MASTER OF SCIENCE THESIS

Model Based Concept Mining Applied to Information Security Data

Author:
Colin Smits

Supervisor:
Eric Verheul (Radboud University)
Erik Poll (Radboud University)
Dik Takken (Northwave B.V.)

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Digital Security Group
Institute for Computing and Information Sciences

November 13, 2018

Abstract

Security Operations Center (SOC) analysts are concerned with triaging alarms coming from an Intrusion Detection System quickly but carefully. In most cases, however, scattering of information throughout multiple sources makes it difficult to do so. We introduce a way to capture information from different sources into one structured dataset, by use of EDXML. We define the notion of Model Based Concept Mining to retrieve entities defined by an ontology. By defining the ontology, we add semantics to the data involved with security events, which make it possible for a computer to understand the data. A framework for the process of going from raw data to the presentation of the relevant information consists of four stages, two of which are application dependent. We show how to implement these steps to create a graph yielding the information necessary for an analyst to gather insights into the context of an alarm, and to aid in making a justified decision on the risk an alarm poses. The process of generating a structured data set and presenting the data consists of four steps, which can be tailored to the needs for each application domain. For the SOC domain, time constraints play a key role, and adaptations to the model are necessary. A preliminary evaluation of evidence presented after applying the model on real data has been conducted. This evaluation did not test the model itself, but its use when analysing an alarm. This evaluation showed that there are potential benefits in terms of speed and focus when analysing an alarm.

Acknowledgements

I would like to thank all my colleagues who helped me with performing the evaluation, and of course my supervisors which provided me with support and critical questions to help shaping the model proposed in this thesis.

Contents

List of Figures

List of Tables

List of Abbreviations and Definitions

IDS	**I**ntrusion **D**etection **S**ystem
SOC	**S**ecurity **O**perations **C**enter
CERT	**C**yber **E**mergency **R**esponse **T**eam
IOC	**I**ndicator **O**f **C**ompromise
DMZ	**D**e-**M**ilitarized **Z**one
IP	**I**nternet **P**rotocol
URL	**U**niversal **R**esource **L**ocator
TCP	**T**ransport **C**ontrol **P**rotocol
MAC	**M**edia **A**ccess **C**ontrol
KB	**K**nowledge **B**ase

Domain Specification	Description of concepts, attributes of concepts, and relations between concepts, created by a domain expert
Concept	Specific object type within a domain described by attributes
Concept Instance	Instantiation of a concept with values assigned to the attributes
Event	Change within the domain affecting attributes and relations
Confidence	Probability describing to what extent an attribute uniquely defines a concept

Chapter 1

Introduction

1.1 Background

It is nearly 30 years ago that Tim Berners-Lee brought forward what is now known as 'the World Wide Web' [3]. This has since led to a transformation of business activity from paperwork to a fully digitised environment.

The last decade, digitisation of companies introduced new problems because valuable data needed to be secured. Attackers, competitors or even state actors are finding ways to steal data continuously, with various reasons. With more people and devices being connected, the risk of a breach is continuously increasing. While attacks could lead to considerable damage, most companies and individuals try to secure their networks and assets.

To be able to identify attacks as they happen, intrusion detection systems (IDS) that continuously analyse data emerged. These systems record all sorts of data travelling around a network, and capture irregularities. In the case of abnormal activity, the system raises an alarm. This alarm is then looked at by an analyst, who tries to find the reason why the system triggered an alarm. This is key to identifying whether the activity found is indeed harmful and thus if actions have to be taken in order to secure the data.

The analyst has to infer reasons for the alarm based on data presented by traffic logs, vulnerability scans or any other piece of information that is recorded by the system. To solve the puzzle, the analyst has to manually find interesting parts of the logs, and trace back the path that triggered the system. In most cases, this requires repeating a series of steps, which in turn makes it possible to make errors. Besides, alarm fatigue may occur, due to the fact that there are a lot of false positives raised.

1.2 Related Work

IDS has been a focus of research for the last decades ([10]). However, most research has looked at how to implement an IDS. We will deal with IDS and how they work

in Section 2.1. Besides the advances in the field of information security, artificial intelligence and data science are major subjects of research currently. Data mining is being applied on a multitude of domains to gather insights in the data contained. This research in related to that, as we will try to combine ideas from both fields into the domain of a SOC.

1.3 Research Question

This research tries to find a way to assist analysts in judging whether an alarm is a true positive or not, and try to find the root cause of an alarm by linking some pieces of the puzzle together. The question to be answered is the following: *To what extent can model based concept mining be used to aid analysts in judging IDS alerts*? The following sub-questions guide this process:

1. How are alarms generated by an IDS?
2. What data is involved in judging an alert?
3. What is Model Based Concept Mining?
4. What is the current workflow for an analyst and how can this be improved?

1.4 Scope

The focus of this research lies in the post-processing of the raw event data generated by an IDS in such a way that an analyst can perform his work more easily. As there are a lot of different attack scenarios, each with its own characteristics, a selection has been made on the following criteria:

- Prevalence
- Impact
- Availability of data

This selection is further discussed in Chapter 3.

1.5 Outline

In Chapter 2, we take a close look at IDS and their properties. Furthermore, we show a couple of prevalent attacks that they try to uncover. Chapter 3 deals with the data sources that are necessary within the research domain of a SOC, and shows the use case that is taken as example to test the model. In Chapter 4, we introduce Model Based Concept Mining and the mathematical properties of the model, as well

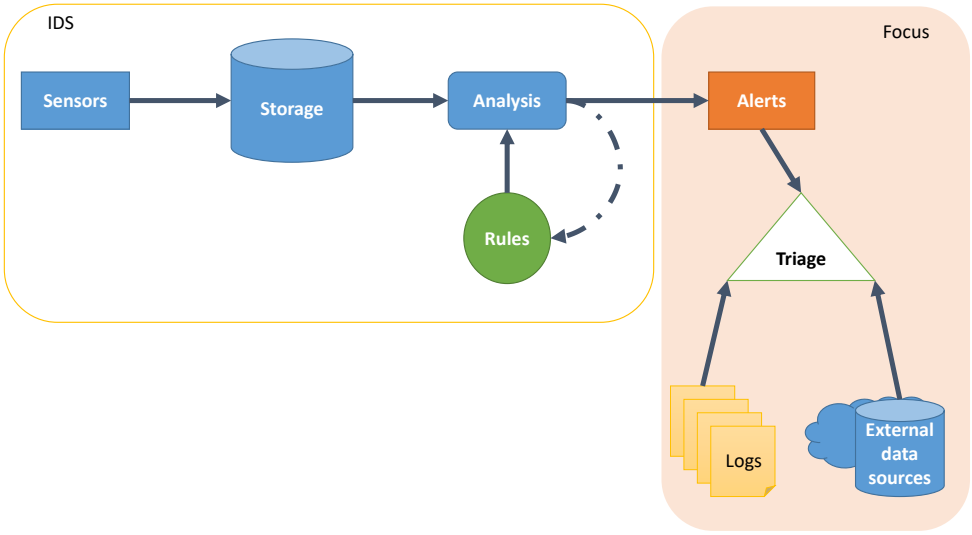

FIGURE 1.1: Outline of the processes from raw events to alarm triage. The focus of this research is mostly based around the triage.

as the general sequence of steps necessary. Chapter 5 discusses alteration to the general model specific to the SOC use case. Chapter 6 shows the results obtained with the Proof of Concept, Glaukos. Finally, Chapter 7 concludes the findings and give pointers to further research topics.

Chapter 2

Theory

This chapter will be focused on identifying theoretical aspects of intrusion detection systems and the way they are built up. This is dealt with in Section 2.1. Then, in Section 2.2, types of prevalent attacks on networks will be shown. Lastly, a couple of analysis methods for IDS data are introduced in Section 2.3.

2.1 Intrusion Detection Systems

As computer systems became more prevalent in offices, people started to identify risks of intruding machines on local networks. The first ideas of an intrusion detection system were proposed during the 1980s. In 1987, Dorothy E. Denning introduced a model to capture events in such a way that rules could decide whether an event was abnormal [10]. The proposed model dealt with the following entities:

Subjects The users of systems in a network.

Objects The devices that are being used by the subjects.

Audit Records Records that are generated when subjects (attempt to) perform actions on objects, for example a login attempt or file access.

Profiles Data that characterises a subject's behaviour with regards to objects.

Anomaly Records Records generated when behaviour deviating from a particular profile is detected.

Activity Rules Descriptions of actions that are taken when certain conditions are met, like updating profiles or generating audit records.

The proposed model was a so-called expert system, because rules that would check whether certain behaviour could be classified as abnormal had to be written beforehand. In addition, the system described profiles which were used to characterise patterns in the behaviour of subjects, in such a way that normal usage could be distinguished from possible intrusion activities.

2.1.1 Types of IDS

On a high level, two types of intrusion detection systems can be identified [9]:

Knowledge-based Systems that use predefined rules or information that characterise which activity should be labelled as an intrusion.

Behaviour-based Systems that depend on deviations from a learned model to classify new behaviour.

Knowledge-based systems accept any action that is not explicitly stated in the rules. Thus, if the system triggers an alarm, the action performed should be anomalous. However, due to the continuous evolution of attack patterns, it is difficult to keep the rules up to date. If a new attack pattern is used on the system, and the rule base is not yet updated, the attack might not be identified. One way to model rules is to have a list of signatures for network traffic that are makred as harmful. If a signature observed is present in the list, an alert is generated.

Behaviour-based systems use a different approach: instead of hard rules stating which activities should trigger an alarm, they model a baseline that describes what normal behaviour on the system looks like. When a pattern is found that deviates too much from the baseline, the system triggers an alarm. However, it might not always be the case that a deviation from normal activity is an attack. This introduces the risk of false positives, which might pollute the alarm base. On the other hand, while behaviour-based systems do not rely on a set of rules, they are capable of identifying new attack patterns. The necessity is to have an accurate model for normal behaviour, which is a complex task.

In some literature [16], the class of *behaviour based* systems is divided into two subcategories: *anomaly based intrusion detection* and *stateful protocol analysis*. Stateful protocol analysis is useful in the case of finding strange sequences of commands being issued in a certain protocol. This might indicate that someone is trying to break in via misuse of the protocol.

In most cases, alerts generated by an IDS have to be checked by analysts. To be able to correctly identify whether an alert has to be investigated further, the analyst has to find out why the alert was triggered. The advantage of using a knowledge-based system is that it is easy to find the reason of the alert, namely the rule that was applicable on the data. On the contrary, an anomaly-based system is less transparent as to why an alarm was generated: it can only show the used statistics and measures that generated the alert. Therefore, the latter is more difficult to process for an analyst.

2.1.2 Scope of IDS

Intrusion detection systems monitor activity on a variety of devices. When the first system was introduced, it took just a single device with multiple users into consideration [10]. Disk accesses, file execution and login attempts were some of the pointers being looked at. Such a system can be defined as *host-based*. Nowadays, we have multiple machines tied together within a network, or even residing in the cloud. This poses the challenge that not only one single device has to be monitored, but data within a large network, and going in and out of the network, must also be incorporated. Such systems can be classified as *network-based*. An advantage of this approach is that more data is available for the creation of a baseline. This can also be considered a disadvantage due to the complexity of generating the baseline.

One way to monitor a network is by means of *packet sniffing*. Information extracted from network traffic includes IP addresses of the source and destination, source and destination ports, the protocol used and the size of a packet. In some cases the full raw content of a packet is scanned. While scanning the whole packet gives the most information, it also has a high consumption of resources. Besides, privacy issues may prevent data from being stored. In turn, only the source and destination indicators might give enough information already.

To elaborate more on different connections, there exists a way to aggregate the packets coming through. One way to do this is by capturing IP flows, which will be further discussed in Section 2.3.1. The idea is to combine packets with the same origin and destination over time, in order to extract information about the duration of the communication as well as the size of the data being sent.

Another way to look at scoping of IDS is by using the OSI model [14]. This model divides a communication system, such as a computer, into a set of 7 layers. Each of these layers has its own function and protocols to communicate with other systems. The example of packet sniffing would be performed on the third layer, also known as the network layer. However, an IDS may also look at file access on a single host, in which case it performs on the application layer. A depiction of the different layers and the possible monitoring of an IDS can be found in Table 2.1.

TABLE 2.1: The OSI model of communication and the possible IDS monitoring per layer.

OSI Layer	**Possible Monitoring**
Application Layer	Antivirus, syslog, resource access, user behaviour
Presentation layer	
Session Layer	Authentication attempts
Transport Layer	Protocol Analysis (TCP/UDP)
Network Layer	Packet inspection, firewall rules
Data Link Layer	MAC Address monitoring
Physical Layer	

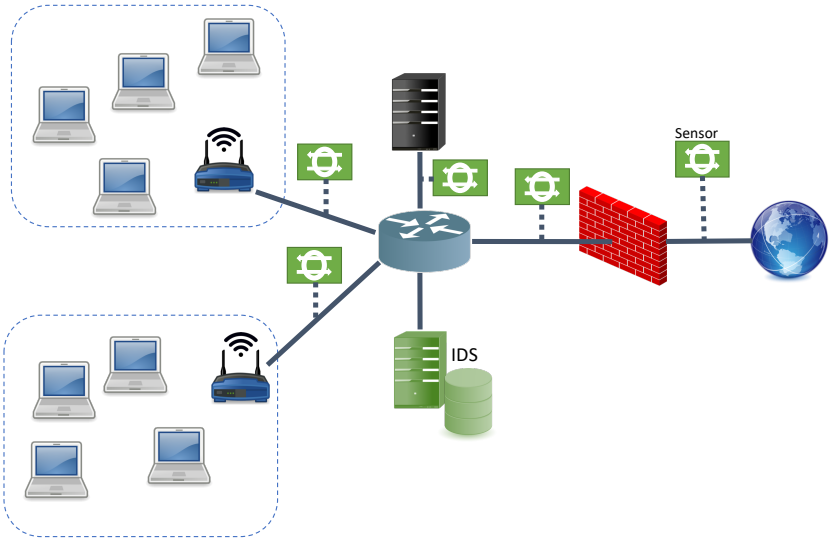

FIGURE 2.1: Typical outline of a network with sensors which provide input for the IDS. These sensors are attached to the different subnets of the network.

2.1.3 Components

A typical IDS is split up into three main components: sensors, storage and an analysis engine, as shown in Figure 1.1. First of all, sensors attached to the network that is monitored are used to capture traffic and generate logs based on the observed traffic. These logs are then stored in the storage mechanism, which in most cases would be a database. Then, the analysis engine can process the logs by applying rules and correlation techniques, producing alarms if anything is matched. The basic architecture of a network being monitored by an IDS is shown in Figure 2.1.

Rules

The basis of IDS alert generation lies in the rules that define what behaviour is anomalous. To exemplify how rules are built up, we use the rule syntax as proposed by Roesch in his introduction of Snort, a network-based IDS [24].

The basic building blocks of a rule are as follows:

Action The action that needs to be performed if the rule matches. Either one of **drop** (drop all traffic that matches the rule), **log** (log all traffic that matches the rule) or **alert** (raise an alert if traffic matches the rule).

Matching Rule The core part of the rule that matches to certain traffic. This consists of the protocol, the direction and the source and destination of the traffic.

Optional Fields Additional flags that indicate for example what content should be matched in the traffic or what message is applied to the alert if generated.

