Semantical rule-based false positive detection for IDS

Salih, R.

Award date:
2020

EINDHOVEN UNIVERSITY OF TECHNOLOGY

MASTER'S THESIS

Semantical rule-based false positive detection for IDS

Author:
Rang SALIH

Supervisor TU/e:
Dr. Jerry den HARTOG

Supervisor Northwave:
Edwin SMULDERS

A thesis submitted in fulfillment of the requirements for the degree of Information Security Technology

in the

Security Group TU/e
Department of Mathematics and Computer Science

January 30, 2020

Abstract

Security Analysts working in a Security Operations Center (SOC) are responsible for triaging incoming alarms coming from Intrusion Detection Systems (IDS) as accurately and quickly as possible. A general problem that SOCs are dealing with is that a large amount of the incoming alarms are false positives (FPs). To minimize the time and energy wasted on these non malicious events, the analyst want to be able to quickly and easily identify these false alarms. However, currently the analyst needs to look into large bulks of low-level data, which can be difficult to analyse, to correctly determine whether the behavior is malicious or not. Here we aim at reducing this effort of the analyst by presenting information at a higher level of abstraction and automating the identification of common false alarms. To this end we extract semantical concepts from the lower level data and allow defining of rules, expressed in terms of such concepts, that identify common false alarms. We evaluate this approach by considering two scenarios that capture common sources of false positives and show that rules can be defined that are correct (capture the decision of the analyst) and are applicable in most cases.

Acknowledgements

Firstly, I would like to thank my supervisor Jerry den Hartog for guiding me through the whole process, and especially with shaping the framework proposed in this thesis. I also want to thank my supervisor Edwin Smulders and other colleagues at Northwave who helped me with performing the evaluation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter gives an introduction to the problem and research topic of this thesis.

1.1 Background

Cyberattacks are on the rise recent years and the factors that contribute to this growth are digitalization, vulnerabilities of systems and dissemination of knowledge to perform cyberattacks. Therefore, it is important for governments and organizations to detect any event on a network that looks malicious and respond to it, but this is not easy.

Monitoring of networks is carried out through different Intrusion Detection System (IDS) looking for suspicious behavior. The logs of multiple IDSs are collected centrally using a Security Information and Event Management (SIEM) system. A SIEM system is used in a Security Operations Center (SOC) to provide real-time analysis of security events generated in networks by security controls. SOC analysts need to determine whether the incoming alarm is an attack and what to do to respond properly. Alarms that are actual attacks are called *true positives* (TP). Attacks are only captured when IDSs and SIEM rules are correctly configured. However, rules that only capture attacks are difficult to create and therefore a lot of false alarms will also be captured. These false alarms are normal traffic that is flagged as intrusion, also called *false positives* (FP). IDSs can also fail to detect malicious network traffic and are referred to as *false negatives* (FN). Lastly, there is also normal traffic passing the IDSs that is not triggering alarms, also known as *true negatives* (TN). There should be a natural trade-off between detecting "all" attacks, with the expense of raising alarms too often (thus having a large amount of FPs), and missing attacks, having high FNs and thus having less FPs. The reason why SOCs are dealing with many false alarms is because the cost of missing an attack (TP) is higher than the cost of a false alarm (FP), but this has negative impact on the workload of the analysts.

1.2 Problem description and existing solutions

As mentioned before, analysts in SOCs have to deal with the problem that lots of alarms being collected are FPs. The number of FPs should be minimized to reduce the workload of a SOC analyst. Another problem when analyzing an alarm is that an alerting system like a SIEM does not always contain the data that is needed to

make a proper judgement as SOC analyst. This crucial data is somewhere in log files generated by one or multiple IDSs or a Knowledge Base (KB) and are both used as sources of data for incoming alarms. The SOC analyst will in this case manually search for evidence by looking into related alarms, IDSs logs and other information sources. This process of searching for evidence wastes a lot of time and makes monitoring costly. The SOC analyst often has to deal with alarm where the logs are of a large size and difficult to analyze. SIEM systems are there to aggregate logs of different sources and correlate these and then generate an alarm. However, SIEM system often lack context and actionable data. The lack of alarm context has in most cases to do with the lack of functionalities and restrictions to the creation of rules within most SIEM systems. Also, the level of semantics that is given to the data is mostly not high enough for the SOC analyst to make a decision, and that is why the SOC analyst has to dig deeper into the sources that are available to get a clearer picture of what actually happened. Even the correlated logs can be difficult to interpret. A framework will be introduced for the problems mentioned to easily identify common false alarm patterns by conceptualizing security data.

1.3 Research questions and methods

The above problem description leads to the following research question:

Can conceptualizing security data reduce the effort needed by a SOC analyst to distinguish attacks from false alarms?

The following sub-questions, provided in Table 1.1, will draw a proper path to answer the main research question:

TABLE 1.1: Sub-questions with corresponding research methods and chapters

Sub-questions	Research method	Chapter
What are IDSs and SIEM and how do they work? What are the existing false alarm minimization techniques for IDSs?	Literature review	2
How are alarms handled by analysts in a SOC? What kind of issues are SOC analysts facing?	Literature review	3
How can semantics be given to security data? How can concepts be derived from IDSs logs?	Development	4
What is the time complexity of pattern matching? How can the process of conceptualization of alarms be automated?	Literature review and design Development	
How well does the suggested approach perform in terms of detection rate?	Evaluation	5

1.4 Proposed solution

The goal of this research is firstly to address the shortcomings explained in section 1.2, and the consequences it has for the work of a SOC analyst. After this we introduce the notion of *concepts* within the framework that is provided. As mentioned, the lack of context and semantics given to data within IDSs and SIEM systems, pushes us to label low-level data. By labeling data with higher level concept we make it possible to add semantics to any kind of low-level data. The goal of the framework is to automate the process of conceptualizing low-level data by looking at patterns of common false alarms. Rules will be used to identify common patterns of false alarms. The approach will be evaluated for the implemented rules for two use cases that identify common sources of false alarms.

1.5 Contribution

The idea behind this work is that giving semantic meaning to data saves (mental) effort for an analyst in analyzing an alarm and allows specification of rules at a higher

level of abstraction, making them more understandable, (re)usable and maintainable. This work establishes feasibility of specifying and checking such conceptual rules. The main contribution of this work is thus demonstrating that rules expressed in concepts can effectively be used and can automate a large amount of work currently done by SOC analysts. This is done through a general framework, introduced in chapter 4, which can also be applied to other types of large datasets coming from different sources. This work has also helped the company to think about their data representation in general.

1.6 Thesis structure

- **Chapter 2**: Theory of IDS and SIEM
  This chapter provides background information about the history of IDSs, what an IDS is, existing types of IDS, which IDS detection methods exist and how *rules* are defined. After this an explanation is given about SIEM systems, what their main functionalities are, examples of SIEM solutions and limitations.
- **Chapter 3**: Alarm handling at the SOC
  Chapter 3 presents the SOC staff, and how alarms are being handled by a SOC analyst and the issues they are facing. After this, the chapter gives a general overview of elements within the SOC and how they are interlinked. Finally the chapter ends with some relevant false alarm minimization methods.
- **Chapter 4**: Framework for alarm handling
  Chapter 4 focuses on the framework, by giving the definitions used in the framework (e.g. *concept*, *pattern*, *CAP* and *CAT*), steps within the framework, its rough time complexity for pattern matching, introduction to the use cases to explain rules.
- **Chapter 5**: Experimental setup and evaluation
  Chapter 5 provides the experimental setup, preparation steps, implementations for the use cases, overview of the datasets, and the results.
- **Chapter 6**: Conclusions and future research direction
  Finally, we end the thesis by giving our conclusions and directions towards future work.

Chapter 2

IDS and SIEM system

This chapter will illustrate the history of intrusion detection and how it is developed and categorized in approach and type.

2.1 Intrusion Detection System

An Intrusion detection system (IDS) monitors passively a network or a system for malicious activity or policy violations. The approach an IDS uses to detect intrusions are *knowledge-based* or *behavior-based* and is later explained in more detail in this chapter. Any activity that has signs of intrusion or violation of policies are either send to an administrator or can be collected centrally using a SIEM system. A SIEM system can aggregate and correlate data from multiple sources (e.g. IDSs, servers, domain controllers, databases etc.) and send out alerts based on automated analysis of the correlated data that match a rule set, described in section 2.2 and chapter 3. Before we dive further into IDSs and SIEM systems, we will review a number of historical developments of IDS.

2.1.1 History of IDS

It was in 1972 when the United States Air Force (USAF) outlined that they had "become increasingly aware of computer security problems" [1]. This report written by James P. Anderson mentioned that the main problem of computer security was due to the fact that there was a growing demand to have shared use of computer systems containing information of different classification levels. This problem was not present before that time because it was usual that a computer was serving one user at a time and therefore information was already accessible for authorized users. Another major development that gave birth to new security problems was that interlinkage of related computer systems into complex networks was demanded. An example is the implementation of the Military Aircraft Command Integrated Management System (MACIMS) into the Worldwide Military Command and Control System (WWMCCS) computers. This expansion will increase the probability to have a untrusted computer that wants to gain classified data within the network of the WWMCCS and by this the entire network can be compromised.

In 1980 the same James P. Anderson published the seminal work about network security monitoring [2]. James P. Anderson is known as the founding father of intrusion detection systems with his report, where he showed ways to improve computer security auditing and surveillance at customers sites. He emphasises that valuable

information is processed within audit trails and can be used to detect misuse of data or systems. The developed system tries to find anomalies in user behavior. Anderson knew that to find anomalies he first needed to figure out which threats and attacks exist to design an IDS. The threat model that he provides consists of three different threats: external penetration, internal penetration and misfeasance. The proposed model is based on whether or not a person is authorized to use a computer and whether or not a user of the computer is authorized to use particular data or program resource. The notion of security policies was here introduced which indicates what secure means for a particular network or system. The attacks that were taken into account were a signature list of known attacks.

