Telematics
Computer Systems

Freie Universität Berlin

Master Thesis

On Long-Term Honeypot Deployment and Data Analysis for Heterogeneous Network Access Types

Marcin Nawrocki

Marcin.Nawrocki@fu-berlin.de

Enrolment Number: 4373549

Supervisor: Dr. Matthias Wählisch
Second Examiner: Prof. Dr.-Ing. Schiller

Institute of Computer Science, Freie Universität Berlin, Germany

March 9, 2016

I hereby declare to have written this thesis on my own. I have used no other literature and resources than the ones referenced. All text passages that are literal or logical copies from other publications have been marked accordingly. All figures and pictures have been created by me or their sources are referenced accordingly. This thesis has not been submitted in the same or a similar version to any other examination board.

Berlin, March 9, 2016

(Marcin Nawrocki)

Abstract

The study of cyber-attacks is a very active research domain and has gained a lot more of attention with the increasing importance of the Internet. A honeypot is a reactive security concept, which has been deployed effectively in the last 20 years in order to observe and learn about malicious network activities. Although the Internet is a heterogeneous network and the communicating devices are embedded in different network topologies, prior to this day no analysis has been conducted considering the various network access types of the honeypot sensors. Revealing differences between attacks for various network access types would help to avoid bias in this field of research, which might arise from premature generalization of results from one to all access types. Such results would be the first step to the study of access-sensitive attacks. Hence, the primary research question of this thesis investigates whether attacks observed by honeypot sensors are dependent to the network access type. The analysis is based on data collected by low-interaction honeypots deployed in a darknet, a publicly announced university network as well as sensors connected by consumer-grade UMTS and DSL. The origins, targets, frequencies, as well as propagation, evolution and the inflicted risk of the attacks has been analysed. Furthermore, several methods for the detection of attack patterns have been applied, including session reconstruction, port sequences and the Apriori algorithm. The suitability of an analysis conducted on the level of autonomous systems is investigated as there is still a clear lack for it in the honeypot research. Eventually, the similarity of the various sensors is determined by a global and local similarity measure based on the SAX timeseries analysis. In the course of this thesis a review of the related work is performed, which leads to two comprehensive surveys on honeypot deployment and honeypot data analysis. The results show that sensors with different network access types indeed experience different attack behaviours and even context-sensitive attacks exist. The autonomous system based analysis perspective extends the results by additional insights.

Contents

List of Figures

List of Tables

List of Listings

CHAPTER 1

Introduction

1.1 Motivation

Effective network security administration depends to a great extent on the understanding of existing and emerging threats on the Internet. In order to protect information systems and its users it is of crucial importance to collect accurate, concise, high-quality information about malicious activities [1, 2].The fact that cyber attacks are a present threat is confirmed by recent statistics such as the Symantec Internet Security Threat Report 2015 [3]. The number of malware increased by 26 percent and data breaches are also still a significant issue, since the number of breaches increased by 23 percent. The discovery of vulnerabilities such as Heartbleed, ShellShock, and Poodle, and their wide-spread prevalence across a number of operating systems draw the public attention to system security. As observed with Heartbleed, attackers were much faster in exploiting the vulnerabilities than vendors could create and roll out patches. Relying only upon tradition lines of defence such as Intrusion Detection Systems (IDSs) and dynamic firewalls alone does not provide a holistic coverage on detecting novel and emerging patterns of attacks [4]. Honeypots are decoy computer resources whose value lies in being probed, attacked or compromised [5]. They complement the traditional detection mechanisms [6]. Honeypots are able to spot zero-day attacks and give insights on attackers actions and motivation.

The Internet is a heterogeneous network whose communicating devices are embedded in different network topologies. To name a few: Private home devices are usually connected by the DSL (Digital Subscriber Line) and mobile devices such as smartphones utilize UMTS (Universal Mobile Telecommunications System). Servers providing various services reside in publicly announced subnets so that their services are always available, however, machines can also reside in unannounced address spaces, the so called darknet. Although honeypots are used to collect information about threats on the Internet, no publication has considered these heterogeneous topologies and various network access types. Furthermore, no study has inspected the attack origins and attack patterns on the level of autonomous systems (AS), which is an administrative unit of the Internet. A study on honeypot attacks designed around these network properties has yet to be conducted.

Revealing differences between attacks for various network access types would help to avoid bias in this field of research, which might arise from premature generalization of results from one to all access types. Such results would be the first step to the study of access-sensitive attacks. An access-sensitive attack is an attack which is aware of the network context of its target. Hence, the primary research question of this thesis investigates whether attacks observed by honeypot sensors are dependent to their network access type. A positive result would motivate researchers to include the access types information into their honeypot analysis in order to receive more precise results. The AS perspective might aggregate single attack events into an subnet-wide event and so provide a holistic view of attacks in the Internet. Such a view leads to a better understanding of the mutual behaviour of attackers.

Honeypots, being only a general idea, developed from simple scripts and proof of concepts to complex applications which collect data sets allowing a deep data analysis. However, the amount of possible software deployments and various analysis steps might be overwhelming for researchers. To the best of our knowledge there is no comprehensive survey on honeypot software and honeypot data analysis. Before performing our data analysis the related honeypot research will be reviewed and such surveys conducted. The motivation behind these surveys is to create a comprehensive overview and to enable the reader to select the best possible honeypot software for his intended purpose. Moreover, the survey shall guide the researcher in the process of analysing the log data and give structured examples. The proposed analysis structure will also be used as guidance for our analysis on the network access type dependency.

1.2 Research Questions

The research questions can be divided into questions concerning the surveys or the data analysis and are as followed.

[Survey]

- What are honeypots and how do they differ from other security concepts?
- Which types of honeypot software do exist?
- How can honeypot data be analysed in a structured manner?

[Analysis]

- What is a possible software architecture for a honeypot analysis framework?
- Does an AS-based analysis perspective create any new insights?
- **Do attacks on honeypots differ across heterogeneous network access types?**

1.3 Structure of the Thesis

The current Chapter 1 is the introductory chapter which presents the field of studies and motivates the research questions. The rest of the thesis is structured as followed.

Chapter 2 gives a basic overview on the related honeypot research with the focus on required definitions, the taxonomies, distinctions to other security concepts and the legal situation. Based on the proposed taxonomy a survey on honeypot software is created in Chapter 3. Chapter 4 extends the review of the related work by categorizing publications which conducted a honeypot data analysis by leading analysis questions. These analysis questions are sorted in such a way that they can serve as a guideline for a step-by-step analysis. The honeypot study on heterogeneous network access types is performed in Chapter 6 with the help of a self-developed framework presented in Chapter 5 on honeypot log data collected by a honeypot network, which is also outlined in this chapter. A final conclusion and overview as well as an outlook can be found in the last Chapter 7.

CHAPTER 2

Honeypot Background

General background informations on honeypots is introduced in this chapter, which is necessary for a further understanding of this thesis.

2.1 Definition Honeypot

The first academic publications containing explicitly the keyword honeypot date back to the beginning of the year 2000 [7, 8]. One of the first broadly accepted formal definitions was introduced by Spitzner [5]: *A honeypot is decoy computer resource whose value lies in being probed, attacked or compromised.* However, the concept of honeypots is not new and was already used implicitly in the 1990s in the field of information protection and network defences [9, 10], but was named differently, for example a (chroot) jail [11]. A network or accumulation of several honeypots is called honeynet.

At the beginning of the year 2000 an exponential spreading of highly effective Unix- and Windows-based worms took place [12]. Despite the extensive spreading, the capture and analysis was fairly difficult as the worms only resided in the systems memory or masqueraded themselves. One of the first honeypots especially set up for a known threat was directed at the sub7 malware. The honeypot emulated a Windows system infected by the sub7 trojan by responding on the port 27374, which many worms used for subsequent attacks. The SANS Institute [13] captured the W32/Leaves worm within minutes. Hereafter, in the year 2002, a honeypot captured the first unknown threat, more precisely a CDE Subprocess Control Service buffer overflow vulnerability, which allowed to gain access to any Unix-based system [14]. A possible defence was introduced quickly. Ever since honeypots were used as an effective concept to detect the exploitation of vulnerabilities.

The general objectives of honeypots are the distraction of attackers from their actual target or to obtain informations about the attackers and attack patterns, like regularity, popular targets and so forth. However, it is important to note that honeypots should not be understood as a solution to a certain problem but rather as a generic concept. Regardless of what is selected as a honeypot, be it a router, scripts emulating specific services, virtual machines or a standard physical system, honeypots contribute by being exposed to attacks.

2.2 Distinction to other Security Concepts

Typically, firewalls and intrusion detection systems like Snort are used in networks to provide network security against intruders. Therefore it is important to separate honeypots from these concepts [15]. This can be done by dividing security into three areas of operation: prevention, detection and reaction [16]. Prevention can be defined as any undertaking which discourages intruders and inhibits breaches in your systems. As prevention will fail eventually, we need procedures which recognize an attack as fast as possible. However, the sole detection of attacks is not sufficient, we need reactions, which adjust the current system configuration to stop and make future attacks impossible, furthermore administrators have to be informed.

Honeypots add little value directly to prevention: Honeypots do not protect from security breaches. At most, they inhibit attacks because attackers are concerned about wasting time and resources attacking honeypots instead of the actual target: production systems. However, they fail against automated toolkits and worms, which form one of the most common attack types and which scan, probe and try to exploit every network participant they can find. This also holds for intrusion detection systems, they might deter, however they will not actively prevent automated attacks. Firewalls are the best concept for prevention, as they hide services and block communication on unused ports or from suspicious IP-address-ranges, so that intruders are kept out or do not find any services to communicate with. Best prevention is achieved by good security practices, such as disabling unnecessary or insecure services, frequent update and patch policies, strong authentication mechanisms and so on.

Concerning the detection honeypots add extensive value. It is very difficult to detect attacks on production systems because the attacks simply submerge in the vast amount of production activity. Honeypots can simplify the detection process. Since honeypots have no production activity, all connections to the honeypot are suspect by nature and therefore detect an unauthorized probe, scan, or attack with almost no false positives and negatives. As the name suggests, intrusion detection systems are designed to detect attacks. However, their effectiveness is highly dependant on the signature quality. Administrators can be overwhelmed with false positives, become numb and start to ignore those messages. On the other hand, false negatives are also possible if no appropriate signature exists as new exploits are used. Snort just developed after years of community contribution an effective signature-based recognition. If we consider a pure firewall concept without any statistics, packet count etc., a firewall does not provide any functionalities for the detection of attacks.

The reaction to attacks can be accelerated with the help of honeypots. The analysis of the attack is substantially easier, as attack data is not mingled with production activity data. Furthermore, in contrast to production systems, honeypots can be taken completely off-line for the analysis, which enables a proper and full forensic analysis. The insights can then be used to clean the production systems and understand the exploit, which is the first step to patch the corresponding vulnerabilities. Intrusion detection systems are primarily passive, that means that it analyses packets and creates alerts, traditionally, it does not alter the network packet flow. The sole reaction is the information of administrators of malicious activities. The reaction to new attacks might also include the creation of new signatures, but this is rather troublesome as it requires manual input. As firewalls do not detect attacks, the furthermore cannot react dynamically to attacks. However, administrators can define new

	Honeypots	IDS	Firewalls
Prevention	+	+	+++
Detection	++	++	+
Reaction	+++	++	+

Table 2.1: Distinction between security concepts based on areas of operation.

firewall rules manually, which are usually simpler then the rules from an intrusion detection system and are based rather on packet meta information than payload and packet flow [8].

The distinction between the security concepts based on areas of operation show, that all concepts tend to have a focus, however they can be used in all three possible areas. Honeypots are *rather* a research and reaction concept, intrusion detection systems seem to be a good alerting tool and a firewall is the best prevention concept. It should be clear to the reader now, that those concepts do not exclude each other, but complement each other and should be used in conjunction. This fact is visualized by Table 2.1.

2.3 Taxonomy of Honeypots

Marty Roesch, the developer of Snort, established one of the first classifications of honeypots [17], separating honeypots primarily based on their field of operation:

- production honeypots
- research honeypots

Production honeypots are characterized by ease of deployment and utilization and are meant to be used in production environments of companies. Their intended purpose is to achieve a higher security level within the network of one specific company by deflecting attacks. However, the production honeypots come with a trade-off between ease of operation and the quantity of collected information. By contrast, research honeypots provide comprehensive information about attacks, but are more difficult to deploy. They are usually used by research organizations and network forensics scientists in order to scrutinize attacks and to develop general counter-measures against threats. Research honeypots help to understand the motives, behaviour, tools, and organization of the black-hat community.

To set an example: A company defends its systems from a SSH password attack by deflecting the attacks to the honeypot. Their production honeypot merely detects the high load. A research organization however is interested in studying the type of the attack, that is which dictionaries were used to guess to passwords etc. in order to suggest a recommendation on password strength, consequently they have to use a research honeypot, which logs tested SSH login credentials.

Another widely used classification of honeypots is based upon the characteristics of interaction, on the one hand considering the level of interaction [18]:

- low-interaction honeypots (LIHP)
- medium-interaction honeypots (MIHP)
- high-interaction honeypots (HIHP)

On the other hand, considering the direction of the interaction [19]:

- server honeypots
- client honeypots

Low-interaction honeypots simulate only a small set of services like SSH or FTP, they do not provide any access to the operating system to the attacker. As the collection of information is limited, LIHP are used mainly for statistic evaluation. However, they suffice to recognize peaks in the number of requests, for example created by automatic worms. LIHP are often described as passive intrusion detection systems, since they do not modify network traffic in any way and do not produce any replies for the attacker. LIHP tend to be production honeypots.

Medium-interaction honeypots are slightly more sophisticated than LIHP, however they still do not provide any operating system functionality. The simulated services are more elaborated and provide a higher level of interaction for the attacker, as a matter of fact MIHP produce reasonable replies in hope of triggering follow-up attacks. Because of the reduced interaction capabilities LIHP and MIHP both have low chances of being compromised.

High-interaction honeypots are the most sophisticated honeypots. They are the most complex to implement, deploy and maintain, as they provide to the attacker a real operating system environment, which is not restricted. Furthermore, a huge set of services is installed. HIHP collect the largest possible amount of information, including complete attack logs, data access, traversing of file trees, executed byte codes etc. Because of the high complexity, HIHP log analysis is usually done by networks forensics scientists and less frequently automatically. HIHP tend to be research honeypots.

An overview of the properties with respect to the level of interaction is presented by Table 2.2. However, it is important to point out that this classification is quite academic and impractical. Over time many different flavours of honeypots were created which are very difficult to divide into specific categories. That is why it became common to only differentiate between low and high honeypots, as recommended by Lance Spitzner [20]. Eventually, all honeypots that are mere port listeners or emulate services became low interaction honeypots (previously presented as low and medium) and anything that provides real services and aspects of an operating system a high-interaction honeypot [21].

Server Honeypots wait until the attackers initiate the communication, whereas client honeypots actively search for potential malicious entities and request an interaction. The most common field of application are web browsers: client honeypots request a homepage and check for unusual activities. Client-LIHP emulate components while client-HIHP use real web browsers. Traditional honeypots are server based.

The last classification is based on the physicality of the honeypot [22]:

- physical honeypots
- virtual honeypots

	LIHP	MIHP	HIHP
real operating system	✗	✗	✓
risk of compromise	low	mid	high
wish of compromise	✗	✗	✓
information gathering	low	mid	high
knowledge to deploy	low	low	high
knowledge to develop	low	high	high
maintenance time	low	low	very high

Table 2.2: Properties of honeypot interaction levels [23].

Evidently, a physical honeypot is a real machine on the network. A virtual honeypot is simulated (virtualized) by a host machine that forwards the network traffic to the virtual honeypot. Multiple virtual honeypots can be simulated on a single host. Virtual honeypots are usually high-interaction honeypots.

2.4 Specification of Honeypot Attacks

A technological attack is defined as an attempt to destroy, expose, alter, disable, steal or gain unauthorized access to or make unauthorized use of an asset; where asset is defined as anything that has value to an organization [24]. However, honeypots are not directly of value, their value lies in being attacked. In order to define what in particular a honeypot attack is, firstly one has to accept the fact that honeypots are attacked inadvertently, they are exposed to attacks which are originally directed at production systems. That means that attacks on honeypots are no different to other systems as long as the attacker does not recognize the presence of the honeypot. Second, one has to differentiate between *anomalous* and *normal* behaviour. As honeypots are controlled environments, one can tag any anomalous behaviour as an attack [15]. However, this specification requires the differentiation between server and client honeypots. Server honeypots are completely passive, therefore all incoming requests form an anomaly and are by definition an attack. Client honeypots actively search and contact communication partners, therefore client honeypots must discern which communications comprise an anomaly. Heuristics usually verify this by looking after uncommon modifications.

2.5 Advantages and Disadvantages of Honeypots

Short and comprehensive lists presenting the advantages and disadvantages of honeypots have been already assessed [8, 18, 25]. However, for the sake of completeness a short overview is given in this chapter.

The advantages of honeypots include:

Valuable Data Collection Honeypots collect data which is not polluted with noise from production activities and which is usually of high value. This makes data sets smaller

and data analysis less complex.

Independent from Workload Honeypots do only need to process traffic which is directed at them or originates from them. This means that they are independent from the workload which the production systems experience.

Zero-Day-Exploit Detection Honeypot capture everything that is used against them, this means that also unknown strategies and zero-day-exploits will be identified.

Reduced False Positives and Negatives Any activity with server-honeypots is an anomaly, which is by definition an attack. Client-honeypots verify attacks by detecting system state changes. These procedures result in reduced false positives and false negatives.

Flexibility Honeypots are a very flexible concept as can be seen on the basis of the vast amount of different honeypot software. This mechanisms that well-adjusted honeypot tools can be used for specific tasks, which furthermore might reduce redundant load.

The disadvantages of honeypots are as followed:

Limited Field of View Server-honeypots have one common problem: they are worthless if no one attacks them. As long as attackers do not send any packets to the honeypot, it will be unaware of any unauthorized activity on production systems.

Being Fingerprinted Low-interaction honeypots emulate services, that means that their services might behave different than the real services, which can be used for fingerprinting honeypots and consequently detect them.

Risk to the Environment If honeypots get exploited, they can introduce a risk into the user's environment. As discussed, the higher the interaction level, the higher the possible misuse.

2.6 Legal and Ethical Concerns

The deployment of honeypots involves the discussion about legal and ethical liability. It is important to announce beforehand, that the legal situation is country-dependent, furthermore, it is often difficult to decide which laws apply if attacker and victim are situated in different countries. However, this chapter aims to highlight possible pitfalls and general reasoning which has to be considered.

Lance Spitzner [26] formulated the first two problems, which are entrapment and privacy. Entrapment can be defined as the induction and persuasion of an entity to commit a crime although no previous intent to commit such a crime existed. It can be argued that this does not apply to honeypots, as they do not actively persuade anyone. Server honeypots mimic production systems or services and wait for incoming connections, they are virtually invisible to other network participants unless they decide to approach those system, for example due to an IP-address range scan. Client honeypots make the first request, however they do not persuade to commit a crime, as the exploitation is already prepared server-sided, which means that the intent of a crime already existed. On the other hand, a honeypot remains a system consciously set up for attacks and sometimes even with known security vulnerabilities which makes an attack possible in the first place.

The privacy issue elaborates the data collection. The question arises whether a honeypot

is allowed to collect information about the attackers without their knowledge or permission and thus violate their privacy? In order to answer that question one has to consider which type of data is saved by the honeypot and how the data is used by the honeypot systems and administrators.

Data is usually divided into two general categories, meta-information and content. Meta-information stores information about content and connections. For the IP this is operational/transactional data like the IP-address, IP-header, session-cookies, timestamp of the communication and so forth. This type of information is rather collected by low-interaction honeypots, however some content might also be stored depending on the level of emulation. Content data is the data which a sender actually intends to transmit, like email-text, keystrokes and files. High-interaction honeypots focus on saving large amounts of content data, although meta-information is stored additionally. Furthermore, it is important to highlight the purpose of a honeypot and how the meta-information or content is used. As already stated, production honeypots can be used to defend production systems. Although research honeypots do not protect an organization directly, they help to understand threats, develop countermeasures and to fix exploitable bugs, therefore they contribute indirectly to the safety. Common practice in the Internet shows that the processing of transactional data is done frequently and without consequences, especially by advertisement companies which cross-reference visits across several websites. But also individual sites use web analytics services like Google Analytics that track and report website traffic. Content data has more privacy issues then transactional data, as it also underlies the copyright of the author. However, one has to consider that on the one hand the attacker does not have any authorized means to save data on those honeypots (like a legitimate account) and on the other hand does decide voluntarily to transmit the content.

As long as tools attempt to secure one's own systems and their usage-emphasis lies in the improvement of the protection, the legal risks *seem* to be negligible. Even more, if we consider that honeypots commonly communicate with criminal, malicious entities. As far as it is known, there was up to date no trial concerning honeypots explicitly. However, this holds only as long as the honeypot does not cause any harm, which leads us to the next problem statement.

The third concern is the liability in case of harming other systems. As honeypots offer known vulnerabilities to attackers, they can be used to harm other systems. Low-interaction honeypots emulate protocols and therefore they can be subject of a IP spoofing and amplification attack. High-interaction honeypot might allow arbitrary code execution on the machine and therefore cause even much higher damage to other systems. This means, that the original attacker is not visible to the victim, which in turn would sue the operator of honeypots as it is the visible attacker. The argument here is, that if an administrator had taken proper precautions to keep the (honeypot) systems secure, the attacker would not have been possible, therefore the administrator is jointly responsible for any damage which has occurred. The higher the interaction level of the honeypot the risk of a harmful utilization, therefore the higher the interaction the more often verification checks should be performed which check for exploits. It is recommended to reset virtual honeypots as often as possible and to use a reverse firewall to limit the amount of malicious traffic that can leave the honeypot, as it shows consciousness and the attempt to minimize possible damage, which might limit the legal liability. Another possibilities are containment systems, which

restrict the rate at which a computer is allowed to make connections to other machines, for example Dantus feedback control system [27].

Not only the deployment has to be considered, but also the development and publication of honeypot software, as many countries including Germany have a valid *hacker paragraph*, compare the German criminal law code §202c [28]. This paragraph makes the publication of software whose purpose is to spy out or intercept data and facilitating hacking attacks a punishable offence, as it is seen as a preparation of a crime. It is not completely clarified which software is affected by this law. However, a honeypot is collecting similar information, in the same technical manner, as many other security tools like IDS sensors or even system logs, and this category of software is not banned.

2.7 Long-Term Honeypot Projects

This section aims to introduce some of the long-term honeypot projects and alliances or meta-projects, which are responsible for the creation and publication of significant honeypot software, data and several research papers. Rather short projects and honeypot deployments are not enlisted here, but considered in the Chapter 4 which presents metrics used in the field of honeypot data analysis.

The first honeypot project was the **Honeynet Project**, which started off in 1999 and wrote a paper series which has been combined in the well-known book Know Your Enemy [29]. This project focuses on investigating the latest attacks and is developing open source security tools to improve Internet security, recently also at the Google Summer of Code. Its chapters are spread out around the world and use different tools to collect information, based on the requirements client/server and low-/high-interaction honeypots were used. The project is still alive and by far the largest honeypot association, which comprises other smaller (regional) honeypot projects.

One of the very first large-scale worldwide honeypot projects was launched by the Institut Eurocom in 2003 and is called the **Leurre.com** project. Its last publication dates back to 2008 and describes the infrastructure and the main insights. This project is based on a worldwide distributed system of low-interaction honeypots present in more than 30 countries. The main objective is to get a more realistic picture of Internet threats by collecting unbiased quantitative data and create a long-term and location-independent perspective [2].

The Network of Affined Honeypots, also known as the **NoAH-Project**, is a cooperation between Foundation for Research and Technology Hellas and the DFN-CERT. This three-year project gathered and analysed Internet attacks by deploying the high-interaction Honeypot Argos. The aim was to help NRENs (National Research and Education Networks) and ISPs (Internet Service Providers) to limit damage to their networks and to better assess threats. This project started in 2005 [30].

Deriving from the mwcollect development, the **mwcollect Alliance** is a non-profit community aiming primarily at malware collection. This closed community deployed Nepenthes sensors on the Internet. Analysis of the data was performed on the Alliance's server in real-time. This project was mainly active from 2006 to 2009. Members were chosen by a simple email verification, however one of the requirements were frequent contributions

of new samples, that means active Nepenthes sensors. Mwcollect alliance consisted of not evenly distributed sensors across European countries [31, 32].

Telekom-Frühwarnsystem was first presented in 2013, however early cooperations with worldwide partners started in 2012. From the beginning, this project was based on a multi-honeypot platform. Initially, partners had to set up the honeypots by themselves, by this time T-Pot exists, which combines common open-source low-interaction server honeypot tools (dionaea, glastopf, kippo, honeytrap) and reduces the maintenance for partners. Furthermore, the project is now open for uncertified contributors, however this data and its analysis are marked and separated as the community data set [33]. We contributed our data to the Telekom-Frühwarnsystem.

An overview of the various honeypot projects is given by Table 2.3.

Project	Topology	Begin	Duration
Honeynet Project [29]	Multi-Honeypot Platform	1999	ongoing
Leurre.com [34, 2]	HoneyD Sensors	2003	5 years
NoAH-Project [30]	Argos Sensors	2005	3 years
mwcollect Alliance [31]	Nepenthes Sensors	2006	3 years
Telekom-Frühwarnsystem [33]	Multi-Honeypot Platform	2012	ongoing

Table 2.3: Overview of long-term honeypot projects and alliances.

CHAPTER 3

Survey: Honeypot Software

Honeypot surveys and software comparisons have been presented before, however an up-to-date comparison and classification does not exist according to our current knowledge.

First surveys in the field of honeypot research presented in 2003 include only a small subset of meanwhile available software and are by this time outdated [35]. Unfortunately, these surveys tend to only discuss a small subset of honeypot software or mention examples which denies a holistic view [36, 37, 38, 23]. Mairh et al. [15] presented in 2011 a honeypot survey, which illustrates the different types of honeypots and suggests to use honeypots in educational environments. Unfortunately, the deployment of honeypots is only done by the example of *HoneyD* [21]. In the year 2012 Bringer [39] published the by far most exhaustive survey on recent advances in the field of honeypots by ordering a large amount of honeypot related papers by contentual category. However, it rather only considers new types of honeypot software and misses the historical development of honeypot software. Also the software release date and maintenance time spans remain unanswered, which are an indicator for the current deployability.

A comprehensive list including a classification of honeypot software is given in this chapter based on the taxonomy presented in Chapter 2, as we see an existing demand for it. This holds especially true for information about the honeypots design and research focus and its maintenance. The honeypot software is classified by the interaction level and their type of communication architecture. This means, that honeypots will be introduced as either client or server honeypots and rated either as low or high in their interaction level. Hereby, the focus of this overview always lies on available and deployable software, so that the reader will be able to choose the proper honeypot for his needs. That is the reason why the maintenance timestamps are also included in this overview, as some honeypot software had a significant impact for the research and were ground-breaking during their release time, but should not be recommended any more as they have not been updated for a long time. Lastly, this overview shall present how versatile the concept of honeypots is and at how many different application areas it can be found on.

3.1 Low Interaction Server Honeypots

The first publicly available honeypot software was released in 1997 and was called Deception Toolkit(DTK) [40]. It is written using C and Perl scripts and emulates vulnerabilities of well-known services on the Unix environment. The developers state: *We use deception to counter attacks (...) the deception is intended to make it appear to attackers as if the system running DTK has a large number of widely known vulnerabilities.*

In the year 1998 Back Officer Friendly (BOF) [41] was developed, originally to notice attacks by the remote administration tool Back Orifice. BOF has an outstanding ease of use considering the early stages of this research area, is runnable on Unix- and Windows based systems and provides an emulation for several well-known services including Telnet, FTP, SMTP, POP3 and IMAP2. Connection attempts are merely logged and a plain response is created - the main objective of BOF is to waste the time of intruders.

A year later the first commercial honeypot was released: CyberCop Sting [42]. CyberCop Sting creates a virtual network on a single host and simulates different types of network devices, including Windows NT servers, Unix servers and routers. Each virtual network device has a real IP address and can receive as well as send genuine-looking packets. The simulated systems include decoys for Telnet- and Nmap-based fingerprinting.

Google Hack Honeypot (GHH) [43] was introduced in 2005 in order to combat a new type of malicious web traffic: Google hacking, which involves usage of advanced operators in the Google search engine to locate vulnerable versions of web applications and unintentional accessible data. Operators specify properties of the URL, title, body text, filetype etc. GHH was used to learn more about this threat by emulating a vulnerable web application linked by a transparent link, which is hidden from casual page viewers, but is found through the use of a crawler.

Glastopf [44] is a modern, easy to deploy low-interaction web server. The honeypot tool collects information about web application-based attacks like local and remote file inclusion or SQL injections. Furthermore Glastopf downloads files from links in incoming requests and tries to respond in such way that attackers expectations are met and subsequent attacks are initiated. Glastopf promotes itself by implementing a vulnerability type emulation instead of vulnerability emulation. Once a vulnerability type is emulated, unknown attacks of the same type can be handled. While the implementation may be tedious, an advantage against attacks is assumed.

FakeAP [45] was developed as a proof of concept at the Def Con X (2002) hacking conference and created a new category of honeypots: wireless honeypots. FakeAPs key task is to deceive attackers, not to log their actions, as it rapidly generates 802.11b beacon frames with random ESSID, BSSID (MAC) and channel assignments which hides ones own access point from plain sight and confuses wardriving tools like Kismet or NetStumbler.

HoneyBOT [46] is a low-interaction honeypot for Windows, which opens a range of roughly 1000 listening sockets mimicking vulnerable services. The log utility is quite sophisticated, as raw packet level data, keystrokes and malware (trojans, rootkits) are saved for analysis. Furthermore, a report to a centralized collection point via the syslog utility is possible. It is closely tied with a GUI, which offers a classification of log events based on port number or attacker IP-address.

HoneyD [22] is one of the best-known and most influential releases in the field of honeypot research, in particular concerning the virtualization of hosts. HoneyD is a small daemon that simulates thousands of virtual hosts at the same time, class B sized networks has been tested successfully. The hosts can be configured to run arbitrary services, and their personality can be adapted so that they appear to be running certain operating systems. Different operating systems are simulated on the TCP/IP stack level by learning TCP personalities from reading nmap/ xprobe fingerprint files, which results in an effective deception of this common tools. Not only are hosts simulated, but also any arbitrary routing topology including dedicated routes and routers. Moreover, the routes can be attributed with latency and packet loss to make the topology seem more realistic. It is possible to ping or to traceroute all virtual machines.

HOACD [47] appears to be the first live bootable distribution embedding honeypot technologies. HOACD incorporates OpenBSD, Honeyd and Arpd. It is a preconfigured honeypot system that runs directly from a CD and stores its logs and configuration files on a hard disk.

Honeyperl [48] is a honeypot written in completely in Perl. Its prominent feature is the ability of being expendable by Perl modules. Many plugins exist for simulation of telnet-sessions, smtp etc..

HoneyPoint [49] is a honeypot which comes in several versions and with several sub-tools (HPPE, HPSS, HPSC, HPNTA) because of commercial reasons. Its targeted platform is Windows. Honeypot offers fake network services and web applications including manipulated documents allowing tracking down an attacker every time the document is opened after the download. Furthermore emulated devices in ICS/SCADA are supported.

Impost [50] is a network security auditing tool designed to analyse the forensics behind compromised and/or vulnerable daemons. There are two operating modes used by Impost; It can either act as a low-interaction server honeypot and take orders from a Perl script controlling how it communicates with connecting clients; or it can operate as a packet sniffer and monitor incoming data to specified destination ports. It is an early honeypot product which combines honeypots with full packet sniffing functionality.

Jackpot [51] is a Java and HTML based honeypot to combat spam. It comes with several configurable options to make trapping and tracking spam as efficient as possible, including a GUI, automatic spam classification by using various antispam databases, automatic distinction between regular spam or relay test messages and simulation of a very slow server by delaying replies. All spam messages and client IP-addresses are logged. Jackpot fools attackers by delivering relay test messages to drop boxes (which are owned by the attackers) but makes it tedious to send spam and does not forward it, if spam is sent to normal mail boxes (which are owned by the victims).

KFSensor [52] is a Windows based commercial honeypot designed for use in a Windows based corporate environment in order to improve an organization's network security. It includes IDS functionality by having a Snort compatible signature engine and emulates Windows networking protocols and vulnerable system services or trojans. Not only can all ports be monitored but it is also written to resist denial of service and buffer overflow attacks.

Kippo [53] is a SSH honeypot designed to log brute force attacks and the entire shell

interaction performed by the attacker. It provides a fake file system resembling Debian Linux with the ability to add and remove files. Some basic tools like cat are integrated, however they are configured to delude the attacker. Moreover session logs are stored in an UML compatible format, so that they can be replayed with the original timings of the prompt. Kippo development has been continued by Cowrie [54], which has already extended the software by e.g. SFTP/SCP support and additional commands. Originally, Kippo was inspired by Kojoney [55], which is an low-interaction SSH honeypot with considerably less features, however Kojoney is one of the very first dedicated SSH honeypots. Kojoney2 [56] is a real extension of the Kojoney code base, which was refined, expanded, and adjusted in response to observed attacker behaviour after a several year long deployment of Kojoney. Due to the popularity of Kippo it also incorporated many of the most attractive features of Kippo, while still retaining its Kojoney core. All 4 SSH honeypots are written in Python and utilize bash scripts.

LaBrea [57] is the first honeypot incorporating the tarpit techniques, sometimes described as sticky honeypots. It takes over unused IP addresses by replying to unanswered ARP requests on a network and emulates hosts which answer to SYN packets with a SYN/ACK. However, those hosts do not prepare a TCP-connection. Consequently, the attacker sends its ACK and data packets, which will time out after a while wasting the attacks time because of TCP retransmission routines. LaBrea is cross-platform tool.

NetBait [58, 59] is a commercial solution that implements honeypot farms. Interestingly, NetBait is advertised as a product as well as a service. That is because a redirector is deployed, which forwards attacks on unused IP-address ranges either to predefined honeypots within the cooperate network or to a honeypot farm which is maintained by NetBait Inc. outside the network. All the organizations have to deploy is a redirector on their networks, no skilled network administrators and security specialists are required, which saves costs. Moreover, NetBait is described as a distributed query processing system over honeypot data as collected by a set of cooperating machines. NetBait supports up to 15000 hosts per network. The company remained active with their service from 2002 up to 2007.

NetFacade [17] is one of the very first commercial low-interaction server honeypots, which simulates an entire class C network with up to seven different operating systems. Its development started in 1998 and was released in 2000. Unfortunately, it has seen little public exposure. However its developer, Marty Roesch, gained valuable knowledge and developed debug tools which led to the development of Snort.

SpamD [60] by OpenBSD is an email honeypot and considers sending hosts to be of three types: blacklisted hosts are forwarded to SpamD and tar-pitted, that means that the communication is biased with delays of 1 character per second during the whole dialogue to waste attackers time; emails are rejected eventually with an error message. A blacklisted host will never be able to talk to a real mail server as opposed to white-listed hosts, whose connections are instead sent to a real mail server. New hosts are grey-listed by default and forwarded to SpamD, which shows a temporary failure message when they try to deliver mail. Additionally, grey-listed hosts experience stutter during the first seconds of the dialogue. A real mail server prioritizing quality of service will try to retransmit for a period of time, however a spammer being paid for emails per minute will lose interest. SpamD is a lot more efficient than simple DNS lookups or spam blacklist checks.

Sandtrap [61] is a historic honeypot, which addresses the problem of war dialing, a technique

of using a modem to automatically scan a list of telephone numbers in search for computer systems. It can log incoming calls on up to 16 lines and emulate reachable modems by answering with login prompts. The caller ID (phone number) and any login attempts are logged, moreover an alert systems warns administrators of suspicious activity in real time.

Another historic STMP honeypot, which does not have an official homepage any more and was not maintained for many years, is ProxyPot [61], which imitates an open proxy mail relay and is designed to solely log spam and record the senders identity.

A simplistic honeypot server written in Perl is single-honeypot [62], which was designed using the KISS-principle. It logs access on all ports without emulating services.

