Available online at www.sciencedirect.com
ScienceDirect
journal homepage: www.elsevier.com/locate/cose
Computers & Security

Combating advanced persistent threats: From network event correlation to incident detection

Ivo Friedberg, Florian Skopik*, Giuseppe Settanni, Roman Fiedler

Austrian Institute of Technology, Safety and Security Department, Donau-City-Straße 1, 1220, Vienna, Austria

ARTICLE INFO

Article history:
Received 11 June 2014
Received in revised form
3 September 2014
Accepted 28 September 2014
Available online 13 October 2014

Keywords:
Advanced persistent threat
Anomaly detection
Log file analysis
Intrusion detection
Event correlation
Self-learning system model

ABSTRACT

An advanced persistent threat (also known as APT) is a deliberately slow-moving cyber-attack that is applied to quietly compromise interconnected information systems without revealing itself. APTs often use a variety of attack methods to get unauthorized system access initially and then gradually spread throughout the network. In contrast to traditional attacks, they are not used to interrupt services but primarily to steal intellectual property, sensitive internal business and legal documents and other data. If an attack on a system is successful, timely detection is of paramount importance to mitigate its impact and prohibit APTs from further spreading. However, recent security incidents, such as Operation Shady Rat, Operation Red October or the discovery of MiniDuke – just to name a few – have impressively demonstrated that current security mechanisms are mostly insufficient to prohibit targeted and customized attacks. This paper therefore proposes a novel anomaly detection approach which is a promising basis for modern intrusion detection systems. In contrast to other common approaches, which apply a kind of black-list approach and consider only actions and behaviour that match to well-known attack patterns and signatures of malware traces, our system works with a white-list approach. Our anomaly detection technique keeps track of system events, their dependencies and occurrences, and thus learns the normal system behaviour over time and reports all actions that differ from the created system model. In this work, we describe this system in theory and show evaluation results from a pilot study under real-world conditions.

1. Introduction

Global connectivity is the core principle of our information age (O'Neill, 2014) and the vital backbone for our economy. From an information provisioning 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. This dependency, however, is a considerable vulnerability too, and increasingly motivates a certain criminal exploitation (Barber, 2001). As ICT networks and their complexity have evolved in recent years, so did the goals and technical progress of attacks (Steer, 2014; Sood and Enbody, 2013). Further, the motivation for attacks has changed from causing immediate damage on abroad basis to more sophisticated and targeted forms of attacks, where stealing proprietary information or personal data is just one step in a multi-stage attack (Kraemer-Mbula et al., 2013; Tankard, 2011; Kjaerland, 2006; Caldwell, 2013).

* Corresponding author.
E-mail addresses: florian.skopik@ait.ac.at, florian.skopik@gmx.at (F. Skopik).
http://dx.doi.org/10.1016/j.cose.2014.09.006

Since the emergence of the first ICT networks significant effort went into securing critical assets. Most companies have numerous guidelines and processes in place to decrease the chance of human failure. Additionally, one can choose from a variety of security solutions that deal with different attack schemes at different levels in the network: Firewalls that filter traffic at network borders between sub-networks, malware scanners that investigate binaries and executables for suspicious behaviour or intrusion detection systems (IDSs) that monitor events all over a network and verify them against predefined rules for anomalies. While IDSs are a widely accepted de-facto standard today, their common signature based approach brings two major drawbacks (Garca-Teodoro et al., 2009):

i Pre-defined rules are often insufficient to detect unique or tailored attacks. Those rules are often commonly known and, given enough effort is invested, can be circumvented. However, producing custom rule sets is mostly not feasible for most organizations.
ii For applications with low market share no sufficient parsers and rule sets are available (e.g., for specific industrial control components). As a consequence, the lack of sufficient rule sets that verify application specific operation sequences make common solutions inapplicable.

System architects and administrators need to tackle these inadequacies in order to increase security. However, recent attacks like Operation Aurora (McClure et al., 2010) or Operation Shady RAT (Alperovitch et al., 2011) 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 (Alperovitch et al., 2011; Thomson, 2011). Some new approaches even deliberately accept successful first stages of attacks, and instead focus on the timely detection to limit negative effects on the longer run (Brewer, 2014).

Eventually, timely detection of intrusions is one major challenge when securing complex critical systems. As current security solutions are often not sufficient to deal with sophisticated and tailored attacks, novel approaches are required. One interesting core concept of these novel approaches is the mining of typically unnoticed relationships between different applications and components of a computer network. While many experts today argue that these disregarded relationships are the major weak spot abused by attackers to compromise systems (Patcha and Park, 2007) (e.g., users that utilize the same passwords on multiple services (Ives et al., 2004) or standard architectural patterns), we emphasize disregarded (and often much more subtle) relationship as one of the major strength for future intrusion detection systems.

This means, if we are able to detect these relations and harness them by correlating events on multiple machines across the entire network – and thus discovering unknown dependencies – we are able to automatically generate a system behaviour model that describes the common events and their relations. For example, in a hosting environment a quite obvious relation exists between incoming connections on a firewall, followed by an http request on a Web server and finally an issued SQL query on a database server. Consequently, a single direct SQL query without a preceding Web server request is an anomaly; moreover a firewall entry without a succeeding Web server request might be the sign of an intrusion too. The stronger these events coming from machines, network devices, and high level services, are connected, the harder it is for an attacker to exploit vulnerabilities without violating some of these implications and thus being detected. The actual art is to find these relations, model them, and enforce them with minimal human intervention – and with acceptable false positive rates.

The contributions of this article are as follows:

- *APT Detection Approach*. We explain why current security solutions are often insufficient to counter APT attacks and motivate the need for novel approaches.
- *Anomaly Detection Model*. We discuss the formal model definition of a novel anomaly detection approach based on log-line analysis (Skopik et al., 2014a; Skopik and Fiedler, 2013) that fulfils the motivated requirements.
- *Real-World Evaluation*. We perform a sophisticated evaluation of a prototype implementation of the presented approach, including an optimal configuration and the performance when challenged with real anomalies in a large-scale dataset.

The remainder of this paper is organized as follows: Section 2 gives an overview about recent research in the field of anomaly detection and intrusion detection systems. Sections 3 and 4 define the novel anomaly detection approach. Section 3 defines the general functionality; Section 4 describes how the system model is generated and continuously refined. Section 5 describes the test environments and the test data generation for the evaluation of the prototype implementation. Section 6 discusses the results of the different evaluation steps, and Section 7 concludes this article.

2. Related work

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 (von Solms and van Niekerk, 2013). Any action attempting a violation of any of those properties can be seen as an intrusion (Yu, 2012). Intrusion Detection Systems (IDSs) (as originally proposed by Denning (1987)) aim at detecting those intrusions to take actions from triggering warnings to actively preventing the attacker from causing further harm. Literature as (Yu, 2012) or Sabahi and Movaghar (2008) classifies IDSs by different means. Differentiations can be made between host based, network based and hybrid approaches (Yu, 2012; Sabahi and Movaghar, 2008). 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 analyse the traffic and protocol data to detect intrusions. Sabahi and Movaghar (2008)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 (Axelsson, 2000). 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 (Sabahi and Movaghar, 2008). Anomaly intrusion detection detects deviations of the events in the monitored domain from an as normal behaviour defined base line (Yu, 2012; Garca-Teodoro et al., 2009). Here further differentiation can be taken between statistical based, distance based, rule based, profile based and model based methods (Sabahi and Movaghar, 2008). This way to differentiate intrusion detection systems also leaves the possibility of hybrid approaches as proposed for example by Kim et al. (2014). Yu (2012) 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 (Sommer and Paxson, 2010; Lee et al., 1999; Zhang et al., 2009b). Intrusion detection systems can also differ by their application domain. Mitchell and Chen (2014) for example provides a comprehensive survey on current IDS research in cyber-physical systems.

