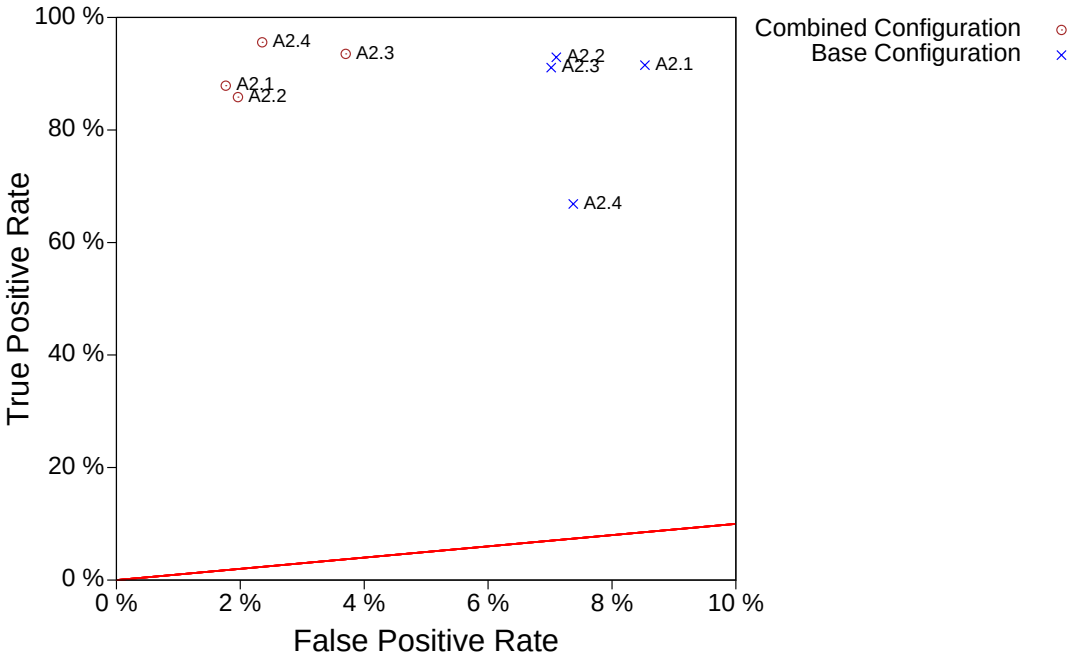


Figure 6.20: ROC scatter for the time slots containing time slots with disabled database logging. The values are calculated with the base configuration as well as the combined configuration.

The discussed results support the assumption that the proposed anomaly detection approach can be used to detect the effects of abnormal requests in a system or network. Not all of these abnormal requests have to be caused by an advanced persistent threat but it was shown that typical anomalies can be detected timely, based on a stable system model. The results further support hypothesis (ii) from Sect. 1.2. All injected anomalies where detected while the anomalous behaviour was still ongoing.

6.5 Anomaly Detection in SCADA networks

Section 6.4 showed stable results of the prototype implementation regarding its abilities to detect different types of anomalies in common ICT networks. But it also showed the limitations of the proposed approach regarding the stable detection of anomalies that only manifest in an increased output of otherwise normal log-events.

This section will evaluate the possibilities to apply the proposed approach in the domain of SCADA systems. The events triggered in a SCADA system are highly structured and well defined. Therefore the limitations of the approach carry less weight while its strengths are more prominent (namely syntax-independent applicability without high configuration effort).

This evaluation is based on the SCADA dataset discussed in Sect. 5.2. Due to the limitations of the dataset we first have to adapt the configurations used. The main reason for the changes is that we have to adapt the period T in our configurations. In our ICT datasets a period of 30 000

General Configuration	
Parameter	*Setting*
`cais.logLinesPerPeriod`	1 000
EventClassificationBalancer Configuration: `cais.atomhandler.balancer.EventClassificationBalancer.*`	
Parameter	*Setting*
`baseCost`	1
`triggeredEventCost`	10
CorrelationRuleBalancer Configuration: `cais.atomhandler.balancer.CorrelationRuleBalancer.*`	
Parameter	*Setting*
`baseCost`	1
`existingHypothesisCost`	100
Anomaly Threshold: `cais.IRuleStatisticsCalculator.*`	
Parameter	*Setting*
`AnomalySignificanceLevel`	0.05