Smoke Detector [35] by Palisade Systems Inc. is another early commercial product. It distinguishes itself by not only providing a software but also offering a hardware unit, which has Smoke Detector installed and preconfigured. Smoke Detector mimics up to 22 services and emulates 19 distinct hosts in a network. Access attempts are reported and some complementary tools for analysing logs are also advertised.

Another simple SMTP honeypot is SMTPot [63], which is written in Python in less than 300 lines of code. Being a simple program it pretends to be an open mail relay and accumulates mail to mailbox files.

Spamhole [64] is another fake open mail relay, happily accepting any email messages that the client wishes to send to it, however rather than actually delivering the messages, it will silently drop them.

Spampot [65] is a Jackpot clone, however written in Python as its author aimed at a higher support of different platforms. Spampot does not use any heuristics to rate mails, it simply stores 5% of the incoming spam.

Specter [66] is one of the longest available commercial products, being purchasable for 16 years, however no new major release has been seen since 2005. It is advertised as a smart honeypot-based intrusion detection system and offers common Internet services such as SMTP, FTP, POP3, HTTP and TELNET which appear normal to the attackers, moreover supports the analysis of ICMP, TCP and UDP packets on all ports. It logs malicious activity and informs the administrator automatically. Specter provides massive amounts of decoy content including images, MP3 files, email messages, password files, documents and all kinds of software.

Also Symantec provided an extension to his anti-malware products by selling the Symantec Decoy Server [67], which acts as a fully functioning server, and can simulate email traffic between users in the organization to mirror the appearance of a live mail server. The honeypot records every action for further analysis.

Tiny Honeypot (thp) [68] is a simple server honeypot program which listens on every TCP port not currently in legitimate use, logging all activity and providing enough replies and interaction to fool most automated attack tools with a fingerprint as small as possible. Netfilter / iptables rules are used to redirect any incoming connection requests to the thp listener. By default a login banner and root shell including some simple emulation of tools like wget are presented. Thp can reside on production hosts with negligible impact on performance and no side effects on real services. The goal is to distract the attackers from real services by a large amount of open ports with fake services.

Nepenthes [69, 70] is a low interaction honeypot, which is designed to emulate vulnerabilities worms use to spread and to capture these worms. Nepenthes provides a modular architecture which can be easily extended with new vulnerabilities. Other modules include functions like the download of files, submitting of the downloaded files and a shellcode handler. Although Nepenthes needs knowledge to emulate vulnerabilities and to conduct a successful conversation with malware, it captures new exploits for old vulnerabilities. Unknown exploits are highlighted in the log files, an information which can be used to build new modules / better dialogues to trigger more downloads.

Dionaea [71] is meant to be a Nepenthes successor overcoming some of its shortcomings like IPv6 support, multi-threading, TLS encryption for some protocols and switching from C++ to Python as the module scripting language in hope of more contributors. The shellcode detection was extended by developing libemu [72], which detects shellcode not only by simple pattern matching but by emulation. The libemu library receives a buffer and detects even unknown shellcode fully automatically. Moreover, Nepenthes never supported the port 445 (SMB) because of too many different exploitable vulnerabilities, Dionaea addresses this problem by emulating valid SMB sessions. Despite all the improvements, one have to point out that Dionaea is not a further development of the Nepenthes code base, but rather the same developers started over using and extending the Nepenthes architecture.

Honeytrap [73] is a low-interaction honeypot daemon which distinguishes itself from other honeypots by implementing a dynamic server concept. It uses stream monitors to check the network stream for incoming packets and starts appropriate listeners just on demand. Each listener can handle multiple connections and terminates itself after some idle time. This concept targets completely unknown attacks which might occur on random ports and unknown protocols - no predefined emulation is required. Service emulation is not in the main focus, however some basic emulation is provided. If no emulation for a protocol exists, the default response is a single newline character, which, according to the developer, is a simple but surprisingly successful approach. Honeytrap differentiates strictly between data capture, which is done by the core system, and attack analysis, which is done by plugins. Honeytrap supports several modes, including a mirror mode, which replies with exactly the same packets as received, hence emulating services implicitly. If the connection fails, the default replies are used. Furthermore, Honeytrap can be used in proxy mode (also called meta-honeypot), which relays incoming connections to other hosts or services while still logging the communication. If ports are configured in the ignore mode, honeytrap simply does not handle those ports.

HoneySink [74] is a honeypot which specializes on sinkholing, which allows security researchers to monitor the communication within a botnet and to prevent interaction between bots and their command and control servers. HoneySink allows its user to sinkhole any number of domains to it and configure the emulation of DNS, HTTP, FTP, IRC on a basis of protocol-domain combinations. The authors envision two main use cases: On the one hand, deployment in internal networks where self-maintained DNS servers redirect traffic to known blacklisted URLs to HoneySink. This helps to detect infected machines within your own networks. On the other hand HoneySink can be configured globally and respond by its own DNS functionality to requests for domains which have been taken over by law enforcement. This prevents criminals to maintain control over their bots. HoneySink is the first freely distributed network sinkhole software, which tries to be a generic framework for

various botnets.

Mwcollect [75], later known as Mwcollectd [76], is designed as a versatile malware collection daemon, attempting to unite the best features of Nepenthes and Honeytrap. Mwcollect was actively developed during 2005-2006 and received a major update in 2009 after a long interruption. Its entire functionality is spread across different modules, which include libcurl for http and ftp downloads, TFTP, dynserv-nfqueue module which implements the Honeytrap principle of package mirroring and dummy newline replies, as well as some Dionaea bindings, http/SMB emulation, shellcode emulation via Libemu and mwserv, which is the malware aggregation/submission service used by the mwcollect Alliance.

Another possible deployment place for honeypots are SCADA or ICS systems, which are systems that monitor and control industrial processes in the physical world, often such systems are part of critical infrastructures and therefore particularly endangered. Conpots [77] goal is to collect intelligence about the motives and methods of adversaries targeting industrial control systems. It is written in Python and speaks several common protocols such as HTTP, but also some ICS-specific protocols like kamstrup, BACnet and mosbus.

Elasticsearch is a search server based on *Lucene* and provides a distributed full-text search engine with an HTTP web interface and schema-free JSON documents. It became the most popular enterprise search engine which led to new attacks for that service. That is why elastichoney [78] has been developed. It is a simple elasticsearch honeypot designed to catch attackers exploiting remote code execution vulnerabilities in the elasticsearch service. It is distributed as a ready-to-deploy Docker file.

The Internet of Things (IoT) is the network of physical devices, which are embedded with electronics and network connectivity, that enables these objects to collect and exchange data over the network, most frequently the Internet. As those devices can be queried and controlled remotely, attacks on those devices emerged. IoTPOT [79] was used by a research team to analyze the increasing threats. It analyzes Telnet-based attacks against various IoT devices running on different CPU architectures such as ARM, MIPS, and PPC. Another proof of concept in this field is honeypot-camera [80], which emulates an openly accessible webcam, including some fake images and device specific deceptions such as watermarks and daylight intensity.

The honeypot Shockpot [81] is designed around a single, critical vulnerability called Shellshock/ CVE-2014-6271 [82] for the bash shell. As the vulnerability was very far-reaching, the study of its exploitation became interesting. The pure python implementation emulates an Apache-server, that appears to process trailing strings after function definitions in the values of environment variables. This allowed remote attackers to execute arbitrary code via a crafted environment in the original vulnerability.

The bluetooth honeypot bluepot [83] was developed to capture attacks on bluetooth devices. It is designed designed to accept and store any malware sent to it and to interact with common bluetooth attacks such as *Blue Bugging* and *Blue Snarfing*. This honeypot does not require any specific device, it is written in Java and runs on any Linux machine, but it does obviously demand from the user to possess at least 1 active bluetooth interface. This honeypot has also some graphical dashboard which allows monitoring of attacks and presents some simple graphs and lists.

As the telephone-infrastructure is shifted towards the VoIP, this service is increasingly under

attack, which is why recent honeypot analysis concentrate around VOIP [84]. Artemisa [85] is a pure VoIP SIP-specific honeypot implementation. It registers multiple SIP accounts, which do not represent real human subscribers, at one or more VoIP service providers, and wait for incoming attacks. It includes conversation recording, a protection against message flooding, correlation rules to infer sequential and stateful attacks as well as rule-based fingerprinting of known SIP attack tools.

HoneyDroid [86] is a prototype of a honeypot especially designed for mobile Android devices. For this purpose the Android smart-phone has to be rooted and extended by Galoula, which makes common UNIX (BusyBox) services and file systems available on Android. This step makes the installation of Kippo and Honeytrap possible. Kippo has been adjusted to resemble the behaviour of the Android OS. Furthermore, an Android-App is included, which reads and visualizes the different log file results and transmits them eventually to a centralized data collection point.

3.2 High Interaction Server Honeypots

Argos [87] was released in 2006 and as it is based on Qemu, it focuses on efficient x86 emulation. A high-interaction honeypot environment is provided, which aims to automatically identify and produce remedies for zero-day exploits. Upon attack detection an intelligent process- or kernel-aware logging is executed, furthermore own forensics shellcodes are injected allowing in-depth analysis. Argos distinguishes itself by applying memory tainting and creating signatures for intrusion detection systems with very few false-positives. There is also a support for running as a client honeypot (using the same infrastructure as the Shelia client honeypot), however this mode became neglected.

Honeywall [88] aims at making honeypot deployments simple and effective. This CentOS-based live-CD utilizes Sebek as the dedicated honeypot software and offers a GUI for system configuration, administration, and data analysis. It features an architecture that allows you to deploy both low-interaction and high-interaction honeypots, but is designed primarily for high-interaction.

High Interaction Honeypot Analysis Tool (HIHAT) [89, 90] transforms arbitrary PHP applications into web-based high interaction honeypots in an automated fashion. HIHAT has been compatible with 4 major PHP frameworks during its active development time. Furthermore HIHAT offers a graphical user interface which enables honeypot monitoring and analysing the acquired data. Extensive statistics are generated and IP-addresses are mapped to a geographic locations. Automatic malware is attracted by insertion of transparent links. Attack types, which has been spotted with HIHAT include command injection, file inclusion and bot self-propagation and are characterized by the HTTP GET request, the four different arrays provided by PHP, and the data transferred. The tool automatically filters for attack patterns via regular expression which were derived from an analysis of known attacks against web applications.

HoneyBow Sensor [91] is based on VMWare images and consists of the following 3 components: The MwWatcher malware collection tool, which monitors file system changes in real time and catches potential malware on a Win32 guest system. MwFetcher malware collection tool, which extracts potential malware from a VMware virtual disk by comparing

the infected file list with the clean file list. Finally, the MwSubmitter malware submit tool, which submits potential malware samples collected by MwWatcher and MwFetcher to the mwcollect Alliance. The host tools are Linux based. Infected guests can be replaced with clean guests automatically after malware extraction.

Sebek [92, 93] is a kernel module installed on high interaction honeypots, usually virtual guest machines, for the purpose of capturing attacker's activities such as as keystrokes or file uploads on Win32 and Linux systems. It works by monitoring system call activity and recording data of interest and is based on two components: The client, which runs on the honeypots, captures activities and sends the data to the second component, the server, as stealthy as possible. The client masquerades its existence by using early root-kit techniques. The server is the centralized data collection point for all honeypots within the network.

3.3 Low Interaction Client Honeypots

PhoneyC [94, 95] is a honeypot which mimics legitimate web browsers and enables the study of malicious HTTP pages. It features the interpretation of remote links (hrefs, iframes etc.) and scripting languages via spidermonkey. Moreover, specific ActiveX and PDF vulnerabilities are emulated and heap spray and shellcode detectors are included. By using dynamic program analysis, PhoneyC removes obfuscation from many malicious pages. Different browser identities are supported. The data flow of PhoneyC is as followed: One or more URLs are passed to the client, which retrieves the content. The content is scanned by an anti-virus program if it is a file. Valid HTML files are broken up by script code languages and interpreted by the specific engines. Alerts are risen on suspicious behaviour.

HoneyC [96, 19] is a honeypot which identifies malicious servers on the web by using different visitor clients, search schemes, and analysis algorithms. It consists of three components: the visitor, the queuer, and the analysis engine. The visitor interacts with the potentially malicious servers, makes the requests and processes the response. The queuer creates a queue of servers to visit. Self-evidently, the analysis engine is the component responsible to evaluate whether security policies have been violated by deploying snort rule matching. All modules use the pipe to communicate with each other.

Thug [97] is a pure Python honeypot and build upon the experiences which were gathered by the development of PhoneyC. It aims at mimicking the behaviour of a web browser on a certain operating system in order to detect malicious contents. Thug uses the Google V8 Javascript engine and an own DOM tree implementation. This framework performs static syntax tree and dynamic code analysis. Vulnerable modules are emulated like ActiveX, Flash, Adobe Reader. Different personalities exist, which include various Windows versions, Linux, iOS, MacOS and Android but also multiple browsers like Safari, Chrome or Internet Explorer. Proxies are supported, which can be used to anonymize the access to a malicious pages with tools such as TOR.

YALIH [98, 99] (Yet Another Low Interaction Honeyclient) is designed to detect malicious websites by integrating a combination of multiple antivirus engines and pattern matching using string or regular expressions for detection. It is capable of extracting embedded JavaScript files and performs de-obfuscation and de-minification of scripts. YALIH has an IMAP plugin in, which scans an email inbox for spam, extracts the URLs and visit

these sites. Its emulated browser handles cookies and session, redirection, referrers and different browser personalties. The developers compared its effectiveness using a malware URL database and determined 15% less false negatives than Thug, 80% than HoneyC and 35% than Monkey-Spider while still requiring only a moderate scanning time.

SpyBye [100] is developed by the same author as HoneyD, however it is a low interaction client honeypot. Originally, one had to enter an URL into a form and wait for the analysis to complete, similar to other honeypot clients. However, SpyBye matured into an HTTP proxy, which intercepts all browser requests while the user just simply visits a homepage. SpyBye intends to determine if embedded links on your web page are harmless, unknown or even dangerous and scan the content against the ClamAV engine. It is a reliable indication for a compromise if administrators scan their homepages and detect foreign URLs which were not set by them.

Monkey-Spider [36, 101] is a crawler based honeypot utilizing anti-virus solutions to detect malware. It is claimed to be fast and expandable with other detection mechanisms. It uses the well-known scalable Heritrix crawler to create arc files and then pass them along to anti-virus scanners such as ClamAV and Avast. Monkey-Spider detection is done solely by external signatures. URLs, binaries and malicious JavaScript are extracted and saved in a threats database.

ADSandbox [102] is a honeypot which utilises at its core the Mozilla JavaScript engine SpiderMonkey within a sandbox and logs every action during the execution. After that a heuristic assesses malicious behaviour. The heuristic performs static and dynamic analysis. The user interacts with ADSandbox in two different methods: First, by means of the browser helper objects, which hands over the URLs visited within the browser to the analysis engine and displays custom error pages if necessary or in case of no suspicion lets the user visit the page transparently. Second, the user can manually supply an URL and some additional parameters using the shell. ADSandbox is meant primarily to provide real-time protection for browser users, however it never left the prototype status.

3.4 High Interaction Client Honeypots

Capture Bat [103] monitors the state of the Win32 operating systems during the execution of applications and processing of documents. The client connects to a central capture server that requests the client to visit an URL with a specific browser. The client is run inside of a virtual machine so that infections can be reverted by resetting the virtual machine. Capture Bat provides insights on how the software operates even if no source code is available by observing state changes on a low kernel level (by the help of a file system, registry and process monitor). Event noise that naturally occurs in an operating system environment while idling or on standard execution of applications is filtered, allowing analysts to easier spot and understand the behaviour of for example malicious Microsoft Word documents.

HoneyClient [104, 105] is the first open source client honeypot. It is state-based and detects attacks on VMWare Windows clients running on Linux hosts by monitoring files, process events, and registry entries. Its architecture is threefold: An Agent, Manager and Util Module exist. The agent component is a SOAP server, running as a daemon within a cygwin environment on the guest. It receives messages which trigger actions like visiting a homepage

with a specific web browser (Internet explorer, firefox are supported) or performing an integrity check of the Windows files. The manager module handles the guests on the host system by communicating with the agents. The Util package provides the SOAP/http protocol integration and a configuration possibility.

Capture-HPC [106] is a honeypot framework designed with efficiency and scalability in mind. Capture-HPC consists of a Capture Server and Capture Client. The Capture server is a simple server that manages various capture clients and the VMware servers, which host the guest OS that run the Capture clients. It distributes each URL it receives to the active clients. The Capture client consists of 2 modules: a kernel driver which uses event-based detection mechanisms for monitoring the system's state changes (file system, registry, and processes that are running) and an user space process, which accepts action requests from the Capture server, and communicates potential state changes back to the server. An exclusion list is used to filter default events.

Strider HoneyMonkey [107, 108] was developed by the Microsoft Research Team and distinguishes itself by a narrow coupling with the Windows OS and by creating heterogeneous virtual hosts, which differ by their patch level. The technology was never provided publicly and was only used for internal purposes. HoneyMonkey utilizes the Internet Explorer as the Browser, however does not allow any pop-ups or installation of plugins or software. Any read or write which happens outside of Internet Explores file directories are considered malicious and highlighted for manual analysis. Furthermore, new spawned child processes of the Internet Explorer are observed. Upon detecting an exploit, the monkey notifies the Monkey Controller on the host machine respawn a clean HoneyMonkey virtual machine, which then continues to visit the remaining URLs.

Trigona [109] tries to improve the efficiency of high interaction honeypot clients by combining the advantage of high interaction where no emulation is used and low interaction, where a high throughput and a small resource fingerprint is possible. As opposed to other frameworks Trigona does not load several VMs or resets one VM repeatedly for each URL, but initiates one VM, which access multiple, for example 200, URLs at once. The network traffic is packet captured for analysis at a later stage. The VM is reverted after a group of URLs. Exploit kits and malware binaries can then be extracted from the packet capture files, however the state information of the operating system is not analysed.

HoneySpider [110] is hybrid honeypot framework, which is based on fairly up-to-date high and low interaction honeypot tools and integrates a crawler application specially designed for the bulk processing of URLs. The framework focuses primarily on attacks involving the use of web browsers and their plug-ins by detecting drive-by downloads and malicious binaries including zero-day exploits. HoneySpider automatically obtains and analyses the malware. The first version included a simple web-client, JavaScript analysis and Capture-HPC. It evolved and possesses now a shellcode and anti-virus scanner; PDF, SWF and office documents analyser, moreover some bindings for Thug, Cuckoo and couchDB.

SHELIA [111] focuses on the analysis of URLs and attachments received in emails. It supports IMAP/POP, so it can read messages from an email in-box, however direct input of links and files is also possible. SHELIAs design philosophy is that false positives are much more important than false negatives because they are possibly more harmful in cases of signature creation, therefore they are avoided. This is done by detecting intrusions not by verifying modifications to the file system or registry after visiting a website, but

by tracking the caller of sensitive operations. More precisely, whenever a call modifies the registry, the file system, or network activity, Shelia tracks whether the call is coming from an unauthorized area which is not supposed to contain code. Shelia runs in a virtual machine which is reset every n checks to prevent infections to survive if they are not detected by Shelia. Upon exploit detection extensive analysis and logging to a database takes place.

UW-Spycrawler [112] was used for research projects, however never made publicly available as the authors focussed on Internet research rather than developing a general user-friendly framework. It focusses on the detection of malware using one of the two attack methods: Piggy-backed malware code, which comes with legitimately looking executables or drive-by download attacks, which exploit a vulnerability in the browser to install malware. UW-Spycrawler supports automatic software installation by automatically accepting the EULA and other installation steps. Piggy-backed spyware is then recognized by using a signature-based anti-spyware program (AdAware). Drive-by attacks are assumed to escape the browser-sandbox and modify the system, so URLs are marked as malicious if one of the event triggers is detected, like file and registry writes, process creation, browser crashes.

Web Exploit Finder (WEF) [113] is designed to detect drive-by-download attacks and consists of a VMware virtualization layer, specially crafted Windows guests and an user dashboard which communicates with the guests and controls the action of the web browsers. Attacks are detected by checking for modifications to the operating system by evaluating the relevant system calls. As such changes to the operating system are not designated, a deep integration and interaction with the kernel was deployed by the so-called rootkit module.

HoneyIM [114] utilises Capture-HPC in a new context. The open-source instant messaging (IM) client Pidgin is used to extend the basic functionalities and to create decoy IM users. A legitimate IM user has to add the decoy IM user to its contact lists, if some IM malware compromises a user and tries to propagate, it will contact all users in his contact list, therefore the decoy will also receive malicious URLs etc. and thus will be registered by the HoneyIM system. Furthermore, the malware can then be analysed in the Capture-HPC environment. This helps to recognize infections in IM networks early, as HoneyIM delivers attack information to network administrators in real-time so that system quarantine and recovery can be quickly performed.

PwnyPot [115] is a honeypot, which does not detect malicious servers based on system changes but tries to identify the malware already during the exploitation stage, that means before any changes to the system or infections have occurred. This approach makes recognition of zero-day exploits easier as no signatures are required. Supported programs include among others Internet Explorer, Firefox, Office Products, Adobe Acrobat Reader and Flash. The main features of PwnyPot are general protections like heap spray mitigation and null page allocation prevention, return oriented programming (ROP) detection and ROP gadget dumps, moreover detection of possible DEP and ASLR bypasses. Shellcode is recognized and analysed dynamically. PwnyPot offers bindings for Cuckoo, one of the leading open source automated malware analysis systems, which can perform an automatic analysis.

3.5 Honeypot Related Tools

The following tools extend the functionality of honeypots or are meant to be used simultaneously with honeypots, for example by making managing tasks easier or detection executables automatically.

Bait-n-Switch [116] was not a honeypot technology as such, however it was one of the first attempts to multiplex hostile and regular traffic between production systems and honeypots. It is a system which reacts to malicious intrusion attempts by redirecting all hostile traffic to a honeypot. Bait-n-Switch is realized by as a Snort extension, based on linux' iproute2 and netfilter. The honeypot software can be chosen arbitrarily. The same developers also developed a low-interaction honeypot called BigEye [35], which only emulated FTP and HTTP services.

Honeynet Security Console (HSC) [117] is an analysis tool to view events on your personal honeynet. It focuses on visualization and grouping of of events from Snort, TCPDump, Firewall, Syslog and Sebek log files. Moreover, it correlates information between those different log file types, so that analysis can be done with a more holistic approach.

GSOC-Honeyweb [118] manages client honeypots via a user-friendly web interface. This application is threefold: The front-end, providing a standardized interface for various client honeypots; a business layer, communicating with a Java wrapper and a back-end, providing the data persistence to collect, store and aggregate client honeypot results. GSOC-Honeyweb should not be confused with HoneyWeb by Kevin Timm [35, 119], which is a HTTP-only low-interaction honeypot compatible with HoneyD. HoneyWeb emulates various web server platforms, including Apache, Netscape, and Microsoft IIS. HoneyWeb looks at incoming URL requests, determines which platform they suit and finally deceives intruders by emulating specific HTTP headers. A mode is possible, which creates an affiliation of a web server platform and an attacker for a certain time frame, so if the same attacker makes a UNIX-style request which is then followed by a Windows-style request, he will see an error page. HoneyWeb is extensible by SSL.

Honeysnap [120] is a diagnostic tool which can be used to perform a number of diagnostics on data which was collected by a server honeypot. The primary intention is to provide an analysis on a directory full of pcap data. It decodes and analyses a variety of protocols supporting: outgoing packet counts and binary extraction for telnet, ssh, http, https, ftp, smtp, and IRC; incoming and outgoing connection summaries; word based inspection of IRC traffic for basic keyword profiling. In addition, it focuses on honeypot specific data sets such as Sebek keystroke data.

PE Hunter [121] is a plugin for snort for extracting Windows portable executables from the network stream and is meant to be used in front of honeypots, which trigger the transfer of the executables. It works by spotting a PE header, using a simple heuristik to calculate the file length and finally dumping the corresponding bytes to a file.

HoneyMole [122] incorporates Capture-HPC and supports administrators to deploy and distribute sensors worldwide which tunnel traffic in a transparent way to a centralized farm of honeypots. Sensors can be understood as simple, encrypted ethernet bridges over TCP/IP. The idea here is that sensors require minimal maintenance efforts, which saves time for administrators. Moreover, data about attacks is collected in one point, which saves

time for analysts.

TraCINg [123] can be described as a cyber incident monitor, which can receive data from arbitrary honeypots as long as it is well-structured with JSON. Currently, only a Dionaea-plugin exists. TraCINg collects data from several honeypot sensors and tries to correlate attacks in order to find emerging worm outbreaks. It considers mutual attack sources as well as timing properties in its analysis.

3.6 Honeypot Detection Tools

Not only do tools exist which extend the functionality but rather are adversaries of honeypots: honeypot detection tools. This class of tools is able to detect low- as well as high-interaction honeypots.

Low-interaction honeypots are detectable because of the service emulation, which will never be able to behave like the real service because of the nature of emulation and security concerns. This means that specific actions trigger different responses, as has been shown for example for Kippo and OpenSSH [124]. Specially crafted messages trigger a characteristic response, which often contains a specific string or number, often called *magic numbers*. This magic number identifies Kippo. Nmap also detects some of the Dionaea services as being part of the honeypot and string obfuscation is necessary to overcome the signature-based detection [123].

High-interaction honeypots use real services in constraint environments, therefore fingerprinting them is based on detection of unusual additional libraries or debuggers and characteristics of virtualization software [125]. Holz [126] presented several techniques, for example VMware uses only a specific range of MAC- addresses for its virtual network interfaces, chroot and jail environments are fingerprintable by special *ls* calls and even the presence of debuggers like the presence of *ptrace* can be detected by simple system calls. It has been shown with the help of a timing analysis, that honeypots, especially those running in virtualized environments, respond slower than real services [127]. This fact could also be used for detection, however has to be used with caution as the response time is also highly dependant on other factors like network load, routing etc.. Zou [128] and Wang [129] demonstrated that botnets can be designed to be aware of honeypots. Their work is based on the following assumption: honeypots must not participate in real (or too many real) attacks because of legal constraints. Attackers can detect honeypots in their botnet by verifying whether the compromised machines can successfully send out unmodified malicious traffic to attackers' sensors. Sebek was detectable by the relative address space positions of the *write*() and *read* calls, as the integration of Sebek positions these two farther away from each other [130]. This approach of verifying address space positions is used even today but for different elements, for example the interrupt descriptor table register (IDTR) [131].

3.7 Overview Honeypot Software

An overview of the various honeypot software, its classification and publication details are shown in Table 3.1 and Table 3.2. One of the first findings is that different honeypots exist which are applied to different protocols. This proves the universality of the concept

of honeypots. Another finding is that certain honeypot software overlap in their field of operation. In this cases, the quality and maintenance life time of the honeypot influence the success.

Despite many different honeypots and related tools, the general trend is pretty clear: Simple proof of concepts developed into complex honeypot tools which are designed to be deployed for a long time. As the intended analysis are getting more complex, management (e.g. GSOC-Honeyweb, Vagrant) and analysis tools (TraCINg) emerge and even combinations with intrusion and malware detection tools are considered (compare Section 2.2, Section 3.5).

Table 3.1 clarifies that the first honeypot and also the majority of available honeypot software are low-interaction server honeypots. The reason for this observations might be the fact, that on the one hand server honeypots require less implementation effort than client honeypots as they do not have to have a sophisticated crawler engine and on the other hand the emulation of services requires less maintenance effort than providing real services on high-interaction honeypots. This circumstance led to the situation, that many developer initially released small proof of concept honeypots, which should just introduce the concept. Hence, such honeypots (e.g. BOF, single-honeypot) had a rather short maintenance life time. After the approval of the honeypots effectiveness, reliable deployment was necessary, which led to more complex honeypots which were longer maintained (e.g. HoneyD, SpamD). The nowadays broadly applied state-of-the-art honeypots (Kippo, Nepenthes-Dionaea, Honeytrap) distinguish themselves with long maintenance life times. Such life times decrease deployment efforts for administrators and foster the development of community based plugins. However, it is difficult to determine if the the long development life time is responsible for success of those tools or vice versa. Another general trend is also the focus on the service, low-interaction server honeypot tend to either specialize in the emulation of on one or a few well-known services, or simply perform default answers on all ports. It is also noticeable, that high-interaction server honeypots were developed later and only few exist - Sebek and Argos has been here predominant. Although being a mere data capture tool designed to capture attacker's activities, Sebek has to be highlighted for its pioneering influence. However, only Argos has received recent updates for newer version of Windows. One of the reasons of Argos success might be its modern memory taint analysis, which is is very good in the detection of recent attack types (compare Section 4.9).

As shown by Table 3.2, client honeypots appeared almost 5 years later than server honeypots. Client honeypots mimic the behaviour of users in order to rate the risk imposed by the Internet, therefore the list of emulated and examined software consists of common www-technologies such as browsers and their plugins, Flash or PDF viewers (Browsers are the primary user interfaces to the World Wide Web and because of that arguably the most frequently used program by the common user). As this complex browser environments are more difficult to emulate, we see the ratio of high and low honeypots shift towards high-interaction honeypots. Client-honeypot rather implement less services than server-honeypots, however this is completely legitimate, as client honeypots actively influence which application are used / are required. As client honeypots do not improve the security of production networks directly, low-interaction client honeypots usually are published in the context of Internet-wide research of several years (HoneyC, Monkey-Spider, PhoneyC etc). This is why this type of software has relatively long maintenance time com-

pared to server honeypots. Besides the complete emulation of services the current trend in low-interaction client honeypots is the performance, so that a larger and faster view of the Internet is possible (Thug, YALIH). Another difference to server honeypots can be observed: For client honeypots low as well as high interaction honeypots appeared at the same time. Furthermore, high-interaction client and server honeypots appeared almost at the same time. HoneyClient is the first high-interaction client honeypot and stands out with his long maintenance. However, it should not be deployed any more as the only currently maintained honeypot (even with an highly scalable associated project network) is HoneySpider.

This overview is limited to the classification, maintenance time and the focus on services, software architecture and its application area. Although this properties are already enough to assess the deployability to a specific scope, future work might consider more quality measures such as robustness, quality of collected data and its ease of analysis, actual containment and detection precision. However, an elaborated long-term deployment test of each honeypot would be necessary, hence it is recommended to narrow down the choice by the presented overviews.

Type	Software	Publication			Focus	
		First	Last	Free	Services / Applications	Design / Details
low	DTK [40]	1997	1999	✓	SMB, SSH, DNS, FTP, Netstat(++)	implement many known vulnerabilities
	BOF [41]	1998	1999	✓	Back Orifice, Telnet, SMTP(+)	waste intruders time, easy deployment
	NetFacade [17]	1998	2002*	✗	*not specified*	class C network emulation
	CyberCop String [42]	1999	1999	✗	Telnet, FTP, SendMail, SNMP	emulating different network devices
	Specter [66]	1999	2005	✓	SMTP, FTP, HTTP and Telnet(+)	commercial deployment, decoy files
	Sandtrap [61]	2002*	2002*	✗	dialup modem	war dialing trapping
	single-honeypot [62]	2002	2002	✓	*all ports, but no emulation*	mere logging, KISS architecture
	HoneyWeb [119]	2002	2003	✓	HTTP	various web server header emulation
	LaBrea [57]	2002	2003	✓	*all ports, but no emulation*	simple TCP tarpit by SYN/ACK
	SMTPot [63]	2002	2003	✓	SMTP	spam accumulation, KISS
	THP [68]	2002	2003	✓	SSH (shell), HTTP, FTP	coexistence honeypot and real services
	Jackpot [51]	2002	2004	✓	SMTP	delay spam, utilizing spam databases
	FakeAP [45]	2002	2005	✓	802.11b AP beacons	p.o.c wireless honeypots
	HoneyBot [46]	2002*	2007*	✓	SSH, SMTP, FTP, HTML(++)	windows vulnerabilities and GUI
	BigEye [35]	2003	2003	✓	HTTP, FTP	emulation of different web servers
	Spamhole [64]	2003	2003	✓	SMTP	silent dropping of emails
	Spampot [65]	2003	2003	✓	SMTP	platform independence
	HoneyPerl [48]	2003	2003	✓	HTTP, FTP, SMTP, Telnet(+)	extensibility by modules
	Decoy Server [67]	2003*	2003	✗	SMTP, POP3	fake email server traffic
	Smoke Detector [35]	2003*	2004*	✗	FTP, HTTP, IMAP, SSH, SMB(++)	honeypot as a hardware
	NetBait [59]	2003	2007*	✗	*not specified*	honeypot as a service
	HoneyD [22]	2003	2008	✓	HTTP, POP3, SMTP, FTP(+)	emulating heterogeneous networks
	KFSensor [52]	2003	2015	✗	HTTP, SMTP, MSSQL, FTP(+)	commercial deployment of honeypots
	SpamD [60]	2003	2015*	✓	SMTP	tarpit against spam
	HOACD [47]	2004	2004	✓	*compare HoneyD*	live bootable CD (HoneyD, Arpd)

low	ProxyPot [61]	2004*	2004*	✓	SMTP	email spammer identification
	Impost [50]	2004	2004	✓	*all ports, but no emulation*	full packet sniffing
	Kojoney [55]	2005	2006	✓	SSH (shell activity)	first dedicated SSH honeypot
	Mwcollect [76]	2005	2009	✓	*compare Nepenthes, Honeytrap*	merging Nepenthes and Honeytrap
	Nepenthes [69]	2005	2009	✓	FTP, HTTP, TFTP, MSSQL(++)	capture worm payload
	GHH [43]	2005	2013	✓	HHTP-Apache, PHP, MSSQL	crawler and search engines
	Honeytrap [73]	2005	2015	✓	HTML, FTP(+), *dynamic emulation*	attacks via unknown protocols
	HoneyPoint [49]	2006	2014	✗	*not specified*	ICS/Scada, back tracking intruders
	Dionaea [71]	2009	2013	✓	SMB, FTP, SIP, MYSQL(++)	nepenthes successor, capture payload
	Kippo [53]	2009	2014	✓	SSH (shell activity)	emulate entire shell interaction
	Artemisa [85]	2010	2011	✓	VoIP, SIP	Bluetooth Malware
	bluepot [83]	2010	2015	✓	Bluetooth	Bluetooth Malware
	HoneySink [74]	2011	2011	✓	DNS, HTTP, FTP, IRC	bot sink holing
	HoneyDroid [86]	2011	2014*	✓	*compare Kippo, HoneyTrap*	p.o.c Android OS honeypot
	Glastopf [44]	2011	2015	✓	HTML, PHP, SQL	web applications, vulnerability types
	Kojoney2 [56]	2012	2015	✓	SSH (shell activity)	applying Kojoneys lessons learned
	Conpots [77]	2013	2015	✓	kamstrup, BACnet, mosbus	ICS and SCADA architectures
	IoTPOT [79]	2014*	2015	✓	telnet	IoT (ARM, MIPS, and PPC)
	honeypot-camera [80]	2014	2015	✓	HTTP	Tornado Web, Webcam Server
	Shockpot [81]	2014	2015	✓	Apache, Bash	Shellshock vulnerability
	Cowrie [54]	2014	2015	✓	SSH (shell activity)	Kippos successor
	elastichoney [78]	2015	2015	✓	elasticsearch	elasticsearch RCEs
high	Sebek [92]	2003	2011	✓	Win32 and Linux systems	attackers OS activities, state-based
	Honeywall [88]	2005	2009	✓	*compare Sebek*, CentOS	live bootable CD
	HoneyBow [91]	2006	2007	✓	Win32 Systems	extraction of malware, state-based
	Argos [87]	2006	2014	✓	Linux, Windows XP-7	zero-day exploits identification, tainting
	HIHAT [89]	2007	2007	✓	php-BB, -Nuke, -Shell, -Myadmin	PHP framework extension, state-based

Table 3.1: Overview and classification of honeypot software.
(+) indicates some additional services, (++) indicates many additional services.