Anomaly detection is an actively researched field in many domains. Chandola et al. (2009) 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 (Chandola et al., 2009; Zhang et al., 2009a). 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 (Zhang et al., 2009a). Chandola et al. (2009) distinguishes three kinds of anomalies: **(i) Point Anomalies**, if a single event can be considered anomalous given the notion of normality we call it point anomaly. **(ii) Contextual Anomalies** when 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. **(iii) 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 and Ji (2003) describe 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 (Chandola et al., 2009; Zhang et al., 2009a; Yu, 2012; Thottan and Ji, 2003) just to name a few. The broadest classification by Chandola et al. (2009) 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: (i) *Neural Networks-Based*, (ii) *Bayesian Networks-Based*, (iii) *Support Vector Machines-Based* and (iv) *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" (Chandola et al., 2009). 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. First, when using *Distance to k^{th} Nearest Neighbour*, the anomaly score is the distance to its k^{th} nearest neighbour. Second, when 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 et al. (2009a) and Thottan and Ji (2003) distinguish anomaly detection approaches with focus on ICT networks. They come up with subsets of the already described approaches before with Zhang et al. (2009a) using *anomaly detection using statistics*, *anomaly detection using classifier*, *anomaly detection using machine learning* and *anomaly detection using finite state machines*. Thottan and Ji (2003) distinguishes between *rule-based approaches*, *finite state machines*, *pattern matching* and *statistical analysis*. Various challenges in anomaly detection are identified by Chandola et al. (2009). 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 and changing border between normal and abnormal behaviour. 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 (Bartoš and Žádník, 2012). The number of papers describing novel or optimised mechanisms of anomaly detection in different contexts is far from countable and ever increasing. Li et al. (2012); Zhang et al. (2009b); Zhao et al. (2010); Thottan and Ji (2003); Yin et al. (2004) 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 Yu (2012).

The approach proposed in this work 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.

3. Anomaly detection through event correlation

The proposed approach is 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 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 approach constructs a model while processing the input. This system model M is built by the following items:

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[1] 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. Given the abstract form the algorithm transfers processed log-lines to, information sharing is possible without compromising privacy. 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.

Fig. 1 provides a conceptional overview of the described approach's functionality, as first published partly in Skopik et al. (2014a). This visualisation splits the approach 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

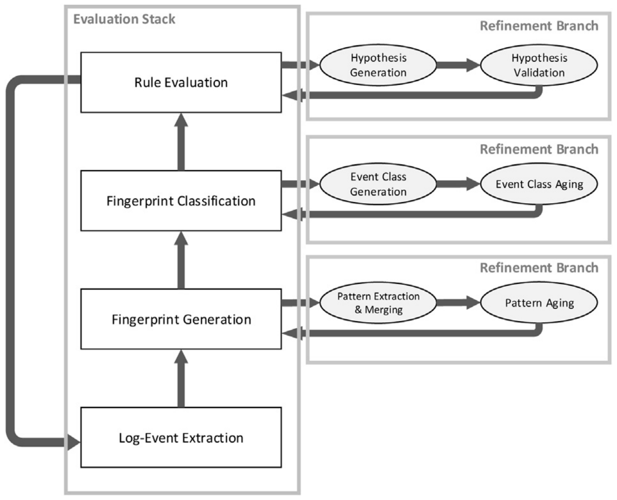


Fig. 1 – Conceptual overview.

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

operations iteratively. The *Refinement Branches* on the other hand, are triggered by the *Evaluation Stack's* tasks; they evaluate and optimise the system model continuously. Fig. 1 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 Eq. (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 $\mathbb{R}$.

$$M = \langle \mathbb{P}, \mathbb{C}, \mathbb{H}, \mathbb{R} \rangle \tag{1}$$

Tasks in the *Evaluation Stack* are performed in sequential order. The tasks describe the general functionality of the system regarding input analyses and anomaly detection:

(Eq. (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 Table 1 for an example).

$$\vec{F} = p_1 \dots p_n, \quad \textit{where} \ \ p_i \in \{0, 1\} \tag{5}$$

Fingerprinting L_a reduces the amount of data the next steps need to process – and speeds up the overall anomaly detection – significantly. $\vec{F}$ 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 $\mathbb{P}$, is inevitably lost for further steps in the *Evaluation Stack*.

Fingerprinting Example

```
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 1: Apache log excerpt from test environment.

Table 1 – Example of a fingerprint. Consider the sample log-line in Lst. 1 from an Apache server running Mantis in our test environment. This table shows an example set of patterns $\mathbb{P}'$ and how a fingerprint of the line in Lst. 1 would look like.

	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

3.1. *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. (2)).

$$L_a = s_1 \dots s_n \tag{2}$$

Further a log event L_e (Eq. (3)) is the association of a log-atom L_a with a timestamp t; L_e describes when L_a has been created.

$$L_e = \langle L_a, t \rangle \tag{3}$$

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. *Fingerprint generation*

The next task vectorises each log-atom using $\mathbb{P}$. A search pattern P (Eq. (4)) is a substring (an *n-gram*) of a log-atom L_a.

$$P = s_{1+i} \dots s_{m+i}, \quad \textit{where} \ \ 0 \le i \ \textit{and} \ m + i \le n \tag{4}$$

The vectorisation process transforms a log-atom L_a into an n-dimensional pattern vector – the so-called fingerprint $\vec{F}$

3.3. *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 $\mathbb{C}_{L_a}$ – the set of all event classes C a log-atom L_a belongs to (see Eq. (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.

$$\mathbb{C}_{L_a} = \{C | L_a \in C\} \tag{6}$$

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. (7)) is defined as the combination of a mask $\vec{C}_m$ and a value $\vec{C}_v$.

$$C = \langle \vec{C}_m, \vec{C}_v \rangle \tag{7}$$

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. (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 by C (see Eq.

(9)). Note that $\vec{C}_v$ does only enforce or prohibit search patterns that are considered relevant by $\vec{C}_m$.

$$\vec{C}_m = p_1 \ldots p_n \quad \text{where } p_i = \begin{cases} 1 & \text{if } P \text{ at } i \text{ is relevant} \\ 0 & \text{if } P \text{ at } i \text{ is irrelevant} \end{cases} \tag{8}$$

$$\vec{C}_v = p_1 \ldots p_n \quad \text{where} \\ p_i = \begin{cases} 1 & \text{if } P \text{ at } i \text{ is enforced} \\ 0 & \text{if } P \text{ at } i \text{ is prohibited or irrelevant} \end{cases} \tag{9}$$

Each generated fingerprint $\vec{F}$ is classified by C if the condition in Eq. (10) holds.

$$\vec{C}_v = \vec{F} \wedge \vec{C}_m \tag{10}$$

Fingerprint Classification Example

The idea of the log-atom classification 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. 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 *GET*-requests on a specific page (see Table 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 Table 4 for an example). In general a prohibited pattern carries less information than an enforced one.

Table 2 – Example of an event class that classifies the sample log-line in Lst. 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.

	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 – Example of an event class that prohibits*/mantis/login_page.php*. The line in Lst. 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	0	0

Table 4 – Example of an event class that enforces*/mantis/login_page.php*. The line in Lst. 1 is classified by this event class C but only apache *GET*-requests on this 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

After classifying L_a the approach generates an event E^C (Eq. (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.

$$E^C = \langle t, C \rangle \tag{11}$$

3.4. 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. (12)) is a non-validated correlation rule between events of two different 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. 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. 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.

$$H = \langle C_{cond}, C_{impl}, \rightarrow, t_w \rangle \tag{12}$$

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 $E^{C_i}\Big|_{C=C_{cond} \vee C=C_{impl}}$. A periodic evaluation process acts on the entries in the respective Q_i; its actions are described in Table 5. 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.

Table 5 – Possible evaluation results of a hypothesis.

Occurrence	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.

The result of an evaluation of Q_i – one evaluation – is called e (see Eq. (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.

$$e = \langle res, H, pos \rangle \tag{13}$$

Every hypothesis H_i stores evaluations in a stream S^{H_i} (see Eq. (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 $pos = 0$.

$$S^{H_i} = \{e | H = H_i\} \tag{14}$$

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. (15)). The system generates k slots at initialization time with $size(Sl_i) < size(Sl_{i+1})$.

$$Sl(S^H) = \{e | e \in S^H \wedge pos < size(Sl_i)\} \tag{15}$$

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. (16) implements a left-sided binomial test; *isS table*(*H*) analyses the current evaluation stream against a pre-defined stochastic distribution that is described by a statistical hypothesis p_0.[2] A binomial test evaluates, if a given sample supports a pre-defined distribution. $B(i|p_0, n)$ 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.[3] 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.

$$isStable(H) = \sum_{i=0}^{|\{e_t^H \in Sl_k(S^H)\}|} B(i|p_0, size(Sl_k) \geq \alpha \tag{16}$$

If *isS table*(*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. (17)). Note that $\mathbb{R} \subset \mathbb{H}$.

$$\mathbb{R} = \{H | isStable(H)\} \tag{17}$$

Rules are considered currently accepted hypotheses; they undergo the same periodical evaluations as all hypotheses. Additionally, their evaluation stream is analysed for anomalies. 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. (19)). 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[4] p_0, that is now given by the ratio of positive evaluations in Sl_k to total evaluations in Sl_k (see Eq. (18)). 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 $i<k$.

[2] p_0 in this case relates to a hypothesis about the statistical distribution of the sample and is not comparable to an implication hypothesis as defined in Eq. (12).

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

[4] 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).

$$p_0 = \frac{\left|\{e_t^R \in Sl_k(S^R)\}\right|}{size(Sl_k)} \tag{18}$$

$$isAnomalous(R) = \bigvee_{j=0}^{k-1}\left(\sum_{i=0}^{\left|\{e_t^R \in Sl_j\}\right|} B(i|p_0, size(s_j)) \leq \alpha_a\right) \tag{19}$$

An anomaly A (see Eq. (20)) 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 $p = 1 - \sum_{i=0}^{\left|\{e_t^R \in Sl_j\}\right|} B(i|p_0, size(Sl_j))$, a rule R, on which the anomaly was detected, and a time t, at which the anomaly was detected.

$$A = \langle p, R, t \rangle \tag{20}$$

Rule Evaluation Example

Example: Consider the event class described in Table 4 as C_1 and the event class described in Table 3 as C_2. We can now construct a hypothesis as in Eq. (21). 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 s Table 6 shows the status of Q_1 at different timestamps. Table 7 shows how different evaluation results affect the evalution stream S^{H_1} and how different slots access different evaluations.

$$H_1 = \langle C_1, C_2, +10s \rangle \tag{21}$$

Taking into account the last snapshot of S^{H_1} in Table 7, we can calculate p_0 by counting the number of positive evaluations that are considered by Sl_3 ($\left|\{e_t^R \in Sl_k(S^R)\}\right| = 9$). The size of Sl_3 is 10 and the resulting p_0 is then calculated based on Eq. (18) as 0.9. Table 8 describes all possible values for Sl_1 and Sl_2 that can be calculated by Eq. (19). Let's assume that $\alpha_a = 0.01$. In this case:

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

Table 6 – Sample status of the event queue Q_1.

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 7 – 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 $size(Sl_3) = 10$.

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 8 – This table describes the values of a Cumulative Distribution Function (CDF) based on the slots from the last timeslot in Table 7. All possible values are calculated; the bold values are the ones valid for the values in Table 7.

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

4. System model management

The analysis of system behaviour, as described in Section. 3, 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 redundant information. The following section 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 $\mathbb{R}$.

4.1. *Search pattern refinement*

Generation. Search patterns $\mathbb{P}$ (see Eq. (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. 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. 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).

Generation of new search patterns P is balanced by generating new patterns more frequently for uncovered or weakly covered log-atoms.[5] 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 cost of a new pattern τ. It is calculated by the configured base cost τ' plus a balancing cost τ'' that depends on the coverage of L_e. This algorithm is an easy way to overcome the *pattern balancing problem* that deals with the fact that rarely occurring log-atoms are not properly indexed by the search patterns currently 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

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

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. The generation of search patterns for well covered log-atoms is not prevented, in order 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: $L_a$, $\tau'_P$
   Result: $P$
1  $bucketSum$++
2  $matchingCount \leftarrow$ sum_matching_search_patterns($L_a$)
3  $\tau \leftarrow \tau'_P + 2^{matchingCount}$
4  if $bucketSum \geq \tau$ then
5      $candidate$ = get_random_substring()
6      if $\exists P \equiv substring$ then
7          $P \leftarrow substring$
8          $bucketSum = bucketSum - \tau$
9      end
10 end
```