Table 6.8: Parameters that are changed for the SCADA evaluation

lines was defined. Now with the SCADA dataset, the whole dataset consists of 30 000 lines. We reduced the period T to 1 000 which corresponds to the same 20 minute interval we chose in the ICT evaluation. Table 6.8 shows the needed parameter changes. Despite the parameters that were defined depending on T we also changed the anomaly probability threshold from previously very restrictive 0.0001 to 0.05. The rules in this very controlled dataset do not have enough evaluations to exceed the original threshold.

6.5.1 Basic Evaluation

Figure 6.21 shows the basic metrics generated from the model that the prototype generated from the anomaly-free SCADA dataset. We see that the size of the model is significantly smaller than the models generated from the more excessive datasets from the ICT network. The trends on the other hand, are very similar to the ones in the ICT models. The figure shows that the system is able to generate a stable system model for the SCADA dataset.

6.5.2 Anomalous Firewall Activity

The first anomalous dataset that gets evaluated includes two injected connections that were accepted by the database. In the same time range (from the time the connections are established until 10 seconds afterwards) no measurement points get sent to the requester. An analysis of the system model results in a set of two rules that should detect the anomaly. Both rules state that every accepted database connection is followed by the transmission of measurement values back to the requester. The detailed rule information and their evaluation results are shown below.

