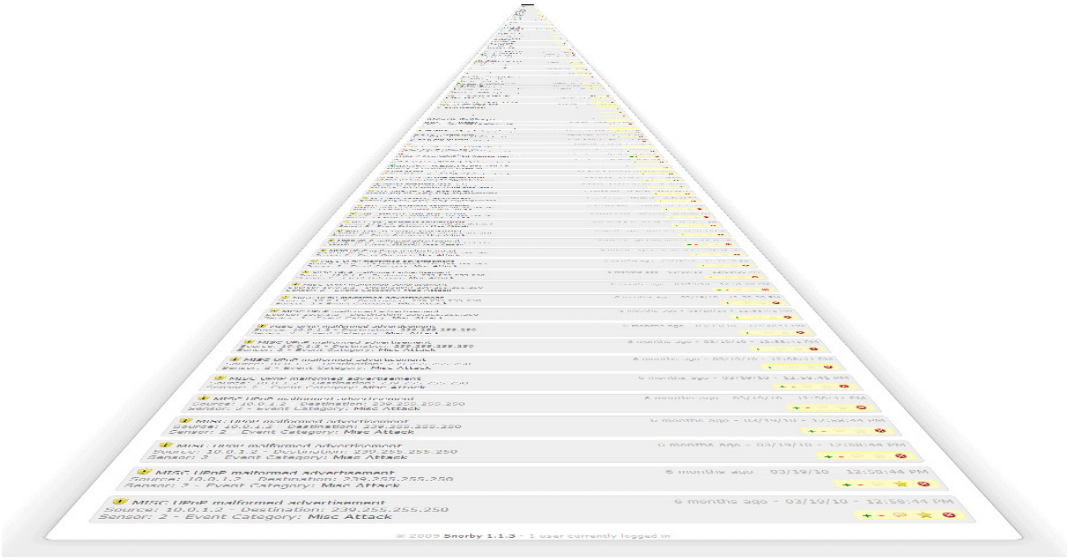
UNIVERSITY OF GOTHENBURG



Alarm management for intrusion detection systems

Prioritizing and presenting alarms from intrusion detection systems.

Master of Science Thesis, Computer Science Programme

SEBASTIAN KLÜFT
EVA LINA STAAF

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, December 2010

Alarm management for intrusion detection systems
Prioritizing and presenting alarms from intrusion detection systems.

Sebastian Klüft
Eva Lina Staaf

Examiner: Erland Jonsson
Supervisor: Magnus Almgren

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden December 2010

This page has intentionally been left blank.

Abstract

Intrusion detection systems (IDSs) are important tools helping the network and system administrators to detect intrusions, but have the drawback of many false positives. Due to increasing bandwidth, an IDS must process a vast amount of data, which results in an ever increasing amount of alarms. For a system administrator to be able to handle the alarms they must be aggregated, correlated and ordered into a manageable form and presented in a way which is easy to overview.

In this thesis we study aggregation, correlation, filtering and ranking as methods for managing alarms from IDSs. We have implemented a ranking functionality in the graphical user interface *Snorby*, a front end to the open source IDS *Snort*. Each alarm starts with a basic rank of 0 and the user is able to prioritize or down prioritize the alarm by pressing either a '+' button or a '-' button, thus influencing its current rank. The rank is calculated from several features, i.e. source IP, destination IP, destination port and alarm signature.

Based on our studies we suggest that ranking systems supported by user votes have several advantages. First, they allow the user to dynamically change the way the IDS lists the alarms through a very simple means. Second, it shortens the time required to locate the more important ones, thus reducing the likelihood that a serious attack will be missed.

Keywords: intrusion detection, IDS, correlation, fusion, aggregation, filtering, ranking, alarm management

Sammanfattning

Intrångsdetekteringssystem (IDS) är ett viktigt verktyg som hjälper nätverks- och systemadministratörer att upptäcka och förebygga intrång, men har en nackdel i många falsklarm. På grund av den växande bandbredden måste ett IDS bearbeta en stor mängd data, vilket resulterar i ett ständigt ökande antal larm. För att en nätverks- eller systemadministratör ska ha en möjlighet att bearbeta detta måste larmen aggregeras, korreleras och ordnas till en hanterbar form och presenteras på ett sätt som är lätt att ta till sig.

I den här rapporten studerar vi agregering, korrelering, filtrering och rankning som metoder för att hantera alarm från ett IDS. Vi har implementerat ett rankningssystem i det grafiska gränssnittet till *Snorby*, som är ett web-gränssnitt till *Snort*. Varje alarm börjar på rank 0 som användaren av systemet sedan kan höja eller sänka med hjälp av två knappar: '+' och '-'. Varje alarms prioritet baseras på fyra egenskaper: käll-IP, destinations-IP, destinations-port och alarmsignatur.

Baserat på vad vi har kommit fram till i vårt arbete kan vi dra slutsatsen att ett system som rankar alarm baserat på hur användarna röstar har flera fördelar. För det första låter det användaren dynamiskt påverka hur IDS:et listar alarmen, genom väldigt enkla medel. För det andra minskas tiden det tar att hitta de viktiga alarmen, och minskar således risken för att en allvarlig attack ska förbises.

Preface

First of all, this thesis would not have been possible without our mentor and guiding star **Magnus Almgren**. We are deeply grateful and awed by the effort you have put into this. The Energizer bunny doesn't hold a candle to you.

We would also like to thank the following people:

Håkan Nohre, security expert at Cisco, who although being very busy still took the time to answer our questions and giving us an interesting tour through the different intrusion detection systems from Cisco.

Pierre Kleberger for giving us figures of IDS systems and helping us when we had trouble installing BASE.

Emilie Lundin Barse and **Ola Söderström** for giving us some insights into the world of being an IDS administrator and answering our questions in person. Also an extra thanks to Ola for providing us with some statistics we could use to run our simulations.

Ulf Larson for answering our questions in an entertaining way and giving us a lot of ideas to think about.

Erland Jonsson and **Wolfgang John** for attending our presentations, giving critique, and asking relevant questions that helped us find a better focus in our work.

Eva Lina wishes to thank her own little cookie monster, spiritual alka-seltzer and gifted toe stepper, **Tomas Olovsson**, for not asking too often how this project was coming along and for knowing who Carl Barks is. Also, thanks for enlightening and entertaining conversations and being an all around nice guy.

Contents

List of Figures

List of Tables

List of Definitions

Definition 1. AGGREGATION *is the process of grouping alarms together based on certain user-defined criteria found in alarms.* (p. 27)

Definition 2. CORRELATION *is the process of building data models from alarms using external knowledge not found within the information carried by the alarms themselves.* (p. 28)

Definition 3. *An* IMPORTANT ALARM*, also referred to as a* CLICKED ALARM*, is an alarm with attributes of interest to the user. When the user spots this alarm, she is surmised to click on it, showing her interest.* (p. 58)

Definition 4. *An* UNIMPORTANT ALARM*, sometimes referred to as an* UNCLICKED ALARM*, is an alarm lacking attributes of interest to the user. When the user spots this alarm, she is surmised to leave it unclicked, showing her disinterest.* (p. 58)

Definition 5. *The* DENSITY $D(i)$*, of ranked alarm position i, is the ratio of important alarms found at positions 1 up to i compared to the total number of alarms found at positions 1 up to i, where $i \leq$ position of last listed important alarm.* (p. 58)

$$D(i) = \frac{\#\ important\ alarms\ at\ positions\ 1\ to\ i}{\#\ alarms\ at\ positions\ 1\ to\ i}$$

Definition 6. COLLATERAL RANKING *is the act of awarding unimportant attributes with higher rank due to their association with important attributes. That is, unimportant attributes get points by simply being present in the same alarms as important attributes.* (p. 67)

Chapter 1

Introduction

1.1 Background

Since 1966, the syrupy sweet song "It's a small world after all" have been sung over and over again in the Disney theme parks by the little dolls in the attraction bearing the same name as the song [33]. This sentiment is truer today than ever. The birth of Internet has let us bridge physical distance with the click of a button. Whether you live in Australia, South Africa or the UK makes little difference, as with the click of a button millions of computers can be attacked and infected by malware. By the same notion that the world is getting smaller it is also growing; unfathomable proportions of information is being pumped into networks connected to the Internet. Individuals, companies, organizations and governments store their lives, financial records, data and secret documents on social communities and databases.

Much has happened in the world of malware since the Morris worm in 1988 [17]. Where Morris operated out of curiosity there are now multi-million dollar operations organized by criminal networks all over the world. Tools for attacking and penetrating networks are easily bought over the Internet. The "small world" we now live in and the illegal industry dealing in malware and hacking tools all make it dangerous to simply connect a network cable to your computer.

As a publicly available company or a public authority, a connection to the Internet is mandatory and crucial. If you cannot be found on the Internet you might as well not exist and without an Internet connection, how are your employees supposed to do their jobs? But a way out is also a way in. Traditional antivirus programs and firewalls are only able to detect and remove files containing code associated with malware and manage incoming and outgoing network communications respectively. There is no way for either of them to discover e.g. password guessing since the activity in itself is neither concerned with malicious code nor network communications.

What is needed is something auditing the system with a holistic perspective. That something is an intrusion detection system (IDS).

1.2 Problem description

For an intrusion detection system (IDS) to properly perform its task, it has to process a huge amount of information from firewalls, logs and audit records. The detection rules of the IDS will only single out a small part of the daily events as suspicious and raise an alarm, but a small part of a huge amount may still be a lot. As attacks in general are meant to go undetected, attackers try their best to masquerade their appearance. To accommodate this, the detection rules are defined in a slightly vague fashion. Unfortunately this vagueness increases the chance of legitimate actions being labeled as suspicious, a prevailing problem known as **false positives**.

In many current IDSs, it is the operator that must manually distinguish between true and false positives.[1] The fact that false positives easily outnumber the true alarms does not, in general, make the operator any happier. Finding the real threats can be the equivalence of finding a needle in a haystack.

1.3 Goal

To investigate the problem described in 1.2 and to see what can be done about it we intend to:

1. study current methods for reducing the number of alarms.
2. improve and implement one of these methods in an available IDS.

1.4 Limitations

This thesis is not an evaluation of IDSs and their effectiveness, nor is it about how to detect suspicious activities in a host or network. The method survey is not a complete or exhaustive survey and the chosen reports should be viewed as representatives of their respective areas.

1.5 Document Organization

Chapter 2 presents a general overview of the workings of an IDS, as well as a presentation of the IDS Snort and two of its most common interfaces, BASE and Snorby.

Chapter 3 presents a limited survey of methods used for prioritizing and reducing alarms.

Chapter 4 presents the methodology behind our work.

Chapter 5 presents our findings and conclusions.

[1] In this report we will use the words operator, user and administrator interchangeably.

Chapter 2

Intrusion Detection Systems

This chapter describes Intrusion Detection Systems (IDSs), how they work in general and a brief overview of systems available today.

2.1 General Overview

An IDS is similar to an antivirus program scanning files for malicious code. But where the traditional antivirus program only scans the content of a file the IDS parses and interprets network traffic and/or host activities. An IDS can use data collected from network packet analysis, logs from routers, firewalls, servers, local systems and system calls to mention a few [1]. Just like the antivirus program an IDS has a set of rules or a database containing known attack signatures. The IDS compares the signatures against patterns of activity, traffic or behaviour deduced from the data and issues an alarm when the pattern or behaviour is a close match. Besides from just issuing an alarm an IDS may in some cases also take active countermeasures such as shutting down Internet links, launch back-traces or make other attempts to identify the attacker and gather evidence. Specifically, an IDS detects unauthorized use of or attacks on a system or network. An IDS can, just like a firewall, either be software-based or hardware-based.

Figure 2.1 shows an overview of a generic IDS. *Target system* refers to where the sensor is situated, i.e. a router or an ordinary host. *Sensor* is a piece of software responsible for collecting data for the IDS. The data can originate from a multitude of sources each with its own format. The *pre-processor* is responsible for weeding out the data required for analysis and transforming it into a format that can be understood by the IDS. The *analysis engine* vets the data against the rules and policies to determine whether it is benign or malicious. *Post-processing* involves correlation of alarms, or construction of attack scenarios, or collection of alarms from other analysis engines. The *response unit* decides for each alarm if the reaction to it should be active or passive [25]. A passive

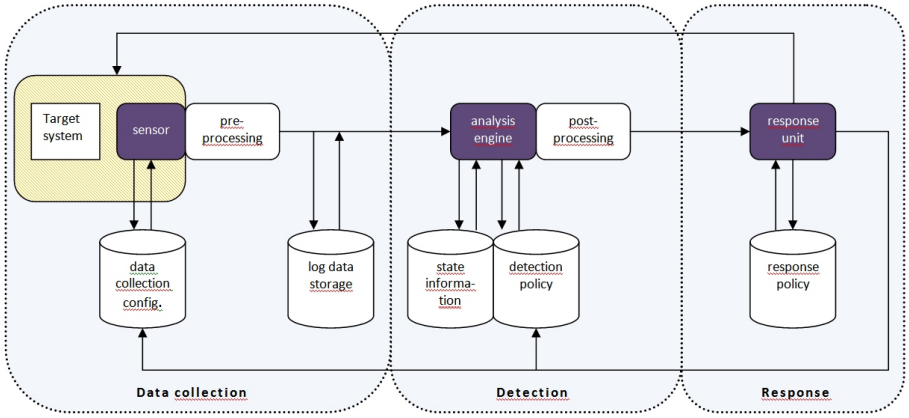


Figure 2.1: Architecture of a generic IDS [25].

response includes generation of alarms or log entries but other than that it does not interfere with the traffic in any way. An active response on the other hand, may reset the TCP connections in order to disrupt communications, simply drop traffic if the IDS is inline, add the malicious host to blocking lists, or in any other way disturb the attacker [1].

The typical IDS will not prevent an attack, its function is to detect and alert. However, some IDSs do have this attack-preventing feature and are called IPSs, intrusion prevention systems. But they are a minority, the typical IDS should be viewed as a complement to a firewall, antivirus software, and such and not as a replacement.

2.1.1 Placement

There are generally three different kinds of IDS,

- Network-based IDSs (NIDSs), monitors network links and backbones looking for attack signatures.
- Host-based IDSs (HIDSs), monitors the operating system and file system for signs of intrusion.
- Distributed IDSs (DIDSs), a number of IDSs used as remote sensors reporting to some kind of central management system.

There are also IDSs specifically dedicated to gateways monitoring the traffic passing in and out of the home network and IDSs dedicated to application-specific traffic understanding the flow of the application logic and the underlying protocols.

NIDS (Network-based IDS) A NIDS is an IDS that can monitor an entire segment of the network (see figure on page 17). It is placed on (an) appropriate

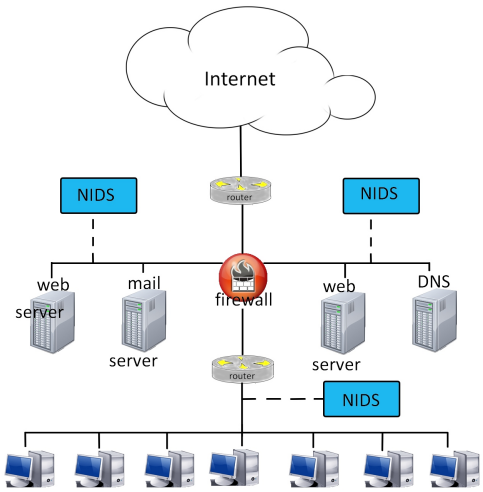


Figure 2.2: Architecture of a NIDS.

segment(s) of the network. All traffic seen by the network interface card (NIC) is passed on to the IDS. The advantages of a NIDS is the fact that it does not affect either the systems of the network or the network itself since it just passively records any traffic seen by the NIC. One disadvantage is that it only sees the traffic by its NIC. Packets taking a different route through the network will be missed. Therefore, a NIDS does not always see the whole picture.

HIDS (Host-based IDS) A HIDS protects only the host on which it is situated (see figure on page 18). Due to its location it has access to other types of data than just network information, including local information such as system calls, file system modifications, and system logs. A plus with the HIDS is the fact that it is possible to adapt the ruleset in great detail for each host. The ruleset only has to deal with the services specific for that host. A NIDS must have rulesets to deal with everything since it will see everything in traffic on the network. The fine tailored ruleset for HIDS reduces processor overhead and enhances performance for each host. The drawback of a HIDS is the fact that it steals computing power from its host, unlike a NIDS.

DIDS (Distributed IDS) The standard DIDS is part of a manager/probe architecture (see figure on page 18). NIDS and/or HIDS sensors placed throughout the network report to a centralized management station which compiles the reports. Attack logs can be uploaded and new attack signatures downloaded from the central management to the sensors as needed. The benefit of a DIDS is the holistic view it presents supplementing the NIDS, unaware of the state of hosts, and the HIDS, unaware of the state of the network. Since all systems have their drawbacks, so has also this one. The DIDS uses the internal network to communicate with the centralized management, and if the network is incapacitated, so is the holistic view.

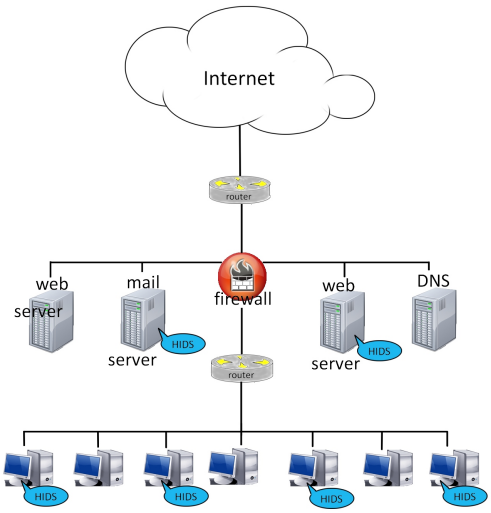


Figure 2.3: Architecture of a HIDS.

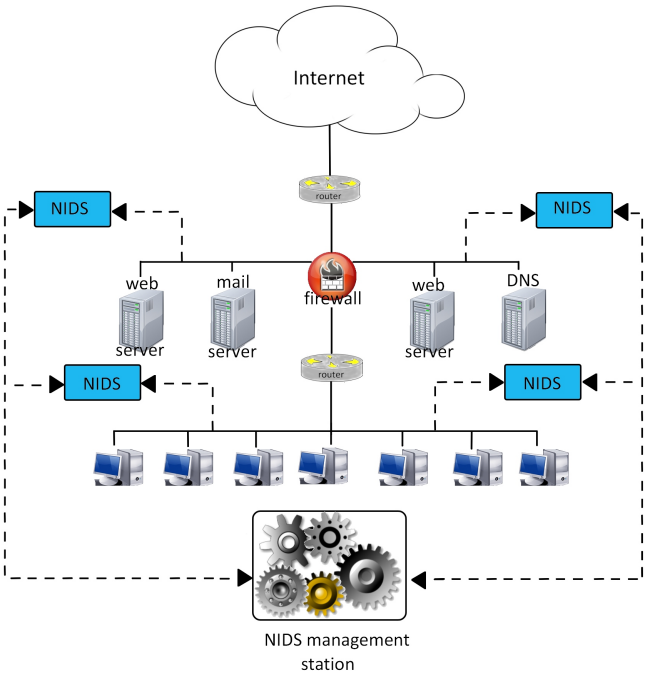


Figure 2.4: Architecture of a DIDS.

2.1.2 Detection Schemes

There are two different approaches an IDS can take when deciding if an event should be reported as an alarm or not, **anomaly detection** and **misuse detection**.

Anomaly detection, also known as statistical detection, strives to define normal, expected behaviour [35]. It is based on the idea that attacks and intruders displays different behaviour than the normal legitimate user. E.g. a trojan setting up a high number of connections where the normal user would yield only a handful of connections will result in an alarm [24]. To be able to decide what actions are anomalous a baseline of normality for user behaviour and system activity has to be established. The baseline is either predefined by the system developers or constructed using statistical sampling, specification-based approaches or neural networks [1]. Recording and deciding normal behaviour is a complicated process. It is hard to exhaustively define normal behaviour since it is dynamic and changes over time. Homogeneous systems with a moderate number of users where the same actions are performed repeatedly are much more readily described than a heterogeneous system with maybe thousands of users, say a university network. It is also hard deciding what is an anomaly and what is a minor deviation in the behaviour of the user. To partly compensate for this uncertainty, the decision function works with intervals and boundary values. The behaviour is allowed to slide a bit from the expected value without triggering alarms. When an alarm is triggered, the alarm is accompanied by a certainty value depending on how far off the behaviour is. This helps when quickly deciding if an event should be considered as to be a true or false positive, although caution should be exerted. Inappropriately set threshold values are a potential disaster. Set too high and attacks are be missed, set too low and the level of false positives skyrockets.

Policing all activities is a costly endeavour since the extensive logging needed by the system to perform has a significant impact on the performance of the host. This makes anomaly detection less suitable for real-time systems [1].

Anomaly detection has its benefits and drawbacks. Anomalous behaviour does not have to be an indication of an attack, the explanation for the altered behaviour could be as simple as new assignments for the user or a new service on the network. This results in costly **false positives** which have to be examined [24]. But this weakness is also the strength of anomaly detection. Although unaware of the ways of novel attacks an anomaly detecting IDS is still able to produce alarms for the novel attacks - as long as the attacks results in a system behaviour different from the normal. Unfortunately this is not the only obstacle to be conquered. When choosing the parameters to build the baseline, one has to make a correct selection of parameters and not choose too few or too many. Too few will result in an incomplete model missing attacks and too many will unnecessarily burden the host system.

Misuse detection, also known as signature-based detection or rule-based detection, strives to define improper behaviour [35]. It follows the same principles as an antivirus program. Patterns of misuse called attack signatures are stored in a database and the audited area, may it be log files or network traffic, is

vetted against these and any match will result in an alarm. A signature can be i.e. a string to search for in packets or a sequence of actions. Intrusion schemes are, just like malware, ever changing; the signatures must therefore be constantly updated as not to miss any attack. Most IDSs adhere to the misuse principle [24].

This method has both advantages and disadvantages if compared with anomaly detection. Where misuse detection cannot catch novel attacks, anomaly detection can. Where anomaly detection can never be quite certain of an intrusion, misuse detection can. Misuse detection suffers from similar definition problems as anomaly detection. The problem with attack signatures is how to define them and still be able to catch all misdeeds without causing false positives to trigger off on normal traffic.

To summarize; an IDS can adhere to a known-good (anomaly detection) or a known-bad (misuse detection) policy [1]. Known-good has configuration files with the acceptable behaviour explicitly stated. All else yields alarms. Known-bad is much simpler since it does not require a comprehensive model of allowed input. Known-bad trigger alarms only on traffic that is known to be bad. Known-bad and known-good both have drawbacks and benefits. Known-bad is easier on the processors, alerting only when something matches a rule. Unfortunately it can only alert on what is stated in the rules. Anything else sails through. Known-good has the benefit of being able to alert on novel attacks as long as they yield a behaviour deviating from the configured acceptable behaviour. The drawback of known-good is the impossible task of modelling all behaviour that is acceptable which results in many false positives. On top of that, it is completely normal for a user's behaviour to change over time. Also if the system is able to cope with a slowly changing behaviour, who is to say an intruder would not slowly accustom the system to a behaviour which would mask the planned intrusion? Most IDSs use a combination of the two.

2.1.3 Key Concepts

A **true positive** is an alarm correctly indicating an attack.

A **true negative** is an event correctly classified as benign.

A **false positive** is an alarm issued for a benign event mistaken for an attack. This is not uncommon in an anomaly detection system where all events not explicitly marked as approved issues an alarm [24]. Misuse detection is also plagued by erroneous alarms, the cause being badly written detection rules or benign traffic sporting a high similarity to malicious traffic [16]. Buggy software and badly configured hardware adds to the load for both systems, effectively drowning the true positives. This is a big problem today [3], and a lot of research and effort goes into finding a way to minimize the effects false positives have on an operator's work situation and the system dealing with alarms.
A high level of false positives is normal for "young" IDSs and drops off with the tweaking of the system over time [1].

A **false negative** is an attack the system fails to detect and is much more serious than a false positive [24]. Whereas false positives may obscure the view, the

true positives are still there, are still logged and retrievable from the database. An undetected attack is not logged anywhere and thus when and if the intrusion is detected there is no way of knowing for how long the system has been compromised, what data to trust, which backup is clean or how much havoc has been wreaked.

2.1.4 Motivation

So why bother having an IDS as it seems like a lot of headache? E.g. say a company one day discovers that someone has gained unauthorized access to their database. If they have no IDS, how can they know if they can trust their data? Find out when the breach happened? And if any of the backups can be trusted?

An IDS, hopefully, grants the ability to discover reconnaissance attempts, attempted system compromise or other malicious activity and this can mean the difference between being compromised or not. If no one is aware of the attacker trying to break into the system, how can she be stopped? With an IDS able to warn the administrator of a successful attack, the administrator can take measures to limit the damage or take the system offline to examine the extent of the breach.

Another reason to have an IDS is its data mining properties helping the administrator finding subtle trends in large amount of data, which would be impossible for a human to find unaided. Correlation between alarms and hosts helps the administrator to see patterns otherwise hidden.

No matter how big a help the IDS is, skilled analysts will always be needed to sift through the alarms the system produces. They have to be able to separate the real alarms from false positives. No matter how much the system learns, it will never be able to assess the level of a threat like a skilled and experienced analyst knowing her network.

2.2 Available systems

There are a great many IDS, IPS, and GUI available today from many different software companies. Many are commercial systems targeted at consumers and their home computers such as antivirus and firewall applications offered by Symantec, ESET and others. The systems that we are interested in in this thesis are however not targeted at home computers. They are designed to handle loads much greater than what a home computer ever will encounter, and require knowledge that most people do not have. During the course of our work we have studied a few of them, and most of them are open-source systems as we do not have the funds to buy any licenses.

2.2.1 Snort

Snort [34] is an open source network intrusion detection and prevention system now developed by Sourcefire and was originally released in 1998 [31]. It has three modes of operation. Packet sniffer mode reads packet headers and displays them on the console. Packet logger mode will log received packets to a file. Network intrusion detection mode will analyse incoming packets and detect intrusion attempts. Snort analyses network traffic with the help of predefined rules containing detailed descriptions of what kind of traffic should be considered harmful to the network. In other words it uses the misuse-based paradigm. These rules may also contain actions to be taken when a specific attack is detected. Appropriate actions can be to log the alarm to a database, send an email to an administrator or terminate the offending connection. With the help of these rules Snort looks at all parts of incoming network packets (header and payload) to determine whether each individual packet should be considered malicious or not. If a packet is considered malicious the appropriate action for that type of alarm will be taken.

The alarms that are captured by Snort are by no means guaranteed to be actual attacks or intrusion attempts made by an adversary. Snort is only designed to recognise possible attempted attacks on the network. This means it is essentially just a dumb sensor filtering out traffic matching a set of predefined rules. In turn this means that there is a relatively high chance that a majority of the alarms are going to be false positives (depending on how large a target for attacks the network is). The hard work then of analyzing the alarms and see the correlation between different alarms is left entirely to the operator to handle.

Snort sensors are most commonly placed in front of the firewall protecting the local network so they can capture the unfiltered traffic coming from the Internet. Figure 2.2 on page 17 is a good example of such a sensor placement. If the local network is very big and divided into subnets it might be a good idea to also place Snort sensors at the border of those subnets. Placing more than one Snort sensor in the network can help tracking the attack as it progresses in the network. An attack most commonly start with someone attempting to penetrate the outer defences (firewalls), if this succeeds they might attempt to infect a computer on the inside. If the computer is successfully infected it will become their bridge into the network. Therefore it would be good to have more sensors inside the network to monitor traffic between computers. The Snort sensor at the border to the internet will only be able to see incoming and outgoing traffic and not traffic moving between the hosts of the internal network.

Preprocessors

Preprocessors are plug-ins which can be used with Snort to rearrange or modify incoming packets before they reach the filtering engine. The Snort preprocessors generally fall into two categories. They can be used to either examine packets for suspicious activity or modify packets so the detection engine can properly interpret them. An “examining” preprocessor has the ability to detect non-signature-based attacks that the detection engine cannot track down. Such

attacks include port-scans where a state has to be saved between many different connections to determine whether there is a port-scan occurring.