For example, we may be presented with the following rule, which indicates that all traffic over TCP to port 80 in the destination IP range (10.0.0.1/24) should be logged:

```
log tcp any any -> 10.0.0.1/24 80
```

When we want to generate an alert based on the previous rule, given that it contains the value *malicious*, one may add the following:

```
alert tcp any any -> 10.0.0.1/24 80 (content:  "malicious";
msg:  "Malicious traffic to a webserver detected")
```

The collection of rules, also called the rule base, is the main resource for a knowledge-based IDS analysis engine for the creation of alerts.

2.1.4 Risk Assessment

In order to characterise the seriousness of an event, an IDS awards a risk score to an alert, based on data that can be linked to the objects involved. This risk score may be calculated using the following factors:

Asset value The importance of the asset involved. Given a server in a DMZ serving web pages and a server containing secret data, the latter is of higher importance, and an alert generated for this asset may have a higher risk score than the first asset.

Impact A threat of gaining unwanted privileges has a higher risk than the detection malicious software that annoys the user by showing certain advertisements.

The risk score may be used by analysts to quickly assess which alerts should be handled first to prevent damage, and which might not be posing any harm to a user.

Even though a single event on itself might not have a high impact, in certain cases and sequences of events, the same event might be threatening. Thus, the recent history of events is important in the assessment of new events.

2.2 Prevalent Attacks and Threats

To examine how analysis of raised alerts can be improved, it is useful to know what kind of attacks are currently performed and what patterns trigger such an alert. The following section explains the background behind attacks and depicts details of different kinds of network attacks.

2.2.1 Motivation

In order to identify attacks, it is important to know why they are executed. Furthermore, to find the root cause of an attack, it is useful to know which actor initiated the attack. There are a couple of reasons which are of importance nowadays, and present the highest risks:

- An attacker might want to have some financial gain out of his actions. Examples of this type include the spread of *ransomware*, that persuades victims to pay a certain amount of money in order to get their files back.
- Some attacks are performed to gather confidential data, for example from a competitor (industrial espionage).
- Attacks may be performed in order to achieve widespread disruption of services at a certain supplier.
- State actors could target vital systems in other countries to disrupt normal life, e.g. causing power outage (cyber-warfare).

The above cases are most likely to be performed by an external attacker. On the other hand, there exist attacks which are motivated by for example anger towards one or more people. This gives rise to so-called *internal attacks*, where the attacker belongs to, or has belonged, to the target. In most cases, the attacker then has access to a number of devices and might have some privileges. During such attacks, there is no real intrusion into the network, and instead only behavioural patterns might indicate wrongdoing. In most cases, these attacks are mostly destructive or disruptive. The attacker does not need financial gains, but wants the target to suffer losses.

Examples of internal attacks include those initiated by disgruntled employees. In 2006, an employee of UBS was convicted for planting a so-called Logic Bomb, a piece of (malicious) software that is triggered when certain conditions are met, in the company's network. By doing so, about 2000 machines were rendered unusable. His motivation was that he thought he did not receive a high enough bonus. According to Verizon, about 25% of cyber attacks they had information about had been initiated from an internal actor [30].

2.2.2 Network Attacks

There are a variety of different attacks, with effects ranging from a minor disruption to full destruction of data and services. Now, the attacks which currently pose the highest risk are identified and explained.

(Distributed) Denial of Service

Among the most destructive attacks belong the (Distributed) Denial of Service, or (D)DoS, attacks. These attacks are extremely difficult to protect against, because the purpose is not to invade into a network, but to disrupt the systems so that normal operation becomes impossible, involving the destruction of machines in some cases. Examples of methods include:

Reflection attacks Reflectors are defined as hosts that always reply with a packet if they are sent a packet [20]. Attackers can misuse this property by sending large volumes of requests to such machines, after which the reflected answers are sent to the targeted machine.

Amplification attacks An advanced form of reflection that uses a property found in some protocols that drastically increases the response size.

Protocol Exploit attacks Attacks that exploit properties which cannot be avoided due to the nature of a protocol. An example is the TCP protocol, which involves the three-way handshake (SYN, SYN, ACK) to start a connection. The essence of this attack is that a large amount of SYN-packets are sent by the attacker, but are never acknowledged. The server waits for the acknowledgement, and in the meantime its buffer is used up. Eventually, the server will be overloaded and stop accepting new requests.

The key feature of DDoS attacks is that they flood a network with large volumes of traffic. The difficulty of resolving such an attack lies in the ability to distinguish legitimate requests from malicious traffic.

To reach the desired amount of traffic, a pool of hosts needs to be set up, called a *botnet*. Machines belonging to a botnet, also called bots, are usually infected personal computers, and users generally do not detect misuse of their systems. The bots are connected with a controller, operated by the attacker, that sends commands to the bots. With the rise of Internet of Things (IoT) devices, the size of botnets has grown, intensifying the volume of traffic used in attacks. The latest DDoS attacks (February 2018) amounted to up to 1.7 Tbps of traffic being sent to a network by means of a reflection attack. [17]. To enable reflection attacks reaching the targeted system, attackers spoof requests by inserting the target IP address as the sender of the packets. In that way, the reflected data is sent towards the targeted machine. Outlines of (reflection) DDOS attacks are shown in Fig. 2.2 and Fig. 2.3.

ISPs play an important role in the possibility of launching an amplification attack. In most cases, the spoofed IP address belongs to a server which is under control of a different ISP. However, back in 2000, Ferguson and Senie proposed a technique called *network ingress filtering* to counter spoofing of IP addresses [12]. It works by checking whether the source of the IP packet is part of the network managed by the ISP, blocking traffic if it does not pass this simple test.

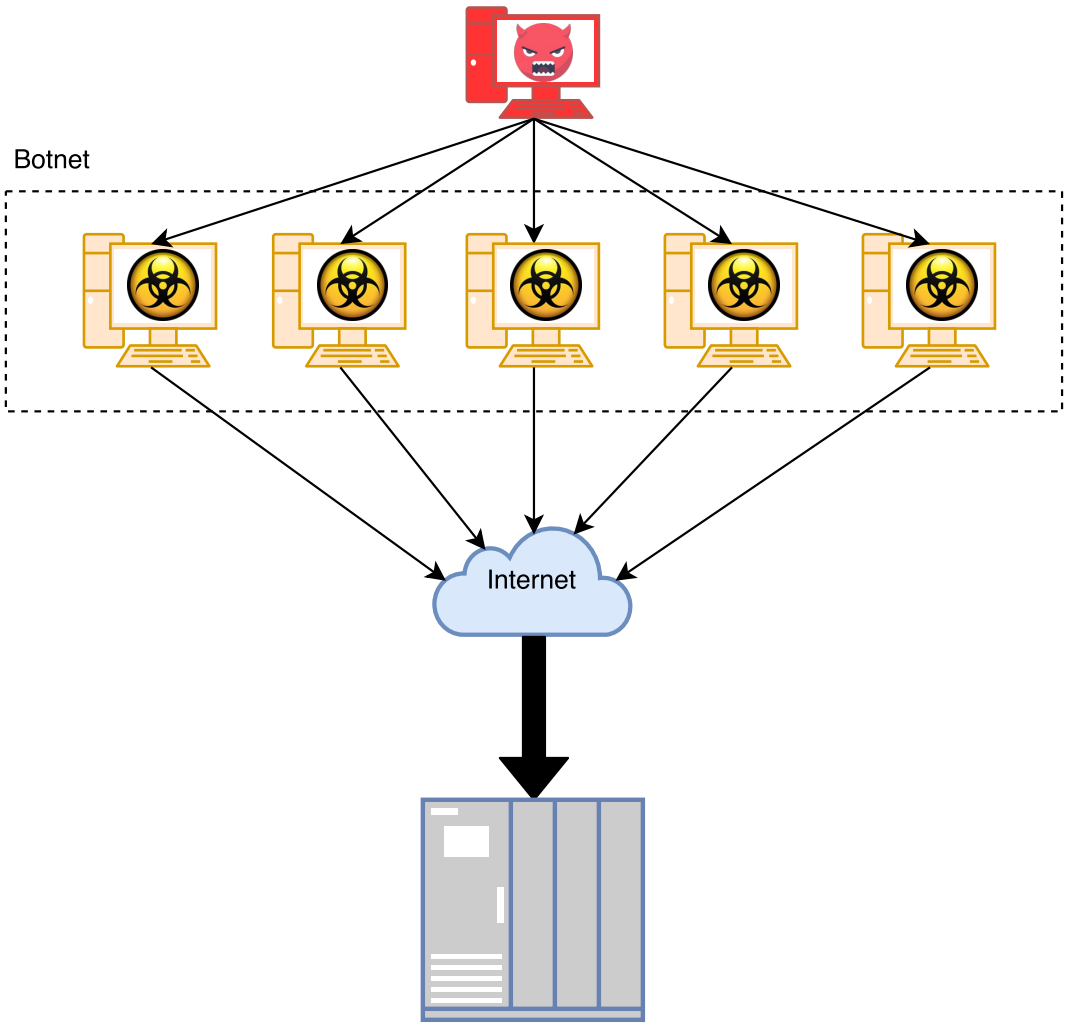

FIGURE 2.2: Simple outline of a DDOS attack. The attacker has control over a number of bots, which he commands to send large volumes of data to a single target.

Espionage

Another kind of attack is motivated by interest in confidential information. Instead of sending huge volumes of traffic to a target machine, this attack is carefully set up. The attacker tries to find some vulnerabilities which can be used to infiltrate into the target network. A common used technique to find the vulnerabilities is to perform a port scan to identify which applications and protocols can serve as a starting point towards attacking a system.

To be able to control the targeted machine, the attacker may set up a reverse shell connection to their own machine in order to transfer acquired data. Another approach is installing key loggers, which may be used to steal authentication credentials, so that the attacker himself may login into secure systems.

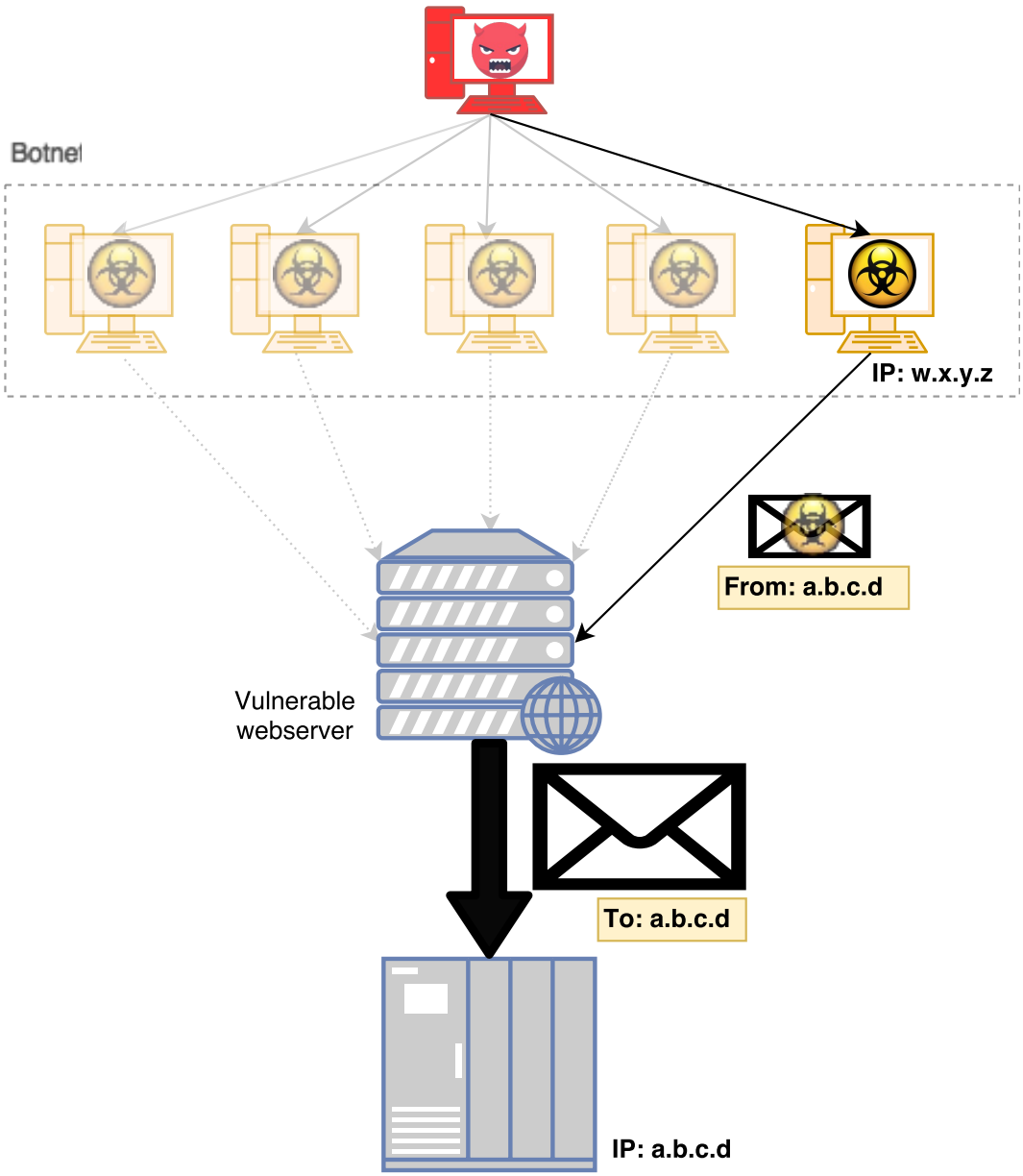

FIGURE 2.3: Outline of an amplification DDOS attack using a spoofed packet. The vulnerable server receives a package it beliefs it got from the target. However, one of the infected bots spoofed the target's IP address. The server sends a larger response to the target device, using up some of the bandwidth.

Phishing

One attack group with widespread effects are phishing attacks. These attacks can be easily deployed and can be targeted at anyone. The main goal is to steal credentials, or other secret information. It is one of the attacks that are based on the idea of social engineering, in which users are tricked into thinking that the communication performed happens with a trusted party. Phishing attacks are mostly carried out by sending e-mails to staff containing links to websites that look like the original one. In reality, these websites are set up by the attacker, and all information sent can be acquired by the attacker before sending the data to the real website. By doing so, the end user might not notice that their data has been stolen. By stealing credentials, attackers could provide themselves access to systems within a network, and might thus be able to get hold of confidential information.

2.2.3 Sequence of Events

A typical attack consists of the following stages:

Reconnaissance In this phase, the attacker starts to search for vulnerabilities in any of the systems on a network. This typically involves a port scan, enumerating all available ports on a target machine, and checking which applications reside there.

Intrusion The second phase involves the real intrusion into the attacked system, using a vulnerability found during the reconnaissance phase.

Insertion In order to control the targeted machine, the attacker needs to insert some file, like a worm, virus, root-kit or any other type of malware, to gain control over the device.

Extraction An attack is most likely motivated by the retrieval of certain data. During the extraction phase, the attacker tries to retrieve the data they are looking for, and copy it to another machine.

Clean-up Finally, the attacker wants to make sure that no traces are left of his intrusion. In the final step, they destroy any evidence that could point towards them. This makes it more difficult to find the cause and initiator of the attack.