Algorithm 1: Creation of a new search pattern P.

Balancing Example

Let's assume $\tau'_P = 10$ and L_a is the log-line from Lst. 1. Lets further assume that $\mathbb{P}'$ is again given by the set of patterns used in Table 1. Based on $\overrightarrow{F}_{L_a}$ the *matchCount* is 6 and the total cost τ would be calculated as: $\tau = 10+2^6 = 74$. So if *bucketSum* ≥ 74 a new pattern will be generated on L_a. Otherwise no pattern gets generated.

Merging. Each L_a can be classified by multiple event classes and can therefore trigger a multitude of events E. These events are then used in hypotheses and rules to analyse implications between event classes. 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. (22) between two search patterns.

$$\text{iff } P_1 \in L_a \rightarrow P_2 \in L_a, \quad \forall L_a \tag{22}$$

A merge candidate P' is a new search pattern, that is created by merging two existing search patterns. A merge is considered possible, if (i) one pattern is a substring of another pattern, (ii) two patterns overlap, (iii) two patterns can be found next to each other in L_a or (iv) two patterns are only separated by a space in L_a. If one condition holds, Eq. (22) gets evaluated before the merge is actually performed. 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. (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.ses the Pr Given a specified threshold[6] the system evaluates the number of times a substitution of P_1 and P_2 by their merge candidate P' would be valid.[7] A merge is considered iff the condition in Eq. (23) holds:

$$P_1 \in L_e \leftrightarrow P_2 \in L_e, \quad \forall L_e \text{ with } t\ now \tag{23}$$

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. After a merge one pattern gets deleted while the other pattern is changed to the merge candidate P'. This merge makes it necessary to update all event classes. Table 9 shows the required changes on $\overrightarrow{C}_m$ and $\overrightarrow{C}_v$ in each event class to mitigate second level effects.

Ageing. 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. Patterns can be categorised into (i) periodically occurring patterns (e.g.: time-stamps), (ii) rare patterns (e.g.: session ids, or measurement values) and (iii) reoccurring patterns (e.g.: enumeration values, usernames, …). Most information is carried by *reoccurring* patterns: **Localisers** can give information about source and destination of a logged event. They can be used to identify involved parties. **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. 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.

Ageing 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 ageing:

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.

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

[7] 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 matched.

Table 9 – Changes performed on event classes after pattern merge.

	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 enforce p'_1 to classify the same log-atoms as before the merge.
$\vec{C}_v$	1	1	1	
$\vec{C}_m$	1	1	–	This event class should get deleted. Since 22 holds: $\nexists L_a : \neg P_1 \in L_a \wedge P_2 \in L_a$.
$\vec{C}_v$	0	1	–	
$\vec{C}_m$	0	1	1	P_1 is ignored. Since Eq. (22) holds $P_1 \in L_a$ has to hold anyway for L_a to be in C.
$\vec{C}_v$	0	1	1	
$\vec{C}_m$	1	1	–	This event class should get deleted since Eq. (22) holds: $\nexists L_a : P_1 \in L_a \wedge \neg P_2 \in L_a$.
$\vec{C}_v$	1	0	–	
$\vec{C}_m$	1	1	1	Both search patterns are prohibited. The same has to be true for there merged successor.
$\vec{C}_v$	0	0	0	
$\vec{C}_m$	0	1	1	One search pattern is ignored, so the other search pattern determines the status of the merged one.
$\vec{C}_v$	0	0	0	
$\vec{C}_m$	1	0	1	One search pattern is ignored, so the other search pattern determines the status of the merged one.
$\vec{C}_v$	1	0	1	
$\vec{C}_m$	1	0	1	One search pattern is ignored, so the other search pattern determines the status of the merged one.
$\vec{C}_v$	0	0	0	
$\vec{C}_m$	0	0	0	Both search patterns are ignored. The same has to be true for the merged search pattern.
$\vec{C}_v$	0	0	0	

Pattern Merging Example

Table 10 – 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 $\mathbb{P}\prime$ transcended into $\mathbb{P}''$ after the old P_4 was deleted. $\vec{C}_m$ and $\vec{C}_v$ are adapted accordingly.

		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	
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	

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).