The second kind of preprocessor normalizes incoming traffic so the detection engine will be able to properly detect attacks. Incoming packets can for example be fragmented because the original packet was too large. This creates a problem for the detection engine since the harmful part of the package might be split into two different packets, which results in the detection engine failing to trigger an alarm. Preprocessors are here to mend this situation by stitching the two fragmented packets together again. Packet fragmentation usually happens when packets travel between different types of networks. I.e. if a packet travels into an Ethernet network and is larger than the MTU (Maximum Transmission Unit) for that network the packet will be split into two or more packets to be reassembled when it reaches its destination.

Rules

Snort rules are divided into two logical sections, the rule header and the rule options. The header contains the rule's action, protocol, source and destination IP addresses and netmasks, and the source and destination ports information. The rule option section contains alarm messages and information on which parts of the packet should be inspected to determine if the rule action should be taken. Actions that can be taken include logging the activity, dropping the packet all together, reject it (with TCP, RST or ICMP port unreachable), and pass (ignore the packet).

```
1 alert tcp any any -> 192.168.1.0/24 111 \
2   (content:"|00 01 86 a5|"; msg:"mountd access";)
```

Listing 2.1: Example Snort rule

Listing 2.1 defines a rule to generate an alarm if there is traffic coming from any ip or port to a local address 192.168.1.0/24 on port 111 with a payload containing the binary data "`00 01 86 a5`". The binary sequence "`00 01 86 a5`" can be located anywhere within the payload for a match to occur. When that happens an alarm will be logged with the message "mountd access". Backslash is used to divide the rule onto multiple rows, the default interpretation is otherwise one row one rule.

2.2.2 BASE

BASE [15] stands for **B**asic **A**nalysis and **S**ecurity **E**ngine and is designed to read and display the logged alarms Snort has generated. Alarms are displayed to the user via a web interface written in PHP.

Figure 2.5 gives a statistical overview of what has been logged by the different sensors such as the distribution of different protocols of the logged alarms. There are also links to the most common views that an administrator will be interested in keeping an eye on, e.g. most recent alarms and most common alarms. The views can be further specialised by using the search function where

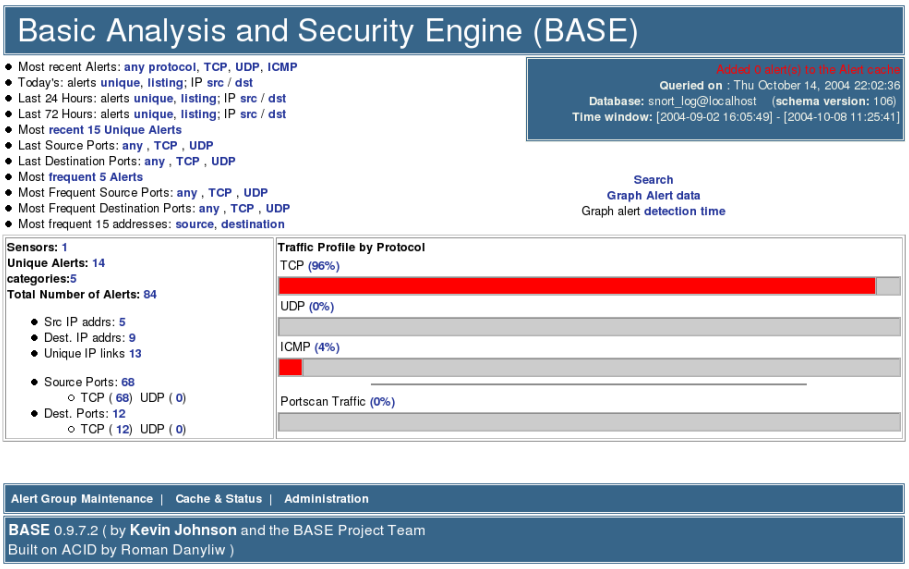


Figure 2.5: BASE home screen.

the administrator can be more precise about what he/she is looking for in the database.

BASE also gives the user the capability to add alarms to groups making it easier for the user to spot trends in the incoming alarms. Adding alarms to groups is an entirely manual process. Alarms that have been placed in a group can then be viewed in a special tab of their own.

To give the user a better overview of the most recent attacks, BASE generates graphs using statistics Snort has gathered. The user controls what is displayed in the graph via comprehensive controls. Simple graphs, such as number of attacks over time can be generated as well as more in-depth graphs only displaying statistics for a single host or port.

2.2.3 Snorby

Snorby [27] (Figure 2.6) is, like BASE, a web-interface for Snort and has many of the same features. It is developed using *Ruby on Rails* [38] and uses many of the modern web development techniques available today, such as Ajax and Flash. The main difference to BASE is that Snorby is built with collaboration between users in mind. All users have two main collaboration tools at their disposal, marking events with a star and leaving comments.

Marking events with a star is similar to placing a bookmark. The user simply clicks on the star icon associated with that particular event to mark it. Starred events are gathered in a separate list common to all users, meaning if one user stars an event everyone else will be able to see it as well. This promotes collaboration in that all users can work together to find interesting or important

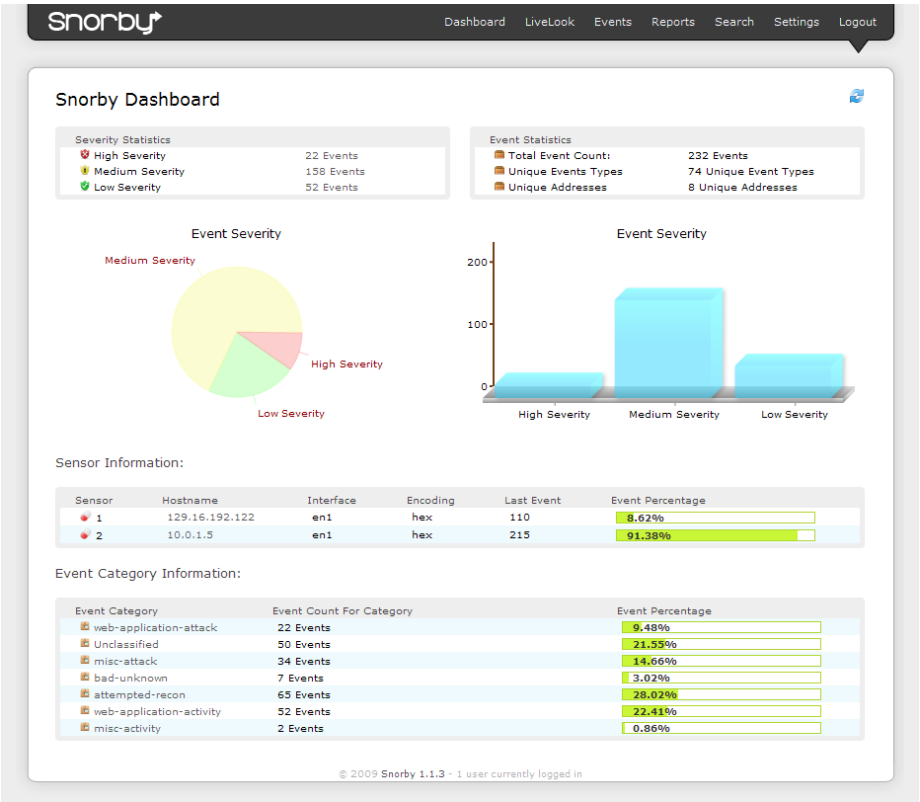


Figure 2.6: Snorby

events and investigate these further. This also works well with the other collaboration tool in Snorby, the commentary system. When starring an event, the user can leave a comment explaining why that particular event is of interest or need further investigation. At a later time, when an administrator logs on to the system she can view those starred alarms, read the comments, review them, and take an appropriate action.

Snorby also has the ability to generate automated reports that are mailed to the system administrator. These reports contain summary statistics about what kind of alarms the Snort sensor has generated the previous week as well as more in-depth information about important alarms.

2.2.4 Cisco MARS

MARS [4] stands for Monitoring, Analysis, and Response System, and is a network security suite developed by Cisco. In contrast to Snort, BASE, and Snorby this system costs money to buy and install in a network. But it also has much more to offer than the previous systems. MARS is more than just a network sensor that looks for malicious traffic on the network. It is more like a command central for all systems that have a security role in the network, such as firewalls, network sensors, antivirus programs, and much more. It combines information from many different sources into one interface that is easier to overview, and more than that it also correlates the collected information.

By correlating the information, MARS can detect chains of events that are related to each other across all the different sensors. If an attacker starts attacking a computer and tries to gain root access this can potentially be detected in a number of different places. The attacker might try to exploit a vulnerability on a mail server, which is visible from outside the network. Before the attack reaches the mail server it has to pass the firewall and probably a network sensor similar to Snort. If it gets past the first firewall, the mail server itself might have a firewall of its own. Lastly the mail server will have a system log that stores messages about local events, i.e. if someone tries to login on the computer a log message will be created. MARS is fed all this information from all these different sources and correlates them into a form more easily overviewed. The operator can then see exactly what has happened from start to finish in the attack, and even if it succeeded or not. MARS will also try to give suggestions about where the attack can be mitigated most effectively so it does not happen again.

Cisco MARS also has the ability to collaborate with other similar MARS systems that are deployed around the world. Useful information about attacks are shared between these to help detect new and unknown attacks. For example, blacklists used to block spam mail are also used to detect suspicious incoming traffic. Traffic sent from a computer known to be a spammer gets logged and reported to an operator automatically. This can be useful to know as much of the spam mail sent today are sent from bot nets operated by criminals. Knowing if such traffic suddenly increases might give an early warning that something bad is about to happen and counter measures can be deployed in time.

Chapter 3

Previous Work

This chapter presents different methods helping the operator of an IDS handle the amount of alarms produced, with a focus on false positives.

The algorithms behind the methods presented below are nothing new and have been used within research since the 80s. One can follow how the area matures in the literature through the decades. The same methods used to find anomalies [36, 10, 22, 14, 11, 32, 21, 20] and detect misuse [36, 13, 23, 19, 37, 18] are used today to form more abstract higher level alarms out of specific lower level alarms.

Reducing the amount of data is not a novel idea either [12]. Focus has shifted though with the growing hardware capacities, from reducing the amount of data for the benefit of the system to the benefit of the operator. Frank [12] also breathes the hope that with good enough rules, systems and/or training examples, false positives and false negatives can be brought down to a minimum. This hope is rebutted by Axelsson [3] where he makes calculations on the base-rate fallacy and points out the consequence, that false alarms are something we will probably never get rid of as the false alarm rate will have to be impossibly low for that to happen. This seems to be a generally accepted truth and the way to reduce false positives today is to apply the methods presented below thus hopefully minimizing the effect from false positives.

3.1 Aggregation

The meaning of aggregation varies greatly between reports and we therefore feel the need to define the word ourselves.

Definition 1. AGGREGATION *is the process of grouping alarms together based on certain user-defined criteria found in alarms.*

Alarms can be aggregated based on any attributes found in alarms such as: destination IP, alarm type, content of data portion.

Of the methods presented in this chapter, aggregation is the most low level and straight forward of them all. Hence, it is not common in the literature to focus solely on aggregation. Aggregation is now more of a ubiquitous step in any scheme processing alarms.

Debar and Wespi are among the few still explicitly mentioning aggregation even though very little attention is given to it, mainly focusing on correlation [9]. They describe how to aggregate and correlate alarms with the ACC – aggregation and correlation component. The task of which is to present a condensed view of the input received from multiple sensors and ACCs. Incoming events are processed by extracting common information and finding a match in previous observations. Correlation relationships are then formed by looking for **duplicates** and **consequences**.

A duplicate is an alarm whose cause have already been taken into account by the ACC algorithm. How alarms are judged to be duplicates is specified in a configuration file. A consequence is an alarm that under some conditions gives reason to expect others to follow. For a following alarm to be considered a part of the consequence chain it must occur within a given interval of its predecessor. As with duplicates, consequence chains are declared in a configuration file.

Lastly, the correlated alarms are aggregated into **situations**, where alarms have certain characteristics in common. This step is performed since isolated events are often considered insignificant. A situation is defined by four terms: *alarms class*, *source*, *target* and *severity level*. The user is presented the **situations** which are fewer in number than the original unprocessed alarms.

3.2 Correlation

Aggregation of alarms still leaves a lot of alarms to be inspected and further methods have been explored to crop the number of alarms presented to the user. The term correlation is used for a variety of methods to compress the number of alarms displayed for the user. To eliminate ambiguity we again feel the need to make a definition.

Definition 2. Correlation *is the process of building data models from alarms using external knowledge not found within the information carried by the alarms themselves.*

External knowledge might be such as how alarms appear together, or the impact attacks have on the network. The idea is that correlation will suppress the impact of false positives by lifting the alarms to a higher level of reason and putting them into context. False positives are marginalized and the true positives are put into focus.

In this chapter, two methods of correlation are presented, correlation using **conditions** and correlation using **probabilities**.

3.2.1 Using conditions

Correlation using conditions continues the chain of thought presented by Debar and Wespi [9] for aggregation, some alarms are **consequences** of others. The idea is that most intrusions are not isolated and are part of a chain of actions where earlier actions are preparing for later. Each alarm is associated with a set of pre-conditions to be fulfilled for the attack to succeed, and a set of post-conditions describing the aftermath of the attack, if succeeded. Attack chains are formed by connecting the pre-conditions of one alarm with the post-conditions of one or several other alarms. The attack chains are meant to 1) result in fewer items for the operator to inspect and 2) remove false positives since they do not belong to any attack chain.

As with the definitions of aggregation and correlation, there is no consensus regarding denomination. Ning et al. [28] have developed a framework using **prerequisites** and **consequences** as opposed to pre- and post- conditions. The alarms and their connections are modelled onto graphs where nodes corresponds to alarms and edges represents the relationships between them. If an alarm has been decided to prepare for another, they are connected by an edge.

The resulting chains are intended to be analyzed by human users to understand the correlated alarms as well as the strategies behind them. But the graphs can get fairly big due to the amount of alarms. For a user to be able to manage the graphs they have to be cropped. Ning et al. propose filtering out irrelevant alarms and aggregating alarms of the same type as means. Here, aggregation shows up as a vital part of the process without being given the same focus as in Debar and Wespi [9].

Cuppens et al. [6] have developed something similar to Ning et al. [28] but do not use graphs as a means to model alarms and relationships. As part of the project MIRADOR, CRIM was developed as a cooperative module for IDSs. It manages, clusters (groups alarms into sets corresponding to the same occurrence of an attack), merges (creates a new alarm that is representative of the information contained in the various alarms belonging to this cluster) and correlates alarms.

Since an intruder, most likely, has to carry out several attacks in a certain order to reach his goals, Cuppens et al. defines the task of the correlation function as anticipating the intrusion plan. A set of candidate plans of the current intrusion is returned by the function. The plans are denominated candidate since the intruder might not have reached his goal yet. The result is used by the reaction function to help the operator take the best course of actions against the intruder.

When an alarm is received, the database is checked to see if the pre- conditions of the new alarm can be linked to any post-conditions of stored alarms. If it can, the correlation conditions are checked. An algorithm is then applied to the pair to see if it might be merged into an existing attack scenario, if not a new is started.

Another variation of this theme is the **requires/provides** model by Zhou et al. [39]. Whereas CRIM by Cuppens et al. [6] and the model suggested by

Ning et al. [28] use **predicates** to describe the state of the network, Zhou's model is based on **capabilities** which describe what the attacker can access on the network. Inference rules are drawn between the capabilities, and if all capabilities of **requires** (cf. pre-conditions of Cuppens et al. and prerequisites of Ning et al.) can be inferred from the capabilities of **provides** (cf. post-conditions of Cuppens et al. and consequences of Ning et al.) the two alarms are correlated.

3.2.2 Using probabilities

The use of conditions for correlation, see previous section of Ning et al. [28], Cuppens et al. [6] and Zhou et al. [39], is a black and white procedure. Either all of the conditions for correlation are fulfilled and the alarms are correlated, or it fails. Correlation using probabilities adopts a grayer view of the correlation process. Instead of all or nothing, the probability of an alarm being correlated to another is calculated. That probability is then used when deciding if the alarms belong together.

Dain and Cunningham [8] have developed an algorithm for fusing and combining alarms into **scenarios**. A scenario is described as a sequence of alarms grouped together as they share a common cause. Whole scenarios are labeled as false instead of individual alarms thus simplifying for the operator. Each new alarm is compared to the existing scenarios using data mining techniques and the probability of the alarm belonging to that scenario is calculated. The alarm is then added to the most likely candidate, the scenario yielding the highest probability. If none exists a new one is created.

The most flexible way to fuse an alarm into a scenario would be to consider all possible ways of combining each new alarm with all previously collected, recalculating for each alarm how the biggest scenarios can be formed. This would quickly become computationally infeasible. Obviously, this is not possible. Instead an **atom model** is proposed. The new alarm is compared to all so far constructed scenarios and merged with the one returning the highest probability of correlation. If all scenarios return a probability lower than a preconfigured threshold, the alarm joins a new scenario. Once assigned a scenario the alarm is not reconsidered or reassigned, hence the term, atom model. This procedure might generate errors but greatly simplifies deciding how to assign alarms.

3.3 Finding the normal

Up to 90% of all alarms have benign causes such as misconfigured hardware and buggy software [16]. By "finding the normal" we refer to the process of finding the alarms caused by benign events. To be able to identify and remove these before applying the correlation methods mentioned in previous section holds a lot of promise. The amount of data that is processed by the models is significantly decreased, as well as the data presented to the user.

A very straight forward method is applied by Gengo and Clifton [5]. By using a data mining technique called frequent episodes, common and recurring

sequences of alarms for a given site are identified. These are then manually analyzed to see whether they result from normal operations. With the help from these alarms, site-specific filters can be constructed which may help reducing the amount of alarms presented to the administrator.

Julisch [16] is of the same idea as Gengo and Clifton but attacks the problem from a higher level of abstraction. Alarms are grouped in clusters and for each cluster a meta-alarm, a generalized alarm, is derived. A generalized alarm is a pattern which the alarm created by the IDS must match in order to belong to that cluster.

Julisch focuses on bringing alarms into clusters where each cluster represents the **root cause** behind the alarms. A root cause is defined as a "problem that affects components and cause them to trigger alarms".

To be able to generalize alarms a single rooted and connected directed acyclic graph (DAG) called a generalization hierarchy is constructed over each attribute, of the alarm. Each parent node in the graph is a more generalized attribute than any of its children nodes. To generalize an alarm, its attributes are located in the DAG. The attributes of the alarm are then replaced by the parents. The problem with DAGs is that there might be more than one parent to a node. When generalizing alarms one can face problems when there is more than one choice. Julisch proposes two strategies to resolve this:

- **choose-one** - user-defined rules decides which parent to choose.
- **explore-all** - pursues all possible generalizations in parallel and retains the one that first leads to a generalized alarm of a user defined size.

A somewhat different approach is used by Manganaris et al. [26] Instead of filtering out or removing the benign alarms as Julisch [16] and Clifton and Gengo [5] do above, they are viewed as something normal. The idea is presented as applying anomaly detection on the output from a misuse detection system. A base template of what the normal alarms are and how they behave is established, when the "normal" alarms suddenly change in frequency or in sequence, an alarm is raised. The basic idea Manganaris et al. have is that repetitive behavior over long periods of time is in all likelihood normal. Combinations of certain alarms occurring in close temporal proximity of each other and always in the same order, for an extended period of time are deemed less suspicious than a sudden burst of novel alarms. To discover the sets of benign alarms occurring frequently, association analysis is used on the bursts of alarms.

3.4 Ranking

Correlation in Section 3.2 and filtration in Section 3.3 helps the operator a great deal, reducing the number of items to go through. They do not however help the operator to rank and process the result in regard of importance. This is addressed by Almgren and Jonsson [2] and Porras et al. [30]. The two reports have diametrically opposed methods to rank alarms.

Almgren and Jonsson [2] pose the question: "is it possible to capture the preferences of the site security officer (SSO) by monitoring her normal interaction with the IDS?" or put a little differently: "is it possible to prioritize alarms based on the feedback from the user?" Their aim is to develop a system that learns how to rank alarms based on the input from the user and local operational conditions. This model assumes the use of a graphical user interface to display the alarms and interaction from the user in the form of clicking on alarms. Four interaction modes are presented:

- **Clicking** - the system stores information on what alarms the user clicks on. Assumes that alarms clicked on is of greater interest than those not touched. Ranks those with a larger amount of clicks higher than those with a lesser amount.
- **Directed learning** - this is an extended form of clicking. On top of the normal procedures the user has the choice of specifying to the system why she finds this particular alarm interesting.
- **Feedback** - the system ranks the alarms and specifies for each a reason why it is raked the way it is. The user then has the option of either agreeing or disagreeing with the ranking. This is in a way directed learning in reverse. Instead of the user telling the system, the system is telling the user.
- **Explicit questions** - this is another extension of clicking where the system occasionally asks the user how to rank an alarm.

Almgren and Jonsson estimate the first interaction mode, clicking, to be the base case of all proposed interaction modes and concentrate on this first. The work flow of the base case is intended to follow

1. alarms are presented based on the current ranking function.
2. the SSO reviews the ranked list to assess the most important alarms.
3. the SSO investigates the most important alarms by clicking on them.
4. the system evaluates the current ranking function by checking what alarms were clicked and if these indeed were ranked highest. If not, the system reevaluates the ranking function using the feedback from the clicks for improvement.
5. a new view is presented to the user based on the new sample ranking function.

The system presented by Almgren and Jonsson operates on an on-the-fly configuration. The ranking M-correlator devloped by Porras et al. [30] operates on utterly opposite premises. It is dependent on configuration files describing the topology of the system, the most critical assets and alarms of interest for the operator.

As can be seen above, the final ranking score is dependent upon a multitude of variables. The correlator starts though with filtering out alarms the operator has expressed low interest in. On the alarms passed by the filters a relevance score is calculated for each, this corresponds to the *relevance* node in the graph. Each alarm is compared against a map of the topology of the network containing

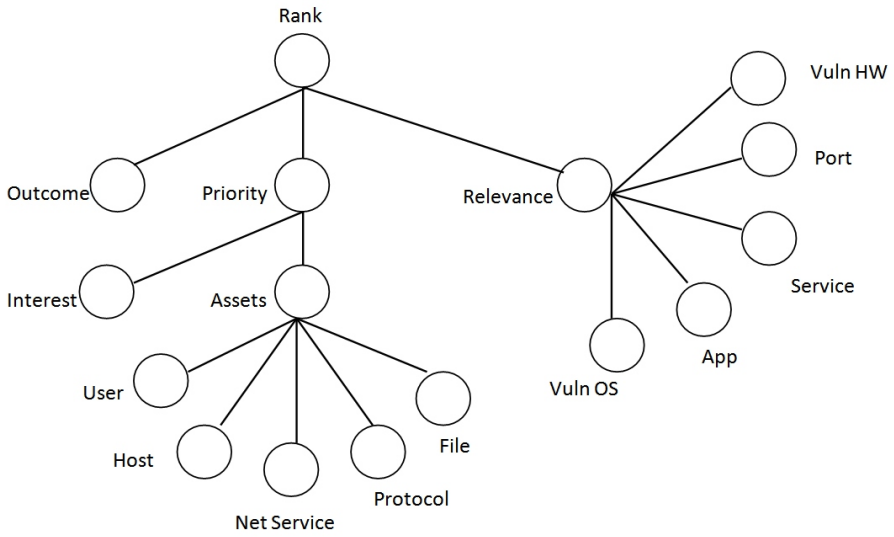


Figure 3.1: The Bayesian network used when calculating the final rank score for alarms in the M-correlator by Porras et al. [30].

information about such things as running applications, OS, services, open ports, to see how relevant the alarm is to the network. The M-Correlator includes an *Incident Handling Fact Base* which provides information about vulnerability dependencies. E.g. an alarm pertaining to a Microsoft Server would yield a low relevance score in an environment running Apache. The map of the network is automatically generated by the tool *Nmap*. The M-Correlator moves on to calculate the priority of the alarm. This relies on two variables (*priority* node in the graph). One is to what degree the alarm is targeting data assets and services most critical for the client users. These are enumerated in a configuration file by the operator. The other concers which alarms are of greatest interest to the operator. This is also specified by the operator in a configuration file. Lastly, an *incident rank* is assigned to each alarm based on the previously calculated relvance and priority scores, reflecting the possible impact the attack would have on the system and the likelihood of its success (*outcome* in the graph). Once an alarm has been given an incident rank the M-Correlator tries to merge related alarms together into higher-level security incidents which in turn are ranked. This is done by aid from the Incident Handling Fact Base which also carries information about what types of alarms can be merged.

The relevance score, priority calculation and incident rank are all computed by using a variation of the Bayes framework for belief propagation in trees.

3.5 User input

Only a few reports mention user input and only Almgren and Jonsson [2] focus exclusively on the interaction between system and operator. The interaction

can be passive with the system simply observing the user and trying to draw conclusions from these observations and any possible user input, or it can be active posing specific questions to the user, initiating the interaction. This makes the system dynamic in the sense that the algorithm behind the ranking possibly changes by every interaction.

The time the correlation system of Cunningham et al. [7] is exposed to the user is limited only to the user tagged sets it is trained upon before launch. This is the only interaction with the system, making it a static one, since beyond the initial training set, there are no modifications to how the algorithm processes the alarms.

Attempting to reduce the number of false positives Pietraszek [29] first lets the user classify a set of alarms, then machine learning techniques are used on training examples the system derives from the original set of user classified alarms to generate rules of how to classify. The alarms classified by the rules are then sent to the user for optional scrutiny. Alarms are re-classified as needed. Although the user is very involved in the process of classifying alarms, this is very little highlighted by Pietraszek. Nothing is mentioned about the process of classifying the starting set of alarms, or the process of reclassifying, or about how the interaction functions, or what kind of input the system expects and what kind of feedback it gives.

As mentioned in Section 3.4, the mission based system of Porras et al. [30] involves the user, although quite extensively, only during start-up, similar to Cunningham et al. [7]. The user has to enumerate critical data assets and services and specify an interest profile describing which classes of alarms are of greater concern. However, there is no interaction with the user once the system is on line and there is no discussion about giving feedback to the system or what to do if the user is dissatisfied with the ranking. However, Porras et al. describe an adaptive mode concerning the conditional probability tables (CPTs) under the section describing the Bayes calculation. Here the analyst presents simulated alarms to the system which are then ranked. For each alarm, the ranking is either accepted or changed to a more accurate one by the analyst. The effect will be that the CPTs change to some extent to better mirror the preferences of the analyst. This adaptive mode is never evaluated or mentioned again in the remainder of the report.

All schemes have user input somewhere in the chain. Somewhere and someplace there has to be a human involved in the process to either analyze an alarm, describe the critical aspects of a network, evaluate performance, separate unimportant alarms form the important or deciding if it is a true or false positive. Clifton and Gengo [5] described in Section 3.3 need the user to determine whether a frequent episode is benign. The same goes for Julisch [16] in the same section. The generalization hierarchies must all be defined by the user and contains the knowledge of the user about the system and about alarms in general.

In the next chapter we discuss how the methods presented here relate to our solution and what aspects were taken under consideration.

Chapter 4

Methodology

In order to investigate current methods of prioritizing and managing alarms, we have studied current research in form of published reports and books, as described in Chapter 3. We have also interviewed four professionals.[1] Based on this material, we had a number of options to choose from when deciding on a possible improvement to an existing IDS. The choice finally settled on the implementation of a ranking functionality in the architecture Snort/Snorby.

Choosing method for implementation

Aggregation was the first to be rejected due to its simple and confined nature. Correlation (see Section 3.2) is a labour intensive and time consuming method in regard to identifying and defining pre-conditions and post-conditions. This makes correlation an activity incompatible with the continuous task of inspecting incoming alarms. As our interest lies with the everyday interaction a user has with the IDS, correlation is rejected. Set up of filters is done prior to the deployment of the IDS. The setup process is in no way part of everyday operations making this less interesting.

Ranking is a necessary step whether the system employs aggregation, correlation, ranking or a collaboration of them all. As long as there are more than one alarm to inspect, the operator has to make a choice in which alarm to inspect first. That choice is the essence of ranking - which is the most important alarm to look at? For these reasons we have chosen to implement a variation of ranking.

Choosing architecture to make implementation on

To be able to make improvements to an existing IDS we needed access to the source code, which is easier with open source projects. Snort is an obvious choice because of its longevity and dominance within its field. Of the interfaces available for Snort only two seem to be in use today, BASE and Snorby, where

[1] See appendix D.

we chose the latter for this project. BASE is written in PHP and Snorby in the framework Ruby on Rails. The latter facilitates implementation and leaves more time to concentrate on the logic of the implementation. Finally, Snorby is actively developed, meaning that our contributions may be used in future releases.