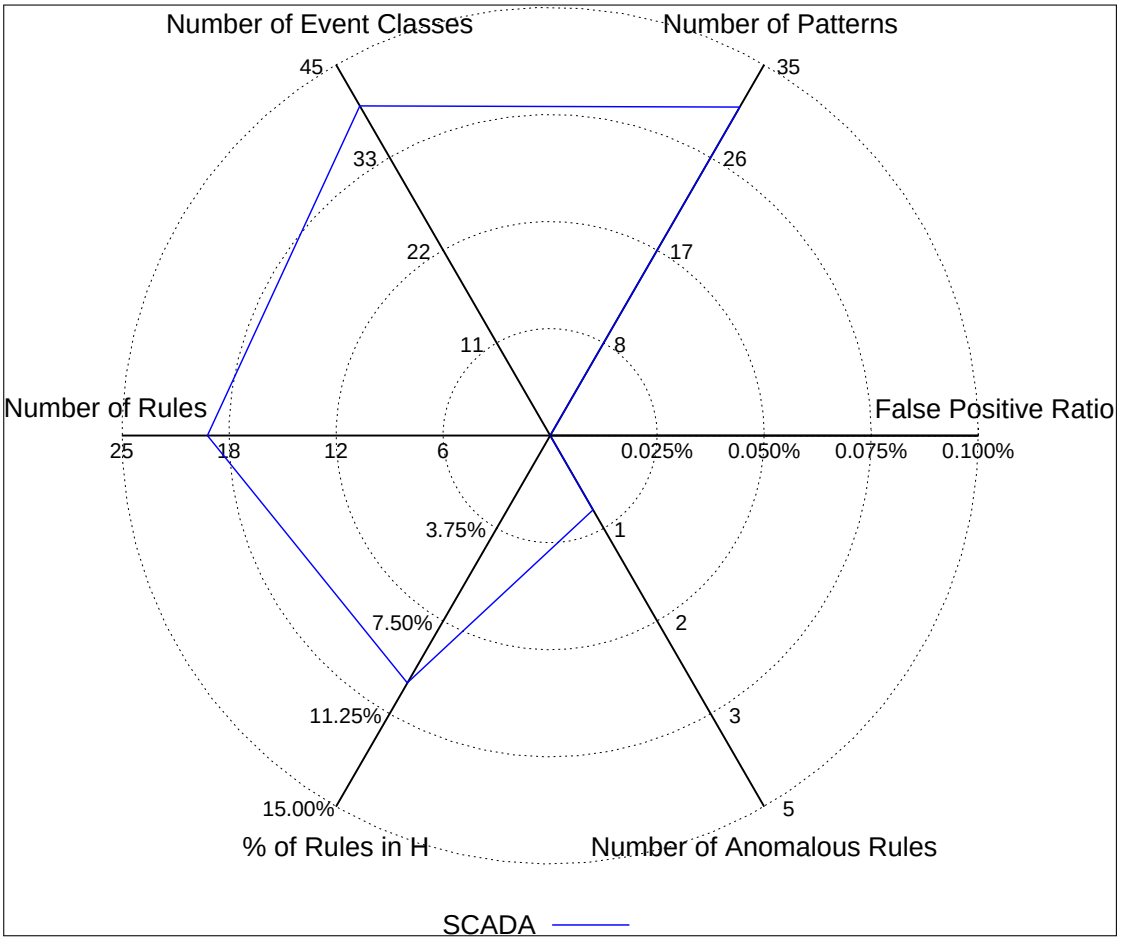


Figure 6.21: General model metrics generated from the anomaly free SCADA dataset.

Rule 163 The last lines[4] classified by the condition event class and the implied event class are given by Lst. 6.1 and Lst. 6.2.

```
1 "eth4.321" "lamfdp33" "Log" "Accept" "ntp-udp" "ntp-udp" "[url-
    removed].at" "[url-removed].at" "udp" "836" "-->␣Netz-PN-Fernwirk␣
    to␣NZ" "345-LAG_ON_PN_CN_Global" "" "service_id:␣ntp-udp" "VPN-1␣
    Power/UTM" "" ""
```

Listing 6.1: Last condition event of rule 163.

```
1 ,76/In /Qelle=4123/Len= 21 Measured float/36    Cause=3()
    Number=1   Common=28/17   floating point Info/Obje=11/122/091
    Val=  130.36 QDS=0x00 Date/Time=06.05.2013/19:59:59,744 - IV= 0
    DST= 1
```

Listing 6.2: Last implied event of rule 163.

[4]Notice that the lines are again obfuscated

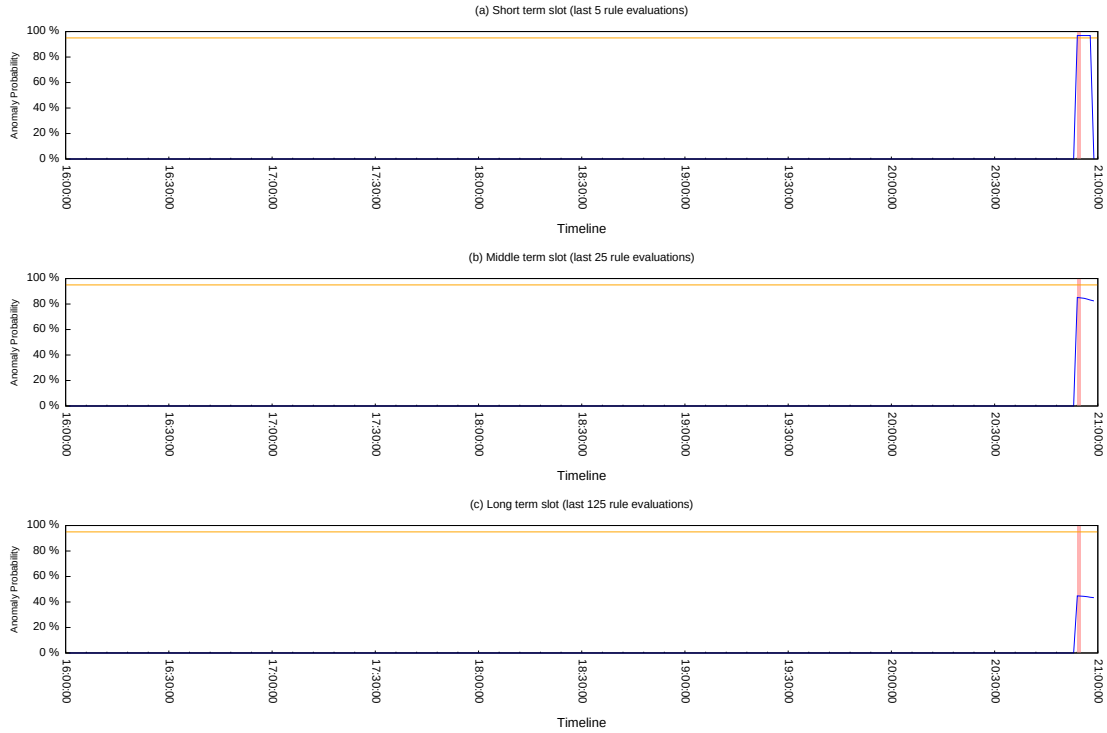


Figure 6.22: Overview of the slot stabilities in rule 163 during the injection of anomalous firewall connections.