A few years later, in 1985, a report of Denning and Neumann [6] commissioned by the government, built upon the idea of detecting intrusion in a network or on a computer by focusing on how a user behaves. The so-called Intrusion Detection Expert System (IDES) was developed by Stanford Research Institute International (SRI), which is a nonprofit scientific research institute that was contracted by the Navy's Space and Naval Warfare Systems Command (SPAWAR). The IDES monitors abnormal patterns in system usage to detect security violations, e.g. outsiders that try to break in and by insiders that misuse their privileges. This abnormal behavior reveals that there are some characteristics of normal behavior and is therefore difficult to define. Anderson called this type the masquerader [2], which is an internal user; either an external intruder that came through the access controls or an employee that wants to exploit another legitimate user id and password. He also indicates that to distinguish the masquerader from the legitimate user you need to look at the "extra" usage of a system based on the following properties: time usage, frequency of usage, volume of data reference and patterns of reference to programs or data. This example of security violation is taken into account by IDES. By creating statistical profiles of normal behavior, as a baseline, that characterize the behavior of users regarding data and programs. The monitoring is done on standard operations on a target system, e.g. logins, program executions, device and file accesses. The whole mechanism completely depends on the created profiles and has no knowledge of the system's vulnerabilities or security controls. The advantage of the system, with the statistical profiles approach, is that it is system independent and is capable of detection intrusions when unknown vulnerabilities are used. New activities within systems will lead to new abnormal behavior and will make updating of profiles essential by the use of rules. The rules will capture new information of normal behavior with the support of statistical tests. A problem mentioned by Denning and Neumann and still present is that modifying one parameter to increase the detection rate will potentially result in a higher false alarm rate. The IDES is the first host-based IDS (HIDS) that is anomaly-based.

The first network-based IDS (NIDS) was developed in 1990 by Heberlein et al. and they called it the Network Security Monitor (NSM) [8]. Here the host-based IDS made a shift to network-based IDS by looking at network traffic in a LAN instead of examining audit trails of a host computer. The following reasons explain why Heberlein et al. have chosen to differ from previous developed IDSs:

- Many IDSs have to deal with different formats of user data from different platforms on a single target machine, while network traffic make use of standard protocols to communicate e.g. TCP/IP, telnet, ftp etc. This makes it possible for the NSM system to monitor a network of heterogeneous hosts and operating systems without having to normalize the different audit trail formats.

- The NSM system has direct access right after network traffic data is transmitted on the network, but with previously seen IDSs the analysis is not available (directly) on every system. Instead, it is transferred to a separate host for analysis.
- Audit trails were vulnerable; incidents showed that intruders were capable to turn off audit daemons or modify the audit trail. This resulted in undetected intrusions and removal of actions with responsible users. The NSM was an invisible sensor that passively listened to the network and was therefore not possible to turn off or modify gained network data.
- The performance will not degrade while monitoring with the NSM system, because it does not make use of resources on the monitored host.
- Lastly, most sophisticated attacks are done remotely and can therefore be detected via a network before reaching a host computer.

The same detection methods used in the past are reused nowadays. The next subchapter will describe those detection methods used in IDSs. It is also notable to say that the issues faced back then with IDSs (also detection technology before having IDSs) are still the same issues we are facing today.

2.1.2 Main detection methods

There are two main detection methods [11]:

- **Knowledge-based**
  This detection method makes use of already gained knowledge about attacks and vulnerabilities. Any activity that is similar to the listed signatures is labelled as intrusion with the use of rules. Activities that are not allowed are in this way defined and this is also called *blacklisting*. A simple example of a signature of an attack could be that if there are more than five login attempts within one minute from an unknown IP address then a conclusion can be that a brute force password attack is taken place by an intruder. A rule here can be that "*if* signature X takes place *then* then send out an alert". This methods is also called *signature-based* and *misuse-based*.

  The ***advantage*** of a knowledge-based IDS is that the false positives are relatively low, since it only detect known attacks. Therefore, the alarms that will be generated will be understandable since it is based on the knowledge of the signature creator and will then be easy to track down the root cause of an alarm. The ***disadvantage*** is that if the activity that is taken place on a system or network is not taken into account within the signature and/or rule then this "new" attack will not be detected. This means that the attack is not profiled and thus unknown. This obliges you that the database with known signatures and rules need to be updated constantly when new attacks or threats are known. Therefore, knowledge-based detection is completely useless against zero-day attacks.

- **Behavioral-based**
  This method first classifies data of network traffic or system logs as normal behavior. Any activity that deviates from the profiles of normal or expected

behavior is labelled as intrusion. Allowed actions are here defined also known as *whitelisting*.

This method is also divided in two sub approaches [7]:

- *Anomaly-based*: Where the model learns what normal behavior is, here the model is based on statistical behavior. Algorithms are used to profile normal behavior. The ***advantage*** of this approach is that unknown threats are being detected, but unfortunately most of the time without proper description to quickly understand the context. The ***disadvantage*** of anomaly-based is that the normal profiles need to be updated once in a while, because the normal behavior will also change.
- *Specification-based*: Where the model specifies what normal behavior is, here the method combines both knowledge and behavioral-based. It specifies manually what behavior is allowed and expected based on known normal behavior. It is also similar to behavior-based because anomalies are produced from deviation on the manually specified normal behavior. The difference with anomaly-based is that the normal profiles are not learned with the support of algorithms.

  The ***advantage*** is that unknown attacks are being detected, while a low false alarm rate is being preserved.

  A ***disadvantage*** is that it is time consuming to maintain the low false positive rate with detailed specifications.

2.1.3 Source of data

The mentioned detection methods can be performed on with an IDS on host-level or network-level. The two primary types IDSs are Network-based Intrusion Detection Systems (NIDS) and Host-based Intrusion Detection Systems (HIDS).

A **NIDS** monitors packet traffic at specific points on a network with the help of a number of sensors. The NIDS examines the traffic packets to detect intrusion patterns. The sensors are placed in the network where it can analyze traffic going into and out of a particular network segment. NIDS can be both knowledge-based and behavior-based. The monitoring of network traffic is mostly done on a mirrored port or inline. An inline sensor is placed into a network segment where all traffic must pass through the sensor. While a passive sensor is a simple copy that can be analyzed in the background; meaning no traffic passes through the sensor.

Most used locations to place a NIDS sensor:

- Outside an external firewall
- Behind each external firewall in the DMZ
- On major network backbones
- On critical subnets

A **HIDS** monitors the activities taking place on a single host, e.g. system logs, running processes, application activity, file access, and configurations changes. The HIDS can be seen as an agent that monitors the state of the operating system and checks if there are any security policy violations.

Figure Figure 2.1 shows clearly where the NIDS and HIDS are situated in the network.

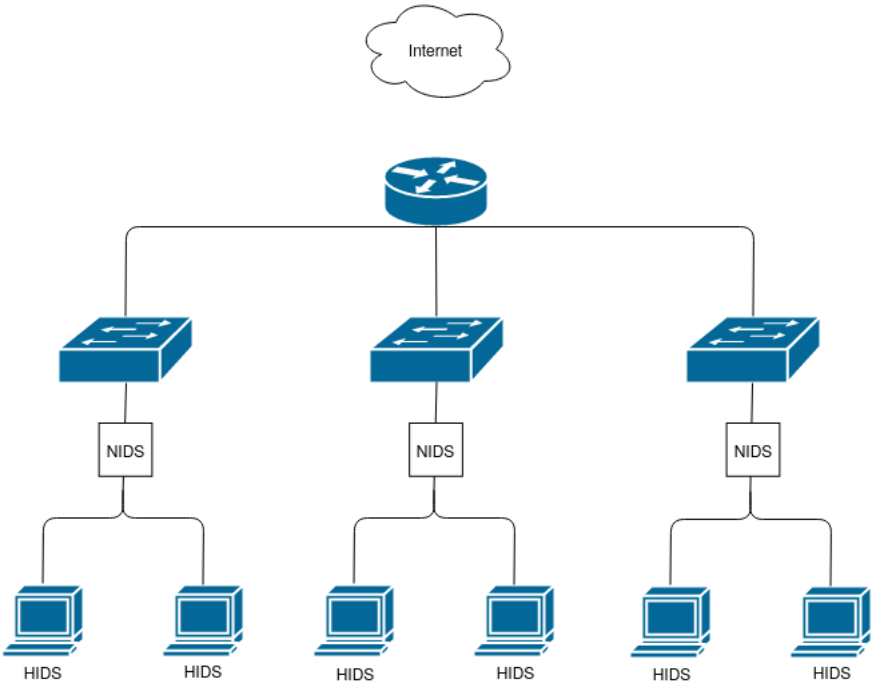

FIGURE 2.1: NIDS - HIDS topology

2.1.4 Rules

A typical and well-known NIDS is Snort [13]. Snort makes use of signatures and rules and can therefore be classified as a knowledge-based NIDS that only detects known intrusions. This is also the basis of how most IDS are being built up; even with behavior-based you need rules to check if it matches normal behavior. Snort rules are easy to write, but its simplicity makes it hard to write rules that cover something specific.

Snort consists of the following components [4]:

- packet decoder
- preprocessors
- detection engine
- logging and alerting systems
- output modules

The detection component consists of rules; if malicious behavior is detected with the predefined rules then an alert is being generated to notify the user. Suricata [15] is also a NIDS that makes use of the same rule syntax as Snort.

A rule for these two NIDSs can be divided into three segments: action, header and option.

Action: tells Snort what kind of action it should perform when detecting a packet that matches the rule description. The most common actions are: alert, log, pass, drop, reject, sdrop. Two less used actions are: activate and dynamic.

The **header** is the part that is at least required for a legitimate rule. The next fields are taken into account:

- Protocol
- Source address
- Source port
- Direction
- Destination address
- Destination port

The **options** makes use of two field: keyword (e.g. msg) and argument (extra information behind the keyword) see Figure 2.2.

The rule **options** consists of four major categories:

- General: extra information about the rule
- Payload: find data inside packet payload
- Non-payload: find non-payload data
- Post-detection: rule specific triggers, happen after a rule has been fired

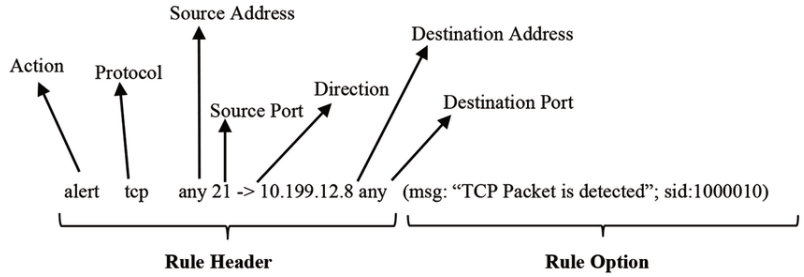

FIGURE 2.2: Components of a Snort/Suricata rule: *taken from* [10]

2.2 SIEM

A Security Information and Event Management (SIEM) system [3] is a software product and service. It provides real-time analysis of alerts generated by applications and network hardware, for example an IDS. There are three main capabilities that every SIEM system contains: data aggregation, correlation and alerting. Data aggregation is collecting security data from different sources, such as servers, IDS and firewalls. Some SIEM systems can also use data from open source intelligence tools. Correlation makes it possible to link events and common attributes into small collections. Different techniques can be use to integrate sources, which can provide information. Alerting stands for the automated analysis of correlated events. A SIEM system looks for known patterns, in order to identify attacks and other

security-related events. For example a SIEM system can correlate HTTP proxy and antivirus product logs, to detect malware downloaded to endpoint devices. However, it can be difficult to identify patterns and perform correlations within large companies due to the number of devices and employees that are generating network traffic.