4.1 How to implement ranking

There are two distinct methods for ranking, either heavily configured [30] or on-the-fly configuration [2]. Out of these, the approach by Almgren and Jonsson [2] seemed the most interesting, focusing on the ongoing interaction between user and system. The system in Porras et al. [30] has a life-cycle similar to filtering, only deployed once in the beginning. Any changes made to the configuration files results in a restart of the system. We believe a dynamic system that can learn is more useful to the operator.

The first two problems immediately encountered after settling on Almgren's and Jonsson's [2] mode of interaction, were how to communicate to the system why certain alarms are important and what information of the alarm to use when deciding on a rank. There is of a lot of information such as destination IP, source IP, destination port, source port, alarm signature, and/or patterns in payload that might be of interest to a ranking system. A number of ways, more or less time consuming were discussed during the course of finding a good interaction model with the IDS. These are described in the following. For the final method, see method 3 on page 37.

Interaction method 1: check boxes

One of the discussed and rejected suggestions was the use of check boxes (Figure 4.1). I.e. each alarm is presented with some check boxes, one each for the attributes chosen for analysis. As we have chosen to follow the attributes destination IP, source IP, destination port and alarm signature, our case results in four boxes. The check boxes corresponding to the reason of interest are checked and the result submitted to the IDS. The same suggestion was also presented by Emilie Lundin Barse during her interview.[2] Hopefully, this will let the IDS make rather precise predictions about new alarms, whether the operator will find them important or not. Over time the IDS will gather more information and improve its performance, saving time for the operator as the critical alarms will already be flagged. This holds a great benefit over the original clicking method proposed by Almgren and Jonsson [2] where the system did not know why an alarm was clicked and in a sense had to guess how to rank the alarms.

The problem and the ultimate reason for its rejection, is that we wanted the necessary interaction to be as minimal as possible. If this model is extended to track additional attributes, more check boxes will have to be added and the users interaction with the system will become more complicated and time consuming.

[2]Emilie Lundin Barse (information security consultant, Combitech, interviewed in person by the authors April 12, 2010), see appendix D.3.

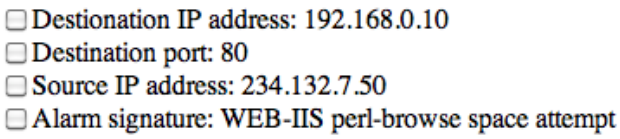


Figure 4.1: Conceptual design of check boxes for interaction with the ranking system.

Interaction method 2: only clicking

A better approach would be to gather information in the background in a more non-intrusive way but still get adequate information to make meaningful predictions. The first method suggested by Almgren and Jonsson [2], clicking, fits our objective. It silently records on what alarms the user clicks and tries to make predictions about rank.

Clicking in its original description has a shortcoming. It is only described as an incrementing value, meaning a value either grows with clicking or remains as is. But what if one would like to lower the priority of something? I.e. there is a server with a serious security flaw. All alarms of a certain type targeting the server is now of immense interest. In all, a few weeks pass by where all such alarms are dutifully clicked and examined, until the vendor of the server software releases a patch. All of a sudden, the previously important alarms are now a hindrance. Here, it would be very beneficial to be able to rank these alarms down and stop them from cluttering the view.

Interaction method 3: positive and negative feedback

The solution decided on, is used in a number of community news sites such as Digg and Reddit and also by the video-sharing website YouTube. It gives the user the ability to give positive and negative feedback to the system through two buttons associated with each item. The system will change the rank of that item according to the vote. News stories and videos appreciated by a lot of votes will be ranked higher than other news stories and videos.

Figure 4.2: The thumbs up button and the thumbs down button to press for liking respectively disliking a video on YouTube.

Our ranking model works in a similar manner, but instead of ranking news stories it ranks alarms generated by Snort. There are two buttons associated with each alarm, a "+" button and a "-" button. If the user clicks on one of the buttons she gives that alarm a higher or lower priority respectively.

In reality, alarms are not clicked because of a single attribute.[3] Instead, it is

[3] Explained by Ola Söderström (security analyst, Omegapoint, interviewed in person by

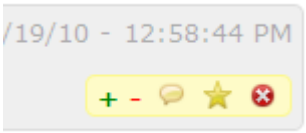


Figure 4.3: Final design of buttons for negative and positive feedback. "+" is for positive and "-" is for negative.

often a combination of attributes and external knowledge not found in alarms that make alarms interesting.[4] The ranking system tracks four basic attributes; source IP, destination IP, destination port, and the signature of the alarms. These are chosen since they are assumed to often be of interest to an operator.

There is a counter keeping track of the number of up and down votes a specific attribute receives. When voting for an alarm, each of the four attributes receives a vote. Prioritizing (giving positive feedback to) an alarm increments the counters for each of the four attributes associated with that alarm by one. I.e. an alarm receives an up vote because of its destination port 79, the counter for port 79 is incremented together with the counters associated with the values of destination IP, source IP and alarm signature. The reason why the counters for all attributes are incremented when clicking on an alarm is that the system has no idea why a user clicks on an alarm. Since there is no way for it to guess the reason, all it can do is to reward each attribute with one point.

This method of positive and negative feedback is less exact than the method using check boxes described above, keeping the element of guess from regular clicking. But, it is possible to add any number of attributes to the chosen method without any difference to the user, something not possible with check boxes as described above. Our hope is that the ranking system will recognize what attributes are of importance when the user is able to give negative feedback on alarms.

4.2 Time based weighing of votes

One problem with this set-up is that old votes influence the current rank as much as new votes. I.e. consider two alarms A and B. If A received 50 votes six months ago B would have to receive 51 votes before it is ranked higher by the system. Most likely, this makes the ranking system slower and slower in response to changes in the current level of threat. As new vulnerabilities crop up and get patched continuously, this is not good. The ranking system should be able to respond to new threats and give them a higher rank when relevant, and as the threat disappears, let its rank dissipate over time.

Another problem is that votes for an attribute might fluctuate a lot over time as Figure 4.4 shows. This poses a problem for the ranking system if it were to use

authors April 23, 2010).

[4] By external knowledge we refer to information such as: is the source responsible for many attacks recently, has the destination host been present in many alarms recently.

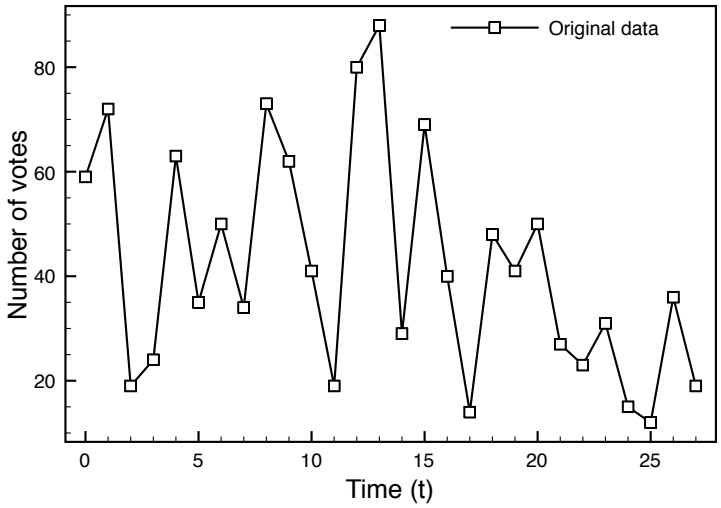


Figure 4.4: Figure illustrating how voting measurements can fluctuate over time.

the number of votes directly as a score for the attribute. The ranking they get would be very inconsistent from day to day and very hard to predict. Instead it would be much better to use some sort of average that has a slower rate of change.

To overcome the problem of equal influence of votes cast far apart in time and fluctuations in the number of votes, we use an **Exponential Moving Average** (EMA). EMA applies weights to data points (or votes in this case), where the weights will become smaller as the data points get older. The formula of EMA is defined as follows:

$$S_t = \alpha \times Y_t + (1 - \alpha) \times S_{t-1}$$

- Y_t is the observation at a time period t.
- S_t is the value of the EMA at any time period t.
- The coefficient α decides the degree of influence the observed value has on the new EMA value. α is a constant smoothing factor between 0 and 1. A higher α discounts older observations faster. Alternatively, α may be expressed in terms of N time periods, where $\alpha = 2/(N + 1)$. For example, $N = 19$ is equivalent to $\alpha = 0.1$. The half-life of the weights (the interval over which the weights decrease by a factor of two) is approximately $N/2.8854$ (within 1% if $N > 5$).

For a given set of observations $\{Y_0, Y_1, ..., Y_n\}$ this formula returns the EMA value S_t for any time period $t \geq 1$. S_1 can be initialized in a number of different ways of which the most common is to set $S_1 = Y_0$, but a more accurate way would be to set S_1 to be the average of the first 4 or 5 observations. However,

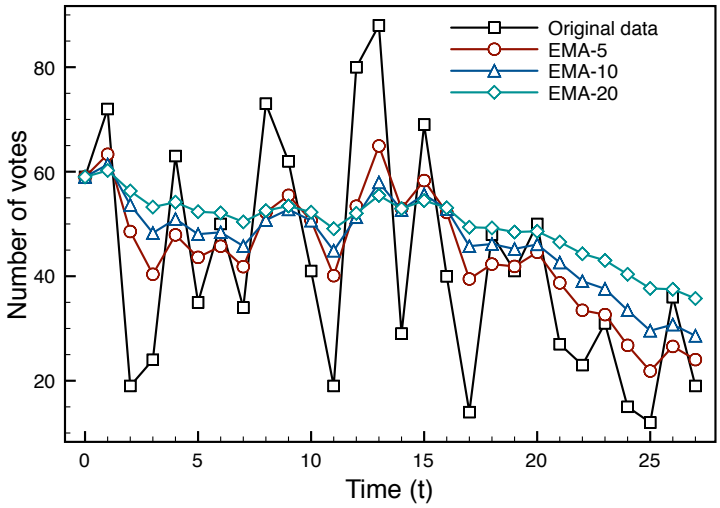


Figure 4.5: Plotted curves showing the effect of using different smoothing factors.

the error created by setting $S_1 = Y_0$ dissipates over time; a smaller α makes the choice of S_1 relatively more important than a larger α value.

The EMA formula consists of two parts; the first part that calculates the influence of current observations ($\alpha \times Y_t$), and the second part which calculates the influence of old observations ($(1-\alpha) \times S_{t-1}$). The value α and $1-\alpha$ decides the influence that each part has on the new EMA value. These two parts are then added together to form the new EMA value.

The old observations are represented with only one value (S_{t-1}) in this formula, which is the previous EMA value that was calculated. However if we expand S_{t-1} in the second part of the formula we get the following:

$$S_t = \alpha \times Y_t + (1-\alpha) \times (\alpha \times Y_{t-1} + (1-\alpha) \times (\alpha \times Y_{t-2} + (1-\alpha) \times (...)))$$

We see that it is a series of calculations where S_{t-1} (when expanded) holds the value of all previous observations. So to be able to calculate S_t we first have to calculate $Y_0, Y_1, ..., Y_{t-1}$ until we finally get to S_t. This is the way it is going to work in our ranking system. For each new observation of how the users vote we recalculate the EMA value using the current amount of votes, the previous EMA value, and an appropriate α value.

Figure 4.5 shows the behaviour of the EMA formula with three different values of N. The line called *original data*[5] is the data set $\{Y_0, ..., Y_n\}$ that is used as input to the formula. In our ranking system this would be a set of cumulative numbers representing how the users voted for different attributes each day. *EMA-5*, *EMA-10*, and *EMA-20* in the figure each represents values produced with the

[5]This data set is a series of fictional votes an attribute received over a period of time.

EMA algorithm using 5, 10, and 20 respectively as values for N in the algorithm. The larger N, the smaller the smoothing factor α is, and as a result, irregularities are smoothed out more than with a smaller N.

As we follow the black line we can see that the number of votes for the fictional attribute fluctuates a lot between different time periods. This variation could happen for a number of different reasons, i.e. the attack might have some time constraint that makes it happen more on some days than others. This fluctuation is not good for our ranking system as it makes the rank very unpredictable. But when the EMA formula is applied the result is much smoother. The big jumps and drops in the numbers disappears. As big jumps or drops happens in the line named *original data*, *EMA-5*, *EMA-10*, and *EMA-20* will move in the same direction but not as much. This is because of the α value in the EMA formula, as it weighs the previous EMA value with the current value of the *original data* line.

As an example we can look at the first point for each of the four lines. They all start at 59 on the y-axis. The next point on the black line is at 72. To calculate where the next point for the EMA-5 line should be we use the following:

$$\alpha = \frac{2}{N+1} = \frac{2}{5+1} = \frac{1}{3}$$

$$\begin{aligned} S_1 &= \alpha \times Y_1 + (1-\alpha) \times S_0 \\ &= \alpha \times 72 + (1-\alpha) \times 59 \\ &= \frac{1}{3} \times 72 + (1-\frac{1}{3}) \times 59 \\ &= \frac{1}{3} \times 72 + \frac{2}{3} \times 59 \\ &= 63 + \frac{1}{3} \end{aligned}$$

This gives us the second point of the EMA-5 line. The same calculations can be made for EMA-10 and EMA-20.

Figure 4.6 shows another way to illustrate how much influence a given data point has after n time periods. At $t = 0$ the data point has as much influence as it can have, it is worth 100% of its original value. As time passes it will lose more and more of its original value in an exponential manner. The line representing EMA-10 will have lost about 50% of its original value after about 3.5 days. This corresponds pretty well with the formula for calculating the half life of an EMA value. With the formula this works out to: $10/2.8854 = 3.4657$.

4.3 Implementation of whole system

The implementation of the ranking model is done on top of Snorby. Snorby adds a few tables on top of the Snort database to support some of the additional

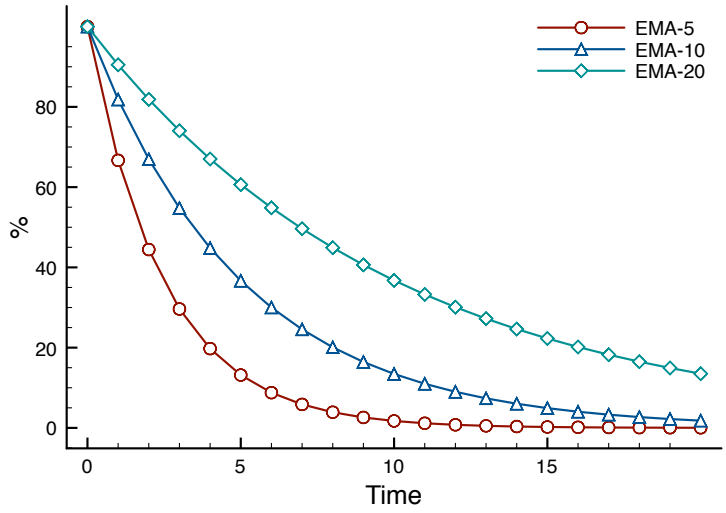


Figure 4.6: Figure showing how fast previous observations lose their influence in the EMA formula with three different weights. X-axis is the number of time periods since the observation occurred, and the Y-axis is the influence in percent that the observation has over the EMA value.

features that it offers, i.e. the commentary system. To add the ranking model on top of Snorby, the number of times a user gives positive and negative feedback to an alarm has to be counted. We decided that the simplest way to store user feedback was to add columns and tables to the existing database where Snort stores the alarms. This is accomplished by making two changes to Snorby; two buttons ("+" and "-") where added to each event (for positive and negative feedback), and vote counters where created for each of the four attributes.

Each attribute (i.e. an IP address 10.0.0.5 or a port 80) will have several counters associated with itself. Each counter will have a variable storing how many votes one attribute received during a day. These values will then be used in the EMA formula so the ranking system will be able to see trends in how the users of the system votes. Three different EMA values will be used to highlight voting changes over short, medium, and long time periods. But before we can do this we have to take closer look at the database structure that Snort and Snorby uses.

4.3.1 Snort database structure

The Snort database stores information about each malicious packet in a structure where the different parts of each packet are saved in different tables. It basically stores the whole packet but divided up in its different parts. Our ranking system will however only be interested in a small part of this information. When a Snort sensor stores an event (alarms are called events) in the database a unique identifier is generated for that event. This unique identifier, *cid*, is an integer the sensor generates and increments for each new event the sensor stores

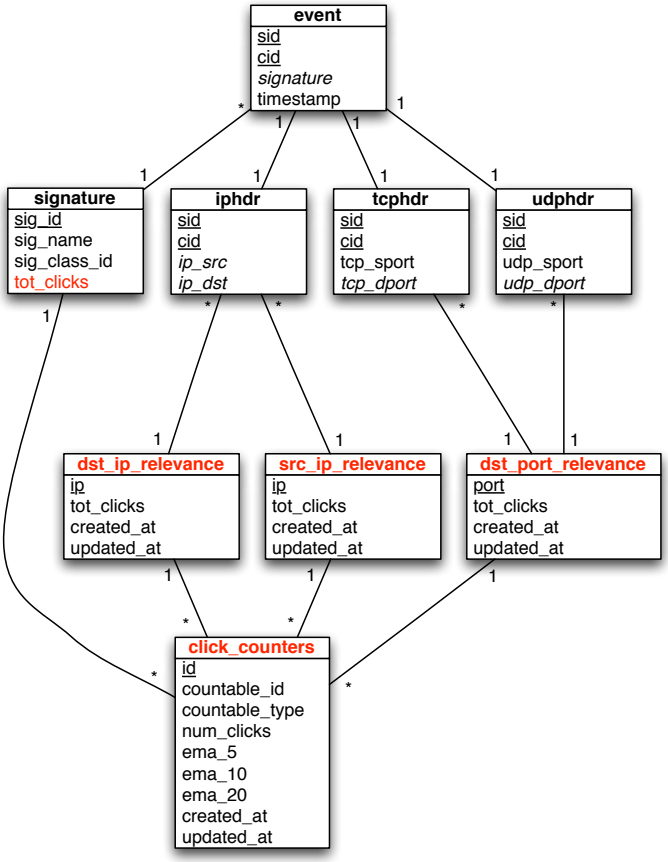


Figure 4.7: The Snorby database structure. This figure does not show the whole database structure, but only the parts that are affected by our implementation of the ranking model. Items marked in red indicate tables and values added to the already existing database structure.

in the database. To allow for more than one sensor to store events without risking collisions between *cid* values, a sensor id, *sid* is stored with every event as well. These two values (*sid* and *cid*) forms the primary key for all the tables storing information about captured packets. Figure 4.7 presents an overview of the Snort database structure. It also presents the tables that we have added to store the relevant information for each attribute to accommodate the needs of the ranking system.

At the centre of this database structure is the *event* table. It stores the *sid* and *cid* of each packet as well as a signature id and date and time about when it was captured. The rest of the information about a packet can be retrieved from other tables in the database as can be seen in Figure 4.7. This figure does not show all the tables that are available in the Snort database, but only the ones important for the ranking model.

Signature For each entry in the event table, there is a corresponding entry in the signature table as well, with the same *cid* and *sid*. The signature of an alarm specifies which filtering rule in Snort matched the current packet, and consequently caused it to be logged in the database. The signature table stores information such as the name of the attack, the id of the signature matching it, priority of the signature, and what version of the filtering rule in Snort capturing it. This can be useful if a faulty filtering rule is detected and rectified, as all events that where captured with the bad filtering rule can be identified and removed. However, the only information important in this table to the ranking model is *sig_id*, the id of the signature that captured the packet.

Iphdr Snort more or less follows the OSI standard in how it saves the information about packets caught by one of its signatures. The *iphdr* table is responsible for saving the source and destination IP of each packet. It will always have an entry for every event that is logged in the database as all packets traversing the network are IP packets.

Tcphdr Has only information from captured packets using TCP. This table has fields for all the information in a TCP header as defined by the OSI standard. The only information needed for ranking decisions in this table is the destination port.

Udphdr Similarly to the *tcphdr* table, this table only saves information about packets using UDP and has a column for every field in the UDP header. The necessary information here for ranking decisions is destination port.

4.3.2 How votes are counted

The database tables described above all save information on a per event basis, meaning there is no table that saves information for one specific IP address for example. This is however exactly what the ranking model needs to do. It has to save information for each of the four attributes: signature, source and destination IP, and destination port. To count the votes we have added a table named *click_counters*, which can be seen at the bottom of Figure 4.7. Each row in this table represents a counter that keeps track of votes for one specific attribute, such as an IP address or a destination port. To connect these counters to *signature*, *iphdr*, *tcphdr*, and *udphdr* we need three additional tables: *src_ip_relevance*, *dst_ip_relevance*, and *dst_port_relevance*. These three tables only store a primary key (i.e. IP address in the case of *src_ip_relevance*) and the total vote count (*tot_clicks*[6]) A new table to connect *signature* to *click_counters* is not needed as signature already stores information about unique signatures. Every counter has four values:

[6] *tot_clicks* is never used in any of the ranking calculations, it is only there because it might be interesting to an operator to see what the total voting tally is for an attribute.

- *num_clicks*: A cumulative value that saves the amount of "+" or "-" votes one attribute receives. If one attributes receives five pluses and four minuses, the value of *num_clicks* will be 1 since $5 + (-4) = 1$.
- *ema_5*: EMA value with a period of 5 days. Adapts the fastest.
- *ema_10*: EMA value with a period of 10. Adapts twice as slow as *ema_5*.
- *ema_20*: EMA value with a period of 20. Adapts twice as slow as *ema_10*.

These click counters correspond to the variables Y and S in the EMA formula (section 4.2, page 39) because it stores information about both the raw vote counter (*num_clicks*), and the EMA values (*ema_5, ema_10, ema_20*). Each counter is equivalent to one data point in Figure 4.5. A counter only collects data during 24 hours, after 24 hours a new counter is created for the same attribute. This creates a set of counters that each represents votes cast for one attribute during one day. If a user votes for an event that does not have a counter that is less than 24 hours old a new counter is created. The EMA values in the old counter is used together with the current amount of votes stored in the new counter (in this case 1, since it is new) as input to the EMA formula to create the new EMA values in the counter for that attribute. In the EMA formula these two input variables are called S_{t-1} for the previous EMA value, and Y_t for the current vote count. This process of recalculating the EMA values also happens every time a vote is cast by the user, but a new counter is only created if the current one is older than 24 hours.

These counters are only created as needed. When an alarm with attributes that have never received votes before is clicked a new click counter is created for that attribute automatically. It is done this way to save space in the database as creating them all from the start would require a lot of space. Creating click counters for all possible IP addresses would require $2^{32} = 4\,294\,967\,296$ entries in the database, which we do not have room for.

4.3.3 Ruby on Rails

Ruby on Rails (often referred to as *Rails*) facilitates retrieving information from the database by creating a so called *model* of each table in the database. Models are classes in Rails with automatically generated methods meant for retrieving and updating data in the table. Each column of the table gets its own set of methods. I.e. a table called *persons* will have a class called Persons in the Rails code, and each column of the table will have a method (i.e. `name`). Relations between different tables in the database will also be represented in a similar manner, but here the developer has to define them in the class definition. I.e. the *event* table is represented by an object *Event* in Rails that has methods for every database relation of the table. Calling Event.signature returns a *Signature* object corresponding to the row in the *signature* table the particular event had a relation with.

The relations between different tables in the database are described in the corresponding Rails objects by using a function call. There are four basic function calls that can be used:

- `has_one`
- `has_many`
- `belongs_to`
- `has_and_belongs_to_many`

These four methods can be used to describe one-to-one, one-to-many, and many-to-many relationships in the database. To describe a relationship between a table containing books and a table containing the libraries that the books belong to would look like this:

```
1 class Library < ActiveRecord::Base
2   has_many :books
3 end
4
5 class Book < ActiveRecord::Base
6   belongs_to :library
7 end
```

The object *Library* is defined with a call to `has_many` with argument `:books` to describe its relationship with another object *Book*. The fact that the argument is `:books` with a plural 's' is interpreted as a reference to the object *Book* and not *Books*, the 's' just makes the code more readable. On the other end of the relationship the class *Book* has a call to the method `belongs_to` with an argument `:library` to signify that each Book object belongs to one specific Library object. These methods also involve some magic that make it convenient when designing a web-application with Ruby on Rails from the start. For example all methods defining relationships make a few assumptions about what columns are present in the tables. The has_many method assumes the foreign key in the table containing books is named library_id. If the foreign key has some other name it can be specified as an argument in the method call.

In the case of the tables that have been added to the Snort database, the relationship that src_ip_relevance has to iphdr would look like:

```
1 class SrcIpRelevance < ActiveRecord::Base
2   has_one :iphdr, :foreign_key => :src_ip
3   ...
4 end
5
6 class Iphdr < ActiveRecord::Base
7   belongs_to :src_ip_relevance
8   ...
9 end
```

This ties the primary key column (named *ip*) of the src_ip_relevance table to the column named src_ip in the iphdr table. Rails then knows that all rows in the iphdr table belongs to one specific row in the src_ip_relevance table. dst_ip_relevance and dst_port_relevance have equivalent relationships defined in

their object definitions. Attached to all of these tables is a click_counter table as can be seen in Figure 4.7. This table has what is called a polymorphic relationship with four tables above it. That means that all of these four tables can have a relationship with the click_counter, and use their respective primary key as a foreign key in the click_counters table. A polymorphic relationship is defined as follows:

```
1  class ClickCounter < ActiveRecord::Base
2    belongs_to :countable, :polymorphic => true
3    ...
4  end
5
6  class SrcIpRelevance < ActiveRecord::Base
7    has_many :click_counters, as => :countable
8    ...
9  end
10
11 class DstIpRelevance < ActiveRecord::Base
12   has_many :click_counters, as => :countable
13   ...
14 end
```

On the second row in the definition of *ClickCounter*, every object of the class *ClickCounter* is defined to belong to something called `:countable`. Countable is not a class, but a name for a group of different classes that want to use this class to store information in its table. On the other side of the relationship *SrcIpRelevance* has a `has_many` relationship with the class *ClickCounter*, and with a second argument `as => :countable`. This tells Ruby on Rails that *SrcIpRelevance* wants to belong to the polymorphic relationship named "countable". Ergo, the four classes Signature, SrcIpRelevance, DstIpRelevance, and DstPortRelevance can all have a relationship with the same ClickCounter class instead of having a unique click counter class for each of the four attribute classes. Rails makes this possible in the database by having an extra column in the table click_counters, in addition to the foreign, key that saves the name of the class owning the object/row. This allows other classes to use their primary key as a foreign key in the click_counters table without causing any collisions between foreign keys from different classes.

4.3.4 Counting Votes

Snorby displays a page listing events from the Snort database in the order captured by Snort (shown in Figure 4.8). To allow the user to cast votes on events she thinks are important two buttons have been added, "+" for when the alarm is interesting and "-" for when it is not. Because the vote is cast on an event and not on each specific attribute, the ranking system cannot know if one, all, or a subset of the attributes are interesting. We decided on the most simple solution which is to share the vote evenly among the attributes of the alarm. For each vote an alarm receives, each attribute's score will be incremented or decremented by one point. Not deciding on individual ratios for each attribute

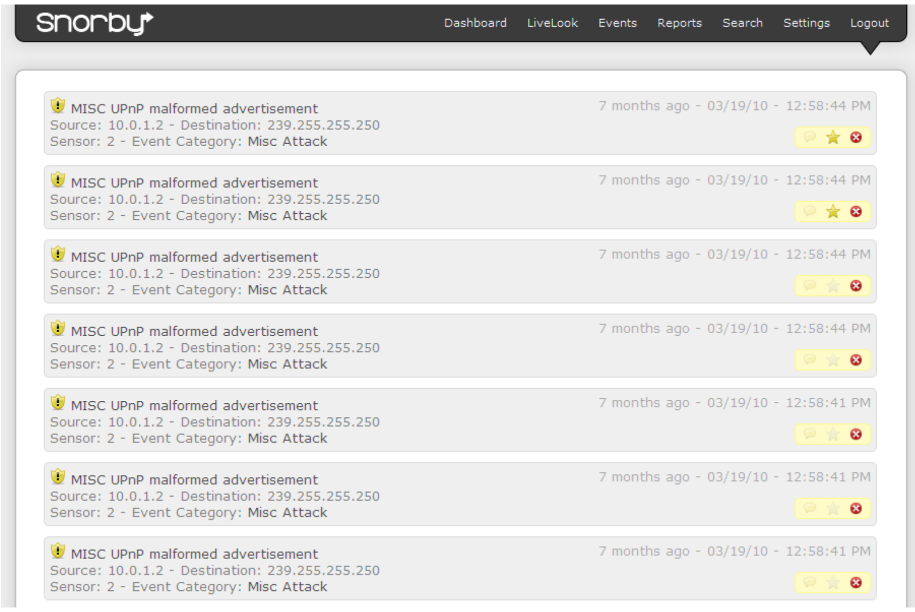


Figure 4.8: The event view in Snorby showing events fetched from the Snort database sorted in the order they where captured.

at this point, i.e. giving source ip two points and destination port only 1, gives us the opportunity to multiply the score later and produce the same results. I.e., if there is an alarm:

```
       Signature: Port scan
       Source IP: 230.42.25.6
  Destination IP: 105.200.156.2
Destination Port: 80
```

"Port scan", "230.42.25.6", "105.200.156.2", and "80" would each get one point added to their counters if the user pressed the "+"-button. The vote affects four different values in the attributes counter: num_clicks, ema_5, ema_10, and ema_20. num_clicks stores an integer value that is incremented or decremented depending on which button was pressed. It starts out with a zero meaning negative values in a counter are possible. The pseudo code for updating the three EMA values looks like:

```
def next(period, current, prev_ema)
  multiplier = (2.0 / (period+1))
  return multiplier * current + (1 - multiplier) * prev_ema
end
```

The *next* function above follows the EMA algorithm presented on page 39, and produces the next EMA value in the sequence. This function takes three arguments, *period* is the value controlling the rate at which the EMA value

decreases. Second one is *current* which is the vote count for the attribute at present, and *prev_ema* is the previous EMA value for the attribute. These three variables correspond to N, Y_t, and S_{t-1} respectively in the EMA formula. *Period* is used to calculate α in the EMA formula. If a previous EMA value does not exist the EMA value will have the same value as the vote count for the current counter. This function is called for all three of the EMA values for each attribute in the alarm that received a click on their "+" or "-" button.