Figure 6.22 shows the anomaly probabilities in all slots of the rule during the whole time range in which the rule is stable. The figure shows that the rule is completely stable up to the time of the injection. The anomaly is detected by the rule, but it is not significant enough to set the middle-term slot in an anomalous state.

Rule 169 The last lines classified by the condition event class and the implied event class are given by Lst. 6.3 and Lst. 6.4.

```
1 "eth4.321" "lamfdp33" "Log" "Accept" "ntp-udp" "ntp-udp" "[url-
    removed].at" "[url-removed].at" "udp" "836" "-->␣Netz-PN-Fernwirk␣
    to␣NZ" "345-LAG_ON_PN_CN_Global" "" "service_id:␣ntp-udp" "VPN-1␣
    Power/UTM" "" ""
```

Listing 6.3: Last condition event of rule 169.

```
1 ,76/In /Qelle=4123/Len= 21 Measured float/36    Cause=3()
    Number=1   Common=28/17   floating point Info/Obje=11/123/22
    Val=  130.36 QDS=0x00 Date/Time=06.05.2013/19:59:59,744 - IV= 0
    DST= 1
```

Listing 6.4: Last implied event of rule 169.

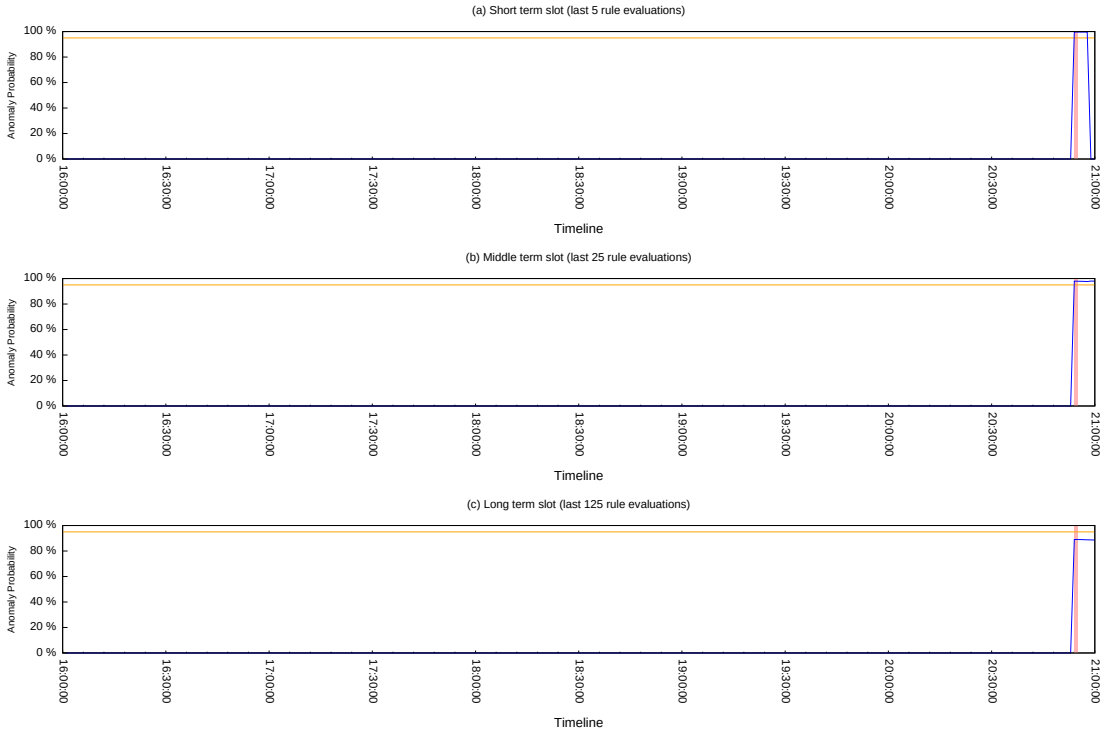


Figure 6.23: Overview of the slot stabilities in rule 169 during the injection of anomalous firewall connections.