Type	Software	Publication			Focus	
		First	Last	Free	Services / Applications	Design / Details
low	HoneyC [96]	2004	2007	✓	HTTP	identify malicious servers with snort
	SpyBye [100]	2007	2007	✓	HTTP	proxy, URL check by ClamAV
	Monkey-Spider [101]	2007	2009	✓	HTTP, JavaScript	threat database creation, several AV
	PhoneyC [95]	2007	2011	✓	HTML, JavaScript, PDF, ActiveX(+)	browser identities, dyn. analysis
	ADSandbox [102]	2010	2010*	✓	HTML, JavaScript	transparent protection, stat/dyn. analysis
	Thug [97]	2011	2015	✓	HTML, JavaScript, PDF, Flash(+)	complete emulation, stat/dyn. analysis
	YALIH [99]	2014	2015	✓	HTML, JavaScript, (IMAP)	precise by combining analysis methods
high	HoneyClient [104]	2004	2010	✓	Windows (Firefox, IE)	proof of concept, state-based
	Capture-HPC [106]	2004	2009	✓	Linux, Windows (Firefox, Office (+))	efficiency, scalability, state-based
	UW-Spycrawler [112]	2005	2006*	✓	Windows (IE)	spyware detection (AdAware), state-based
	HoneyMonkey [107]	2005	2007*	✗	Windows (IE)	IE vulnerabilities, state-based
	WEF [113]	2006	2007	✓	Windows (IE)	drive-by download attacks, state-based
	Capture Bat [103]	2007	2007	✓	Windows (Word, IE)	state changes on a low kernel level
	HoneyIM [114]	2007	2007*	✓	*compare Capture-HPC*	instant messaging
	SHELIA [111]	2008	2009	✓	Windows (IMAP, POP)	email malware, call-tracing
	Trigona [109]	2010	2010	✓	Windows (Browsers)	high throughput, —
	HoneySpider [110]	2011	2015	✓	*Capture-HPC, THUG*	hybrid client honeypot framework
	PwnyPot [115]	2013	2013	✓	Windows (Browsers+Plugins)	memory corruption & shellcode detection

Table 3.2: Overview and classification of client honeypot software by their interaction level type.

CHAPTER 4

Survey: Honeypot Data Analysis

The deployment of honeypots creates log files which describe the occurred incidents. The possible incidents are dependant on the type of honeypot deployed. Therefore it is important to define the problem statement that should be investigated before the actual deployment. If honeypots should look actively for communication partners, a client honeypot has to be used, a server honeypot otherwise. If we rather investigate meta-information of protocols and transactional data for specific services, low-interaction honeypots are the right choice. High-interaction honeypots should be used, if content, shellcode execution and the integrity of the operating system are of concern. A single problem statement can be answered by various metrics, each with its own approach and accuracy.

The overview in this chapter will follow the presented honeypot classification and will be divided into problem domains or statements and the associated metrics. The focus of this chapter lies in the log data analysis, performance metrics which are usually applied to benchmark a honeypot are not described, as they are mostly self-explanatory and just measure CPU, RAM, HDD load and the scalability. We focus slightly on low-interaction server honeypot log data, as this type of honeypots was deployed by us. The related work is structured by the means of assigning metrics (and research papers) to leading analysis questions, which shall give an overview about different methods and metrics in the field of honeypot data analysis.

In general, no research presents any in-depth honeypot data analysis before the year 2004. Prior to this, analysis consisted of mining IDSs alarms. However, IDSs can only be seen as complementary analysis tool. Honeypots can bring more information than simply provided by IDSs, especially compared to static signature based IDSs [132]. The following chapter will therefore review research paper released after 2004.

4.1 Attack Sources

If attacks occur on a honeypot, one has to specify where the attacks came from. The identification of an attacker is independent from the architecture or interaction type of the

honeypot and can be done with different granularities.

IP-Address or IP-Prefix
Autonomous System Number
Domain Name / URL / URL-Type
Country
UserID / Email
User Agent
Operating System

However, in the case of the server honeypots one has to consider that they might have received a spoofed IP-address. This might be a valid IP-address with a reachable or unreachable host, or it is a martial IP-address which should not have ever left the local segments like the broadcast address 0.0.0.0 [133]. Client honeypots usually are seeded with URL-lists and crawl for new URLs. The resources behind those URLs might be off-line. Moreover, one has to acknowledge that this identifiers are very changeable. IP-addresses can move from host to host because Internet network providers use IP-address-pools. That is the reason why some analysis combine the IP-address with a timeout and define an attack source as an IP-address that targets the honeypot environment for example within one day [132]. IP-Prefix announcements from autonomous systems change offer time or might be even hijacked. The Domain Name System inherently allows the abstraction from IP-address and/or hosts, which might lead to misleading results. Client honeypot analysis like Monkey-Spider [36] also perform some type of URL classification based on the URL and page-content, e.g *adult content*, *pirate*, *typos* etc.. The country might be extracted from the AS-registration information or some (commercial) third-party product has used to be used to retrieve the data [134, 135]. Trends show that usually the top 3 countries cause 60% of the traffic, which countries are observed is depended on the geographical location of the honeypot node. Another way used to identify spammers in instant messaging networks is the user name or the advertised URL [136]. Spammers tend to create a lot of accounts, which distribute man different URLs, which however lead/redirect to only a small subset of websites. A minor correlation between spim and spam senders exist. Research based on honeypots supporting the SIP/VoIP protocol also use the name of the user-agent for the fingerprinting of the attack source [137]. This information can be used by any protocol with such protocol-tag, however one has to keep in mind that such information can be omitted or easily spoofed. In order to infer the operating system from which the attacks originate from [2] usually additional passive OS fingerprinting tools like p0f are used, which recognize the attacking OS by analysing the packet composition as each OS creates the packets slightly differently. Almost all attacking machines are Windows-based [2].

4.2 Attack Target

If it is specified who attacks the honeypot, the next step might be to characterize the attack, more precisely one has to determine the target of the attack. Server honeypots classify the target by a specific application service, which is usually bound to a dedicated port. Port numbers are controlled by the Internet Assigned Numbers Authority (IANA)

and visible in official lists. However, services might be bound to another port. Therefore it is important to differentiate between ports and services, as an intruder might try to brute force a SSH-service on another port, which can be recognized by incoming valid SSH-packets on non-default ports. Most of the time, services are bound to default ports to improve reachability, that is why many researches treat a port as representative for a service [135, 132].

Port

Service

Client honeypots use a software client, which accesses a potentially malicious remote service. Therefore the target is usually the specific client software. It might be an emulated web-browser for low-interaction honeypots or a real one with plugins such as Flash for high-interaction honeypots.

Software Client

Software Plugins

Moreover, high-interaction honeypots (client as well as server) allow modifications to the operating system. Therefore OS-specific changes have to be analysed, which might differ across systems. For Linux it usually means the loading of some hidden kernel-modules and new cron jobs, for Windows Systems changes in the registry, system files and auto start entries. So an analysis might examine which OS is preferably attacked.

OS, OS component

4.3 Attack Frequency

One of the fundamental questions which has to be answered while deploying honeypots is if and how often honeypots are attacked? Interestingly, honeypots are exposed to attacks minutes after they have been activated [135]. However, this holds only true if the honeypot is accessible from the internet, if a firewall blocks all external incoming connections to the honeypot and only internal communication is allowed, attacks are observed rarely as they would have to come from infected hosts from within this specific network or because of a local misconfiguration.

Time until First Incoming Connection

Yegneswaran et al. [1] defined three metrics to describe the source arrivals in order to find differences for the events of misconfiguration, bot-attacks and worms-attacks. Those are the temporal source counts, the arrival window and the interarrival distribution. The first is analysed by the number of sources per time interval and shows distinctive patterns. Worms show a logistic growth with a steep begin and end, as they propagate very fast and autonomously and are shut down abruptly by a patch. Bots show similar characteristics, however bots usually apply a poll and pull communication pattern with their C&C server using a wake up time every x seconds, which results in less steeper curves. The arrival

window checks how many new sources have arrived in a specific time frame. Using a CDF plot no differences betweens the events could have been spotted. To evaluate the source interarrival characteristics, the data is broken up in successive intervals, each with an equal number of sources (e.g., 10 intervals each with 10% of new sources). Then the distribution of interarrival times is plotted. Bot-attacks and worm-attacks show exponential interarrivals. Moreover, the source-net dispersion can be interesting. This is usually only applied to TCP connections as UDP suffers from spoofing. Worm outbreaks have a much higher dispersion than bot nets and misconfigurations. A histogram can be computed on the count of sources seen from each /8 address aggregate if IP-addresses are considered, however other aggregates can be used.

Number of Sources per Time Unit
Number of new Sources per Time Unit (CDF)
Interarrival Time Distribution for equally-sized Source Intervals
Number of Sources in specific IP-Aggregates

Another metric combining the attack sources and frequency is relating the number of IP-addresses as a function of the number of attacks for each address [138]. This histogram follows the power-law distribution.

Number of Sources versus the number of attacks per Source

As we already have clarified the term attack is dependent on the honeypot type used. Server honeypots do assess any communication as malicious, whereby low-interaction server honeypots can describe the attack frequency based on network properties such as:

Received Packets / Requests per Time Unit
Received Data (kB) per Time Unit

Attack frequencies usually show specific peaks, instant massaging spam for example shows two daily peaks and one if observed on a weekly scale [136]. Moreover, peaks in attack frequencies can usually be linked to a single service, worm-activity etc. [134], which is heavily exploited at this specific point of time. Instant messaging and email honeypots use the following metrics:

Messages/Emails Received per Time Unit
URLs / Attachments Received per Time Unit
Received Data (kB) per Message

For high-interaction server honeypots the same metrics apply, however they can be extended by a OS-specific metric:

Exploitations per Time Unit

Client honeypots only count, independent from their interaction level, the exploitations per time unit and do not consider network features for the attack frequency as they ac-

tively begin the communication (it is preconfigured at which rate a client honeypot makes communication requests). So only exploitations are considered.

Another procedure used is the sessionization [138] of the data. All the packets received from the same IP-address within a time frame or without triggering a timeout are supposed to belong to the same attack session. 24 hours frames or 30 minutes timeouts are common. Furthermore, the time between the occurrence of an attack and the next attack can be examined. The probability density function (PDF) for this metric follows a heavy-tailed power-law and can be modelled by mixture of a Pareto and an exponential distribution. The lifetime of a source can be described as the complete time we see a source active on a honeypot [1], that means it is the time span from the source's first occurrence up to is lasts activity and might include several sessions. Botnets and misconfigurations cause short lifetimes, however worms prove to be persistent as they often miss a mechanism to stop scanning. If a specific source is observed regularly (that means it has frequent sessions or one long ongoing session), then it has a long source lifetime. Additionally, Song et al. [134] differentiate their sessions based on a IDS classification: All traffic data to the honeypot which triggers an IDS alert is labelled as a known attack session, all traffic data which ends with the transmission of shellcode but does not trigger an IDS alert is an unknown attack session. The time between sessions is also worth considering, as it displays the pause between active sessions.

Number of Sessions per Time Unit
Session Duration
Time between Sessions
Source Lifetime
(Un-)Known Attack Sessions per Time Unit

Song et al. [134] demonstrated a plot, which visualizes the frequency of the most targeted port per day. They plot the destination port as a function of time using a log-scale for the ports. The log-scale is an advantage because most of the attacks happen on the well-known root ports. The curve shows jumps across discrete levels representing ports of well-known ports for SSH, SMB and so on.

Similar to the notation of sessions are the flows [139]. The basic flows are based on the basic IP-flow and described by a 5-tuple consisting of source and destination IP-address, source and destination port, protocol type. The attack frequency can also be described by the occurrence of basic flows: If a packet differs from another packet by any key-field or arrives after a time-out, it is considered to belong to another flow. Therefore flows are a more strict requirement than sessions. Activity flows are an aggregation of basic flows based on the source IP-address only with a timeout for the inter-arrival time between basics flows. Hence, they resemble the definition of session.

Number of Basic Flows per Time Unit
Number of Activity Flows per Time Unit

4.4 Attack Evolution

If we observe certain temporal patterns for a specific source, port, country etc., it can be important to detect unusual behaviour for it automatically, because those anomalies might mark important events. That means, we want to learn, what normal behaviour is and to spot if this normal behaviour changes.

One possible method is to calculate ratios for different time aggregates and compare those values for different days or to the average ratio. This method is useful to recognize temporal trends which are only visible on a specific time scale [140]. The choice of a good time granularity depends on the kind of attack phenomena which is investigated: For short high-intensity attacks, like botnet probes or flash worms, it may be more useful to apply smaller time units, while for worms with a stealthier propagation scheme a larger time unit should be used. For example, the *stickiness* of mobile app users is rated by the ratio of average daily active users to monthly active users. Analogously in the context of honeypots a high variance of the following ratio might highlight some important events:

$$\frac{\text{Unique Attack Sources per Day}}{\text{Unique Attack Sources per Month}}$$

Francois et al. [133] demonstrated that graph intersections can be used to analyse distributed honeypot platforms. This method allows to highlight changes in the relationships between honeypots, for example if the percentage of mutual attacking IP-addresses changes for two nodes. Their research is based on two metrics, which create one value for the complete honeynet and not one value for each honeypot, which makes the analysis more simple. First, the maximal locality statistic, which is strongly related to the centrality measurement of graphs $G = (V, E)$:

$$\psi_k(v) = \textit{number of arcs of the subgraph of neighbours of } v \textit{ at a max. distance } k$$

$$M_k = \max_{v \in nodes} \psi_k(v)$$

Second, the standardized locality statistics at time t of the distributed system, which is calculated with respect to previous values of a sliding window with a size of τ.

$$\tilde{\psi}_{k,t}(v) = \frac{\psi_{k,t}(v) - \hat{\mu}_{k,t,\tau}(v)}{\max(\hat{\sigma}_{k,t,\tau}(v), 1)} \quad \text{with}$$

$$\text{common average value} \qquad \hat{\mu}_{k,t,\tau}(v) = \frac{1}{\tau} * \sum_{t'=t-\tau}^{t-1} \psi_{k,t'}(v)$$

$$\text{and variance} \qquad \hat{\sigma}_{k,t,\tau}(v) = \frac{1}{\tau - 1} \sum_{t'=t-\tau}^{t-1} (\psi_{k,t'}(v) - \hat{\mu}_{k,t'}(v))^2$$

$$\tilde{M}_{k,t} = \max_{v \in nodes} \tilde{\psi}_{k,t(v)}$$

This definition causes nodes which tend to remain constant in their relationship properties to have a low value. The first analysis using this metric was done to identify honeypots which capture unique attacking IP-addresses. Two nodes in the graph are linked only if the intersection between the corresponding sets represents less than a threshold α of the union

of addresses. Therefore, central nodes (high locality value) capture unique IP-addresses. When the maximal standardized locality statistics is low, no changes have occurred, however a high value indicates a major topology change: the relationships of attacking IP-address differ to the previous time instances. A similar analysis can be done for ports, an edge connects to nodes only if the set intersection of their attacked ports is lower then a threshold β.

Another way to distinguish between known and new patterns and highlight their occurrence was presented by Yegneswaran et al. [1]. They deployed a combination of honeypots and the intrusion detection system Bro. Therefore their events are based on Bro-profiles. However, their methodology can also be applied to pure honeypot data. They use a deviation value β to detect large-scale and unusual events. $\beta > 10$ indicates botnet-waves and fast-scanning worms, $\beta > 3$ slow- scanning worms.

$$\beta_{p_i} = mp_i / \sum_{j=0}^{i-1} p_i \quad \text{where}$$

p_i number of sources triggering profile p in time interval i,
m number of intervals prior to i where p was observed

Bro was also used to create profiles for new events, however one of the findings concludes that usually new events are just new minor variations of known activity.

Kaaniche et al. [138] presented a time evolution model created by linear regression. They examined, if a model based on attacks originating from only one specific country can describe the complete data set reliably. Surprisingly, they found a strong correlation between the models for single countries and the overall data set, independently from the countries proportion of the total number of attacks (some of the best fit countries account only for 2%-20% of attacks). The linear regression model is defined by:

$$Y^*(i) = \sum \alpha_j X_j(t) + \beta, \quad j = 1, 2, ..k \quad \text{where}$$

$X_j(t)$ denotes attacks from country j for time t, α_j and β best fit linear model parameters

The correlation between models is measured by the correlation factor:

$$R^2 = \sum (Y^*(i) - Y_{av})^2 / \sum (Y(i) - Y_{av})^2 \quad \text{where}$$

$Y(i)$ observed attacks, Y_{av} average number of attacks

Such correlation factors can be used to rate if the observed events are expected or vary substantially, that means have a high similarity distance to the model.

4.5 Propagation of Attacks

Besides analysing the attack activities in a isolated manner, one should also try to identify the propagation of attacks across several honeypots if a large honeynet is deployed. Propagation takes place, when one attacking IP-address is observed on one platform, then

subsequently on another [138]. Because of the IP-address-pools, this check for reoccurrence should happen within a specific time-frame for more precise results. Already early distributed honeypot analysis show that it is beneficial to deploy a large amount of honeypots from different IP-subnets and different geographical positions [58], as it is more likely to spot an attack, local events can be characterized as such and the propagation of an attacker across targets can be depicted.

Propagation can be modelled by a propagation graph [138], where nodes represent the individual honeypots and the edges (i, j) describe the probability to discover a seen IP-address at node i also at node j. However, nodes tend to show low propagation values if not set in the same subnet.

Propagation Graph

Similarly, Vasilomanolakis et al. [6] describe the propagation of attacks by *single-dimensional correlation* and *two-dimensional correlation*. Single-dimensional correlation groups attacks with the same origin, e.g. the same source IP-address. This correlation is analysed by two visualizations. A directed graph is created with all IP-addresses included in the communication, that means that honeypots and attackers are represented by a node. A directed edge represents an attack to a honeypot, so multiple edges originating from one node highlight its presence on several honeypots. Additionally, the number of targeted sensors in relation to the ratio of unique attacks is plotted, which shows that presence on more than two sensors is very unlikely for attackers. Two-dimensional correlation includes time as an additional dimension, which means that a mutual attack has to be observed on at least two sensors within a specific time-frame. As already discussed, this time frame should be below one day because of IP-address pooling, however Vasilomanolakis argues reasonably that internet-wide scanning has been significantly improved in the last years: Publicly available tools like ZMap [141] are capable of performing a complete scan of the IPv4 address space for one port with one probe per host in about 1 hour. Therefore, in order to find strong relations one can set the time frame to 1 hour and even lower. Hence, they use a scatter plot, which uses time slots of one hour and plots the number of unique attackers present on several sensors, in addition a colour signifies on how many sensors the attackers were present. Their observations suggest, that at least one unique attacker is targeting multiple sensor per time slot.

Attack Graph

Ratio of Unique Attackers in Relation to Number of Targeted Sensors

Number of Unique Mutual Attackers in Relation to Targets per Time Slot

A phase plot can also be used to visualize the successive targets, showing the next target as a function of the last target for a specific amount of attack samples [1]. Sequential IP-address scans will appear as a straight diagonal line in this visualization, however, as it is not working with mean values, it is dependant on the amount of attack samples used. Such plots can also visualize the attackers coverage, by which is meant how many of our honeynet IP-addresses were probed by a specific source. A full coverage can be recognized

by a horizontal line.

Phase Plot of Successive Destination Targets
Plot of Destination Targets for specific Sources

Attacks on different targets can be visualized by a destination-net scan footprint [1], which is a plot counting the number of attack sources over the number of targets they have attacked. Obviously, this visualization works best if many honeypots are deployed or whole subnets are redirected to a honeypot. Misconfigurations tend to show hot spots, worms and bot cause an evenly distributed pattern. Furthermore, the first destination preference might be interesting to analyse, as this might reveal some ordering in the scanning of subnets by worms and bots.

Number of Sources per Number of Targets
PDF of First Destination Preference

4.6 Attack Patterns

The general concept of many data mining tasks, like common pattern detection and clustering, involves the following procedure with three steps [142]:

1. feature selection and/or extraction, pattern representation
2. definition of a pattern proximity measure appropriate to the data domain
3. grouping similar pattern

The first step includes the extraction of certain features characterizing the relevant aspects of the data set and representing them with an adequate means, for example an array of values. An effective measure to describe the similarity of two data series is done by a similarity distance like defined by Mahalanobis, Pearson, Spearman and or alike. The grouping or clustering of patterns is done by clustering algorithms like the K-Means-Algorithm.

Unfortunately, the discipline of pattern detection does not offer a straight forward method for all data types. Not every algorithm can handle all cluster shapes or sizes and runtime and output quality might differ severely on different data dimensionality and types. Different clustering algorithms produce different partitions of data, and even the same clustering algorithm is dependant on its initializations and configurable parameters. Indeed, the real skill in pattern detection is the choice of a proper clustering algorithm (and similarity measure) as hundreds of clustering algorithms exist [143]. This is the reason why we see so many different approaches in the field of honeypot attack pattern detection, but also any other clustering discipline.

The problem of network traffic clustering and anomaly detection is not a new discipline and has been extensively studied. Approaches commonly utilise signature based methods in combination with intrusion detection systems, statistics (e.g. Moving Average Models[144, 145], Principal Component Analysis [146, 147]) or data mining and unsupervised machine learning (e.g hierarchical clustering [148], KNN-clustering [149]). However, only some research was done explicitly in the field of honeypot traffic analysis [150]. We

concentrate on the publications which effectively deployed and/or analysed honeypot and honeynet traffic.

The widespread procedure of association rule mining was applied by Pouget [132] to find interesting relationships and patterns between observed events. With the induction of association rules one tries to find sets of *items*, i.e. events, port numbers, IP-addresses and so on, that frequently occur together. It originates from customer behaviour analysis, that tries to recommend products based on sets of collectively bought items. This means that an association rule R states that if we see specific action a and b, we can be confident, quantified by a percentage, that also action c will be observed: $a \cap b \Rightarrow c$. The metrics which are applied are *support* and $confidence$. Support is the ratio between the number of transactions that include all items of the rule and all transactions. Confidence is the ratio between the number of transactions that include all items of the rule and the transactions that contain the premise. Rules should have a minimal support threshold so that only meaningful rules are found.

$$Support(R) = \frac{\#transactions\ containing\ \{a \cap b\}}{\#transactions}$$
$$Confidence(R) = \frac{\#transactions\ containing\ \{a \cap b \cap c\}}{\#transactions\ containing\ \{a \cap b\}}$$

Pouget applied association rules to each port sequence group and mined on the following features: number T of machines in the environment targeted by one attack source, n_i number of packets sent by attack source to honeypot i and N, which is the total number of packets sent by one attack source to the whole environment. The resulting rules represented meaningful clusters, which have been ascribed to attack tools and proofed themselves to be a better clustering technique than port sequences only.

One of the first extensive works on honeypot data was done by Thonnard et al. [140], who group attacks on a honeynet by detecting common time series patterns. They have used the data from the leurre.com project and their data aggregation is based on the ordered list of ports targeted by a source identified by an IP-address. All attackers having the same attack fingerprint are classified into one set, then the number of unique source counts per time unit for each class is calculated. Furthermore, only time series' which have at least one peak of activity with a minimum of $x = 10$ sources for a given time unit are considered. That means, that featureless data (port sequences which are not mutual) is filtered to assure a certain quality. The symbolic aggregate approximation (SAX) is used to reduce the dimensionality of the data and to make a fast similarity distance evaluation possible. SAX approximates time series data by segmenting into time intervals of equal size and summarizing each of these intervals by its mean value. Each interval is then mapped to a finite alphabet symbol. The alphabet is chosen relatively, i.e. symbol B representing the range 20%-30% of the maximum unique source count etc.. Time series can then be interpreted and compared as a string. Thonnard defined similarity if more then 90% of the symbols match for a given pattern and used an alphabet with 8 symbols. If N is the number of elements in time series T, the $dist()$-function returns the inter-symbol distance and ω is the number of intervals in the SAX representation, then the *minimum SAX distance* can

be calculated as follows:

$$SAX(W_{T_1}, W_{T_2}) = \sqrt{\frac{N}{\omega}} \sqrt{\left(\sum_{i=1}^{\omega} dist(W_{T_1}(i), W_{T_2}(i)) \right)^2}$$

The grouping algorithm applied is a greedy algorithm, which takes a pattern and combines all other patterns to a cluster which exceed the similarity threshold and then excludes the found cluster from the pattern list. Afterwards the next remaining pattern are analysed. Clique-similar clusters emerge. Despite the local decision making scheme, fairly good results are produced and another advantage is that the number of total cluster has not to be known in beforehand.

Ultimately, the resemblance of temporal behaviour is compared, even if the attacks differ by their port sequences and by the IP-addresses of the sources. Surprisingly, only three patterns of attacks have been spotted by Thonnard et al.:

1. continuous activity
2. sustained bursts
3. ephemeral spikes

Especially the ephemeral spikes can lead to a false clustering, because the symbols of the alphabet are chosen relatively due to a standardization process. This means that temporal patterns with only a few spikes and zeros or very small values have a mean value close to zero and SAX calculates a high similarity degree because all those values are represented by only one symbol. However, a similarity has not to be the case. Therefore a global and local similarity measure is necessary. Global similarity SIM_G is defined using the largest lower-bounding distance that is theoretically possible, as denoted by the abstract variables W_T and $\tilde{W}_T$ which have maximal distance for every pair of symbols. Local similarity SIM_L compares only values, if one of the patterns exceeds the upper quantile value UQ at a given time unit, 0.975 was used by Thunnard. Again, W_T and $\tilde{W}_T$ are used to calculate the largest distance between every pair of symbols based on X_{UQ}. Both measures are combined to obtain the total similarity.

$$\begin{aligned}
SIM_G(W_{T_1}, W_{T_2}) &= 1 - \frac{SAX(W_{T_1}, W_{T_2})}{SAX(W_T, \tilde{W}_T)} \\
X_{UQ} &= \cup_{k=1}^{2} \left(x_i | W_{T_k}(x_i) > UQ, \quad \forall i \in \{1, ..., |W_{T_k}|\} \right) \\
SIM_L(W_{T_1}, W_{T_2}) &= 1 - \frac{\sum_{j=1}^{|X_{UQ}|} SAX(W_{T_1}(x_j), W_{T_2}(x_j))}{\sum_{j=1}^{|X_{UQ}|} SAX(W_T(x_j), \tilde{W}_T(x_j))} \\
SIM_{total} &= \frac{SIM_G + SIM_L}{2}
\end{aligned}$$

Krasser [151] presented a way to make attack patterns on honeypots easily detectable to the human eye. It is not based on a mathematical calculations, rather it presents an intuitive way of visualizing network traffic information in real-time for monitoring or in playback mode for forensics. It is combination of animated scatter plots and parallel coordinate plots. Figure 4.1 shows an sketch of the visualization. The left vertical line denotes the source IP-address, 0.0.0.0 is at the bottom, 255.255.255.255 is at the top. The right vertical

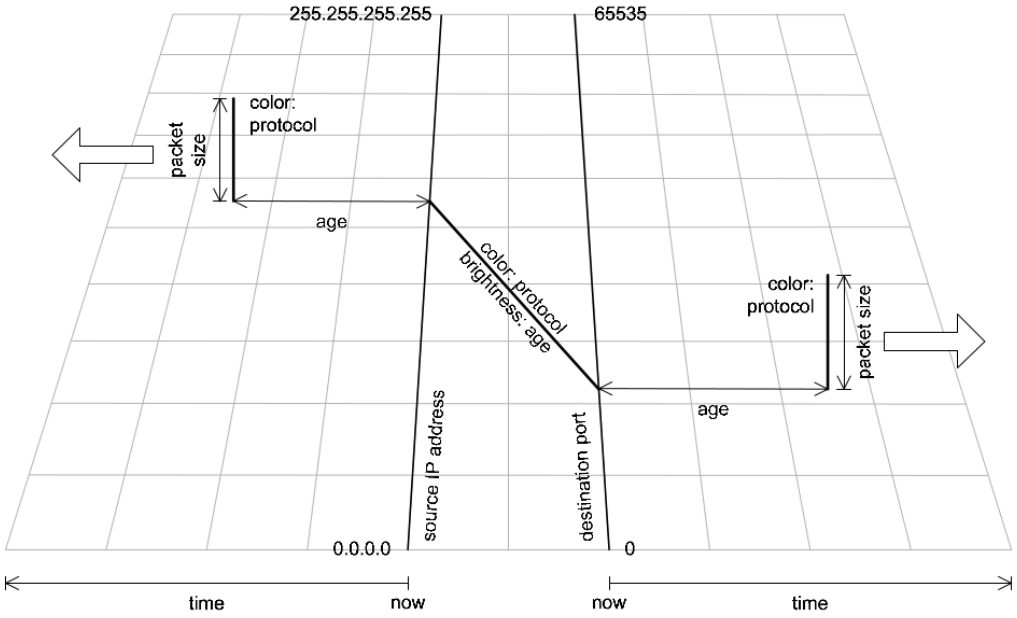


Figure 4.1: SecVis visualization overview by Krasser.

line denotes the destination port, port 0 at the bottom, port 65535 at the top. Coloured lines connect those vertical lines, one line for each packet. UDP packets are visualized by blue links, TCP packet by green links. Each packets also triggers two bars, whose height represent the packet size. The bars move away from the vertical lines as the packets gets older. Krasser successfully demonstrated that this visualization proves to be useful for inbound as well as outbound traffic and is capable of highlighting differences in the traffic bit rates, common attack ports and sources or patterns like reoccurring worm attacks.

Almotairi [139] has used the principal component analysis (PCA) to separate latent groups of activities and to find outliers from cluster groups. The PCA is a multivariate statistical technique which is used to reduce the dimensionality of a data set into a few linearly uncorrelated variables, called the principal components. The resulting number of principal components is less than or equal to the number of original variables p and the components are defined and ordered in such a way that the first component has the largest variance. Therefore much of the variance in the original set can be retained be choosing the first k PCs with $k < p$. PCs are usually selected by the Kaisers' rule [152], which suggests the elimination of all PCs with a Eigenvalue of less than one.

$$
\begin{aligned}
C_i &= \frac{X_i - \bar{X}_i}{\sqrt{s_i}} \\
Z &= A^T C \\
Z_1 &= A_1^T C = a_{11}C_1 + a_{12}C_2 + a_{13}C_3 + ... + a_{1p}C_p \\
&\vdots \\
Z_k &= A_k^T C = a_{k1}C_1 + a_{k2}C_2 + a_{k3}C_3 + ... + a_{kp}C_p
\end{aligned}
$$

To calculate the PCs the p-dimensional vector $X = (X_1, .., X_p)^T$ is standardized by C_i for all $i = 1..p$, where $\bar{X}_i$ is the sample mean and s_i is the sample variance for X_i. Empirical experience shows that PCAs on honeypot data should be calculated using the correlation matrix R of C. The Eigenvector of R is $A = (A_1, ..., A_k)^T$, the Eigenvalue vector of C is $l = (l_1, ..., l_p)$ and the first component equals to Z_1. The components can be interpreted by analysing the loading of the components variables, as high loading indicates significance, which allows the assignment to events such as *targeted attacks against open ports* or *scan activities*. Eventually, the results can be visualized with scatter plots having a PC on each axis, which Almotairi used to find interrelations between components and also to identify extreme activities or outliers. Those can be identified by sight, however also automatically by applying contours (the population ellipsoids) around the mean values (Z_{ik} is the score of the k^{th} PC of ith observation, l_k is the k^{th} Eigenvalue). The first and last components can give interesting insights, as they mark the most and least important properties.

$$\sum_{k=1}^{p} \frac{Z_{ik}^2}{l_k} = const, \qquad \text{e.g.:} \quad \frac{Z_{i1}^2}{l_1} + \frac{Z_{i2}^2}{l_2} \leq 5.8$$

One of the latest honeypot attack analysis was published by Owezarski [150], whose approach uses unsupervised machine learning methods and applies robust clustering techniques. Eventually, this method creates signatures automatically based on the clusters. This analysis is founded on a general algorithm which was applied to recognize anomalies in production traffic [153] and was altered to work on server honeypots traffic. Owezarski aims at increasing the robustness of the clustering algorithm by the divide and conquer approach sub-space clustering (SSC) and the notion of clustering ensembles. A clustering ensemble P consists of a set of N partitions P_n produced for the same data with $n = 1, ..., N$ by the same clustering algorithm. Each of these partitions provides different and independent exposure of data patterns, which then can be combined to a global cluster including all metrics and facilitating a holistic understanding of the threat. A shift of the similarity measure from patterns to the clustering results takes place. This is possible because of the downward closure property [154]: *If a collection of points is a cluster in a d-dimensional space, then it is also part of a cluster in any (d-1) projections of this space.* Moreover, high-dimensional data tends to be sparse, therefore clustering algorithms create better results in lower dimensions, which also have a smaller computational cost. The partitions are combined by the Inter Clustering Result Association (ICRA) method to correlate clusters and Inter Outlier Association (IOA) method to correlate outliers. The notion of graphs is used: a vertex is a cluster/outlier from any sub-space and an edge represents a high similarity. For ICRA, similarity CS is calculated by the ratio of mutual flows the clusters C_i are based on, a threshold of 0.9 was effective for Owezarskis data: $CS(C_1, C_2) = \frac{|(C_1 \cap C_2)|}{max(|C_1|,|C_2|)} > 0.9$. IOA links outlier vertices if and only if completely the same flows are responsible for that outlier. Similar to Thonnard [140] a greedy algorithm is used to recognize cluster-cliques because of performance reasons. The intersection of all flows present within cluster-cliques form the anomaly of interest.

Gregio et al. [155] have demonstrated the effectiveness of classical data mining and supervised classification approaches in the honeypot data log analysis using the well-established k-nearest neighbour (KNN) algorithm, neural networks and decision trees. The purpose of this analysis is to differentiate between internet noise like inoffensive or known traffic and

anomalies and real attacks. So their setup does not only include honeypot traffic which is labelled as an attack but also productive traffic which is labelled as normal. Both traffic types are used to support a learning process and to make a classification of traffic possible. KNN uses a simple classification principle: For each instance of data of unknown class the distance to data with known classes is calculated and the k nearest neighbours decide by a majority vote which class is selected for the unknown element [143]. Artificial neural networks are computing structures that are composed of single processing units called neurons. Different forms of neural networks exist, however a well-understood and commonly used model is the multilayer perceptron (MLP). MLP utilizes a supervised learning technique called backpropagation for training the network and consists of multiple layers of neurons in a directed graph [156]. A decision tree is a machine learning algorithm which performs successive partitioning of the original data set into successively more homogeneous subgroups. Each node in a decision tree resembles a partition decision. Hence, the higher a decision tree is the higher the detail level of clustering [157]. Even for the most simple classification of traffic, either suspicious or normal, the KNN algorithm was too slow to create reasonable results. Neuronal networks performed good and produced good results, however with a quite high share of false negatives. Decision trees preformed best and produced correct results with few false positives and a moderate number of false negatives.

Honeycomb [158] is one of the most famous HoneyD plugins and scans incoming traffic to detect repeating patterns in packet payloads using the longest common substring (LCS) algorithm. This implementation is based on suffix trees, which are used as building blocks for a variety of string algorithms. Using suffix trees, the longest common substring of two strings is straightforward to find in linear time. Suffix trees can be generated for example with the Ukkonen's algorithm [159]. Honeycomb performs as the first step a *protocol analysis*, by which traffic is ordered by network and transport layer header information (IP-address or Ports). After that LCS is applied in two different ways: *Vertical pattern detection* concatenates for two connections the incoming messages into one string, respectively, and compares then the resulting string. *Horizontal pattern detection* compares for two connections two messages at the same depth in the dialogue, that means that LCS is applied to the n-th messages.

4.7 Attack Root Cause Identification

Attack root causes can be defined as the most basic cause that can be reasonably identified as the origin of the attack. The root cause can be associated to a specific attack tool, or one of its variants or configurations. One of the main tasks of honeypot data analysis is the assignment of clusters and recognized patterns to root causes. However, this is not necessarily an one-to-one relationship, as it is difficult to guarantee that a found cluster is caused by only one attack tool and that one attack tool does not cause two clusters, for example by different attack configurations. Ultimately, a cluster should always remain explicable in its formation [132].