To be able to see how the operator's interest in different attributes changes over time, counters are only active for one day, as described in Section 4.3, after that new ones are created and the old ones will remain as inactive counters in the database. When a vote is cast on an event and there are no recent counters for the attributes of that event, new ones will be created. When that happens, the values in the old counters are used for calculating the next EMA values for the new counters.

4.3.5 Displaying the alarms

The EMA values calculated above are used to sort the events in three different ways, one for each EMA value that each attribute stores. This gives the user three different lists displaying alarms according to the perceived importance the operator herself has contributed to with her votes. The three sorting alternatives are called "Short", "Medium", and "Long". Short means that it is easily influenced by changes in the short time span. Medium sees changes over a medium time span and long over a long time span. Short uses the value EMA-5, Medium that of EMA-10, and Long EMA-20.

The actual sorting works as follows. The sorting function is presented with a list with the most recent alarms, in our implementation it is limited to the 100 most recent alarms. Each of the alarms has voting information stored for each of the four attributes that we are interested in. This voting information includes EMA-5, EMA-10, and EMA-20 values. In the case of the short sorting order the sorting function looks at the EMA-5 value for each of the four attributes in each alarm. These four EMA-5 values are then added together to form the score for that alarm. This score is then used by the sorting function to sort the alarms in the list. The sorted list will then contain all the alarms sorted according to their EMA-5 or short sorting order score.

If the operator chooses the Short sorting alternative, alarms high up in the list will recently have received votes and therefore have a very short term relevance to the operator. The Medium sorting alternative is a little more long term and the sorting order will not be affected that much by a lot of votes in the short term. Long is even more robust and should reflect what the operator thinks is important over a very long time span.

Chapter 5

Simulations

To evaluate our ranking methodology a series of experiments were performed. A Ruby program was developed for this purpose. As they would be a proof of concept it was decided to start out with a simple base case, which could easily be extended by demand. It consists of two parts, 1) an alarm generator, presented in Section 5.1 and 2) a click simulator, presented in Section 5.2. With configuration we can direct the simulation and let it emulate different types of networks and scenarios.

5.1 Alarm generation

The alarm generator generates a list of random alarms with different sets of attributes. To make it more realistic it uses a configuration which specifies ratios for how likely an attribute is among the generated alarms. Based on our expertise, a low security risk signature might for example be more common than a high security risk signature.

The configuration variables pertaining to alarm generation:

- `number of alarms`,
  the number of alarms to be generated during a test run.
- `alarm ratio high`,
  `alarm ratio medium`,
  `alarm ratio low`,
  decides the ratios between severities in the number of alarms generated. If `high = 15%`, then 15% of the number of generated alarms should have an alarm type classed as a high severity type. The total ratio has to add up to 100%.
- `number of destination IPs`,
  the number of destinations the generator can choose from when choosing a value for the destination IP attribute of the alarm.

- `number of source IPs`,
  the number of sources the generator can choose from when choosing a value for the source IP attribute of the alarm.
- `number of alarm types`,
  the number of alarm types the generator can choose from when choosing a value for the alarm type attribute of the alarm.
- `severity ratio high`,
  `severity ratio medium`,
  `severity ratio low`,
  the ratios of the alarm type severities. Snort labels its alarm types to have either a high, medium or a low severity. The ratios here decide how many of our total number of alarm types should be classed as severity high, medium and low. The total ratio has to add up to 100%.

As it is difficult to understand the difference between `alarm ratio` and `severity ratio` at first, we will now give an example to further explain their functions. Assume that the following variables have values:

- `number of alarms = 10`
- `alarm ratio high = 20%`
  `alarm ratio medium = 30%`
  `alarm ratio low = 50%`
- `number of alarm types = 8`
- `severity ratio high = 25%`
  `severity ratio medium = 50%`
  `severity ratio low = 25%`

`severity ratio x` decides how the different alarm types is labeled in terms of severity. In our example above, this would mean that out of the eight different alarm types, `1-8`, types `1` and `2` are classed as types of high severity, types `3-6` as types of medium severity and the remaining two types, `7` and `8`, as types of low severity. `alarm ratio x` decides how many of the generated alarms should be of what severity. In the example this results in that two out of the ten generated alarms should have an alarm type classed as being of high severity. Those two alarms can then be of either type `1` or `2`. Three out of the ten generated alarms should have a type classed as being of medium severity. Those three alarms can have either of the types `3-6`. Lastly, the remaining five generated alarms can have either of the types `7` or `8`.

5.2 Simulating clicks

The configuration also allows us to specify how many of the attributes should be considered important in the simulation. This is used later in the simulation, after the random list of alarms has been generated, to simulate which alarms the operator of the system clicks on. All alarms in the list of generated alarms that has one or more important attributes will receive a "click". Each click gives a point to all the attributes of that particular alarm. The points accumulated for

each individual attribute will later determine the score that each alarm receives. The score of an alarm is determined by simply adding up the points for each individual alarm as follows:

Attribute	Points
Source IP	2
Destination IP	10
Destination Port	8
Alarm type	7
Total score	27

Figure 5.1: An example showing how the score for an alarm is calculated.

This score is used to sort the alarms in order of importance. However this way of sorting does create some anomalies, which we will return to in the Results chapter (6.1). The anomalies happen when an alarm with less than four important attributes receives a click, because it will also reward any unimportant attributes of the alarm with a point. Unimportant attributes will by this process receive points simply by the fact that they are associated with an alarm containing important attributes.

- `source IP click worth`,
  `destination IP click worth`,
  `alarm severity high click worth`,
  `alarm severity medium click worth`,
  `alarm severity low click worth`,
  decides how many points each attribute gets added to its ranking score when an alarm is clicked.
- `important destinations`,
  how many of the destination IPs should be labeled as interesting to the user and be clicked when present in an alarm.
- `important sources`,
  how many of the source IPs should be labeled as interesting to the user and be clicked when present in an alarm.
- `important type high`,
  `important type medium`,
  `important type low`,
  how many of the alarm types with a certain severity should be labeled as interesting to the user and clicked when present in an alarm.

5.3 Details of configuration

The configuration behind the tests were set up in collaboration with Ola Söderström to as much as possible mimic a real life scenario.[1] The numbers repre-

[1] Ola Söderström(security analyst, Omegapoint, e-mail correspondence on October 28, 2010), see appendix D.2.

senting IPs and alarm types are chosen by us as a means to simulate how clicks would work and have nothing to do with real IP addresses or alarm types.

Basic configuration:

- `number of alarms = 100`,
  since 100 is a nice number to have when calculating percentages.
- `number of destination IPs = 50`,
  we have decided to emulate a smaller network.
- `number of source IPs = 100`,
  logically, the number of external IPs has to be larger than the number of internal. We decided on 100 which is more than 50 but still small enough to produce alarms with the same source.
- `number of alarm types = 50`,
  not at all reflecting the actual number real IDSs uses, but a number small enough to result in alarms having the same type but large enough to give something to choose from when generating alarms.
- `severity ratio high = 15%`,
  `severity ratio medium = 40%`,
  `severity ratio low = 45%`,
  15% of the alarms types should be classed as high severity alarms, 45% as medium severity and 40% as low severity. These numbers are coupled to the distribution of severity among the alarms.
- `alarm ratio high = 15%`,
  `alarm ratio medium = 40%`,
  `alarm ratio low = 45%`,
  15% of all generated alarms should have an alarm type of high severity, 40% should have an alarm type of medium severity and 45% should have one of low severity. These numbers are chosen based on the direct experience of Söderström.
- `source ip click worth = 1`,
  `destination ip click worth = 1`,
  `alarm severity high click worth = 3`,
  `alarm severity medium click worth = 1`,
  `alarm severity low click worth = 1`,
  when an alarm is clicked, 1 is added to the weight of the destination, source and alarm, with the exception of an alarm of high severity. In that case 3 is added to the weight. This is to counter the effect of the smaller ratio of high severity alarms. In a real system, one has to analyze the ratios between the different severities before configuring the weights. It is also possible to let the system continuously monitor the ratios and automatically update the weights.

The initial goal was to reach a clicking frequency of 10-15%, a number Söderström usually averaged. However, this was not possible without leaving an attribute unclicked resulting in the following configuration of how to simulate clicks:

- `important destinations = 4%`,
  4% of the possible destination IPs are selected as important. If a generated alarm has a destination IP selected as important a click will be simulated on that alarm and the weights associated with the destination IP, source IP and alarm type will all be incremented. This is the smallest percentage resulting in more than one IP being chosen important.

- `important sources = 2%`,
  2% of the possible source IPs are selected as important. If the source IP of an alarm is important the attributes are incremented. This is the smallest percentage resulting in the system choosing more than one source IP to be important.

- `important type high = 75%`,
  `important type medium = 10%`,
  `important type low = 5%`,
  75% of the alarm types chosen to be of high severity by `severity ratio` are selected as important and of interest. These will be clicked if they show up in a generated alarm. The same goes for 10% of the alarms of medium severity and 5% of the ones of low. 5% is the smallest percentage `important type low` can have and still result in one alarm of severity low being chosen as important. Feeling that alarms of medium severity should be more interesting than alarms of low severity we chose to mark 10% of the medium severity alarm as important. This results in more important alarms than those of low severity but still keeping the number low enough not to produce too many important alarms. An alarm of high severity is surmised to most often be clicked. Too low a percentage would seem silly and too high would generate too many clicked alarms. A compromise between the two settled the percentage on 75.

We are aware that the tests do not truly mirror real life. In reality one does not judge an alarm important based solely on source IP, destination IP or type of alarm but at the very least a combination of the three. The attribute source IP might as well be interchanged with the triple `(source IP, destination IP, alarm type)`. The reasons for clicking on an alarm can be complex, depending on a range of possibilities, a range we have no possibility to emulate. These tests are here as a proof of concept - that ranking is a viable means to help the user manage alarms. We have therefore chosen three simple attributes.

By the same reasoning, we have not simulated the effects of down prioritizing an alarm. As a proof of concept we wanted to keep the experiment simple and concentrate on only one factor. More importantly, we do not feel there is any need for the feature since the user would only utilize this if her interest changes over time or she does something wrong. As the tests are clicked by the program and no time elapse between the alarms, the conditions under which an alarm is down prioritized are not replicated.

Chapter 6

Results

Here we present the results from Chapters 4 and 5. Section 6.1 presents the performance of our simulation runs viewed from two angles; how well alarms carrying important attributes are separated from those who do not and how well the ranking order reflects the true severity of the alarms. Section 6.2 presents the changes done to Snorby.

6.1 Simulations

Using the configuration desribed in Chapter 5 a series of test runs were performed. To smooth the effects of diverging test runs an average of 10 test runs was used. The number of alarms with simulated clicks spans from 22 up to 28 averaging 26, making this fairly homogeneous.

Run	1	2	3	4	5	6	7	8	9	10	average
# clicked alarms	27	28	28	27	25	25	22	28	24	26	26

Table 6.1: Number of important alarms generated each test run.

The alarm generation ratios given to the click test simulation (destination IP, source IP, and alarm type), resulted in 6 IPs and 8 alarm types being chosen as attributes to be clicked. Table 6.2 summarizes the result.

Dest IPs	:	1 - 3
Src IPs	:	51 - 53
alarm high	:	1 - 5
alarm medium	:	8 - 9
alarm low	:	30

Table 6.2: The IPs and alarm types chosen as important. Whenever an alarm shows up carrying any of these attributes, it gets a simulated click.

Two words frequently recurring within this chapter is **important alarms/ clicked alarms** and **unimportant alarms/unclicked alarms**.

Definition 3. *An* IMPORTANT ALARM*, also referred to as a* CLICKED ALARM*, is an alarm with attributes of interest to the user. When the user spots this alarm, she is surmised to click on it, showing her interest.*

Definition 4. *An* UNIMPORTANT ALARM*, sometimes referred to as an* UNCLICKED ALARM*, is an alarm lacking attributes of interest to the user. When the user spots this alarm, she is surmised to leave it unclicked, showing her disinterest.*

To explain the result, two aspects are highlighted: the **density** with which the important alarms are presented and the **ordering**.

6.1.1 Density

To be able to understand what is meant by density it has to be defined first.

Definition 5. *The* DENSITY $D(i)$*, of ranked alarm position i, is the ratio of important alarms found at positions 1 up to i compared to the total number of alarms found at positions 1 up to i, where $i \leq$ position of last listed important alarm.*

$$D(i) = \frac{\#\ important\ alarms\ at\ positions\ 1\ to\ i}{\#\ alarms\ at\ positions\ 1\ to\ i}$$

The density after all important alarms have been listed remains the same, no matter how many unimportant alarms are listed beyond that. This is because density is a measure for how quickly the important alarms are listed. Once they are all listed, how the rest of the list looks like is unimportant.

Out of the 10 test runs only test run 1, 2, 5 and 10 yielded a density less than 100%, meaning that these four runs were the only runs not able to list the important alarms without interspersing them with unimportant alarms. Table 6.3 shows the density of a test run averaged from all ten test runs. Positions, i are shown in *italics* and density, `D(i)`, in `verbatim`.

On average, the alarm listed on the first position is an important alarm. In the first 10 positions, 10 important alarms have been listed. $D(10) = \frac{10}{10} = 100\%$. Since this is the average of 10 test runs it is obvious that in all cases, in the first 18 positions, all alarms are important. The first change shows up at position 19 where the density on average is 99.5% instead of 100%. This means that in one run out of the ten, an unimportant alarm is listed at index 19. More unimportant alarms are interspersed with importants alarms in the following positions. This can be deduced from from the decreasing density.

Figure 6.1 shows how well the ranking is able to sort alarms and separate important ones from unimportant. The x-axis represents the positions of the listed alarms. The y-axis represents the density of the important alarms in the first x positions. Test runs 3, 4, 6, 7, 8 and 9 all managed to achieve a density of 100% and are stacked on top of each other following $y = 100$. The curves achieving

i	D(i)	i	D(i)
1 :	100%	*18* :	100%
2 :	100%	*19* :	99.5%
3 :	100%	*20* :	99%
4 :	100%	*21* :	99%
5 :	100%	*22* :	99.1%
6 :	100%	*23* :	99.1%
7 :	100%	*24* :	98.8%
8 :	100%	*25* :	98.8%
9 :	100%	*26* :	98%
10 :	100%	*27* :	97.4%
11 :	100%	*28* :	97.5%
12 :	100%	*29* :	97.2%
13 :	100%	*30* :	95.6%
14 :	100%	*31* :	96.3
15 :	100%	*32* :	96.4%
16 :	100%	*...* :	...
17 :	100%	*100* :	96.4%

Table 6.3: Density of important alarms listed after i positions.

less than 100% are from bottom up: test run 1, 2, 10, 5 and the average. All of them starts to drop off somewhere in the interval between position 20 and 25. The bump in the curves, most clearly visible in run 2, means that important alarms was listed once again. This means that the density rises. Once the graph levels out, all important alarms of that test run (see Table 6.1) have been listed.

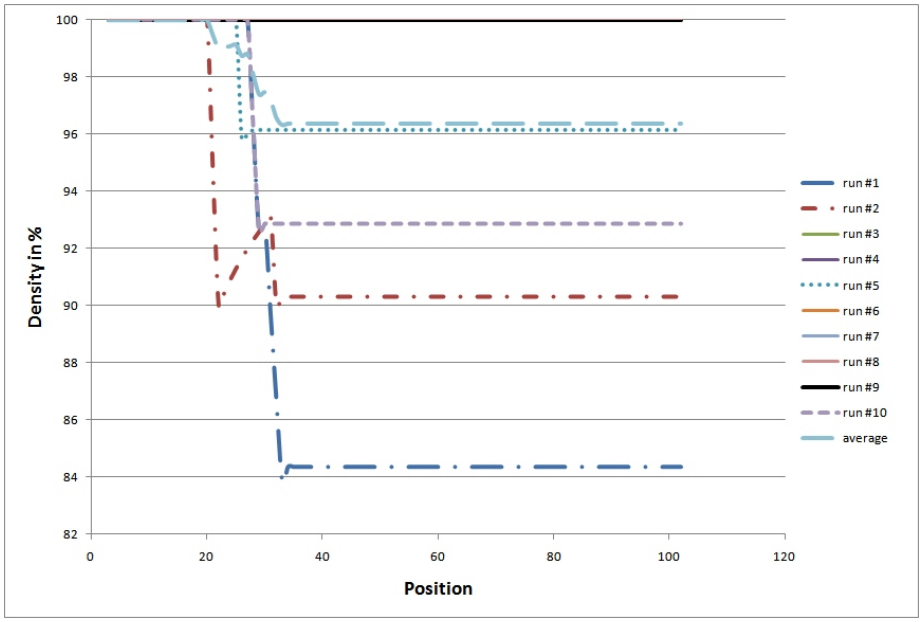


Figure 6.1: Density of all ten test runs and the average after ranking.

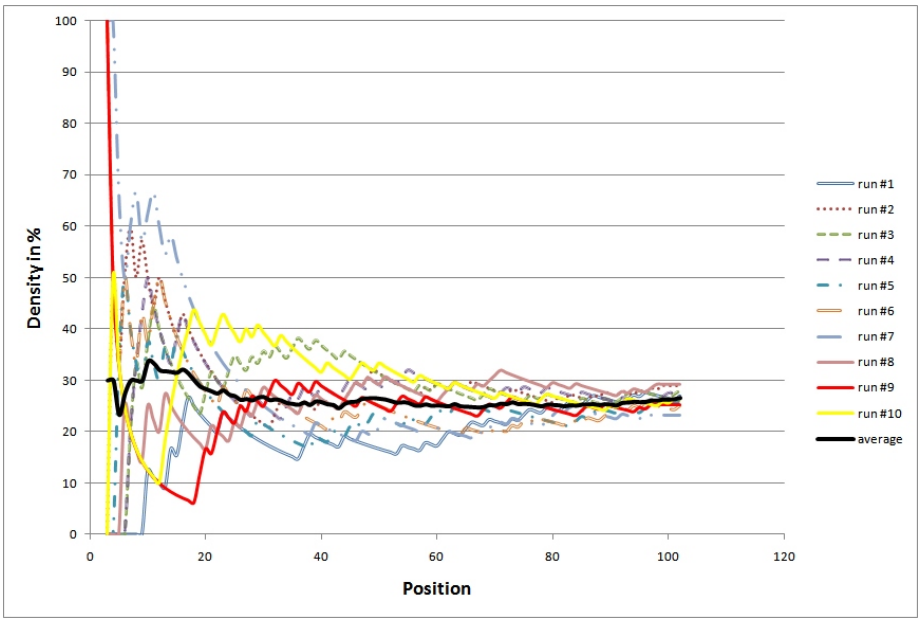


Figure 6.2: Density of all ten test runs and the average before ranking.

Figure 6.2 shows the densities of the alarms before ordering them after rank. The graph is quite messy but a definitive trend is discernible. As it is pure chance when an important alarm shows up, the curves covers a larger interval in the beginning, $0\% \leq y \leq 100\%$. But they level out as the position gets closer to $x = 100$ and y stabilizes in the interval 20% - 30%. This is expected since the average number of important alarms is 26 in a set of 100.

6.1.2 Ordering

The density, how well the important alarms are listed together without being interspersed with unimportant alarms, does not tell anything about the order of the alarms. The clicked alarms can be ordered in a way not reflecting the true importance of the alarms, putting the least important clicked alarm at the top and the most important clicked alarm right above the first unimportant alarm and still achieve 100% density. Since there is no way of averaging the placement of an alarm, figures of a few illustrative runs were chosen to indicate the ordering.

Figures 6.3, 6.4 and 6.5 show how well the ordering of test run 2, 4 and 7 matches the optimal ordering.[1] The x-axis, represents the positions in the list. The y-axis represents the order the alarms should have. The optimal dots show

[1]The optimal order is simulated by only awarding points to important attributes when an alarm receives a click. This results in a ranking score only regarding the attributes of interest to the user. As no unimportant attributes are awarded points, they cannot contribute to the rank as in our ranking system. The optimal order is, in other words, the way the user would have ordered them. Note that the optimal order only refers to the order of important alarms, it does not pertain the unimportant alarms.

Figure 6.3: Ordering of alarms of test run two.

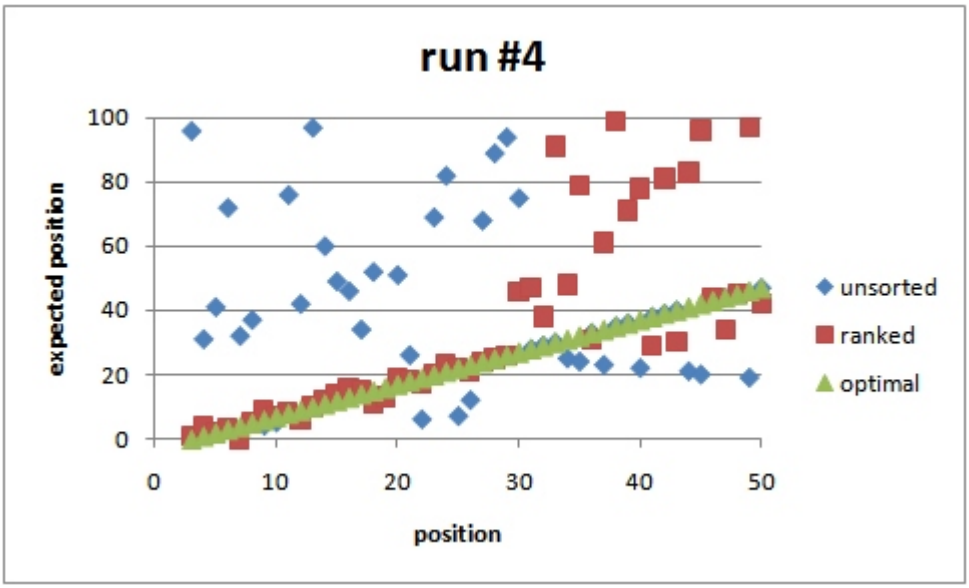


Figure 6.4: Ordering of alarms of test run four.

how the alarms really should be ordered. The first position of the list, $x = 0$, is where the most important alarm should be found, $y = 0$. In the second position the second most important alarm should be found, in other words, the optimal case matches $y = x$. The "unsorted" dots represent the unsorted alarms and in Figure 6.3 the first alarm to be listed ($x = 0$) is really the 64th most important ($y = 63$). This is quickly followed by an alarm that should have been at the end

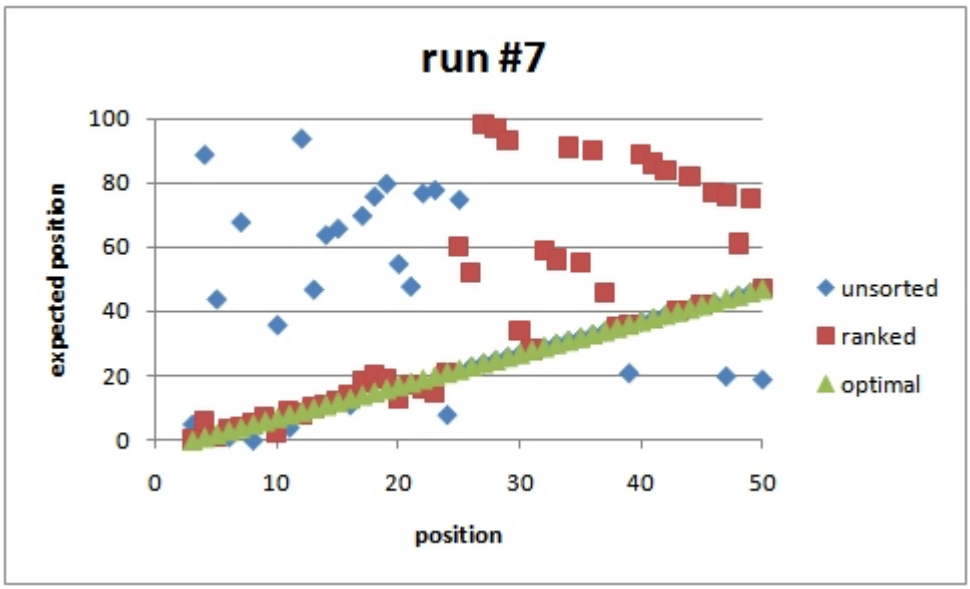


Figure 6.5: Ordering of alarms of test run seven.

of the list and then by one which should have been first. From this it is easy to see the closer the curve follows the optimal case the better, and the farther off the curve is the more the alarms are out of correct order.

The "ranked" dots representing the ranked case follow the optimal more closely than the unsorted case, which is not surprising as the ranking here has on average 26 alarms to rank and does so in at most 32 listings (see Figure 6.6). Although not fluctuating as wildly as the the dots representing the unsorted case, it still fluctuates meaning that the ranked alarms are not in the correct order. The alarms ranked in test run 4 and 7 have orderings closer to the correct one, than that of test run 2. After around 30 positions, the "ranked" dots lose their somewhat symmetry to the optimal ordering and start to look like the "unsorted" dots. This happens when all the important alarm have been listed and those left to list is of no importance, hence the ordering is of no interest. The graphs are cropped at $x = 50$ to display the behaviour after all the important alarms have been listed. The data points between $x = 51$ and $x = 100$ are left out since the same behaviour as in $35 \leq x \leq 50$.

Another way to illustrate the order is to focus on the rank given to an alarm for each of the ten test runs. In Figure 6.6, the intervals of the 12 most important alarms are tracked (read more about box plots in appendix A). The x-axis represents the position resulting from the ranking function, the y-axis represents the position the alarm really should have according to the reason behind the click. In Figure 6.6 the alarm which should have been first in the list, $y = 1$, is ranked anywhere between first and 12th with a concentration around the first four positions. The alarm which should have been ranked second most important, is most often placed as fifth.