Figure 6.23 shows the anomaly probabilities in all slots of the rule during the whole time range in which the rule is stable. Similar to rule 163, the events are considered completely normal up to the injection of the anomalous connections. In contrast to rule 163, here the anomaly is considered much more significant and also the middle-term slot reaches an abnormal state.

Conclusion. The prototype implementation was able to detect all injected anomalies with all rules that were expected to detect the injections. At this point we do not show a ROC scatter diagram. The true positive rate in this dataset is 1 while the false positive rate is nearly 0 (see Fig. 6.21). The figure would not show new information. Instead we can conclude that the suggested approach performs very well in the limited SCADA dataset. A solid prove of hypothesis (iii) in Sect. 1.2 is not given by the results. The limitations of the evaluated dataset prohibit a more detailed investigation. But the conducted experiment rather supports than contradicts the hypothesis.

6.5.3 Invalid Values

The second anomalous dataset that was generated from the SCADA dataset included altered measurement values in a time period of one minute. All transmitted measurement values were 0.0. During analysing the system model it was not possible to identify any rule that is able

to detect the injected anomaly. It is therefore not possible to visualise any results about the anomaly. This example is included to show the limitations of the approach. As discussed in Sect. 3.3.1 the set of patterns describes the basic knowledge the system has about the system. If we compare the situation with the altered result values with Fig. 3.3 the values have to be classified as unique or at least very infrequent substrings. It is highly unlikely (and also not intended by the described approach) to include such patterns in the model. They cannot be used to monitor relations between different system events. The monitoring of correct functionality of one system has to be performed by additional tools that are able to intelligently interpret the measured data.

It seems to be a limitation at first that the prototype cannot detect the injected anomaly. But it would only mean redundant information. The goal of SCADA systems is to monitor and control industrial systems. The monitoring of measurement values is a core functionality of the SCADA application. To detect the changed values as anomaly would be redundant information for an suervisor.

CHAPTER 7

Conclusion

Section 7.1 summarises the main contributions of this thesis. Section 7.2 describes the key findings that were concluded during the course of this work. Finally, Sect. 7.3 gives an outlook on how the proposed approach can be extended and improved in the future.

7.1 Summary

In this thesis a novel anomaly detection approach that utilises log-lines produced by various systems and components in ICT networks has been resented. After reviewing the current state of the art, we concluded that preventive security mechanisms and signature-based detection methods are insufficient to handle novel, targeted and persistent threats. The novel approach was first defined formally. This definition started by positioning the approach in common security settings before we split the actual approach into two parts. First, the definition of the core functionality was given and we stated how different parts of the automatically generated system model are used to detect anomalies. The second part defined how the system model is built. Three types of atoms were defined that build the system model: search patterns, event classes and hypotheses. After the formal definition, we presented a prototype implementation of the approach before we evaluated it with different aspects. To generate testdata for the evaluation in a common ICT network, we proposed and evaluated a semi-synthetic testdata generation approach. Two clean datasets were generated using this generator. We started with an evaluation of the system's parameters before we evaluated the quality of the generated event classes and rules.

Finally, different types of anomalies were injected into different datasets, and the detection capabilities of the prototype were evaluated. We performed the evaluation of the anomaly detection capabilities in two domains. At first we evaluated two types of anomalies in a common ICT network. Afterwards we evaluated the approach with a dataset from a SCADA system in a real industrial ICT infrastructure.

7.2 Key Findings

Based on the conducted evaluations we can split the results into two parts. First, we can conclude that the proposed approach is able to extract a system model from the combination of log-information that is collected from distributed systems and nodes in a network without prior knowledge about syntax and semantics of the log-lines (see hypothesis (i) in Sect. 1.2). The extracted system model can be used to detect and distinguish different meaningful subsets of log-lines (by the means of event classes), and further contains complex information about implications (so-called rules) between different log-events. Single rules often describe implications between events from different components or systems in the network. The set of rules is therefore sufficient to describe the most important relations between different components in the network.