Before a cluster can be assigned to an attack root cause, it is necessary to validate the cluster coherency, that is if we found good (meaningful) clusters. Undoubted, different attacks can create the same number of packets on the same ports, therefore a pure statistical analysis of transactional data might be not enough. One possible way of determining the coherence

is by considering the packet data content. The payload of all packets sent from one attack source can be transformed into strings and concatenated. This creates an attack fingerprint, which then can be used to check the cluster coherency by comparing the fingerprints by a simple string distance measure like the Levenshtein distance. Pouget [132] used this method to prove that his clustering technique is meaningful, as the fingerprints i from a Cluster C mostly have a distance $d_i = 0$ to their cluster partners, resulting in a very low average distance D_C for a cluster, which in turn means that a mutual payload exists.

$$\begin{aligned} d_i &= \sum_{j \in C} \frac{D(i,j)}{n-1} \\ D_C &= \sum_{i \in C} \frac{d_i}{n} = \sum_{i,j \in C, i<j} \frac{2D(i,j)}{(n-1)n} \end{aligned}$$

Polymorphic attacks are attacks that are able to change their appearance with every instance. Thus, polymorphic worms pose a big challenge to the honeypot pattern detection, and more specifically to root cause identification, as worms might change the attack vector for exploiting the vulnerability or the attack body might change by garbage insertions, encryption, instruction reordering and so on. Therefore (sub-) string based methods like LCS are insufficient. Different approaches exist, however their research is based on the mutual premise that despite the polymorphism the worms must have some invariant substrings. Indeed, such invariants exist [160] and the meaningful strings which can be used for classification have to be found.

Tang and Chen [161] proposed the design of a *double-honeypot* as a counter measure to worms. The novelty of this system is the ability to distinguish worm activities from non-attacking behaviour on honeypots, as for example misconfigurations. This system is composed of two independent honeypot arrays, the inbound array consisting of high-interaction honeypots which allows compromise and an outbound array consisting low-interaction honeypots. If a compromised inbound honeypot tries to find and infect other victims, all outgoing traffic initiated by the honeypot will be redirected by a network translator to the outbound array. If one of the outbound honeypots sees network traffic, a compromise on the inbound honeypot took definitely place. Expanding this work, Mohammed, Hashim et al. [162] proposed a *double-honeynet* to solve some of the limitations of the double-honeypot. The double-honeynet is a combination of two honeynets consisting only of high-interaction server honeypots. One honeynet is destined to receive incoming connection requests Once a honeypot from the first honeynet H_1, is compromised, a worm will attempt to make outbound connections. An internal translator intercepts all outbound traffic and redirects it to the second honeynet H_2, which also allows the infection. This procedure repeats, causing the worm to spread back and forth across the honeynets and to manifest different instances. Those instance are collected centrally and a signature generation algorithm is initiated: First a substrings extraction process takes place which creates the set of creates all possible substrings and determines their frequency. Then the PCA is applied on the frequency count to reduce the dimension and get the most significant strings, which then can be used to create a signature and to define a root cause.

Another way of assigning the attacks root cause is by determining which **attack tool** was used to convey the attack. This can be done by analysing port sequences or the TCP Initial

Sequence Number (ISN).

ISN

Port Sequence

Some attack tools use always the same ISN or a bad random number generator with a low entropy [133], which makes it possible to assign some ISN to specific attack tools. Moreover, honeypots receive backscatter packets, which are ACK-replies to TCP-SYN packets and therefore possess the ISN+1. Those can also be used to observe attacks on other systems by these specific tools. Backscatter exists because there is no way to reliably recognize spoofed IP-addresses. Moreover, hosts which receive spoofed addresses will receive ICMP Destination unreachable messages after answering such requests. As ICMP messages are structured in type and codes, both have to be considered.

ICMP Type / Codes

The analysis by port sequences was introduced by Pouget at al. [132] in order to show that frequent/repetitive attacks on honeypots create large amount of data, which might lead to misleading results if general statistics are applied. However, a closer look on port sequences can reveal related attack tools. A port sequence is a time ordered sequence of ports without duplicates [2] that represents the order in which the attack sources (IP-address with a timeout of 1 day for Pouget) sent packets to specific ports, for example: Attacker sents TCP requests to port 135, again on 135 and then on port 4444 creates $\{135T, 4444T\}$. Port Sequences can be created for observation on a single honeypot or for observation on several honeypots. Preliminary results from Pouget showed that each sequence is often limited to only one port, and that a port sequence represented as a *set* is almost uniquely identified by this set, however because of some rare cases the ordered sequence was preferred. Pouget observed more than twice as many port sequences as unique targeted ports. The distribution is similar to other metrics, as the top 8 sequences already characterize the activity of about 75% of the attacks [163]. These results motivated a further in- depth investigation.

Another possibility to deduce an attack to a specific attack tool was discovered by Kohlrausch [164]. During a buffer overflow the instruction counter (EIP) is overwritten with a new return address at which the shell code can be assumed. Moreover, honeypots like Argos track the preceding value of the instruction counter (faulty EIP) which is the last legitimate instruction before the exploit had taken over the control. Evaluations show, that the values of the EIP and faulty EIP are characteristic for an exploit tool and operating system pair. However, this analysis might be strongly biased by active address space randomization algorithms.

EIP and faulty EIP

Another major problem in identifying a single root cause for attacks on honeypots were discovered by Alata and Pouget [165, 166]. Pouget made the observation that different sets of compromised machines are used to carry out the various stages of planned attacks. That means, that a single attacker causes different attack patterns from different machines on the honeypot. In addition, Alata observed this phenomenon also on a SSH high-interaction

honeypot. Two groups of attacking machines have been spotted: The first group is composed specifically to scan hosts and perform dictionary-attacks. If they are successfully, usually a day later a machine from the second group appears. This group has no intersection with the first group in terms of IP-address and geographical lookups even reveal different countries. After a login the second machine tries to run own services or get root access. Interestingly, comparing the attack sources between low- and high-interaction honeypot data sets demonstrates that mutual IP-addresses are from the scanning group only, the second intrusion group also never appears on low-interaction honeypots.

In general, attack root cause identification requires a good knowledge of black- hat tools or recent participation on security pages and mailing pages, which usually inform about Common Vulnerabilities and Exposures (CVE). That is why it can be really difficult to assign an honeypot attack cluster to a known attack tool. Moreover, because of the increasing complexity of worms it is necessary to rather perform payload analysis (byte sequences, shellcode commands etc.) [160] than pure statistical evaluations and the mere detection of a exploit. Honeypots are still used to collect worms, however the signature generation for worms evolved into its own broad field of studies and are rather the domain of intrusion detection systems [167].

4.8 Attack Risk Assessment

The risk estimation is rather done on high interaction honeypots, as it can be assessed based on the severity of the vulnerability and the analysis of the exploit. However, risk estimation can also be done for low-interaction honeypots based on the scope of the attack, which can be measured by three features describing the amount of communication: The number of packets of the attack, the amounts of bytes exchanged in the attack and the communication duration.

$$risk = \log(nPackets) + \log(nBytes) + \log(duration + 1)$$

Ozewaski [150] extended this risk estimation by multiplying the risk value by the number of sub-spaces the attack is found in, which is an indication for how many network-features the attack affects.

$$risk_{subspace} = C * \log(nPackets) + \log(nBytes) + \log(duration + 1)$$

SweetBait [168] is designed to be an automated response system that protects from random IP scanning worms using low-interaction (honeyD) and high-interaction (Argos) honeypots. Honeypots are used to create signatures, which then are sent to IDS/IPS sensors in order to determine the virulence of worms on production systems. The expected virulence of worms is based on their aggressiveness, which is quantified by the exponentially weighted moving average (EWMA) of the number of alerts generated by each signature on each period: $m' = w \times a + (1 - w) \times m$, where m' is the new value, m the previous value and the weight $0 < w \leq 1$ configures the computation to follow more or less aggressively the recent changes in activity levels, whereby values below 0.5 proved to be not useful. This EWMA value is then used to predict the virulence by adjusting the value by port and protocol bias value,

which is useful for especially active ports such as 145: $A = m \times port\,bias \times protocol\,bias$. The EWMA can be used not only to predict signatures observed by IDS, but also any temporal pattern on honeypots.

4.9 Exploit Detection

As high interaction honeypots are actively exploited, they also consider the vulnerabilities which were exploited in their analysis. Two main procedures to detect exploitations and to find vulnerabilities are either the data-driven technique (example: Argos) or the operating system state monitoring (example: Capture-HPC) [164]. The former detects exploitations by dynamic taint analysis, which is based on the idea that all data from the internet is potentially malicious and therefore is marked as tainted. The data-flow of tainted data is monitored. The exploitation of a honeypot is then specified as the direct execution of tainted shell-code. Dynamic taint analysis is very accurate and reliable to detect attacks utilizing buffer overflows. The latter inspects the states of operating systems and tries to spot illegitimate actions in the file system or process management. An exploitation has occurred if modifications (sometimes even read operations) are done to this specific locations, no active execution of those modified files is required by definition, however this usually happens implicitly. Modifications can be recognized comparing the files with a backup, by comparing hash-values or by controlling the kernel-log for sensitive calls.

Argos memory tainting technique was extended by SweetBait [168], which inserted its own shellcode into the code that is under attack, which makes the gathering of more information about the memory and process states possible.

4.10 Overview Honeypot Data Analysis

Table 4.1 gives an overview over the various honeypot metrics used in research projects. Interestingly, one finding of this overview is that most of the researchers tend to pose the first three presented questions, which refer to the attack source, attack target and the frequency. Furthermore, a common consensus exists in identifying sources or targets and in describing the frequency, as many of the metrics and analysis methods are reused throughout the publications. The reason behind this circumstance is that *direct (apparent) information* is evaluated and in cases like the country-mapping extended by simple lookups. Direct information describes the observations and is recorded in honeypot logs during common operation: Usual honeypot logfiles contain the source, the target and the timestamp of an attack based on the IP information. It is important to note, especially for the IP, that a communication without an IP- address for the source and target would be not possible and that each event has a timestamp. Therefore, it is straightforward and natural to pose analysis questions based on these features.

However, this situation is different for the remaining questions, because they attempt to *derive information*. Derived information explains, assesses or localizes the cause of the observations which is fundamentally more complex than mere description. Since the analysis is more complex, such research appeared later than simple descriptive analysis and less overlaps between methods exist. This is especially true for the pattern-detection, which can be done by many different similarity measures and clustering algorithms (as explained in Section 4.6). The conduct of such analysis is growing into an interdisciplinary approach, because in order to derive information, basic statistics usually do not suffice any more: sophisticated honeypot networks and methods from other fields like Association Rule Mining, Neuronal Networks, Memory Tainting in virtual machines to name a few, have become necessary. In general, the bond between honeypots and other research fields has intensified during the last years.

Problem Statement	Analysis	Examples
Do common attack origins exist?	IP-Address or IP-Prefix	[2, 132, 133, 134, 1, 138]
	Autonomous System Number	[169, 170]
	Domain Name, URL, URL-Type	[36, 135, 134, 136]
	Country	[2, 132, 135, 134, 136]
	UserID / Email	[136, 165]
	(Worm-) Signature	[58, 134, 1, 168]
	User Agent	[137]
	Operating System	[2, 132]
What is the target of the attack?	IP	[138, 150, 162]
	Port and Transport Protocol	[2, 132, 133, 139, 140, 155]
	Service	[132, 135, 165]
	Software Client, Plugins	[36, 98]
	Vulnerability	[87, 165]
	OS	[2, 134]

What are the attack frequencies?	Time until First Incoming Connection	[135, 165]
	Number of Incoming Connections per Time Unit	[138]
	Number of Sources per Time Unit	[34, 133, 1, 140]
	Number of new Sources per Time Unit (CDF)	[1, 2]
	Interarrival Time (Distribution for equally-sized Source Intervals)	[1, 138, 162]
	Number of Sources versus the number of attacks per Source	[138]
	Received Packets per Time Unit	[133, 155, 162]
	Received Data (kB) per Time Unit	[133, 155, 162]
	Messages/Emails Received per Time Unit	[136]
	URLs / Attachments Received per Time Unit	[136]
	Received Data (kB) per Message	[136, 151]
	Exploitations per Time Unit	[164, 165]
	Sessions per Time Unit	[138, 155]
	Session Duration	[138, 155]
	Time between Sessions	[135]
	Source Lifetime	[1]
	(Un-)Known Attack Sessions per Time Unit	[58, 134, 1]
	Number of Basic Flows per Time Unit	[139, 150]
	Number of Activity Flows per Time Unit	[139]

How to detect changes in attacks?	Ratios between different Time Units	[140]
	Sliding Window and Locality Statistics	[133]
	Deviation β value	[1]
	Linear Regression	[138]
How to compare propagation?	Propagation Graph	[138]
	Attack Graph	[6]
	#Unique Mutual Attackers per #Targets per Time Slot	[6]
	Phase Plots	[1]
	Association Rule Mining	[132]
	Number of Sources per Number of Targets	[1]
	Ratio of Attackers per Number of Target Sensors	[6]
	PDF of First Destination Preference	[1]
How to detect attack patterns?	SecViz Visualization	[151]
	Symbolic Aggregate approXimation (SAX)	[140]
	Principal Component Analysis (PCA)	[139, 162]
	Sub-Space Clustering (SSC)	[150]
	Multilayer Perceptron (MLP)	[155]
	Longest Common Subsequence (LCS)	[158, 162]

How to identify a root causes?	Cluster Coherency	[132]
	Double-Honeynet	[162]
	ISN	[133]
	Port Sequence	[132]
	Faulty EIP	[164]
How to assess the risk?	Communication Scope	[150]
	EWMA	[168]
How to recognize exploits?	Data-Driven Technique	[87]
	Memory Tainting	[106]

Table 4.1: Metrics used in honeypot data analysis and related publications.

CHAPTER 5

Framework for Honeypot Data Analysis

5.1 Design

This chapter describes the requirements for the analysis framework which will be used to perform a data analysis on a large honeypot data set. Two types of requirements exist, the functional and non-functional requirements. Both will be discussed in the following. Based on these requirements a software architecture is introduced.

5.1.1 Functional Requirements

Functional requirements define what a system is supposed to accomplish. The intended framework should be able to transform the heterogeneous log files from different applications into one unified log format. This format should be the basis for an exploratory analysis, which does reveal similarities and differences between honeypots using a different network connection type. The framework shall output its results whenever possible by plots or other visual medium, but nevertheless provide textual results which make an in-depth view possible. The framework should support the analysis with respect to specific supplied properties, such as smaller time periods, perspectives on only one port or one attacker-ID and so one. A filter-mechanism has to be present, which filters unwanted logged events, for example administrative actions or communication with the multiple data collection points. Eventually, single results should be merged into a single report to allow a faster comparison of results.

5.1.2 Non-Functional Requirements

Non-functional requirements are also known as quality requirements. The most important non-functional requirement for the analysis framework is the extensibility. As it is not clear at the starting point, which analysis methods will be required, the extension by arbitrary analysis methods should be supported, which then are automatically applied on the data set. Moreover, the performance of the framework has to be as good as possible, more precisely, the runtime of an analysis should be as short as possible. Reason for that is the exploratory

nature of the analysis, which requires multiple queries with different configurations. As we deal with a data set which does not fit into the main memory (and might perform memory-intensive operation on it) we have to define a procedure which keeps the memory foot print appropriate to the machine it is currently running on. Lastly, the framework has to be robust considering missing data. Honeypot sensor failures produce gaps in the data set, furthermore mapping functions such as IP-address to ASN do not always have successful lookups. Those incomplete information should never interfere with available data and bias the result.

5.1.3 Architecture

Large log files motivate to use distributed, scalable computing architectures and to spread out small jobs to workers in a worker pool via the network. However this is not always necessary and recent recommendations suggest that tools incorporating such architectures, for example Hadoop, are used too early. Such architectures are designed for data sets larger than 5TB [171]. For smaller data sets the programming effort is too high as functions have to be rewritten to match distributed computing. Moreover, the speed up on small data sets can be negligible. This is the reason why a *local* architecture is proposed, as our data set does not exceed this limit.

Data traverses the framework in a *linear* manner, that means that if a specific information reaches a component, it is processed and then forwarded to the next component, which is known a priori. The four main components of this framework are responsible for data preparation, data extension, data analysis and data output.

Data Preparation The first component transforms the heterogeneous log files into a unified homogeneous data set. This is accomplished by accessing the various log files in a structured manner and then parsing the log files by regular expressions, database queries and so on, depending on the log format used. Important information such as attacker-ID have to be stored, however unnecessary messages such as debug information have to be omitted to save memory.

Data Extension This component extends the data set by additional information, which was not collected by the honeypots, however is interesting for further analysis. Possible extensions are mappings of IP-addresses to countries or ASN. This component as well as the data preparation component are meant to be executed once, as they prepare and enrich the data set which then is the basis for the data analysis.

Data Analysis The analysis component is responsible for the application of analysis functions. It has the memory management system, which loads appropriate sized data chunks consecutively so that the available memory is not exceeded. If the analysis engine notices a specific function-keyword in the query-list, it automatically reserves memory for this function and computes intermediate results on the data chunks and derives eventually the final result by applying a merging method, which combines the intermediate results. This system architecture resembles the MapReduce [172] programming model, which is originally designed for distributed computing, but is here used on a singular computing device to combine results from the data chunks.

Analysis functions are defined in a separated module of the framework called the

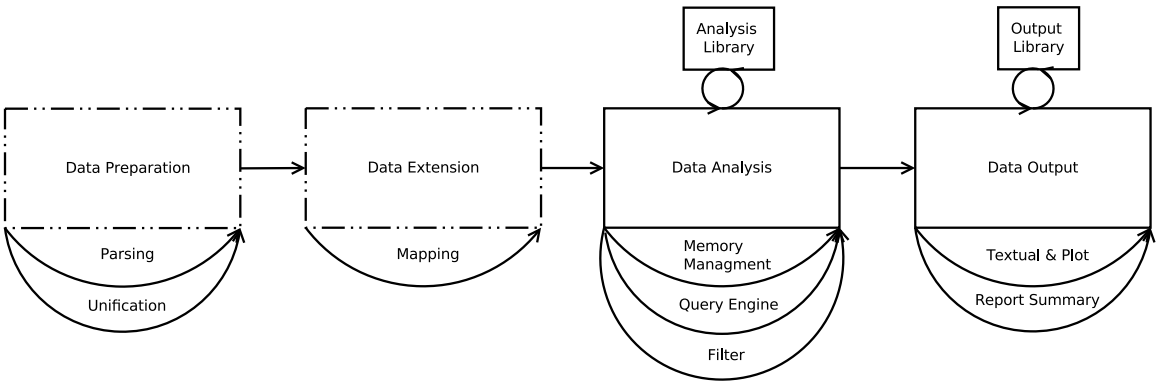


Figure 5.1: Framework architecture and the data flow across components. Dashed components are designed to be executed once, while continuous lines indicate repetitive usage.

analysis library. The library is responsible for the extensibility of the framework as it can be extended by arbitrary amount of analysis functions. After they are defined, they can be specified in the query-list along with an argument in the data analysis. The query-list can contain multiple analysis-functions, which all will be applied to the data chunks while only reading the data once into to the memory. The same query can be used several times with different arguments to allow concurrent analysis, which makes the process of exploring data easier. The analysis engine also supports a filter, which can omit specific data during runtime. Such data might be unwanted for certain analysis functions or has to be disposed because it is clearly distinguished as noise by some new analysis insights.

Data Output The last component is responsible for the output of the results. For each analysis function an output function can be defined in the output library, which configures the format of the output, such as CSV or SVG file types, plot type and dimensions, colours and so on. The files are stored in a directory which is automatically created for each analysis function. This component might be configured to only output the final result but also the intermediate results for specific time period such as specific quarters of a year. A post-analysis script is eventually responsible of merging the various results from the result directories into a comprising report, which makes it possible to view all final results combined in a single PDF-file.

The components of the framework and the direction of the data flow are visualized in Figure 5.1, which differentiates between components which are designed to be called once (or occasionally) and those who are called frequently.

5.2 Implementation

The selection of programming languages, tools and existing libraries is explained in this section. Moreover, relevant code snippets are presented.

5.2.1 Selection of Tools

After a common programming language is found as the basis of the implementation, each tool is chosen with its specific purpose in mind. The tools are marked by their components they are deployed in, that is data preparation, extension, analysis and output, or $\alpha, \beta, \gamma, \delta$ respectively.

Python Each component of the framework can be implemented using its own selection of programming languages and libraries, however, the inter-component communication is easier considering the programming effort and faster if no translating interface is used. Python is an established general-purpose, high-level programming language, hence it can be used for every component. A good documentation and many useful packages and libraries exist, which allow to rather focus on the problem solutions than implementation difficulties, which makes it a very good candidate for the frameworks main programming language.

SQLite -α- SQLite is used to save data homogeneously and unified in one place during the data preparation process. SQLite is the most widely deployed database engine in the world and exceeds especially for read speeds. Interestingly, a singular file is created on the hard drive to store the database, which makes it very clear to the user where which information is stored.

SQLAlchemy -α- SQLAlchemy is an open source SQL toolkit and object-relational mapper (ORM) for Python, which make it possible to perform SQL queries on databases such as insertions while keeping the python syntax. The developer works on python objects and calls the related objects-methods, which then are mapped transparently to SQL commands. Obviously, this library has been used to insert the honeypot data into the SQL databases.

Bgpdump -β- Bgpdump is a library designed to help with analysing dump files produced by routing software suites such as Zebra or Quagga. It creates CSV files, which contain AS-routing information and so on. This information is used to extend our database by mappings of IP-addresses and ASN.

Pandas (Numpy, Matplotlib) -$\beta\gamma\delta$- *R* is maturing into its role as a go-to language for data analysis, however new alternatives have appeared in the last years. One of the promising is the Python library *Pandas* [173], which is partially based on *Numpy* (fundamental package for scientific computing) and *Matplotlib* (plotting library which produces publication quality figures) but extends them by providing high- performance, easy-to-use data structures and data analysis tools. More precisely, it supports DataFrames, which are similar to tables in databases, each column being called Series. All data analysis functions are performed on either DataFrames or Series. Like Numpy, Pandas uses the Python syntax, however on the low level highly optimized C-functions are utilized, which makes this package so interesting if speed is of matter. Not only

the speed of Pandas is of essential role, but also its configurability and robustness in dealing with missing data: all implemented statistical methods function properly despite missing or faulty values. Moreover, as python is used, a familiar syntax is used and the advantages of this language come into play. Because if these reasons it is favoured to *R*.

Pdftk -δ- Pdftk is a cross-platform tool for manipulating PDF-documents. Is is used in the last component to merge the resulting plots from different result directories (resembling an analysis function each) into one summary report.

5.2.2 Coding Details

Each step of the linear data processing by the framework is demonstrated here by a representative code snippet.

5.2.2.1 SQLite database structure

Only the essential information should be stored in the SQLite database. Hence, the timestamp of an attack, its source and target based on the IP-protocol and some OS-fingerprinting information about the attacker are stored. Moreover, a tag indicates which honeypot sensor has logged the event. For that purpose of query speed only one table is used. Although a more complex database schema would obviously save much space, the necessary joins for each analysis would slow down the analysis process. This is a clear space-time tradeoff, and the decision is made in favour of time allowing more queries to explore the data set faster. The one-file-policy of SQLite became a limitation since the files exceeded the maximum (network) file system file size. This was circumvented by creating a database for each quarter of the year. The database/ table are defined and created in Python using SQLAlchemy, as shown by listing 1.

5.2.2.2 Python Parsing

The heterogeneous log files are collected for each sensor in zip-containers representing $12h$ periods. Those zip-containers containing various text-files, binaries, databases are unzipped and parsed by python. File system traversing is done by the *os* module, unzipping by the *zipfile* module and the parsing of textual log files by comparing lines with regular expressions by the *re* module. For database log files SQLAlchemy is used to query the necessary information. Listing 2 gives an example of parsing of textual data with regular expressions in Python.

5.2.2.3 SQLAlchemy Data Insertion

SQLAlchemy is also used to insert data into the SQL databases. Listing 3 shows how a *ConRequest* object is created from the previously parsed data, which now is simply forwarded to *sqlDB_add()*. Each object is added to an insertion-queue, which is emptied after

```
# Database Model - simple tables due to performance reasons
sqlBase = declarative_base()

class ConRequest(sqlBase):

    __tablename__ = 'ConRequest'

    id = Column(Integer, primary_key=True)
    timestamp = Column(DateTime)
    attackerIP = Column(String)
    attackerPort = Column(Integer)
    honeypotIP = Column(String)
    honeypotPort = Column(Integer)
    protocol = Column(String)
    tag = Column(String)
    remoteOS = Column(String)
    remoteOSversion = Column(String)
    remoteLink = Column(String)

# creates sql database and initiates a session, which can be used to add data
def sqlDB_start(DBdir, quarter):

    # create database
    sqlEngine = create_engine('sqlite:///' + DBdir + quarter + '.sqlite', echo=False)

    # create database schema
    sqlBase.metadata.create_all(sqlEngine)

    # start database session
    Session = sessionmaker(bind=sqlEngine)
    sql_session = Session()

    return sql_session
```

Listing 1: SQL-Schema definition and table generation using SQLAlchemy.

committing.

5.2.2.4 Data Extension using Bgpdump

As we want to provide the possibility of an analysis based on AS information, the data set has to be extended by mapping IP-address to ASN. However, this is not a trivial problem as IP-addresses can change their affiliation to AS. After the gathering of old routing data from RIPE Bgpdump was used with a precision of a quarter of the year to dump prefixes and their AS. As a matter of fact, the dumps contained whole AS-routes for specific prefixes, however the last entry contains the owner of this prefix. This information was retrieved by utilizing

```
# RegEx example
regex = re.compile(
        r"""^\[
        (?P<timestamp>\d{4}-\d{2}-\d{2}\s\d{2}:\d{2}:\d{2})\]\s+
        (?P<src_ip>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\:
        (?P<src_port>\d+)\s+requesting\s+
        (?P<protocol>udp|tcp)\s+connection\s+on\s
        (?P<dst_ip>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\:
        (?P<dst_port>\d+)
        \.$""",
        re.VERBOSE)

# get all honeytrap log files $
honeytrap_tag = "honeytrap"
ht_filelist = [ f for f in os.listdir(tmp_dir) if honeytrap_tag in f]
for file in ht_filelist:
    with open(tmp_dir+file) as f:
        print "--- [Honeytrap]    ", file
        for line in f:
            match = regex.match(line)
            if match:
                # collect data from attack log lines
                result.append([
                    datetime.strptime( match.group("timestamp"),
                        '%Y-%m-%d %H:%M:%S'),
                    match.group("protocol").upper(),
                    match.group("src_ip"),
                    # and so on...
                ])
```

Listing 2: Parsing of text-based files with Python using regular expressions.

pandas converters which allow data manipulation during reading from a CSV-file. A prefix-tree was created in the following to allow lookups for IP-addresses. Eventually, for each spotted IP-address the related AS was mapped for each quarter year of the measurements. Compare listing 4.

5.2.2.5 Memory Management with Pandas

One of the challenges of the analysis framework is its memory footprint, which has to be adjusted for the machine it is running on. Two possibilities exist.

The SQL-based solution utilizes the keywords *LIMIT, OFFSET* to retrieve partial results which fit into the main memory. Using these keywords only lines within the *OFFSET→LIMIT* range are considered from the result set. However, this information retrieval proved to be not efficient enough, because higher offsets slow the query down. Since every

```
# receives a list of new honeypot attacks and adds them to the sql database
def sqlDB_add(sql_session, honeypot_tag, attacks):

    for atk in attacks:

        # create request object/line in db
        tmp_request = ConRequest(
            timestamp = atk[0], protocol = atk[1],
            attackerIP = atk[2], attackerPort = atk[3],
            honeypotIP = atk[4], honeypotPort = atk[5],
            remoteOS = atk[6], remoteOSversion = atk[7],
            remoteLink = atk[8], tag = honeypot_tag)

        # append data to next insert
        sql_session.add(tmp_request)

    # actually commit to the database
    sql_session.commit()
```

Listing 3: Inserting data to an SQL-database using SQLAlchemy.

```
# read only relevant columns from dump, get last element of AS route info
bgbDF = pd.read_csv(bviewFile, sep="|", engine='c',
            header= None,
            names = ["info", "timestamp", "flag", "collector",
                        "collectorAS", "prefix", "AS", "protocol"],
            usecols = ["prefix", "AS"],
            converters={"AS": lambda x: x.split(' ')[-1]})

# create tree structure for efficient lookups
prefixTree = subt.SubnetTree()
for prefix, AS in zip(bgbDF.index.tolist(), bgbDF.tolist()):
    prefixTree[prefix] = AS

# check for longest matching prefix in prefixTree / CIDR
prefixTree[ip]
```

Listing 4: Creation of an IP-prefix tree and lookups.

query needs to count off the first *OFFSET+LIMIT* records and take only *LIMIT* of them. This means, the higher this value is, the longer the query runs. Some workarounds exist including sorting and the lazy evaluation and exclusion of IDs, however alternatives were considered to keep the implementation as simple as possible.

The second possibility is Pandas-based. Pandas read-functions provide a *chunksize* parameter, which allow reading data in chunks of a specific size. As internally file pointers are used, lines are not recounted for consecutive reads, which ensures a constant read time for chunks of the same size for multiple reading operations. However, this method works only

for text-based files, therefore CSV dumps of the SQL-databases have to be created once, which is especially easy as only one table exists. Moreover, this data manipulation step/ abstraction can be used to filter data which will definitely not be required and sort the data in a way which is convenient, for example by the timestamp. This chunk-based reading from CSV files resulted in an efficient data retrieval which was 3-times faster than the original SQL-based version and offered an useful abstraction.

Especially because of the speed improvements the Pandas-based solution was implemented, as shown by listing 5. The amount of data which is loaded is determined by the pagesize variable and is configured by the *pFac* factor. The pagesize is usually between 20 and 30 millions lines depending on the amount of columns and number of analysis functions. More lines force the OS to transfer data to the file swap file on a 16GB RAM machine, which was determined empirically for the available data.

During first runtime tests, the garbage collector of Python proved to be very conservative. Even after pointers to large dataframes have been set to *null*, the data which was referenced remained for a relatively long time in the memory. This led to 2 concurrently existing chunks in the memory, one being referenced and one unreferenced, which caused the memory to overflow. Because of this reason the garbage collection is triggered manually after all analysis functions have been applied.

```
# expects a csv file and gets information for specified columns
# paginates automatically (in chunks of pagesize)
def loadData(con, columns, pagesize=1000000):

    if 'timestamp' in columns:

        return pd.read_csv(con, sep=";", chunksize=pagesize, usecols=columns,
                    engine='c', parse_dates=['timestamp'],
                    infer_datetime_format=True)
    else:
        return pd.read_csv(con, sep=";", chunksize=pagesize,
                    usecols=columns, engine='c')

pagesize = (int)(1000000 * pFac)
for chunk in loadData(db, colQuery, pagesize=pagesize):

    # analyse chunk and save results...
    chunk = None
    gc.collect()
```

Listing 5: Loading of Data Chunks with Pandas respecting the maximum available memory.

5.2.2.6 Filter Attack Events using Pandas

Events like administrative communication, data transfers, unwanted misconfigurations should be cleared using a filter. Pandas dataframes offer a convenient way of filtering a dataframe

based on multiple conditions, as demonstrated by listing 6 with a trivial example.

```
# define filter arguments
filter_from = ["160.45.114.26"]
filter_to = ["255.255.255.255", "0.0.0.0", "224.0.0.251"]

# filter on specific columns (drop specific values)
chunk = chunk[ ~chunk.attackerIP.isin(filter_from) &
        ~chunk.honeypotIP.isin(filter_to) ]
```

Listing 6: Filtering lines in a Pandas Dataframe.

5.2.2.7 Query Engine based on Pandas

The query engine applies analysis functions to the data chunks and merges the intermediate results. These analysis functions are defined in the analysis library and since the data chunks are dataframes, these functions are defined as Pandas Dataframe operations. Every library function is required to return a dataframe (or series), as merging functions are also defined to combine pandas objects. The default operation for merging is a simple *DataFrame.add()*, which sums up integers and concatenates strings for cohesive indexes, however non-default operations can be defined for each function if necessary.

The administration of intermediate results is done in a pure pythonic way using nested dictionaries, which manages the results categorized by honeypot sensor and analysis function. A two-layered approach is selected here, which first merges all results for one specific database (a SQL-file representing a quarter of the year in this specific case) and then merges the total results of each database into a final result.

The query engine uses time checkpoints to calculate the execution duration for analysis function, so that if queries are slow, the function which causes the bottleneck can be easily spotted. Extracts of the query engine are provided by listing 7.

5.2.2.8 Plotting and CSV-Creation with Pandas

The plotting of dataframes and printing results to PDF/CSV-files etc. narrows down in Pandas to a simple function call. However, as matplotlib is used by pandas, the plots can be extended by its numerous configuration options, as shown in listing 8.

5.2.2.9 Summary Report by Pdftk

The final summary report which is created by combining various plots in the PDF-format is accomplished by traversing directories and calling system commands with python: *os.system(cmd)*, where *cmd* is a Pdftk command.

```
# calls function on all available databases and merges the temporary results
def mergeCall(databases,funcList,colListDict,argList,htags,pFac=10,quadPrint=False):

    # count how many functions will be applied
    n = len(funcList)
    # create a total result dict for each function for each honeypot
    rList = [{k:v for k, v in zip(htags,[None]*len(htags))} for _ in range(n)]
    # find all necessary columns for the data chunk
    colQuery = np.unique(np.hstack( colListDict.values() ))

    # (...)
    for db in databases:

        # (...), create intermediate result dictionary set
        qList = [{k:v for k,v in zip(htags,[None]*len(htags))} for _ in range(n)]

        for chunk in loadData(db, colQuery, pagesize=pagesize):

            # (...), split chunk by honeypots and apply each function
            chunk = chunk.groupby(['tag'])
            for name, group in chunk:
                for i in range(n):

                    # call function with arguments only for necessary columns
                    columns = colListDict[funcList[i]]
                    arguments = argList[i]
                    i_result = funcList[i](group[columns], arguments)

                    # save or combine intermediate results for current db-file
                    if qList[i][name] is None:
                        qList[i][name] = i_result
                    else:
                        qList[i][name] = add_append(funcList[i],qList[i][name],i_result)

            # (...)
        # (...), add intermediate qList to total result set rList for each database
```

Listing 7: Extract of the Query Engine showing the application of various functions and the merging of intermittent results. Some lines omitted for better a overview.

```
# plotting n-elements of a normalized Series
plt.figure()
plot = sr.div(sr.sum())[:n].plot(kind='barh', color='b', alpha=0.5)
plot.get_xaxis().tick_bottom()
plt.grid(True)
# (...)
```

Listing 8: Plotting with Pandas, extending with Matplotlib arguments.

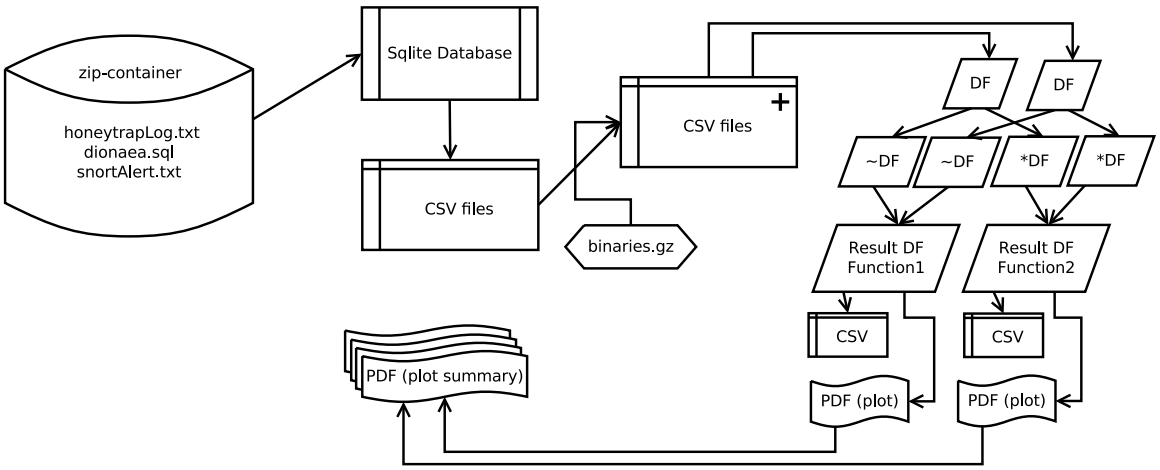


Figure 5.2: Linear data transformation performed by the framework with a view on the data types.

5.3 Illustration of Data Transformation

In conclusion of this chapter, a summarizing illustration shall be presented, which visualizes the different data transformation steps with respect to the design architecture and the actual implementation. The linear modifications of the four components as well as the result merging and the different data types are shown in more time detail in Figure 5.2.