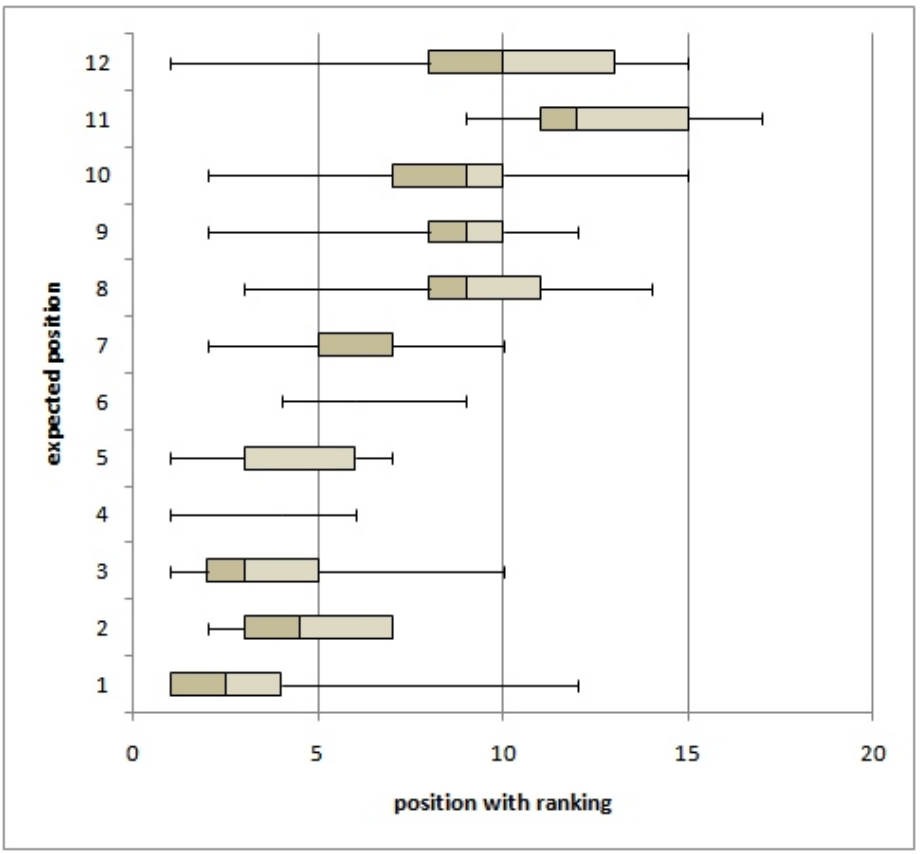


Figure 6.6: Intervals of the twelve most important alarms.

6.2 Snorby

The visual changes that have been made to Snorby are quite small and not obvious if the user is unfamiliar with the interface. The reason behind this is that our goal was to design an interface helping our ranking system but that at the same time does not interfere too much with the user's normal way of interacting with the interface. Figure 6.7 shows the event view in Snorby before the modifications, a list of events retrieved from the database where Snort stores captured packets. Each grey box represents an event with some basic information about the event, such as name of signature and source and destination IP. The order in which the events are displayed is the order in which they where captured. If the user clicks on one of these events, a page showing more detailed information about that alarm is displayed.

In the lower left corner of each grey box there is a small yellow box with three icons; a speech bubble, a star, and a red circle with a white cross. These three icons are buttons letting the user perform three different tasks; view comments, mark event as favourite, and delete event from database. In Figure 6.8 two new buttons are present, a "+" and a "-". These two buttons are what gives the

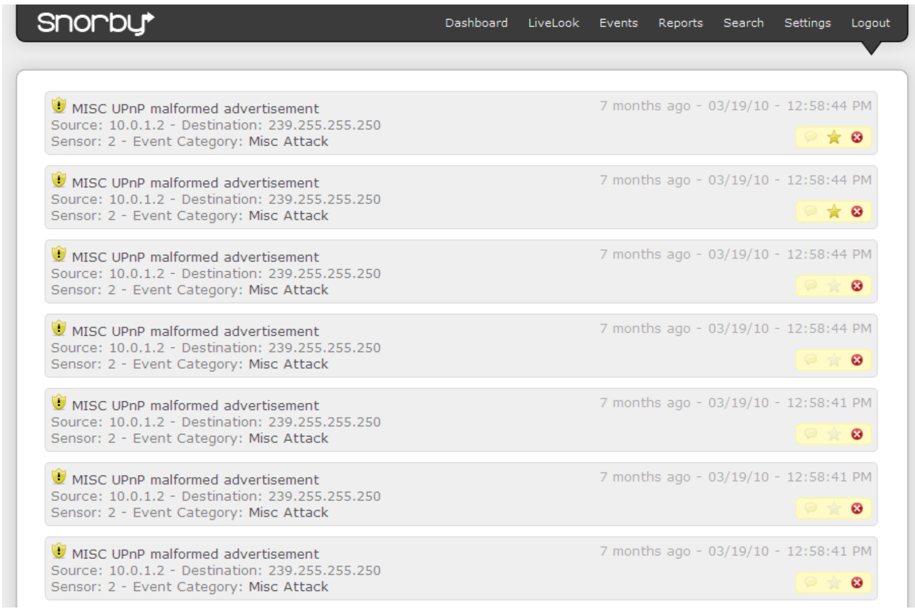


Figure 6.7: Unmodified version of Snorby displaying the event view.

user the capability to vote for which events she thinks are important. Clicking on "+" tells the ranking system that this alarm and its attributes are important to the user, a click on "-" has the opposite meaning.

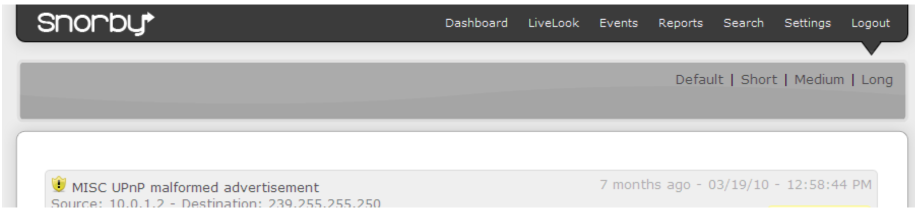


Figure 6.9: Three links that allow the user to choose how alarms should be sorted.

Figure 6.9 shows the menu that is added to the top right corner of the event view page. This menu was added to allow the user to choose which of the three EMA values that should be used when sorting the alarms. "Short", "Medium", and "Long" refer to how much influence votes cast a long time ago should have when sorting the alarms. "Short" uses a shorter time window, "Medium" uses a little larger window, and "Long" will take into account votes that where cast even longer ago. There is also a link called "Default" resetting

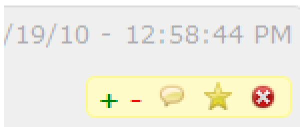


Figure 6.8: The buttons that where added to the Snorby interface.

the sorting order to the standard order that Snorby uses, in this case sorted by date in descending order.

Chapter 7

Discussion

7.1 Ranking system

The ranking function seems to be performing quite well. Its performance matches optimal density in 6 cases out of 10. In the other four cases the density drops due to unimportant alarms being listed before important ones. Why unimportant alarms get listed among the important ones goes back to how clicks are handled. As the system has no way of knowing why the user clicks on a specific alarm it can only add to or subtract from the current score of the attributes concerned by the alarm. A phenomena we call **collateral ranking** is thus created. Unimportant but common attributes get a high ranking score only due to association, as they are often present in alarms with other attributes viewed as important.

Definition 6. Collateral ranking *is the act of awarding unimportant attributes a higher rank due to their association with important attributes. That is, unimportant attributes get points by simply being present in the same alarms as important attributes.*

The phenomena of collateral ranking is even more clear when the ordering is studied. In Figure 6.6 it is obvious that collateral ranking affects ordering even more than the density. The bulk of the box plots hover in the main vicinity of the expected placement but the extremities go much farther off. In our small scale experiment of 100 alarms the effect is not overly drastic but if this were to be scaled up to 1000 alarms perserving the trends evident in our experiments, the interval of the most important alarm would grow proportionally from $[1, 12]$ to $[1, 120]$. Figures 6.3, 6.4 and 6.5 show the variation one can be expected to deal with. Depending on the alarms, their distribution and the clicks of the operator(s) can end up with a ranking following the expected ordering fairly well, or one can get a ranking having little in common with the expected ordering.

How well the phenomena of collateral ranking translates into a real scenario we do not know and see the need to make studies in a real environment before

definitive conclusions can be drawn. The alarms in our tests are randomly generated without internal connections which is not the situation in the real world. The whole practice of the methods mentioned in Chapter 3.2 is based on the assumption that alarms are not random. We do not know if the results obtained are in any way skewed by the data and tests on real data have to be performed to see if they support our findings.

7.1.1 Issues outside of the system

The model tracks three attributes in the simulations and four in the implementation in Snorby which is a minuscule subset of all the attributes possible to track. The question is whether the trends found here manifest themselves in a more complex model, tracking more attributes. Will the ranking become more accurate in presenting the expected order or will the collateral ranking be amplified and wreak even more havoc? As Ola Söderström described, alarms are not clicked because of a single attribute, instead it is often a combination of attributes which make an alarm interesting.[1] These kinds of conditional clicks is not something our model respects. To explicitly tell the system under what conditions attributes are important, it might be a good idea to apply a data mining technique to the clicked alarms to find relationships between them. The discovered relationships can then be displayed for the user and she can either agree or disagree with the estimated relationship.

In our ranking system, it is absolutely vital that the operator knows what she is doing. Voting for unimportant alarms that at first seem important or voting against important alarms that at first seem unimportant might lead to the wrong alarms to be highlighted and the alarms needing the most attention being hidden somewhere at the bottom of the list. Therefore, overconfidence in the ranking is dangerous no matter how good an analyst one views oneself to be, since there is no way to check if the system ranks alarms correctly.

A precise level of exactness will probably never be attained with a ranking system producing estimates based solely on user input. A remedy could be to have configuration files containing information otherwise difficult to convey to the system. Difficult information might be conditional clicks, e.g. alarms of the following type: `C` is only interesting if `destination IP = 3`, `port = 79` and `protocol = UDP`. The problem with configuration files is maintenance. They have to be kept up to date to be of any use to the system. Configuration can be anything from minimal, only keeping a few choice decisions on how to rank, or it can be huge (see Porras et al. [30]) containing every detail. Less configuration and more on-the-fly results in a system easy to start using but less exact. More configuration and less on-the-fly results in a system more exact in its ranking but much more difficult and time consuming to launch and maintain. There will always be a trade-off.

[1] Ola Söderström (security analyst, Omegapoint, interviewed in person by the authors April 23, 2010).

Chapter 8

Future Work and Conclusions

8.1 Future Work

The most obvious and given task in future work is to test our model using more realistic data and see how it performs.

Our ranking system has a number of weaknesses that are in part inherent to how it fundamentally works, but it also has a few weaknesses which probably could be mended with some additional features. Even if an alarm signature is completely uninteresting and is categorically down voted, it still might get a lot of points just by association - collateral ranking. This could be mended with an administrative console which allows the operator to manually lock a certain attribute at a lower rank. The opposite is also attractive, allowing the user to raise the rank of an alarm drastically if the need presents itself.

Collateral ranking is also the perpetrator of how the expected order of alarms is reflected in the ranking. Can this be coped with by adding more attributes to follow?

Point adjustments could also be made depending on which version of a signature generated the alarm. Snort has a column called *sig_ver* in the *signature* table, which saves the version number of the signature that generated the alarm. Adjusting the points awarded depending on the version might be of interest in certain scenarios. A signature is maybe discovered to be causing a lot of false alarms and patched as a result. The updated signature will receive a new version number. A penalty can then be attached to the old version of the signature to lower the rank of all the alarms which where captured with the old version. How useful this would be also depends on how new signatures are deployed. If they are first tested on one Snort sensor while the others continue using the old one, then it might be of use. But if all Snort sensors start using the updated version immediately alarms generated by the old defunct signature will become obsolete quickly and giving them a lower score might not have any effect at all

on the priority of the alarms.

A more advanced addition to the ranking system would be to look at votes in a wider context, as chain of votes. The user of the system might place votes on alarms similar to each other in some regard. These similarities would be of interest to the ranking system as they can give more information about why the user is prioritising one alarm with signature *A* but not another with the exact same signature. The most recent alarms receiving votes could then be analysed to find what common traits they have. The user might only prioritize alarms with signature *A* if the destination IP is *10.0.0.1* and port is *80*. This is something our ranking system at present cannot handle.

8.2 Conclusions

We have studied different techniques to help the user of an IDS manage alarms. Among the techniques for processing alarms, we find aggregation, correlation, filtering and ranking. Out of these, ranking was chosen for an in-depth analysis due to its many possibilities of being tweaked through normal interaction with the user. Based on our analysis, we have implemented a ranking system in the architecture Snort/Snorby that the user is able to interact with. Our system tracks four attributes of each alarm: source and destination IP, destination port and alarm signature, and keeps a ranking score for each. The alarms are ranked based on the total score of these four attributes.

The ranking system seems to be performing quite well based on our generated test data. The alarms carrying important attributes are all ranked at the top of the list presented to the user. This gives the user a much smaller interval to scan for important alarms and lets her detect alarms of importance faster. There are no guarantees that the alarms will be sorted according to actual importance, though. The ordering is dependent on how often different attributes occur in alarms and the alarms of highest importance might not occur as often as others and therefore end up lower down among the important alarms. This behaviour is credited to the phenomena we call collateral ranking (defined on page 67).

This is not a system for operators in training. No guidance is given to the user about alarms and the user needs to be knowledgeable in this area to be able to properly use the ranking system. Prioritizing alarms not having any real importance and down prioritizing crucial alarms will severely skew the final list and might lead to important alarms being missed as they are located at the bottom of the list.

The data used in the simulations is randomly generated and does not originate from any real world context. The results therefore have a certain amount of uncertainty about them. Before any work is done expanding this model, it would be beneficial to test it in a live network.

Still, our model is based on user input in simple terms and might not catch the more complex reasons why the user decides an alarm is important or unimportant. The phenomena of collateral ranking also disturbs the result. We propose the use of configuration files but we have not formally tested this. The prob-

lem with configuration files, though, is the generally time consuming work of maintaining them and keeping them up to date. This will always be a trade-off, with less configuration and more on-the-fly operation the system will be readily maintained but with a certain amount of uncertainty about it. With more configuration and less on-the-fly operation the system will be more exact in its ranking but harder to maintain.

Overall, our ranking system ranks alarms with minimal configuration and input from the user and shows promising results. We believe it will be a great asset for intrusion detection operators in their daily work.

Bibliography

[1] Raven Alder, Jacob Babbin, Adam Doxtater, James C. Foster, Toby Kohlenberg, and Michael Rash. *Snort 2.1 Intrusion Detection, second edition*. Syngress Publishing Inc., 800 Hingham Street, Rockland, MA 02370, 2004.

[2] Magnus Almgren and Erland Jonsson. Tuning an IDS - learning the security officer's preferences. In *11th Nordic Workshop on Secure IT Systems - NordSec06*, pages 43–52, Linköping, Sweden, 2006.

[3] Stefan Axelsson. The base-rate fallacy and the difficulty of intrusions detection. *ACM Trans. Inf. Syst. Secur.*, 3(3):186–205, 2000.

[4] Cisco. Cisco MARS. `http://www.cisco.com/en/US/products/ps6241/index.html`.

[5] Chris Clifton and Gary Gengo. Developing custom intrusion detection filters using data mining. In *Military Communications International Symposium (MILCOM2000)*, pages 440–443, 2000.

[6] Frédéric Cuppens and Alexandre Miège. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 202–, Oakland, California, USA, 2002. IEEE Computer Society.

[7] Oliver Dain and Robert Cunningham. Building scenarios from a heterogeneous alert stream. In *IEEE Workshop on Information Assurance and Security*, West Point, NY, USA, 2001.

[8] Oliver Dain and Robert Cunningham. Fusing a heterogeneous alert stream into scenarios. In *Proceedings of the 2001 ACM Workshop on Data Mining for Security Applications*, pages 1–13, 2001.

[9] Hervé Debar and Andreas Wespi. Aggregation and correlation of intrusion-detection alerts. In *RAID '00: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, pages 85–103, Toulouse, France, 2001. Springer-Verlag.

[10] D.E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, SE-13(2):222–232, 1987.

[11] S. Forrest, S.A. Hofmeyr, A. Somayaji, and Longstaff T.A. A sense of self for UNIX processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120–128, 1996.

[12] Jeremy Frank. Artificial intelligence and intrusion detection: Current and future directions. In *Proceedings of the 17th National Computer Security Conference*, 1994.

[13] P. Helman and G. Liepins. Statistical foundations of audit trail analysis for the detection of computer misuse. *IEEE Transactions on Software Engineering*, 19:886–901, 1993.

[14] Harold S. Javitz and Alfonso Valdes. The SRI IDES statistical anomaly detector. In *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pages 316–326. IEEE Computer Society, 1991.

[15] Kevin Johnson, Axton Graham, Jon Hart, Kevin Johnson, Doug Mackie, Sean Muller, Tim Rupp, Christian Svensson, and Max Valdez. Base. `http://base.secureideas.net`.

[16] Klaus Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Trans. Inf. Syst. Secur.*, 6(4):443–471, 2003.

[17] Brendan P. Kehoe. *Zen and the art of the internet*. Prentice Hall, 1995.

[18] C. Ko. Logic induction of valid behavior specifications for intrusion detection. In *Proceedings of the 2000 IEEE Symposium of Security and Privacy*, pages 142–153, 2000.

[19] M. Ko, C. Ruschitzka and K. Levitt. Execution monitoring of security-critical programs in distributed systems: a specification-based approach. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 175–187. IEEE Computer Society, 1997.

[20] Terran Lane and Carla E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Trans. Inf. Syst. Secur.*, 2(3):295–331, 1999.

[21] Wenke Lee and Salvatore J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Trans. Inf. Syst. Secur.*, 3(4):227–261, 2000.

[22] Wenke Lee and Dong Xiang. Information-theoretic measures for anomaly detection. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 130–143, Oakland, California, USA, 2001.

[23] Ulf Lindqvist and Phillip A. Porras. Detecting computer and network misuse through the production-based expert system toolset (P-BEST). In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 146–161. IEEE Computer Society, 1999.

[24] Alex Lukatsky. *Protect your information with intrusion detection*. A-List Publishing, A-LIST, LLC, 295 East Swedesford Rd., PMB #285, Wayne, PA 19087, 2003.

[25] Emilie Lundin Barse. *Logging for Intrusion and Fraud Detection*. PhD thesis, Chalmers University of Technology, Göteborg, Sweden, 2004.

[26] Stefanos Manganaris, Marvin Christensen, Dan Zerkle, and Keith Hermiz. A data mining analysis of RTID alarms. *Comput. Netw.*, 34(4):571–577, 2000.

[27] Jason Meller. Snorby. `http://snorby.org/`.

[28] Peng Ning, Yun Cu, and Douglas S. Reeves. Analyzing intensive intrusion alerts via correlation. In *RAID '02: Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection*, pages 74–94, Zurich, Switzerland, 2002. Springer-Verlag.

[29] Tadeusz Pietraszek. Using adaptive alert classification to reduce false positives in intrusion detection. In *RAID '04: Proceedings of the 7th International Symposium of Recent Advances in Intrusion Detection*, pages 102–124, Sophia Antipolis, France, 2004. Springer Berlin / Heidelberg.

[30] Phillip A. Porras, Martin W. Fong, and Alfonso Valdes. A mission-impact-based approach to INFOSEC alarm correlation. In *RAID '02: Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection*, pages 95–114, Zurich, Switzerland, 2002. Springer-Verlag.

[31] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of USENIX LISA'99*, Seattle, Washington, USA, 1999.

[32] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni. A fast automaton-based method for detecting anomalous program behaviors. In *Proceedings of the 2001 IEEE Symposium of Security and Privacy*, pages 144–155, Oakland, California, USA, 2001.

[33] Dave Smith. *Disney A to Z.* Disney Book Publishing Inc., 2006.

[34] Sourcefire. Snort. http://www.sourcefire.com/security-technologies/snort.

[35] William Stallings. *Network Security Essentials: Applications and Standards.* Pearson Education, Upper Saddle River, NJ 07458, 2007.

[36] Aurobindo Sundaram. An introduction to intrusion detection. *Crossroads*, 2(4):3–7, 1996.

[37] D. Wagner and R. Dean. Intrusion detection via static analysis. In *Proceedings of the 2001 IEEE Symposium of Security and Privacy*, pages 156–168, Oakland, California, USA, 2001.

[38] Dustin Willis Webber. Snorby - a web interface for Snort. `https://github.org/Snorby/snorby`.

[39] Jingmin Zhou, Mark Heckman, Brennen Reynolds, Adam Carlson, and Matt Bishop. Modeling network intrusion detection alerts for correlation. *ACM Trans. Inf. Syst. Secur.*, 10(1):4, 2007.

Appendix A

Box Plots

A box plot is dependent on five values:

1. x_{min}, the minimum value among the data points.
2. x_{lower}, the lower quartile.
3. $x_{0.5}$, the median.
4. x_{upper}, the upper quartile.
5. x_{max}, the maximum value among the data points.

$x_{0.5}$ = the most central value of the data points ordered in ascending order. If n is even $x_{0.5}$ is the average of the two most central data points.

Let the median split the data points in two halves. If n is odd, the median is accounted to both halves and if n is even, none of the halves.

x_{lower} is the median of the lower half and x_{upper} is the median of the upper half.

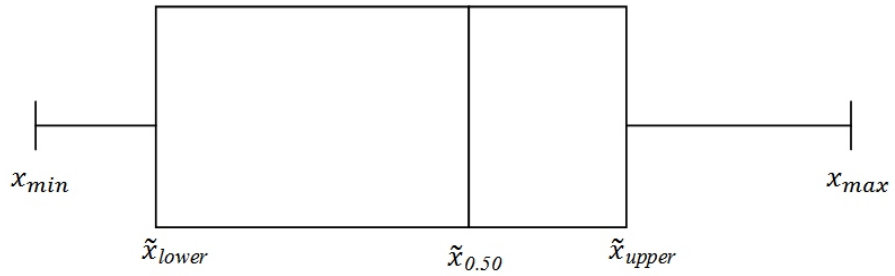


Figure A.1: A generic box plot.

Appendix B

Specification of Requirements

Give positive feedback to an alarm
Actor: System administrator
Goal: The vote should be evenly divided between all or a subset of the attributes of the alarm and saved to a database.
Interaction: The user clicks on a button that gives this alarm and alarms similar to this one a **higher** ranking.
Difficulty: Easy

Give negative feedback to an alarm
Actor: System administrator
Goal: The vote should be evenly divided between all or a subset of the attributes of the alarm and saved to a database.
Interaction: The user clicks on a button that gives this alarm and alarms similar to this one a **lower** ranking.
Difficulty: Easy

Changing sort order
Actor: System administrator
Goal: Sorting order should change.
Interaction: It should be possible for the user to change the sort order of alarms between three different orders; short, medium, and long. Short will sort the alarms in a way that reflects their short term changes in the ranking. Medium should reflect medium term changes in the ranking of alarms, and long is for long term changes.
Difficulty: Hard

Appendix C

Source Code

C.1 click_test

click_test is a small program we developed to test how effective it would be to sort alarms using votes from the user. It does not directly mimic how our implementation in Snorby works in that it does not simulate any kind of negative feedback.

```
#!/usr/bin/ruby

### CONFIGURATION ###

# Number of alarms to generate
$num_alarms = 100

# Destination IPs, how many in total and what percent of them
    that will receive
# clicks
$num_dest_ip = 50
$important_dest_ip = 4 # in percent

# Source IPs, same deal as with destination IPs
$num_src_ip = 100
$important_src_ip = 2 # in percent

# Alarm types.
# How many alarm types should there be?
$num_alarm_types = 50
# How should they be divided between the three severities high
    , medium and low?
$severity_ratios = { :high => 15,      # in percent
                     :medium => 45,
                     :low => 40 }

# What percentages of the different alarm severities should
    receive clicks?
```

```
26 $important_alarm_types = { :high => 75,
27                            :medium => 10,
28                            :low => 5 }
29
30 # Alarm generation ratios in percent
31 $alarm_ratios = { :high => 15,
32                   :medium => 40,
33                   :low => 45 }
34
35 # How much is a click worth?
36 $src_ip_click      = 1
37 $dest_ip_click     = 1
38 $alarm_type_click = { :high => 3,
39                       :medium => 1,
40                       :low => 1 }
41
42 ### END OF CONFIGURATION ###
43
44 # IP ranges
45 $dest_ips     = (1..$num_dest_ip).to_a
46 $imp_dest_ips = $dest_ips[0..($num_dest_ip*($important_dest_ip
       /100.0)).round]
47 $src_ips      = ($dest_ips.last+1..$dest_ips.last+1+
       $num_src_ip).to_a
48 $imp_src_ips  = $src_ips[0..($num_src_ip*($important_src_ip/
       100.0)).round]
49
50 # Alarm type ranges
51 $alarm_types = {}
52 prev_v = 0
53 $severity_ratios.each_pair do |k,v|
54   new_v = prev_v + ($num_alarm_types*(v/100.0)).floor
55   $alarm_types[k] = (prev_v+1..new_v).to_a
56   prev_v = new_v
57 end
58
59 # Important alarm types
60 $imp_alarm_types = {}
61 $important_alarm_types.each_pair do |k,v|
62   types = $alarm_types[k]
63   $imp_alarm_types[k] = types[0..(types.size*(v/100.0)).round
       -1]
64 end
65
66 # Print what was generated
67 puts "Dest IPs: #{$dest_ips.first}-#{$dest_ips.last}"
68 puts "Important dest. IPs: #{$imp_dest_ips.first}-#{
       $imp_dest_ips.last}"
69 puts "\nSource IPs: #{$src_ips.first}-#{$src_ips.last}"
70 puts "Important src. IPs: #{$imp_src_ips.first}-#{$imp_src_ips
       .last}"
71 puts "\nAlarm types:"
72 $alarm_types.each_pair do |s,v|
73   puts "\t#{s.to_s}: #{v.first}-#{v.last}"
```

```
74  end
75  puts "\nImportant alarm types"
76  $imp_alarm_types.each_pair do |s,v|
77    puts "\t#{s.to_s}: #{v.first}-#{v.last}"
78  end
79
80  # two hashes that will hold the weights for different ips and
        types of alarms
81  $sip_w = {}
82  $dip_w = {}
83  $alarm_type_w = {}
84
85  #hashes that will hold the real weights
86  $real_sip_w = {}
87  $real_dip_w  = {}
88  $real_alarm_type_w = {}
89
90  # will hold all the alarms that are generated
91  $alarms = []
92  # keeps track of how many alarms in total
93  $tot_alarms = {:high => 0, :medium => 0, :low => 0}
94
95  class Alarm
96
97    attr_accessor :severity, :src_ip, :dest_ip, :type, :
          unordered, :ranked, :optimal
98
99    def initialize(i)
100     @id = i
101     # random number that will decide which alarm to generate
102     random = rand(100)
103
104     alarm_ratios = $alarm_ratios.sort { |a,b| a[1] <=> b[1] }
105
106     cum_r = 0
107     alarm_ratios.each do |i|
108       s = i[0] # severity
109       r = i[1] # ratio
110
111       if random < cum_r+r then
112         $tot_alarms[s] += 1
113         # generate alarm of severity s
114         @severity  = s
115         @src_ip    = $src_ips[rand($src_ips.size-1)]
116         @dest_ip    = $dest_ips[rand($dest_ips.size-1)]
117         types      = $alarm_types[s]
118         @type      = types[rand(types.size-1)]
119         @unordered = 0
120         @ranked    = 0
121         @optimal   = 0
122
123         # initialize values in sip_w, dip_w and alarm_type_w
124         $sip_w[@src_ip]       = 0
125         $dip_w[@dest_ip]      = 0
```

```
126             $alarm_type_w[@type] = 0
127       $real_sip_w[@src_ip]   = 0
128       $real_dip_w[@dest_ip] = 0
129       $real_alarm_type_w[@type] = 0
130             break
131           end
132           cum_r += r
133         end
134       end
135
136       def add_click
137         $sip_w[@src_ip]       += $src_ip_click
138         $dip_w[@dest_ip]      += $dest_ip_click
139         $alarm_type_w[@type] += $alarm_type_click[@severity]
140
141         if $imp_src_ips.include?(@src_ip)
142           $real_sip_w[@src_ip] += $src_ip_click
143         end
144         if $imp_dest_ips.include?(@dest_ip)
145           $real_dip_w[@dest_ip] += $dest_ip_click
146         end
147         if $imp_alarm_types.values.flatten.include?(@type)
148           $real_alarm_type_w[@type] += $alarm_type_click[@severity
                ]
149         end
150       end
151
152       def importance
153         $sip_w[@src_ip] + $dip_w[@dest_ip] + $alarm_type_w[@type]
154       end
155
156       def real_importance
157         $real_sip_w[@src_ip] + $real_dip_w[@dest_ip] +
              $real_alarm_type_w[@type]
158       end
159
160       def important?
161         $imp_src_ips.include?(@src_ip) ||
162           $imp_dest_ips.include?(@dest_ip) ||
163           $imp_alarm_types.values.flatten.include?(@type)
164       end
165
166       def set_optimal(index)
167         @optimal = index
168       end
169     end
170
171     # generate some random events
172     1.upto($num_alarms) { |i| $alarms << Alarm.new(i) }
173
174     #keep a counter for how many important alarms, used for
          statistics later.
175     no_imp_alarms = 0
176
```

```
177 # simulate clicks for important events
178 $alarms.each_index do |i|
179   $alarms[i].unordered = i
180   if $alarms[i].important?
181     $alarms[i].add_click
182     no_imp_alarms += 1
183   end
184 end
185
186 # make a copy to preserve original order
187 $alarms_un = Array.new($alarms)
188
189 $alarms.sort! { |x,y| y.real_importance <=> x.real_importance
        }
190 # record the order of the optimally ranked alarms
191 $alarms.each_index do |i|
192   $alarms[i].set_optimal(i)
193 end
194
195 # sort events by importance
196 $alarms.sort! { |x,y| y.importance <=> x.importance }
197
198 # record the order of ranked alarms
199 $alarms.each_index do |i|
200   $alarms[i].ranked = i
201 end
202
203 count_imp_alarms = 0
204
205 puts "\nNumber of generated alarms"
206 puts "\tHigh   : #{$tot_alarms[:high]}"
207 puts "\tMedium : #{$tot_alarms[:medium]}"
208 puts "\tLow    : #{$tot_alarms[:low]}"
209 puts "\tTotal  : #{$alarms.size}"
210
211 puts "\nNumber of clicked alarms : #{no_imp_alarms}"
212
213 puts "\nAlarms sorted by importance"
214 puts "\t Legend: <Alarm type>, src ip-->dest ip"
215 $alarms.each do |a|
216   print "\n"
217   print "#{a.importance} | ".rjust(7)
218   print "#{a.real_importance} | ".rjust(7)
219   print "<#{a.type}>".rjust(6)
220   print " (#{a.severity})".ljust(11)
221   print "| #{a.src_ip}-->#{a.dest_ip}".ljust(13)
222   print " | "
223   if a.important?
224     print "important |"
225     print "  #{(count_imp_alarms += 1)} |".rjust(6)
226   else
227     print "|".rjust(11)
228     print "  #{(count_imp_alarms)} |".rjust(6)
229   end
```

```
230   print "#{a.unordered} |".rjust(6)
231   print "#{a.ranked} |".rjust(6)
232   print "#{a.optimal}".rjust(4)
233 end
234 puts "\n"
235
236 puts "\t Legend: <Alarm type>, src ip-->dest ip"
237 $alarms_un.each do |a|
238   print "\n"
239   print "#{a.importance} | ".rjust(7)
240   print "#{a.real_importance} | ".rjust(7)
241   print "<#{a.type}>".rjust(6)
242   print " (#{a.severity})".ljust(11)
243   print "| #{a.src_ip}-->#{a.dest_ip}".ljust(13)
244   print " | "
245   if a.important?
246     print "important |"
247     print "  #{(count_imp_alarms += 1)}"
248   else
249     print "|".rjust(11)
250     print "  #{(count_imp_alarms)}"
251   end
252 end
253 puts "\n"
```

Appendix D

Interviews – Q&A

D.1 Håkan Nohre

Interview with Håkan Nohre from Cisco via e-mail in Swedish.