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.[8] In a corporate environment T will typically be set to one week. Given that there are daily and weekly backup processes, a 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. (24) [9] and Eq. (25) hold:

$$\nexists L_e | (t_{now} - \rho_P * T < t_{L_e} < t_{now}) \wedge P \in L_e \tag{24}$$

$$\nexists C | \vec{C}_m(i) = 1 \quad \text{with } P_i \text{ being the ageing candidate} \tag{25}$$

This approximation substitutes the proof, that a search pattern P is not occurring any more in future log-events. In the condition in Eq. (24) T is used to ensure that search patterns, that

[8] This definition considers normal behaviour in the monitored system.

[9] 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.

describe periodic log-events, do not get deleted during normal system behaviour. Search patterns that get deleted because of ageing, cannot be substituted by other search patterns, carrying the same information. Second level effects on event classes, from deleting a search pattern because of ageing can only be eliminated by the condition in Eq. (25); it ensures 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.

4.2. Event class refinement

Generation. The system defines event classes C automatically using a similar token bucket algorithm[10] 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 to detect anomalies. The system ensures the following two cases in order to balance event class generation:

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

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, is inverse proportional to the percentage of log-events in a finite input set classifiable by C (see Eq. (26)).

$$Entropy(E_C) = \frac{|\{L_e\}|}{|\{L_e | C \text{ classifies } L_e\}| - 1} \quad (26)$$

Trivial event classes that classify all possible log-events, carry no information about a specific log-event. Table 11 establishes three parameters used to ensure proper levels of entropy on newly generated event classes.

Table 11 – Configuration parameters used to ensure 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.

[10] 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.

Algorithm 2 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.

```
Data: $\vec{F}, \mu_e, \phi_p, \phi_e, \tau'_C, \tau''_C$
Result: $\vec{C}_v, \vec{C}_m$
1  $bucketSum{+}{+}$
2  $matchingCount \leftarrow$
   numberOfClassifyingEventClasses($\vec{F}$)
3  $\tau \leftarrow \tau'_C + (matchingCount * \tau''_C)$
4  if $bucketSum \geq \tau$ then
5      $matching\_bits \leftarrow \{p_i \mid p_i \in \vec{F} \wedge p_i = 1\}$
6      $number\_enforced\_bits \leftarrow \phi_e * |matching\_bits|$
7      $not\_matching\_bits \leftarrow \{p_i \mid p_i \in \vec{F} \wedge p_i = 0\}$
8      $number\_prohibited\_bits \leftarrow \phi_p * |not\_matching\_bits|$
9      if $|matching\_bits| < \mu_e$ then
10         abort
11     end
12     if $number\_enforced\_bits < \mu_e$ then
13         $number\_enforced\_bits \leftarrow \mu_e$
14     end
15     enforceNRandomPatterns($number\_enforced\_bits$,
       $\vec{C}_m, \vec{C}_v, \vec{F}$)
16     prohibitNRandomPatterns($number\_prohibited\_bits$,
       $\vec{C}_m, \vec{C}_v, \vec{F}$)
17 end
```

Algorithm 2: Creation of a new class C.

Ageing. 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 Section 3, 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 ageing of event classes is a periodic process on $\mathbb{C}$ that identifies and removes unused elements $C \in \mathbb{C}$.

An event class gets deleted if the conditions in Eq. (27) and in Eq. (28) hold:

$$\nexists L_e | (t_{now} - \rho_C * T < t_{L_e} < t_{now}) \wedge C \text{ classifies } L_e \quad (27)$$

$$\nexists H \in \mathbb{H} | C \quad \text{relevant for } H \quad (28)$$

The concept is similar to the concept of the ageing process on $\mathbb{P}$. Hypotheses depend on certain event classes. The condition in Eq. (28) ensures that the ageing 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 ageing process therefore has to take $\mathbb{H}$ into account. Given that the condition in Eq. (28) holds, the condition in Eq. (27) 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.

rules $\mathbb{R}$ is a subset of the set of hypotheses $\mathbb{H}$. A hypothesis is considered to be a rule if *isS table*(H) (see Eq. (16)) evaluates to true. A Probability Density Function (PDF) states the probability for every number of positive elements in the sample; a Cumulative Distribution Function (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. Eq. (29) shows the formula to calculate a PDF; Eq. (30) shows the calculation of a CDF.

$$PDF(x) = \binom{n}{x} p^x (1-p)^{n-x} \tag{29}$$

$$CDF(x) = \sum_{k=0}^{x} \left(\binom{n}{k} p^k (1-p)^{n-k} \right) \tag{30}$$

Eq. (16) describes 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. (13)) can be found in S^H (see Eq. (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. (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.

Event Class Generation Example

Table 12 – 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 Table 1.

	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

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 $\{L_e|C \text{ classifies } L_e\}$.

4.3. *Hypothesis and rule refinement*

Generation. The system generates hypotheses based on $\mathbb{C}_{L_a}$ (see Eq. (6)) – the set of event classes that classify the currently processed log-atom. 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 choice of the connected event classes and the time window t_w is random. 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 $\mathbb{H}$ and (iii) hypotheses that were already discarded by the ageing process (as well as substitutes of those).

Stability. The periodic generation process only generates hypotheses and cannot extend the set of rules directly. The set of

Ageing. The ageing 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. (31) states, that the ageing 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 ageing process before H got generated (see Eq. (27)). But now that $H \in \mathbb{H}$ the condition in Eq. (28) prohibits the deletion of the condition event class. H can never be evaluated, and therefore it has to be deleted by the ageing process. Eq. (32) describes this condition. The ageing 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. (14) e_0 is the newest evaluation in any S_H.

$$isStable(H) = \text{ false} \quad (31)$$

$$time(e_0 \in S^H) < now - \rho_H * T \quad (32)$$

In contrast to the ageing processes for search patterns and event classes, the ageing process of hypotheses is applied if any of the before mentioned conditions holds. There are no atoms in M that H depends on. Eq. (31) also shows, that ageing can only be applied on hypotheses, never on rules since $\{\nexists R \in \mathbb{R} | isStable(R) = \text{ false}\}$.

Table 13 – Differences between recorded datasets.

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

5. Test environment

The following evaluations in Section 6 use three datasets that were generated with a semi-synthetic data generation approach as described and evaluated by Skopik et al. (2014b); 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 malicious users or users with erroneous behaviour. The behaviour of virtual users is based on an analysis of user behaviour in a productive system with a similar setup over 3 months.

For the following evaluation, two datasets were recorded on a clean system. They will be used in the further evaluations of system parameters and performance when generating the system model M. 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 13 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.

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 h and 30 min 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 h 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 min long and for each type of anomaly, four different slots are generated. Fig. 2 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. 2 a malicious script dumps all databases on the central database server. This dump is not performed remotely but directly on the 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. 2, 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 s to 4 min; each time slot doubles the anomalous time span of the previous one.

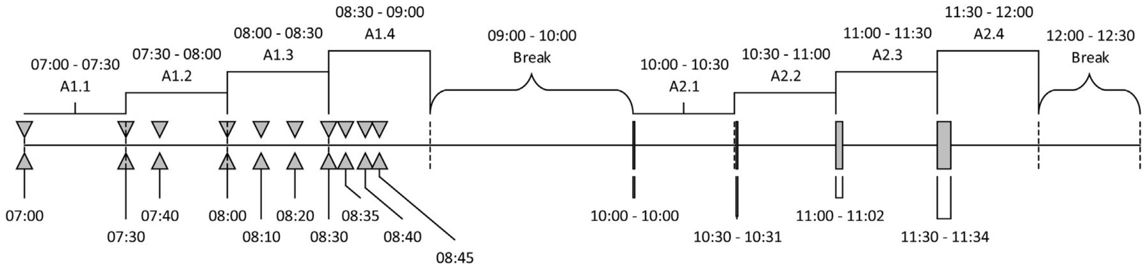


Fig. 2 – Timeline of the *Anomalous Phase* in the anomalous dataset.

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.

6. Evaluation

The following section discusses the evaluation of the approach, performed on the dataset described in Section 5. An evaluation of the configuration parameters in Section 6.1 precedes a detailed evaluation of the anomaly detection capabilities in Section 6.2. Additional evaluations were performed about the quality of the system model or the applicability of the approach in the domain of SCADA systems.