2.3 False alarm minimization

The goal of security engineers is to tune all rules and control in a way that the FP and FN will stay as low as possible. This is in practice difficult. If a lot of rules are covered in the IDS then a lot of FPs will be generated. However, you know that you increased the chance to detect an attack, therefore your FNs will be lower when making use of a lot of rules. In general, in the beginning when setting up IDS and SIEM systems, a lot of rules are covered and after a while alarms will be suppressed within the SIEM system. In a lot of cases within the rules of IDS it is very difficult to mitigate FPs. Alarms that are triggered because of a specific rule can in some cases be false positives and in some cases attacks. This is why you cannot easily delete used rules. The cyber analyst will in this case perform some actions to investigate whether the alarm is actually an attack or a false positive.

False alarms are time consuming and costly and that is why a lot of research is done to look at techniques to minimize these false alarms. Figure 2.3 show rich number of different techniques to minimize false alarms.

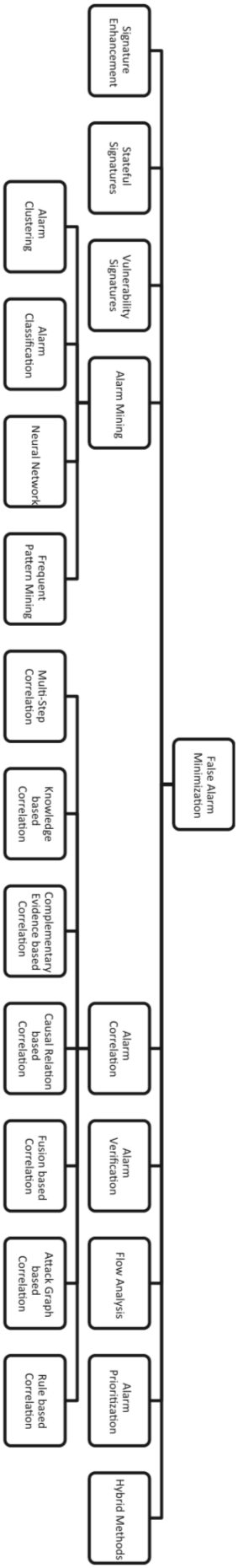

FIGURE 2.3: False alarm minimization taxonomy: *taken from* [9]

The top technique layers, of Figure 2.3, are described below to introduce general techniques to minimize false alarms.

Signature Enhancement: Enhance signature with context information of history or network information. Network context information is added to the signatures, which makes each signature stand alone. The context information is added using regular expressions. The extra information could for example be the OS used by a destination host. By this means the IDS can indicate whether a specific attack will have effect on the target system.

Stateful Signatures: IDS analyze individual network packets, by nature they will fail to detect an attack that span multiple packets. Therefore, the IDS needs to know what has appeared in the previous packet. This previous packet is called the state of the network. A type of IDS that work this way are the so-called "stateful IDS" with the use of stateful signatures. The stateful signatures only analyze relevant parts of traffic instead of the whole traffic.

Vulnerability Signatures: The most common IDS techniques work by string matching or regular expression matching. More advanced attacks can stay undetected with these techniques, these attacks are metamorphic and polymorphic attacks. To detect these advanced attacks, semantics is used in the form of vulnerability signatures. This is done by adding rich application semantics and protocol awareness. The generated signatures consist of characteristics of a vulnerability that covers exploits of that vulnerability. The IDS will then have the functionality to filter out the traffic exploiting a vulnerability.

Alarm Mining: Normally alarms are generated based on attributes like IP addresses, port numbers and protocols. Data mining techniques use attributes like these to mine alarms with the goal to classify them as TP or FP. Most of the used techniques are: *clustering*, *classification*, *neural network* and *frequent pattern mining* models.

Alarm Correlation: This techniques correlates data of alarms of one or multiple IDS and of different types (NIDS/HIDS) to generate more context to alarms. The added value here is that extra information can help the analyst to identify the root cause more easily. The different type of correlations are also divided into details correlation techniques, see the paper of Hubbali [9]. To correlate alarms from different sources the following steps need to be taken, see Figure 2.4:

1. *Normalization*: transcode all data to same format.
2. *Clustering*: group similar alarms: caused by same event, common vulnerability, TCP session and attribute matching (exact or inexact).
3. *Correlation*: analyze the clusters generated and merge related ones.
4. *Intention Recognition*: identify steps of attacker.
5. *Report to Human Expert*: collect and represent data of the attack scenario to the expert.

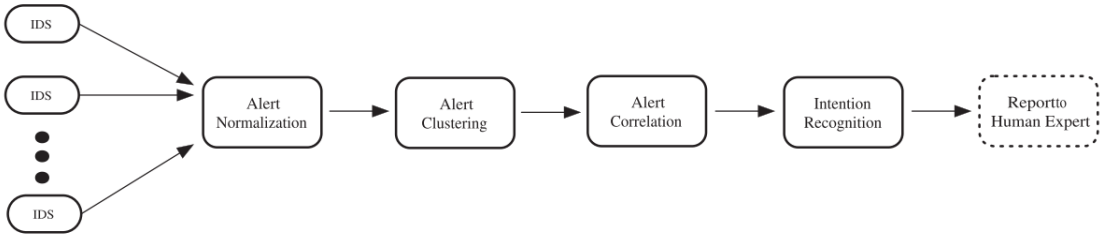

FIGURE 2.4: Steps to correlate alarm data: *taken from* [9]

Alarm Verification: verify if the attack has impact on the targeted network. Verification can be done either *active* or *passive*. When an attack is verified online it can be considered as active verification, if the attacks are verified against a database of successful attacks then it is called a passive verification.

Flow Analysis: an analysis of a set of alarms generated under normal operating environments, but also in abnormal situations. This due to the fact that some IDS generates alarms, even under normal operations. Repeated (false) alarms indicate some pattern for the analyst. Aggregating alarms which are not symptoms of attacks, and come often, can create a shortened view for the SOC analyst.

Alarm Prioritization: alarm prioritization consists of techniques that rate the alarms based on post assessment or an evaluation. The ranking depends on different aspects, for example target network topology, IDS history or placement of an IDS in the network. Alarm prioritization can lead to a reduced number of alarms which needs attention of the administrator.

Hybrid Methods: mixing the best of different filtering based schemes and data mining schemes in order to reduce false alarms. With a good mix it is possible to block attacks at early stage as few of these include IPS capabilities too.

Chapter 3

Alarm handling by the analyst at a SOC

This chapter will address how a SOC currently typically works; we explain how information from the IDS is combined and processed to find and react to (possible) intrusions by the SOC analyst and define related terminology used in this thesis.

3.1 Security Operations Center

A Security Operations Center (SOC) is a facility where SOC analysts, SOC engineers and SOC managers are making sure to monitor, assess and defend: web sites, applications, databases, data centers, servers, networks and endpoints. Any activity that indicates unusual behavior should be captured in the SOC by its tools used and be collected in an alarm. The tools used to capture such activity are IDSs and SIEM systems, as explained in chapter 2.

The alarms in a SOC are handled by SOC analysts. The SOC analysts are divided into three levels: tier 1 analysts, tier 2 analysts and tier 3 analysts. The tier 1 analysts are the ones that start the triage process to identify the seriousness of an alarm and what the reason why the alarm is triggered. In case the tier 1 analyst cannot find the right information to handle the alarm, a tier 2 analyst can be informed about the alarm. The tier 2 analyst is a more experienced SOC analyst that know what to do to gather more information and evidence to make a correct decision. The tier 2 can inform the tier 3 in case there is a serious threat that a certain attack can occur again in the future. The tier 3 analyst can, alone or with more analysts, start with threat hunting. Threat hunting is an active cyber defence activity which in contrast to traditional threat measures, such as firewalls, IDS, SIEM systems, deals with investigation of evidence-based data before there has been a warning of a potential threat. The tier 3 analysts are often also seen as the SOC team lead. The SOC manager is the one that coordinates tasks within the teams of both engineers and analysts, which is a more managing role with hands-on skills to understand what needs to be done. The task for the SOC engineers is in first place to implement the sensors at the client's network to receive network data and make monitoring possible. To make sure that all the tooling for monitoring are kept up and running, SOC engineers need to be ready to intervene and resolve the issues. Their main task is to make sure that the sensors (IDSs) are working properly to receive the logs, alarms and all other necessary data for the SOC analyst to handle alarms.

The Serice-Level Agreement (SLA) [5] is a commitment between a service provider and a client. In the security and SOC domain as, the party that offers network monitoring for the client, should stick to the established agreements in the SLA. One specific subject is the time a SOC analyst has to handle an alarm, also known as the *SLA Due Time*. The SLA due date of an alarm is based on the severity level of an alarm, also known as *risk level*. In general cybersecurity companies and most tools make use of three levels for the *risk level* of an alarm, namely: low, medium and high.

- **Low**: minimal impact with the potential for significant or severe impact on operations.
- **Medium**: significant impact, or the potential to have a severe impact on operations.
- **High**: severe impact on operations.

The alarms with a risk level *high* have a earlier SLA due date and the analyst thus has less time to handle the alarm. Alarms with a risk level high have a great chance, depending on the use case, of being a policy violation or is certain a rogue attack that can harm the intellectual property of the client. Obviously it is easier for the SOC analysts to meet the SLA due date when the alarm queue as small as possible, meaning a minimum number of false alarms in the queue.

In case the SOC analysts found a TP and the attacker already made some damage and the monitored party needs long investigation then a Computer Emergency Response Team (CERT) can be contacted to get support. The CERT can deal with the situation and make sure that the company can be put back in business. After this the CERT can also do forensic investigation to exactly figure out how the attacker got access to do damage.

3.2 Alarm handling and data sources

The Security Operations Center (SOC) is where monitoring and alarm handling is taking place within cybersecurity companies. In general, there are two main teams: the analysts and the engineers. The SOC analysts make sure all incoming alarms are being handled properly. The SOC engineers make sure that the all machines are up and running, to make sure monitoring works properly, and that new clients are implemented correctly for monitoring. This chapter will be more focused on the course of action of the SOC analysts and problems they are facing.