Bold = us

Normal = Håkan Nohre

I hur stor utsträckning kan man aggregera alarmen i er IDS? (AGGREGERA = t.ex. visa alla alarm som inbegriper samma källa som ett alarm så att man lätt kan se om någon verkligen attackerar nätverket, alltså sorterar alarmen på något gemensamt attribut som källa, destination, port, protokoll, etc.)

Kan ske på olika nivåer:

- I själva IPS så kan alarm aggregeras innan de skickas, så att t.ex. flera event av samma typ mellan samma sort/dest skickas som ett alarm istället för flera (minskar bruset).
- I managementverktyget IME kan man lätt sortera attacker efter t.ex. per destination, per source, per attacktyp, per sensor etc. Dvs man kan få en omedelbar vy av alla attacker från ip adress X, fördelat på target T1,T2,T3...TN, och under dessa attacktyper (A1, A2, A3) per X och Ti.
- I management verktyget Cisco MARS aggregeras inte bara IPS alarm, utan även alarm från brandväggar, operativsystem för att aggregeras till incidenter. En incident kan bestå av hundratals events, där kanske ett par kommer från IPS, övriga från brandväggsloggar eller serversystem.

I hur stor utsträckning kan man påverka hur alarmen aggregeras?

Helt konfigurerbart i alla nivåer.

I hur stor utsträckning kan ert system korrelera alarm/föra upp dem på en högre nivå?
(KORRELERA = t.ex. kan man bygga scenarion där en hel attack-kedja behövs för att uppnå hackarens mål. Att kunna se att ett alarm eller flera kan kopplas till ett scenario och varna för scenariot istället för att bara visa alarmen var för sig är ett exempel på korrelation.)

I management verktyget Cisco MARS aggregeras inte bara IPS alarm, utan även alarm från brandväggar, operativsystem för att aggregeras till incidenter.

En incident kan bestå av hundratals events, där kanske ett par kommer från IPS, övriga från brandväggsloggar eller serversystem.

Vilka fördelar har Ciscos system över konkurrenterna när det gäller att korrelera/aggregera alarm?

En fördel med Cisco MARS är att den kan aggregera alarm från flera olika källor, inte bara IPS, utan brandväggar, serversystem. En IPS kan sägas ge bra focus med mycket detaljer om en attack, men ofta kan man tappa den stora bilden eftersom IPS bara lyssnar/ser trafiken på ett par ställen i nätverket.

Med MARS kan man alltså utnyttja information inte bara från IPS för att få se en fullständigare bild. T.ex om IPS visar att A försöker hacka B : Vilka övriga maskiner har A talat med sista 24 timmarna? Denna trafik kanske inte ens har gått genom IPS, så man vill alltså utnyttja brandväggsloggar eller netflow här.

Kan er IDS lära sig hur viktiga vissa typer av alarm är för en administratör genom dennes interaktion med systemet? T.ex. genom att registrera vilka alarm som klickas på.

- **Är det något ni tillämpar i ert system?**
- **Om ja, hur samlar ert system in denna information?**
- **Ser du några fördelar/nackdelar med detta interaktionssätt?**

I vårt system bestäms viktigheten i ett alarm av ett metavärde, “Risk Rating”.

Risk Rating är en funktion av:

- event severity (hur allvarlig attack)
- event fidelity (hur säker är man att det ej är false positive)
- attack relevancy (om attack är relevant för OS som attackeras, där detta fastställs med passive OS fingerprinting
- target value (administratörsspecifierad vädering av target maskin)
- reputation (källans rykte på internet, tas från Cisco Sensorbase)

Ju högre RiskRating desto allvarligare är attacken. Vilka actions en IPS skall ta bestäms av Risk Rating, t.ex. droppa paket om Risk Rating > 80.

Ser ej så stora fördelar med att IPS anpassar sig efter användarens klickande, detta kan ju variera med tiden.

Hur ser ni på framtiden? Kommer IDS-system bli bättre på att upptäcka nya attacker själva eller kommer det förbli en "katt och råtta"-lek där tillverkaren av IDS-systemet alltid ligger steget efter dom som hittar på nya attacker?

Ett IPS system med signaturer som är sant vulnerability baserade (motsats till exploit baserat) bör hitta även nya attacker om det är en känd sårbarhet.

Cisco IPS har även funktion för ryktesbaserad filtrering, en stor andel av även nya attacker sker idag från komprometterade hostar, s.k botnets. Dessa attacker kan blockeras även om de är helt nya eftersom blockering sker med hjälp av rykteshanteringen.

Vilka problem har era kunder som de vill att ni skall lösa när det gäller intrusion detection och intrusion prevention?

Ofta någon form av Compliance, t.ex. PCI compliance.

Vilken ny funktionalitet i ert IDS-system är det era kunder efterfrågar mest?

Just nu pågår mycket uppgraderingar av datacenter till 10 gig ut mot servrarna, vilket ger mycket hög total throughput i datacenter. Så prestandakraven växer hela tiden.

D.2 Ola Söderström

Transcript of interview we did with Ola Söderström from Secode on 23 April 2010. The interview was done in person and recorded on an mp3-player.

Vad har du gjort sen du slutade på Chalmers?

–När jag slutade på Chalmers så hade jag inget jobb direkt. Så jag hjälpte lite mindre företag med deras IT-miljöer i största allmänhet. Oftast windows miljö och vardagliga problem som dom hade. Och det var väl ganska lärorikt. Detta var hösten 2003.

–Sen gick det drygt ett år och så fick jag jobb på Secode som jag har jobbat hos sen 2005. Där har jag haft ungefär samma arbetsuppgifter, har väl utvecklats lite kanske men det har blivit mycket sitta och klicka på alarm. Där är jag nu men sen nyår är jag pappaledig.

–Vissa saker har man i färskt minne när man sitter och håller på med det. Men som jag nu känner att vad fort man kommer i från det tänket och arbetsgången. Får man tänka efter lite gran så kommer det fram så smått.

Vilka interface har du använt?

–Cisco Works, Juniper, Tipping Point, ISS (iss.net) och Snort

---

Ifall du tänker på dom olika IDS:er som du använt, i vilken utsträckning känner du att du kunnat aggregera alarm.

–Tipping Point har en aggregerings-funktion inbyggt i GUI:t, men den var väll kanske inte så användbar.

–Det man är intresserad av oftast är att se längre tillbaks än bara det som man får in i GUI:t. Det kommer så stor mängd larm så att bara aggregera dom som finns där fyller inte så stor funktion.

–Om man däremot hade kunnat haft något som håller koll historiken för en source ip adress så man vet om den har triggat en viss signatur flera gången innan (ett par veckor tillbaka).

I hur stor utsträckning känner du att du har kunnat påverka hur systemen aggregerar alarmen?

–Ja, det har jag kunnat göra. Har framför allt kunnat stänga av aggregeringen. Jag har kunnat aggregera på godtyckliga fält i det GUI:t om jag vill. Nästan alla GUI:n tillåter att man sorterar på en godtycklig kolumn. Som vårt system ser ut där vi samlar allting i ett eget GUI där man ser alla händelser kommer in. Det har en sorterings funktion som jag använder ganska ofta. Då kan jag direkt se om det många som klumpas ihop under ett attribut. Jag kan sortera på signaturen och se om det är många många olika ip-adresser som triggar den då kanske den får lite lägre tillförlitlighetsgrad. Så just sorterings funktionaliteten tycker jag är värd nästan mer än aggregering.

Ett typiskt alarm har information om tid, alarmtyp, källa/destination, etc. Räcker denna information för dig eller behöver du mer?

–Det räcker för att utesluta en hel mängd larm. Men om det är något som är lite intressant då kastar jag alltid med mer information för att kunna komma till någon slutsats om hur vida det är intressant i verkligheten eller inte. Då är det typiskt att man tittar närmare på i vilken omgivning som den här signaturen triggar. Om det är på ett väldigt begränsat nätverk eller om det kommer från internet osv. Omgivningen är väldigt viktig. Kollar även på vilken riktning om attacken sker, på själva signaturen om den är väldigt gammal osv.

Använder du några kompletterande verktyg till IDS:en? (tcpdump)

–Ja det gör man. Tcpdump inte minst. Är det så att det är något som är tillräckligt intressant då loggar man direkt in på sensorn och kör igång en tcpdump.

Har du några script som automatiskt går igång vid ett visst alarm?

–Om man inte kan kolla till en slutsats genom att titta på larmet. Men om vi talar Juniper, där finns det tillgång till några packet före och efter alarmet i.o.m. att man kan ställa in att den ska fånga det. Då kan jag ju titta där och komma ganska långt på det. Skulle inte det vara tillräckligt, då loggar jag in på sensorn och kör igång en tcpdump för att bilda mig en uppfattning. När man väl har fått tcpdump:en, antingen så kollar jag på den direkt i skalet eller wireshark om man vill får lite bättre koll på vad som har hänt. Annars har vi lite hemmabyggda system som dekodar om det är obfuskerade javascript t.ex. eller om det är payloads har mycket hexkod o.s.v.

Sebastian: –Har ni det som en plugin/preprocessor till Snort?

–Nej det har vi inte. Utan det är fristående små Python/Pearl script där vi kan klistra in packeten i för att få ut någonting vettigt. För att se vad som däljer sig bakom så att säga.

–Det vi har byggt till Snort det är verktyg för att hämta pcap:en från Snort till skillnad från Juniper där man kan klicka på ett larm och se pcap:en. Så i Snort har vi konfat det så att den alltid sparar packet data för varje signatur. Men den sparar det i ett inte så lättillgängligt format. Det är svårt pcap till rätt signatur/händelse. Så det har vi byggt verktyg så att vi kan logga in och sen komma åt det då.

Hur ofta brukar du granska alarmen närmre? Är det dom flesta eller bara några få?

–Det är nog ganska få som jag går in och granskar närmare i den bemärkelsen. Jag tror att det rör sig om max 10-15%.

Vad är det som gör att du inte klickar på dom andra 90% av alarmen?

–Dels är det så att man alltid jobbar med att redusera antalet falsklarm, och det arbetet ligger man alltid lite efter med. Så man får in en del alarm där man vet att det här borde inte finnas här, men jag har inte tid just nu att filtrera bort det. Och dom som har jobbat före mig de senaste veckorna har inte heller haft tid att filtrera bort det. Så det finns där och man vet att det triggar och är ofarligt för det har man kollat på.

–Sen är det då den andra typen av alarm som inte är falsklarm men som inte är intressanta av andr anledningar. Dom kanske inte förekommer i en omgivning som är intressant, t.ex. om man ser maskaktivitet från externa adresser. Det är ju inte falkslarm för dom försöker faktiskt attackera men man ser dom precis hela tiden och dom är inte intressanta att kolla på.

–Sen är det lite gråskala där i mellan av olika anledningar.

Hur skulle du beskriva ett typiskt tillvägagångssätt för dig när du ser ett alarm?

–Då brukar jag först och främst göra någon typ av historikuppslagning/korrelering på source-ip-adressen för att se om vi har någon information om den här värden. Det kanske är så att kunden har meddelat att just den här värden får göra nmap-avsökningar men att vi inte har missat att ge vårt backend system den informationen så att larmet triggar ändå. Eller också kanske det är så att kommer det från dom här värdarna då är det jättekritiskt, då ska man skicka insidentrapport direkt. Så först försöker jag kolla upp source-ip-adressen i våra egna system och göra någon typ av historik och korrelering.

–Sen går jag vidare. Är det en extern värd då kanske man kollar om den är i från Korea/Kina osv. Nästa steg är nog att titta på packetdata om sånt finns tillgängligt. Vidare kollar jag på hur signaturen är definierad. (Återigen är Juniper bra där till skillnad från Cisco och TippingPoint där man inte har tillgång till hur signaturen är definierad. Så kan jag se det reguljära uttryck som dom har använt för att fånga just den här attacken.) Då kan jag göra en bedömning om just den här signaturen har en hög tendens att falsklarma eller om den är av god kvalité, och matcha det mot den packetdata som jag har för att se om den borde trigga eller inte.

–Sen läser jag på om själva attacken/exploiten som dom försöker utnyttja. Det finns oftast i GUI:t där jag kan komma åt CVE-nummer. Behöver jag bilda mig en bättre uppfattning ändå kollar jag på internet och läser på om attacken och sårbarheten.

Bygger du själv några kopplingar mellan alarm eller gör systemet det åt dig?

–Nej inte så mycket faktiskt. Det är inte så ofta som vi har den bilden framför oss. Vi har kanske bara en ganska begränsad del av nätverket som vi ser. Vi ser inte vad som händer sen riktigt. Vi vet kanske inte om att det målet som dom försöker attackera att det är en mail/web-server för den är inte åtkomlig utifrån. Nej jag tror inte det, inte just den typen av kopplingar. Klar man bygger ju ihop vissa signaturer som i kombination är extra intressanta om dom förekommer. Men just attackkedjor på nätverket, kan hända att jag glömt det för att jag inte jobbat på ett tag, men inte så mycket tror jag.

Eva Lina: –För Emilie hade det i SSIM. Där kunde hon ställa in sådana korreleringssaker som att om du har fått in såhär många alarm av den här typen då kanske det börjar bli intressant. Då kanske du bör titta om du har fått in alarm av dom här typerna innan dess. Eller om du får det här alarmet efteråt då är det jättefarligt.

–Lite grann kanske man gör det undermedvetet. Man kollar ju ändå den historiken som finns när man får ett alarm. Man kollar source-ip-adresserna och man slår ju även på själva signaturen för att se vad man får upp. Men det är väldigt mycket på manuell basis isf som man gör det.

–Fast det är klart har man proxy-, brandväggs- och ids-loggar hos en kund, då har vi mycket större möjlighet att göra den typen av korrelering. Man kan koppla en användare till en source-ip-adress t.ex.

Enter Magnus:: meddelar att Ola är sen...

Sebastian: –Jag tänkte på OSSIM; det hade ju även tillgång till syslog loggar... Den kunde i syslog:en se om det är någon som försöker göra en root-attack. Då kunde den se i loggen om det är någon som får misslyckade login-försök eller kanske t.o.m. lyckade login-försök.

–Precis, men det är inte så ofta vi har den möjligheten. Men om vi har det åtagandet från kunden att vi kanske har syslog:ar från deras servrar som vi kan koppla till händelser i IDS:en osv. Sånna system har kommit ganska starkt på sistone. Hos någon kund så har dom ett sådant system som vi loggar in på och tar fram information för att göra den korreleringen. Annars så är det våra egna system där det är lite mer eller mindre på manuell basis som vi gör den här typen av korrelering och kopplar händelser till varandra.

Eva Lina: –För läser man rapporterna som producerats, då är det ju dom här stora rosaskimrande landskapen som målas upp hur allt detta skall automatiseras med machine learning algorigthms. Men det finns ingenting av det ute i den riktiga världen än då?

–Nja. Det finns men dom säljer skitbra nu och dom kostar enorma summor pengar. En kund dom kör nått som heter Arcsight. Det är ett system som skall svälja allting och som skall göra den här typen inbyggda intelligensen i sig då. Men det känns som att dom missar lite. Dels så tror jag inte att den har så mycket intelligens i sig utan den samlar nog bara ihop allting på ett ställe och så spottar den ur sig det till användaren så får den sitta där bäst den vill. Och i slutändan är det operatörens erfarenhet som avgör om man kan fastställa om det är ett larm eller inte. För man kan omöjligen gå tillverkarens skala på om det är allvarligt eller inte. Det är så många fler parametrar som avgör om det är intressant eller inte. Så jag tror att det låter skitbra för någon som inte jobbar med det. Hos oss skulle det inte sälja så bra.

Eva Lina: –Det nämnde Emilie också att hon var inte riktigt säker på den här universala lösningen. Ett nätverk är som en individ, den är unik, den finns ingen annanstans i världen i princip. Så det finns inget som passar för alla nätverk.

–Det känner man också när man sitter och jobbar, man lär sig kundernas nätverk och vad som brukar trigga där och vad som är legitimt och vad som är avvikelser i det legitima så att säga. Så som jag ser det; visst det är signaturbaserade system som vi jobbar med men på vår nivå så är det avvikelsedetektering som vi arbetar med. Hur upptäcker vi någonting som inte brukar förekomma i den här enorma strömmen med händelser som vi får emot oss hela tiden. Och då gäller det att bygga system som upptäcker dom avvikelserna i det normala mönstret så att säga.

Sebastian: –Så dom bra systemen är helt enkelt dom som bara hjälper en att hitta avvikelserna. Det är inte så att systemet skall upptäcka dom riktiga alarmen själv utan det skall fungera som ett verktyg åt operatören bara?

–Ja. Jag har inte själv arbetat med Arcsight och dom här större systemen så jag har själv inte så stor erfarenhet av hur dom fungerar. Det var ju bara några år sedan som Unified Threat Management var ett begrepp där man skulle bygga ihop IDS med brandvägg med antivirus, all funktionalitet i ett. Men det slog aldrig igenom riktigt för att ingenting var riktigt bra från början. Bygger man ihop många lite halv-pissiga system tillsammans blir det inte bättre.

–Så nej, jag tror inte så himla mycket på det här att man skall aggregera ihop system i varandra och få ett flashigt GUI och sitta och arbeta med. Man måste nog tänka lite modulärt. Det skall vara skalbart systemet men ändå så att man kan skräddarsy det så som man själv vill.

Sebastian: –Så öppna standarder då helt enkelt?

–Öppna standarder ja. Helt klart. Hade vi kunnat arbeta med Snort bara så hade det varit bra men det kan vi inte.

–Men Sourcefire hade varit intressant att titta på. Det har vi snackat om att vi skall ta in i vårt testlabb och utverdera. Så vi får väll se. Det vore intressant att se skillnaden mellan Snort och Sourcefire och se om det blir väldigt mycket högre kvallite på larmen än vad Snort har.

Hur kategoriserar du alarm?

–Är det så att ett alarm kommer över den där tröskeln där man inte bara klickar bort det, utan det är lite intressant. Då försöker man fastställa vilken kvallite på det är på signaturen helt enkelt.

Sebastian: –Hur ser signaturen ut då. Är det ett regular expression eller nått annat?

–Om man nu har tillgång till det. I t.ex. TippingPoint har vi inte någon access alls till hur signaturen ser ut. Men i Snort och Juniper har man ju det. Det är reguljära uttryck. Dom använder ett Perl-bibliotek för att parsa dom här reguljära uttrycken. Kan man reguljära uttryck så kan man bilda sig en uppfattning om signaturen. Sen finns det lite fler parametrar man kan skruva på när man skapar en signatur. I Juniper kan man skapa custom signaturer själv, om man tycker att den är usel så kan man göra en kopia av signaturen och använda den som bas för en egen signatur. Då kan man modifiera det reguljära uttrycket om man t.ex. tycker att sql injection bara skapar en massa falsklarm hela tiden så kan man peta in någonting som minskar falsklarmen.

–Ne, så det är inget magiskt alls. Det är bara att komma åt den informationen helt enkelt och kolla på det reguljära uttrycket.

Hur kategoriserar du alarm?

- **“låg-nivå alarm”**
- **“ointressanta”**
- **“dubblett”, (orsak redan känd)**

- **"grad av tillförlitlighet", t.ex. en regel som triggar alarm kan vara väldigt allmän och orsaka många alarm varav de flesta inte beror på intrång och då ha en låg grad av tillförlitlighet eller en regel kan vara väldigt specifik och därför är alarm som genereras utifrån den regeln av hög tillförlitlighetsgrad.**
- **"bekräftelse", kan man hitta ett nytt event som bekräftar att alarmet är korrekt?**

–Jag tycker att man arbetar lite gran efter alla dom där. Vissa alarm är ju inte intressanta av olika anledningar och när man går lite djupare in på dom då har dom ingen större tillförlitlighet när man väl kollar på hur signaturen är definierad t.ex.

Eva Lina: –Du har inga egna kategorier förutom dom som vi nämnt?

–Nej, faktiskt inte. Jag har inga egna. Jag har nog inte så mycket mer att säga om det här utöver det som jag redan sagt. Magnus får trakasera mig efteråt om han tycker att han vill ha fler svar.

Hur skulle du betygsätta dom grafiska interface som du använt? Finns alla dom saker du vill ha där eller är det något du saknar?

–Det varierar såklart. Men om man ska sammanfatta det jag har sagt innan lite. Det är enormt värdefullt att dels kunna se definitionen på signaturen för att kunna bedöma om det är falsklarm eller inte. Och dels att kunna fånga paket, så att den fångar paket när signaturen triggar eller helst då innan så man ser vad som har skickats. Annars blir man alltid tvåa om man får sätta igång den efter att det har hänt. Det är otroligt värdefullt att se paket runtomkring själva händelsen.

Eva Lina: –Emilie ville ha ett enkelt sätt att gå direkt från GUI:t till command line så att man kunde köra sql-frågor direkt till databasen. För att ibland kände hon sig hemmad av GUI:t.

–Jo men absolut. Det är det som vi jobbar med också. Där finns det sätt där man kan gå hur långt som helst för att underlätta för användaren. Men i dom här kommersiella systemen där finns det ju oftast, i alla fall som vi jobbat med, den typen av funktionalitet. Utan det är ju i våra egna system. Dels så krävs det att man har en ganska stor databas med händelser som man kan söka i GUI:t. Men man kan ju inte spara hur mycket som helst för då blir det för långsamt, så kanske två månader tillbaka max.

–Men visst i det egenutvecklade gränssnittet som vi har där är det ju typiskt sådana där saker man vill göra. Man vill högerklicka på larmet, sätta igång en tcpdump på sensorn för att fånga in mer traffik. Göra uppslag/korreleringar direkt. Just den här typen av korreleringar och historiken där det är ju sådan funktionalitet som jag tycker skall finnas med när man klickar upp ett larm. Så att det har tagits om han om av backend-systemet så att säga. Så att man själv inte skall behövs sitta och göra massa slagningar i någon databas, utan den informationen finns där oavsett om man är ute efter den eller inte.

Sebastian: –Menar du typ vilka 5 källor som triggade samma alarm och sådär?

–Ja, den typen av statistik hade ju varit intressant. Om man tänker på en source-ip-adress som har triggat och man vill veta historiskt sett hur det har sett ut på den här sensorn, har den triggat många gånger, kanske hur andra operatörer har bedömt den här händelsen, osv.

–Men visst sen finns det ju mer högnivåfrågor som man kanske skulle vilja ställa. Hur ser det ut lite mer generellt med signaturer och source och destinations portar för att blida sig någon typ utav uppfattning just i den här omgivningen då. Och allting och allting kan man ju inte bygga in i bakomliggande system.

–Att man bygger in i alla fall dom viktigaste grejerna så att man får upp den informationen när man klickar på alarmen. Typ om man har ett obfuskerat javascript eller payloads osv, att man får upp det i larmet och kan dekoda det direkt. Så att man inte behöver url decoda någonting genom att gå ut på nätet och klistra in en url, bara för att se vad fan det är som han har skickar för jävla GET request här eller inte.

–Allting har ju som syfte att reducera den tiden det tar att komma till en slutsats. Och desto fortare man kan skicka iväg en rapport eller informera kunden om att det hänt nått här desto bättre. All dom här funktionerna som man kan bygga in i GUI:t som underlättar, det är liksom det man strävar efter hela tiden.

Du sa att ungefär 10% av alarmen brukar du klicka på. Hur ofta brukar du gå utanför interfacet och ut på nätet för att hitta information innan du kan bestämma dig för hur du ska klassificera ett alarm?

–Dom 10 procenten, då gör jag nog i princip alltid en korrelering i alla fall. Kollar i våra egna system. Historiken på händelsen och så där. Och i 5% av fallen så går jag in och tittar på signaturen och kollar på pcap datan om den finns tillgänglig. I 2-3% så går jag ut på sensorn och sniffar för att få ännu mer information.

Vilka förbättringar vill du se för att göra det enklare att upptäcka attacker?

–Rent GUI-mässigt, ju mer information som kan förmedlas direkt till operatören desto bättre. Jag skrev ner tre punkter där.

–Bättre korrelering. Det är en sådan där grej som är så oerhört tungt och kräver så väldigt mycket hårdvara för att uppnå det man vill för att det är så tunga frågor man måste ställa för att få den där korreleringen. Men mer korrelering och kanske bättre förståelse för vilka typer av enskilda signaturer som är intressanta tillsammans med andra signaturer. I sig så är dom aldrig intressanta men tillsammans med andra blir dom intressanta. Om tillverkarna kunde själva komma fram med egna metasignaturer mer så hade det hjälpt till.

–Sen generellt sätt så är ju kvalliten ganska låg på signaturer, det är väldigt väldigt mycket falsklarm. Om man kunde anställa några nissar som gör lite mer tester innan dom släpper signaturerna, så att det blir lite bättre kvalite och true positives.

–Sen skrev jag avvikelsedetektering. Det var lite grann det som jag nämnde tidigare att man på nått sätt kan bygga system som upptäcker avvikelserna i dom här enorma strömmen av händelser som man får. Där kan man ju gå in hur djupt som helst på den source-ip-adressen på just det här nätverket. Varför kommer avvikelsen, är det på signatur-nivå? osv.

–Men molnet kanske räddar oss och allt kommer bli bra...

Eva Lina: –Ja, hoppas kan vi ju alltid göra.

Skulle du föredra ett system som lärsig automagiskt hur du vill ha det, ett där man måste konfigurera allt (har full kontroll) eller kanske något där i mellan?