CHAPTER 6

Experiments and Results

This chapter describes the experiments which were performed in order to collect the honeypot data. The data analysis of the acquired data set from the first Section 6.1 is then conducted in the following sections. Each analysis step has one guiding question based on the extensive survey in Chapter 4. Each subsection will introduce the problem, present the methodology and then show the results.

6.1 Acquisition of Honeypot Data

The process of data acquisition is depicted in this section. The deployed honeynet is described with focus on the configuration of a single honeypot and its interaction with the honeynet, as well as the duration of the measurements. This deployment sets itself apart from previous research by deploying a heterogeneous honeynet with respect to the network access type. Four different network access types were deployed. Each honeypot has a specific access to the Internet, such as DSL (Digital Subscriber Line) or UMTS (Universal Mobile Telecommunications System).

6.1.1 Honeynet Architecture

In total, 4 honeypots have been deployed as described in Subsection 6.1.2, each representing a different network connection type. The honeypots are identified by a 3-4 character ID:

UMTS The honeypot in the mobile network. A (3G) UMTS modem device is used to establish a connection with the Internet and configured to reconnect if the connection disconnects for any reason. The IP-address is allocated dynamically.

DSL Common home DSL network. A home router is configured to forward all incoming connections to our honeypot. The IP-address is obtained dynamically by the router.

BCIX This honeypot is situated in the network of the Berlin Internet Exchange and Peering Point (BCIX) and is listening on dark, i.e. unused or not announced, address-space of the network for incoming requests. The IP-address is assigned statically.

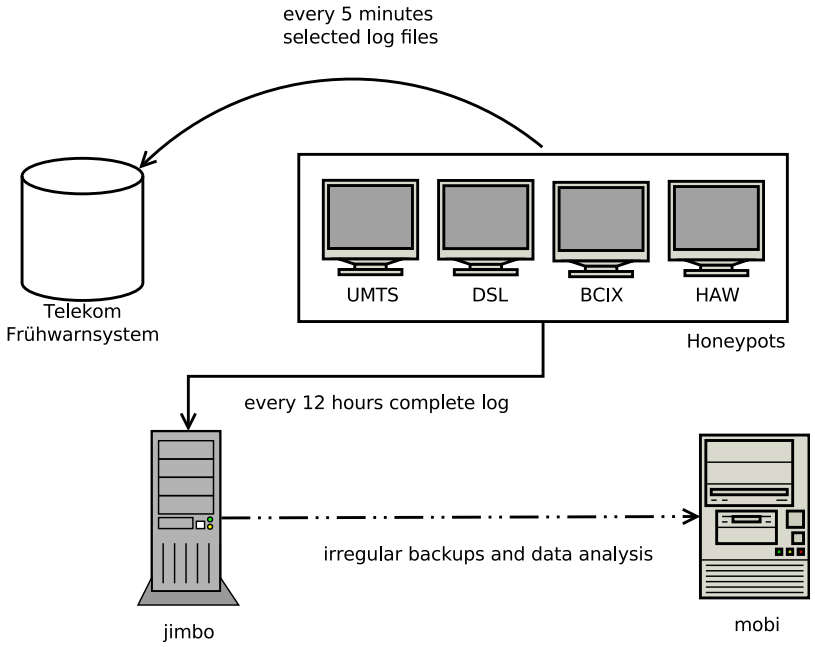


Figure 6.1: Honeynet Architecture and associated devices.

HAW A honeypot within the network of the Hamburg University of Applied Sciences (HAW), which is a well-known network with many announced services. A static IP-address is set.

Special precaution has been taken in the design of the UMTS and DSL honeypot, as it was necessary to choose an ISP which allows incoming requests on all ports and permanent connectivity. It has been verified, that this is the case by full-range *nmap* port scans and long-term connectivity tests.

All honeypots send their complete log files every 12 hours to a centralized server (*jimbo*) administered by us. Moreover, a selected portion of the logs is sent to Telekoms EWS Frühwarnsystem. Due to the slow nature of UMTS devices, the UMTS honeypot has a second interface with a higher bandwidth, which is only used to send log files, incoming connections are not logged by the honeypots. These data transfer processes happen automatically as cronjobs.

At irregular intervals the data from jimbo is moved manually to a backup- and data analysis machine called *mobi*. This step introduces another assurance, as mobi provides a more reliable backup storage and the CPU/RAM-intensive computations do not interfere with the actual data collection from the sensors.

Figure 6.1 shows the honeynet architecture in a simplified manner and visualizes the data flow.

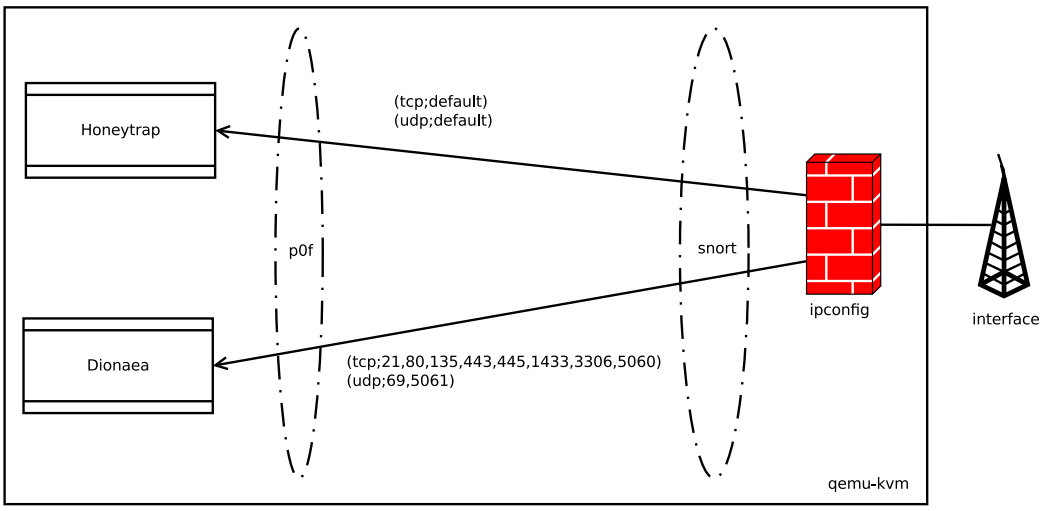


Figure 6.2: Low-interaction server honeypot setup which was deployed for data aggregation.

6.1.2 Honeypot Setup

Considering that the focus lies on finding similarities and differences between attacks on honeypot with various network access types, a low-interaction server honeypot setup is chosen as it suffices the demands. This is because the access type is a network property, which is independent from the operating system, which uses the interface to access the network. The necessary network-based informations are captured completely by low-interaction honeypots while keeping a low administrative demand.

A single honeypot consists of the following software. Dionaea is configured and bound to every service it supports. Dionaea was chosen because of its state-of-the-art emulation and the active community as well as development process. Honeytrap is utilized to answer incoming requests on all remaining ports. The advantage of honeytrap is, that listening sockets will be bound dynamically to ports, which saves some resources while still listening on the complete port range. Two additional tools are used to extend this setup: p0f fingerprints the operating system of the attacker and Snort alerts, if known attacks and signatures are observed. The honeypot is extended by several scripts which check periodically as a cronjob if all the applications are running correctly.

Although this setup resembles a low-interaction (server) honeypot and therefore no contamination can be expected, the honeypot is configured as a virtual machine image using qemu-kvm. Multiple reasons support this decision. As several honeypots are deployed, the image ensures the equality across different deployments and makes the process of updates easier, as it is abstracted to the to exchange of one image file. Moreover, if the unlikely case of contamination or outage happens, the honeypot can be easily reset. Qemu-kvm was chosen as the virtualization software as it provides a good support for bridging and passing through of different network devices, which is necessary if different network access types should be analysed.

Initial insights gathered by this setup with focus on the operation and evaluation of the mobile honeypot has been introduced by Wählisch [169, 170] in 2012.

The setup was updated to recent software and signatures versions during the creation of this thesis, however this action shall not be subject of this document, as it does not give any additional insights.

Figure 6.2 visualizes the honeypot setup.

6.1.3 Duration of Measurements

The measurements for the honeypot sensors do vary in their duration. Although the honeypots have been active simultaneously for a long period of time, slight differences exist and some sensors are not active any more. Table 6.1 gives an overview about the different honeypot sensors and their deployment time. In the following chapters the time range between December 2011 and March 2013 is referred to as the mutual runtime. The idea behind such long-term deployments of honeypot sensors is that the impact of failures for a specific sensor will smooth out the longer the measurements are. Comparing Table 6.2, one can spot that such long-term honeypot measurements are rare. Although quite a few honeypot research paper exist, only some have conducted own measurements and did not reuse available data.

Sensor	Begin	End
UMTS	December 2011	August 2014
DSL	December 2011	March 2013
BCIX	December 2011	*ongoing*
HAW	December 2011	*ongoing*

Table 6.1: Duration of measurements for all honeypot sensors.

Duration	Related Work
24 hours	[168]
1 Week	[133, 135]
2 Weeks	[136, 155]
1 Month	[58, 174]
3 Months	[101, 175, 176]
4 Months	[165, 177, 178]
5 Months	[164, 137, 6]
6 Months	[1]
10 Months	[132]
1 Year	[150]
3 Years	[134]
4 Years	[2]

Table 6.2: Honeypot measurement duration present in related work.

6.1.4 Data Set Properties

The overall size of the raw data set for all honeypots combined without any compressions amounts to 10TB. This is due very verbose logfiles with debug informations. However, the raw data set is stored in zip-containers each equivalent to 12 hours of logging, which amount to 626GB of data.

The data preparation step of the framework reduces the data into a 63G Sqlite database. The dumping of the databases into CSV-files results into a 60GB CSV file set. Table 6.3 shows the CSV-filesize per quarter of the year.

This set is extended by 1GB of session information and 10GB of routing information and autonomous system mapping. Further explanations why these extensions are required and interesting follow in the analysis chapter. However, it is important to note that the AS mapping considers the historical affiliation of an IP-addresses to an autonomous system. That means, that if an IP-addresses changes the affiliated AS, then the extension will incorporate this change with a precision of 3 months. If a mapping is not unambiguous for a specific time frame, which is caused mainly by multiple-origin autonomous systems (MOAS), then the configuration is set to choose the AS with the smaller ASN in order to ensure consistency. For all quarters the share of MOAS mappings accounted approximately 0.06. Table 6.4 counts, how many IP-addresses had 1 or more different AS mappings given the resolution of 3 months. Therefore, it indicates the stability of the IP-addresses to autonomous system affiliation. The AS mapping did not change for most observed IP-addresses.

Quarter	Datasize
2011Q4	436M
2012Q1	3,5G
2012Q2	7,8G
2012Q3	18G
2012Q4	7,9G
2013Q1	6,4G
2013Q2	5,6G
2013Q3	3,2G
2013Q4	3,2G
2014Q1	2,4G
2014Q2	767M
2014Q3	1,3G
2014Q4	937M

Table 6.3: CSV log file size, quarterwise.

ASNs	1	2	3	4	5	6	7	8	9
IPs	504471	65806	15843	8946	12318	2840	165	17	1

Table 6.4: Histogram - ASN mappings for IP-address.

6.2 Attack Sources

The first step in the analysis is to investigate who actually attacks the honeypot sensor. In particular, how many unique attackers exist and how the distribution of their events looks like.

6.2.1 Methodology

The identification of the attackers is done on two granularities. We identify the attackers based on two topological identities, the IP-address and the AS number. The the IPv4-address of the attacker is directly collected by the honeypots. The IP-address is independent from any application service and therefore an universal identifier, as it is based on the network layer, which makes it suitable for identifications of attackers on every port and different network interfaces. The second identification method, ASNs, are derived after the measurements as they were not stored be the honeypot setup. The data for the ASN identification is gathered during the extension process in the framework, which maps IP-prefixes to a ASN based on historical routing information from RIPE [179]. This identifier is used because there is clear lack in the honeypot research concerning its application. Furthermore, new insights are anticipated from the more coarse overview, which might eliminate some misleading details. Moreover, the attack sources are also depicted by their opearting system using the data collected by the p0f fingerprinting tool. The description of attack sources based on the operating system is only used in this sub-analysis, those information are not combined with other methods.

As already discussed in Section 4.1, IP-addresses are subject of spoofing. The validity of the attack source is a huge challenge for server honeypots, especially in retrospective. The honeynet did not have any anti-spoofing mechanisms installed. One possible idea to get a feeling for the level of spoofing might be the counting of martian packets on a honeypot sensor. Martian packages possess source addresses within special-use ranges [180]. However, the kernel of the operating system has to be configured to not automatically drop such packets. Since this setup was not configured to do so, this test could not have been completed.

This section performs a time-independent analysis. This means that an overview on total or most-active attackers, averages, ratios and so one are given without discussing specific frequencies (which is done in the following sections). In particular, the following properties are:

- Number of unique IP-addresses and ASN
- Number of observed unique IP-addresses per ASN

- Number of Requests for IP-addresses and ASN
- Request Fraction of Top Attackers (IP and ASN)
- Operating System

6.2.2 Results

	UMTS	DSL	BCIX	HAW
All Requests	1591855	949344	11157451	532986410
Unique IP-addresses	131790	45140	129469	337811
Ratio unique IP-addresses/ Requests	0.083	0.048	0.012	0.001

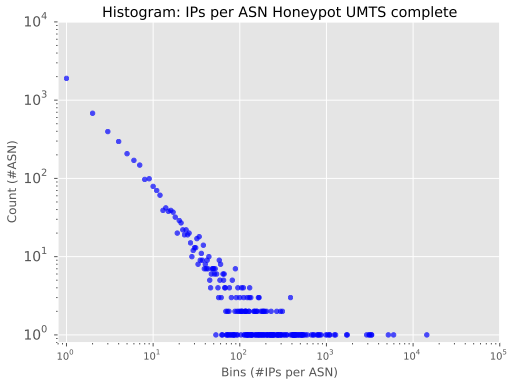
Table 6.5: Honeypot attackers in total numbers based on IP-address.

	Requests				IP-Addresses			
	UMTS	DSL	BCIX	HAW	UMTS	DSL	BCIX	HAW
count	5161	4263	7438	7230	5161	4263	7438	7230
mean	208.5	223	1500	73718.5	25.5	10.5	17.5	46.5
std	4216	1290.5	38654.5	1193850	259	64.5	269	580
max	263011	43884	3159247	62896795	14393	2972	18354	33180
$\max_{\text{ratio}}$	0.165	0.046	0.283	0.118	0.109	0.066	0.142	0.098

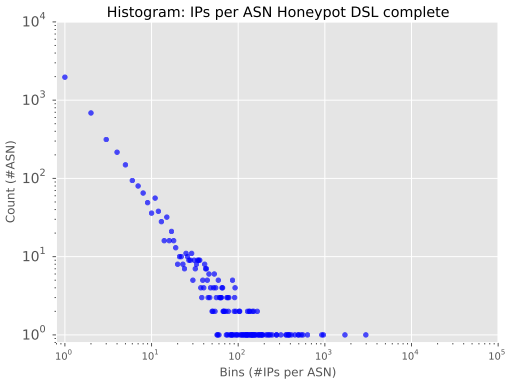
Table 6.6: Observed activities from AS.

A comparison of honeypot attackers in total numbers based on the IP-address is shown in Table 6.5. Immediately, a severe difference in the number of requests is recognizable, which is not explicable only by the different runtime of each sensor as the largest difference (DSL to HAW) is determined by the factor 500. The UMTS and DSL sensor have similar amount of requests, BCIX has a little more requests and HAW exceeds by far. The numbers of unique IP-addresses follow a different pattern. Although having much more requests, the HAW sensor observed only twice as much unique IP-addresses as the other sensors. UMTS and BCIX sensor are on the same level, DSL observed again the least. This result reveals that only a small amount of unique IP-addresses exists in comparison to the number of total events, as shown by the ratio. Furthermore, this result might indicate that the increase in the number of requests does not trigger a linear increase in the number of unique IP-addresses or that each node has its own characteristic ratio between between those two metrics.

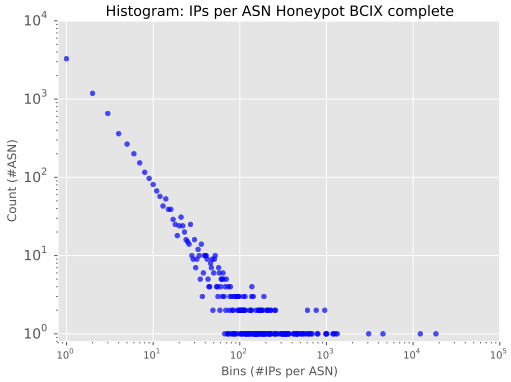
Table 6.6 shows the number of total requests and number of observed IP-addresses from the AS. Considering only the total amount of observed ASNs, the values range from 4263 (DSL) to 7438 (BCIX). Currently around 50000 autonomous systems are registered [181], which means that the BCIX sensor has been contacted from almost 15% of all possible AS at least



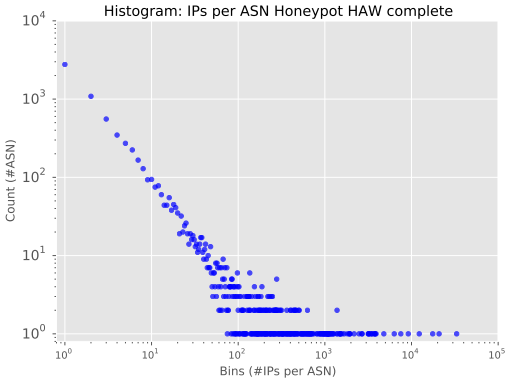
(a) UMTS



(b) DSL

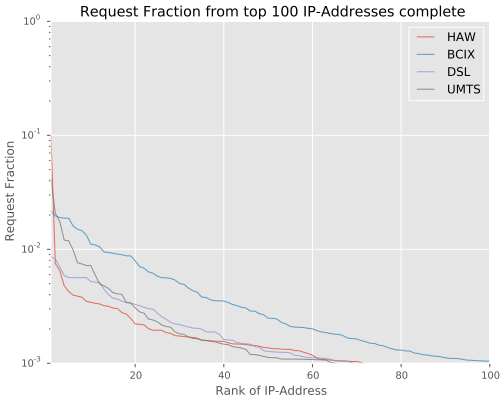


(c) BCIX

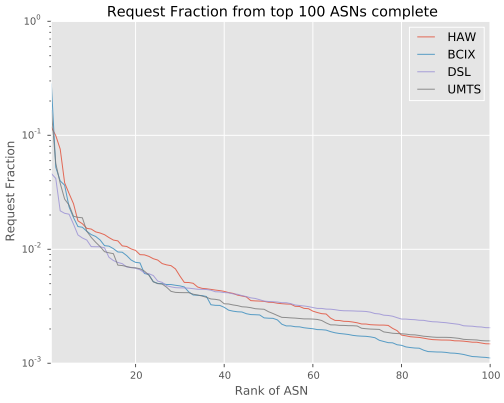


(d) HAW

Figure 6.3: Histogram: Number of IP-addresses observed from autonomous systems.



(a) Top 100 IPs



(b) Top 100 ASNs

Figure 6.4: Request Fraction of top attackers based on IP-address and ASN.

once. Interestingly, BCIX was the second least if it comes to unique IP-addresses, however exceeds at unique ASNs. Noticeable is also that the UMTS sensor was contacted from more than 5000 ASNs, although having a significantly shorter lifetime. The AS with most requests is responsible from 4% (DSL) of requests up to 28% (BCIX) of requests, compare $\max_{\text{ratio}}$. Moreover, the AS with the most unique IP-addresses holds a large portion of all spotted unique IP-addresses, the BCIX sensor owns relatively the AS with the most unique IP-addresses. The ratio for the unique IP-addresses does not deviate from each other as strong the ratio for requests. Furthermore, the AS with the most requests is not necessarily also the one with the most unique attackers.

As already indicated by table Table 6.6, investigations in more detail show that the most active attackers are responsible for a large fraction of requests. Figure 6.4 demonstrates, that this holds especially true for the HAW sensor if IP-addresses are considered. Despite this, the fraction for an attack whether identified by an IP-address or ASN shrinks very fast and a long-tail distribution can be spotted. In general, the fractions for the AS-based observations are higher than the IP-based, which makes totally sense as every AS holds at least one IP-address, therefore the fraction is equal or higher if more IP-addresses exist. In order to visualize how many AS actually have one or more IP-addresses, the histograms for Figure 6.3 were created. One can see, that the most common relationship between an ASN and IP-address is 1:1 and that the count is almost steadily decreasing. The sensors reveal a very similar distributions. This visualizations reveals, that the previously found ASN with most request are very rare and outsiders.

Most attacks originate from Windows-based operating systems, as is demonstrated by Table 6.7. Another common characteristic for all sensors is that Linux-based attack origins are usually very seldom, which is indicated by their high position in this table. However, the UMTS and DSL sensors show a similar trend with Linux on the third position accounting for about 10% of connections. This value is significantly lower for the other two sensors. Other operating systems such as FreeBSD have been observed, however they were exceptions.

Sensor	Rank	Fraction	p0f Signature (OS)
UMTS	1	0.65	Windows 2000 SP4, XP SP1+
	2	0.10	Windows 2000 SP4, XP SP1+ (2)
	3	0.07	Linux 2.6 (newer, 3)
DSL	1	0.39	Windows 2000 SP4, XP SP1+
	2	0.34	Windows 2000 SP4, XP SP1+ (2)
	3	0.11	Linux 2.6 (newer, 3)
BCIX	1	0.61	Windows XP/2000 (RFC1323+, w+, tstamp-)
	2	0.31	Windows 2000 SP4, XP SP1+
	...	...	...
	6	0.009	Linux 2.6 (newer, 3)
HAW	1	0.75	Windows 2000 SP4, XP SP1+
	2	0.08	Windows 2000 SP2+, XP SP1+ (seldom 98)
	...	...	...
	18	0.00001	Linux 2.6 (newer, 3)

Table 6.7: Top Operating system attack sources and first Linux-based occurence.

6.3 Attack Targets

The second analysis raises the question which targets were chosen by the attackers. In general, what are the most popular targets for different types of events?

6.3.1 Methodology

The target is identified by the port and the transport layer protocol, which seems reasonable since low-interaction server honeypots were deployed. However, as the traffic to the honeypot sensors was not captured completely including the data dumps, it cannot be verified whether specific service probes has been sent to uncommon ports, such as a SSH-request on port 2222 and so on.

This analysis is based on all observations. A perspective on total numbers which does not consider the frequency might be misleading to some extent because of temporary misconfigurations which cause much traffic on only one port for a short period of time, yet it provides first insights on what is actually happening on the honeypot sensors. Moreover, the results are expressed as fractions as far as possible to allow a better comparability, as the numbers of requests differ severely between the honeypot sensors which has been shown in the preceding section. The most attractive targets are initially determined by pure counting of request. However, a single attacker might perform several requests on one port if this port is of vital importance for his attack, at the extreme he might be the only attacker of one port with many counted requests, which therefore might seem attractive to us. In order to eliminate this problem the attractiveness of a port is also compared by counting the unique attackers. Similar to the last chapter, a distinction between attacks on ports from IP-addresses or AS is performed. Ports are divided into (common) ports and well known ports. Port numbers range from 0 to 65536, but only ports numbers 0 to 1024 are reserved for privileged services and therefore designated as well-known ports. Well-known ports can only be bound by the root on Linux machines, which is why they are sometimes called root ports. Most important and popular services are reserved by the IANA to the well-known port range, therefore it is interesting to check if honeypot traffic also concentrates on this range.

The most popular attack targets are determined by sorting by the following properties:

- Fraction of requests on port (relative to all requests)
- Fraction of requests per protocol on port (relative to requests via protocol)
- Fraction of unique attackers per port (relative to all unique IP-addresses)
- Fraction of unique AS per port (relative to all unique AS)

6.3.2 Results

The results in Table 6.8 demonstrate evidently, that most attacks occur using by TCP. However, the BCIX sensor is an exception here, as this sensor receives more UDP packages. As investigations in more details reveal, this is attributable to many requests on ports (more precisely services) which are vulnerable to amplification attacks such CHARGEN (19) or

	UMTS		DSL		BCIX		HAW	
total requests	1591855	—	949344	—	11157451	—	532986410	—
TCP requests	1447327	0.91	878518	0.93	4685621	0.42	531747081	0.997
UDP requests	44528	0.09	70826	0.07	6471830	0.58	1239329	0.003
w.k. port requests	359717	0.22	136317	0.14	6453062	0.58	530013655	0.994
ports (TCP)	10608	0.16	1369	0.02	65535	1.0	65535	1.0
ports (TCP only)	10250	0.15	1162	0.018	57422	0.88	63407	0.97
ports (UDP)	1213	0.02	1386	0.021	8114	0.12	2129	0.03
ports (UDP only)	855	0.01	1179	0.018	0	0.0	0	0.0
ports (TCP&UDP)	358	0.005	207	0.003	8114	0.12	2129	0.03
w.k. ports	1019	0.99	247	0.24	1024	1.0	1024	1.0
total ports	11463	0.17	2548	0.04	65535	1.0	65535	1.0

Table 6.8: Number of requests on ports and protocol-wise distribution. Fraction refers either to total requests, all ports or well-known (w.k.) ports respectively.

SSDP (1900).

Another learning from this table is that well-known ports receive a comparatively large amount of traffic. 0.015 of all ports are well known ports, if ports were attacked by random, this distribution should be reflected in the request number. However, well-known exceed this number by several magnitudes, especially the HAW sensor receives most of its traffic on well-known ports. Noticeable is also the fact, that the UMTS sensor only received attacks on 0.17 of possible ports, however on almost all (0.99) well-known ports. The DSL sensor is the only sensor with a small amount of attacked ports, which might be due to the short life time of this sensor.

Table 6.9 highlights the most popular targets by the means of requests and unique attackers. Considering the requests, all sensors exhibit a similar behaviour by following an exponential distribution. The first port usually is far ahead of the subsequent top ports. However, the degree varies: The HAW sensor is the most extreme with 85% of all attacks designated to port 445 (SMB), BCIX the least with 22% (CHARGEN). This situation holds also true for protocol-wise observations. For the UMTS and the DSL sensor the decline is very similar for TCP and UDP, for the other sensors two different behaviour between TCP and UDP can be observed: the TCP-based decline is for the HAW sensor very steep, but not as much for UDP. The BCIX sensor experienced the other way around, having a more flatten curve for TCP.

With regard to unique attacking sources, a similar exponential behaviour can be spotted if unique IP-addresses are considered. Interestingly, the AS-fractions are linear for about the first top 5 values, after that a fast exponential decrease can be observed. Remarkable are here the HAW and UMTS sensor, which reach 68% and 50% for the top attacked port, respectively. Please note, that the sum of the fractions does not equal to one but exceeds one. This is an indication, that attackers make requests on more than one port. Especially if the AS-based perspective is applied, the numbers are notably larger. This seems plausible, as the AS-perspective is more general than IP-addresses.

Comparing the different metrics for finding the most attractive targets, the following can be

deduced. First, there is a clear difference between the protocols (TCP/UDP), that means that ports are contacted by the protocol which is expected by the common service for the particular port (TCP for SSH etc.). Second, comparing the unique sources (IP/ASN), it is obvious that the rankings undergo only a minimal change, usually by a single rank. One reason for this observation is that often only a single IP-address has been observed from 1 ASN (compare Table 6.15), however further investigations are required only for ASNs with more than 1 IP-address to eliminate errors caused by an overall perspective. Right now, no difference between those two granularities exists, which might indicate that attackers (IP-addresses) are spread out uniformly across ASs. Third, comparing the requests and unique sources columns, one can spot the rankings are similar for many ports, however some ports only appear in one of the metrics. This means, that ports exist, which have many requests and a small amount of unique attackers, and vice versa, ports exist which have many different attackers but a smaller amount of requests (each unique attacker causes at least one request, therefore this difference is not that drastic). Examples are the BCIX sensor, port 139 for the first finding and UMTS sensor, port 8080 for the second finding.

Sensor	Rank	Requests						Unique Sources			
		Total		TCP		UDP		IP		ASN	
		Port	Frac	Port	Frac	Port	Frac	Port	Frac	Port	Frac
UMTS	1	45783	0.42	45783	0.45	53	0.54	45783	0.68	45783	0.5
	2	55670	0.18	55670	0.19	19	0.19	55670	0.29	55670	0.4
	3	22	0.12	22	0.12	5060	0.12	23	0.075	3389	0.25
	4	53	0.05	1433	0.02	2842	0.025	3389	0.03	22	0.18
	5	1433	0.02	143	0.02	45783	0.012	22	0.025	23	0.16
	6	19	0.02	3306	0.02	14852	0.01	5060	0.02	80	0.1
	7	143	0.02	2842	0.018	55670	0.009	8080	0.02	5900	0.09
	8	3306	0.02	110	0.015	123	0.005	80	0.015	8080	0.09
	9	2842	0.018	23	0.014	161	0.004	2842	0.01	5060	0.08
	10	5060	0.017	3389	0.013	39455	0.002	1433	0.008	2842	0.08
DSL	1	6881	0.58	6881	0.58	6881	0.52	6881	0.37	6881	0.36
	2	22	0.11	22	0.12	51099	0.11	23	0.10	3389	0.25
	3	8435	0.05	8435	0.06	28694	0.075	3389	0.08	51099	0.18
	4	1433	0.05	1433	0.06	5060	0.07	51099	0.07	22	0.17
	5	51099	0.035	51099	0.03	46011	0.06	22	0.04	23	0.14
	6	28694	0.02	28694	0.03	1058	0.026	28694	0.03	46011	0.11
	7	23	0.01	23	0.02	4851	0.02	5060	0.025	28694	0.09
	8	110	0.01	110	0.02	31887	0.01	46011	0.025	80	0.08
	9	3389	0.01	3389	0.015	11907	0.01	4899	0.02	8080	0.08
	10	25	0.01	25	0.01	51309	0.01	8080	0.02	4899	0.07
BCIX	1	19	0.22	22	0.13	19	0.37	1900	0.17	36284	0.33
	2	53	0.15	139	0.1	53	0.25	1433	0.14	1900	0.27
	3	1900	0.15	5900	0.04	1900	0.25	3389	0.08	3389	0.25
	4	22	0.06	110	0.04	5060	0.07	23	0.08	22	0.21
	5	139	0.04	3306	0.04	123	0.03	36284	0.07	80	0.16
	6	5060	0.04	1433	0.03	161	0.001	22	0.07	23	0.16
	7	123	0.02	137	0.03	33437	0.001	5060	0.05	1433	0.15
	8	5900	0.02	25	0.03	1300	0.001	8080	0.04	445	0.15
	9	110	0.02	3389	0.02	36284	0.001	3306	0.04	8080	0.15
	10	3306	0.02	1900	0.02	33441	0.001	19	0.04	5900	0.12
HAW	1	445	0.85	445	0.85	19	0.41	445	0.81	445	0.68
	2	139	0.15	139	0.15	1900	0.19	139	0.32	139	0.58
	3	22	0.005	22	0.005	161	0.11	80	0.18	80	0.25
	4	80	0.001	80	0.001	123	0.1	1900	0.03	3389	0.22
	5	19	0.001	25	0.001	53	0.07	3389	0.03	1900	0.18
	6	53	0.001	636	0.001	5060	0.04	23	0.02	22	0.17
	7	25	0.001	2701	0.001	111	0.03	19	0.02	23	0.12
	8	161	0.001	1283	0.001	15	0.02	22	0.02	5090	0.11
	9	636	0.001	5900	0.001	1434	0.01	8080	0.01	8080	0.09
	10	2701	0.001	53	0.001	27960	0.01	5060	0.01	19	0.08

Table 6.9: Top attacked ports by requests and unique sources measured by respective fraction.

6.4 Attack Frequency

This subsection introduces the first analysis which includes the time. The previously metrics were presented in total numbers for the whole observation period. Now the goal is to find out how those events occurred over time.

6.4.1 Methodology

In order to describe the temporal characteristics of the attacks and to allow a better understanding of what is actually happening on a sensor, a combination of different time series is chosen:

- TCP Requests per Day
- UDP Requests per Day
- Unique Attacked Ports per Day
- Unique IP-addresses per Day
- Unique ASNs per day

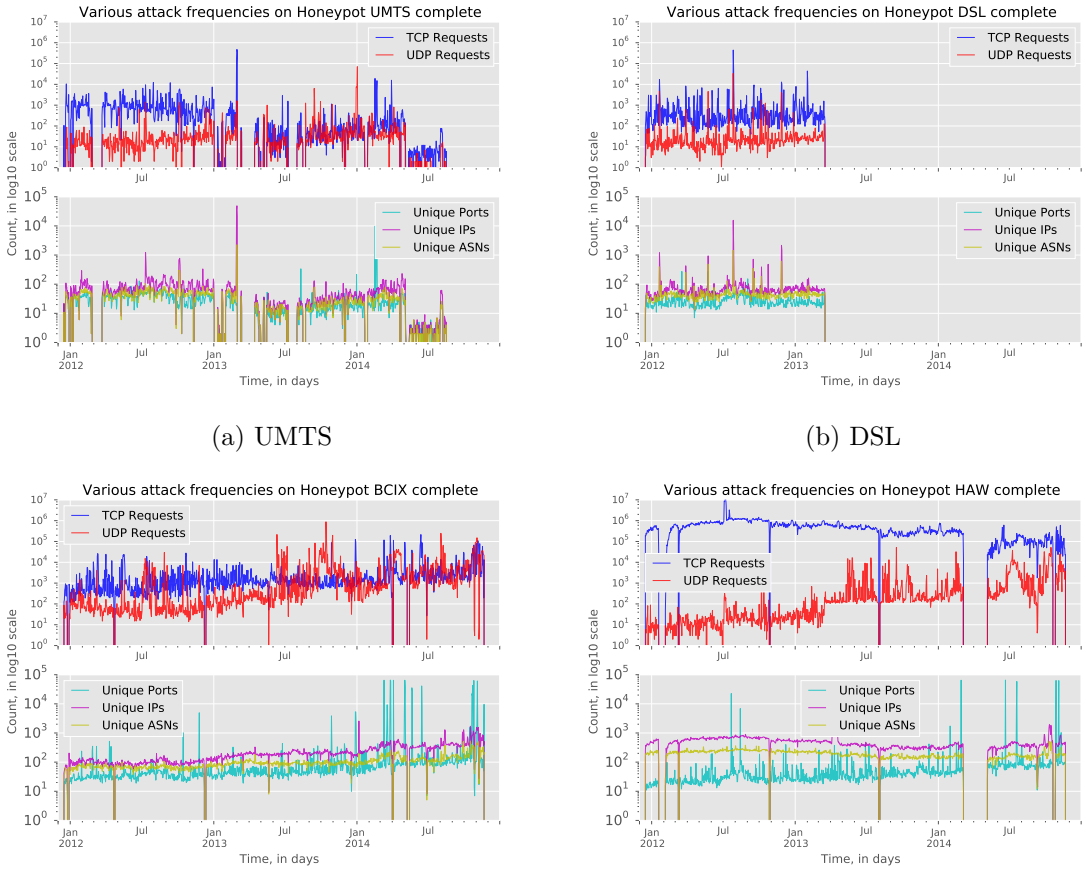
The idea behind this 5 different time series is to enable a deeper insight than a pure requests-per-day view as various situations could become observable. Section 6.3 showed, that targeted ports and the total number of attacks differ for different transport protocols. Therefore a distinction into a TCP and UDP time series can clarify which events are linked to which transport protocol. The unique ports, IP-addresses and ASNs per day give a good overview on how many attack sources and targets exist. For example, the situation of a port scan triggers a peak for the unique-attacked-ports plot, if now the the protocols are considered, one could check for correlations and see whether large port scans are rather TCP or UDP based. Moreover, comparing the number of IP-addresses, one could deduce whether one or many attackers are involved.

Of course, one has to be cautious, to an extent, as simultaneous appearance of in- and decreasing values does not mean that those two observations are necessarily linked: A portscan might be conducted by one attacker and by chance this day can experience a more versatile attacker set. In the end, individual verifications are necessary.

Ultimately, the goal is to see temporal developments and maybe short-term deviations from the overall ratios and trends which were calculated previously, as they could mark special events.