Using this automatically generated and continuously evolving system model, the proposed approach is able to detect meaningful anomalies in the monitored system (see hypothesis (ii) in Sect. 1.2). The detection rate of different types of anomalies varies depending on the complexity of the dataset as well as on the effects of the anomalies on the generated log-lines. In very structured and well defined SCADA systems, even small anomalies can be discovered with a very high accuracy (see hypothesis (iii) in Sect. 1.2), but due to the limitations of the dataset these results have to undergo further evaluation. In more complex networks, a reoccurring anomaly might not be detected consequently by a single rule. Only by combining multiple rules in the model, the approach is able to detect the evaluated anomaly reliably. Although this behaviour is sufficient to detect most anomalies, it causes a high workload on the administrator when performing a root-cause analysis.

7.3 Future Work

The goal of this work was to implement and evaluate a novel anomaly detection approach. The results of the prototype that uses balancing methods and a simple statistical evaluation model look very promising. Future work should develop a more intelligent approach for the generation of event classes. The lack of information about similarities of event classes results in redundant hypotheses that choke the system model. However, a more intelligent approach for the generation of hypotheses is not possible without knowledge about relations or a hierarchy between event classes. Given an event class hierarchy, an improved algorithm for the generation of hypotheses should be developed. One approach would be an algorithm that generates all possible rules in a limited subtree of similar event classes and only allows the most meaningful or most stable hypothesis to become a rule.

Another potential direction is the improvement of the statistics model behind hypothesis and rule evaluations. The results show that only a set of rules can detect more complex anomalies with sufficient certainty. It is therefore not optimal to detect anomalies based only on single rules. Instead, a more complex model should be developed that considers dependent, instead of independent, events (as is done by simple binomial tests).

This work has established a stable implementation of the proposed approach. Therefore, it forms the basis for further development and research around the proposed algorithms. A lot of

new research questions opened up during the course of this project, such as those discussed here. Their solutions can significantly extend and improve the results achieved by this work.

Bibliography

[1] Dmitri Alperovitch et al. *Revealed: operation shady RAT*. McAfee, 2011.

[2] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical report, Chalmers University of Technology, 2000.

[3] Rafael Ramos Regis Barbosa and Aiko Pras. Intrusion detection in scada networks. In *Mechanisms for Autonomous Management of Networks and Services*, pages 163–166. Springer, 2010.

[4] Rafael Ramos Regis Barbosa, Ramin Sadre, and Aiko Pras. Towards periodicity based anomaly detection in scada networks. IEEE Industrial Electronics Society, 2012.

[5] Václav Bartoš and Martin Žádník. Network anomaly detection: comparison and real-time issues. In *Dependable Networks and Services*, pages 118–121. Springer, 2012.

[6] John Bigham, David Gamez, and Ning Lu. Safeguarding scada systems with anomaly detection. In *Computer Network Security*, pages 171–182. Springer, 2003.

[7] A Carcano, A Coletta, M Guglielmi, M Masera, I Nai Fovino, and A Trombetta. A multidimensional critical state analysis for detecting intrusions in scada systems. *Industrial Informatics, IEEE Transactions on*, 7(2):179–186, 2011.

[8] Andrea Carcano, Igor Nai Fovino, Marcelo Masera, and Alberto Trombetta. State-based network intrusion detection systems for scada protocols: a proof of concept. In *Critical Information Infrastructures Security*, pages 138–150. Springer, 2010.

[9] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.

[10] Eduardo B Fernandez, Jie Wu, MM Larrondo-Petrie, and Yifeng Shao. On building secure scada systems using security patterns. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*, page 17. ACM, 2009.

[11] Vinay M Igure, Sean A Laughter, and Ronald D Williams. Security issues in scada networks. *Computers & Security*, 25(7):498–506, 2006.

[12] M.G. Kanabar, I. Voloh, and D. McGinn. Reviewing smart grid standards for protection, control, and monitoring applications. In *Innovative Smart Grid Technologies (ISGT), 2012 IEEE PES*, pages 1–8, Jan 2012.