–Jag har nog inga problem att släppa kontrollen. Men samtidigt tycker jag att sen sånt system som lär sig av hur en opreatör klickar på alarm osv. Det tror jag skulle vara väldigt värdefullt. Men det känns nödvändigt att det är någon typ av IDS-administratör som kan gå in och se vilka beslut som har fattats och kan overrida dom beslut som har fattats och göra justeringar och ha insyn i den processen. Men jag tror att man skulle kunna komma ganska långt bara på att titta på klick om det är hyfsat erfarna operatörer som sitter där. Om man ser att de här larmen dom klickar man alltid bort, man hackar alltid bort dom utan att ens klicka upp dom.

–När jag börja där och man sitter med nytt GUI, sen blir man helt förstörd. Allt det kreativa tänkandet bara försvinner. Men då hade man massor med idéer om vilka förbättringar man skulle kunna göra med just GUI:t. Och då kommer jag ihåg att en ide var att system skulle lära sig att om fyra operatörer hackar bort ett alarm i en veckas tid eller två dagars tid, det kanske man kan klassa som grönt (inte så intressant). Sen kanske någon IDS-administratör kan samla alla gröna alarm och göra en översikt och sen säga att; okej dom här alarmen dom tar vi inte in i vårt system längre. Och så kanske man har röda alarm som man alltid dubbelklickar på och man är alltid på sensorn och gör en tcpdump. Dyker någonting sådant upp igen då taggas det automatiskt rött.

–Någon sådan typ av visuella hjälpmedel så att man direkt kan bilda sig en första uppfattning skulle vara bra.

Eva Lina: –Mmm, för det snacka vi om att vi skulle göra. Att vi har toppen med dom högst rankade, dom röda, sen tänkte vi ha en lista med bubblare med dom som ökar mest, där deltat är som störst. Och sen sen lista med helt nya alarm som aldrig setts på nätverket innan och det är ingen som har klickat på dom någonsin. Och så naturligtvis så att man kan få tillgång till alla alarm. Men att ha dom där tre tyckte vi kändes rätt vettigt.

–Absolut, det känns jätte vettigt.

Eva Lina: –Men vad bra då var vi inte helt fel ute då iaf.

Är det viktigt att kunna ge negativfeedback? Att kunna ranka ett alarm lägre och ange anledning(ar) för detta till systemet? Om man t.ex. har en dator i nätverket med en känd bugg/svaghet där det

är viktigt att veta om en attack sker. Men om den senare patchas och man inte längre vill se den och isf ta bort den m.h.a. negativ feedback.

–Absolut det är också en funktionalitet som är väldigt viktig i ett sådant system. Så att jag ser dom inte som ömsesidigt uteslutande dom här olika systemen ni har. Sen kanske man kan tänka sig att i dom fallen det är en viss typ av sårbarhet och man vet att nästa tisdag så skall Microsoft släppa sitt service pack för det här. Och vi kommer skicka incidentrapporter till kund fram tills dess men efter det är det inte intressant. Då skulle man vilja sätta någon life-time på den här signaturen: den är röd tills tisdag, och sen går den ner på grön. Så man kan sätta någon tidstag på dom.

–Men absolut, manuell negativ feedback. Vem som ska göra det, om operatören skall kunna göra det själv eller om det skall vara en IDS-administratör som skall göra det, det är väll frågan man kan ta ställning till. Men möjligheten den känns viktig.

Eva Lina: –Magnus hade ju fyra olika interaktionssätt med IDS:en. Dels bara att man tittar på var klickar användaren. Dels att användaren själv kunde ange anledningen till varför han klicka på alarmet. Och så att systemet själv gissade på hur användaren ville ha det, så får man sitta och hålla med eller inte hålla med. Och så tillslut att den frågar explicit om hur den ska ranka alarm.

–Vad säger du om dom här olika graderna av interaktion? Eller vill du bara att den skall hålla käften?

–Det är rätt stressigt att sitta där. Man hinner inte ge feedback till systemet så att det lär sig. Och att sitta och fundera på var det hamnar på en skala mellan ett och tio, det tror jag inte skulle funka för oss i alla fall.

Eva Lina: –Så det här med att ange varför du är intresserad, det är inget för dig?

–Nej. Bara ren klickning och sen i efterhand att någon med stor erfarenhet kan gå in och göra dom förändringar som behövs. Det tror jag är mycket bättre.

Eva Lina: –Men det här med att man kan hålla med eller inte hålla med att man har t.ex. en ja- eller nej-knapp. Så att man kan tala om för systemet att ja det här var bra rankat eller nej det här var dåligt. Vad säger du om det?

–Den kräver ju inte så mycket interaktion. Den är nog inte så dum. Är det ett larm som klassas rött och sen tycker man att det här har blivit fel. För det är bara en massa urusla operatörer som har suttit och rankat för jag vet att jag vet att det här är inte intressant. I dom fallen hade nog varit bra att ha den möjligheten i GUI:t.

Eva Lina: –För då har man ju dels det här att det blir ointressant över tid, att man kanske slutar klicka på det och det långsamt dör bort. Och dels så kan man aktivt döda det väldigt fort.

–Just det, så man får båda världarna där.

Eva Lina: –Jag antar att den där sista väldigt verbala och pratglada IDS:et som hela tiden frågar om hur alarm skall rankas inte heller är något för dig?

–Nej, det är absolut ingenting för mig.

Ser du några nackdelar eller risker med ranking?

–Ja det är klart att det finns. Det gör det ju. Det kan vara så att man lägger för stor tillförlitlighet i den här rankingen. Och sen vaggas man in någon typ av... Säg att man har en skala på ett till fyra och sen är det nått som är på två och dom ger man fan i att titta på för att dom bara är på två. Men det finns andra orsaker till att den är där nere och den borde inte egentligen vara där nere, utan den borde vara lite högre. Hade man inte haft rankingen från början då hade man kanske ägnat den större uppmärksamhet. Det är där det krävs någon med lite större erfarenhet som kanske går in med jämna intervall och kollar den hur rankingen sker och av vilken anledning den har fattat besluten så man kan göra justeringar.

–Och man kanske också ska ha ett system där man låser vissa signaturer till en viss rank för att dom är alltid intressanta. Det skall inte användare och operatörer kunna hacka bort, utan dom är låsta och man vet att dom larmen är kritiska. Men annars kommer jag inte på några omedelbara nackdelar med rankingen.

Eva Lina: –För det var väll det som Håkan Nohre på Cisco sågade ranking med. Han tyckte att någon kan ju ranka det fel och då kan det tippa hela systemet. Och vad kommer hända om användare ändrar sig efter ett tag, vad ska vi göra med det då? Ne det verkar inget bra, tyckte han.

Sebastian: –Ja det är ju det som är problemet. Grupppsykologi kan ju få en inverkan med. Då kanske man får tänker nått i stil med; det här är ju grön, då kanske det är någon som vet bättre än mig som har rankat. Man börjar tvivla på sig själv.

–Exakt. Den risken finns där. Men jag tror att ett sådant system sätter lite krav på dom som jobbar med det, så att dom har ganska stor erfarenhet. Den här problematiken den kan man ju komma run också genom att man kanske viktar användares möjligheter. Så att en som bara har jobbat där i ett år kanske inte har möjlighet att ranka så hårt. Utan hans klick betyder mindre än någon som har jobbat där i tio år. Då reducerar man riskerna att en nykomling sabbar hela systemet.

–Men jag tror att den mesta av den problematiken som Cisco snubben tar upp, den kan man nog komma runt om man lurar lite grann. Om man ser det som en hjälpreda och inte en absolut sanning. Utan att man tar det för vad det är.

Har du några övriga kommentarer eller synpunkter?

–Nej jag tror inte det...

Sebatian: –Jag tänkte på det du sa att när man klickar på alarm så kan man få upp relaterad information, statistik och annan intressant information. Tror du man skulle kunna använda den informationen för att på nått sätt ge en slags rank- eller relevans-poäng till alarmet? Om man hittar mycket relaterad

information och ser att ett source ip har orsakat en jävla massa alarm de senaste tio minuterna...

–Absolut. All den typen av korrelering som görs den borde ju användas. Den har ju som syfte att man ska bilda sig en bättre uppfatting om händelsen och den hjälper ju. Den skulle ju också kunna vara inbyggt i rankningssystemet. T.ex. om en signatur triggar men source-ip-adressen triggade också en annan signatur igår på det här nätverket, och det gör att den här händelsen klassas som allvarligare än om den bara var isolerad.

–Där kan man gräva jätte mycket känns det som. Vad ska man titta på där. Signaturen? Source-ip-adressen? Vilken miljö den förekommer i? Vilka tröskelvärden skall man sätta? Tillsammans med vilka andra signaturer kan det vara intressant? Hur långt tidsspan skall man ha, någon månad bara eller ett år?

–Men absolut. Jag tror att det skulle kunna hjälpa om man förde in den informationen i rankningsprocessen.

D.3 Emilie Lundin Barse

Transcript from interview of Emilie Lundin Barse from Combitech. The interview was done in person and recorded.

Intervjun börjar med en demonstration av verktyget OSSIM.:

–Snort är ju egentligen huvudkomponent i OSSIM och det som du får intrångsdetekteringslarmen ifrån. Idén med dessa verktyg är att du ska kunna få in information från andra källor och loggar, etc. I den gamla installationen plockar vi in loggarna från brandväggen och då kan man korrelera droppade anslutningar, till exempel folk som försöker komma in genom brandväggen på portar som är stängda, med larm om de IP-adresserna dyker upp i något sammanhang. Det är intrångsdetektering med add-on-information.

–Så här i den gamla versionen har du BASE-gränssnittet men de håller på att lägga upp sin egen eventviewer som de har valt att kalla den. Där kan du dela upp saker i olika flikar. Du kan också installera *snare* för att plocka in windowsloggar och OSsec som är en hostbasered linux-IDS och då poppar de upp här.

–Den kollar lite på vad som händer på själva sensormaskinen också. Alla larmen hamnar i den här fliken, men detta är ju det som kallas *forensic*, det mer att gräva i när man redan vet att något har hänt. De har en larmflik där korreleringsregler eller direktiv gör att högprioriterade larm eller events poppar över i larmfliken. Tanken är väl att man bara ska kolla på de som är högprioriterade.

Eva Lina: –Ställer man in i någon konfigureringflik vilka som prioriteras?

–Ja, man får lägga in sådana regler. Men den har ett gäng inbyggda också. Om man får ett snortlarm på en attack mot en port har den nätverksverktyg som kollar om porten svarar på trafiken. Om den gör det får det högre prioritet och blir till exempel ett larm. Man kan lägga in en massa sådana som de kallar policys.

–Om man tittar på direktien, har den sådana här *recurrent snort event*. Får man väldigt många *snort events* av samma typ larmar den på det.

Sebastian: –Kan man göra motsatsen också? Om man får väldigt många, kan man ignorera dem då kanske?

–Jo man kan skriva en policy. Man använder den till direktiven men man lagrar dem inte, så de ligger inte kvar i den här eventfliken. Den gör bara korreleringen i minnet men lagrar inte dem i databasen.

Eva Lina: –Spar ju minne.

–Ja, jo. Den använder ju p0f som är en *passive OS fingerprinting* och där kan den säga *host same* och det är inget man sparar på. Den har detekterat att det forfarande är en windowshost, samma maskin, det är kanske inte något man vill utreda direkt.

–Om man gör utredningar så är det faktiskt ofta de positiva händelserna som man vill ha och de felaktiga. Ta brandväggsloggar som exempel, om du hittar någonting som gör att du tror att en server är hackad så är det ju mer intressant med vilka sessioner som har kommit igenom brandväggen. Faktum är väl att det ofta är många utredningar där det är viktigare med vad som har lyckats n vad som har misslyckats. Lyckade inloggningar kan vara viktigare än misslyckade.

–När du får upp ett alarm kan du klicka på det och då få upp listan över events, det kan ju vara flera events som orsakar samma larm. Vi kan ta och titta på några olika regler, vi har gjort en del egna, just då med brandväggsloggarna och lite andra saker. *Rare but open destination port* har lite sådana regler. Om det är mer än 30 sekunder som sessionen pågår prioriterar man upp den. Det är ganska bra, att inte bara ha intrångsregler utan också andra saker.

Sebastian: –Kan jag använda de här reglerna rekursivt? Om jag då vill ha ett attackscenario, kan jag då säga att det här beteendet liknar den här trojanen, och sedan händer det här. Jag refererar till ett id och säger att det här händer först och sedan det här?

–Du får lägga ihop det i samma direktiv tror jag, men det går ju att göra flera egna grejer i samma direktiv. Men jag vet inte riktigt hur, har inte använt det.

–OSSIM är det jag använt mest. Jag tycker man kan känna sig lite begränsad här och skulle vilja ha bättre möjligheter att sortera själva på vilken parameter man vill och ha ett friare gränssnitt, mer ett *sql query*-gränssnitt med stöd för regexpar.

Sebastian: –Så att man kan göra ungefär som views i sql så att man får en egen?

–Det finns, jag har provat ett par stycken sådana här *security information management*-verktyg som är ännu mer inriktade på att man stoppar in alla sin loggar i dom snarare än ids-larm. RSA enVision är ett verktyg där de har ett sqlgränssnitt. Fast man får lite stöd så att man inte behöver skriva queries utan du har lite hjälp.

Sebastian: –Men finns det inte någon sökfunktion? Det har en i BASE.

–Du kan ju lägga in tidsperioder. Gränssnitter är ganska krångligt men du kan lägga in tisdperioder och du kan lägga in IP-adresser, vilken sensor larmet kom-

mer ifrån och så vidare, men du kan inte söka på riktigt vad du vill. Man skulle vilja ha något mer liknande googleindexeringssökning.

–Jag har använt *Splunk* också. Det används till att stoppa in textbaserade loggar i och det gör någon slags sökmotorindexering av loggarna. Det är ganska fräckt, för man skulle vilja ha den hjälpen i OSSIM.

Eva Lina: –Men du brukar alltså klicka runt på alarmen?

–Ja det gör jag. Jag tycker att man nästan alltid behöver gå in och titta på innehållet. Den har ju många källkodlarm och *sql injection*-larm och då får man i princip hexkoden där nere. Jag tycker att det nästan är alltid man behöver titta på det för att se om det är ett falsklarm eller inte, särskilt när det gäller källkod. Om man har det aktiverat falsklarmar det väldigt mycket. Ofta triggar den på vissa websidor.

Eva Lina: –Tycker du att det räcker med informationen som finns här?

–Nej, egentligen inte.

Eva Lina: –Vad brukar du vilja ha förutom detta?

–Sessionsinformation, man ser om det har etablerats en session och hur mycket trafik som har gått och eventuellt vilka andra maskiner attackeraren har pratat med. Det andra verktyget jag använt heter *Sguil*, är nästan ett snortgränssnitt. Det kör jag faktiskt hemma. Det har nog utvecklats av det amerikanska försvaret som sitter i 24/7-övervakning och verkligen tittar på varje larm som kommer. Så i och för sig, det är gjort mer så att man verkligen tittar på varje alarm.

–Här har de markerat ett alarm, *mysql insertion overflow attempt*. Det är typiskt sådana jag ser hemma också, bakgrundsbrus på internet kan man väl säga. Det är speciellt infekterade maskiner i Asien, jag antar att de inte har någon update på sina maskiner...

Eva Lina: –Det är väl rätt mycket piratkopierat där borta också?

–Ja, så de betalar väl inte för sina windowslicenser direkt och då har de ingen uppdatering på dem.

–I det här gränssnittet delar dom upp det i högprioriterade alarm, mellanprioriterade och lågprioriterade i olika *panes*. Man kan konfigurera om det lite grann och sedan kan man klicka upp regeln som har triggat och så visar den paketinnehåll och så kan du även här göra en lookup på IP-adressen. Om man söker på nätet på det som står i paketet ser man att det är så här det ser ut när en infekterad maskin försöker sprida sig vidare.

–Det här gör åtminstone aggregering, aggregerar på käll-IP tror jag. Den är ju lite korkad då, för vissa saker ill man ju aggregera på distinationsIP och andra på käll-IP. Den har *count* här, så den här har 64 stycken likadana larm här.

Eva Lina: –Hur många möjligheter har man här att aggregera annorlunda?

–Här är det bara att den samlar ihop samma, så här har man inte så många möjligheter. Tanken är att man ska eskalera, att man tittar på varje alarm och sedan kan kategorisera dem som ointressanta och då försvinner de.

–Jag har gått nån kurs *SANS Institute*-kurs, de har bra säkerhetskurser. Jag har gått deras *intrusion detection*-kurs och där berättar de hur de jobbar i de här verktygen. De har *junior analysts* som sitter och kollar på alla alarm som kommer upp. Sedan har de *senior analysts* som kollar på det som har eskalerats.

–Man kan ju alltid sätta på coh stänga av snortregler och lägga in egna regler. I vissa sammanhang har man väldigt fokuserad detektering och då har man bara väldigt specifika regler för saker som man vill upptäcka för en viss tjänst.

–Inom försvaret tittar man oftare på vissa IP-adresser, övervakar lite vad dom sysslar med. Man använder i princip inte standardreglerna.

Sebastian: –Hur mycket folk har dom som sitter och kollar? Är det så de löser problemet? Att de har senior och junior? Hur mycket låter de systemet klura ut själv vad som är farligt och inte farligt eller gör de helt enkelt så att de har en himla massa folk som tittar på allt och så får de bedöma vad som är farligt och inte farligt?

–Det blir ju så, man tittar på vissa saker mer noga, man har ju begränsat med tid. Ett tips jag fick på hur man kan kolla är att titta på de mest allvarliga alarmen och sedan titta på sådant som dyker upp som enstaka händelser för att inte missa något. Just det här med droppade saker i brandväggen, vad som är ganska vanligt är att man tittar på *top 10* och *bottom 10*. Få händelser är antagliggen någon som manuellt försöker göra någonting. *Top 10* är oftast scanningar då de bara scannar igenom allt.

–Men *Sguil* gillar jag. För hemmabruk där man har rätt lite trafik är det bra. Det kommer bara att rulla föbi alarm om man har det mesta påslaget som vi har gjort i vissa sammanhang. Det här är mer gjort för att trimma ner lite på reglerna skulle jag nog tycka, medans i OSSIM har man filosofin att man sparar på alla alarm medans man inte tittar på alla om man inte tror att något specifikt har hänt. Det är lite olika sätt att jobba på.

Sebastian: –Men då funkar inte OSSIM lika bra i högtrafiksammanhang?

–Jo det gör den ju, för du behöver inte titta på allting som kommer in. vi har haft lite samarbete med Saab Microwave så vi har det installerat hos dem. Det blir jättemycket men det går rätt bra, man får rensa lite så att diskarna inte blir fulla. Och det finns begränsningar, man kan inte ha hur mycket som helst i databasen samtidigt. Men det är ju problemet med alla databaser. Det blir ju rätt segt om man har ett par 100.000 events i databasen. Sökningarna blir väldigt långsamma. Då kanske man inte kan ha mer än ett par dagars händelser samtidigt. Man får nästan ha en maskin som importerar och exporterar data för de tidsperioder man vill söka.

Sebastian: –Det har man ju läst om, att då det går för långsamt att logga till databasen dumpar dom till något binärt format och så får andra maskiner stå och tugga igenom den där filen och lägga den i databasen.

–Man får nästan bygga upp en sån struktur, att man har en maskin som samlar in data, en som gör realtidsanalys och sedan en separat maskin där man gör de här manuella sökningarna eftersom de stör realtidsanalysen. Så man behöver två analysmaskiner.

–På Microwave har de kört Snort ganska länge och om du köper Snort har den ett litet bättre gränssnitt än BASE på den kommersiella versionen.

–Det tar oftast två år innan man är nöjd, konfigurerar lite regler, stängt av det som stör. Ofta upptäcker man konfigurationsfel i nätverket och då åtgärdar man kanske de grejerna i början, så det är väldig tmycket som ligger och stör i början. Jag har hört det från flera att det tar cirka två år innan man känner att man kan jobba i det.

–Men jag kan säga att jag har egentligen inte jobbat med intrångsdetektering. Jag har gjort två installationer av OSSIM i två olika nät och vi har gjort vissa anpassningar, grejat med regler och policys och lite direktiv, men vi har inte varit med hela vägen tills det blev användningsbart. Det blir ju mycket skräphändelser. Det var ju som någon sade att det första man behöver konfigurera bort är övervakningsmaskiner som säkerhetsavdelningen använder, för de ligger ofta och spyr ut trafik som triggar regler. Att just konfigurera bort att maskinen själv generar händelser är en sak man behöver göra.

Sebastian: –Då är de alltså de hetaste maskinerna man kan infektera i ett företag? Det är de maskiner som säkerhetssystemet ignorerar.

–Precis. Faktiskt, det är väldigt känsliga data om man hittar en sådan databas och kan gå in och titta. Det har varit vissa sådana verktyg som per default varit öppna mot internet. Att gå in och kunna titta på ett företags snortlarm är ju inte bra. Man kan se vad det är för IP-adresser, vad maskinerna har för funktion. Jag kommer inte ihåg vad verktyget hette, jag tror de tipsade om det i kursen jag läste och fick prova igenom några olika verktyg.

Sebastian: –Google är ju bra på det där också, hitta sådana saker alltså. Jag kan tänka mig att man kan göra en sökquery i google för att hitta alla BASE-installationer.

–Just det, det var nog så man hittade de här maskinerna som hade det här gränssnittet öppet. Det var någon sökstärng man kunde använda för att hitta alla öppna sajter.

Vilka gränssnitt och IDSer har du använt?

–Gränssnitt: Sguil, BASE, det är snortgränssnitt allihop. Sedan har jag också använt Cisco MARS. Det kan man väl också se som ett gränssnitt, men det är egentligen ett *security information managment*-verktyg. Det är ju ett ramverk.

Hur tyckte du att man kunde sköta aggregering i det?

–Nu körde de ju inte Snort så jag såg inte riktigt det egentligen, men det hade andra roliga features i gränssnittet. När du fick upp ett larm kunde du klicka på någon liten ikon och få upp hur trafiken gått i nätverket. Man samlade inte loggar från routrar, switchar och brandväggar. Man använder ju inte Cisco MARS om man inte har rätt mcyket ciscoutrustning, men man kunde få upp att "jaha då har det gått genom den switchen och den routern och blev stoppat i den brandväggen". Det är ganska snyggt. Då vet man ju att trafiken faktiskt inte gick fram till servern den försökte attackera. Det är sådan övrig information

som är väldigt bra att ha, just det här med brandväggens logg där man får veta både vilka sessioner som har gått igenom och vilka som blivit stoppade.

Sebastian: –Men det är bara om man använder Ciscos? Det finns inga öppna program för att...?

–I OSSIM kan du göra det. Du kan plocka in loggarna från brandväggarna och egentligen göra spårningen själv. Cisco har ju tagit det ett steg länger än OSSIM. I OSSIM kan du också se om maskinen har svart på trafiken.

–I Sguil har en agent för sessionsloggning. På det nätverkssegementet du lyssnar på loggar du alla sesioner, vilka IP-adresser som har pratat med varandra på vilka portar och hur mycket trafik som har gått. Du kan högerklicka på ett event i Sguil och få upp vilka andra maskiner den attackerade maksinen har pratat med mer på nätverket. Där har du kanske ännu bättre möjligheter till det.

Men om du tänker på alla interface du använt, är det bra aggretionsmöjligheter eller blir man frustrerad och känner att man vill göra mer och annorlunda?

–Ja, man skulle vilja att de var mycket bättre (skratt).

Eva Lina: –Kan man påverka dem på något sätt och tala om för dem att "så här vill jag aggregera och gör det åt mig"?

–I princip. OSSIM är open source så du kan ju lägga till dina egna plugins. Du kan ju lägga till en egen flik där du gör en massa annat. Men det är ju klar, det är ju en massa jobb.

Eva Lina: –Det finns förbättringspotential alltså?

–Ja, det finns förbättringspotential i alla verktyg. Kan ju lägga till RSAenvision där också. Men det är mer allmänt för loggar och jag har använt ett antal sådana verktyg. Det finns ett som heter ArcSight också som är generellt för loggar. Men dessa två är generellt för loggar och inte bara för nätverkssäkerhetsövervakning. ArcSight har jag inte använt men jag har sett lite av det. De är ganska mycket för korreleringsmöjligheter, men det är som med OSSUM, att man får lägga in det själv. Alla system är unika så alla nstallationer av sådana system är flerårsprojekt. Man kan inte bara installera det och tro att det fungerar.

–Du skulle kunna stoppa in snortlarmen i *Splunk*. Det är mer googleindexering på textloggar, men du skulle kunna stoppa in alarm i den också. Där har du ganska bra stöd att tala om vad olika fält är för någonting. Där har man lite mer sådana sökningar som jag skulle vilja göra.

Sebastian: –Så det skulle man kunna kombinera med någon av de andra?

–Fördelen med Snort är ju att du kan logga i olika format och inte bara binärt. Det finns ett plugin som heter Barnyard, det är en egen process. Snort loggar binärt och så har du Barnyard som gör om det till flera olika format. Jag tror på att man vill ha flera analysverktyg till samma loggdata. Det är nog inte ett verktyg som är rätt för alla system.

Vad vill du ha ut av ett alarm? Det vanligaste är ju att du får ut tid, alarmtyp och källa, destination osv.

–Paketdata vill jag definitivt ha, annars kan man oftast inte säga någonting om det.

Eva Lina: –Du sa att du titta på sessionerna också, är det något mer än det du vill åt?

–Egentligen vill man ju ha allt (skratt). Det första man gör är att klicka på larmet, titta på paketinnehåll. Nästa steg är att kolla vilka andra maskiner den här maskinenn har pratat med. Vilka andra alarm har den här maskinen orsakat. Nästa steg är att gå in i serverloggarna och kolla, kan man ha separat, om det ser ut som om något har hänt på den här maskinen. Om det ser ut som om den larmar för att det finns någon bakdörr på en maskin är det klart att man går in och kollar om bakdörren finns där. Men då har man kommit lite utanför det som intrångsdetekteringssystemet ska göra.

–Men om det har hänt någonting kan man spela in fullständig nätverkstrafik under en viss teidsperiod. Jag vet folk som har gjort det. Man kan lägga upp en egen process som spelar börjar spelar in. När du får ett visst snortlarm spelar du in all nätverkstrafik under kanske 5 minuter för att få mer information och då får du också en mer komplett bild utav vad som har hänt. Men har du många larm kan det bli hur mycket trafik som helst, fast du kan ju välja att bara ta för en viss IP-adress då i och för sig.

Eva Lina: –Är det några andra verktyg förutom de du har visat oss som du använder för att komplettera bilden?

–Informationsmässigt sett?

Eva Lin: –Magnus pratade om tcpdump...

Tcpdump är väl egentligen standrad 1A, så det använder jag. Man kan skriva filter i det också. Det är ju också till för konfigurationen, att förstå varför den triggar på vissa saker och för att felsöka. Tcpreplay för att testa, man kan spela upp en fil md trafik som man vet att det finns attacker i, så kan man ser att den triggar på det.

–Vad har man mer? Wireshark har man för att kunna göra ytterligare analys. Men OSSIM innehåller flera andra verktyg, så den innehåller p0f som är en passive OS fingerprinting. Den kollar trafiken och ser vad det är för operativsystem på maskinerna som pratar. Och så har den NTOP som gör viss trafikanalys så att man ser vilka protkoll som gå. Man ser trafikmängd.

–Vad som även är intressant är att se hur många events man får. Det finns ett verktyg som heter pnGraph som tittar på Snorts statistikfil och ritar grafer på det. Man ser över tiden hur många alarm man har fått, och så ser du trafikmängd över tiden.

–Över huvudtaget trafikbeteende, NTOP visar det. Det är ett open source-verktyg, men där får man göra lite konfigurering. Man behöver en liten server men det ingår i OSSIM om man installerar det.

–SGUIL har ju fördelar helt klart för hemmabruk där man får ganska lite händelser. Du har bättre sökmöjligheter tycker jag. OSSIM har varit lite buggigt

också men det är ju klart, det har utvecklats en del. Det finns en kommersiell version med appliance, en hårdvarulåda som du installerar.

Eva Lina: –När det händer något och du till exempel slår på tcpdump, har du automatiserat det eller går du in manuellt och startar det när du ser att något händer?

–Det har ju varit pilotinstallation det jag har gjort. Jag har ju inte drivit det hela vägen tills det är fullt fungerande, så det har bara varit att jag har gått tin manuellt egentligen. I och för sig, jag har ju gjort ett script som spelar in med tcpdump hela trafiken under en viss tridsperiod efter ett larm. Det har jag gjort någon gång.

Sebastian: –Läggs det in i snortfilterreglerna?

–Nej, det är helt separat, du gör en plugin som är kopplad till snortreglerna.