There exist different ways to identify intrusions for each of the stages. The reconnaissance stage typically uses port scans, which generate traffic to one or more servers and ports within a short period of time. This traffic can easily be identified by an IDS and will most likely trigger an alarm. When uploading certain malware, the files come with a signature that can be easily identified in the case of a well known piece of malware. Again, an alarm is raised stating that some form of malware has

travelled over the network from A to B. Finding these patterns for each of the stages can help build rules for an IDS.

The outlined sequence of events can be illustrated using the Diginotar case as an example [22]. Diginotar was a provider of digital certificates for various governmental parties in The Netherlands, and was recognised as a Trusted Third Party by multiple vendors. A security breach meant that rogue certificates were issued, which lead to a man-in-the-middle attack on Google-credentials.

The point of entrance for this attack were two servers that were running outdated software vulnerable for know exploits. It is likely that the attacker found these entry points using a scan. This marks the *reconnaissance* phase. When the attacker found the vulnerabilities, he found a way to enter the systems and get into the target network, thus performing the *intrusion*. In this specific case, the attacker then had to find their way into another segment of the network. This may be regarded as another reconnaissance step, which was used to find a way into the secure part of the network.

The *insertion* phase started when the attacker had access to the secure part of the network. They crafted a script that was capable of creating new rogue certificates, which they then used to initiate a Man-in-the-Middle attack. In this particular case, the *clean-up* phase may have been composed of the removal of some access log files, which may have indicated the intrusion. However, a notable feature of this incident was that the attacker left a *signature* in the form of comments in a configuration file, detailing his actions.

2.3 IDS Data Analysis

Data generated by an IDS can consist of multiple log files, each of which contains knowledge of a certain part or protocol used on the system. Combining this data is essential in finding the cause of an alert or when gathering forensic evidence. In recent years, multiple ideas on how to process and interpret data have been introduced. This goes along with the rise of the field of Data Mining and Machine Learning. Below, some of the ideas presented are introduced.

2.3.1 IP Flows

Network logs often contain information about which machines are connected. Not only is it interesting to see which machines are connected, but also *when and how long* machines are connected. These details might show a possible intrusion in the network. To structure this data, the concept of IP Flows was introduced. An IP Flow is described in the IP Flow Information Export (IPFIX) protocol as a set of IP packets that are captured in the network within a set time interval, sharing a set of

common properties [7]. Using these flows, one can aggregate multiple records into a single record describing activity over time and export this for later analysis. A widely known implementation of flow export is Netflow, which was developed at Cisco [6]. Not that IP flows only consist of metadata about the connections.

Monitoring flows consists of four stages [13]:

Packet Observation Collect the packets, and apply some pre-processing steps. This might include timestamping and sampling.

Flow Metering & Export Aggregation of packets into flows happens here. Descriptive properties of the flow are attached. These include but are not limited to: source and destination addresses, protocol identifier, number of packets in the flow.

Data Collection A Flow Collector handles the flow data and performs some further processing on them, like anonymisation and compressing.

Data Analysis The flows are analysed to find evidence of threats and to monitor the performance of services on the network.

The final step, data analysis of flow data, is of most particular interest for intrusion detection. The data obtained can be used in two ways:

1. *Forensic analysis*: Create an overview of who communicated with whom, at which time and so forth. This can be used as evidence for certain behaviour.
2. *Model creation*: The other way around, flow data can serve as a model of anomalous network behaviour. This can then be fed to the IDS.

In short, aggregating traffic information from network data can help give a better understanding of traffic patterns through the network over time. Besides, the size of data to be analysed can be lowered without losing a lot of information.

2.3.2 Graph-based Approach

As shown in Section 2.2, attacks go through a series of phases in sequence through time. To find the root cause of the attack, one has to traverse through the sequence upward. A way to model this sequence of events is by generating a graph. Wang and Daniels have described an approach on handling the raw network data for forensic analysis [32], specifying IDS data as a main source of information for forensic analysts. They classify evidence generated from data in two categories:

Primary evidence Direct indications of attacks or violations.

Secondary evidence Information that does not on its own represent attacks, but could in certain contexts be an indication of malicious activity.

Before analysing the data, it has to be normalised. While network traffic data is considered, it is easily deduced which features play a key role in the raw data, namely:

- Source/Destination IP
- Protocol
- Source/Destination Port
- Timestamp

Besides these base features of network traffic, features special to alert formats in IDS may be introduced. In the case of Netflow [6], for example, it would be likely to incorporate all properties of a flow.

Wang and Daniels also propose to introduce *hyper alerts* to aggregate raw data [32]. These hyper alerts then correspond in a one-to-many relation with raw alerts. It may be useful to append a mapping from the raw event data to the hyper alerts.

Aggregation of the raw events into hyper alerts is in essence based on alert correlation as proposed by Valdes and Skinner [29]. Several features are extracted from the raw alerts to classify in which hyper alert they must be contained. Features include: the address pair, the attack class of the alert, and events that occurred within a certain set time window.

Graph Model

Wang and Daniels describe the model of the evidence graph [32] as a quadruple $G = (N, E, L_N, L_E)$ with:

N	Set of nodes, describing a single entity in the network.
E	Set of directed edges, describing a piece of forensic evidence gathered.
L_N	Labels that indicate the attributes of nodes.
L_E	Labels that indicate the attributes of edges.

There are different levels at which entities can be described. In most cases, it is useful to model an entity as a single host or device in the network, as this can be identified quite simply (IP or MAC).

Some properties of nodes and edges describe their significance within the analysis. They are as follows:

Node Value A value $0 \leq v \leq 1$ describing how important the node is as an asset within the network. For example, a web server with the task of hosting a website within the DMZ has a lower value than a database server with corporate data in the private network.

Edge Weight A value $0 \leq w \leq 1$ representing the seriousness of the evidence.

Edge Relevance A three-valued descriptor of the relevance of the evidence, being either non-relevant or false positive (0), non-verifiable (0.5), or relevant (1).

Edge Host Value Binds the importance value of the edge to its connected nodes. Displays the maximum value of the node values it is connected to: $V_e = max(V_{src}, V_{dst})$

Based on the above information, a graph can be constructed showing each of the properties defined. For each of the edges, a priority score $p(e)$ is calculated, as a product of the weight, relevance and host value of edge e. This allows for a quick analysis of all the evidence gathered, by conveying which information is most relevant for the attack.

A typical issue that comes up with IDS is the high volume of alerts. Analysts are thus presented with the task of properly assessing which alerts are worth looking at. Problems that might arise from generating a graph with incorrect values for evidence priority and node importance is the neglect of some evidence. It might well be possible, in this case, that the overlooked evidence is of vital importance, although obtained from a lesser prioritised system.

2.4 Conclusion

This chapter introduced the main theory behind intrusion detection systems and the way they are used. Furthermore, some of the attacks performed on a network have been outlined. Finally, a number of analysis techniques that can be used to build IDS have been shown. With this theory, the next sections will try to delve deeper into what data is generated and how this data can be transformed in such a way that an analyst can quickly grasp the background behind an alert.

Chapter 3

Current Situation

This chapter deals with the observations done within a single working environment of SOC analysts. The first section deals with the workflow that is currently in use. Section 3.2 shows the various sources of information available and how the data they deliver has to be transformed for later usage. Finally, Section 3.3 deals with the use cases that will be looked at throughout the rest of the research process, based on the acquired experience.

3.1 Current Workflow

To understand the way in which analysts can be supported, it is useful to know the approach of handling an alarm. Therefore, an analysis of the processes that are performed in the SOC was carried out. The SOC employees are grouped into multiple tiers that play a role at different moments in time. When an alarm is triggered, a tier-1 analyst has to start a process of triage in order to identify the seriousness of an alarm. The triage consists of checking the information that accompanies the alarm, after which the decision is made whether or not the alarm is a false positive. The triage is thus a vital step in the process, as an incorrect analysis could lead to serious breaches. It thus requires focus from the analyst.

To be able to assess the alarm, the analyst gathers information from multiple sources, which is a time consuming operation. This introduces a clash of interests. On one hand, the analysis has to be performed as precise as possible, while on the other it also has to be performed within a short period of time. Because gathering information from multiple sources is time-consuming, an improvement can be made here.

Based on the triage, there are multiple follow-up steps. The first one is that the alarm is ignored, either being irrelevant or being a false positive. A false positive means that the alarm triggered by the IDS really should not have happened based on the evidence presented. An irrelevant alarm means that the IDS itself has correctly triggered the alarm, but that the alarm can be ignored given the circumstances defined by the client. This difference can be vital in later analysis of the alarm.

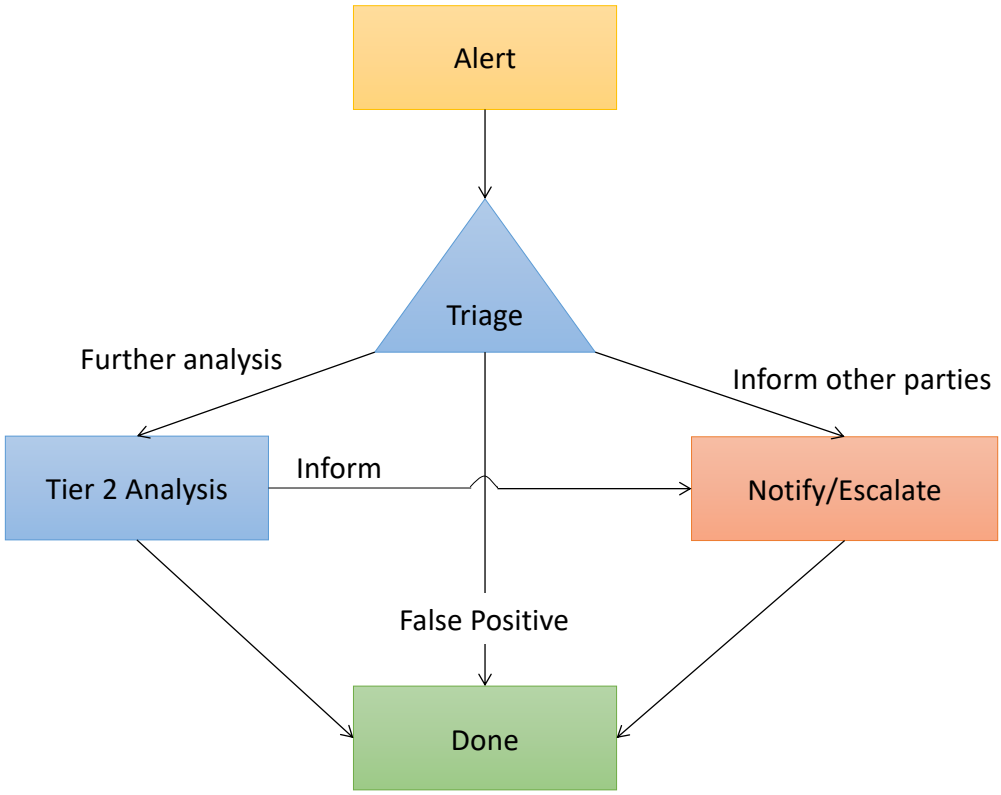

FIGURE 3.1: Overview of the phases when an alarm is triggered

Secondly, there is a possibility that the tier-1 analyst cannot capture enough information to make a decision. In this case, the alarm is passed on to a tier-2 analyst, who can then spend more time gathering all necessary information and make a well thought decision.

The last option is that the alarm is handled as a true positive. In this case, the follow-up steps are based on the importance and risk of the alarm. The possible steps include notifying the client up till escalating directly and handling the alarm as a crisis.

3.2 Data Sources and Preprocessing

While the raw data may provide detailed information of what is going on, it is useful to pre-process the data for further usage. This might mean that some information is lost along the way. However, for analysis purposes, it is important that all the data can be processed as if it were the same. So formatting the data to a predefined model is more consistent. In order to create a conforming model, we first identify the data sources.

3.2.1 Data sources

There are a couple of data sources that could be used in a SOC. We discuss the various types of data sources, and how they can be combined to create an alarm. Some examples for each of the sources are given.

System/Application Log Files

The main source of data for security analysis are log files. In most cases, these contain the raw events that happened while using certain applications. They provide the detailed low-level performed tasks, as well as errors or warnings when illegal operations are performed. Likewise, certain security applications, like anti-virus software, may log actions they perform on untrusted applications and possible malware samples. This class of data is typically host-based, thus providing information about a single machine. Collectors of various log files include the Windows Event Log on Microsoft systems, and syslog on UNIX-based machines. From these, one can check on login attempts, file accesses, and errors that occurred when operating the machine.

Network Traffic Logs

Besides host-based log files, another important source of information are the network logs. These contain information about all traffic from and to a network, ranging from protocol checks (DNS, FTP, HTTP) to application level logs (MySQL) and certificate information. All this information combined in a proper way can give a detailed insight in the processes that take place in a network. An example of a network log collector and analysis tool is Bro, first described by Paxson in 1999 [21]. Bro consists of two elements: an event engine that processes low level data into higher level events, as well as an interpreter that can provide handling of events.

Another source of information for network traffic are firewall logs. These may provide insight in blocked connection attempts, and size of data flows. For anomaly based detection, mainly the latter comes in handy, while patterns may be observed on a day-to-day basis. Anomalies in sizes of traffic might indicate malicious activity. As discussed in Section 2.3.1, network traffic can be aggregated to create a high-level overview of connections and data. This can also provide information to an analyst in how data has been flowing to, from and through a network.

Vulnerability scans

Information that is of vital important when assessing risk of exploits is knowledge about which vulnerabilities are present on assets in a network. Scanning assets the

way that attackers do in order to find gaps in security is useful in finding and resolving the problems. Furthermore, data about lingering vulnerabilities on systems in a network can provide information to the analyst as to what might have caused an intrusion in the first place.

In addition, an analyst may be able to check whether an asset is vulnerable to a certain attack, when an attack pattern is found to be performed on an asset. For example, when an asset is running a UNIX-like operating system, attacks specific to the Windows platform will not be effective. Combining this knowledge may assist the analyst in resolving such an event quickly.

Examples of vulnerability scanners are Nessus and its open source descendant OpenVAS [18]. Both can be used to scan systems for misconfiguration, vulnerable application versions reachable at open ports or lingering default passwords. The purpose is to mimic actions an attacker might perform, in order to be able to harden the system in advance. A report of the found problems can be produced as a source of information for analysts and system engineers.

Open Source Intelligence

Besides the data sources that are within a network described above, there are some additional helpers which may be used to gather information. For example, when presented with a file hash, one may check whether this hash is known in any antivirus database. Open source intelligence platforms exist which gather information presented by the community about new exploits or indicators of compromise (IoCs) that can be used to assess alarms.

A number of tools that may be used by analysts currently include the following:

VirusTotal When presented with a suspicious file, the analyst can send the file, or an URL, to VirusTotal, which then returns whether the file is classified as clean or not in a number of virus scanners [31].

Threat Intelligence services To check whether a device is known to be hostile, an analyst might turn to data available through threat intelligence services, which are mostly fed by members of the information security crowd.

Web sandboxes Sandbox for requesting a specified URL. These tools can show an overview of the content on the page, the redirects that occurred, and information on the location of the server. An example of such a tool is *urlscan.io*. [28].

In addition, analysts use tools to find out to which domains and companies IP addresses belong by using Border Gateway Protocol and Who-is data, which are publicly available. It would be useful to do these checks automatically instead of the analyst needing to perform the checks manually, while it would speed up investigation of alerts.