6.1. *Parameter evaluation*

As a first step this section 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 Section 4). For the given datasets, a period of 20 min 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 Skopik et al. (2014b)), the actions of the simulated users reach a request distribution, similar to a productive setting after 15 min. 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 min, 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 14 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 14 states 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 section.

Starting from the described base configuration all parameters in Table 14 get changed to various values separately. This evaluation is performed on both clean datasets from Section 5 (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:

Table 14 – Basic configurations of relevant system parameters.

Parameter	Setting
Pattern Base Cost (τ'_P)	1
Event Class Base Cost (τ'_C)	1
Event Class Balancing Cost (τ''_C)	30
Hypothesis Base Cost (τ'_H)	30
Hypothesis Balancing Cost (τ''_H)	300
Prohibited Patterns (ϕ_p)	50%
Enforced Patterns (ϕ_e)	40%

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 $\mathbb{H}\setminus\mathbb{R}$ in M (also $\frac{|\mathbb{R}|}{|(\mathbb{H}\setminus\mathbb{R})|}$).
Number of Anomalous Rules: The number of rules $\mathbb{R}$ that detected any anomaly during the execution.[11]
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

[11] 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.

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. *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*.
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.

Fig. 3 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. Fig. 4 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. *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 Section 5. 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.

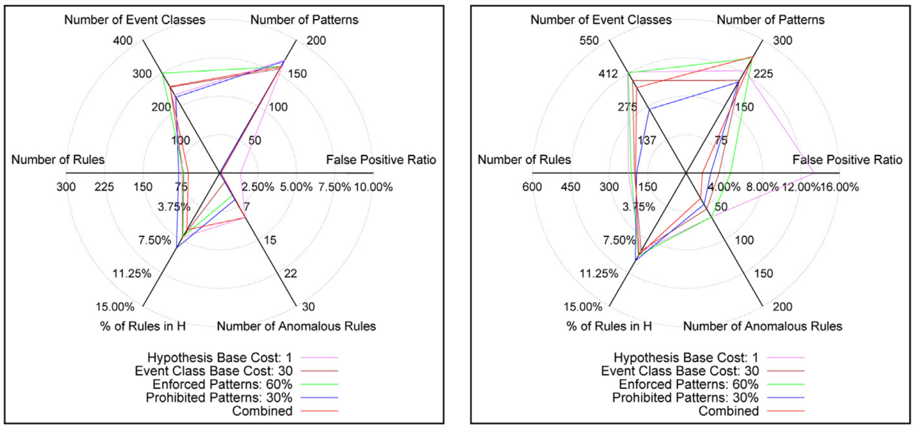
6.2.1. *Metrics*

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 Eq. (33) is therefore the sum of false positives and true negatives (i.e., the number of events correctly considered normal).

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{33}$$

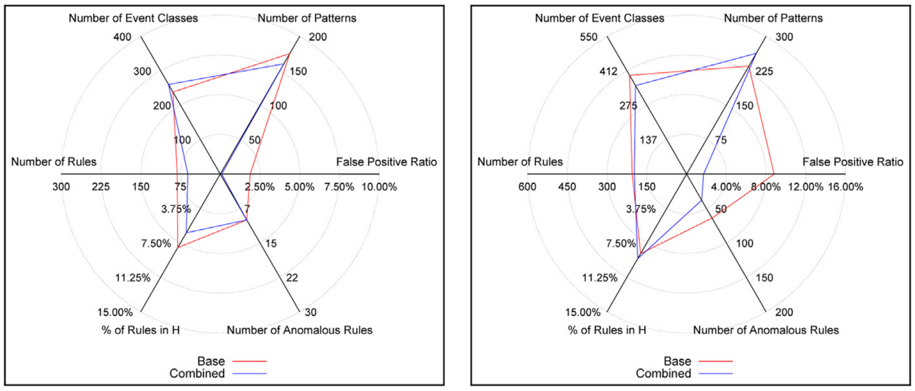
In order to calculate this rate a definition of false positives and true negatives regarding the system under evaluation is required. We define them as follows:

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) 2U8h (b) 4U12h

Fig. 3 – Comparison of the combined configuration with the different single parameter adaptions.



(a) 2U8h (b) 4U12h

Fig. 4 – Comparison of the combined configuration with the base configuration.

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. Eq. (34) 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).

$$TPR = \frac{TP}{TP + FN} \tag{34}$$

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.

6.2.2. *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 min. Each timeslot contains one more injected anomaly than the previous one (see also Section 5).

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 s. This means that at most 10 s before a `SHOW TABLES` command is logged, a connection to the database has to has happened. The last lines matched by the condition and the implied event are the following:

Condition Event: database-0.v3ls1316.d03.arc.local mysql-normal #011#01133179 Query#011SHOW TABLES
Implied Event: database-0.v3ls1316.d03.arc.local mysql-normal #011#01133179 Connect#011mantis_user_4@service-4.v3ls1316.d03.arc.local on

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.

Fig. 5 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

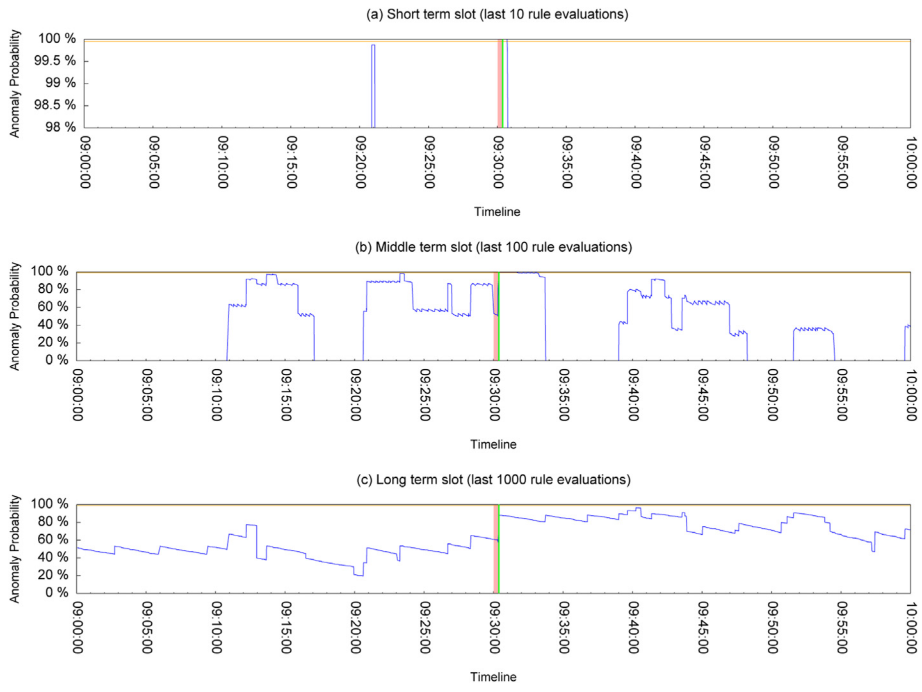


Fig. 5 – Overview of the slot stabilities in rule 849 during the injection slot A1.1.

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 2 s after the injection finished), is marked by the green line.