In most SOCs the alarms that are coming in are collected in a ticketing system (e.g. Jira). The ticketing system is the central place where the SOC analysts can have an overview of the alarms received, and from there they can easily reach the relevant SIEM content and logging servers. Figure 3.1 shows that firstly the tier-1 analyst will get notifications about a *new alarm*. The tier-1 will start the triaging and can end with one of the following three conclusions: (1) the alarm does not show malicious behavior and is a false alarm **(FP)** and can be *closed*, (2) the alarm does show suspicious behavior but it is not exactly clear whether it is malicious, to make sure what happened, the tier-2 will be notified, and it will be *escalated* as **potential attack** to the client, (3) the alarm does clearly show malicious behavior (or policy violation) and should be *escalated* to the corresponding client as an attack **(TP)**.

In case of an alarm being classified as **FP**, then it means that the corresponding network traffic is wrongly seen as an attack. In this case it mostly means that IDS rules are wrongly set. When an alarm is classified as **TP**, then it means that the detected (malicious) traffic is correctly classified as an attack. The third option, when suspicious network traffic is seen, but depending on other information that is not in the hands of analysts which can be asked at the monitored party to complete the picture. In that case, the alarm is classified as **potential attack** in first place. As mentioned the tier-1 analysts are the ones that start the triage when receiving the alarm from the ticketing system. However, if they cannot handle an alarm individually, then a *tier-2* can be *informed* about the alarm and the tier-1 will get support. The explained workflow of a tier-1 analyst can be seen in Figure 3.1.

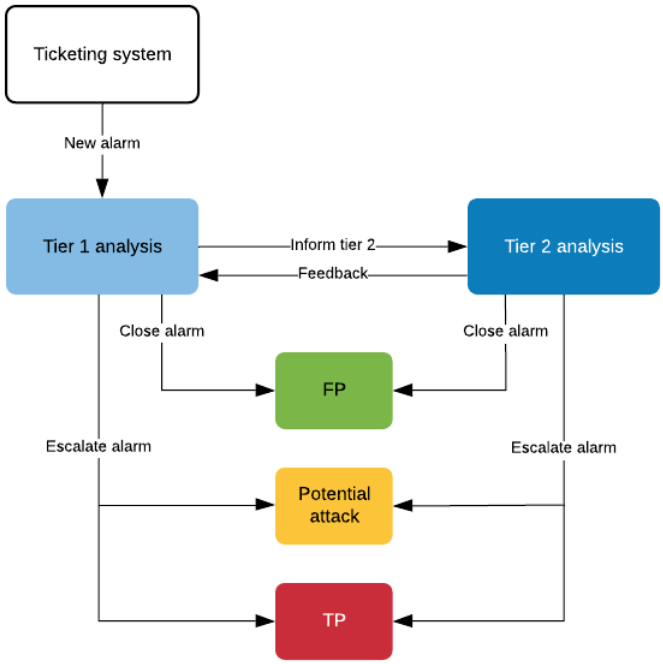

FIGURE 3.1: Tier-1 workflow

Below a list of definitions with short description is presented:

SOC

A Security Operations Center (SOC) is a dedicated place where information systems, such as applications, databases, servers, networks and endpoints are monitored, analyzed and defended by security specialists.

IDS

IDS is a device or software application (e.g. snort) that monitors a network for malicious activity or policy violations and can also be used to record long-term network behavior. Any malicious activity or violation is typically reported or collected centrally using a Security Information and Event Management (SIEM) system.

Log

File produced by an IDS with consisting of sequence of log entries.

Log entry

A *log entry* is a tuple of n *log entry fields*, where n is a positive integer. An entry is basically a line within a log file that represents a detected network action. We assume the field *timestamp* is always present in a log entry.

Log entry field

A *log entry field* (or 'feature') is a single field within the log entry. The field can be a string or an integer and both can also be domain specific.

A specific field in the log entry, e.g. timestamp, source IP, destination IP, authentication success (specific value domain), connection-state (specific value domain) etc. The value domain differs per field, depending on what feature it is capturing for a specific log file. In Table 3.1 a few examples can be seen of fields with corresponding type and description.

TABLE 3.1: Log entry fields examples

Field name	Type	Description
ts	value	Timestamp
uid	string	Unique ID
src-h	string	Source IP address
src-p	value	Source port number
dst-h	string	Destination IP address
dst-p	value	Destination port number
proto	string	Protocol of connection
duration	value	Time of last packet seen - time of first packet seen
conn-state	String	Connection state

SIEM

SIEM is a software application that *collects* log data and *correlates* log entries from different sources. The correlated data results in one or more events that which can lead to an alarm being triggered.

Event

An event is a subset of log entries, can be from different IDS logs. E.g. all login attempts by a specific user within a certain time span on a server.

Alarm

An alarm is a non-empty set of events that is send as a ticket to the ticketing system. Every alarm consist of a risk level that can be low, medium or high.

SIEM event rule

Correlates log entries from different log sources.

SIEM alarm rule

Correlates events to trigger an alarm.

Ticketing system

A ticketing system is an application that manages and maintains lists of issues within an enterprise. The shared system helps to make issues manageable by the users of the application. The output of the SIEM can be alarms send to the ticketing system. If a company makes use of multiple SIEM solution, it can easily send all the alarms to one central place.

Knowledge Base (KB)

A database with information about devices, their roles and features.

Open Source Intelligence (OSINT)

OSINT is information collected which is available to the general public, such as those available on the Internet, but the term isn't strictly limited to the internet. The information can be from any publicly available sources. Some examples are VirusTotal [19], Urlscan [18] and ThreatMiner [17].

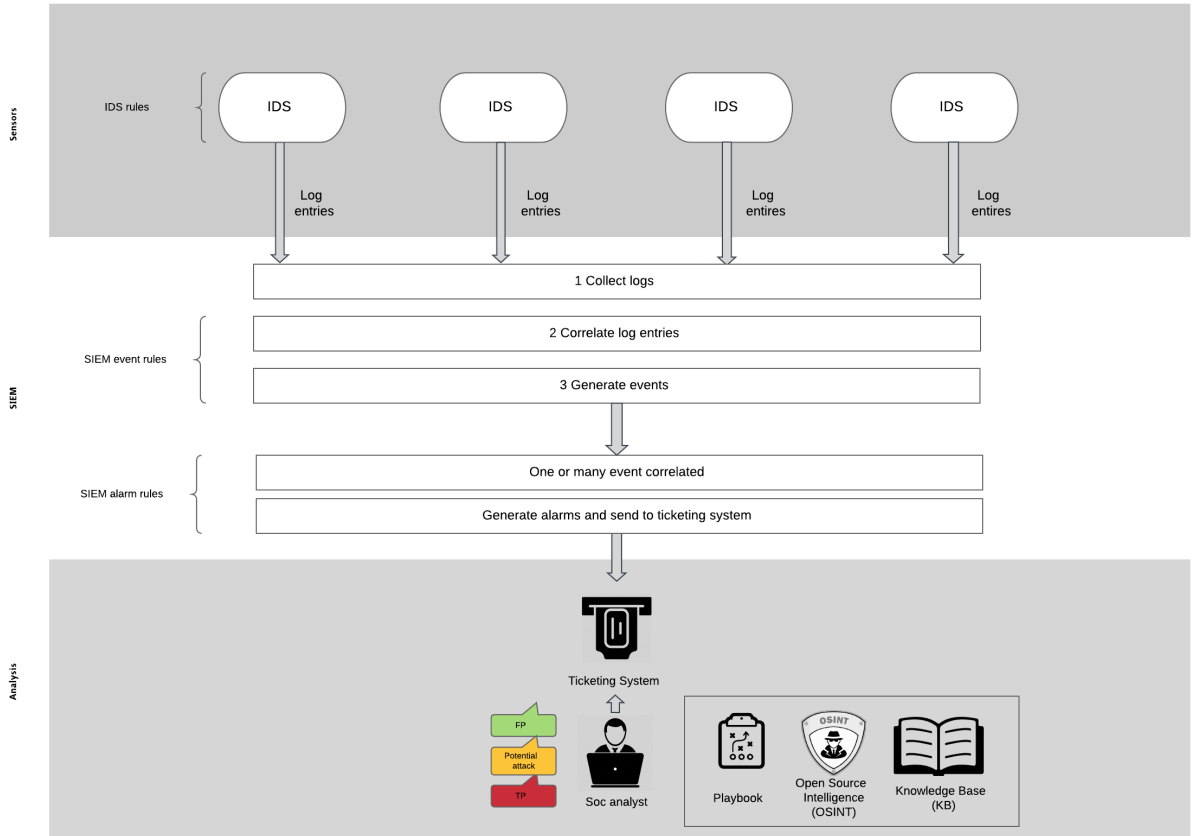

FIGURE 3.2: Overview current situation

The SOC consists of security specialists with different roles. The SOC analysts make sure that monitoring of all the client's their infrastructure is being done by handling incoming alarms, as explained above. To do so, the IDSs will be tuned to capture

and record necessary network traffic according to the IDS rules that are set. Every single IDS will save network traffic in log files according to the type of the IDS. The log files will consist of a sequence of log entries sorted on timestamp. A log entry will contains fields, where some are standard fields that are needed to know when and which computer devices communicated such as source IP, source port number, destination IP, destination port number and the timestamp of the log entry (see Table 3.1). Figure 3.2 shows that in the current situation that the IDSs send their log entries to the SIEM solution. The SIEM solution collects all the log entries captured by one or multiple IDSs and correlates the log entries. The correlation of log entries will make the SIEM system generate events. The SIEM event rules will correlate the log entries and output one or more events. In the next layer, as can be seen in Figure 3.2, the events will be correlated to generate alarms that will end up in the ticketing system. The alarm generated consist of one or multiple events and the event can be of one or multiple types. There can be an alarm that will be triggered with different event types. The SIEM alarm rules will look at the events generated and based on the rule and its event(s) it will generate an alarm. For both the event rules and alarm rules, it will mostly be based on a string or a counter.

To handle the alarms the SOC analyst can also make use of Knowledge Base (KB) where information about the assets and network infrastructure is stored to know where and how every server and machine is stationed. The additional information that can be found in KB can influence on the decision taken by an SOC analyst when handling alarms. Other sources of information to support the SOC analyst to handle incoming alarms are OSINT tools that are used by large communities to share information about suspicious and malicious hosts.

To make life easier for the tier-1 analyst, conceptualizing the whole SOC domain can help the analyst to make decisions faster. The next subchapter will explain how log entries and field are linked to the right incoming alarm. In this way, the analyst can use his time for alarms that really matters and spend time on those. Instead of looking at a large data set of logs, which is difficult to analyze, the analyst will get a description of the analyzed log entries and fields of an alarm. The next subchapter will explain how the right log entries are linked to the right alarm. In the next chapter we will explain how the framework can be used to add semantics to the logs of an alarm to detect false alarms and actual attacks.