Event correlation

Considering all data sources on their own may provide a description of the events that happened, it may be more useful to combine them all, and infer events based upon that. Event correlation is used to combine these low-level events, that themselves might not indicate wrongdoing, but together may show a possible risk. For example, consider an event where a login attempt is made, but fails. These attempts on their own may not indicate a malicious attack. When the correlator sees 200 more failing login attempts, it might indicate a brute-force attack taking place on the account. Then, in the network logs, a log entry shows that the login attempts have taken place from a location out-of-office. The event correlation engine may then conclude that this pattern is indeed malicious, and should be taken care of. Combining information from multiple sources may thus indicate an attack with higher risk than what the events on their own might have indicated.

SIEM

We can combine all of the above into one single package, which is called a SIEM, or a Security Information and Event Management tool. This tool can be used to collect all the different log data, perform correlation on the logs, and raise alarms to analysts. Common examples of SIEM software include Alienvault [19], Splunk [26] and ArcSight [1]. They provide an environment to perform asset management, compliancy tracking and alarm generation.

A SIEM is likely to be the first platform an analyst depends upon when receiving an alarm. Based on the information they find to be linked within the alarm, further analysis is conducted into what may have caused the alarm. For that, they mostly utilise the logs that have been described above themselves, because they provide the best context.

Analysis of SIEM alarms

The problem that is currently noticed in the SOC lies around the fact that information that is needed to analyse an alarm generated by the SIEM is spread through all the different log sources. Furthermore, the alarm only shows the events it is directly based upon. In some cases, information about what happened just before or after the events is of importance as well. This information can typically be found in the various log files named earlier. The goal is to provide this information to the analyst given the alarm, so that the analyst does not have to search through the different log files themselves to find indicators of malicious or otherwise remarkable behaviour.

The data we can gather from the SIEM and that we can use as pointers for searching the data are the following:

Timestamp The timestamp of the alarm and the underlying events that were correlated to generate the alarm.

Source and Destination The source and destination involved in the events that generated the alarm. This includes the IP addresses and ports that sent or received traffic.

Rule signature The signatures of the rules that generated the alarms. This may be based on the rules described in Section 2.1.3. In some cases, the rules triggered contain information to which exploits are involved, and what applications are targeted.

Past Events: Issue Trackers

Besides checking the context of the current event, it may be useful to look at similar cases that have happened in the past. There are popular software tools for tracking issues in software projects, but can also be used when tracking IDS alarms. Popular tools for tracking issues are JIRA and Redmine. Tickets from these issue trackers provide information about who handled past events and how previous cases of a certain alarm have been resolved. Also, it is useful to see who has knowledge of a certain type of attack. Data from issue trackers can thus provide further details to an alarm, and pointers about how to deal with an alarm accordingly.

3.2.2 Preprocessing

All of the above explained sources define their own logging language. It is thus useful to transcode all the data into a single format for analysis.

At Northwave, Takken has proposed a XML format that is able to add semantics to any kind of data, called EDXML [27]. EDXML (Event Dataset XML) has the advantage that it can handle any information, but in a structured way. Also, being based on XML, it is easy to query and also traversable.

Although other options for preprocessing data into a structured set may exist, we proceed with this option as it is already available in the environment in which the research is conducted.

The building blocks defined within EDXML require some background information. An EDXML file consists of one or more *ontologies*, which describe the layout of the underlying events. *Events* are the main data model within EDXML. An event is described to be *"an environment providing coherence and context for a group of one or more objects"* [27]. Events are grouped by event type and event source into an *event group*. An event type could be, for example, a phone call. The event source might be specified as the object or instance that created the event.

Noteworthy is that the ontology defines what information is present in the underlying event groups and events. Besides, it gives meaning to the data included in the events, the semantics. This allows for machines to interpret the data. In Chapter 4, a more detailed explanation of an ontology and the usage of EDXML are given.

3.3 Use Case

Based on the observations in the SOC, two distinct events are being looked at in the remainder of this research. The workflow analysis described in Section 3.1 was extended with checking how a Tier-1 analyst processed alarms. The analysis involved with over-the-shoulder monitoring of the Tier-1 at duty, by checking which actions were performed when analysing an alarm. This analysis has been extended by hands-on experience as a Tier-1 analyst by processing alarms myself.

Case: Phishing

During the workflow analysis, I found that the alarms that cost the most time are related to phishing. First, the information given by the alarm had to be processed, and involved objects have to be identified. The IP address belonging to the host suspected of phishing that was noted in the alarm has to be investigated. First, a scan of the URL was made to check what type of web page was accessed. Then, to identify whether it indeed was a phishing attempt, the analyst had to find out how the connection was established, and what redirects were made during this process. Finally, he had to check whether there had been an exchange of confidential data to the other server. This requires looking in multiple logs to establish which protocols were used, to which sources, and how much data was sent during the connections. Checking for all this information requires opening multiple sources, and switching context a lot of times. This takes up a lot of time for the analyst.

To illustrate such a case, consider the following: an alarm is triggered on the rule signature below:

```
alarm tcp any any -> any 80 (content:  "bank.com-")
```

This rule may trigger when a user visits a website using an URL that they think is for the website *www.bank.com*. However, the signature states that the URL is not finished, due to a dash being appended to the domain. This may be an indication of phishing, while the user may be tricked into visiting a different website than was intended. For example, the resulting URL might look like *www.bank.com-malicious.com*. The domain that the user is thus sent to is *com-malicious.com*. If the user is presented with a login screen, they might give away credentials. Therefore, in such a case, the analyst would like to know whether an actual connection was made and whether possible confidential data was leaked.

In Chapter 5, we take a look at what further information is needed and how this can be deduced using the information in the alarm.

The option have been chosen based on prevalence as well as the possibility to increase the speed of handling an event. Another option for a use case would have been DDOS attacks, which have high impact on businesses involved. However, due to the low occurrence of these kind of events, and lack of testing data, the decision was made to leave this use case aside.

3.4 Conclusion

In this chapter, we have established a set of information sources that one may find at a SOC. While all of the systems themselves can be used to identify risks and threats in advance, they can act as a perfect starting point when an attack has taken place or when an anomaly has been detected. By combining the different sources of data, one may be able to identify new patterns or lay out the track that has been travelled to get into a system. The next chapter deals with a way of turning the different sources of data into a useful outline of the systems involved in an alarm.

Chapter 4

Ontology, Description Logic and Model Based Concept Mining

The focus of this research is on how to add semantics to raw log data for the purpose of creating an overview of relevant concepts involved in a SOC alarm. To do so, we need to effectively represent knowledge contained in the data. This chapter focuses on the theory behind knowledge representation and the reasoning that can be performed. We introduce the notion of Model Based Concept Mining, and elaborate on the involved objects.

4.1 Ontology

Most information can be classified into a certain domain. Such a domain is characterised by the objects that belong to it, as well as relations between these objects. This is represented by an *ontology*. The following entities typically form the basis of an ontology:

Instances Instances of the concept classes, also called individuals.

Concepts Concept classes to which individuals/instances belong.

Relations Descriptions of how concept classes relate to each other.

Attributes Characteristics that individuals may have.

Events Changes in attributes of or relations between instances.

Figure 4.1 shows how an ontology, concepts and instances relate to each other. Information about individuals can be fragmented across multiple data sources. In order to facilitate analysis of these individuals, it is necessary to integrate data from multiple sources into a single consistent data set and model it using an ontology that defines these individuals. We use EDXML to do so, because this is readily available in the research environment, and is tailored to describe an ontology.

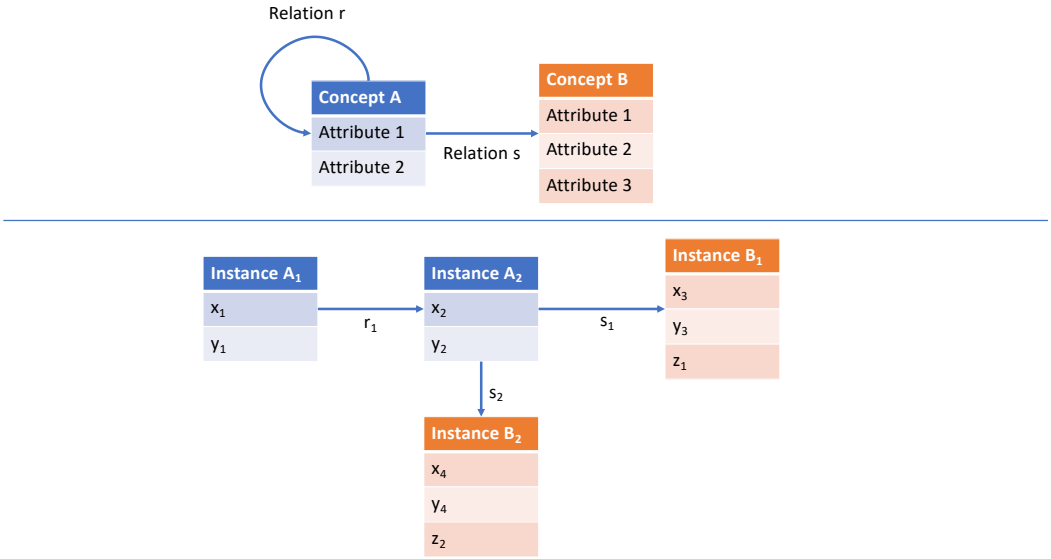

FIGURE 4.1: Depiction of how concepts, their relations and attributes are defined in an ontology (top). An instantiation of the ontology (bottom).

4.1.1 Describing Ontologies Using EDXML

In Section 3.2.2, an introduction of EDXML was given. We will now look at the structure of EDXML as well as how this translates to components of an ontology. Example code can be found in Appendix A.1.

Starting off with events, we have to find a way to capture various kinds of events in a structured way. In EDXML, this is solved by defining event types, showing what kind of properties exist within a single event from a certain source, and what these properties mean. The event types and their properties define the backbone of the ontology. All other entities within an ontology are derived from here.

First of all, a concept can be defined by a certain event property. For instance, an IP address or MAC address define (with some probability) a unique device. In EDXML, this is modelled as an attribute to the property, defining the name of the concept as well as the confidence that the given property implies the concept. The notion of confidence will be further detailed in Section 4.5.2.

Secondly, for each event type, relations between properties defining concepts can be defined. The properties linked are required to define a concept, because a relation can only exist between concept instances. The relations may either be between two different concept instances, so-called *inter-relations*, or within one instance, known as *intra-relations*.

By defining an event type with its relations, semantics can be added to the raw data. Furthermore, a more structured dataset is created, which can be more easily processed. The described ontology can be used to parse log lines into events. In Appendix A.2, an example of a parsed event is given. A log line is parsed into XML, with each of the properties being a tag.

4.2 Description Logic

To be able to reason about specific domains and their contents, a set of rules specially tailored for the domain has to be set up. To generalise this problem, Description Logic (DL) was introduced [2]. This section briefly introduces the main ideas behind DL, and will touch upon the basic syntax and semantics of DL.

As an analogy, one could look at a domain using a network of nodes and links. The nodes represent the different classes, and links show the relationships between classes. Sometimes, a relation itself is modelled as a class to which individuals might belong, e.g. being a parent. The first ideas of such a network were dealt with in the introduction of the KL-ONE representation system proposed by Brachman and Schmolze [5].

4.2.1 Intensional vs. Extensional Knowledge

Until now, we only specified what the domain looks like. However, to understand how knowledge actually can be extracted from the base, we need to look at how the knowledge behaves.

We can split the knowledge contained in the knowledge base into two classes. First, we can state facts about individuals that are always true. For example, a book always has a single identification number (ISBN). This is known to be general information within the domain. We call this *intensional knowledge*.

On the other hand, a instantiation of the domain in the form of an individual might be influenced by the given circumstances. Knowledge might change over time, e.g. when a book is being loaned by someone, the book has its features changed due to the event of loaning. This is known as *extensional knowledge*.

We can thus state that the intensional knowledge is always present in the domain, and extensional knowledge is obtained by the series of events that happen over time.

4.2.2 Handling Unknown Features

In first-order logic, defining a rule makes it either *true* or *false*. If a statement cannot be solved, then it is classified as 'false' under the *closed world assumption*. One of the

key features of DL is that it does not assume a *closed world* [4]. Instead, a variable can actually be classified as **unknown**. This can be translated loosely into the fact that 'it might be the case that an attribute or relation holds, but we cannot guarantee it does'. This is very important when dealing with partial information.

To elaborate on this, it also seems that ignoring the *closed world assumption* is a better way to capture the actual state of the world. When a feature is unknown to the system, normally it would say that it is *false*, although in reality the feature is *true*. Then, when querying the knowledge base, one would be confronted with incorrect information.

For this research, the representation of *unknown* data is of key importance. While analysts would like to have an as much detailed picture of the world as possible, it would be a disaster if incorrect assumptions were to be made about unknown data. Therefore, it is better to just state that the value is not known, so that the analyst is informed properly, and can act responsibly on the data presented.

4.2.3 Representing the SOC Domain

To capture data from the different sources in the SOC, we need to find a representation that is suitable for querying, without having to know the precise data each source has in it. It is therefore useful to find the relations between parts of the data to make the connections between the sources implicit. For example, we could identify a concept class **Computing device**, with a unique MAC-address (intensional knowledge), an IP address and a user logged in (extensional knowledge). We could then derive a relation 'ConnectsWith' to the class itself that defines a connection of two devices. Note that this does not imply that an individual can only make a connection with itself, but that this is possible between multiple individuals belonging to the same class. Note that this also is an extensional relation, as it is influenced by events (connecting and disconnecting).

By representing the data sources in an ontology, we might be able to build a knowledge base of the sources, which can then be queried along the lines of an alarm coming in.

4.3 Extracting Knowledge: Model Based Concept Mining

The problem we are dealing with is the following: an analyst is presented with an alarm which contains a slight amount of information, and we want to assist the analyst by expanding the information presented. Based on the theory described in the previous sections, we take a look at how we can approach this problem. To extract the knowledge, we introduce a new technique which we call Model Based

Concept Mining. This section elaborates on the processes that are performed by elaborating on a small example.

Suppose we have the following situation: an alarm of a phishing attempt is triggered. The alarm contains the following basic information[1]:

Attribute	Value
timestamp	2018-02-24T12:34:56.0000
uid	x
src_ip	12.34.56.78
src_port	5243
dest_ip	222.32.42.52
dest_port	6345
protocol	TCP

An analyst presented with the phishing alarm would typically start off investigating by asking the following:

1. To what devices do the IP addresses belong?
2. What kind of system services are bound to the ports?
3. To what devices has a connection also been made (redirects)? And what are their respective URLs?
4. What data has been sent over the connection?
5. Is the destination known to be a threat?
6. Have there been similar events in the past and how were they solved?

The first logical step is to derive concept instances, or individuals of the domain, from the information in the alarm. After that, we would like to have a representation of the domain instances over the last x minutes/hours. We thus want an overview of everything that has happened in the near past.

The first question can be represented in a query to the knowledge database asking for the different attributes known to the instance whose IP address is given. We might be able to identify the current user, current host-name and MAC-address of the device. This might give information about the risk that is involved with the current alarm.

The next section dives into the subject of how we can represent the given event, and how we can extract the right knowledge from the created ontology.