Sebastian: --Jag har bara läst manualen, men där kan man lägga in samtidigt som man skriver reglerna för vad det är den ska reagera på, säga till den att göra vissa saker.

–Jo, men det var via det. Men det var ganska länge sedan så det är knappt att jag kommer ihåg hur jag gjorde. Sedan fick man göra ett annat script som såg till att rensa bort efter en viss tid så att det inte blev fullt på disken, för det har man ju också problem med om man spelar in trafik. Man får ha lite koll på vad man gör.

Eva Lina: –De här verktygen du har använt, finns det någon möjlighet för dem att samarbeta, att få dem att prata med varandra? Får man hacka själv eller är det hopplöst?

–Du menar tcpdump...?

Eva Lina: –Jo, och de andra verktygen man kan tänka sig att använda.

–Man får hacka lite själv, men de är ju gjorda för det. Tcpdump använder man ofta lite separat. I och för sig, just när du kopplar det till Snort har den stöd för att du ska kunna köra vilket program du vill när ett larm trippar.

Sebastian: –Omvänt då? Jag änkte på Ciscos MARSsystem. Där är det tvärtom, att de helst vill att det bara ska vara inom Ciscos sytem. Eller tillåter de också att man kopplar in lite vad som helst?

–Snort kan du använda ihop med Cisco MARS. Cisco har sin egen IDS och så stödjer de Snort. Men Snort är så spritt att man måste nästan stödja den. Men Cisco stödjer ju inte sina konkurrenters. Men däremot, OSSIM stödjer några olika IDSer, men de är ju inte heller bundna till någon och även ArcSight och RSAenvision stödjer flera olika. De är ju inte heller bundna till något IDS-företag. Tittar man på till exempel Symantec har de också någon programvara för att knyta ihop IDS-larm och andra loggar. Och då stödjer de ju framför allt sin egen. Men nästan alla stödjer sin egen IDS och Snort i alla fall.

Brukar du kategorisera dina alarm?

Eva Lina: –Du visade oss låg-, mellan- och högprioriterade larm. Brukar du lägga till egna prioriteringar förutom dessa? Magnus hade på förslag "ointressanta", "dubletter - orsak redan känd". Har du några egna indelningar förutom detta?

–I SGUIL kan man lägga in olika kategorier, fast man känner alltid att de inte passar. Man vill ha lite egna. Men sedan har du att du kan lägga in din egen kommentar till larmet när du behandlar det. Det är bra. Över huvudtaget behöver man ha något så att man kan följa upp efteråt. Särskilt om man konfigurerar alarm måste man ha någon slags changelog på vad man har gjort och varför. Annars kommer man inte förstå det. Jag har använt mest det inbyggda men man skulle nog vilja göra mer än vad det finns stöd för egentligen. Jag skulle vilja göra egna kategorier och kunna flytta över dem och ha flera olika flika för olika kategorier.

–Jag tror att man altlid kommer vilja ha så att man kan konfigurera upp själv. Sedan har man kanske några som följer med.

Det finns allmänna regler som triggar på lite allt möjligt och så finns det mer specifika. Har du ngåon indelning av hur tillförlitligt ett larm är? Till exempel att för detta larm finns det 90% sannolikhet att det är riktigt och för ett annat larm är det bara 20% sannolikhet att det är riktigt. Använder du dig av en sådan indelning på olika alarm också?

–OSSIM har det själv. Snort har tre prioritetsnivåer på larmen men OSSIM har även reliabilitynivåer. Jag tror även att den lägger in det på snortlarmen, att vissa är mindre pålitliga än andra. Det där skulle man behöva sitta och konfigurera upp, för det är olika för olika system. I OSSIM kan du per larm gå in och sätta den här siffran och det är viktigt.

–Problemet är att vissa opålitliga larm handlar om väldigt viktiga saker så man kan inte bara gå in och stänga av dem.

Eva Lina: –Den korrelering jag sett får man ställa in själv i config-filer. Du har inte haft några automagiska system? De lär sig, korrelerar, dumpar ut en vacker fin, bild och talar om för dig att ditt nätverk är helt säkert?

–Näpp!

–De kan ju ge sken av att vem som helst skulle kunna använda dem då man ser grafer och så vidare, men i princip så bygger det alltid på att du har någon som verkligen vet vad gör i botten. Sedan kan du göra vissa managementvyer som man kallar det. Man lägger upp vissa grafer, till exempel antal larm över tid. Sådant kan ju management tycka är intressant. Ökar eller minskar hotbilden på något sätt?

Eva Lina: –En allmän regel med 20 eller 10% tillförlitlighet, hur bär du dig åt om du ska verifiera om det är riktigt eller ett falsklarm?

–Ibland kan man se i paketinnehållet. Är det något som pågår kan man spela in nätverkstrafiken och kolla på den. Annars är det bara att gå in på servern och kolla om det verkar ha hänt något, har trafiken ens kommit fram?

Eva Lina: –Och de grafiska interfacen? Känner du att du har något att tillägga eller känner du dig terapeutiskt rensad?

–Ibland tycker man att kommandoradsgränssnitt har sina fördelar, men det är ju både och. Man vill kunna få upp grafer över vissa saker. Men att gå in i databasen och göra sql-frågor eller sql-script används rätt mycket. När man har snortlarm är det rätt ofta man går in direkt mot databasen och kollar saker. Det går fortare, det är mer kraftfullt och du kan söka på vad du vill. De grafiska gränssnitten brukar ju tyvärr begränsa vad man kan söka. Problemet är att det alltid är så unikt hur man jobbar, det är svårt att få ett gränssnitt som passar alla.

–Men jag ser inget problem med att man har ett grafiskt gränssnitt som man gör vissa saker i och sedan kan du göra sql-sökningar bredvid. Men det är ju inte så dumt om det finns en liten flik som gör att du kan göra sökningar på vilka fält du vill i gränssnittet också.

Sebastian: –Det är väl det som är problemet. Det är två saker som är väldigt motsatta varandra. Att ha ett intuitivt GUI och sedan ha alla kraffulla verktyg som finns under. Det går inte att få ihop.

–Det finns inget intrågsdetekteringsystem som en idiot kan sköta egentligen - idag. Du får ju en massa konstiga falsklarm. På en kursen som jag gick, om du körde en viss p2p-trafik ihop med Junipers IPS tyckte den plötsligt att man hade trojaner på alla sina system.

–Jag tror inte riktigt på IPS. Man kan ju i sin brandvägg stoppa viss trafik som man är säker på är dålig. Men IDSen har en annan roll, den gör övervakning och den ska man helst ha passiv så att man själv kan agera på det man tycker är viktigt.

Om du får välja, fantisera fritt, vad vill du ha i framtiden från dina IDSsystem? Vad är önskeegenskapen du saknar idag?

Sebastian: –Vad är det manuella tråkiga arbetet som du helst skulle vilja slippa?

–Jag har ju inte gjort det manuella tråkiga arbetet. Det skulle man kunna fråga de på Ccode som sitter med heltidsövervakning till exempel. Det hinner ju bli tråkigt. Jag har bara gått in och tittat ibland. Vi har inte haft något uppdrag egentligen där vi haft ett kontinuerligt jobb med att gå in och titta på det. Men det är kanske det jag skulle vilja jobba med. Att man kan lägga en halvdag i veckan på analys och förbättringar åtminstone.

–Man kan ju ha samma information på flera gränssnitt, jag skulle kunna tänka mig att kombinera det som OSSIM gör med SGUIL. Man skulle vilja ha ganska många funktioner att välja mellan som att man kan göra en googleliknande sökning i loggarna och att man kan göra sql-frågor mot vilken information som helst i larmen.

Eva Lina: –Vad vill du helst ha, full kontroll eller skulle du kunna tänka dig ett system som automagiskt lär sig hur den ska klassificera trafik? Eller känns det lite läskigt?

Sebastian: –Eller som lär sig göra gissninar, göra kvalificerade gissningar vad användaren är ute efter?

–I så fall tror jag mer på idén att den frågar mig "jag tror det här" och så får du säga ja eller nej. Interaktiv inlärning. Jag har läst en artificiell neuronnätkurs och dom lär sig väldigt konstiga saker. Jag skulle nog inte lite på det. Självlärande är inte redo för någon användning just nu i alla fall. Men om man kan vara med och bestämma och förstår varför den klassar saker å ett visst sätt.

Eva Lina: –Då kanske du gillar Magnus idé med att den tittar på var du klickar och frågar lite frågor när den är osäker.

–På det temat så har ju Stefan Axelsson gjort en artikel, han var i vår forskargrupp tidigare. Han har gjort just det härmed interaktiv inlärning, att den frågar användaren och klassificerar saker. Det kanske kan vara lite intressant för det här. Som related work tycker jag nog att det kan vara bra.

Sebastian: –Så systemet ska helst INTE ta några egna initiativ utan att fråga först?

–NÄ!(skratt) Framför allt så vill jag ha kontrollen, jag vill gärna logga mycket saker så att det finns där. Men sedan måste man plocka ut vad det är man ska titta på i övervakningssyfte i realtid. Sedan vill jag kunna gå tillbaka in i databasen, så att man prioriterar upp vissa händelser som man faktiskt tittar på. Man vill ju ändå ha den stora högen med saker, att man går in där ibland, kollar vilka kategorier av larm man fått.

Sebastian: –OSSIM hade det här med relevans. Om man skulle ta och låta den påverka relevansen, när den försöker lära sig vilka saker som ...

–Jo den kan få lära sig vad som ska poppa upp i larmfliken kanske. Men jag vill ändå ha den stora massan med allting så att jag kan gå in och söka i det själv. Jag vill ha den kontrollen att allt finns kvar i originalformat och jag kan gå in och söka i det. Men sedan måste man ha något gränssnitt för realtidsövervakning för att på något sätt konfigurera det så att man hinner titta igenom det.

Sebastian: –Det skulle ju kunna vara en grej, att den i överblicken lär sig vad man tycker är relevant.

–Statistik behöver man också. Man behöver kunna få statistik på IP-adresser, vilka IP-adresser som har genererat mer allvarliga händelser, vilka som genererat mest händelser och vilka som genererat få men allvarliga till exempel. Sådana saker behöver man kunna få ut. Man vill kunan göra statistiksökningar i data för att veta vad manska titta på. I OSSIM kan du få upp grafer. Typiska är ju "top ten attackers". Men det är bara på antal larm. Sedan kanske du skulle vilja ha några smartare sätt, hur allvarliga de är också.

Eva Lina: –Men om du fick välja mellan de fyra sätten, att systemet bara tittar på vilka alarm du klickar på och antar att du är mest intresserad av dem du klickar på eller föredrar du att man klickar men väljer själv när man säger åt systemet varför något är viktigt?

–Jag skulle nog vilja ha att när man klickar upp ett larm, att man kan klicka i några radio buttons, att man har kanske fyra nivåer eller någonting, "hur viktigt var det här?". Och så vill man kanske ha den här kategoreseringen på något sätt.

Kanske med fördefinierade kategorier och kanske kunna lägga till egna. Så att man enkelt själv kunde sätta informationen.

Eva Lina: –Det här med att systemet försöker gissa sig till och istället för att du säger till den uttryckligen, gissar den och ger anledningen, och så kan du ange "ja jag håller med" eller "jag håller inte med"?

–Det är nog bra, så länge det inte är dumma frågor(skratt). Man kan ju bli väldigt irriterad.

Sebastian: –Då blir det som gemet i Word. "Du försöker skriva ett brev!" '..... näe .....' "Det ser ut som om du försöker skriva ett brev. Vill du ha hjälp!" 'NEJ!'

–Man måste nog kunna stänga av det där.

Tycker du det är viktigt att kunan ge negativ feedback till systemet? Att ett alarm är viktigt just nu, men sedan kommer det inte att vara viktigt efter att jag har patchat eller bytt ut den här servern. Att man kan slå ner det?

–I ArcSight, RSAenvision och även OSSIM så kan du ha en sådan här inventory list. Den vet vilka IP-adresser som är servrar som faktiskt finns och den vet vilka operativsystem de kör och vilka tjänster som kör på maskinerna, vilka programvaror som är igång på dem. Men det är också en massa konfiguration och saker byts hit och dit, så helst skulle det uppdatera sig självt, helst skulle det ha någon koppling till något configuration management på maskinerna så att man autmatiskt fick in det. Men det är kackligt att få det fixat.

–Men jag kan tänka mig att man vill ha en agent ute på maskinerna som faktiskt vet vilka programvaror som körs på dem, det finns ett antal sådana. Det gör att du kan göra en korrekt korrelering mot snortlarmen och prioritera upp dem om det är mot en sårbar version på maskinen. Det tycker jag definitvt, just för att sortera larmen. Sedan kanske man inte slänger bort de andra larmen heller, bara att de inte är högt prioriterade.

Kan du komma på några katastrofala nackdelar med rankning?

–Det är risken att den tar vissa saker och placerar högt och det andra hamnar där nere och så tittar man aldrig på det. Om inte rankningen är korrekt är det ganska stor risk att man missar något. Då upptäcker man bara det om något händer för då går du kanske in och tittar på alla larm som rör den maskinen och först då hittar du det.

Eva Lina: –Fast det är klart, den risken finns ju med felskrivna regler också.

–Visst.

Eva Lina: –Undrar vad som är lättast att hitta, en felskriven regel eller felaktig rankning?

–Testning är viktigt när det gäller regler. Men det gäller kanske även rankningen, att man behöver en testmetod för att kontrollera vad som händer. Över

huvudtaget saknar jag att det inte finns någon test bench där man kan spela upp trafik och få ett facit på hur det borde se ut, rankningsmässigt och vilka larm man över huvudtaget får av IDSen.

–Det är ganska lätt egentligen det jag gör nu. Jag har fått några TCP-dumpar när jag gått kurserna som man kan spela upp och det skulle vara jättelätt att göra testning med dem.

–Men sedan är det ju alltid så att om det finns test bench-data så kommer de anpassa sig så att de får bra resultat på det. Men det är rätt svårt att enbart installera de här verktygen och veta att allt fungerar som det ska. Bara det att se att allt är uppe och snurrar är steg ett. Men att kunna jämföra olika verktyg och se vilket som är bättre än något annat blir ju svårt när det är kända data. Man måste ju ta fram nya data till varje test. Men bara det att kolla att allt är uppe och snurar och beter sig som det ska vore jättebra.

D.4 Ulf Larson

Interview with Ulf Larson from Omegapoint via e-mail in Swedish.

Vilka IDS system har du använt?

- **Tillverkare?**
- **I hur stor utsträckning känner du att man kan aggregera alarmen i de system du använt?**
- **I hur stor utsträckning känner du att du har kunnat påverka hur systemen aggregerar alarmen?**
- **I hur stor utsträckning känner du att du har kunna arbeta med alarmen på en hög nivå / korrelera dem?**
- **Fördelar/nackdelar med det systemet?**

Jag har jobbat en del med Snort. Sedan har jag tittat en del på RSA envision, som snarare är ett SIEM (Security Information and Event Management System), vilket är ett steroidstint logg- och analysmonster med en IDS-modul. Jag har också använt pentestningsverktyg som burp-scanner (webbappar) och nessus (server/infra), vilken innehåller beslutsmoduler (om än lättviktiga) för att avgöra om en möjlig sårbarhet är en möjlig sårbarhet.

I enVision har man mycket stor möjlighet att aggregera alarm. Mycket goda möjligheter att välja bort och filtrera analysdata och larm, samt grafik, exempelvis cirkeldiagram, som visar hur stor del av angrepp som kommer från en viss IP-adress. Burp är ett mer lättviktigt verktyg och där har man något sämre möjligheter att påverka. Dock grupperar scannern själv resultat av ”samma” typ och utnyttjar en trädstruktur som följer strukturen på en skannad webbsite.

Mycket av iden med de stora SIEM-verktygen, som enVision är att det skall vara enkelt för en operatör att sättas framför verktyget. Om man bygger in mycket vettighet i verktyget skall operatören inte behöva vara expert för att

tolka vad som står (det hjälper, givetvis, men det är inte nödvändigt). Burp är på lägre nivå och erbjuder inte korreleringsmöjligheter på det sättet, eftersom den bara jobbar med en typ av "alarm", eller vad den tror är sårbarheter. Vill man korrelera får man utnytta något annat verktyg parallellt. EnVision är också ett Enterprize- system, vilket innebär att den i mycket stor utsträckning kan korrelera information från andra servrar som står "nån helt annanstans", och som "gör nåt helt annat". Dessutom (och detta gillar jag MYCKET), kan det korrelera in information från sårbarhetsdatabaser, dvs man kan utesluta vissa alarm, då den förmodade sårbarheten inte finns. Dessutom (igen) kan man ta in information från säkerhetsskanningar av ex... Nessus dårå.

För/nackdelar... I regel verkar det vara så att man får det man betalar för, men samtidigt så får man ju också vad man förtjänar... Mindre verktyg är lättare att komma överens med, men till en lägre finkornighet på inställningsfronten. Större verktyg tar längre tid att lära sig, men kan erbjuda nanokorn(ighet).

Har ni stora problem med "false positives"?

- **Hur hanterar du dom isf?**
- **Hjälper IDS systemet till med det på något sätt?**
  - **Hur?**

False positives i min nuvarande värld är en sårbarhetsskanner som säger, woohoo, en XSS i rapportgeneratorn. Sen är det upp till mig att avgöra om det är en fp eller inte. Med aktiv skanning kan verktyget själv försöka verifiera om det är en sårbarhet eller inte, exempelvis med strängen "45987276Isthisreflected234-", vilket skall föreställa en tillräckligt unik sträng som blir reflekterad till mottagaren. (Oj, svarade på nästa fråga av bara farten).

Man skall givetvis skilja på ett kritiskt larm, kritisk värd. Jag skulle påstå att detta är det lätt viktigaste överhuvudtaget (se vidare diskussion angående slipsnissar nedan). Det bästa sättet att göra detta är att låta systemet ta emot feedback från användaren. Systemet kan själv avgöra om det är en attack eller inte (→ alarm), på en teknisk nivå (den kan t.ex se att ett nätverkspaket innehåller "/bin/sh"). Däremot hur stor påverkan paketet har på just det egna systemet skiljer sig mycket och det är ytterst upp till operatören att tagga upp larmet i det specifika kontext han/hon sitter. (I extrem förlängningen borde man tjöta ned affärsansvarig som kan tala om att "om attack X går genom, då går vår business åt helvete, direkt"). Det kan vara bra input också, men försök få slipsnissarna till det, får du se hur det går.

I mitt jobb måste jag klicka på intressanta larm för att göra manuell uppföljning.

Vilka förbättringar vill du se för att göra det enklare att upptäcka attacker?

Grafik är alltid bra, och givetvis korrelering och aggregering. Feedbackbaserade system som tar data från den miljö de står i. Sen bör man fundera över att inte bara titta på inkommande trafik utan även i viss utsträckning på utgående trafik. Om en stor ISP får många datorer infekterade av en trojan och därefter

blir noder i ett botnät måste ISPn kunna märka detta. Dels på ingående när kommandocentralen talar om för noderna att börja mangla, och dels på utgående när noderna börjar kräkas ut uppkopplingsförsök mot externa målet.

Skulle du föredra ett system som

1. **"lär" sig automagiskt hur du vill ha det**
2. **ett där man måste konfigurera allt (har full kontroll)**
3. **eller kanske något där i mellan?**

Ett system måste givetvis lära sig under tiden. Inte nödvändigtvis bara av dig som operatör, utan även av hur nätverkstrafik beter sig, hur värden beter sig, hur din webbapplikation beter sig. Normalbeteenden är alltid bättre än signaturer. Skall man bli filosofisk kan man givetvis mena att det går att konfigurera upp "allt" så att systemet tar full hänsyn till allt dynamiskt beteende. Talar man endast om interaktion med operatören beror graden av interaktion på hur bra operatören är. En sån luffare som jag behöver förmodligen ett ganska statiskt system eftersom jag inte kan avgöra om larm X är en 1a eller en 10a på allvarlighetsskalan, medan en analysguru som Ola förmodligen hade fått mycket mer ut av systemet om han hade kunnat ge expertfeedback.

Vilket område utav de fyra vi tar upp, tror du har störst framtidspotential? Är det enbart ett utav dem eller är det en kombination?

Jag sätter min peng på feedback, dvs alternativ 3. Jag håller med er om funderingarna kring alternativ 2.

Vad tror du om att systemet lär sig användarens preferenser genom att t.ex. föra statistik på vilka typer av alarm som han/hon klickar på för att sedan kunna göra antaganden om hur nya alarm skall rankas?

Klickning är bra.

Samma som ovan men istället med en tumme upp och tumme ner knapp för alarm man tycker är viktiga/oviktiga?

Viktning är bra. Man bör förmodligen kombinera viktning med klickning. Frågan är ju om man skall vikta binärt, dvs bu (tumme upp) eller bä (tumme ned), eller om man skall ange en siffra mellan 1 och 10, eller snarare -5 till 5.

Är det viktigt att kunna ge "negativ" feedback? Att kunna ranka ett alarm lägre och ange anledning(ar) för detta till systemet?

Det är viktigt att vikta negativt. Man får då en grupp larm som man normalt sett kan strunta i (av olika anledningar). Det kanske till och med kan vara så att man kan vikta ett larm som irrelevant (motsvarar att man plockar bort signaturen från konfig-filen... typ).

Hur skulle man kunna tala om för systemet att från och med nu ska den inte räkna med de klick som gjorts på ett alarm? T.ex. innan en patch så var alarmen som hängde ihop med det problemet viktiga och man klickade på vart enda ett. Efter patchen däremot, inte av intresse och bör inte längre tas i beräkning när man rankar alarmen.

Man utnyttjar viktningsparametern, vilket innebär att ett "irrelevant" larm kommer att vara irrelevant oavsett om det tidigare har varit en 10a på allvarlighetsskalan och blivit klickad så höger musknapp totalhavererat.

Skulle det vara bättre/sämre om systemet istället ställde specifika frågor om hur den skall ranka alarm?

- **I så fall, vilka parametrar ska systemet ta hänsyn till?**

Vet ej. Hur kan ett alarm bara vara intressant första gången (om det inte är ett specialfall av ovan fråga)?

Vad är din omedelbara tanke (bra/dålig) om de fyra sätt att ranka alarm som vi tar upp? (clicking, directed learning, feedback, explicit questions)

Det är bättre, men för att vara generell kan systemet förmodligen bara ställa frågor utifrån rent teknisk påverkan (affärsvärden är specifika för specifik miljö, ju). En sårbarhetsscanner talar om för testaren att det här angreppet som den precis identifierat, fungerar på "det här sättet" och det är allvarligt "därför". Parametrar som man bör ta hänsyn till är hur stor potentiell påverkan en attack kan få. En attack som kan ge administratörsrättigheter eller lista tabeller i en databas bör viktas högt.

Klickning är jag lite kluven till. Man måste som analytiker förstå varför man klickar innan man klickar, dock. Detta får man ingen hjälp av från systemet så vitt jag förstår. Inget för nybörjare då systemet förmodligen kan bli fellärt om man inte riktigt vet vad man klickar på, eller varför.

Directed learning är också svårt för en nybörjare. Här måste nybörjaren själv tala om varför han/hon klickar alarmet. Möjligen kan en expert (som Magnus) ge understöd genom att först klicka sig genom en mängd testdata och sedan när systemet är tränat (och det förstår varför), kan man utnyttja "feedback" för att hjälpa mer juniora analytiker (som jag...) att fortsätta jobba med systemet.

Feedback gillar jag. Säkert olika. För nybörjare tror jag att feedback är den starkaste kandidaten eftersom den är den enda som hjälper analytikern att fatta vettiga beslut. Detta är superviktigt då de flesta inte har tid att bli experter (ett uttalande som får stå för mig, men ponera att det finns en sysadmin som skall rodda brandväggar, AD, IDSer, patchhantering, och som dessutom har 30 servrar som går upp och ned som en... karusell som går upp och ned).

Explicit questions tror jag blir lite samma problematik som directed learning.

Ser du några nackdelar med rankning?

De två nackdelarna jag ser är att systemet kan bli felinlärt, samt att om man inte tar höjd för trender, dvs har en korttids- och en långtidsstatistik, kan man missa viktiga trender.

---

1. **Ett typiskt alarm har information om tid, alarmtyp, källa/destination, etc.**
   - **Räcker denna information för dig eller behöver du mer?**
     - **Vad använder du i så fall för verktyg för att hämta mer information? T.ex. tcpdump, full packet capture, fångar all utgående trafik, minnesdumpning? Spårar du paket genom nätverket?**
     - **Vad är det för information i så fall du söker efter?**
   - **Hur ofta granskar ni alarmen närmre?**
     - **Vad är det för information som spelar roll för dig då?**
     - **Har du tillgång till verktyg som automatiskt hjälper dig med alarm så att du slipper göra vissa moment själv?**
   - **Om du använder olika verktyg för att komplettera informationen i alarmet, finns det någon samverkan mellan de olika verktygen och mellan verktygen och IDSen?**
     - **Om verktygen samverkar, finns den funktionaliteten inbyggd eller är detta något du fått utveckla/göra själv?**
2. **Hur skulle du beskriva ett typiskt tillvägagångssätt för dig när du ser ett alarm?**
   - **Vad får du för tankar när du ser ett alarm?**
3. **Vad är den vanliga arbetsgången? T.ex. jag ser alarm X. Letar efter händelse Y. Om X & Y finns, starta verktyget Z (kanske låter tcpdump logga allt mellan source IP och dest IP?) för att samla mer information för att bekräfta att alarmet stämmer.**
4. **Hur kategoriserar du alarm?**
   - **"låg-nivå alarm"**
   - **"ointressanta"**
   - **"dubblett", (orsak redan känd)**
   - **"grad av tillförlitlighet", t.ex. en regel som triggar alarm kan vara väldigt allmän och orsaka många alarm varav de flesta inte beror på intrång och då ha en låg grad av tillförlitlighet eller en regel kan vara väldigt specifik och därför är alarm som genereras utifrån den regeln av hög tillförlitlighetsgrad.**
   - **"bekräftelse", kan man hitta ett nytt event som bekräftar att alarmet är korrekt?**

5. **Bygger du själv kopplingar mellan olika alarm eller gör systemet det åt dig?**
6. **Hur verifierar du alarm?**
7. **Är de grafiska interfacen på de IDSer du använt ok?**
    - **Finns all funktionalitet du vill ha där?**
    - **Saknar du någon funktionalitet i dem?**
8. **Hur går din interaktion till med interfacet?**
    - **Klickar du på alarm?**
    - **Tittar du enbart utan att klicka?**
    - **Brukar du söka mer information än vad interfacet presenterar?**

Ser ut som om frågan är hårt kopplad till nätverksbaserade IDSer. Sannolikt behöver man information om vilken enhet som genererar larmet (om man har en korreleringsmotor och flera insamlande enheter, och vill ha möjlighet att följa flöden genom nätet). Sedan är det väl också viktigt att se vilken process som tar emot paketet. För att kunna avgöra vad som finns i paketet behöver man sannolikt också lite applikationslagerinfo.

Wireshark är alltid vettigt att plocka fram, i och med att den har så pass mycket mapp till olika protokoll och liknande.

Följande frågor besvaras utifrån mitt perspektiv som sårbarhetstestare och bör tas med en nypa salt. Dock finns likheter.

Tittar man på min situation med sårbarhetsskanningar tittar jag noga på varje larm. Nu är ju detta inget kontinuerligt jobb utan ett jobb som utförs under en begränsad tid, vilket ger fog för att titta på allt noga.

Om jag hade använt olika verktyg hade jag satsat min sista surt förvärvade slant på ett verktyg som enVision som har möjlighet att plocka in data från nära 300 olika enheter. Sedan hade jag spikat en datainsamlingspolicy (med avseende på att följa compliance-regler och samtidigt ha möjligheter att upptäcka angrepp). Slutligen hade jag kopplat i de olika enheterna och sett till att de samlar in den datan som policyn föreskriver.

När jag ser ett larm tänker jag: Gött! Verktyget hittade en sårbarhet, jag försvarar min orimlig höga timpeng. ;-)

Kategorisering – låg, medel, hög. Det är svårt att ge råd angående huruvida en sårbarhet skall åtgärdas eller inte. Jag använder också graderingsmodeller som CVSS v2.

Verifiering av alarm innebär oftast i mitt fall att man försöker bygga en Poc, dvs bygga en metodik för hur sårbarheten kan utnyttjas, och sedan visa att med metodikens hjälp kan man pålitligt återskapa attacken som utnyttjar sårbarheten.