Chapter 4

Framework for alarm handling based on conceptualization of data

In the previous section, the current situation has been explained and the problems SOC analysts have to deal with. This chapter explains how the framework works, its time complexity and the use cases where the framework is used on. As mentioned in chapter 3, SOC analysts need to analyze large datasets of low-level data for specific incoming alarms and this chapter will introduce a new way to add semantics to that low-level data to support the analysts to reduce the number of FPs.

4.1 Framework

As described in chapter 2, IDSs detect certain behavior by looking at the network traffic, where signature matches or deviations from normal behavior is seen. They create a log with *entries* for each of the detected network actions. The entry consists of a number of *fields* each describing some feature of the detected action, such as timestamp, source IP, destination IP, source port number, destination port number, protocol, connection-state etc. The problem of SOCs is that a lot of incoming alarms are FPS and the SOC analysts are wasting a lot of time to recognize these FPs, when looking at low-level data. Instead of wasting time on FPs, the SOC analysts should spend more time on alarms that are potential attacks or TPs. We want to add semantics to the low-level data by introducing higher level concepts.

Before explaining the framework, we first summarize (our formalization of) notions used in the framework by giving a short description and examples:

Low-level data

Abstract data from a system without contextual information.

Recorded network traffic by an IDS is what we call low-level data. To understand what the log events actually mean a documentation of the logs of the IDS is needed to understand the syntax and abbreviations used in event fields. Some can be deduced logically, but surely not all of them. Low-level data can be any data that is related to an alarm, from any IDS. Data from other domains where semantics can be added to make it more understandable can also be considered as low-level data.

For easy reference we use a field name (which we assume is unique to the field) to identify the field in our formalization. In practice fields can also be identified by their position in a tuple rather than an explicit label.

Concept

A concept is a label that is attached to low-level data. Some examples in the domain of cybersecurity are: *"server"*, *"client"*, *"malicious host"*, *"a successful login on SSH server"* or *"vulnerability scanner"*.

Predefined Concepts

The predefined concepts are concepts taken from low-level data and other additional information sources such as OSINT and KB. Done in design time.

Pattern

A pattern captures a situation in terms of concepts, which is a regular expression over concepts. It is used to capture a sequence of concepts that represents a certain scenario. The capturing can for example be based on the order of the concepts and the length of the sequence.

Output

This is a label given to an alarm, as seen in Figure 3.1, but now done automatically with a tool. The output can be either be *FP, potential attack or TP*, and in case there is no match it will give *no match* as label. Every pattern has a corresponding output. This means that for every pattern the two possible outputs are predefined. For example, if only successful logins are seen then the two possible outputs are *FP* and *no match*.

Conceptualized Entry

A pair of conceptualized entry and set of concepts labeled to the fields of the corresponding entry. There can be one or more concepts be attached to an entry. For example: *<Entry1>, {Successful login on SSH server}*.

Conceptualized Field

A Conceptualized Entry is a pair of entries and a set of concepts. In this way we get a field labeled with related concepts, such as *<SourceIP,1.2.3.4>, {httpserver, ftpserver}*.

Concept Assignment Process (CAP)

The Concept Assignment Process is an online process of labeling log entries and fields with predefined concepts. The steps are done according to the algorithm of the framework. The predefined concepts are used to label the entries and fields of an alarm. Done in runtime.

Conceptualized Alarm Trace (CAT)

A sequence of conceptualized entries. The following CAT T is an example of a sequence of entries with all concepts sl attached to, which stands for "successful login":

$$T = \{sl0 \ldots sl_m\}$$

Rule

A rule is a pair of a pattern and an output. Rules are used to classify a matched pattern in an alarm to the corresponding output (FP, potential attack, TP) if there is a match. Rules are defined in design time.

With these notions in place we now describe how the framework will conceptualize low-level data to detect common false positives. Figure 4.1 gives a visualization of the framework steps with the goal to detect patterns in conceptualized data. The concept is a label that is given to low-level data to add more semantics. A concept can be anything that adds semantics to any kind of low-level data, some examples are: "vulnerability scanner", "malicious host", "known host". The context and ideas of defining concepts can be taken from any source, such as KB or OSINT. However, there are also so-called standard concepts within every domain. The less domain specific the concept is, the more it is usable in other domains. An general concept could be "message", which can be a message from any application or maybe a letter. The concepts that are defined in section 4.3 are only usable in the domain of network security. In that domain concepts as "client", "server", "host" are general reusable concepts within the domain. In the domain of the IDSs, a concept can be attached to log entries (*conceptualized entry*), but also specific fields within log entries (*conceptualized fields*). It is also possible to label more than one concept to an entry and field. The complexity and problem of attaching more than one label to a specific entry or field is later on described in section 4.2.

The Concept Assignment Process (CAP) is the process where the predefined concepts are assigned to specific log entries or fields in runtime. For instance if a connection to a (known) malicious server is seen, then the destination IP of that connection can be labeled with the concept *malicious server*. If new connections are seen from that server, then that same field that has the same IP address will be labeled with *malicious host*. Another example is when that malicious host is trying to a SSH server that is externally reachable, then that entry can be labeled with *malicious host connecting to ssh server*. Concepts that are assigned to entries should be a summary of multiple concepts given to the fields. However, it is not obligated to assign concepts to fields and then assign a summarized concept to an entry. It is possible but unusual to only assign a concept to the entry, without assigning concepts to fields. The information of the logs are in the fields and not the log entry.

The conceptualized entries and fields are a sequence called the Conceptualized Alarm Trace (CAT). A pattern could be captures by matching the regular expression against the CAT. This means that everything that matches the regular expression against the sequence will get the output specified in the rule. The output that is given to an alarm are the same as seen in Figure 3.1, where the analyst had to carry out the action. If an alarm does not match the regular expression then the corresponding rule will give *no match* to the alarm. The matching process is done in runtime.

To specify rules to capture the pattern, we have to look at specific use cases. Every type of alarm needs its own rules to capture the right information. For this, we selected two specific use cases that will be explained in detail in section 4.3. The rules are used to replace the tier-1 analyst, by giving an output. The patterns that analysts try to find are mostly actions that have a certain order. Analysts will typically search for certain entries and concepts that occur in a certain order, so we can catch that with a regular expression on concepts. However, regular expressions are not usable to count certain occurrences. In our framework the order of occurrences are

important, which can be defined in patterns and that can be captured with regular expressions.

Below the steps taken in the framework are summarized to clarify the algorithm of the framework.

Summarized steps of the framework in Figure 4.1:

1. Alarm A_i consists of n events.
2. Sort the events of A_i according to their timestamp.
   Note: the events consist of log entries.
3. Alarm A_i now consists of M sorted log entries.
4. The predefined concepts are attached to the log entries in the Conceptualized Assignment Process (CAP).
   Note: log entries consist of log entry fields and can also contain concepts.
5. A Conceptualized Alarm Trace (CAT) contains all the conceptualized entries. CAT_i consists of M conceptualized entries.
6. Match the regular expression against the resulting CAT.
7. If a match, the corresponding *match* output for the rule will be labeled to the alarm or a *no match*.

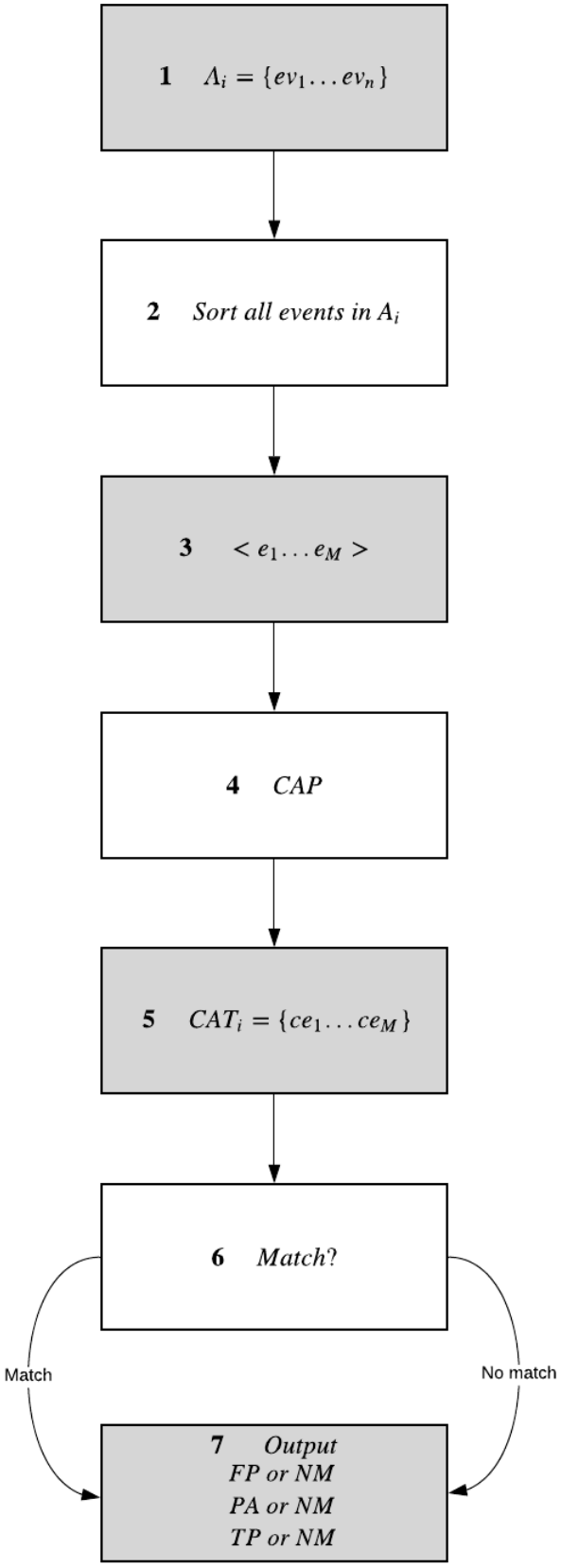

FIGURE 4.1: Components of the framework: where the white boxes represent *processes* and grey boxes *data*

4.2 Complexity analysis of pattern matching

As mentioned before, we make use of regular expressions because they match efficiently with a sequence. To make use of regular expressions we have to define rules. In this subchapter we will go more in detail about what the time complexity is of the used algorithm for pattern matching. The reason why this is necessary is because a single entry or field can have multiple concepts attached to it. Therefore, we have to verify whether the process of conceptualization and extraction is still efficient enough. For this, we want to turn regular expressions into an Non-deterministic Finite Automaton (NFA) [20]. We then simulate the conceptualized alarm trace using the NFA, where in each step we consider all possible concepts for that entry or field.