[1]This data is fictional, but the structure is based on real events

4.4 Representing Concepts

In order to depict individuals, we need to define a format that can capture the properties outlined in the structured source (EDXML). We also need to show the relations that exist between individuals, based on the information gathered. The solution is to use a graph-based structure. Let graph $G = (V, E)$, with V the set of vertices and E the set of *directed* edges. The individuals are represented by the vertices, which contains additional information based on the events that captured the individual. The relations can then be represented by edges connecting the different individuals.

4.4.1 Vertices

As explained, the vertices in the graph represent individuals that have been extracted from the dataset. These individuals have been defined by properties that according to the ontology describe a particular concept. Thus, each vertex can be decorated using the information that created the individual. For example, an individual of the concept "Computer" may be described by an IP address x. We may then denote this vertex as $Computer(IP = x)$, showing how the node was derived.

4.4.2 Edges

The edges in the graph show the relations that exist between or within the individuals. EDXML provides three types of relations that can be distinguished. First, an event may contain intra-relations, meaning that the properties are shared within an individual. Secondly, there exist inter-relations, which relate properties of two different individuals with each other. Finally, we denote all other relationships in a separate class. These relations do not necessarily link one of two individuals and their properties, but might give extra context.

Relations in the ontology may be either directed or undirected. For example, a connection attempt from A to B gives a directed relation from source A to destination B. On the other hand, a relation linking an IP address to a device works both ways. Because building a graph with both directed and indirected edges might pose problems, and we want a directed graph, we add a translation step in describing indirected relations. We represent this type of relation by two directed edges.

4.5 Mining Concept Instances

From the set of nodes and edges, we need to define the concept instances. A concept instance can be described as a set of nodes sharing intra-relations between properties

that define the same individual. More formally, we define what nodes share intra-relations, and how this translates to a concept instance.

Definition 4.5.1 (Intra-reachable). Let $G = (V, E)$, and nodes $N, M \in V$. Node M is intra-reachable from N if and only if there exists a path $\pi(N, M)$ and all edges $\{E_1, .., E_n\}$ on that path are intra-relations.

Definition 4.5.2 (Concept Instance). Let $G = (V, E)$, and $N \in V$ being a node representing an individual, used as seed. Concept Instance $\mathcal{CI}$ is the set of nodes $N_I \subseteq V$ which are intra-reachable from N.

Important to note is that intra-relations can only connect nodes which represent individuals of the same concept. Thus, the set of nodes in N_I described in Definition 4.5.2 all denote individuals of the same concept.

As a consequence of Definition 4.5.2, it is trivial that it does not matter which node of the instance is used as seed, while this node should be intra-reachable from all other nodes in the instance. So any node in a single concept instance yields the same concept instance when used as seed.

4.5.1 Describing a Concept Instance

We have seen that a concept instance can be described by a set of nodes that share intra-relations. We would like to have an instance being defined by a set of properties in order to combine information and reduce the amount of nodes. The provided definitions can be used to extend the notion of a single concept instance and the attributes defining it. The intuition is that an instance is defined by all the features that are reachable through the intra-relations. We can thus provide an alternative representation of a concept instance, denoting the feature set and the values therein. Let $\mathcal{I}$ be an individual of concept t with seed $x_1 = v_1$, denoted $\mathcal{I}_t(x_1 = v_1)$. For each intra-reachable node $n_i \in \{n_0, .., n_m\}$ that describes an individual $\mathcal{I}_t(x_i = v_i)$, we can join the individuals, yielding the concept instance described by all available attributes: $\mathcal{CI}_t(\{x_1 = v_1, x_2 = v_2, .., x_m = v_m\})$.

TABLE 4.1: Example event of a connection between two objects

Property	**Value**
timestamp	t
source-ip	x
destination-ip	y
source-hostname	pc01
destination-hostname	pc02
source-macaddress	a:b:c:d:e:f

Suppose we are represented event depicted in Table 4.1, and an ontology describing which properties define individuals of concepts shown in Table 4.2. We can describe

TABLE 4.2: Ontology excerpt for a connection event

Property	Defined concept
source-ip	Computer
destination-ip	Computer
source-hostname	Computer
destination-hostname	Computer
source-macaddress	Computer

Relation	Type	Properties
hasHostname	intra	source-ip $\leftrightarrow$ source-hostname
hasHostname	intra	destination-ip $\leftrightarrow$ destination-hostname
hasMac	intra	source-ip $\leftrightarrow$ source-macaddress
connectsWith	inter	source-ip $\rightarrow$ destination-ip

the following list of individuals:

$$Computer(\textbf{Source-IP} = x)$$
$$Computer(\textbf{Destination-IP} = y)$$
$$Computer(\textbf{Source-hostname} = pc01)$$
$$Computer(\textbf{Destination-hostname} = pc02)$$
$$Computer(\textbf{Source-MAC} = a:b:c:d:e:f)$$

Based on the individuals and the relations that are found in the ontology, we can generate the graph for this event. The graph is depicted in Figure 4.2. The intra-relations have been drawn as bidirectional arrows. The next step is to derive the concept instances from these individuals. We follow the described process of combining all nodes that are connected with intra-relations, and merge the individuals with their attribute values. This yields the graph shown in Figure 4.3. We are left with two concept instances of type **Computer** connected through a relation *Connects with*.

4.5.2 Confidence

While some of the relations are not guaranteed to exist, we need to find a way to model uncertainty in the graph. Furthermore, some of the properties defining individuals might not be unique. For example, an IP address may exist twice within different sub-networks, resulting in the fact that two different devices may have the same IP address. One way to solve these problems is by defining a confidence factor for each relation and for each property defining an individual. This confidence value defines the probability that two concepts are equal if the property is equal. As the confidence level describes a certain probability, this value must adhere to the range $[0..1]$. This means that when a property receives the confidence value 1, concepts with an equal value for this property are the same.

The confidence value plays an important role when combining and merging different instances. When combining, it is important that it is very clear that there are assumptions made based on the information presented, and that there is a chance that the merged instance is not really there. We look at two important cases where the confidence factor plays a role.

Within this research, the domain is manually created by a domain expert. That means that the expert also decides what confidence value is linked to a property. This in turn means that a change in the domain specification may have a big influence in the outcome of evidence. For now, we assume that the expert has correctly specified the domain.

4.5.3 Similarity Score

In Section 4.5.1, a concept instance was described as an object made up of a set of attributes which define the instance. As outlined, a conflict may appear when a non-unique value described different concept instances. To solve this conflict, we could either merge the two instances, or make sure that the instances can be distinguished

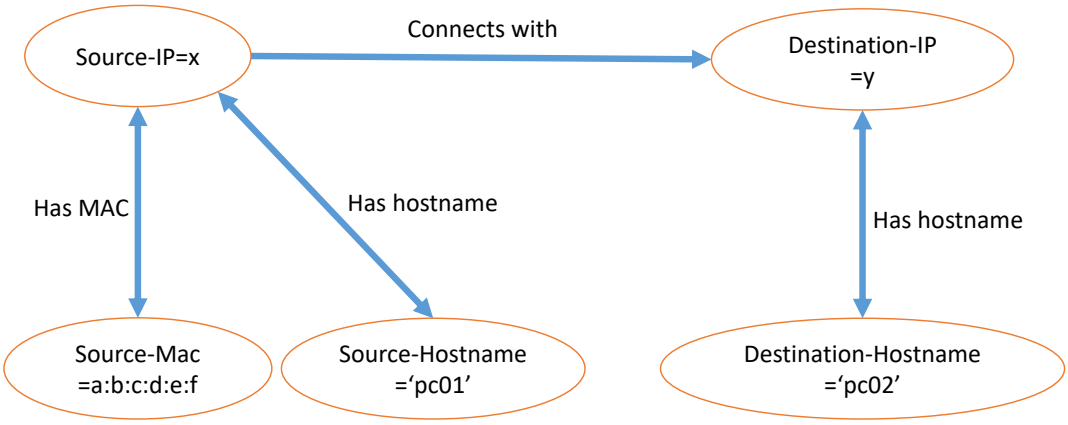

FIGURE 4.2: The graph showing attributes and relations between properties generated from the example event and ontology described in Table 4.1 and Table 4.2

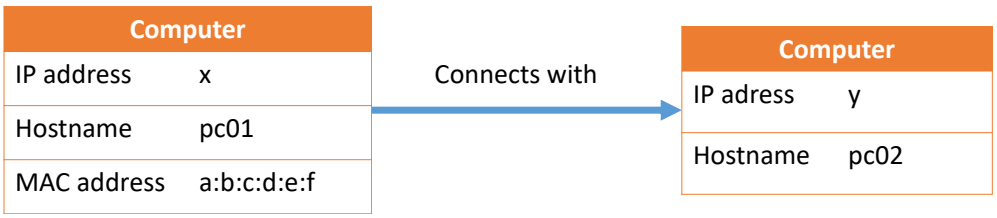

FIGURE 4.3: The concept graph after merging the different nodes belonging to a distinct concept instance. We are left with two concept instances of type **Computer**, each with their own set of attributes.

through other factors. We thus want to check whether the instances are similar or not.

Checking similarity of objects in an ontology has been investigated multiple times [11, 23]. However, in most cases a similarity score over concepts is derived from IS-A relations in an hierarchical ontology. In this case, we are interested not in the similarity of concepts, but in the individuals belonging to a concept. Besides, we do not have strict IS-A relations in the ontology. Therefore, the proposal is to create a similarity score between instances based on their attributes and their particular confidence values.

To be able to calculate the similarity, a confidence factor is attached to each of the properties defining an individual. Given the concept "Computer", individuals may be derived from properties like the IP address, MAC address and a hostname. However, some properties are more detailed than others. For example, a MAC address should be a unique identifier of a device, whereas a hostname or IP address can be found multiple times in different cases. The model should reflect this probability in the confidence value. A MAC address would receive a confidence factor which is close to 1, meaning that when individuals are defined by equal values for the MAC address, they have a high probability of being the same within the domain. On the other hand, when presented with two individuals defined by the same hostname, it is less certain that they are the same. We should base this decision then on the other information that is available in the individual or concept instance.

The similarity then can be calculated using the combination of all attributes present. Given are two concept instances $\mathcal{CI}_A(\{x_1 = v_1, ..., x_n = v_n\})$ and $\mathcal{CI}_B(\{x_1 = u_1, ..., x_n = u_n\})$. We define a weight function $c(x) \rightarrow \{0..1\}$ that maps a property to a confidence value. We define the similarity score $sim(\mathcal{CI}_\mathcal{A}, \mathcal{CI}_\mathcal{B})$ as:

$$sim(\mathcal{CI}_\mathcal{A}, \mathcal{CI}_\mathcal{B}) = \frac{\sum_{i=1}^{n} c(x_i) * f(v_i, u_i)}{\sum_{i=1}^{n} c(x_i)} \tag{4.1}$$

where the function comparing the values for the attributes is defined as:

$$f(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

If the confidence score exceeds a set threshold, the concept instances are merged into one. This means that all the attributes that were available in the separate instances should be combined into one. This may lead to conflicting values for certain attributes. This is solved by keeping track of all values that are merge in a set. As a consequence, the calculation of similarity of attributes has to be adapted to take the prevalence of a single value into consideration. Given that X now represents a set

of values for a given attribute, and $count(y, X)$ gives the number of occurrences of y currently in X, the comparison function changes into the following:

$$f(X, y) = \frac{count(y, X)}{|X|} \tag{4.3}$$

The intuition behind this function is that as the amount of similar values increases, checking a new value will have a higher influence on the similarity than a value that has a low occurrence in the set. Furthermore, the requirement that a value that does not appear does not count in the similarity is satisfied, because the function will then evaluate to 0.

Unknown Values

As described in Section 4.2, we need to have a representation for unknown values for a given concept instance. We currently base the similarity score only on the values that are actually known. The reason behind this is that we also need to satisfy the *commutative* property of merging instances. Thus, the order of comparing and merging instances should not influence the similarity scores when comparing a new instance.

A conflict of interest exists between these two properties. In Section 4.6, an example calculation given a pool of four concept instances for the same concept type is given, based on the similarity score outlined in the previous section. If we were to adapt the similarity function f to handle empty values or unknowns, we might get the following function:

$$f(X, y) = \begin{cases} 0.5 & \text{if } X = \emptyset \text{ or } y = \emptyset \\ \frac{count(y,X)}{|X|} & \text{otherwise} \end{cases} \tag{4.4}$$

This would create an **asymmetric** operation when merging instances based on their similarity. For example, if an empty value is propagated first, and then compared to a new instance which has this value, the calculation would be different from when the value was already there, and then checked against with a non-present value. The choice was thus made to omit unknown values in checking similarity in favour of satisfying commutativity of the operation. Another argument that backs this approach is that checking empty values is involved with added or removed information, but when the other values are similar, this should not influence the similarity of instances.

4.5.4 Unique attributes

In some cases, an attribute can uniquely identify a single instance. For example, every inhabitant of the Netherlands has their own social security number, which is used to uniquely identify a single person. Consider two events that contain the same social security number, we may directly state that they deal with the same person.

In the described model, this could only be captured with the confidence value. If this value is set to 1 for a particular attribute, it means that the probability that two concept instances are the same is 1, if defined by the same value for a unique attribute.

We adapt the similarity score to accommodate this change. That is, if in the set of similar attributes an attribute with a confidence value of 1 is present, we return 1 as similarity score:

$$A = \{a_i | 1 \leq i \leq n \wedge f(v_i, u_i) = 1\} \tag{4.5}$$

$$sim(\mathcal{CI}_\mathcal{A}, \mathcal{CI}_\mathcal{B}) = \begin{cases} 1 & \text{If } \exists x \in A : c(x) = 1 \\ \frac{\sum_{i=1}^{n} c(x_i) * f(v_i, u_i)}{\sum_{i=1}^{n} c(x_i)} & \text{otherwise} \end{cases} \tag{4.6}$$

If all attributes defined in the individuals are equal, we also obtain the value 1, even though these attributes may not be unique. For now, we assume that these situations are similar.

4.5.5 Propagation of Uncertainty

When merging instances, we introduce new levels of uncertainty. We have to make sure that this uncertainty is reflected as we continue comparing the instances, and expanding the domain. The merge concepts into one instance by performing two operations, as outlined before:

1. Merge the attributes that describe the different concepts.
2. Merge the relations that exist from the added concept.

There are thus two possibilities to reflect added uncertainty. We would also like to be able to retrieve the single concepts back from the merged concept if necessary. We can combine both by retaining the old concepts on their own, but storing the similarity score for each of the merged concepts. We then also need to make sure we can trace back the root, or first instance that created the merged concept. By doing so, we can define the uncertainty by multiplying attribute confidence with the similarity score for the given concept. This also enables the possibility to cut off edges from the result graph if the certainty is too low.

4.6 Example

To illustrate the described model, an example is now discussed. We are confronted with 4 events, each generating a concept based on information contained in these events. The concepts all belong to the same concept class, so we can try to combine them. The data extracted is shown in Table 4.3.

TABLE 4.3: Data of four concepts derived from events.

Attribute (x)	$c(x)$	A	B	C	D
MAC Address	0.9	1:1:1:1:1:1	-	1:1:1:1:1:1	-
IP Address	0.8	10.11.12.13	10.11.12.13	10.11.12.14	10.11.12.15
Hostname	0.2	PC1	PC1	-	PC2
OS	0.1	WIN7	WIN7	WIN7	WIN8

We start out with an empty instance pool. The first concept, A, is then added as the first base instance. We do not merge or compare anything.