Fig. 5 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 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 s. 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. The last lines matched by the condition event class and the implied event class were as follows:

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

Implied Event: v3ls1316.d03.arc.local kernel: [165361.740449] iptables:ACCEPT-INFO IN=lo OUT= MAC=00:00:00:00:00:00:00:00:00:00:00:00:08:00 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)

Fig. 6 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 2 s 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. Fig. 7 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 with either configuration. With each configuration it had difficulties to detect the

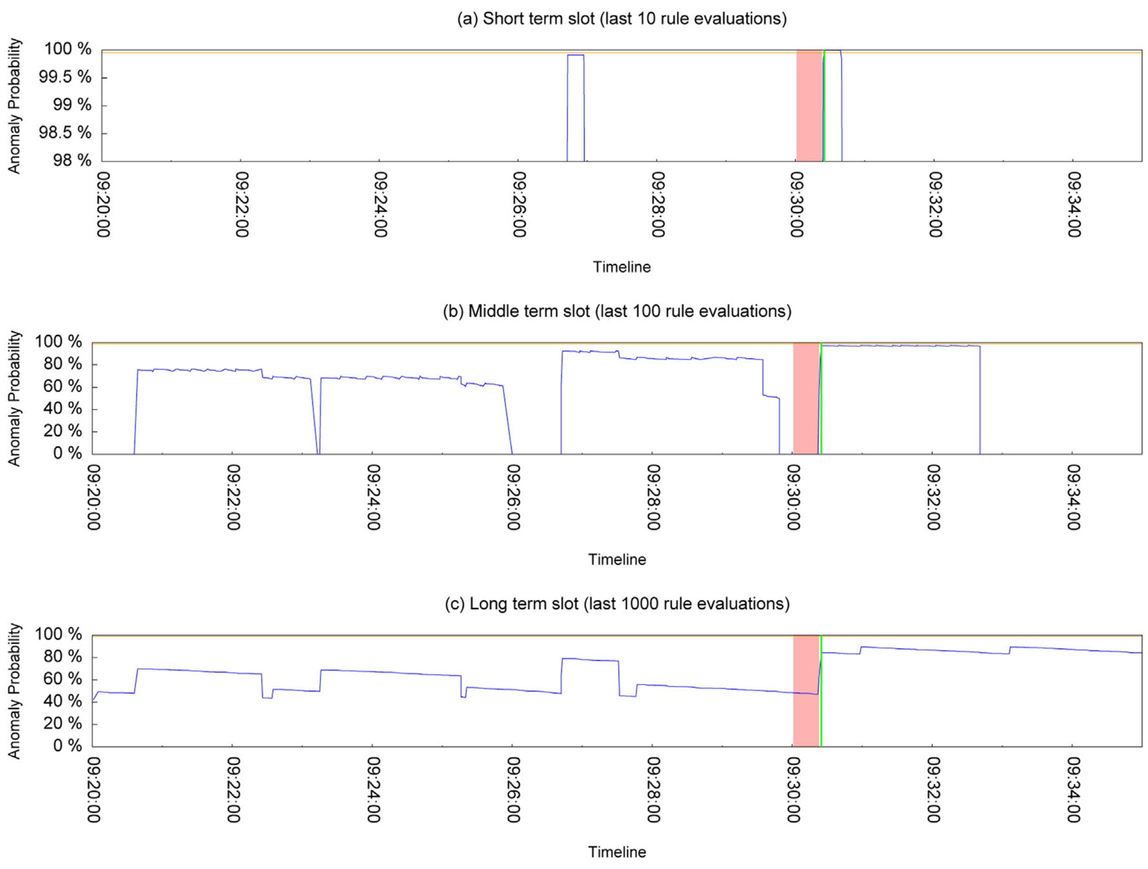


Fig. 6 – Overview of the slot stabilities in rule 286 during the injection slot A1.1.

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.

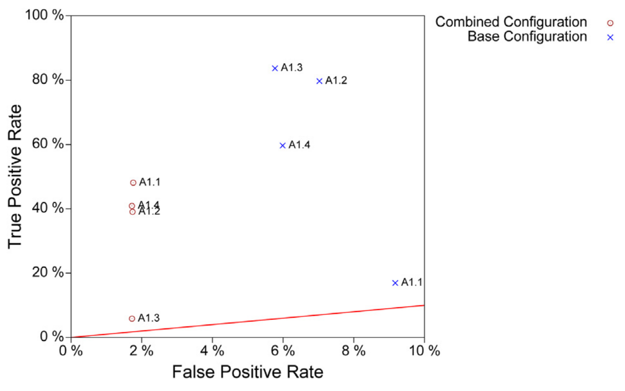


Fig. 7 – ROC scatter for the time slots containing injected database dumps calculated with the base configuration as well as the combined configuration.

6.2.3. *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 min and the duration of the injection differs. The injection times are: 30 s, 1 min, 2 min and 4 min. As in the previous case, a set of rules is selected which are expected to detect the injected anomaly. Two examples are 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. 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 (020405B40402080A012C924D000000000103 0306)
Implied Event: database-0.v3ls1316.d03.arc.local mysql-normal #011#01133179 Connect#011mantis_user_4@service-4.v3ls1316.d03.arc.local on

Fig. 8 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. 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] 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 (020405B40402080A012C924D0000000001030 306)
Implied Event: database-0.v3ls1316.d03.arc.local mysql-normal #011#01133179 Query#011UPDATE mantis_user_ table

Fig. 9 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

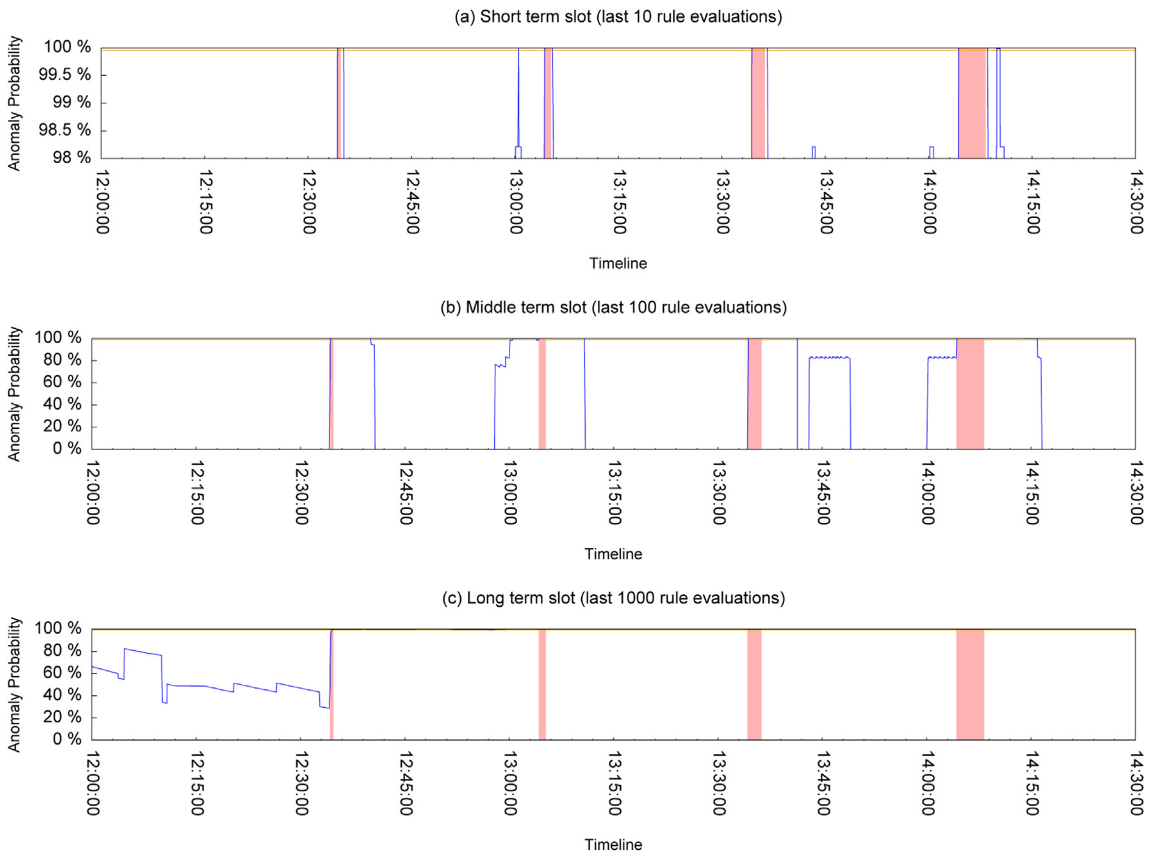


Fig. 8 – Overview of the slot stabilities in rule 708 during all four injection slots of the second anomaly type.