Another reason to verify the efficiency is because IDSs in practise should also be efficient when capturing network traffic. Most of the time the logging is not done by an IDS in real-time, meaning that it is not done in nano- or microseconds. Instead of that, the logging is done on-line meaning the user should not be bothered by the speed of feedback. Therefrom, we want to realize some form of efficiency as is the case with IDSs capturing network traffic.

NFAs are mostly used in the implementation of regular expressions. Regular expressions and Non-deterministic Finite Automaton (NFA) are two representations of regular languages. Where text processing utilities are regular expressions to describe advanced search patterns, defining matching with the conceptualizsed alarm trace and evaluating its complexity is easier using NFA. It is well known that regular expressions can be efficently (linear time) converted to NFAs that are the same order of size. We need not restrict to regular expressions; the algorithm and its complexity analysis apply to any input format that can be efficiently expressed as an NFA.

The input string will be accepted if the path (of the input string) leads to a final state in NFA. The following features are allowed in NFA : (1) null (or ϵ) move is allowed, (2) it can transition to any number of states for a certain input. The features will not have effect on the power of NFA.

Our analysis of the complexity Algorithm 4.1 shows that the complexity of the algorithm is very reasonable; it should be efficient enough to make the framework operable.

Definition 4.2.1. A Non-deterministic Finite Automaton (NFA) A is defined as a 5-tuple:

$$A = (Q, C, q_0, Q_f, \delta)$$

where
Q is a finite set of states
C is a set of input concepts
$q_0 \in Q$ is the initial state
$Q_f \subseteq Q$ is the set of final states
δ : Q x $(C \cup \epsilon)$ –> 2^Q is the state transition function.

We also define notation:
$C_\epsilon = C \cup \{\epsilon\}$.
N = size of regular expression (pattern) and NFA A.

M = length of the Conceptualized Alarm Trace (CAT).

The obtained automaton A when converting a regular expression R to an NFA using Thompson's construction yields a transition function δ that respects the following properties:

1. There is only one final state q_f, $Q_f = \{q_f\}$.
2. There is no transitions leaving the final state, $\delta(q_f, c) = \emptyset$ for all c in C_ϵ.
3. $\#\delta(q, c) \leq 2$ for all $q \in Q$, c in C_ϵ, meaning there are at most two outgoing arrows from a state.

The following algorithm matches a regular expression (or any other input efficiently expressible with an NFA) with a CAT a. Note that each conceptualized entry $a[i]$ in a consists of a set of concepts.

Algorithm 4.1 *Framework Algorithm(FA)*

```
1: procedure FA(r: regular language, a: alarm sequence)
2: Express reg lang r with an NFA A = (Q, C, q_0, q_f, δ)
3: S_0 = ε − close({q_0})                          # O(N)
4:     for i = 1 to |a| do                         # O(M)
5:         S_i = StepMatch(S_{i−1}, a[i])              # O(Nx|C|)
6:         S_i = ε − close(S_i)                        # O(N)
7:     return q_f ∈ S_{|a|}
```

Here $\epsilon - close(S)$ denotes the set of states reachable from a state in S with only ϵ steps. We see that the overall complexity is $\mathcal{O}(M \times |C| \times N)$ as the loop (with complexity $\mathcal{O}(N \times |C|)$, see below) is run M times.

Complexity in total: $\mathcal{O}(M \times |C| \times N)$.

Algorithm 4.2 *Step Match*

```
1: procedure StepMatch(S: set of states, e: set of concepts)
2:     S' = ∅
3:     for c ∈ e do                                # O(|C|)
4:         for s ∈ S do                            # O(N)
5:             S' = S' ∪ δ(s, c)                       # O(1)
6:     return S'
```

The first for loop in line 2 in algorithm 4.2 is added and different from a normal NFA step matching algorithm. The reason to add this for loop is because a log entry or field can have multiple concepts, and therefore we should also loop through the set of concepts of one entry or field.

4.3 Use cases

As mentioned, the rules defined are use case specific because every type of alarm has its own type of connections that needs to be analyzed differently by an SOC analyst. In this subchapter we will introduce two use cases to make it more understandable.

4.3.1 SSH bruteforce attack

Secure Shell (SSH) is a cryptographic network protocol that makes sure that a insecure channel becomes secure by establishing a secure channel within the insecure channel. It connects a SSH client securely with an SSH server. The purpose is to remotely open a shell (command-line) on another computer (server) so that commands from a computer (client) can be execute on the server side.

Authentication methods
For authentication SSH makes use of public-key cryptography. The public-key cryptography is used to authenticate the server and allow it to authenticate the client. There are multiple ways to use SSH:

- Automatically generate public-private key pairs to encrypt a network connection, after this use a password to authenticate to login.
- Manually generate public-private key pair to perform the authentication and after this no password is needed. In this way, public keys are placed on all clients that are allowed to connect to the server that has the matching private key, where the private key stays secret. Therefore, it is important to verify unknown public keys, and also know each single identity of the public keys. If this is not done, then the risk of authorizing an attacker as valid user is still present.

Dictionary attack
A dictionary attack is a brute force attack technique to break an authentication mechanism. Such an attack contains of trying lots of possibilities, like words from a dictionary. That is also the reason why it is named a dictionary attack. In a dictionary attack only the possibilities, which are most likely to succeed, will be used. This type of attacks are often successful because many people use common and short passwords and combine them with a characters like a digit or punctuation character. However, a dictionary attack is relatively easy to block, for example by using a passphrase or choosing a more difficult password and not a word or variant of it that can be found in a dictionary.

It is possible to protect a SSH server from a brute force or dictionary attack with the following things:

- Disable root access: Log in from your non-privileged user account and escalate privilege only when it is necessary. There are some tools that allow privilege escalation, for example SUDO and SU.
- Disable unused services
- Filter traffic to the SSH server with a network or host based firewall
- Run the SSH server on a high port

- Install and maintain anti-brute-force tools, like Pam_abl and SSHBan. The first tool provides a blacklist of hosts and users which are responsible for repeated failed authentication attempts, the second one receives data directly from the loggers.
- Use strong passwords
- Limit connection rates/SYN packets

This alarm is dealing with clients that are constantly trying to connect to a SSH server, without checking whether the connections are successful or not. If more than X connections are seen within a certain timespan, then it will trigger an alarm. This means that in this case, triggering an alarm is based on a counter. The SOC analyst will check within the logs whether the connection comes from an internal or external host, and also look at if authentication was successful or unsuccessful. Depending on whether the source host is from an internal host or external host, the risk can be decided on by the analyst in question. If the source host is external then that host can be blacklisted, and if the host if internal then that host should be investigated. The internal host that is trying to connect to the SSH server could be compromised in the worst case, but can also be misconfigured.

Remember that a rule of the IDS has the form explained in subsection 2.1.4.

IDS Rule: *alert tcp any any -> $HOME_NET 22*

The rule here will perform the action *alert*, having tcp as protocol, when an (internal or external) host is trying to connect to any internal host using port 22 (standard TCP port for contacting SSH servers).

Threshold: *type both, track by_src, count 5, seconds 120*

There is a threshold set to trigger the rule, which is a counter that keeps track of the number of connections. In this case the alert will be triggered if there are 5 connections made within 120 seconds. This rule will in this case create SIEM events for each 5 connections made within 120 seconds. The alarm can consists of multiple events, where the number of connections is equal to the number of events times 5.

We will for this type of alarm use the framework explained in section 4.1 to add semantics to the corresponding logs of such alarms. When such an alarm is seen in the ticketing system, the SOC analyst will start looking at the number of connections seen and what the entry field *auth_success* contains in all log entries. The field *auth_success* can have a false or a true flag, where true stands for successful login and false for unsuccessful login on the SSH server. The first step is to label the fields of the entries of the logs, of an alarm, with the following two concepts: *successful login* and *unsuccessful login*. By this, we end up with an alarm sequence of concepts. Before doing something with the sequence, we have to know which patterns represent an attack or are FPs.

One easy pattern seen is seeing only successful logins, this means the sequence only contains concept *successful login*. An example rule of this pattern can be seen below.

Framework Rule: *(all_successful, FP)*

4.3.2 Network Scanning

Network scanning is often seen as the first phase for an attacker to gather information about the targeted network. The attacker can use a tool to send raw IP packets to discover things as: available hosts, services offered by the hosts, operating systems running, and firewall types. A popular tool to discover services on networks is Nmap, which has the ability to scan (large) networks, but can also map out whole networks.

In general there are two main types of scanning:

- **Port scanning**: used to identify open ports and services available on a network host. The attacker can eather send requests to connect to the targeted hosts, and then keep track of the ports which are open, or those that respond to the request. As a burglar, you first check if there are windows or doors open, if not, you at least know what kind of doors and windows are installed.
- **Network scanning**: used to list IP addresses, open ports, or a range of IP addresses, and to discover the service available in the scanned range.

For both types of scanning there could be two actors. The first one is an attacker that is performing a port scan or a network scan to gather information about the network of the target host. The second actor can be a vulnerability scanner that is performing assessing configurations of servers on a network. This second actor "vulnerability scanner" is also used in this use case as concept to attach to the source IP of a connection. In case we know what time and day a specific host is fulfilling the role of a vulnerability scanner, then we can for that timespan attach the concept "vulnerability scanner" to it. In case a Conceptualized Alarm Trace consists of many "vulnerability scanners" then it can easily close the alarm as a FP.

IDS Rule: *alert tcp any any -> $HOME_NET any*

The IDS rule for this use case will perform the action alert, having tcp as protocol, when an (internal or external) host is trying to connect to any internal host using any port.

Threshold: *type both, track by_src, count 10, seconds 60*

The IDS rule will be triggered if there are 10 connections made within 60 seconds. This rule will in this case create SIEM events for each 10 connections made within 60 seconds. The alarm can consists of multiple events, where the number of connections is equal to the number of events times 10.

No failed connections is a pattern where the concept *not a failed connection* is present. This means that the host that is trying to connect to the other host does exactly know which service are available on which IP and/or port and therefore the chance of being a scan is low. Often when an attacker is trying to scan a network, then it has no clue which services are running on IPs and port numbers, that is why in the logs, when scanning, it should contain connection attempts that are failed.

The connection states in the log entry field determines whether the connection is failed or not. Two connection state examples: S0 (connection attempt seen, no reply)

and REJ (Connection attempt seen rejected). Such low-level data within the field *conn_state* is used in this use case.