We then add concept B, by comparing it to the instance in the pool. The following calculation is performed:

$$\begin{aligned} sim(A,B) &= \frac{\sum_{i=1}^{n} c(x_i) * f(v_i,u_i)}{\sum_{i=1}^{n} c(x_i)} \\ &= \frac{0.8(1)+0.2(1)+0.1(1)}{0.8+0.2+0.1} \\ &= 1 \end{aligned} \tag{4.7}$$

The similarity score is equal to 1, because the unknown value in B is ignored in the comparison. We thus decide to merge the concepts. The relations linked to B will now be added towards to merged instance AB.

The next concept to be added is C. We are comparing this to the merged concept AB, which has the same attributes as A. The calculation that to be performed is the following:

$$\begin{aligned} sim(AB,C) &= \frac{\sum_{i=1}^{n} c(x_i) * f(v_i,u_i)}{\sum_{i=1}^{n} c(x_i)} \\ &= \frac{0.9(1)+0.8(0)+0.1(1)}{0.9+0.8+0.1} \\ &= 0.56 \end{aligned} \tag{4.8}$$

We are now dependent on the **threshold** value that is set. The threshold is a predefined value, and greatly influences the way data is aggregated. We only merge

instances if the similarity is higher than the threshold. Suppose the threshold is set to 0.5, the concept instances would be merged, yielding the following instance:

TABLE 4.4: Merged concept ABC, with the set of IP addresses defined in the single concepts.

Attribute (x)	$c(x)$	ABC
mac-addr	0.9	1:1:1:1:1:1
ip-addr	0.8	{10.11.12.13, 10.11.12.13, 10.11.12.14}
hostname	0.2	PC1
os	0.1	WIN7

We include two copies of the same value in the set, in order to show the importance of each of the values when comparing to a new instance. For instance, when a concept with ip-addr $=$ 10.11.12.13 is compared, it will have a higher similarity than a concept with ip-addr $=$ 10.11.12.14, given all other attributes are equal. We show this by merging another concept with the following attributes:

TABLE 4.5: Two concepts to be added.

Attribute (x)	$c(x)$	E	F
mac-addr	0.9	1:1:1:1:1:1	1:1:1:1:1:1
ip-addr	0.8	10.11.12.13	10.11.12.14
hostname	0.2	PC1	PC1
os	0.1	WIN7	WIN7

The similarity scores based on the combined event ABC and events E and F respectively yield:

$$\begin{aligned} sim(ABC, E) &= \frac{\sum_{i=1}^{n} c(x_i) * f(v_i, u_i)}{\sum_{i=1}^{n} c(x_i)} \\ &= \frac{0.9(1) + 0.8(\frac{2}{3}) + 0.1(1)}{0.9 + 0.8 + 0.1} \\ &= 0.85 \end{aligned} \tag{4.9}$$

$$\begin{aligned} sim(ABC, F) &= \frac{\sum_{i=1}^{n} c(x_i) * f(v_i, u_i)}{\sum_{i=1}^{n} c(x_i)} \\ &= \frac{0.9(1) + 0.8(\frac{1}{3}) + 0.1(1)}{0.9 + 0.8 + 0.1} \\ &= 0.70 \end{aligned} \tag{4.10}$$

This calculation continues as the concept is expanded further.

It is trivial that the last concept in Table 4.3, D, will have a similarity score of 0, while all attributes are different from the values in the merged concept ABC. This means we have a pool of concepts $\{ABC, D\}$ after processing the current list. Because an added concept will always only have a single value based on a single event, we do

not consider the case of merging two sets of values during this research. However, the set comparison using union and difference can easily be extended for this purpose.

Section 5.2 further outlines the process of merging instances. While the above process lists the scoring technique generally, there may be constraints per domain which need to be taken care of as well.

4.7 Conclusion

In this section, the background behind concepts is explained. We looked at how to represent a domain within an ontology, and how this can be transferred to depicting raw logs into graphs presenting individuals. In the next section, we outline the full process from raw events to a presentation of the evidence. Besides, some practical issues and how to solve them are discussed.

Chapter 5

Applying Model Based Concept Mining on the SOC Domain

In the previous section, the process of mining concepts has been explained generally. This section deals with the implementation steps, as well as a description for the given SOC use case.

Recall the problem that we are dealing with: an analyst is presented with en alert, which contains minimal information about the event and the involved entities. This mostly means the source and destination IP addresses and the time and type of the event. Instead of letting the analyst need to perform multiple tasks, the proposed solution takes on this work to provide a way to give the analyst enough context to make a justified decision more quickly.

5.1 Methodology

The goal of the process is to generate a graph of the domain that is described in the different log files, each containing its own format. The process from the raw data to the graph can be split into four distinct phases: Preprocessing, Concept Mining, Post-processing and Enrichment. Each of these phases is now briefly discussed.

Phase 1: Preprocessing
In the first phase, we need to align the data into a standard format, which can be further processed to generate graph data. Earlier, a description of EDXML has been given. This first phase is exactly transcoding the raw data formats to the structured EDXML format. The EDXML data describes the ontology of the data presented in the log files, and thus adds semantics and relations to the data. This process has been described in Section 4.1. The domain specification describing the concepts and their relations in this research is created by a domain expert (supervised).

Phase 2: Concept Mining
The second phase deals with the mining of the concepts from the preprocessed data into sub-graphs, as detailed in Sect. 4.3.

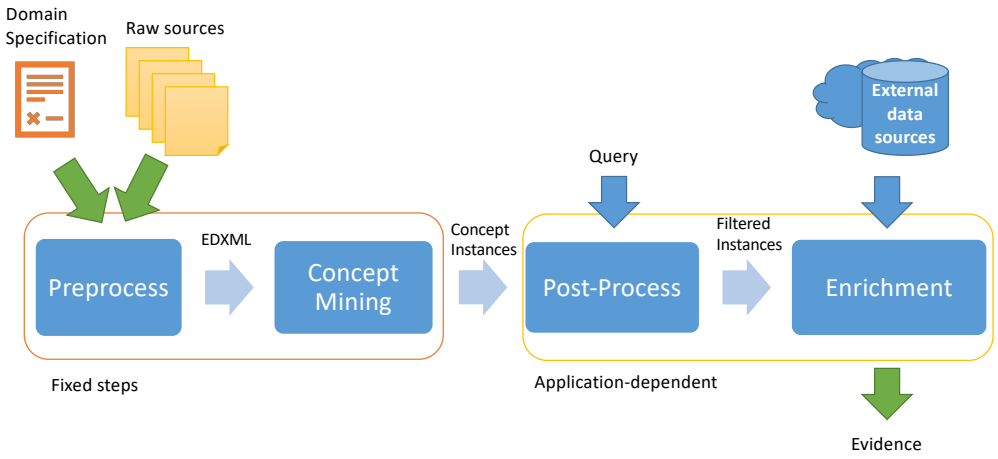

FIGURE 5.1: The different phases of processing raw data into the concept graph.

Phase 3: Post-processing
The third phase deals with processing the mined concepts toward a working state. This may involve grouping, splitting, combining and merging concept instances, or filtering on certain criteria. This also may include an external query.

Phase 4: Enrichment
The final phase can be used to enrich the concept graph with data from external sources.

Globally, the first two phases always have to be the same: retrieve a pool of concept instances from raw data. The last two phases are dependent on the solution that needs to be built. Processing the mined concepts into a data format that is suitable for an application is necessary. Furthermore, different external sources may be used to enrich the data. Figure 5.1 shows the outline of the processing steps.

5.2 The SOC Use Case

The previous section outlined the basic steps that need to be performed to go from raw data to a suitable data format for analysis. This section dives deeper into the specific use case that deals with the SOC processes. We take a look at the constraints of the context, but also at a suitable way of filtering the important data from the whole.

5.2.1 Time Constraints

One of the biggest challenges that come with the analysis of alarms in the SOC is the constraint of time. Most of the events leading up to the alert happen within a short time period before the alert is sent. This also means that at any given moment, the graph representing the raw data that is linked to the alert is different over time. We have to reflect this property in the creation of the graph and the presentation of the data to the analyst. The solution that is presented does not only work for the given domain, but also for other time constraints.

Most of the data sources contain pointers as to when an event happened. For example, a log line in most cases contains the timestamp of the creation of the log. We can reflect this property in the processed form (EDXML). We can then use this property to create time-based instances of concepts contained in the structured data. The idea follows along the lines of combining instances which are closely related in terms of time.

Decoration of Nodes

As presented in Chapter 4, we can represent a single concept instance as a subgraph with a root node or seed defining an individual. We can decorate each node in the graph with extra information, which in this case is the timestamp of the event. This is possible while each node is created from an event, and each event thus contains the timestamp of its creation. In addition, we may also add the timestamp to each of the relations that is generated from the raw events. This may aid in the creation of the edges within the graph.

When presented with an alarm, we want to reconstruct the graph containing all the data necessary, but with the addition that the data must be valid at the time of the alarm. We thus have to look for the instances that were most likely to be present at the time of the alarm. That means, we have to search for instances with seeds that have a timestamp which is in the search range. This range may be different for each type of data. For instance, the validity of a vulnerability scan, which may only take place once a week, spans over a longer period of time than a DHCP lease event, which is more volatile. We have to incorporate these differences in the data as well.

One solution to the latter is to provide a value for each type of source indicating for how long an instance will be valid. That means that a node has to be decorated with not only a timestamp indicating when the event was created, but also a timestamp indicating the expiration of the validity of the instance. By doing so, we can eliminate the instances that are not valid for a given search range immediately. Note however that this might pose risks regarding loss of valuable data.

Building Merged Concepts

In Chapter 4, we dealt with the way how concepts should be merged. To trace back the uncertainty in these merges, we defined that we need to know the root of the merged instance, and generate the graph from there, until the certainty for a relation gets too low. Because we are now dealing with a time-based order of events, we can easily define the event that happened first to be the root of a merged instance.

In addition, merging concepts should ideally be delayed as long as possible. Due to the introduction of uncertainty, the reality may be reflected in a less detailed way. When postponing the merging process, we retain most of the raw data, and thus a more detailed depiction of the real world.

Reconstruction

Consider the following case: we want to have a depiction of the situation from within the time range $[T_{begin}, T_{end}]$. We now go through the reconstruction phase for this range.

Before we can take multiple concepts into consideration, we need to aggregate the instances for a single concept into one, as a representative of that concept instance for the given time range. This means that the length of the time range may influence the amount of instances being combined. The idea of the aggregation process is to merge the similar seeds which all lay within the time range into one single instance using the similarity score, containing all edges present in the unique seeds.

After aggregating the single concept instances, we are left with only one instance for each concept in an ideal situation. We then have to recombine these concept instances based on the inter-relations. While either side of an inter-relation should be the seed of a concept instance, this process is merely substituting the singular nodes at the sides of the inter-relation with its respective concept. This is shown in Figure 5.2.

5.2.2 External sources

Enrichment with external data can be very useful to fill in gaps of information that cannot be found in the processed sources. Most of the data used in the enrichment process originates from open sources. In the specific SOC use case, we can have the following sources for enrichment, most of which have been described in Sect. 3.2:

VirusTotal VirusTotal offers information about malicious hosts and files based on analysis of multiple anti-virus tools.

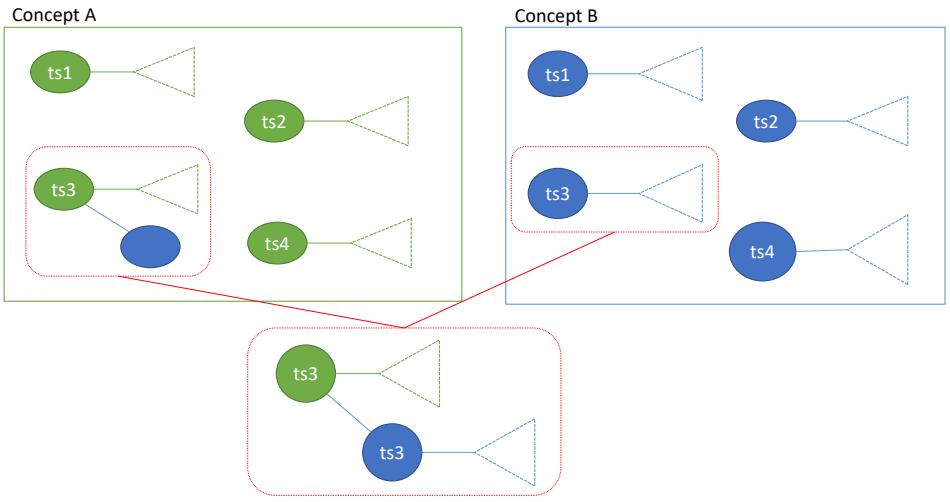

FIGURE 5.2: Substitution of nodes of an inter-relation with its respective concept based on the time-aggregated instance.

Threat Intelligence Threat intelligence sources provide information on indicators of compromise of IP addresses or domains. In addition, *Who-is* information for an IP address can be retrieved.

Urlscan.io Urlscan.io can be used to visit a site in a sandbox. Information that can be returned shows the routing and loaded external resources when visiting the site.

Knowledge Base The Knowledge Base may provide information about the infrastructure of the client's network, as well as what assets are present. By enriching the graph with this information, an analyst may have pointers about what assets are more valuable and might pose extra risks when attacked.

Another interesting enrichment source are the rules that triggered the alert, which have been described in Section 2.1. This information may aid the analyst in deciding which links in the graph are of most particular interest. Moreover, this information may be used to assist filtering the concepts before enrichment.

5.2.3 Filter relevant information

To ensure that the analyst is presented with the most relevant information, we can use the document ranking technique tf-idf that is used in most data mining systems [25]. We can model the alarm that the analyst obtains as a document itself. Subsequently, we can rank each of the concept instances as well as relations between them against the alarm using a tf-idf with cosine similarity score. We chose not to use the same similarity score as used between concepts, while attributes of relations

may contain raw string values which are not further processed. When scoring exact matches, these may not be selected and thus not shown initially.

The tf-idf score depends on the weight a term has within a data set. TF, or *Term Frequency*, describes the amount of occurrences of a single term within a document. IDF, or *Inverse Document Frequency*, provides a measure of the information content a single term has. For example, the word *the* does not have a high information content, because it is likely that this word occurs a lot within a document set.

Based on the model described in Chapter 4, and the constraint handling defined in the previous sections, we are presented by a set of instances along with the relations that exist between concepts. All of these are modelled as a set of key-value pairs. The analysis starts out with an alarm generated by the IDS. We can also model this alarm as a set of key-value pairs. To be able to examine the relevancy of different relations, we should look at the values. We can perform the TF/IDF similarity score with the document pool being all relations defined in the current environment, and have the IDS alarm as the query. By comparing these values, we are left with a ranking of relations that are most relevant. These should be shown first, as they might already contain enough information for the analyst to check out the alarm, being either a real alarm or a false positive.

5.3 Conclusion

This chapter introduced the overall processing steps that can be performed when applying Concept Mining on raw data, and showed the application thereof in the SOC domain. Note that the described process can be performed on any data and for any purpose, which might require tuning or restructuring the post-processing and enrichment steps. Also, the constraints within the SOC domain have been explained, as well as possible solutions for tackling these problems. In the next section, we take a look at a proof of concept that applies the outlined theory on a practical case.