6.4.2 Results

Figure 6.5 visualizes various attack frequencies for the sensors. The time axis has been adjusted, so that it is equally sized for all honeypots. This means, that events which might occur at the same time, such as the increase in UDP-requests, are visible at the same x-position. However, due to the severe difference in the number of incoming requests, especially for the visualization of requests, a log-scale was necessary.



(a) UMTS

(b) DSL

(c) BCIX

(d) HAW

Figure 6.5: Time Series: Requests (TCP/UDP) and Unique Observations (Ports/IPs/ASNs).

The first insight from inspecting the request frequency graph is that attacks occur every day on all sensors. The visible gaps and singular zero-values for the frequencies are honeypot failures. This analysis shows that analogously to the overall perspective in Section 6.3 the sensors experience a different volume of attacks. Although attacks can be spotted for each day, the attacks follow a burst and ephemeral pattern. That means that the volume of events on the honeypot sensors tends to alter frequently.

Considering the different protocols, one can determine that the ratios between TCP and UDP requests found in Section 6.3 are quite stable. The sensors UMTS, DSL and HAW showed a lower share of UDP attacks in contrast to TCP attacks for the overall time span. Short-term deviations from this observations are present but rare: the HAW sensor does not have any day with more UDP than TCP requests, the DSL sensor has two events and the UMTS sensor has about 10 events. The BCIX sensor displayed a very evenly distributed ratio (0.42, 0.58) in the overall analysis. The frequency analysis reveals that this ratio is fluctuating strongly, long periods of times exist with UDP or TCP being the more common protocol. This shows that attacks on the BCIX sensor do not focus on any protocol which might indicate that they are rather random, whereas the other sensors experience specially crafted attacks which tend to shift the protocol distribution in favour of one protocol.

Inspecting the graph for the unique observations, one can spot (without applying any strict correlation factors) that the number of unique IP-addresses and unique ASNs correlate. The number of unique IP-addresses and ASNs over time is quite constant for all sensors. Obviously, as ASNs are a means of aggregation for IP-addresses, the number of ASNs is steadily less or equal the number of IP-addresses. For the UMTS and DSL sensor these numbers are closer to each other, which means that a 1:1 relationship between observed IP-addresses and ASNs exists. In contrast to these observations the BCIX and HAW sensors show a larger difference, which implies that multiple attackers identified by the IP-address attacked the sensors from a single AS.

Another interesting observation can be made for the unique ports frequency. Except for the peak events at the HAW and BCIX sensor, all sensors usually count less than 100 unique ports per day. This value seems to be low, if one considers the number of all possible ports and the distinctly higher number of requests at each sensor. This is especially true for the HAW and BCIX sensors, which do not observe much more unique attacked ports per day than UMTS and DSL although the volume of incoming requests differs severely.

The BCIX and HAW sensors display several, very large unique ports peaks, which go as high as the maximum port range. These are clear indications for port scans. As each scanned port resembles a request, the request frequencies correlate with this events. Scans appear to be done either with UDP or TCP, both scenarios exist. In contrast, the frequencies for unique attackers do not, which suggests that port scans are performed by a single attacker.

6.5 Attackers Propagation

So far, the attackers of the sensors have been only compared by total numbers. Although this gives good insights on the distribution of attackers, it has not been examined whether attackers are specialized to one sensor type. It is left to this chapter to find how common or rare mutual attackers are for the various network access types.

6.5.1 Methodology

The possibilities of analysing the propagation of attacks has been discussed in Section 4.5, hereby the propagation graphs [138] create easily understood results introducing the probability of propagation from one node to another.

In order to check if it is worth investigating the propagation, at first a simple statistic is created, which describes with absolute numbers how many ASNs or IP-addresses have been observed on multiple nodes. Based on the positive results from this validation two types of propagation graphs are created.

The first type is the *occurrence based propagation graph*. If an attacker has occurred on a node, does this attacker also occur on the other nodes? As loops are allowed in propagation graphs, one has also to check if the attacker occurs a second time on the same sensor. For all sensors this check is performed and the probability for each possible edge is determined by $\frac{\#\text{ Detected Propagations}}{\#\text{ Unique Attackers}}$.

The second type is the *action based propagation graph* and it is dependent on a time window. If an action (a request on a specific port) is spotted from an attacker, is this action followed by another action from the same attacker in the next 24 hours? The 24 hours window is adopted from Kaaniche [138], who applied this timeout because of the reassigning of IP-addresses from IP-pools, which happens to be done every 24 hours by the ISPs. This means that this graph is different to the first type in two aspects: First, as the time is considered, the order of actions and occurrence does matter. If IP-address IP_1 appeared on sensor i and j at timestamps $t_i, t_j, t_i < t_j$, the first graph will account for this in both edges. This graph will only account for this at the edge $i \rightarrow j$ because of $t_i < t_j$ and only if $|t_i - t_j| < 24h$. Second, this graph is action based, this means that attackers with many requests influence the graph more, as the propagation check is done for each action. The probability is calculated by $\frac{\#\text{ Propagated Actions in Timewindow}}{\#\text{ Actions}}$.

The propagation graphs are done for two granularities: IP-addresses and ASNs. Furthermore, only the mutual runtime is considered, as all checks would fail the the longer running sensors from the point on the shorter running sensors became inactive, thus biasing the result.

6.5.2 Results

Table 6.10 shows that mutual attackers across sensors indeed exist. However, the propagation for IP-addresses is very small. The decrease of IP-addresses who appeared on one and on two sensors is drastic (372205 $\rightarrow$ 7625). The decrease for ASNs is rather linear. As

#Sensors	#IP-address	#ASN
1	372205	3730
2	7625	2025
3	2335	1557
4	1234	1218

Table 6.10: Number of IP-addresses and ASNs which appeared on several sensors during the mutual runtime.

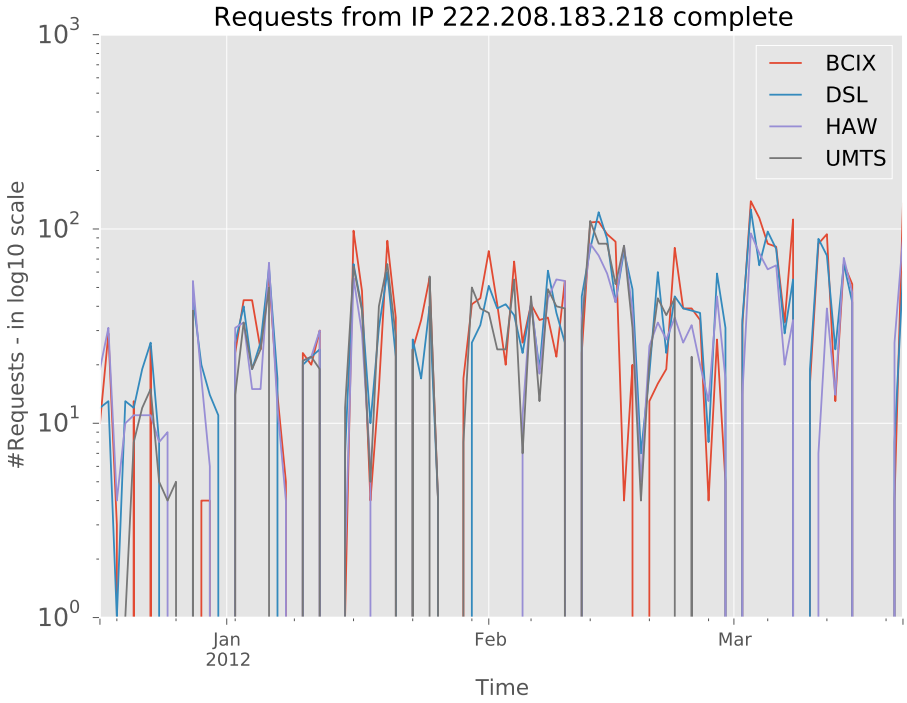


Figure 6.6: Attacks from the same IP-address on several nodes.

mutual attackers exist, further investigation became attractive.

Figure 6.7 illustrates both types of propagation graphs for both granularities. The *occurrence/ip* graph shows the strongest relation between nodes for the $BCIX \rightarrow DSL = 0.141$ edge, the weakest relation between $HAW \rightarrow DSL, HAW \rightarrow UMTS = 0.013$. However, the strongest relations are in the loops, around 0.95 for HAW and 0.60 for the other nodes. This means that the probability is quite low, that an attacker will reoccur on other sensors, however he will most likely perform another action on the same sensor. The set of attackers for each sensor differs severely.

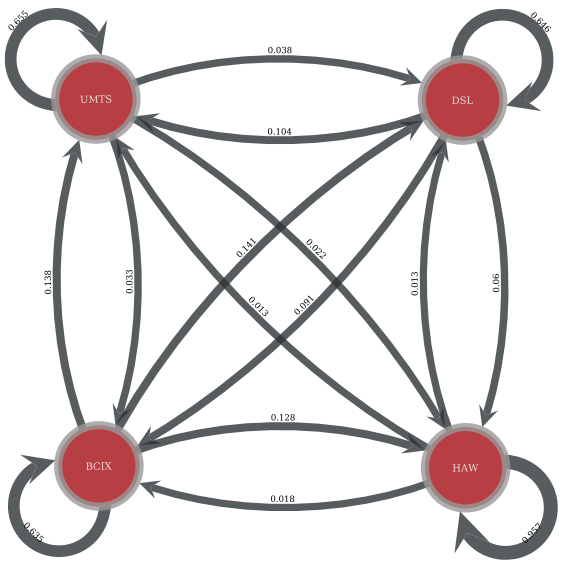
The *occurrence/asn* graph has considerably higher probabilities. As the ASN granularity is more coarse, it results in this higher values. Again, the loops have the highest values. The strongest link between sensors is $DSL \rightarrow UMTS = 0.681$, the weakest $HAW \rightarrow BCIX =$

0.465.

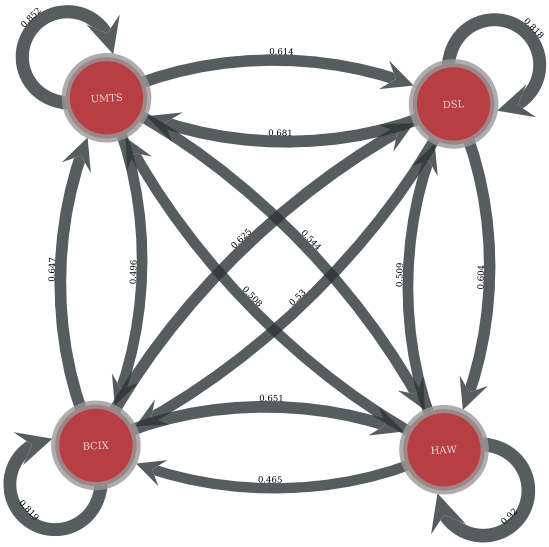
The *action/ips* graph intensifies the already observed trends. The probabilities between nodes are now even smaller, and the loops are severely higher, all above 0.90. The strongest link between sensors is $BCIX \rightarrow HAW = 0.053$, the weakest $HAW \rightarrow BCIX = 0.0001$. This means that attackers perform multi-step actions within a narrow time window on the sensors. Those multi-step actions definitely should be analysed in more detail, as they imply a sequence of actions. However, the attacks across sensors seem to be rare special cases, which can also happen as observed for IPs such as 222.208.183.218, compare Figure 6.6.

The *action/asn* graph has again considerably higher probabilities. The loops have the highest values. The strongest link between sensors is $BCIX \rightarrow HAW = 0.462$, the weakest $DSL \rightarrow UMTS = 0.134$.

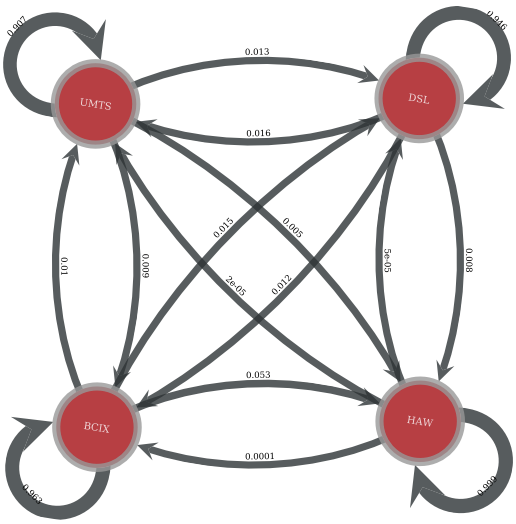
In general, the largest difference between the occurrence and action based graphs is between the HAW and the other sensors. This is due to the large amount of requests which this sensor receives. If one attacker makes many requests and does not propagate to other nodes, the probability decreases severely, as mentioned before.



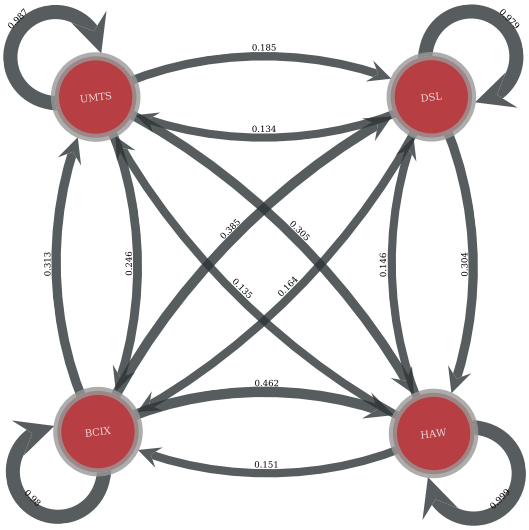
(a) Occurence - IP



(b) Occurence - AS



(c) Action - IP



(d) Action - AS

Figure 6.7: Propagation graphs for total occurrence and actions in time frame. Edge-thickness indicates probability.

6.6 Pattern Detection: Sessions

The analysis on the propagation of attackers revealed, that it is very likely that attackers perform more than one action in a short time window on one sensor. But how can those actions be aggregated?

6.6.1 Methodology

The aggregation is done by time-oriented session reconstruction. The definition of *session* varies across different resources, however in the fields of web analytics it is understood as the *sequence of requests made by a single end-user during a visit to a particular site* [182], in the context of honeypots it can be understood as the ordered list of attacks made by a single attacker on one sensor within a time frame. Time-oriented session reconstructions make use of an inactivity threshold. Each request of an attacker has a time delay to the last performed request. Once this period of inactivity is larger than the inactivity threshold, that attacker is assumed to have left, further requests from the same attacker are considered a second session. The attacker is identified by the IP-address.

Formally: If request $R_{i+1,(H_1,A_1,t_{i+1},p_{i+1})}$ from attacker A_1 arrives at the honeypot H_1 at timestamp t_{i+1} on port p_{i+1} after request $R_{i,(H_1,A_1,t_i,p_i)}$ and the time delay $t_\Delta = t_{i+1} - t_i$ is smaller than the inactivity threshold $t_\Delta \leq t_\Omega$, then the requests R_i and R_{i+1} are both considered to be part of the same session S_n for attacker A_1. If $t_\Delta > t_\Omega$, request R_i is set to be the final request of session S_n for attacker A_1, while request R_{i+1} is the initial request of session S_{n+1} for attacker A_1. Sessions are independent from attackers, several attackers can have an active session concurrently, however each attacker has at most 1 active session at a time. Note, that for two different, subsequent requests R_i and R_{i+1} from the same attacker it is possible that $t_i = t_{i+1}$ and $p_i = p_{i+1}$. This is due to the fact that the events are stored by the honeypots with a 1 second precision. Despite of that precision the honeypots save the correct order of the incoming requests, which makes the session reconstruction possible.

$$\forall\, R_{i,(H_k,A_b,t_i,p_i)},\, R_{i+1,(H_k,A_b,t_{i+1},p_{i+1})} :$$
$$t_\Delta = t_{i+1} - t_i \leq t_\Omega \Rightarrow [\ldots, R_i, R_{i+1}] = S_n$$
$$t_\Delta = t_{i+1} - t_i > t_\Omega \Rightarrow [\ldots, R_i] = S_n, [R_{i+1}] = S_{n+1}$$

for arbitrary but fixed:

$$H \in \{\text{BCIX, HAW, UMTS, DSL}\}$$
$$A \in \{\text{Full IPv4 Range}\}$$
$$p \in \{\text{Full Port Range}\}$$

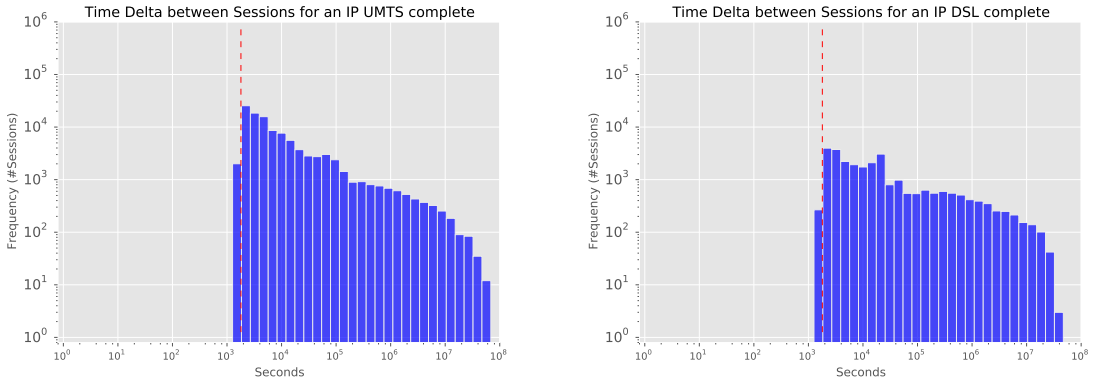
In this chapter a time-oriented session reconstruction is performed. Initially, the duration of the inactivity threshold is discussed and then basic statistics are computed for the aggregated view, in order to verify that sessions successfully group the existing data into meaningful chunks. In particular:

- distribution of return timeouts (delta-return)
- duration of session, mean, std, scatter plot
- requests per session, mean and std
- number of sessions per IP-address, scatter plot

6.6.2 Results

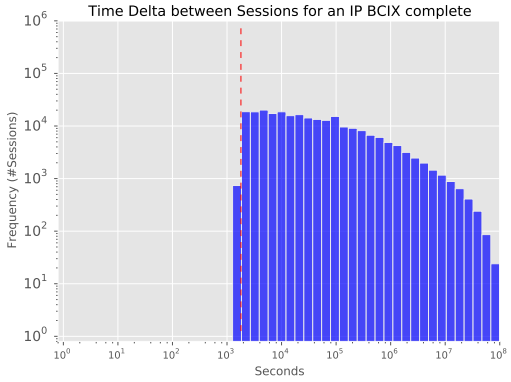
Before a session based analysis can be performed, a suitable inactivity threshold t_Ω has to be found. Two values are used quite often, 24 hours as it resembles a (week-)day and because many devices are assigned new IP- addresses from IP-pools after this timeout [138]. The threshold of 30 minutes also established as a standard in web analytics [182]. First, a time-oriented session reconstruction is performed with $t_\Omega = 30m$. In the following, for all IP-addresses with at least two sessions the time between each session is computed, that means, the distance between the last request from S_n and the first request in S_{n+1}. The durations are plotted as a histogram in Figure 6.8. The red line resembles $t_\Omega = 30m = 10^3 s$, 10^4 and 10^5 seconds are roughly 3 and 24 hours respectively. It becomes apparent from this plot, that in general the longer the absence of an attacker the less likely his return. This fact is indicated by the decreasing size of the right bars. Especially in the range of 1 to 24 hours many returns can be observed. Its seems reasonable to say that periods of such long inactivity form a new attack group. A 24 hours inactivity threshold would combine those sessions into one, which would be too coarse and lead to a too severe loss of detail and therefore information. A small increase can be spotted for each sensor at around $10^5 s$, which in fact shows that many attackers return after the 24 hours mark. But as the session reconstruction is rather done to cluster request events and find explicit attack patterns instead of to characterise all actions performed during an attack day, the smarter choice seems to be to keep the 30 minutes inactivity threshold.

Table 6.11 presents an overview on the sessions for the different sensors. The order of the sensors with respect to the number-of-sessions and the mean-requests-per-session is the same as has been seen for the total requests, compare Section 6.3. Interestingly, this order is shattered for the mean duration of sessions. Especially DSL stands out, which has the second highest mean duration of almost $20m$. BCIX has the shortest sessions with a mean value of only $4m$, it also has the highest amount of sessions which have the smallest possible duration ($1s$). This might be an indication that many attackers found the BCIX sensor by chance and therefore only communicated with the sensor for a short period of time, whereas the communication with the public HAW sensor was expected and therefore long-lasting attacks have been prepared, which result in the longest session durations. This finding seems reasonable if the network access types are considered, as the BCIX resides in an unannounced network and the HAW sensor is part of a well-known, public university network. Expanding this overview, the sensors UMTS/DSL and BCIX/HAW show similar results in Figure 6.9, which shows a scatter plot for the session durations. The similarities are due to dynamic/static IP-addresses of the sensors. As the UMTS and DSL sensors reconnect after 72 or 24 hours, no session can be longer than that. Considering this fact, the long session mean duration of the DSL sensor is even more impressive as no extremely long session exist that could cause an upward shift of the mean. Again, this might indicate that

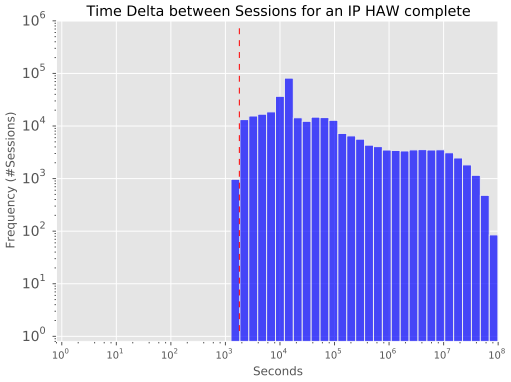


(a) UMTS

(b) DSL



(c) BCIX



(d) HAW

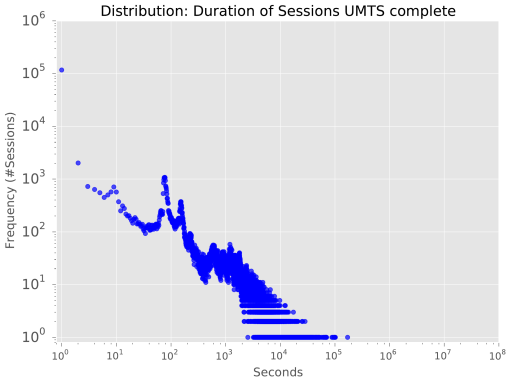
Figure 6.8: Distribution of return timeouts (delta-return) of IP-addresses.

attackers expected to find these sensors and performed longer lasting attacks. Figure 6.10 shows a scatter plot of the sessions per IP-address. Again, UMTS/DSL sensor have similar results, as the plots converge at around 50 sessions. As the sensors change their IP-addresses, it makes it more difficult for attackers to reconnect thus resulting in lower sessions per IP-address. In contrast, the plots for the BCIX/HAW sensors still show many sessions in the $10^2 - 10^3$ range. All sensors have in common that most IP-addresses have only one session.

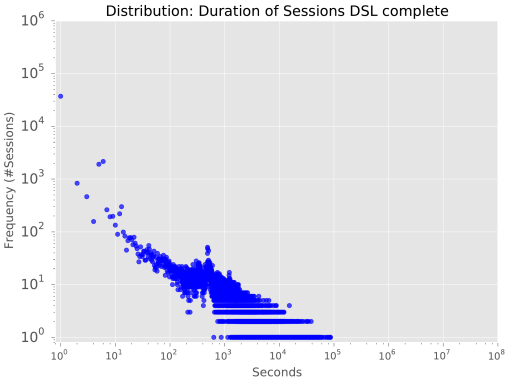
It is worth highlighting the fact that the value of *mean requests per session* also denotes the compression factor, which means that we created an effective means of aggregation: On average, this aggregation method clusters the data by at least factor 6 (UMTS) or factor 818 (HAW).

	UMTS	DSL	BCIX	HAW
Sessions	238486	72201	372598	651090
Duration-Mean	0D 00:13:22	0D 00:19:54	0D 00:04:09	0D 00:36:15
Duration-STD	0D 00:38:37	0D 01:03:01	0D 01:32:11	0D 06:11:06
Frac. Instant (1s) Sessions	0.49	0.51	0.62	0.34
RequestsPerSession-Mean	6.67	13.14	29.94	818.60
RequestsPerSession-STD	159.23	99.52	982.72	24628.69

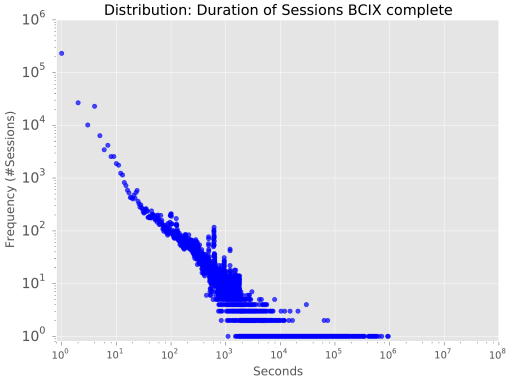
Table 6.11: Overview: Sessions on the honeypot sensors.



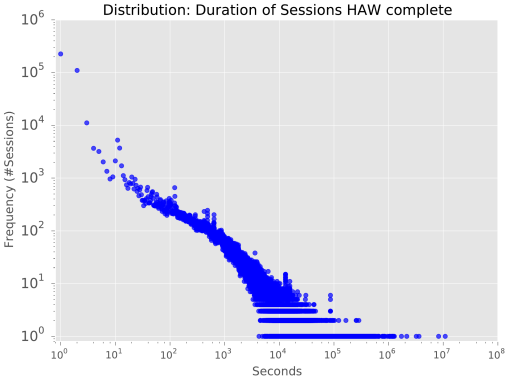
(a) UMTS



(b) DSL

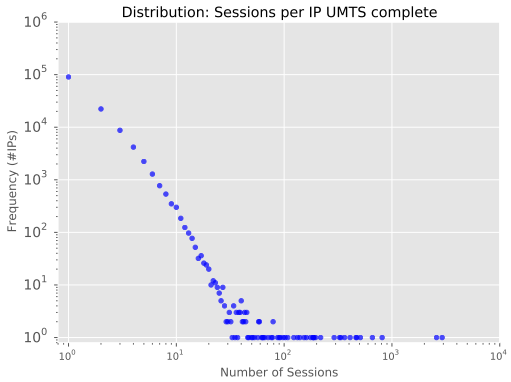


(c) BCIX

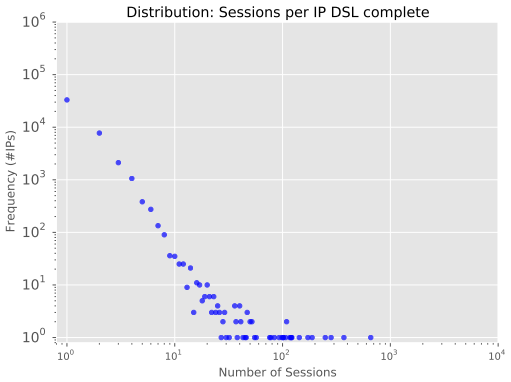


(d) HAW

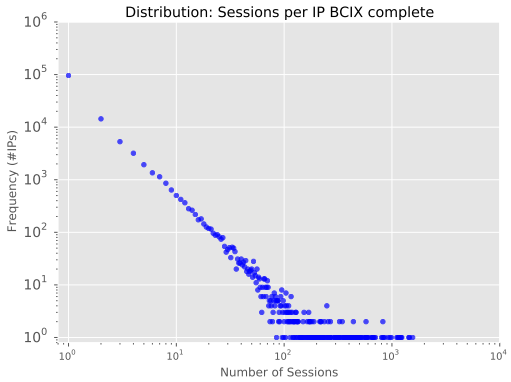
Figure 6.9: Distribution of the Duration of Sessions.



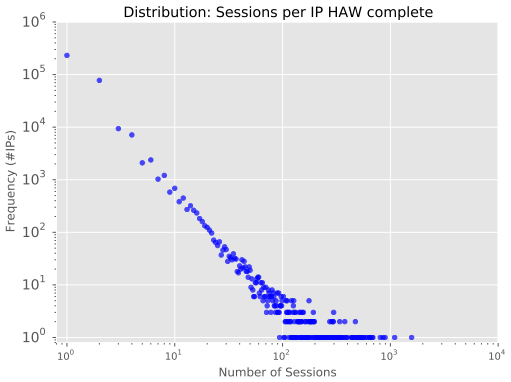
(a) UMTS



(b) DSL



(c) BCIX



(d) HAW

Figure 6.10: Distribution of Sessions per IP-address.

6.7 Pattern Detection: Port Sequences

Sessions have been introduced successfully as a means of aggregating attacks in terms of time. However, how can be the attack pattern of an attack session be described?

6.7.1 Methodology

The attack pattern of an attack session can be described by the *port sequences*. A port sequence is a *time ordered sequence of ports without duplicates* [2], which are build based on the sessions from Section 6.6.

For all requests R of each session $S_n = [R_{i,(H_k,A_b,t_i,p_i)}, R_{i+1,(H_k,A_b,t_{i+1},p_{i+1})}, \ldots, R_{i+j,(H_k,A_b,t_{i+j},p_{i+j})}]$ the list is reduced based on the the associated port p and timestamp t. A request R_{i+y} is removed if and only if another request $R_{i+x}, x < y$ exists which occurred sooner , that means $t_{i+x} \leq t_{i+y}$, and the same port was attacked, $p_{i+x} = p_{i+y}$. In the following, the list is reduced to only contain the port numbers. A time ordered sequence of ports PS without duplicates emerges. Formally:

$$
\begin{aligned}
&\forall S_n; \quad R_{i,(H_k,A_b,t_i,p_i)}, R_{i+j,(H_k,A_b,t_{i+j},p_{i+j})} \in S_n : \\
&S_n^* = S_n \backslash [R_{i+j,(H_k,A_b,t_{i+j},p_{i+j})} \mid \exists (R_{i,(H_k,A_b,t_i,p_i)} \wedge (p_i = p_{i+j}) \wedge (t_i \leq t_{i+j}) \wedge j > 0] \\
&PS_n = [p_i \mid (R_{i,(H_k,A_b,t_i,p_i)} \in S_n^*]
\end{aligned}
$$

One evident advantage of the port sequences is that heavy spammers, frequent scanners and so on do not distort the results. To be exact, if one attacker performs many attacks on only one port, the target might seem common although being only attacked by one attacker. This has been taken into account previously by also considering the number of unique attackers for each port, compare Section 6.3. Another advantage is that targets are described by a pattern, that means, that different actions of one attacker are combined into one information. It is now possible to qualify the variety of ports and sophistication of an attacker by looking at the complexity of his port sequences.

6.7.2 Results

First results based on port sequences are presented in Table 6.12. An overview about the list of all unique port sequences is given. In general, the longer the mean sequence length is, the more unique port sequences can be assumed, as simply more ports exists in those sequences, which in turn allows more permutations. The BCIX sensor has the most, the DSL the fewest unique sequences, and also the largest and smallest mean value respectively. However, the UMTS sensor has severely more unique port sequences then the HAW sensor although HAWs mean port sequence length is larger. This means, that the HAW sensor is attacked by long, reoccurring (specialised) attack patterns, whereas the UMTS sensor is attacked with many different, shorter patterns. Additionally, the table shows the top port sequences by rank ordered by occurrence - sequences with more than one port are highlighted. The HAW sensor is the only one with many longer port sequences in the top

10, moreover the longer sequences show a strong relation ship between the ports 139, 445 and 80, which are well known application ports. Similar, BCIXs most common sequence with more ports is (445,139). However, the other sequences are dominated by ports from the 33434-33523 range, which are the default traceroute ports. The top 10 sequences contain single ports only. The DSL sensor shows the same behaviour for its top 10, however the longer sequences show more common, lower application ports, although no stronger relation between certain ports can be spotted. The UMTS sensor also shows mostly lower, known application ports. However, the most common sequences are on ports 45783 and 55670, both not assigned to any application or known to be used unofficially. All sensors have in common that sequences with more than 3 ports are seldom and sequences longer then 10 ports exceptions, as can be seen in Figure 6.11. That means that attacks are performed with a specific aim.

Comparing these results to the previously found targets in Section 6.3, the results are quite similar to the perspective which counts the unique attackers per port. This is due to the fact that many attackers have only one session, as was shown in the previous chapter. However, it is now possible to say that some port combinations are more likely than other, a relation between ports exists in attacks. However, the relation is not quantified in this analysis. As is shown by Table 6.13, many sequences contain only one port, however, a good amount of port sequences contain more than one port, therefore those relations are investigated in the next chapter. The longest port sequence from UMTS, BCIX and HAW been a sequential port scans. The ports in DSLs longest sequence seem to have no order and to be chosen manually, as they are known application ports with no structure. Lastly, Figure 6.12 confirms the already observed situation for the portwise analysis: a huge amount of traffic on the sensors can be described be few port sequences. All sensors share this property with minor variations.

Sensor	Unique Sequences Statistic			All Sequences		Len(Sequences)>1	
	Count	Length		Rank	Sequence	Rank	Sequence
		Mean	Std				
UMTS	7987	7.89	164.66	1	(45783)	2	(45783, 55670)
				2	(45783, 55670)	3	(55670, 45783)
				3	(55670, 45783)	26	(8080, 9090)
				4	(55670)	28	(80, 8080)
				5	(23)	32	(23, 80, 8080)
				6	(3389)	37	(8080, 9090, 8081)
				7	(22)	41	(23, 210)
				8	(5060)	44	(8080, 80)
				9	(2842)	45	(23, 22)
				10	(1433)	54	(3389, 21, 25)
DSL	2669	2.51	6.48	1	(6881)	31	(23, 210)
				2	(3389)	38	(3389, 25)
				3	(23)	45	(8080, 9090)
				4	(51099)	47	(25, 110)
				5	(22)	61	(8080,3128,1080)
				6	(5060)	69	(6515, 8080)
				7	(28694)	73	(1080, 8080)
				8	(5900)	75	(6515, 1080)
				9	(8080)	77	(6515, 9090)
				10	(80)	84	(6881, 4662)
BCIX	8121	104.95	2291.79	1	(1433)	19	(445, 139)
				2	(3389)	31	(33450, 33451)
				3	(22)	38	(33449, 33450)
				4	(1900)	39	(33443, 33444)
				5	(5900)	40	(33441, 33442)
				6	(5060)	42	(1433, 137)
				7	(23)	43	(33453, 33454)
				8	(53)	46	(33442, 33443)
				9	(36284)	47	(33440, 33441)
				10	(19)	50	(33452, 33453)
HAW	5248	73.79	2040.53	1	(445)	2	(139, 445)
				2	(139, 445)	3	(445, 139)
				3	(445, 139)	4	(445, 80)
				4	(445, 80)	8	(139, 445, 80)
				5	(3389)	9	(445, 139, 80)
				6	(5900)	37	(445, 80, 139)
				7	(22)	40	(80, 445)
				8	(139, 445, 80)	46	(139, 80, 445)
				9	(445, 139, 80)	54	(8080, 80)
				10	(1900)	64	(137, 135)

Table 6.12: Number of unique sequences and ranking of sequences by occurrence.