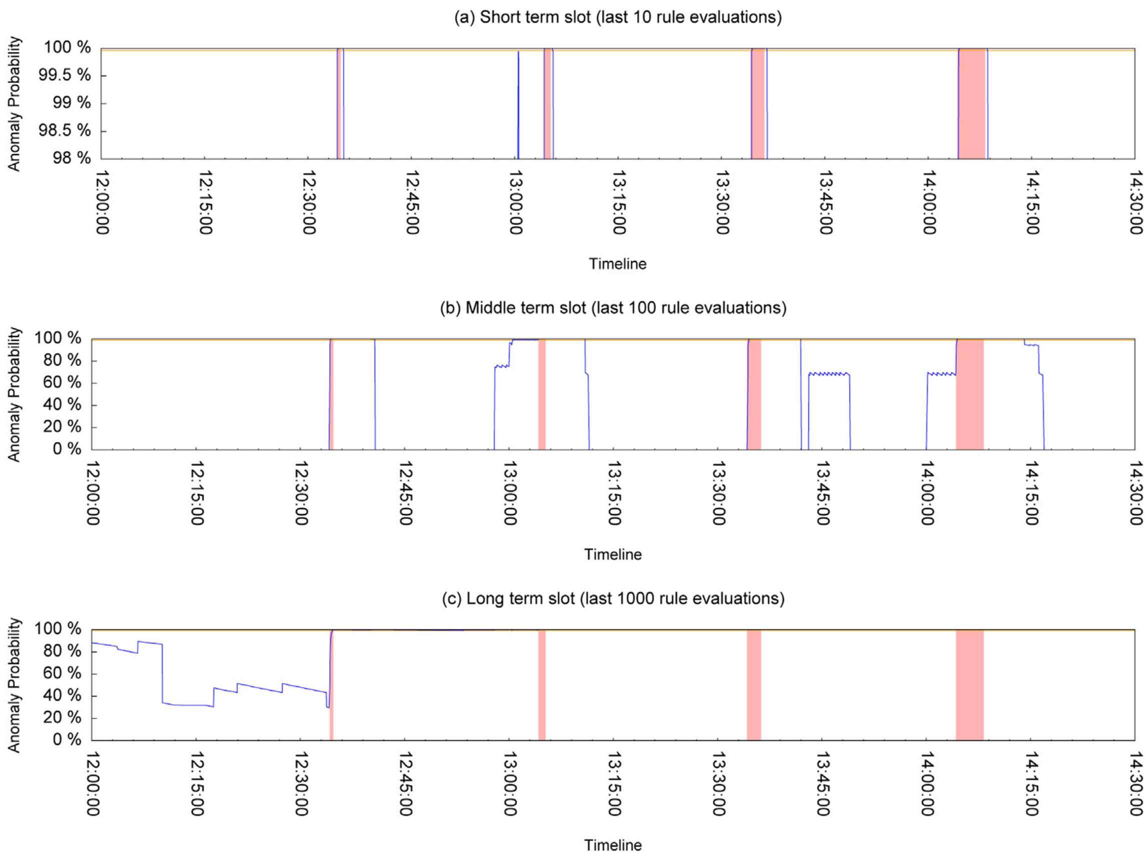


Fig. 9 – Overview of the slot stabilities in rule 804 during all injections of anomalies of the second anomaly type.

rate and the true positive rate of the evaluated time slots. Fig. 10 shows the familiar 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.

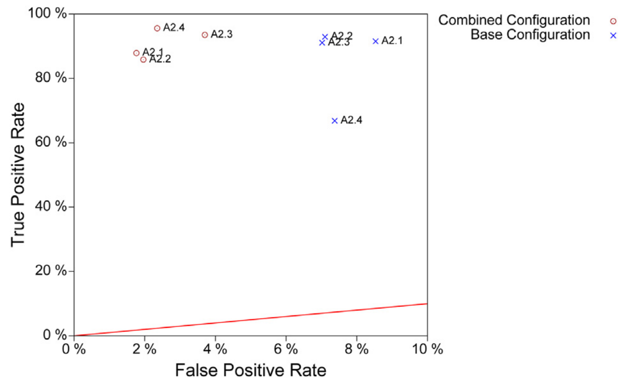


Fig. 10 – ROC scatter for the time slots containing time slots with disabled database logging. The values are calculated with the base configuration as well as the combined configuration.

6.2.4. *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.

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. All injected anomalies where detected while the anomalous behaviour was still ongoing.

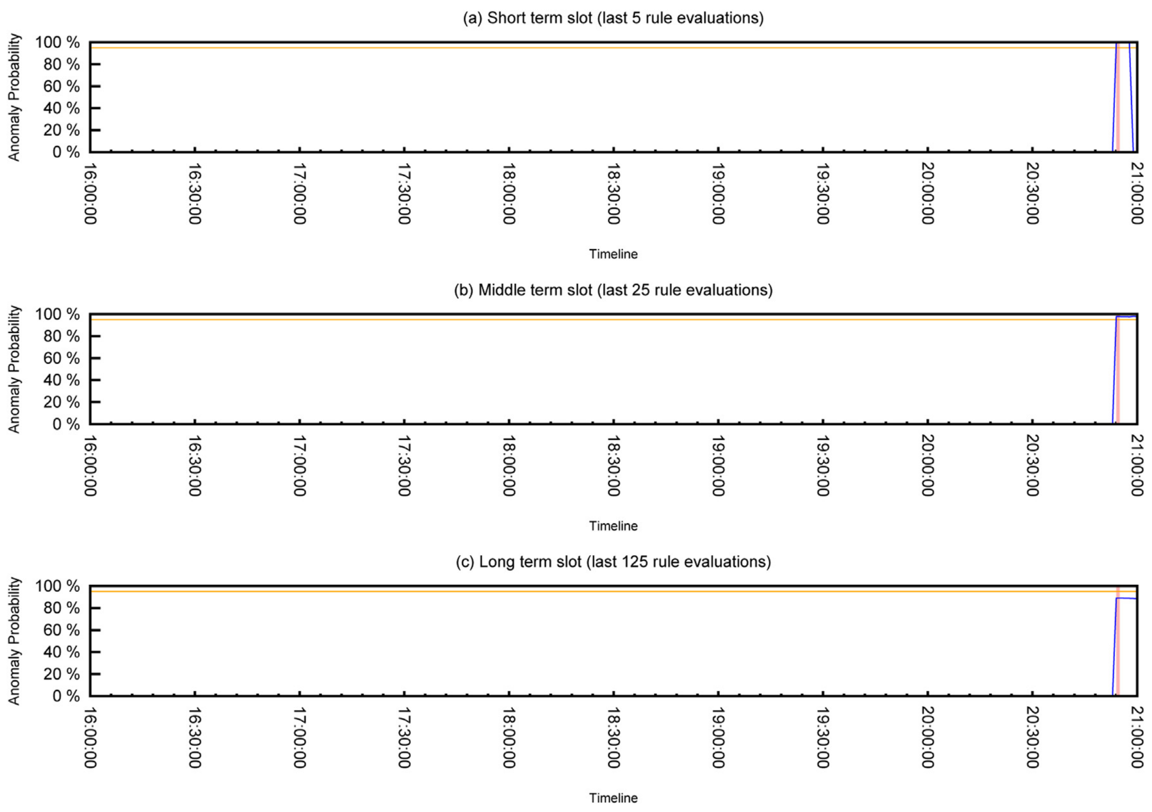


Fig. 11 – Overview of the slot stabilities in rule 169 during the injection of anomalous firewall connections.

6.3. *Anomaly detection in SCADA networks*

This section covers the evaluation of the anomaly detection capabilities in a real world SCADA network. The used dataset was recorded during 1 h on a branch of a real network infrastructure from an Austrian utility provider's SCADA system that is coupled to the corporate LAN. Three different datasources were monitored: a **Firewall** that recorded and filtered incoming connections, a **Switch** that forwarded traffic to and from the SCADA system and the actual **SCADA system** that issued switching commands and provided measurement values on request.

The events triggered in a SCADA system are highly structured and well defined. Therefore the limitations of the proposed anomaly detection approach carry less weight while its strengths are more prominent (namely syntax-independent applicability without high configuration effort). On the other hand the monitored systems create a very limited number of log-events while they are in a normal state. The number of log-lines in the recorded dataset is not sufficient for an evaluation. 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 h of recorded data. This duplication results in a highly periodic dataset. But given the very stable nature of the log-lines this way to extend the dataset is applicable.

The 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 s 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. An analysis of the system model results in a set of rules that state that every accepted database connection is followed by the transmission of measurement values back to the requester. Since the results of all rules in this set are very similar, the results for one rule are considered sufficient as an example (see Fig. 11). The events are considered completely normal up to the injection of the anomalous connections. The anomaly is considered very significant. Even the middle-term slot reaches an abnormal state.

This evaluation showed that 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. We can conclude that the suggested approach performs very well in the limited SCADA dataset. The limitations of the evaluated dataset prohibit a more formal conclusion but the results show that the suggested approach is able to analyse data from a real-world setting in a sample domain.