Chapter 6

Glaukos - Proof of Concept

In the previous chapters, we have looked at the theory that describes the process of turning raw data from different sources into a model combining the information retrieved into a single object that can be used to analyse an alarm. The research question however not only deals with the theory of concept mining, but also how it can be applied to a situation where the analyst is aided by this model. In this chapter, we look at a code implementation named Glaukos, a proof of concept based on the theory described, and evaluate how the proof of concept can aid analyst given their current workflow.

6.1 Used Technology

In order to evaluate whether evidence generated the model is suitable for use by an analyst, we had to find a way that made sure the data represented by the model could be conveyed to the user. As denoted in Chapter 4, we model the universe as a directed graph, with nodes being concept instances, and edges being relations between concept instances. This can be directly transferred to the user, while graphs can depict almost all the information that is needed by the analyst.

Glaukos contains two components. One is the back-end module, written in Python, which is currently used to do the following:

1. Process the EDXML-formatted files into suitable objects in Python.
2. Generate a graph given a query from the user.
3. Obtain extra information (see phase 4, enrichment) for concept instances that are part of the graph.

On the other hand, we have a user interface in a browser based on JavaScript. For showing the graph in JavaScript, the D3-Library is used [8]. In the interface, the analyst can query the data set to generate and show a graph, either:

- Within a given time range, all data

- Within a given time range, all data with a defined root property and maximum depth of traversing the graph.
- For a given issue tracker ticket describing an alarm.

The defined root property in the second option is an attribute value that is part of a concept instance, which can be used to query and select the instances from which to start the depth-first search procedure from.

The last case is the most important. If all data is captured live, we can also query tickets that appear in an issue tracker. In this ticket, important information about affected hosts, the cause of the alarm, and maybe some alarm identifiers may be included. The concept instances that are directly related to the ticket are likely to be of most importance.

For all nodes and edges, the relevant information is shown after clicking or hovering over them. This allows the analyst to easily retrieve the wanted information within a short time.

While this research was focused on generating a model to collect and combine information from different sources in a single source, the process of creating the interface falls outside the scope of the research. Therefore, the current depiction may not be the final one used, nor may it be the best for the given model. The design is mostly based upon first-hand experience with alarm analysis and the ideas of what information should be available within the blink of an eye. Some figures show the representation of the data contained in the model as a graph. Figure 6.1 shows the graph on its own, showing the concepts and the relations between them. Figure 6.2 shows the interface as presented currently, with the option to search for a concept within a given time range, and a specified depth optionally for cutting of the Breadth First search process. Figure 6.3 shows the information that is shown when an edge is selected. One relation contained in this edge can be further analysed, as shown by Figure 6.4. This UI was used in the evaluation process, which is described in the next section.

6.2 Evaluation process

There are two questions that can be looked at to evaluate the model that has been proposed for the problem dealt with:

1. Is the information contained in the model similar to the information an analyst retrieves when manually checking? (Model precision)
2. Can an analyst efficiently use the model in the way it is presented, and does it have additional value over the current process? (Efficiency)

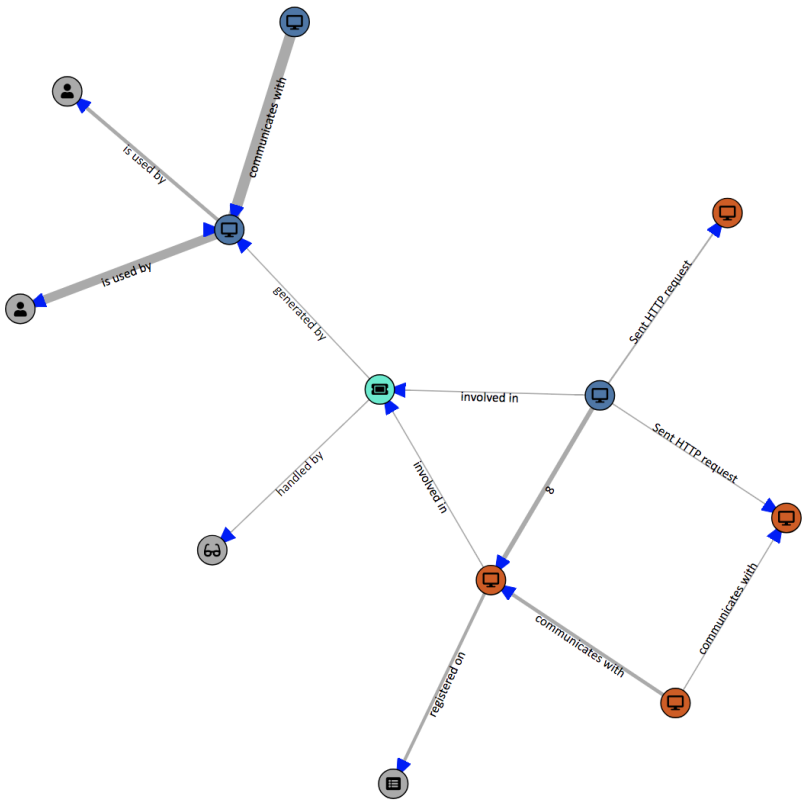

FIGURE 6.1: Instance of a graph as shown in the user interface.

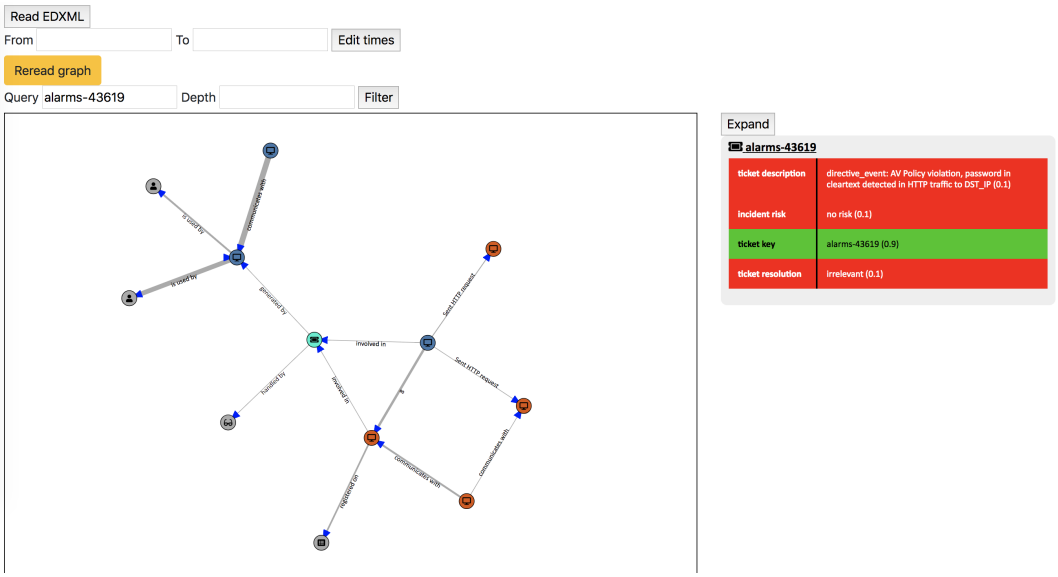

FIGURE 6.2: The full basic user interface, with the possibility to filter items.

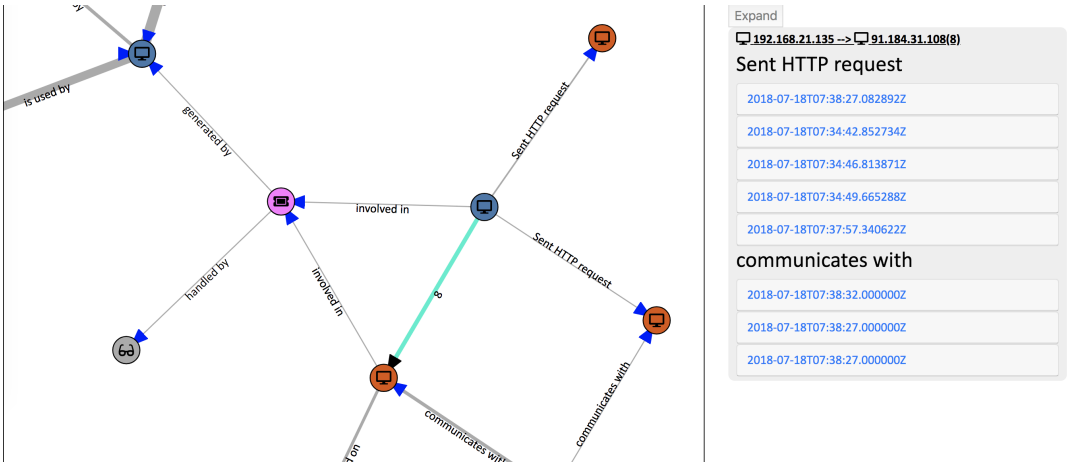

FIGURE 6.3: Information for a selected relation between two instances.

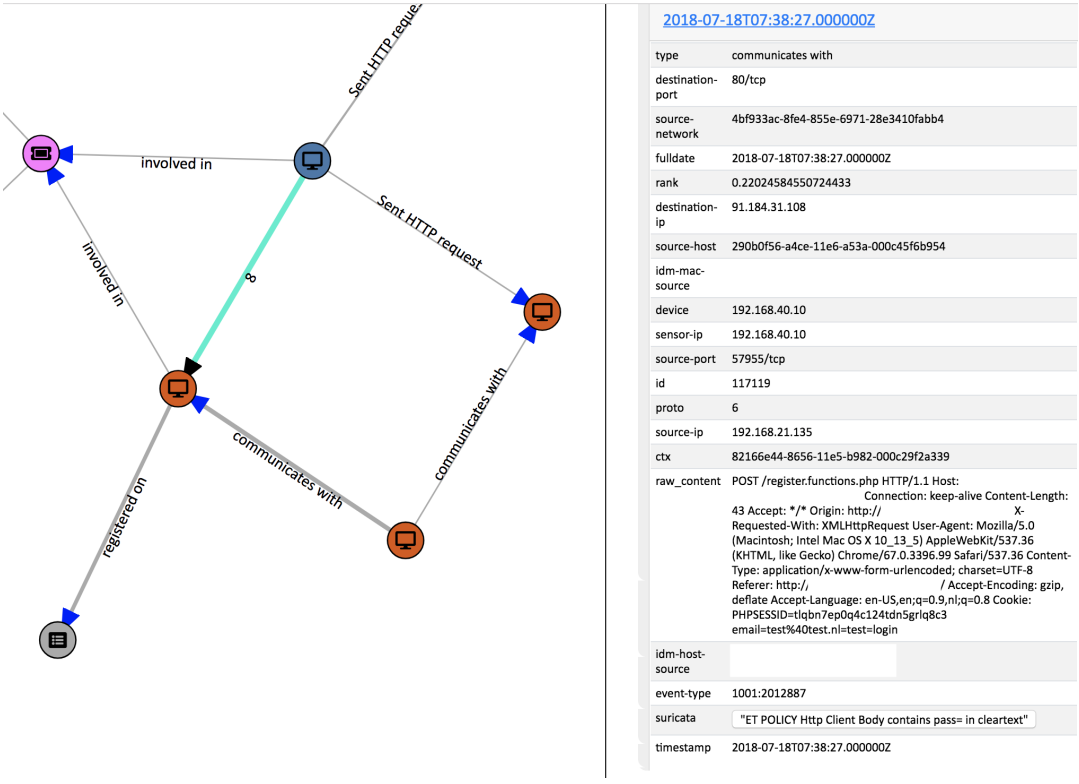

FIGURE 6.4: Expanded information for a single event contained in the relation.

To calculate model precision, one would need a lot of data that can be processed on the go. However, due to the nature of the data (security sensitive) and rules regarding ownership of data, it was very difficult to generate the data necessary to perform this analysis. There is therefore no possibility yet to measure the precision. However, in the future, this may be a topic of interest in order to test the model described

Therefore, the decision was made to perform an interview with the analysts to check whether their work could be enhanced when the data obtained using the model was presented to them. As explained before, the presentation that is contained within the proof of concept is based on personal experience, this may not be tailored to suit the needs of everyone.

6.3 Methodology

The main question to be answered was: *Can a depiction of the domain with the relevant data provide enough insight into the context of an alarm that an analyst can effectively solve the case?* This question can be answered using the following sub-questions:

- Can you correctly analyse the alarm given the presented evidence?
- What information would you consider vital but is missing from the presented evidence?

First of all, the SOC domain was defined in the domain specification by a domain expert, in order to perform the preprocessing step. An alarm that was generated for the phishing use case was captured, as well as the data that was available in the different sources used by a SOC analyst. This data was preprocessed to generate the concept instances (step 2 of the process).

Then, a number of analysts was asked to analyse the alarm using Glaukos. During their analysis, they could ask questions about the interface, but not about the alarm itself. The evaluation was conducted using an adaptation of the think-aloud protocol, described by Lewis and Rieman [15]. Afterwards, they were asked the questions described above.

6.3.1 Results

During the evaluation, a total of 6 analysts were asked to perform the analysis. I will outline the most important findings as well as some pointers to the improvements named by the analyst.

The general impression from most of the analysts was that the combination of raw information as well as a contextual presentation using the graph could provide more

information than just plain data. Furthermore, a frequently mentioned benefit is that no context switching is necessary any more. In the current situation, analysts have to switch to different systems to gather the information necessary. Using Glaukos, they were able to find all the data within only one source, so they can be more focused on the task.

Furthermore, the conclusion of most analysts is that the proposed model is of high additional value for performing an analysis, and could speed up analysis of most alarms. There surely are cases in which the model will fail to generate the most relevant piece of data, but in general, the data presented should be enough to solve the alarm efficiently.

Therefore, the questions can generally be answered as follows:

Can you correctly analyse the alarm given the presented evidence?
All analysts were able to correctly solve the alarm using the provided context and the data contained in the presented evidence. This required performing an extra search, which all of the interviewed analysts were able to perform. The right data came out of the model.

What information would you consider vital but is missing from the presented context?
One of the analysts proposed to include an extra external source with which an image of a web page could be shown, as is currently used by some analysts to check the legitimacy of a website. Although this could provide extra insight, he did not consider this to be vital. Otherwise, no data was missed by any analyst.

6.3.2 Possible improvements

The current system only provides one way of showing the context. That is, the graph represents the connections that are present between different concepts. However, some analysts proposed to include a timeline of events. The graph does not show the order in which events come in, although this can be deduced from the data if necessary. If the analyst could see a graph with the relations as well as the sequence of events, this would mean a complete picture of the event can be generated instantly in the analyst's mind.

Given the assumption that the UI would not be part of the analysis whether the model succeeds in conveying the series of events that lead up to an alarm, we can conclude that the model contains all necessary information to correctly analyse this type of alarm in the given test case. While the phishing use case shown is on the more complex side, involving a lot of context, the model may work efficiently on alarms which are more frequent but pose an equal amount of risk.

Besides, analysts also mentioned a danger of this tool. The representation in a graph may force the analyst into only looking at the data contained there, although other relevant data is not shown. This may induce tunnel vision, and should at all costs be prevented. A solution could be to include the graph as an option, and find another way of displaying the data.

6.4 Conclusion

In this chapter, we looked at a proof of concept to perform an analysis of the application of the model. A use case regarding phishing was presented to the analyst using Glaukos as a Proof of Concept to evaluate the effectiveness of the data contained in the presented evidence generated by the model in solving an alarm. We saw the opinions of the analysts which have used the tool as well as some improvements proposed by them.