Framework Rule: *(no failed connections, FP)*

The simplicity of the two mentioned IDS rules is the reason of generating lots of false positives. They do capture traffic that can be malicious, but they capture a lot of benign network traffic.

Chapter 5

Experimental evaluation

In this chapter, the experimental evaluation, setup, implementation and results are described. For both uses cases ***Network Scanning*** and ***SSH bruteforce attack*** the framework will used on two use cases to evaluate the detection of false alarms. To do so, we have to add attacks to the dataset to evaluate the rules that capture false alarms. The focus of this experiment is to test how many false positives are correctly labeled. The tool is a standalone product that gets data as input and gives a final label to the alarm analyzed. A contextual description was given in section 4.3 of the two tested use cases, in this chapter we will go more into the rules implemented.

5.1 Preparation

During the preparation phase of the framework test data was used to test the rules created. The sample size of the test data were smaller in the number of log entries that it had to process, but the number of alarms were the same size as for the experiment.

The first step taken for the experiment was to search and collect the data which consists of alarms with the corresponding logs, that represent the two use cases. To measure the correctness labeling by the tool, the *ground truth* should be determined. To get the ground truth, a tier-2 SOC analyst went through the alarms one by one to analyze the alarms (again) to make sure that there are no mistakes within the analyzes of an alarm. A lot of alarms are analyzed by tier-1 analysts that are less experienced and therefore the alarm could contain errors. The final analysis that the tier-2 analyst made per alarm is used as the ground truth for the experiment.

The alarms are taken from a ticketing system, which is the output from the SIEM system. The logs are taken from an IDS where logs are being recorded. An important aspect that was missing and needed is the linkage of the logs to the right alarm. To get this done, a tool that is described in [14] was available that does this for us. However, there were some bugs that were needed to be fixed before it was usable. One crucial issue was that logs were missing when multiple log entries had the same Epoch time. The linkage of all logs to the correct alarm properly to be able to link all logs to the corresponding alarm The tool only accepted one file format which was EDXML [16]. The files of the alarms and logs had to be transcoded to an EDXML format. A transcoder was available that directly took the alarms from the ticketing system and transcoded the data to an EDXML format. The logs needed more attention. I had to manually collect the logs that had to do with the right alarm. The same

tier-2 analyst that helped me with creating the ground truth, also supported me with checking if all the logs were properly collected.

5.2 Implementation

We will follow the steps of the framework to explain the implementations of the rules. The below explanation starts from step 6, which concerns the pattern matching. The other previous steps are the same in terms of structure and implementation for every alarm. Therefore, the focus will be mainly on the pattern matching process. The only difference within the first 5 steps of the framework is number of events and entries. The Concepts Assignment Process is done using heuristics and rules that are domain specific. The Conceptualized Alarm Trace can also differ in length per alarm.

Some regular expressions contain a variable x, which can be changed for all rules. Every network environment or person can has its own reason to select the right number to use for the variable x. The size is such that it can be used as the N in the complexity formulas. It is not exactly the size the automaton A will have but it is same order of magnitude N = O(|r|). We use the following common notation for regular expressions (we give a size for the expression to indicate the complexity of matching it against the CAT):

The most basic expressions (size 1) are ϵ (empty word) and a concept c. We use rr' to denote r followed by r' and $r|r'$ to denote a choice between r and r' (both have size equal to the sum of the component sizes plus one), $r*$ for 0 or more times and $r+$ for one or more times r (size is size of r plus one). We also use $r\{a, b\}$ to denote at least a and at most b repetitions of r (size b times size of r plus one) and ra, to denote at least a repetitions or r (size a times size of r plus one).

We also use a construction not common in regular expressions: $r \wedge r'$ to indicate that both r and r' should be matched. We can use this construction in our language as we can easily build an NFA for it. However, the size of $r \wedge r'$ is the *product* of the size of r and r' so this construction should be used with care, nesting it may drastically increase the complexity of matching.

The CAT is expected to only contain entries that are login attempts. This will helps us to introduce simple rules that capture certain patterns.

5.2.1 SSH bruteforce attack

For this type of alarm, we labeled the log entry field *auth_success*, which got either the value *T* or *F*, which stands for True and False. The predefined concepts used for the alarm are: *successful login* (sl) when reading *T* and *unsuccessful login* (ul) when reading F. A *T* means that the client successfully logged in to the SSH server.As mentioned in subsection 4.3.1, the IDS rule used to capture the logs does not check the value within the field *auth_success*. It simply triggers when an internal IP with any port number connects, 5 times within 120 seconds, to any destination IP with port 22.

- **Rule 1**: (all successful logins, FP)
  Defined regular expression: **sl+** One or more of sl.

  Rule 1 captures the pattern named *all successful logins* and if there is a match when looking at the CAT then the alarm will give FP as output. In case all those login attempts are successful it always means that the client had the right information to login meaning it can never be an attack, and must be successful attempts by a legitimate user.

- **Rule 2**: (few unsuccessful logins, FP)
  Defined regular expression: **ul{1,x}**. **ul{1,10}**.

  Rule 2 captures the pattern named *few unsuccessful logins* and we used 10 for the value *x*. It will match when between 1 and 10 unsuccessful logins are seen in the CAT. A few unsuccessful logins does not represent a bruteforce attack.

- **Rule 3**: (unusual unsuccessful logins, potential attack)
  Defined regular expression: **ul{x,} ul{10,}** 10 or more unsuccessful logins.

  For Rule 3 captures the pattern *unusual unsuccessful login* and for the value of *x* we use 10. More than 10 unsuccessful logins can indicate that an attacker is trying to bruteforce the SSH server or that a host is misconfigured and for instance tries to login every *m* minutes to the SSH server. One of the two actors is malicious and therefore we get *potential attack* as output if the pattern matches. The CAT does not contain *successful login* and is therefore not directly a danger, but has the potential. In this case the SOC analyst could contact the client to ask why that traffic occurs, if the source IP is an internal IP address. In case the source is an external host, then it can for instance be an attacker that is still busy with a bruteforce attack e.g. dictionary attack.

- **Rule 4**: (multiple unsuccessful end with successful, TP)
  Defined regular expression: **ul{x,}sl ul{10,}sl** 10 or more unsuccessful logins followed by a successful login.

  Rule 4 captures the pattern *multiple unsuccessful end with successful*, when there is a match then a TP will be given as output. This pattern represents a typical dictionary attack where for instance an attacker tries a list of most common passwords.

- **Rule 5**: (mix, FP)
  Defined regular expression: **(ulsl | slul)***, where #ul $\approx$ #sl.

  Rule 5 captures the pattern named *mix of successful and unsuccessful logins* and if there is a match when looking at the CAT then the alarm will give FP as output. This rule is created for the alarms that contain unsuccessful login and

successful login, where the number of both concepts in the CAT are almost the same. Note that while implementing side condition #ul $\approx$ #sl added here is relatively straightforward, it cannot be captured as a regular language. Therefore, the complexity analysis that we performed in section 4.2 thus does not cover the side condition. For this specific side condition it is clear that we can check it independently from matching the regular expression and within $\mathcal{O}(|C| \times M \times N)$time so it does not increase the complexity of matching.

A scenario could be that a user or server that tries to access multiple servers with the same credentials, where for some of the servers the credentials works and some not. This will then generate a mix of successful and unsuccessful logins. For the implementation the word *some* needs to be specified. The difference between the number of unsuccessful logins and successful logins are at most 10%. If the CAT contains more difference than 10%, then the CAT could possibly contain a pattern that consists of malicious behavior.

5.2.2 Network scanning

For this type of alarm, we labeled the log entry field *conn_state*, which gives information about the status of the TCP connection. The status of the TCP connection can either represents a *failed connection attempt* (fc) or an *established connection* (nafc). As explained in subsection 4.3.2, a scan blindly tests every port or destination, and thus most of the connections should contain failed connection attempts. The value within the log entry field *conn_state* that represent failed connection attempt are labeled with the concept *failed connection* and the ones that represent established connections are labeled with the concept *not a failed connection*.

- **Rule 1**: (No failed connections, FP)
  Defined regular expression: **nafc+**

  Rule 1 captures the pattern *no failed connection attempts*, when there is a match then a FP will be given as output. In case the CAT does not contain a failed connection, then it is straightforward to say that it is not a scan.

- **Rule 2**: (Contains failed connections, TP)
  Defined regular expression: **nafc*fc{x,}(nafc+fc)***.

  In rule 2 we try to capture the pattern *contains failed connections*, when there is a match then a TP will be given as output for the concerned alarm. For this rule we give 50 to the value of x, in case of a match then it means that the CAT contains at least 50 failed connection. A scan should contain failed connections else the actor knew what kind of services was running on a given port.

5.3 Datasets

To collect sample of the alarms which is a representative dataset from the original dataset, we make sure to meet the following requirements[12]:

1. The sample is significantly smaller in size compared to the original dataset.
2. The sample captures the most patterns seen in alarms.

3. The sample size has low redundancy, meaning that the dataset consists of alarms with multiple scenarios.

The dataset for the experiment of the first use case has a sample size of 30 alarms. For the *SSH Brute force attack* use case 3 (10%) of the alarms are actual alarms that need to be escalated by a SOC analyst, 3 (10%) other alarm are *potential attacks* and 24 (80%) are false positives and are closed.

In the dataset of the use case ***Brute force attack*** there are two types of attacks: a unsuccessful bruteforce attack and a successful brute force attack, where a number of unsuccessful attempts end with a *successful authentication* to the ssh server. This type of attack is called a dictionary attack, which is a form of bruteforce attack techniques to defeat the authentication mechanism by trying passwords by trying a large amount of likely possibilities. The attack on itself is weak, but can be successful because many people tend to choose short passwords that are common to use. The attack type is generated and is no a real attack performed by a real attacker. It was hard to find real attacks, if there were any, and the tool is primarily created to detect false positives. The reason to add the attacks to the dataset is to evaluate the rules that capture the false positives. If no attacks are present in the dataset, then you can simply label everything as false positive, without knowing if the rules that should capture false positives also capture (wrongly) the attacks. If that is the case, then the rules created would be unusable and therefore did not meet its goal.

The dataset for the experiment of the second use case has also 30 alarms. Where 6 (20%) alarms are actual alarms that need to be escalated by a SOC analyst, and 24 (80%) are false positives and are simply closed by the SOC analyst.

Figure Table 5.1 gives a complete overview of the explained datasets for both use cases.