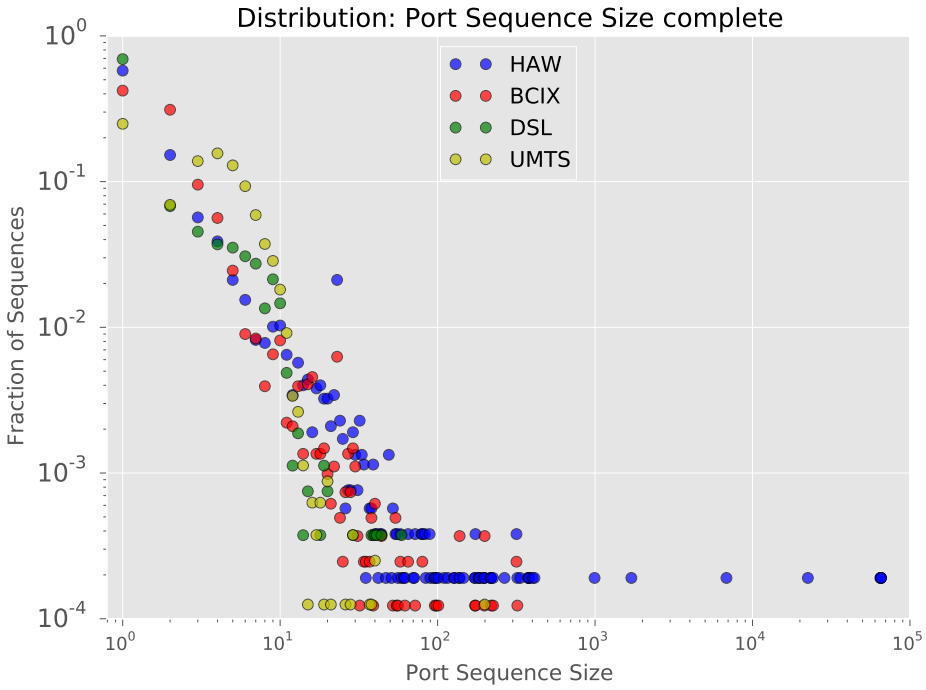


Figure 6.11: Distribution of Sequence Size for Unique Port Sequences.

	UMTS	DSL	BCIX	HAW
fraction single port sequences	0.24	0.69	0.42	0.57
largest port sequence size	9884	268	65118	65467

Table 6.13: Fraction of smallest sequences and largest sequence size for unique sequences.

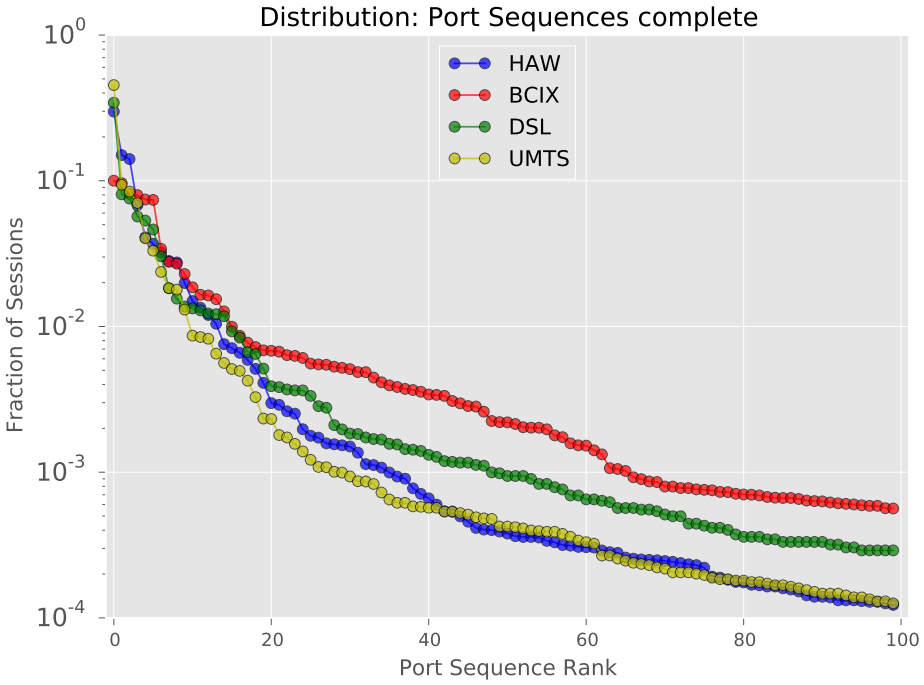


Figure 6.12: Session-Share of Attacks by Top 100 Sequences.

6.8 Pattern Detection: Apriori Analysis

The previous analysis on port sequences and especially the ranking of port sequences consisting of at least two port numbers indicated that some port combinations are more likely. However, no mathematical model was presented to quantify the relationships. How strong is the relationship for some ports?

6.8.1 Methodology

Analogously to Pougets work [132], compare Section 4.6, the relationship between ports will be described by *support and confidence*. The classic and therefore well-understood Apriori Algorithm [183] is used to determine those values. A real-world example of the Apriori algorithm is the market basket analysis that reveals items which are frequently bought together, a customer basket corresponds to the occurrence of a port sequence. This algorithm proceeds by identifying the frequent individual ports and extending them step by step to larger port sets as long as those sets appear sufficiently often in the data. Association rules are derived from frequent sets which highlight general trends. In summary:

- Support (Apriori on >1 port sequences)
- Confidence (Apriori >1 port sequences)

6.8.2 Result

Table 6.14 presents the result of the induction of association rules. Moreover, it shows the support and confidence for those rules. The rule mining is performed on the set of all sessions, which have at least two ports in their related port sequence. That means, that all results can be interpreted as followed: If the attacker targets more than one port p_x, he will attack port p_y with the probability of the confidence cf for that specific rule.

The sensor UMTS has only two really distinctive rules, which show the confidence 1.0 for port 45783 and 55670. Such a strong confidence can only be observed at the sensor HAW for 139 and port 445, furthermore even a rule with 3 ports has such a strong confidence. The DSL sensor has the most rules with 3 ports in this ranking. The relation between the traceroute ports has been verified with this analysis for the BCIX sensor. The rules compared across the sensors rather show no mutual findings. The registered services at IANA or unofficial default ports of known applications are written next to the rules. One can easily spot, that attacks are for example HTTP-based (80, 8080, 8081) or database-related (1433, 3306) and so on.

Sensor	Support	Confidence	Apriori Rule	Services
UMTS	0.9	1	45783 -> 55670	n/a
	0.9	1	55670 -> 45783	n/a
	<0.1	0.8	23 -> 8080	telnet, HTTP/Alternate
	<0.1	0.8	23 80 -> 8080	telnet, HTTP, HTTP/Alternate
	<0.1	0.8	3128 -> 8080	Squid HTTP, HTTP/Alternate
	<0.1	0.8	8081 -> 8080	both HTTP/Alternates
	<0.1	0.8	8909 -> 9415	n/a
	<0.1	0.7	23 8080 -> 80	telnet, HTTP/Alternate, HTTP
	<0.1	0.7	22 -> 23	SSH, telnet
	<0.1	0.6	23 -> 80	telnet, HTTP
DSL	0.1	0.8	8085 9000 -> 9415	n/a
	0.1	0.7	9000 9415 -> 8085	n/a, SqueezeCenter web server
	0.1	0.7	8085 9415 -> 9000	n/a, SqueezeCenter web server
	0.1	0.6	6515 -> 8080	n/a, HTTP/Alternate
	0.1	0.6	8085 -> 9415	n/a
	0.1	0.6	8090 -> 1080	HTTP/Alternate, SOCKS proxy
	0.1	0.5	9090 -> 1080	XMPP Openfire, SOCKS proxy
	0.1	0.5	6515 -> 1080	n/a, SOCKS proxy
	0.1	0.5	8000 -> 1080	Intel Rmt Dsktp, SOCKS proxy
	0.1	0.5	6515 -> 9090	n/a, XMPP Openfire
BCIX	0.1	0.9	33454 -> 33453	traceroute
	0.1	0.9	33452 -> 33453	traceroute
	0.1	0.9	3306 -> 1433	MySQL, MSSQL
	0.1	0.7	137 -> 1433	NetBios, MSSQL
	0.1	0.7	137 -> 135	NetBios, MS EPMAP
	0.1	0.6	33453 -> 33454	traceroute
	0.1	0.6	1433 -> 135	MSSQL, MS EPMAP
	0.1	0.5	135 -> 1433	MS EPMAP, MSSQL
	0.1	0.4	33453 -> 33452	traceroute
	0.1	0.4	1433 -> 137	MSSQL, NetBios
HAW	0.5	1	139 -> 445	NetBios, MS SMB
	0.1	1	80 139 -> 445	HTTP, NetBios, MS SMB
	0.5	0.9	445 -> 139	MS SMB, NetBios
	0.1	0.9	8888 -> 8080	HyperVM HTTPS, HTTP/Alternate
	0.1	0.8	3128 -> 8080	Squid HTTP, HTTP/Alternate
	0.1	0.6	80 -> 445	HTTP, MS SMB
	0.1	0.6	8080 -> 80	HTTP/Alternate, HTTP
	0.1	0.5	80 445 -> 139	HTTP, MS SMB, NetBios
	0.1	0.5	8080 -> 3128	HTTP/Alternate, Squid HTTP
	0.1	0.4	8080 -> 8888	HTTP/Alternate, HyperVM HTTPS

Table 6.14: Top rules detected by apriori algorithm for port sequences with more than one port.

6.9 Pattern Detection: AS Uniformity

Sessions have been used to aggregate attacks originating from IP-addresses. Moreover, the targets have been described by port sequences and relationships between ports revealed. However, no AS-based perspective has been given. How similar are attackers from one AS if sessions and port sequences are considered?

6.9.1 Methodology

In order to describe the AS uniformity the notion of *representative port sequences RPS* is introduced. RPSs are a means of making the results more meaningful, that is robust against frequent attackers with many sessions. Assuming that for an autonomous system ASN_0 two unique IP-addresses IP_1, IP_2 have been observed, and IP_1 has severely more sessions than IP_2, then analysing all port sequences originating from AS_0 would be biased by the preferences of IP_1. This drift is prevented by appointing the most common port sequence of each attacker IP_x from the set PS_{IP_x} to be the RPS (if two sequences are equally common then the sequence is chosen that was observed first).

After determining the RPSs for each IP-address from one ASN, the largest cluster c_{max} of common RPSs is determined for each ASN. The AS uniformity U_{ASN_y} is the relative size of the found cluster. Hence, the AS uniformity can only be calculated meaningfully for ASNs with more than 1 IP-address as otherwise the AS uniformity would be straight-away 1.0. However, this is not a real limitation as its purpose is to measure the similarity of several attackers from one ASN. If all RPS_n differ, the uniformity is by definition 0.0. To sum up:

$$
\begin{aligned}
&RPS_{ASN_y} := [RPS(PS_{IP_x}) \mid \forall IP_x \in ASN_y] \\
&\forall RPS_q, RPS_r \in RPS_{ASN_y} : RPS_q, RPS_q \in c_k \Leftrightarrow RPS_q = RPS_r \\
&c_{max} : size(c_k) < size(c_{max}) \, \forall k \\
&U_{ASN_y} = \frac{size(c_{max}(RPS_{ASN_y}))}{\#unique\,IPs\,in\,ASN_y} \quad \text{or 0 if } size(c_{max}(RPS_{ASN_y})) = 1
\end{aligned}
$$

6.9.2 Results

Since the AS uniformity can only be done for AS with more than one attacker, one has to identify how man AS fulfil this requirement. Table 6.15 shows that more than half of the ASNs have at least two IP-addresses. Consequently our results about the uniformity of AS are based on more than half of the observed AS, which seems expressive enough.

Figure 6.13 is a histogram of U_{ASN_y} for each AS monitored on a sensor. In general, all sensors share a common result: The AS either show total uniformity (1.0) or no uniformity (0.0). Clusters in between are rather seldom, especially on the DLS sensor. However, no uniformity is even more common than total uniformity, the ratios are roughly 1:1 for UMTS and BCIX, 2:1 for HAW and 3:1 for BCIX.

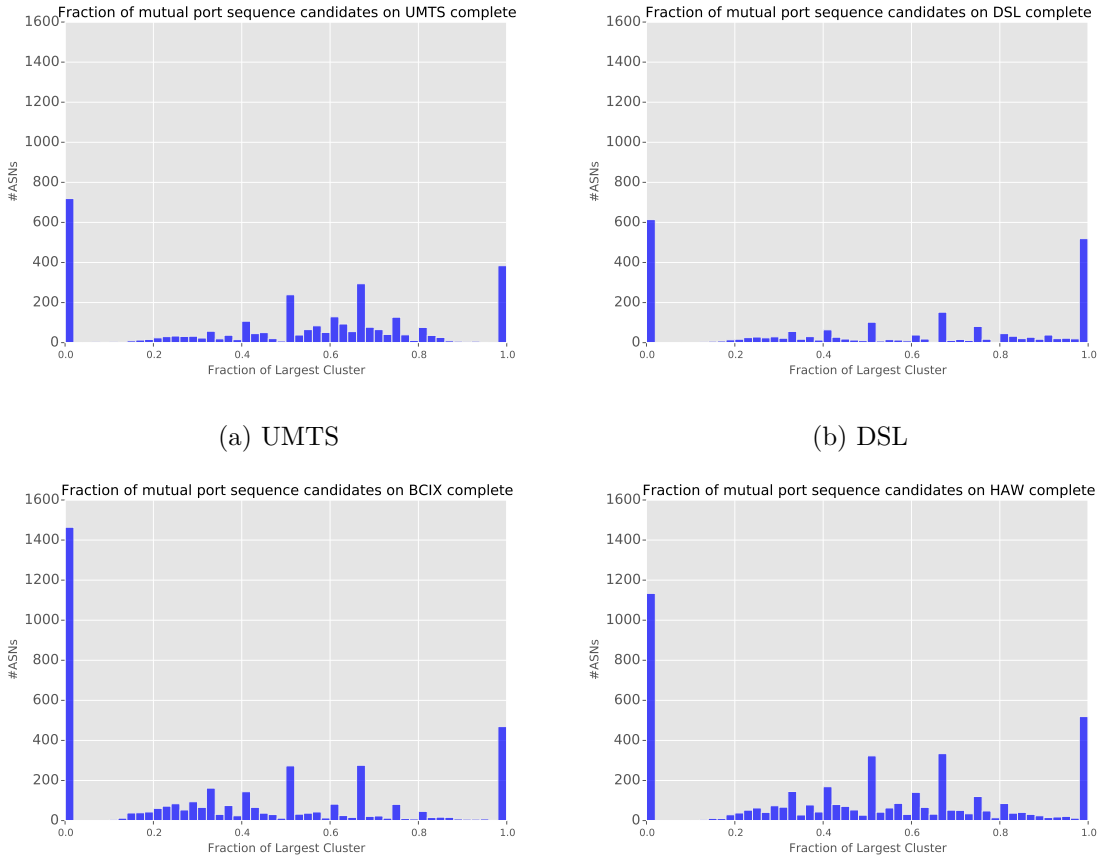
Lastly, Table 6.16 presents details on the clusters with maximum uniformity. Interestingly, DSL and HAW both have a very large maximum cluster size of 491 and 468 respectively. This means, that AS exist, which attacked the sensors with over 400 hundred unique attackers, who all used the same port sequence. However, the most common cluster size is smaller with about 2 attackers. Analysing the length of the *RPS* the BCIX sensor stands out the most, as the longest *RPS* is 5 ports long, which means that all attackers from the AS used a particular attack pattern that included actions on 5 ports, which is quite a complex characteristic.

	UMTS	DSL	BCIX	HAW
AS ratio with >1 IP-address	0.63	0.53	0.55	0.61

Table 6.15: Ratio of ASNs with more than one IP-address.

	UMTS	DSL	BCIX	HAW
most common cluster size	2	2	2	2
largest cluster size	41	491	69	468
RPS length interval	[1,2]	[1]	[1,5]	[1,2]

Table 6.16: Statistics about ASN with maximum uniformity.



(a) UMTS

(b) DSL

(c) BCIX

(d) HAW

Figure 6.13: Size of largest cluster of IPs with same port sequence candidates.

6.10 Pattern Detection: Autocorrelation

Up to this point different time series analysis have been performed. However, although the timeseries appealed fairly random, no mathematical model has been applied to verify this observation. So far, no analysis checked for random or repeating patters in the data: Do periodic patterns exist?

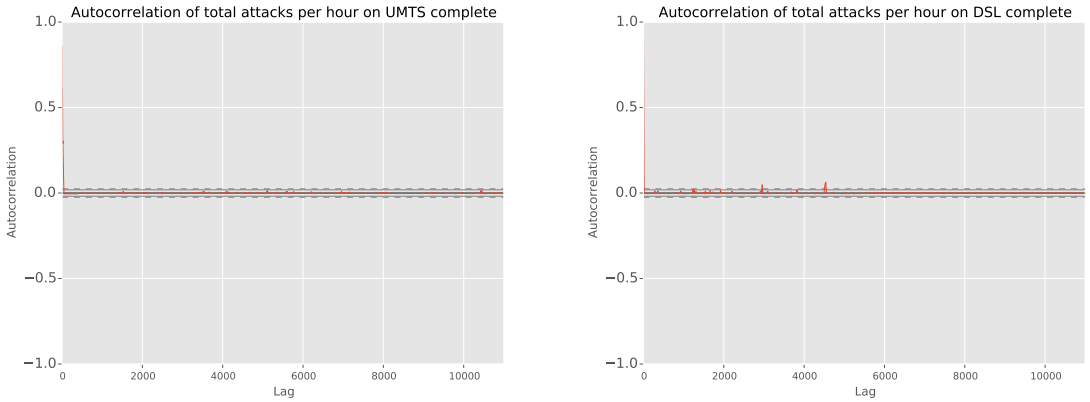
6.10.1 Methodology

To check for repeating patterns the autocorrelation [184] is used. Autocorrelation compares the similarity between timeseries as a function of the time lag between them. Autocorrelation plots visualize the standardized similarity with the interval [1:-1] for the varying time lags. If a timeseries is random, such autocorrelations should be near zero for all time lags. Periodic patterns create significant non-zero peaks. Pandas autocorrelation libraries standardize and de-trend the data by subtracting the mean and dividing by the standard deviation of the data. Horizontal lines are added, which correspond to 95% and 99% confidence bands. The following plots with the different time granularities have been created:

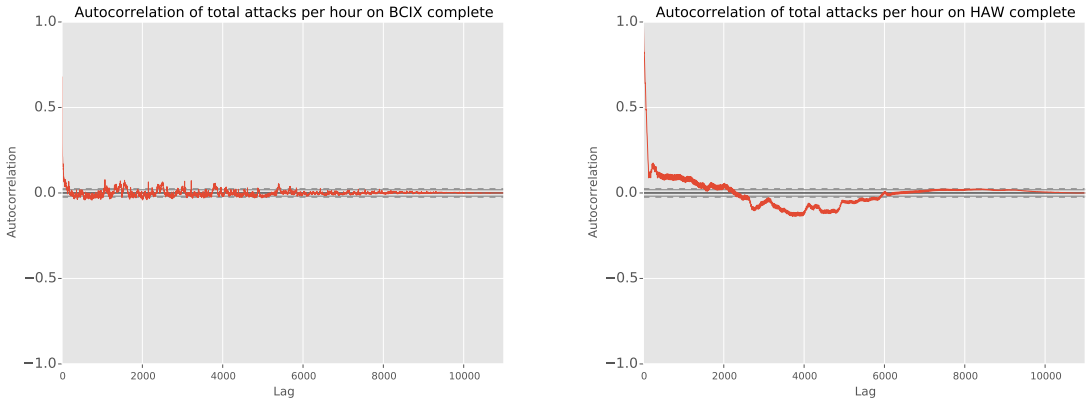
- autocorrelation for attacks per hour timeseries
- autocorrelation for attacks per day timeseries

6.10.2 Results

The results are shown by Figure 6.14 and Figure 6.15. The day-wise autocorrelation plots are omitted here as both granularities reveal the same findings. Satisfying the expectations, both granularities demonstrate a random behaviour for all sensors. Sensors HAW and BCIX show some non-zero values and therefore some reoccurring behaviour, however the values are neglectfully large and do not show any sine-pattern. Consequently, no periodicity can be determined. The only high peaks are observable at the beginning of the plots, which means that a specific level of traffic persists for several hours (up to 3 days for HAW), which again means that specific events / attack types persists for specific time spans and then vanish. This would coincide with the findings of “burst attacks” discussed in the related work [140]. An ongoing DDOS-attacks can be an extreme example. The higher values might be explainable because of the static IP-addresses which these sensors possess, which allows attackers to easily reconnect.

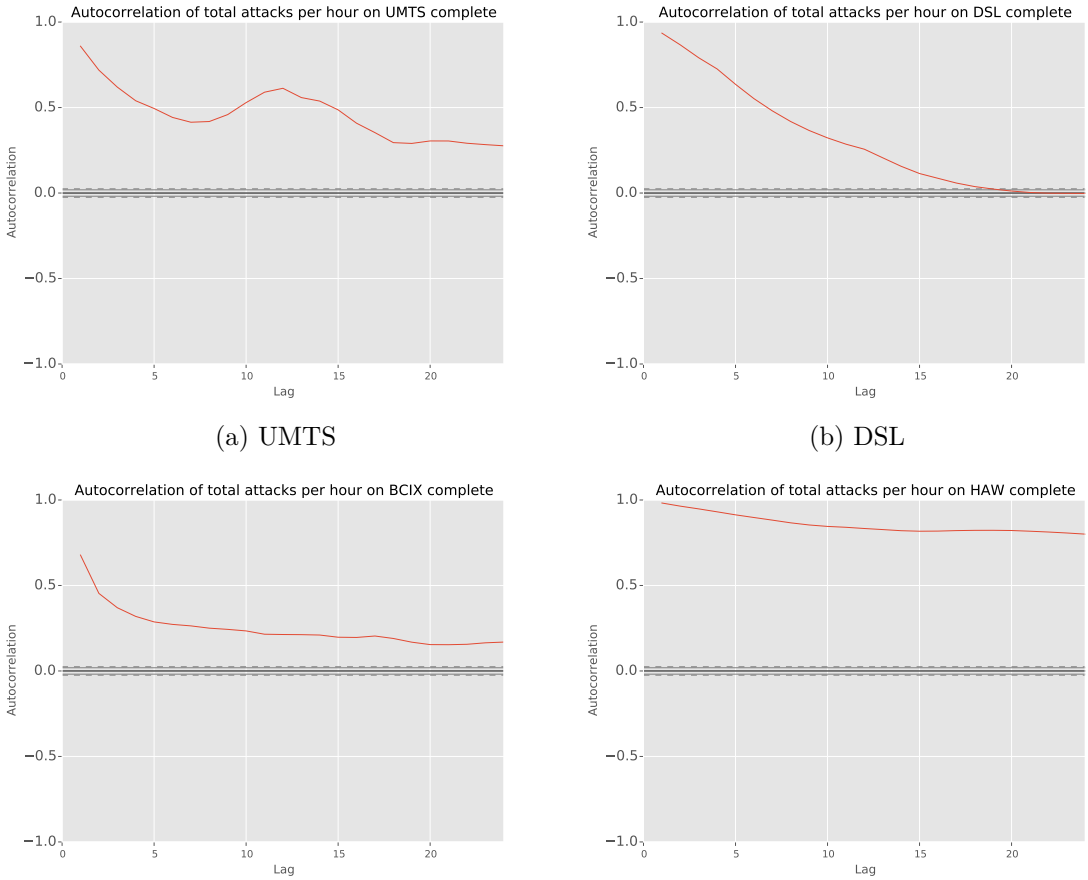


(a) UMTS (b) DSL



(c) BCIX (d) HAW

Figure 6.14: Autocorrelation of the attacks per hour timeseries for mutual runtime.



(a) UMTS

(b) DSL

(c) BCIX

(d) HAW

Figure 6.15: Autocorrelation of the attacks per hour timeseries for mutual runtime - first 24 hours.

6.11 Pattern Detection: SAX Timeseries Analysis

In order to ultimately compare the similarity of the sensors a mathematical model is required. Up to this point only metrics have been calculated and then compared. Only the autocorrelation has been applied to measure the self- similarity. But how similar are sensors to each other, which sensors reveal the most similar attack patterns?

6.11.1 Methodology

The previously presented SAX timeseries analysis is used to describe the attack patterns on a portwise basis. Moreover, Thonnards et al. similarity measure consisting of the local (SIM_L) and global (SIM_G) similarity is used to compute the total (SIM_{Total}) similarity, compare Section 4.6 and Thonnard et al. works [140]. Especially the usage of SIM_L is necessary for the available data as the breakpoints used during the SAX quantization process are defined assuming that time series has a Gaussian distribution. However, as can be seen in the results from Section 6.4 and Section 6.10 the attacks exhibit temporal spikes. In particular, if timeseries for requests on only one port are considered, usually a pattern with few spikes and many zeros or very small values elsewhere becomes apparent. This leads to a distorted distribution and a median value close to zero, which results in a obviously biased high similarity degree given by pure SAX (SIM_G).

Two SAX analysis are performed with different scopes of ports. The first creates a list of the 10 most attacked ports from each sensor, merges these lists and compares the similarity. Those ports are especially interesting as they exhibit feature-rich data, i.e. many non-zero values. The second SAX analysis iterates through the complete port range. In general, for each port of each sensor a timeseries is created which depicts the attacks per day. If no attacks have occurred on a port of a sensor (timeseries with pure zero-values, no data), this timeseries is marked as a *null-series*. If sensors are compared and both have a null-series for the current port, the test is skipped for this particular port. As our sensors have different runtimes, the data set has been also reduced to the mutual time range. The upper quartile (uq) value was chosen to be 0.975, which resembles up to 12 peak days for each timeseries. Moreover, for the first analysis the most similar ports are found, as they are very interesting because they possess the same patterns despite being feature-rich. To sum up:

- SAX top 10 ports analysis
- SAX top 10 ports – most SAX-similar ports
- SAX full port range analysis

6.11.2 Results

Table 6.18 demonstrates the results for the SAX top 10 ports analysis. Choosing the top 10 ports from each sensor and merging these lists resulted in 27 unique ports. *Checks* demonstrates that all SAX-comparisons were only possible for the DSL-UMTS combination, which indicates that rather uncommon ports have been attacked frequently on this sensors but also that the top ports from the other sensors have been attacked. *Sum* shows the total sum of all possible SAX-similarity calculations, *mean* and *std* the functions on those

calculations respectively. The *similarity counter* counts in absolute values how often a certain similarity value has been exceeded.

The sensors DSL and UMTS seem to be the most alike, however they are only slightly more similar. If we consider Thonnards definition of similar, which means a similarity level above 0.90, then with mean values around 0.75 all sensors seem to experience different attack pattern for the top ports. Table 6.17 shows the 3 most similar ports from the top ports list for each sensor combination. Port 2701 for UMTS and DSL was a perfect match, as all zero symbols and peaks coincide. The least similar combinations are between sensor HAW and DSL if the mean values are compared, UMTS and BCIX if the sums are compared.

If the Table 6.19 is considered, which depicts the SAX analysis for the full port range, the trend continues. However, now UMTS and DSL have the least successful checks. Interestingly, they have the highest amount of portwise similarity with over 0.95. This means that only few ports are attacked on those sensors, however, if they are attacked, the probability is high that the attack patterns coincide severely. Comparing the mean values, the UMTS and BCIX combination is again the least similar one.

Generally speaking, the mean values for all sensors are between 0.53 and 0.58, which is a very low value of similarity. This value is mostly affected by SIM_G, which calculates a high similarity for uncommon ports due to many zero values, SIM_L then adds up only a small value because of not matching peaks. This means that for unusual ports the sensors rather experience different burst-type attacks.

Sensor1	Sensor2	Port_1	Port_2	Port_3	Similarity_1	Similarity_2	Similarity_3
UMTS	DSL	2701	6881	161	1.0	0.95	0.91
UMTS	HAW	143	1433	19	0.94	0.90	0.89
UMTS	BCIX	19	2701	53	0.97	0.88	0.87
HAW	BCIX	19	25	110	0.91	0.91	0.89
HAW	DSL	6881	143	25	0.90	0.87	0.86
DSL	BCIX	6881	53	2701	0.91	0.90	0.88

Table 6.17: Highest ranked, most SAX-similar ports from top-10 ports analysis.

Sensor1	Sensor2	Checks	Sum	Mean	Std	Similarity Counter				
						>0.7	>0.8	>0.85	>0.9	>0.95
UMTS	DSL	27	20.54	0.76	0.10	20.0	9.0	4.0	3.0	2.0
UMTS	HAW	25	19.10	0.76	0.09	18.0	11.0	4.0	2.0	0.0
UMTS	BCIX	24	17.85	0.74	0.11	17.0	9.0	5.0	1.0	1.0
HAW	DSL	25	18.47	0.73	0.10	18.0	8.0	3.0	1.0	0.0
HAW	BCIX	23	17.08	0.74	0.11	14.0	7.0	6.0	2.0	0.0
DSL	BCIX	24	18.02	0.75	0.10	15.0	8.0	4.0	2.0	0.0

Table 6.18: SAX similarity comparison based on top ports.

Sensor1	Sensor2	Checks	Sum	Mean	Std	Similarity Counter				
						>0.7	>0.8	>0.85	>0.9	>0.95
UMTS	DSL	3518	2040.95	0.58	0.14	675.0	399.0	330.0	254.0	199.0
UMTS	HAW	24414	13318.36	0.54	0.07	932.0	313.0	165.0	105.0	72.0
UMTS	BCIX	9915	5242.47	0.52	0.08	652.0	299.0	231.0	150.0	127.0
HAW	DSL	24773	13514.54	0.54	0.07	987.0	320.0	187.0	107.0	66.0
HAW	BCIX	30518	16381.34	0.53	0.07	915.0	311.0	194.0	123.0	95.0
DSL	BCIX	10360	5494.19	0.53	0.09	714.0	344.0	258.0	171.0	129.0

Table 6.19: SAX similarity comparison based on all ports.

6.12 Attack Evolution

So far, a quite erratic attack behaviour has been determined on the sensors. However, we have not discussed if those attack spikes are new events or just a burst of known events. How often do new observations occur on the sensors and how steady are certain request types?

6.12.1 Methodology

The evolution of observations is defined in a rather unsophisticated manner. Whenever an observation takes place which is scouted for the very first time, an evolution is spotted. New observation are counted for new TCP and UDP port, as well for new IPs and ASNs.

In order to find out how steady attacks on ports are, the days of attacks for each port are determined. An attack day is a calendar day on which at least one attack appeared on the specific port. A scatter plot is created. The ports with most days of attack are pointed out. An overview of the analysis steps:

- new TCP ports per day
- new UDP ports per day
- new IPs per day
- new ASNs per day
- Number of ports with respect to days of attack

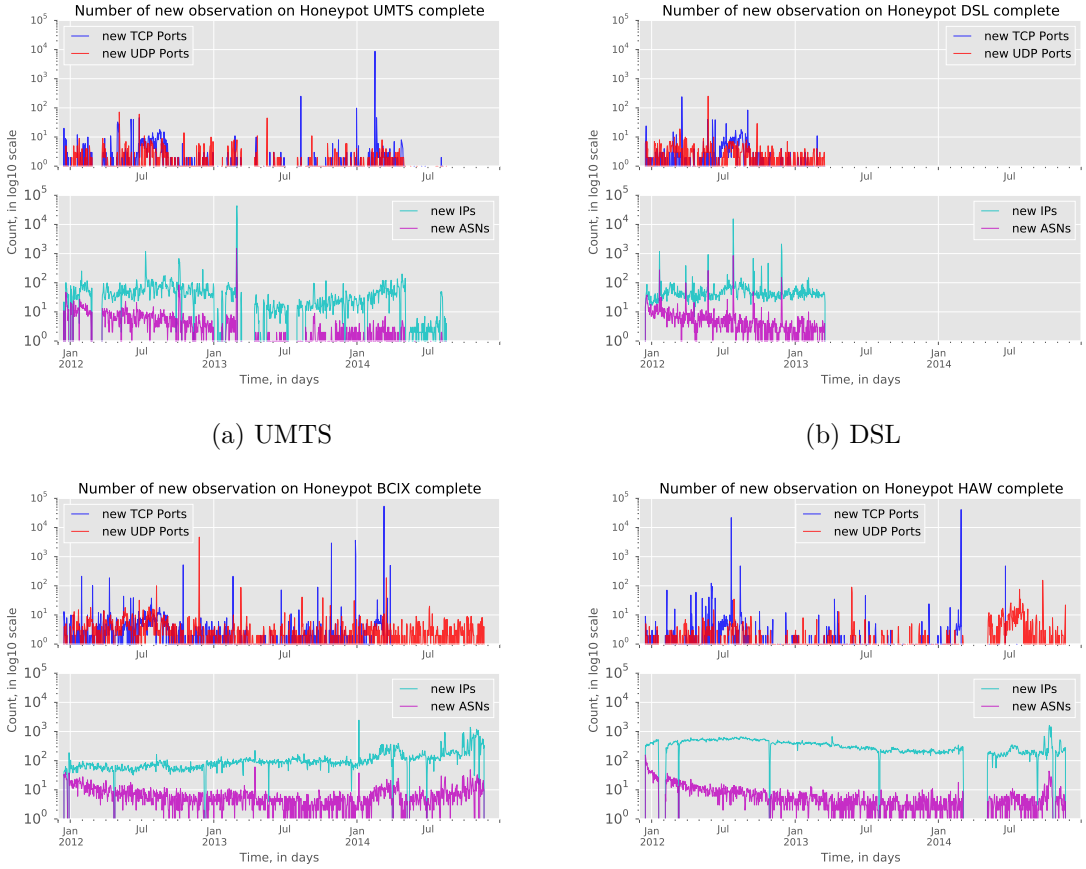
6.12.2 Results

Figure 6.17 demonstrates a combined plot of the new observations for the ports, IPs and ASNs. Comparing the TCP and UDP graphs, it becomes apparent that new port events are present throughout the whole runtime, even for BCIX and HAW sensor. UMTS and DSL sensors show new UDP and TCP events for their whole runtime, no protocolwise dependence can be spotted, the spikes of both curves are independent. BCIX and HAW sensors experience a similar attack pattern, however for both sensors all TCP ports are requested at certain point of time and only new events occur for UDP ports. This might indicate, that more services are expected to use the statefull, connection oriented TCP and therefore are scanned earlier. However, it takes time until the full port range is scanned.

The plots for new IPs and ASNs show a stronger correlation, whenever many new IP-addresses appear, the number of new ASNs increases. This is due to the frequent 1:1 relation of IPs and ASNs, for about half of the spotted ASNs only one IP-address could have been spotted, compare Section 6.9. Moreover, the most interesting observable fact is that the curves are quite constant. Surprisingly, there is no decrease for the number of ASNs even for the long running sensors, sensors BCIX and HAW experience even a small rise in the last months. This means that the attacks on the sensors constantly originate from new IPs and ASNs. Moreover, many new attackers do not inherently mean that many new ports will be attacked, attack targets reoccur.

These results can be compared to the attack frequencies from Section 6.4. In this section

already a steady curve for unique IPs and unique ASNs has been determined. Now it has been shown that these attackers were not only unique, but also new. If the curves for the ports are compared, one can easily learn that spikes in the number of requests do not overlap with spikes for new ports. This once again means, that many attacks reoccur on a certain port and that full port scans are rather seldom, attacks concentrate on few ports.



(a) UMTS

(b) DSL

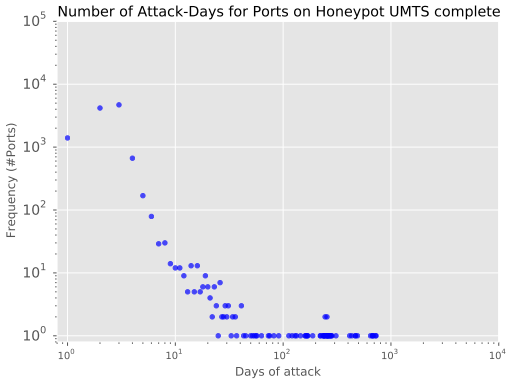
(c) BCIX

(d) HAW

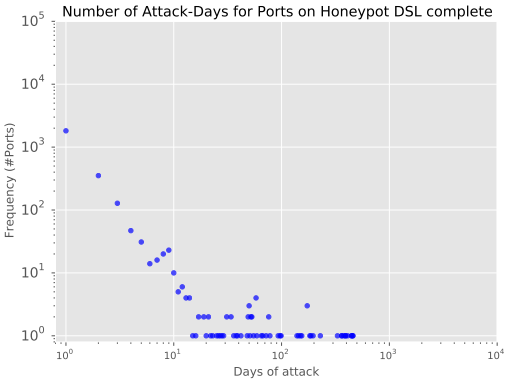
Figure 6.16: Number of new observations per day.