[13] Guichong Li, Nathalie Japkowicz, and Lian Yang. Anomaly detection via coupled gaussian kernels. In *Advances in Artificial Intelligence*, pages 343–349. Springer, 2012.

[14] Stuart McClure et al. *Protecting Your Critical Assets*. McAfee, 2010.

[15] PS Motta Pires and Luiz Affonso HG Oliveira. Security aspects of scada and corporate network interconnection: An overview. In *Dependability of Computer Systems, 2006. DepCos-RELCOMEX'06. International Conference on*, pages 127–134. IEEE, 2006.

[16] Takashi Onoda and Mai Kiuchi. Analysis of intrusion detection in control system communication based on outlier detection with one-class classifiers. In *Neural Information Processing*, pages 275–282. Springer, 2012.

[17] F Sabahi and A Movaghar. Intrusion detection: A survey. In *Systems and Networks Communications, 2008. ICSNC'08. 3rd International Conference on*, pages 23–26. IEEE, 2008.

[18] Florian Skopik and Roman Fiedler. Intrusion detection in distributed systems using fingerprinting and massive event correlation. *43. Jahrestagung der Gesellschaft für Informatik e.V. (GI) (INFORMATIK 2013)*, 2013.

[19] Florian Skopik, Ivo Friedberg, and Roman Fiedler. Dealing with advanced persistent threats in smart grid ict networks. In *5th IEEE Innovative Smart Grid Technologies Conference*. IEEE, 2014.

[20] Florian Skopik and Lucie Langer. Cyber security challenges in heterogeneous ict infrastructures of smart grids. *Journal of Communications*, 8(8), 2013.

[21] Nils Svendsen and Stephen Wolthusen. Modeling and detecting anomalies in scada systems. In *Critical Infrastructure Protection II*, pages 101–113. Springer, 2009.

[22] Colin Tankard. Advanced persistent threats and how to monitor and deter them. *Network security*, 2011(8):16–19, 2011.

[23] Mahbod Tavallaee, Natalia Stakhanova, and Ali Akbar Ghorbani. Toward credible evaluation of anomaly-based intrusion-detection methods. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 40(5):516–524, 2010.

[24] Chee-Wooi Ten, Chen-Ching Liu, and Govindarasu Manimaran. Vulnerability assessment of cybersecurity for scada systems. *Power Systems, IEEE Transactions on*, 23(4):1836–1846, 2008.

[25] Marina Thottan and Chuanyi Ji. Anomaly detection in ip networks. *Signal Processing, IEEE Transactions on*, 51(8):2191–2204, 2003.

[26] Walter F Tichy, Paul Lukowicz, Lutz Prechelt, and Ernst A Heinz. Experimental evaluation in computer science: A quantitative study. *Journal of Systems and Software*, 28(1):9–18, 1995.

[27] Jacques Wainer, Claudia G Novoa Barsottini, Danilo Lacerda, and Leandro Rodrigues Magalhães de Marco. Empirical evaluation in computer science research published by acm. *Information and Software Technology*, 51(6):1081–1085, 2009.

[28] Jian Yin, Gang Zhang, Yi-Qun Chen, and Xian-Li Fan. Multi-events analysis for anomaly intrusion detection. In *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, volume 2, pages 1298–1303. IEEE, 2004.

[29] Yingbing Yu. A survey of anomaly intrusion detection techniques. *Journal of Computing Sciences in Colleges*, 28(1):9–17, 2012.

[30] Weiyu Zhang, Qingbo Yang, and Yushui Geng. A survey of anomaly detection methods in networks. In *Computer Network and Multimedia Technology, 2009. CNMT 2009. International Symposium on*, pages 1–3. IEEE, 2009.

[31] Ya-ling Zhang, Zhao-guo Han, and Jiao-xia Ren. A network anomaly detection method based on relative entropy theory. In *Electronic Commerce and Security, 2009. ISECS'09. Second International Symposium on*, volume 1, pages 231–235. IEEE, 2009.

[32] Ying Zhao, Zhigao Zheng, and Hong Wen. Bayesian statistical inference in machine learning anomaly detection. In *Communications and Intelligence Information Security (ICCIIS), 2010 International Conference on*, pages 113–116. IEEE, 2010.