Chapter 7

Conclusion

During this research, we studied how to capture security data in a semantic model, in order to aid SOC analysts in their task of triaging alarms.

First of all, there are differences in how alarms are generated by an IDS. In some cases, a hard rule defining exceptional content triggers an alarm. This rule details the exact circumstances around why an alarm was generated, and these alarms are thus easy to examine. In other cases, however, alarms are generated based on anomalous behaviour with respect to a baseline of known traffic. Alarms generated by such systems are in general more difficult to examine, as they rely on statistics. No hard indicators can be found, and the analyst has to dive deeper into the data.

For an analyst in a SOC, there are multiple data sources which should be looked at to analyse an alarm. This usually starts off with the alarm itself, after which the analyst has to check network logs, application logs or outputs of a vulnerability scan. In addition, open source data streams may provide extra context to domains and IP addresses found in network traffic. All this data originates from a variety of data sources. This poses a workflow that is not optimised, because an analyst has to switch context very often.

We tried to solve this by proposing a model to grasp the semantics of the security domain within the data, so a computer can aid the analyst in providing a full context around an alert. By using Model Based Concept Mining, multiple data streams can be combined into a single structured data stream, which can be handled by a computer. The model is based on concepts and relations which are currently defined by an expert. The process can be split out into four phases, which can be extended to the needs of the domain.

For the domain regarding security information in the SOC, I have found that some adaptations have to be made towards accommodating for the time constraints that have to be dealt with within the SOC when analysing alarms.

A Proof of Concept to evaluate the possibilities of the model has been built. Although the evaluation was dependent on the user interface the analyst was confronted with, the overall feeling was that the model itself, and the context it provides

for an alarm, is useful. The major benefit is that all data is now contained within one system, and no context switching is necessary to analyse an alarm. Furthermore, a specific use case regarding a phishing alarm could be correctly analysed using the model by all of the analysts that were asked to use the system. Because the phishing alarm is one that required a lot of context, the model should be suitable for other alarms requiring less context as well.

All in all, the model proposed thus seems to be a good fit for the analysts in the SOC, and should be of additional value to analyses performed, although improvements may be necessary to fully suit the needs of an analyst.

7.1 Discussion

Based on the results achieved using the evaluation, we may be able to conclude that the model proposed is of additional value to the analyst. However, there are some considerations regarding the outcome presented. First of all, the model is not fully tested for precision. That is, the model was only tested in combination with the user interface. This may generate noise as to whether the model is suitable for aggregating data correctly. Secondly, only one use case has been tested. This, however, should not be a problem while a use case which needs a lot of context is chosen. On the other hand, it would have been better if multiple types of alarms could be analysed in order to check whether the model is a good fit for the general purpose of analysing alarms.

A shortcoming of the model is that the domain specification has to be done by hand. This makes the model and the data represented by the model dependent on the hands of the expert specifying the domain. Furthermore, some relations might be incorrect, and others might be missing. This could lead to problems when analysing alarms because incorrect information, or no information at all, is present.

7.1.1 Further Research

The current domain specification is generated by hand. Because this can be erroneous, there may be a way to perform a feedback operation to improve the domain specification, or generate the domain specification using further data analysis tools.

As confidence values are defined within the domain specification, there may be the need for a feedback loop regarding changes to the confidence value, so incorrect combinations based on assumptions on the confidence value can be corrected.

As explained before, the model could not be tested on a dataset that consists of alarms and the data from the different data sources related to them. To really check

whether the model has the right precision, research could be done into the precision of the model on the SOC domain.

Furthermore, the proposed model is defined in four general steps, and can be extended to other applications. This raises the idea to perform an analysis whether other domains can benefit from this technique as well, with alterations specific for the domain.

Besides, the interface generated to evaluate the model has some shortcomings, and may be adapted to suit the workflow even better. This creates another topic of research which is more tuned to user experience.

The similarity score proposed in Section 4.5.3 can also be extended to suit other types of variables. For example, string matching may be performed using a distance score, which may map nearly equal strings to the same concept attribute. This may extend the model to perform more fuzzy matching. However, a good look should be taken here that properties of the model are kept in place, like symmetry in merging instances.

Furthermore, the model can be extended in the way merges are performed. In the proposed model, we combine sets with single values. The model can be altered to provide a way to define combinations of sets being merged.

7.1.2 Reflection

During the research, some problems were experienced. First of all, the data to perform tests with was not available at the start. This in the end still was not available, which hampered the research regarding the precision of the model. In an ideal environment, the test data is available before the research is conducted to eliminate any dependencies.

Secondly, this research taught me not to fall deeply into the literature, because studying irrelevant literature may take up a lot of time which could be spent in a better way. On the other hand, this helped scoping the research correctly, and may in the end even be positive with regards to the result.

Appendix A

EDXML

A.1 EDXML Event Type

Below is the event type for an event coming from the dns.log file of Bro. The type is defined by properties that should be present in the log. Furthermore, properties *host-ip* and *responder-ip* contain the attributes *concept* and *concept-confidence*, indicating that these properties define a certain concept (*nad.computer*) with a certain confidence (*9, or 0.9 in terms of probability*)

```
<eventtype classlist="" description="Bro DNS Log Event"
    display-name="Bro DNS Log Event/Bro DNS Log Events"
    name="com.bro.log.dns" story="no description available"
    summary="Bro log: [[origin]]">
<properties>
<property description="Origin of the event-data" merge="drop"
    multivalued="true" name="origin" object-type="com.bro.log.name"
    optional="true" similar="" unique="false"/>
<property cnp="128" concept="nad.computer" concept-confidence="9"
    description="IP of the Host machine" merge="drop"
    multivalued="true" name="host-ip"
    object-type="computing.networking.host.ipv4" optional="true"
    similar="" unique="false"/>
<property cnp="128" concept="nad.computer" concept-confidence="9"
    description="IP of the responding machine" merge="drop"
    multivalued="true" name="responder-ip"
    object-type="computing.networking.host.ipv4" optional="true"
    similar="" unique="false"/>
<property description="Protocol name" merge="drop" multivalued="true"
    name="proto" object-type="computing.networking.protocol.name"
    optional="true" similar="" unique="false"/>
<property description="Port of the responding machine" merge="drop"
    multivalued="true" name="responder-port" object-type="count.small"
    optional="true" similar="" unique="false"/>
<property description="Linux Epoch time" merge="match"
    multivalued="false" name="timestamp" object-type="datetime"
    optional="false" similar="" unique="true"/>
```

```
<property description="Specifies whether the query was rejected by the
    server" merge="drop" multivalued="true" name="rejected"
    object-type="boolean" optional="true" similar="" unique="false"/>
<property description="Round trip time of the query and response.
    Indicates delay" merge="drop" multivalued="true" name="rtt"
    object-type="datetime.duration.seconds" optional="true" similar=""
    unique="false"/>
<property description="Port of the Host machine" merge="drop"
    multivalued="true" name="host-port" object-type="count.small"
    optional="true" similar="" unique="false"/>
<property description="Unique ID linked to a connection" merge="drop"
    multivalued="true" name="unique-interface-id"
    object-type="com.bro.log.connectionuid" optional="true" similar=""
    unique="false"/>
<property description="DNS Query subject" merge="drop"
    multivalued="true" name="query"
    object-type="computing.networking.host.dns.name" optional="true"
    similar="" unique="false"/>
<property description="Response code of the DNS query" merge="drop"
    multivalued="true" name="resp-name"
    object-type="computing.networking.protocol.name" optional="true"
    similar="" unique="false"/>
<property description="Response code of the DNS query" merge="drop"
    multivalued="true" name="rcode" object-type="count.small"
    optional="true" similar="" unique="false"/>
<property description="Random number chosen by the client for
    transaction" merge="drop" multivalued="true" name="trans-id"
    object-type="count.large" optional="true" similar=""
    unique="false"/>
</properties>
<relations>
<relation confidence="10" description="Bro produced a log based on
    communication from [[host-ip]] to [[responder-ip]]"
    directed="true" property1="host-ip" property2="responder-ip"
    type="inter:communicates via DNS with">
<descriptors>
<descriptor description="The connection is identified with id:
    [[unique-interface-id]]" property="unique-interface-id"/>
<descriptor description="The connection used protocol [[proto]]"
    property="proto"/>
<descriptor description="The host requested [[query]]"
    property="query"/>
<descriptor description="The query was responded to with status code
    [[rcode]]" property="rcode"/>
<descriptor description="The query had a RTT of [[rtt]]"
    property="rtt"/>
<descriptor description="The query was rejected: [[rejected]]"
    property="rejected"/>
<descriptor description="The query was performed at [[timestamp]]"
    property="timestamp"/>
</descriptors>
</relation>
```

```
</relations>
</eventtype>
```

A.2 EDXML Event

An event that has been parsed from a Bro dns.log file is shown below. *Note that the information has been slightly adapted to prevent showing real-world data.*

```
<event>
    <properties>
        <origin>dns</origin>
        <host-ip>11.1.1.11</host-ip>
        <responder-ip>11.1.2.12</responder-ip>
        <proto>udp</proto>
        <responder-port>53</responder-port>
        <timestamp>2018-04-25T22:00:22.222222Z</timestamp>
        <rejected>false</rejected>
        <rtt>4.041000E-003</rtt>
        <host-port>37239</host-port>
        <unique-interface-id>abcdefghijklmnopqr</unique-interface-id>
        <query>github.com</query>
        <resp-name>NOERROR</resp-name>
        <rcode>0</rcode>
        <trans-id>12345</trans-id>
    </properties>
</event>
```

Bibliography

[1] *ArcSight*. URL: https://software.microfocus.com/en-us/products/siem-security-information-event-management/overview (visited on 06/12/2018).

[2] Franz Baader. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

[3] Tim Berners-Lee and Robert Cailliau. *World-Wide Web*. 1992.

[4] Alex Borgida and Ronald J. Brachman. "Conceptual Modeling with Description Logics". In: *The description logic handbook: Theory, implementation and applications*. Vol. 45. 2. 2003. Chap. 10, pp. 359–381. ISBN: 9780521781763. DOI: 10.2277/0521781760.

[5] Ronald J. Brachman and James G. Schmolze. "An overview of the KL-ONE Knowledge Representation System". In: *Cognitive Science* 9.2 (1985), pp. 171–216. ISSN: 03640213. DOI: 10.1016/S0364-0213(85)80014-8.

[6] B Claise. *Cisco Systems NetFlow Services Export Version 9*. RFC 3954. 2004, pp. 1–33. DOI: 10.17487/rfc3954. URL: https://tools.ietf.org/html/rfc3954.

[7] B Claise, B Trammell, and P Aitken. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. Tech. rep. 2013, pp. 1–76. DOI: 10.17487/rfc7011. arXiv: arXiv:1011.1669v3. URL: https://tools.ietf.org/pdf/rfc7011.pdf.

[8] *D3 - Data Driven Documents*. URL: https://d3js.org/ (visited on 10/05/2018).

[9] Hervé Debar, Marc Dacier, and Andreas Wespi. "Towards a taxonomy of intrusion-detection systems". In: *Computer Networks* 31.8 (1999), pp. 805–822.

[10] Dorothy E. Denning. "An intrusion-detection model". In: *IEEE Transactions on software engineering* 2 (1987), pp. 222–232.

[11] Hai Dong, Farookh Khadeer Hussain, and Elizabeth Chang. "A context-aware semantic similarity model for ontology environments". In: *Concurrency Computation Practice and Experience* 23.5 (2011), pp. 505–524. ISSN: 15320626. DOI: 10.1002/cpe.1652.

[12] Paul Ferguson and Daniel Senie. "Network Ingress Filtering". In: *Network Working Group Request For Comments,(May 2000)* (1998), pp. 1–10. URL: https://tools.ietf.org/html/rfc2827.

[13] Rick Hofstede et al. "Flow monitoring explained: From packet capture to data analysis with netflow and ipfix". In: *IEEE Communications Surveys & Tutorials* 16.4 (2014), pp. 2037–2064.

[14] International Organization for Standardization. *ISO/IEC 7498-1:1994 Information Technology - Open Systems Interconnection - Basic Reference Model: The Basic Model*. 1996. URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/s025022_ISO_IEC_7498-3_1997(E).zip.

[15] Clayton Lewis and John Rieman. *Task-Centered User Interface Design*. Tech. rep. 1993.

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

[17] Carlos Morales. *NETSCOUT Arbor Confirms 1.7 Tbps DDoS Attack; The Terabit Attack Era is Upon Us*. 2018. URL: https://www.arbornetworks.com/blog/asert/netscout-arbor-confirms-1-7-tbps-ddos-attack-terabit-attack-era-upon-us/.

[18] *OpenVAS*. URL: http://www.openvas.org.

[19] *OSSIM*. URL: https://www.alienvault.com/products/ossim.

[20] Vern Paxson. "An analysis of using reflectors for distributed denial-of-service attacks". In: *ACM SIGCOMM Computer Communication Review* 31.3 (2001), pp. 38–47.

[21] Vern Paxson. "Bro: a System for Detecting Network Intruders in Real-Time". In: *Computer Networks* 31.23-24 (1999), pp. 2435–2463. URL: http://www.icir.org/vern/papers/bro-CN99.pdf.

[22] Ronald Prins et al. "Black Tulip: Report of the investigation into the DigiNotar Certificate Authority breach". In: December (2012), pp. 1–101. DOI: 10.13140/2.1.2456.7364.

[23] Philip Resnik. "Using Information Content to Evaluate Semantic Similarity in a Taxonomy". In: 1 (1995). ISSN: 1045-0823. DOI: 10.1.1.55.5277. arXiv: 9511007 [cmp-lg]. URL: http://arxiv.org/abs/cmp-lg/9511007.

[24] M Roesch. "Snort: Lightweight Intrusion Detection for Networks." In: *LISA '99: 13th Systems Administration Conference* (1999), pp. 229–238. ISSN: 15437221. DOI: http://portal.acm.org/citation.cfm?id=1039834.1039864. arXiv: arXiv:1011.1669v3. URL: http://static.usenix.org/publications/library/proceedings/lisa99/full_papers/roesch/roesch.pdf.

[25] Gerard Salton and Christopher Buckley. "Term-weighting approaches in automatic text retrieval". In: 24.5 (1988), pp. 513–523.

[26] *Splunk*. URL: https://www.splunk.com/ (visited on 06/12/2018).

[27] Dik Takken. "EDXML v3.0". 2018. URL: www.edxml.org.

[28] *URLScan.io*. URL: https://urlscan.io/ (visited on 04/03/2018).

[29] Alfonso Valdes and Keith Skinner. "Probabilistic alert correlation". In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2001, pp. 54–68.

[30] Verizon. "2017 Data Breach Investigations Report". In: *Verizon Business Journal* 1 (2017), pp. 1–48. ISSN: 1098-6596. DOI: 10.1017/CBO9781107415324.004.

arXiv: arXiv : 1011 . 1669v3. URL: http : / / www . verizonenterprise . com / verizon-insights-lab/data-breach-digest/2017/.

[31] *VirusTotal*. URL: https://www.virustotal.com/ (visited on 04/03/2018).

[32] Wei Wang and Thomas E. Daniels. "A graph based approach toward network forensics analysis". In: *ACM Transactions on Information and System Security (TISSEC)* 12.1 (2008), p. 4.