TABLE 5.1: Dataset overview

Use case	FP/TP/PA	Number of alarms
SSH BFA	FP	24
	PA	3
	TP	3
Scanning	FP	24
	TP	6

5.4 Results

The **Ground Truth** (GT) establishes for each of the 30 alarms whether an alarm belongs to the scenario (M) which the rule is intended to capture or not (NM). This ground truth is made with the tier-2 analyst. As the scenarios are mutually exclusive and cover all of the alarms, we see that the total number of matches count up to 30.

The results of the framework are shown in the **Correctly Labeled (CL)** and **Incorrectly Labeled (ICL)** columns. Each attempt to match a rule against an alarm can have one of four possible outcomes:

1. [$CL - M$]: The rule Matches(M) when it should; the alarm gets assigned the correct label.
2. [$CL - NM$]: The rule does Not Match (NM) when it should not; this rule, correctly, does not assign any label.
3. [$ICL - M$] (*soundness error*) : The rule matches when it is not supposed to; the alarm will be given an incorrect label. When the outcome of the rule is FP but the matched alarm is actually an attack, this could lead to missing the attack. It is thus essential to avoid this type of error.
4. [$ICL - NM$] (*completeness error*) : The rule does not match when it should. No label is assigned. While the handling of this alert is not automated this type of error is less critical as the alert will still be evaluated by the analyst.

The next lines will (to make it more understandable) define calculations for each certain scenario:

- Total number of alarms = GT(M) + GT (NM)
- Total number of correctly labeled alarms = CL(M) + CL(NM)
- Total number of incorrectly labeled alarms = ICL(M) + ICL(NM)

  Note: every rule, within the use case, is used against all the alarms.

TABLE 5.2: SSH BFA results

Pattern	GT		CL		ICL	
	M	NM	M	NM	M	NM
All sl	10	20	10	20	0	0
Mix sl and ul	10	20	5	20	0	5
Few ul	4	26	4	26	0	0
All ul	3	27	3	27	0	0
Multiple ul ending with a sl	3	27	3	27	0	0

GT = Ground Truth, CL = Correctly Labeled, ICL = Incorrectly Labeled,
M = Match, NM = No Match.

The results in Table 5.2 shows that there are no soundness errors which is essential; all assigned labels are indeed correct. Of the 10 alarms that fit the scenario of the "*Mix sl and ul*" rule only 5 were matched, the other 5 failed to match.

Overall 25 out of 30, which is approximately 83%, of the alarms were assigned the correct label in use case *SSH BFA*.

TABLE 5.3: Scanning

Pattern	GT		CL		ICL	
	M	NM	M	NM	M	NM
No fc	24	6	18	6	0	6
Contains fc	6	24	5	24	0	1

GT = Ground Truth, CL = Correctly Labeled,
ICL = Incorrectly Labeled, M = Match, NM = No Match.

For the second use case *Scanning*, the results in Table 5.3 also shows that there are no soundness errors which is essential; all assigned labels are indeed correct. Of the 24 alarms that fit the scenario of the "*No fc*" rule 18 were matched, the other 6 failed to match.

Overall 23 out of 30, which is approximately 76%, of the alarms were assigned the correct label in use case *Scanning*.

Chapter 6

Conclusion

This chapter describes the conclusion of this thesis and provides different suggestions for future research. Firstly the main goal of the research is given, followed by the problem statement and research question. Then the steps towards the results are mentioned, which ends with the answer to the research question. The second part points out the possible directions for future research.

6.1 Overall conclusion

The main goal of this research was to reduce the effort needed by an SOC analyst to identify false alarms, because lots of incoming alarms in the Security Operations Center (SOC) are False Positives (FPs). A solution is proposed by introducing high level concepts that add semantics by labeling low-level (alarm) data with predefined concepts. The process of conceptualizing low-level data related to an alarm, helps us to find common patterns. This will result in a conceptualized log entry, which is a sequence containing concepts. The sequence of concepts, so-called Conceptualized Alarm Trace CAT, can contain a certain pattern that represents a false positive, potential attack or true positive. A pattern will capture a scenario in terms of concepts, which is a regular expression over concepts. The rules capturing the patterns are domain specific, and therefore we looked at two specific use cases to test the framework. This lead us to the following research question: *Can conceptualizing security data reduce the effort needed by a SOC analyst to distinguish attacks from false alarms*?

First we saw that the performance of an IDS is determined by the trade-off between FPs and FNs. Engineers within SOCs often choose to capture lots of alarms consisting of FPs instead of having the fear of missing actual attacks. This choice has effect on the workload of SOC analysts. They will get many alerts, most of which are false positives. As such we search for methods to automatically identify clear false alerts. In chapter 4 we proposed a framework that can be used to conceptualize low-level data, such as logs of IDSs. The framework makes it possible to find patterns of common false alarms by conceptualizing logs, which are used by SOC analysts to find evidence. The rules that are used to find certain patterns are domain specific and therefore we tested the framework on two use cases. In the two use cases we made sure that the correct logs were collected for every alarm. After that, the ground truth was defined by an experienced tier-2 SOC analyst to exactly know whether the alarms are attacks or false alarms. The first use case represents alarms that were triggered because of a brute force attack on SSH servers. The second use case represents alarms that were triggered because of scanning behavior. The logs

of the alarms for the two use cases were considered as low-level data that needed to be conceptualized. Concepts were defined based on the context of the alarm and the information within the logs. These concepts were then assigned in the Concept Assignment Process (CAP) using heuristics and rules to get a Conceptualized Alarm Trace (CAT). The CAT is a sequence of conceptualized log entries that is used in the next step of the framework where pattern matching takes place with regular expressions. A match triggers the rule to give the corresponding output to the alarm. All rules are used in each use case for its alarms in the dataset. The results have shown that for use case "SSH brute force attack" 83% of the alarms was correctly labeled with the used rules, while in the second use case "Scanning" 76% were correctly labeled. The results are mostly dependent on the concepts and rules created. Therefore, we could say that the patterns used in the rules will probably perform as good as in the experiment, because the dataset captures the most common scenarios from the original dataset. The results definitely showed that most of the false alarms are correctly labeled and will support the SOC analyst to distinguish attacks from common false alarms. Also, the Service-Level Agreement due date of the alarms can easier be met, because of the already the labeled false alarms that need less attention and thus less effort for the SOC analyst to handle those false alarms.

6.2 Discussion and future research direction

The tool that correlates the logs to the alarms was lacking in processing speed. The test-data used for developing the framework was in size smaller than the datasets used for the experiment. The alarms within the test-data had less log data to process than the log data used in the experiment. Furthermore, the log data of the attacks were manually created for the use case "SSH brute force attack", because it was challenging to find attacks, if there were any. On the other hand the two use cases chosen represent approximately 7% (SSH Brute force attack) and approximately 5% (Scanning) of the total number of alarms that contains all the use cases implemented in the SOC for the clients. Automating a good portion of these cases means that the given rules already cover a significant number of all alerts. Considering additional use cases will further increase this percentage, although diminishing returns will set in when moving to less common use cases. These results confirm that the conceptualized rules can automate a significant portion of the SOC analyst's workload.

Some suggestions for future research:

- We mentioned that the rules are domain (alarm) specific. Every type of alarm has it own set of rules. Therefore, it would be interesting to also use the framework for other types of alarms with its own set of rules. To create these other rules, new concepts should be introduced based on the context of the alarm.
- The process of defining concepts can possibly be replaced with a tool that does this for us. The tool should recognize different type of data and extract concepts out of that data. It would be interesting to investigate how that process can be improved with such technology.
- Now that feasibility of automating work with conceptualized rules has been established, it would be interesting to measure and optimize the benefit of

conceptualizing in ease of specifying, maintaining or reusing rules across different SOC setups and the benefit of concept labels for alarms the analysts still need to cover manually.

Bibliography

[1] James P Anderson. "Computer Security Technology Planning Study. Volume 2". In: (1972).

[2] James P Anderson. "Computer security threat monitoring and surveillance". In: *Technical Report, James P. Anderson Company* (1980).

[3] Sandeep Bhatt, Pratyusa K Manadhata, and Loai Zomlot. "The operational role of security information and event management systems". In: *IEEE security & Privacy* 5 (2014), pp. 35–41.

[4] Youksamay Chanthakoummane et al. "Improving intrusion detection on snort rules for botnets detection". In: *Information Science and Applications (ICISA) 2016*. Springer, 2016, pp. 765–779.

[5] Kassidy P Clark et al. "Secure monitoring of service level agreements". In: *2010 International Conference on Availability, Reliability and Security*. IEEE. 2010, pp. 454–461.

[6] Dorothy Denning and Peter G Neumann. *Requirements and model for IDES-a real-time intrusion-detection expert system*. SRI International, 1985.

[7] Davide Fauri et al. "Leveraging Semantics for Actionable Intrusion Detection in Building Automation Systems". In: *International Conference on Critical Information Infrastructures Security*. Springer. 2018, pp. 113–125.

[8] L Todd Heberlein et al. "A network security monitor". In: *Research in Security and Privacy, 1990. Proceedings., 1990 IEEE Computer Society Symposium on*. IEEE. 1990, pp. 296–304.

[9] Neminath Hubballi and Vinoth Suryanarayanan. "False alarm minimization techniques in signature-based intrusion detection systems: A survey". In: *Computer Communications* 49 (2014), pp. 1–17.

[10] Nattawat Khamphakdee, Nunnapus Benjamas, and Saiyan Saiyod. "Improving intrusion detection system based on snort rules for network probe attack detection". In: *Information and Communication Technology (ICoICT), 2014 2nd International Conference on*. IEEE. 2014, pp. 69–74.

[11] Hung-Jen Liao et al. "Intrusion detection system: A comprehensive review". In: *Journal of Network and Computer Applications* 36.1 (2013), pp. 16–24.

[12] Feng Pan et al. "Finding representative set from massive data". In: *Fifth IEEE International Conference on Data Mining (ICDM'05)*. IEEE. 2005, 8–pp.

[13] Martin Roesch et al. "Snort: Lightweight intrusion detection for networks." In: *Lisa*. Vol. 99. 1. 1999, pp. 229–238.

[14] Colin Smits. "Model Based Concept Mining Applied to Information Security Data". In: (2018).

[15] *Suricata*. URL: https://suricata-ids.org.

[16] Dik Takken. *EDXML v3.0 2018*. URL: www.edxml.org.

[17] *ThreatMiner*. URL: https://www.virustotal.com.

[18] *Urlscan IO*. URL: https://urlscan.io.

[19] *VirusTotal*. URL: https://www.virustotal.com.

[20] Guangming Xing. “Minimized thompson NFA”. In: *International Journal of Computer Mathematics* 81.9 (2004), pp. 1097–1106.