7. Conclusion

In this work a novel anomaly detection approach that utilizes log-lines produced by various systems and components in ICT networks has been presented. 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. Using a semi-synthetic test-data generation approach, two clean datasets were generated. We started with an evaluation of the system's parameters before different types of anomalies were injected into one dataset and the detection capabilities of the prototype were evaluated. Additionally the approach was tested using a real-world dataset provided from an Austrian utility provider that shows the approach's applicability to non-artificial input data.

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 of syntax and semantics of the log-lines. 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 anomalies that are the consequence of realistic APT attacks (e.g., direct access to database servers, copying large amounts of data). 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 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.

Future work deals with a more intelligent approach for the generation of event classes. The lack of information about similarities of event classes results in redundant hypotheses that might overload 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.

Acknowledgements

This work was partly funded by the Austrian FFG research program KIRAS in course of the project CIIS (840842) and the European Union FP7 project ECOSSIAN (607577).

REFERENCES

Alperovitch D. Revealed: operation shady RAT. McAfee; 2011.
Axelsson S. Intrusion detection systems: A survey and taxonomy. Technical Report. Chalmers University of Technology; 2000.
Barber R. Hackers profiled who are they and what are their motivations? Comput Fraud Secur 2001;2001(2):14–7.
Bartoš V, Žádník M. Network anomaly detection: comparison and real-time issues. Dependable networks and services. Springer; 2012. p. 118–21.
Brewer R. Advanced persistent threats: minimising the damage. Netw Secur 2014;2014(4):5–9.
Caldwell T. Spear-phishing: how to spot and mitigate the menace. Comput Fraud Secur 2013;2013(1):11–6.
Chandola V, Banerjee A, Kumar V. Anomaly detection: a survey. ACM Comput Surv (CSUR) 2009;41(3):15.
Denning DE. An intrusion-detection model. Softw Eng IEEE Trans 1987;2:222–32.
Garca-Teodoro P, Daz-Verdejo J, Maci-Fernndez G, Vzquez E. Anomaly-based network intrusion detection: techniques, systems and challenges. Comput Secur 2009;28(12):18–28.
Ives B, Walsh KR, Schneider H. The domino effect of password reuse. Commun ACM 2004;47(4):75–8.
Kim G, Lee S, Kim S. A novel hybrid intrusion detection method integrating anomaly detection with misuse detection. Expert Syst Appl 2014;41(4, Part 2):1690–700.
Kjaerland M. A taxonomy and comparison of computer security incidents from the commercial and government sectors. Comput Secur 2006;25(7):522–38.
Kraemer-Mbula E, Tang P, Rush H. The cybercrime ecosystem: online innovation in the shadows? Technol Forecast Soc Change 2013;80(3):541–55. Future-Oriented Technology Analysis.
Lee W, Stolfo SJ, Mok KW. A data mining framework for building intrusion detection models. In: Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on. IEEE; 1999. p. 120–32.
Li G, Japkowicz N, Yang L. Anomaly detection via coupled gaussian kernels. In: Advances in Artificial Intelligence. Springer; 2012. p. 343–9.
McAfee Labs and McAfee Foundstone Professional Services. Protecting your critical assets. McAfee; 2010.
Mitchell R, Chen IR. A survey of intrusion detection techniques for cyber-physical systems. ACM Comput Surv 2014;46(4). 55:1–55:29.
O'Neill M. The internet of things: do more devices mean more risks? Comput Fraud Secur 2014;2014(1):16–7.
Patcha A, Park JM. An overview of anomaly detection techniques: existing solutions and latest technological trends. Comput Netw 2007;51(12):3448–70.
Sabahi F, Movaghar A. Intrusion detection: a survey. In: Systems and Networks Communications, 2008. ICSNC'08. 3rd International Conference on. IEEE; 2008. p. 23–6.
Skopik F, Fiedler R. Intrusion detection in distributed systems using finger-printing and massive event correlation. In: 43 Jahrestagung der Gesellschaft fr Informatik eV (GI) (INFORMATIK 2013); 2013.

Skopik F, Friedberg I, Fiedler R. Dealing with advanced persistent threats in smart grid ict networks. In: 5th IEEE Innovative Smart Grid Technologies Conference. IEEE; 2014a.

Skopik F, Settanni G, Fiedler R, Friedberg I. Semi-synthetic data set generation for security software evaluation. In: 12th Annual Conference on Privacy, Security and Trust. IEEE; 2014b.

Sommer R, Paxson V. Outside the closed world: on using machine learning for network intrusion detection. In: Security and Privacy (SP), 2010 IEEE Symposium on; 2010. p. 305–16. http://dx.doi.org/10.1109/SP.2010.25.

Sood AK, Enbody RJ. Crimeware-as-a-servicea survey of commoditized crimeware in the underground market. Int J Crit Infrastructure Prot 2013;6(1):28–38.

Steer J. The gaping hole in our security defences. Comput Fraud Secur 2014;2014(1):17–20.

Tankard C. Advanced persistent threats and how to monitor and deter them. Netw Secur 2011;2011(8):16–9.

Thomson G. Apts: a poorly understood challenge. Netw Secur 2011;2011(11):9–11.

Thottan M, Ji C. Anomaly detection in ip networks. Signal Process IEEE Trans 2003;51(8):2191–204.

von Solms R, van Niekerk J. From information security to cyber security. Comput Secur 2013;38(0):97–102. Cybercrime in the Digital Economy.

Yin J, Zhang G, Chen YQ, Fan XL. Multi-events analysis for anomaly intrusion detection. In: Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on, volume 2. IEEE; 2004. p. 1298–303.

Yu Y. A survey of anomaly intrusion detection techniques. J Comput Sci Coll 2012;28(1):9–17.

Zhang W, Yang Q, Geng Y. A survey of anomaly detection methods in networks. In: Computer Network and Multimedia Technology, 2009. CNMT 2009. International Symposium on. IEEE; 2009a. p. 1–3.

Zhang Yl, Han Zg, Ren Jx. A network anomaly detection method based on relative entropy theory. In: Electronic Commerce and Security, 2009. ISECS'09. Second International Symposium on, vol. 1. IEEE; 2009b. p. 231–5.

Zhao Y, Zheng Z, Wen H. Bayesian statistical inference in machine learning anomaly detection. In: Communications and Intelligence Information Security (ICCIIS), 2010 International Conference on. IEEE; 2010. p. 113–6.

Ivo Friedberg finished his master's studies of Software Engineering & Internet Computing at Vienna University of Technology in 2014 and is currently a researcher at the Austrian Institute of Technology. His research interests lie in intrusion detection, machine learning, and control systems.

Florian Skopik joined AIT in 2011 and is currently working in the research program "IT Security", focusing on applied research of security aspects of distributed systems and service-oriented architectures. Current research interests include the security of critical infrastructures, especially in course of national cyber defence. Before joining AIT, Florian received a bachelor degree in Technical Computer Science 2006, and master degrees in Computer Science Management and Software Engineering and Internet Computing from the Vienna University of Technology in 2007. He was with the Distributed Systems Group at the Vienna University of Technology as a research assistant and post-doctoral research scientist from 2007 to 2011, where he was involved in a number of international research projects. In context of these projects, he also finished his PhD studies. Florian further spent a sabbatical at IBM Research India in Bangalore for several months. He published around 60 scientific conference papers and journal articles, and is member of various conference program committees and editorial boards. Florian is IEEE Senior Member.

Giuseppe Settanni joined AIT in 2013 as a junior scientist and is currently working on national and European applied research projects regarding security in communication and information systems. Before joining AIT, Giuseppe Settanni worked for 2 years at FTW (Telecommunication Research Center in Vienna), as a communication network researcher, on the development of a network-based anomaly detection tool in the context of DEMONS European Project. His current research interests include security of critical infrastructures, information sharing and anomaly detection in national cyber defence. Giuseppe Settanni will contribute to this project by working on design, development and evaluation of the system components.

Roman Fiedler is Scientist at the AIT Austrian Insititute of Technology and runs projects in the areas of telehealth and ict security. Roman has got a decade of experience in network security and operations. He finished his Master studies in the domain of biochemistry at the University of Technology Graz.