Table 6.20 reveals how steady the attacks on ports are. All sensors exhibit only few days of attacks for a large range of ports. Around 100 days of attacks seem to be the breakpoint between frequently and seldom attacked ports for UMTS and DSL. Due to the longer runtime and probably the static IP-address of the sensors this breakpoint is shifted to the right for HAW and BCIX, as it is at around 150 days. Figure 6.16 shows the most attacked ports with respect to days of attack and the total runtime of the sensors. Unsurprisingly, almost all ports in this list resemble well-known services, such as SSH (22), Remote Desktop Protocol (3389), VOIP (5060) and so on. UMTS and DSL have exactly the same ports in their list, however in a different order. Furthermore, the UMTS sensor has a unique distribution in this list, as the top ports have a lower fraction of attack days compared to the other sensors. One can conclude, that if it comes to well-known service ports, they

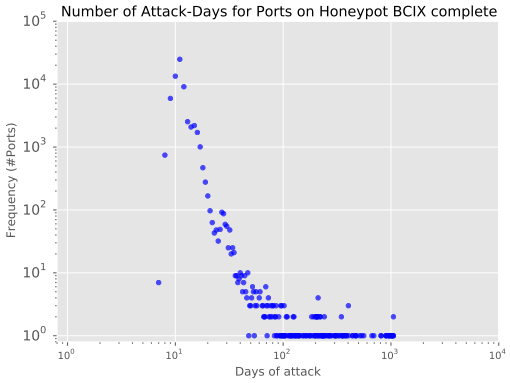
experience an continuous flow of attacks, attacks on rather uncommon and not registered ports are seldom.



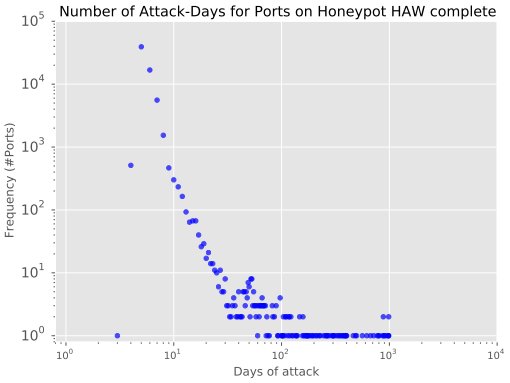
(a) UMTS



(b) DSL



(c) BCIX



(d) HAW

Figure 6.17: Number of ports with respect to days of attack.

Sensor	Rank	Frac. Days	Port
UMTS	1	0.74	5060
	2	0.73	3389
	3	0.70	23
	4	0.69	22
	5	0.68	8080
DSL	1	1.0	3389
	2	0.99	22
	3	0.97	23
	4	0.97	5060
	5	0.89	8080
BCIX	1	0.98	5060
	2	0.98	1433
	3	0.98	3389
	4	0.98	22
	5	0.96	8080
HAW	1	0.92	445
	2	0.92	139
	3	0.92	3389
	4	0.92	80
	5	0.91	22

Table 6.20: Ports with most days of attacks.

6.13 Attack Risk

The last analysis investigates the dangerousness of the attacks, or how large is the risk imposed by the attacks onto the honeypots?

6.13.1 Methodology

As low-interaction honeypots were deployed the assessment of the risk is very limited. However, it can be done by reviewing the amount of incoming traffic, compare Section 4.8. No full package recording was deployed in the setup, therefore only the amount of requests can be reviewed. However, as our setup was extended by snort, the volume and type of snort alerts can be brought additionally into consideration. The following metrics are calculated:

- Mean Requests per Day
- Mean Snort Alerts per Day
- Type of Snort Alerts

6.13.2 Results

Table 6.21 shows the result for the mean values for the mutual runtime of the sensors. If it comes down to the mean requests per day, the sensor HAW is by far the most endangered one. Interestingly, although it receives more traffic by two orders of magnitude than the other sensors, it is exceeded by the sensor BCIX in the mean alerts per day. However, the differences for this metric are not that severe. Based on this observation one can conclude, that the HAW and BCIX sensor are the most endangered ones.

Sensor	Requests per Day		Alerts per Day	
	Mean	STD	Mean	STD
UMTS	3000.7	27887.5	1555.8	3468.0
DSL	2072.6	22592.6	898.7	6297.5
BCIX	2515.0	4446.9	5892.9	11062.1
HAW	823424.0	1058933.0	4259.4	11984.4

Table 6.21: Average Attacks and Alerts per Day on Honeypots for the mutual runtime.

This conclusion is particularized by Table 6.22, which shows the types of the top 5 alerts for each sensor. The HAW alerts are most versatile, signatures for ICMP errors, SSH brute force login attempts and even transfer of executable binary files have have been detected quite often. Based on these types of alerts the sensor HAW seems to be the most endangered one. The other sensors show VOIP-SIP floods, all other alerts are ICMP related. Especially the UMTS and DSL sensor receive many VOIP-SIP floods. This might be a sign of context-sensitive attacks. Since attackers are actually able to identify mobile devices from the Telekom-network by WHOIS lookups, compare listing 9, they might focus their attacks on the VOIP protocols.

In general, it is interesting that many signatures in this list are ICMP related. That shows, that machines are sought by PING and that the alleged senders of incoming traffic do not really expect a reply, which causes the *host/ port unreachable* alerts. This might be an indication of misconfigurations, spoofing or poorly implemented attacker scanning tools.

Sensor	Rank	Fraction	Snort Signature
UMTS	1	0.91	VOIP-SIP REGISTER flood
	2	0.03	ICMP Destination Unreachable Port Unreachable
	3	0.01	ICMP PING
	4	0.01	ICMP PING *NIX
	5	0.01	ICMP PING BSDtype
DSL	1	0.87	VOIP-SIP REGISTER flood
	2	0.06	ICMP Destination Unreachable Host Unreachable
	3	0.03	ICMP Destination Unreachable Port Unreachable
	4	0.01	ICMP Echo Reply
	5	0.01	ICMP PING
BCIX	1	0.69	ICMP Destination Unreachable Port Unreachable
	2	0.15	VOIP-SIP REGISTER flood
	3	0.12	ICMP Microsoft remote unauthenticated DoS/bugcheck vuln.
	4	0.01	ICMP Echo Reply
	5	0.01	ICMP PING
HAW	1	0.50	ICMP Destination Unreachable Port Unreachable
	2	0.15	ICMP Microsoft remote unauthenticated DoS/bugcheck vuln.
	3	0.11	BAD-TRAFFIC SSH brute force login attempt
	4	0.08	ICMP Destination Unreachable Host Unreachable
	5	0.05	WEB-CLIENT Portable Executable binary file transfer

Table 6.22: Top signatures detected by snort.

```
whois 37.84.153.6

inetnum:        37.80.0.0 - 37.87.255.255
netname:        TDMOBILE-2
descr:          Telekom Deutschland GmbH
country:        DE
admin-c:        TH12429-RIPE
tech-c:         AS8728-ripe
tech-c:         MS47198-RIPE
status:         ASSIGNED PA
```

Listing 9: WHOIS lookup for an IP-address of the UMTS sensor.

CHAPTER 7

Conclusion

The final chapter outlines the methods that haven been designed and conducted in this thesis and concludes the insights.

7.1 Summary

A honeypot is decoy computer resource whose value lies in being probed, attacked or compromised. Honeypot are efficient reactive security measures, especially if they are compared to IDSs and Firewalls which are rather centred around the prevention and detection of attacks. The advantages are the collection of valuable data, the independence from workloads, the detection of zero-day exploits and the general flexibility of the concept. Some honeypots, especially those configured in mirror-modes and reflecting attacks, might cause legal issues, however no legal case is known to this day with respect to entrapment, liability and privacy concerns.

Honeypots allow extensive log data analysis of attacks which can be used to generate knowledge. However, various types of honeypots exist with many different implementations, each with a different purpose. A taxonomy for honeypots has been introduced, dividing the available honeypot software into server and client honeypots in the first, and then into low and high interaction honeypots in the second category. Based on this taxonomy, an attack could have been defined for each honeypot type. Using a general definition, each anomaly observed on a honeypot is an malicious attack, although the definition of an anomaly differs, as mentioned, across different honeypot types. Moreover, this taxonomy was the basis for a survey of available honeypot software, which introduces the various implementations with its specific properties, specialization on services and their maintenance time. The maintenance time of most honeypots tends to be rather short. Therefore, although a plethora of implementations exist, only some solutions have established themselves as advisable standards.

Several publications were released in the field of the honeypot data analysis, however no survey suggesting a guidance for honeypot analysis has been introduced. This thesis presents a set of analysis questions, which should by answered in a step-by-step manner during the analysis. Each question is linked to publications and their metrics, which were applied to

answer the respective question. Questions, which request direct, apparent information have been answered more often than questions, which derive information, that means, information which explains, assesses or localizes the cause of the observations. The questions of the first type ask for the evident attack origins, attack target and the attack frequencies. The second type includes the evolution and propagation of attacks, the detection of patterns and the risk estimation.

A honeypot log data analysis framework has been presented. First, a linear software architecture has been created which reduces the dataset from unnecessary bulk and then extends it by further information. The subsequent two components are introduced, which analyse the data set and output the results to the user. The implementation makes heavy use of the scientific data analysis package Pandas. The analysis component has an configurable memory management section, as well as a query and filter engine. It creates intermediate results on small data chunks, as the complete data set does not fit into the main memory, and then merges those, similarly to MapReduce.

With the guidance of the analysis questions and the self-developed framework a honeypot data analysis has been performed. The dataset contained information from 4 different honeypot sensors, each with a different network access type to the Internet: DSL, UMTS, a public university network and Darknet sensor. This data set has been enriched with historical BGB information, so that an AS-based perspective was possible. The AS-mappings have been done with a precision of 3 months, which proofed to be accurate enough. The attack origins have been described by IP-addresses, ASNs and operating systems. The targets have been identified by the the port number. The frequencies were consistently aligned by a day, however applied for different metrics such as requests, unique attackers and ports. The propagation of attacks has been quantified by propagation graphs. The most extensive sub-analysis was done for the pattern detection. Attacks have been accumulated by the method of session reconstruction and the attack targets described by the means of port sequences, a time-ordered port list without duplicates. The composition of port sequences and the relationship of ports within a port sequence has been quantified by the Apriori algorithm, which derived association rules measured by support and confidence. Moreover, the uniformity of attacks originating from one AS has been described by the help of representative port sequences, which are the mostly used port sequence of an attacker from one AS – the uniformity of the AS was then expressed by the share of the largest cluster of representatives. Until this point, the similarity of the sensors has been only deduced by comparing results, a mathematical model was missing. Therefore, the autocorrelation has been used to quantify the self-similarity of attacks on one sensor with respect to a time-lag, and a SAX analysis combined with a local and global similarity measure to calculate the total similarity between sensors. The evolution of attacks was based on counting new, unknown events per day, including different metrics such as new ports and new IP-addresses. The risk estimation was done with the help of snort alerts and as low-interaction honeypots have been deployed on the basis of the amount of incoming traffic.

Results for the attack sources and especially for the attack targets (ports and port sequences) indicate an exponential distribution with a long tail on all sensors. If the top attacked ports are compared some mutual, common, well-known ports can be spotted, although some distinctive ports are also definitely observable on all sensors. Moreover, mutual attackers are very seldom (IP-addresses as well as ASNs), only few attackers propagated across sensors.

Concerning the attack frequency, all sensors experience burst like attacks, although the amount of attacks differs tremendously. The session reconstruction reduces these differences slightly, as requests from continuous or massive spammers are accumulated. The burst like behaviour has been confirmed by the autocorrelation, as it shows a higher similarity up to a lag of about two days, which means that some event types appear randomly and than last for this period. New events are common on the nodes, especially new attackers based on IP-addresses as well as ASNs are constantly appearing. Also new ports are attacked quite often, it took surprisingly long until all ports have been attacked, which clearly shows goal-oriented attacks, full port scans are seldom. The Apriori algorithm confirmed this observation, as port sequences often contain several ports, which are used by different applications but all speak the same protocol or are all related to the same type of applications, such as databases. Checks on the uniformity of AS revealed that some source AS have very clear, uniform attack patterns for different IP-addresses. The most endangered sensor is the one from the public university network, HAW. Interestingly, the UMTS sensor reveals many IDS-alerts which are related to VOIP, which indicates access-sensitive attacks. A possible explanation for this awareness of attackers was introduced based on WHOIS queries. Finally, the SAX analysis and the global similarity measure revealed, that the sensors indeed experience different attack patterns on the ports. The difference is especially strong, when the full port range is compared. The level of similarity based on the most active ports is a bit higher. The UMTS and DSL sensor are the most similar ones, yet their similarity value is only slightly higher then from the other possible combinations and still far away from *similar* – attacks differ across various network access types.

7.2 Limitations

The limitations of the setup and the analysis include two major concerns. First, the honeynet does not have any possibility to verify whether the attack source of collected packages denoted as IP-address has been spoofed. This is unfortunately an inherited issue from the Internet architecture [185]. Second, only one sensor exists for each network access type, which means that each sensor-specific event is automatically a network access-specific event. This relationship could be too firm.

Another limitations arouse because of the long-term measurements. Many additional data sets which could have been consulted were inappropriate, as information could not have been restored for the past years.

Besides that, the honeynet was not configured to log events for two protocols: ICMP and IPv6. Although ICMP was not explicitly logged by the honeypot software, the snort alerts indicated that the honeypots received ICMP traffic. Logging ICMP events allows detection and analysis of events such as pure ping scans, which went undetected in our analysis. No information was collected about incoming attacks on IPv6-addresses. No comparison between IPv4 and IPv6 could have been drawn. Considering current statistics [186] this protocol is not yet wide-spread but its usage is constantly increasing and it is becoming more important. However, as only the addressing of the devices changes and the network layers above the IP layer remain unaltered, analysis methods such as port sequences or Apriori should reveal similar results. Small changes for the attack frequencies are assumed, as the IPv6 address space is severely larger. This implies that random scanners are less

likely to connect to our sensors.

7.3 Insights Compared to Prior Work

This section compares the obtained results with related publications and reveals similarities but also differences to existing research.

Considering the distribution of attack sources and attack targets we have observed a rather unsurprising heavy-tail distribution: The top targets or top sources are responsible for the majority of traffic observed on a sensor. This trend has been previously seen in several publications [132, 133, 134, 139]. Our analysis has used a novel specification of attackers by using ASNs, which allowed new insights into the source of attacks but which analogously followed the heavy-tail distribution.

Repeating the time series analysis, the attack frequencies observed by the related work [134, 138, 140] follow a pattern of sustained bursts and mostly ephemeral spikes. We have observed the same patterns. However, we also verified additionally by the means of autocorrelation that this ephemeral spikes do not follow a certain pattern, the self-similarity of events on our sensors is only given for a short time offset, namely for the period of the current ephemeral spike.

Propagation graphs for honeypot sensors have been previously used by Kaaniche [138]. Kaaniche concluded that honeypot sensor tend to show low propagation values, if not set in the same subnet. We also observed low propagations values across sensors. As our sensors have heterogeneous network access types and hence are placed in different subnets, these results coincide.

Similar to the early results of Pouget [132] we have observed that attacks on honeypots create a very large amount of data. We show analogously that it might be misleading to consider general statistics obtained on these data without carrying an in depth analysis and aggregate repetitive spamming behaviour from single attacks sources. Session reconstruction and port sequences have proofed to be effective measures of aggregation. Comparing the structure of the port sequences observed by Pouget [2] [132] and our results from the Apriori test [183], we make the same observation about the port sequence length, which is most commonly one. However, our results show a stronger causal relationship between ports for similar services (e.g. databases) and protocols (e.g. HTML). As our analysis has been conducted much later, this might indicate a specialization of attackers over time.

We have shown that our sensors have a low similarity based on the SAX analysis and the applied global and local similarity measure. This was even the case for common target ports with much traffic. The temporal similarity for a certain port was rather an exception. These are contrary findings compared to the related SAX analysis from Thonnard [140]. Thonnard observed that several sensors show strong temporal similarities and that a temporal correlation between attacks for specific ports exists. He applied his analysis on the the leurre.com honeypot data set, which was collected in academic and industrial subnets. Although the details on the network access types remain unspecified, it is assumed that similar access types were used for all leurre.com sensors. This could be the reason why Thonnards sensors show a strong similarity and our sensors do not. The comparison between our work and Thonnards is insofar important as it is an indication for bias in related

research that emerged because of not considering the network access type.

Our findings extend the findings of McGrew [135], who spotted that attacks on honeypots are dependant on the geographical location of the sensors. Considering our results, one can say that the geographical location as well as the network access type of sensors are influential factors.

Except for the leurre.com project [2] most honeypot analysis are conducted only for time ranges of a few months, compare Table 6.2. Our analysis is based on a long-term measurement, which should make the result more significant and more independent from temporary deviations. Furthermore, this allowed us a unique view on the evolution of attacks throughout several years.

7.4 Significance of the Findings

The results have several implications for the related research area.

First and most importantly, the network access type dependency of honeypot sensors has been shown, which means that not only the geographical location [135] but also the network access type is a relevant factor for the honeypot deployment. Future honeypot analysis should always include that factor in their reasoning. Interestingly, not only has been shown that events differ for different network access types, but also that attacks are access-aware, which is indicated for example by the increased number of VOIP attacks on the UMTS sensor. This implies that attacks on honeypots should be analysed with respect to their network access type and also be screened for access-sensitive properties.

Second, it has been been shown that a large portion of traffic on honeypots can be described by only a couple of aggregation rules because of a heavy-tail distribution of attacks. This means that filter rules based on those are able to filter a large amount of malicious traffic on honeypots, making it easier to focus on the scientific analysis of less common events. If the productive operation of services is considered, such rules could be used to reduce the load on the specific network and to protect its own services. If the post data analysis is considered, such filter rules would create smaller data sets and hence result in a performance gain. Considering the network type dependency, one can enhance this improvement by applying filter rules according to the specific network access type.

Third, preliminary results revealed that some source AS have very clear, uniform attack patterns for different IP-addresses. The AS-based perspective should not be omitted in honeypot research any more, as it extends the analysis with a more broad, holistic view on the attacks originating from the Internet.

7.5 Outlook

Further research on the network access type dependency of honeypots should go in the following direction. For each access type several sensors should be deployed in order to eliminate sensor-specific attacks and to allow deducing the access-specific attacks from the mutual attacks on sensors of one access type. Propagation of attackers across sensors of one access-type is assumed to be more likely, so this test should be extended for propagation

checks between sensor of one access type. In order to overcome the limitations, additional data which is time-dependant should be collected by the honeynet, as the retrieval of the data in retrospect can be very challenging, sometimes impossible. It is advisable to save information from spam, spoof and ASN rating lists. Moreover, the honeynet sensor should possess IPv6-addresses and log events for this protocol. The honeypot software should also be configured to log ICMP events such as pure ping scans.

To allow a better screening of context-sensitive properties of attacks, the honeynet should be also extended by high-interaction honeypots. This would enable to conduct attack root identifications and exploit detections. If attackers really perform context-sensitive attacks for specific network accesses, then eventually different binaries should be collected which utilize the exploited sensors dependant to their access type.

Bibliography

[1] Vinod Yegneswaran, Paul Barford, and Vern Paxson. Using honeynets for internet situational awareness. In *Proceedings of the Fourth Workshop on Hot Topics in Networks (HotNets IV)*, pages 17–22. Citeseer, 2005.

[2] Corrado Leita, VH Pham, Olivier Thonnard, E Ramirez-Silva, Fabian Pouget, Engin Kirda, and Marc Dacier. The leurre. com project: collecting internet threats information using a worldwide distributed honeynet. In *Information Security Threats Data Collection and Sharing, 2008. WISTDCS'08. WOMBAT Workshop on*, pages 40–57. IEEE, 2008.

[3] Symantec. Internet security threat report 2015. `https://www4.symantec.com/mktginfo/whitepaper/ISTR/21347932_GA-internet-security-threat-report-volume-20-2015-social_v2.pdf`. Last accessed Frebruary 2016.

[4] Emmanouil Vasilomanolakis, Shankar Karuppayah, Max Mühlhäuser, and Mathias Fischer. Taxonomy and survey of collaborative intrusion detection. *ACM Computing Surveys (CSUR)*, 47(4):55, 2015.

[5] Lance Spitzner. The honeynet project: Trapping the hackers. *IEEE Security and Privacy*, 1(2):15–23, March 2003.

[6] Emmanouil Vasilomanolakis, Shankar Karuppayah, Panayotis Kikiras, and Max Mühlhäuser. A honeypot-driven cyber incident monitor: lessons learned and steps ahead. In *Proceedings of the 8th International Conference on Security of Information and Networks*, pages 158–164. ACM, 2015.

[7] Loras R. Even. Honey pot systems explained. `https://www.sans.org/security-resources/idfaq/honeypot3.php`, July 2000. Last accessed January 2016.

[8] Lance Spitzner. The value of honeypots, part one: Definitions and values of honeypots, october 10, 2001. `http://www.symantec.com/connect/articles/value-honeypots-part-one-definitions-and-values-honeypots`. Last accessed on January 2016.

[9] Clifford Stoll. *Cuckoo's Egg*. Pocket, 1990.

[10] Fred Cohen. A note on the role of deception in information protection. *Computers and Security*, 1999.

[11] Bill Cheswick. An evening with berferd in which a cracker is lured, endured, and

studied. *AT&T Bell Laboratories*, 1991.

[12] Al-Sakib Khan Pathan. *The State of the Art in Intrusion Prevention and Detection*. CRC Press, 2014.

[13] Networking SANS Institute (SysAdmin and Security). Cooperative research and education organization. `https://www.sans.org/`. Last accessed on January 2016.

[14] Computer Emergency Response Team CERT/CC. Exploitation of vulnerability in cde subprocess control service. `https://www.cert.org/historical/advisories/CA-2002-01.cfm`. Last accessed on January 2016.

[15] Abhishek Mairh, Debabrat Barik, Kanchan Verma, and Debasish Jena. Honeypot in network security: a survey. In *Proceedings of the 2011 International Conference on Communication, Computing & Security*, pages 600–605. ACM, 2011.

[16] Bruce Schneier. *Secrets & Lies: Digital Security in a Networked World*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 2000.

[17] Al-Sakib Khan Pathan Mohssen Mohammed. *Automatic Defense Against Zero-day Polymorphic Worms in Communication Networks*. Auerbach Publications, 2013.

[18] Iyatiti Mokube and Michele Adams. Honeypots: Concepts, approaches, and challenges. In *Proceedings of the 45th Annual Southeast Regional Conference*, ACM-SE 45, pages 321–326, New York, NY, USA, 2007. ACM.

[19] Christian Seifert, Ian Welch, Peter Komisarczuk, et al. Honeyc-the low-interaction client honeypot. *Proceedings of the 2007 NZCSRCS, Waikato University, Hamilton, New Zealand*, 2007.

[20] Lance Spitzner. Seclists email lists, honeypot tools categorization, 2004. `http://seclists.org/honeypots/2004/q3/5`. Last accessed on January 2016.

[21] Niels Provos. Honeyd - honeypot background, 2004. `http://www.honeyd.org/background.php`. Last accessed January 2016.

[22] Niels Provos et al. A virtual honeypot framework. In *USENIX Security Symposium*, volume 173, 2004.

[23] Navneet Kambow and Lavleen Kaur Passi. Honeypots: The need of network security. *International Journal of Computer Science and Information Technologies, Vol. 5*, 2014.

[24] ISO/IEC. Information technology - security techniques - information security management systems - overview and vocabulary. *INTERNATIONAL STANDARD, ISO/IEC 27000*, 2009.

[25] Tejvir Kaur, Vimmi Malhotra, and Dheerendra Singh. Comparison of network security tools-firewall, intrusion detection system and honeypot. *nternational Journal of Enhanced Research in Science Technology & Engineering, ISSN*, pages 2319–7463, 2004.

[26] Lance Spitzner. The value of honeypots, part two: Honeypot solutions and legal issues. `http://www.symantec.com/connect/articles/value-honeypots-part-two-honeypot-solutions-and-legal-issues`. Last accessed on January 2016.

[27] Ram Dantu, Joao W. Cangussu, and Sudeep Patwardhan. Fast worm containment using feedback control. *IEEE Trans. Dependable Secur. Comput.*, 4(2):119–136, April 2007.

[28] German laws. Deutsch strafgesetzbuch - german criminal law code. `https://dejure.org/gesetze/StGB/202c.html`. Last accessed December 2015.

[29] Honeynet Project. *Know Your Enemy: Learning about Security Threats*. Addison-Wesley, 2004.

[30] Foundation for Research DFN-CERT and Technology Hellas. Network of affined honeypots, noah-projekt. `https://www.fp6-noah.org/publications/index.html`. Last accessed January 2016.

[31] Markus Kötter Based on work of Paul Bächer and Georg Wicherski. mwcollect alliance collaborative malware collection and sensing. `https://alliance.mwcollect.org/`. Last accessed January 2016.

[32] Georg Wicherski. Medium interaction honeypots. *German Honeynet Project*, 2006.

[33] DTAG Telekom. Frühwarnsystem, sicherheitstacho. `http://www.sicherheitstacho.eu/`. Last accessed January 2016.

[34] F Pouget, M Dacier, and VH Pham. Leurre.com: On the advantages of deploying a large scale distributed honeypot platform. In *Proceedings of the E-Crime and Computer Evidence Conference*, 2005.

[35] Fabien Pouget and Marc Dacier. White paper: Honeypot, honeynet: A comparative survey. Technical report, Technical Report RR-03-082, Institut Eurecom, 2003.

[36] Ali Ikinci, Thorsten Holz, and Felix C Freiling. Monkey-spider: Detecting malicious websites with low-interaction honeyclients. In *Sicherheit*, volume 8, pages 407–421, 2008.

[37] Noor Al-Gharabally, Nosayba El-Sayed, Sara Al-Mulla, and Imtiaz Ahmad. Wireless honeypots: survey and assessment. In *Proceedings of the 2009 conference on Information Science, Technology and Applications*, pages 45–52. ACM, 2009.

[38] Abdulrazzaq Almutairi, David Parish, and Raphael Phan. Survey of high interaction honeypot tools: Merits and shortcomings, 2012.

[39] Matthew L Bringer, Christopher A Chelmecki, and Hiroshi Fujinoki. A survey: Recent advances and future trends in honeypot research. *International Journal*, 4, 2012.

[40] All.Net & Affiliated Companies. The deception toolkit home page. `http://www.all.net/dtk/`. Last accessed on January 2016.

[41] Feng Zhang, Shijie Zhou, Zhiguang Qin, and Jinde Liu. Honeypot: a supplemented active defense system for network security. In *Parallel and Distributed Computing, Applications and Technologies, 2003. PDCAT'2003. Proceedings of the Fourth International Conference on*, pages 231–235. IEEE, 2003.

[42] Inc. Networks Associates Technology. Cybercop sting - getting started guide. `http://www.scn.rain.com/~neighorn/PDF/Cstguide.pdf`. Last accessed on December 2015.

[43] Ryan McGeehan Greg Smith Brian Engert Kevin Reedy Kevin Benes. Google hack

honeypot. http://ghh.sourceforge.net/. Last accessed on January 2016.

[44] Lukas Rist Glastopf Web Honeypot Project. Glastopf honeypot. glastopf.org/. Last accessed on January 2016.

[45] DEFCON X Proof of Concept 2005 Black Alchemy Enterprises. Fakeap. https://github.com/tbennett6421/Black-Alchemy---FakeAP. Last accessed on December 2015.

[46] Atomic Software Solutions. Honeybot - a medium interaction honeypot for windows. http://www.atomicsoftwaresolutions.com/. Last accessed on December 2015.

[47] Brazilian Distributed Honeypots Project. Announce: Hoacd 1.0 (bootable openbsd + honeyd cd). https://lwn.net/Articles/90941/. Last accessed on December 2015.

[48] Brazilian Distributed Honeypots Project. Honeyperl honeypot. http://sourceforge.net/projects/honeyperl/files/honeyperl/. Last accessed on December 2015.

[49] Inc. MicroSolved. Honeypoint security server. http://microsolved.com/HoneyPoint-server.html. Last accessed on December 2015.

[50] b4b0 security ziplock / Sickbeatz. Impost. http://impost.sourceforge.net/. Last accessed on December 2015.

[51] Roger A Grimes. *Honeypots for Windows*. Apress, 2005.

[52] Keyfocus. Kfsensor. http://www.keyfocus.net/kfsensor/. Last accessed on January 2016.

[53] Upi Tamminen (desaster). Kippo, ssh medium-interaction honeypot. https://github.com/desaster/kippo/. Last accessed on December 2015.

[54] Github Michel Oosterhof. Cowrie - medium-interaction honeypot. https://github.com/micheloosterhof/cowrie. Last accessed on December 2015.

[55] Jose Antonio Coret. Kojoney - a honeypot for the ssh service. http://kojoney.sourceforge.net/. Last accessed on January 2016.

[56] Justin Klein Keane University of Pennsylvania's School of Arts & Sciences. Kojoey2 ssh medium-interaction honeypot. https://github.com/madirish/kojoney2. Last accessed on December 2015.

[57] Tom Liston. Labrea: Sticky honeypot and ids. http://labrea.sourceforge.net/labrea-info.html. Last accessed on January 2016.

[58] Brent N Chun, Jason Lee, and Hakim Weatherspoon. Netbait: a distributed worm detection service. *Intel Research Berkeley Technical Report IRB-TR-03*, 33, 2003.

[59] NetBait Inc. Netbait service faq. http://netbaitinc.com/support/supp_faq.shtml. Last accessed on December 2015.

[60] OpenBSD. spamd - spam deferral daemon. http://www.openbsd.org/cgi-bin/man.cgi/OpenBSD-current/man8/spamd.8. Last accessed on December 2015.

[61] Honeypots.NET. Honeypot software, honeypot products, deception software. http://www.honeypots.net/honeypots/products. Last accessed on January 2016.

[62] Luis Wong. single-honeypot. http://single-honeypot.sourceforge.net/. Last

accessed on January 2016.

[63] Karl A. Krueger. Smtpot - a simple smtp honeypot. `http://mirror.unpad.ac.id/orari/library/library-sw-hw/linux-1/HONEYPOTS/smtpot/`. Last accessed on January 2016.

[64] druid@caughq.org Druid Cau. Spamhole. `http://sourceforge.net/projects/spamhole/`. Last accessed on January 2016.

[65] Neale Pikett. Spampot. `https://mail.python.org/pipermail/spambayes/2003-January/002883.html`. Last accessed on December 2015.

[66] NETSEC Inc. Specter. `http://www.specter.com/default50.htm`. Last accessed on January 2016.

[67] Symantec Inc. Symantec decoy server. `https://www.symantec.com/about/news/release/article.jsp?prid=20030623_01`. Last accessed on January 2016.

[68] George Bakos. Tiny honeypot. `https://github.com/tiny-honeypot/thp`. Last accessed on December 2015.

[69] Carnivore Project. Nepenthes - finest collection -. `http://nepenthes.carnivore.it/documentation:readme`. Last accessed December 2015.

[70] Paul Baecher, Markus Koetter, Thorsten Holz, Maximillian Dornseif, and Felix Freiling. The nepenthes platform: An efficient approach to collect malware. In *Recent Advances in Intrusion Detection*, pages 165–184. Springer, 2006.

[71] Carnivore Project. Dionaea honeypot. `http://dionaea.carnivore.it/`. Last accessed December 2015.

[72] Carnivore Project. libemu – x86 shellcode emulation. `http://libemu.carnivore.it/`. Last accessed on December 2015.

[73] Tillmann Werner. Honeytrap – a dynamic meta-honeypot daemon. `http://src.carnivore.it/honeytrap/`, `http://honeytrap.carnivore.it/details.html`. Last accessed January 2016.

[74] Google Summer of Code 2011. Network sinkhole, honeysink. `https://redmine.honeynet.org/projects/sinkhole`. Last accessed January 2016.

[75] Paul Bacher, Thorsten Holz, Markus Kotter, and Georg Wicherski. Know your enemy: Tracking botnets. `https://www.honeynet.org/papers/bots/`, 2005. Last accessed December 2015.

[76] Georg Wicherski. Placing a low-interaction honeypot in-the-wild: A review of mwcollectd. *Network Security*, 2010(3):7–8, 2010.

[77] MushMush Foundation at Github. Conpot. `https://github.com/mushorg/conpot`. Last accessed February 2016.

[78] Jordan Wright. elastichoney. `https://github.com/jordan-wright/elastichoney`. Last accessed February 2016.

[79] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. Iotpot: Analysing the rise of iot compromises. In *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, Washington, D.C., August 2015. USENIX Association.

[80] Github. Alex Bredo, EDAG Production Solutions. honeypot-camera. https://github.com/alexbredo/honeypot-camera. Last accessed February 2016.

[81] ThreatStream. Shockpot. https://github.com/threatstream/shockpot. Last accessed February 2016.

[82] Common Vulnerabilities and National Vulnerability Database (NVD) Exposures (CVE). Cve-2014-6271. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271. Last accessed February 2016.

[83] Andrew Michael Smith. bluepot. https://github.com/andrewmichaelsmith/bluepot. Last accessed February 2016.

[84] Markus Gruber, Dirk Hoffstadt, Adnan Aziz, Florian Fankhauser, Christian Schanes, Erwin Rathgeb, and Thomas Grechenig. Global voip security threats-large scale validation based on independent honeynets. In *IFIP Networking Conference (IFIP Networking), 2015*, pages 1–9. IEEE, 2015.

[85] Rodrigo Do Carmo, Mohamed Nassar, and Olivier Festor. Artemisa: An open-source honeypot back-end to support security in voip domains. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 361–368. IEEE, 2011.

[86] Collin Mulliner, Steffen Liebergeld, and Matthias Lange. Poster: Honeydroid-creating a smartphone honeypot. In *IEEE Symposium on Security and Privacy*, 2011.

[87] Georgios Portokalidis, Asia Slowinska, and Herbert Bos. Argos: An emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, EuroSys '06, pages 15–27, New York, NY, USA, 2006. ACM.

[88] Honeynet Project. Honeywall cdrom project documentation. https://www.honeynet.org/project/HoneywallCDROM. Last accessed on December 2015.

[89] Michael Mueter, Felix Freiling, Thorsten Holz, and Jeanna Matthews. A generic toolkit for converting web applications into high-interaction honeypots. *University of Mannheim*, 280, 2008.

[90] Michael Mueter, Felix Freiling, Thorsten Holz, and Jeanna Matthews. High interaction honeypot analysis tool. http://sourceforge.net/projects/hihat/. Last accessed on December 2015.

[91] Chinese Honeynet project. Honeybow sensor. http://sourceforge.net/projects/honeybow/. Last accessed on December 2015.

[92] Honeynet Project. Sebek - a data capture tool. https://projects.honeynet.org/sebek/. Last accessed December 2015.

[93] The Honeynet Project. Sebek a kernel based data capture tool. *Know Your Enemy Journal*, November 2003.

[94] Jose Nazario. Phoneyc: A virtual client honeypot. In *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, pages 6–6. USENIX Association, 2009.

[95] J Nazario. phoneyc python honeyclient. `https://github.com/honeynet/phoneyc`. Last accessed December 2015.

[96] The Honeynet Project. Honeyc. `https://projects.honeynet.org/honeyc`. Last accessed January 2016.

[97] Angelo Dell'Aera. Thug honeypot. `https://github.com/buffer/thug/`. Last accessed on January 2016.

[98] Masood Mansoori, Ian Welch, and Qiang Fu. Yalih, yet another low interaction honeyclient. In *Proceedings of the Twelfth Australasian Information Security Conference - Volume 149*, AISC '14, pages 7–15, Darlinghurst, Australia, Australia, 2014. Australian Computer Society, Inc.

[99] Ritchie Lam Qiaowei) Victoria University of Wellington (Masood Mansoori) Singapore Polytechnic (Lai Qi Wei. Yalih - yet another low interaction honeyclient. `https://github.com/Masood-M/YALIH`. Last accessed on January 2016.

[100] Niels Provos. Spybye - finding malware. `http://www.monkey.org/~provos/spybye/`. Last accessed January 2016.

[101] Ali Ikinci. The monkey-spider project. `http://monkeyspider.sourceforge.net/`. Last accessed January 2016.

[102] Andreas Dewald, Thorsten Holz, and Felix C Freiling. Adsandbox: Sandboxing javascript to fight malicious websites. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 1859–1864. ACM, 2010.

[103] New Zealand Honeynet Project and Christian Seifert Victoria University of Wellington. Capture-bat download page. `https://www.honeynet.org/node/315` (reupload). Last accessed on December 2015.

[104] MITRE. Honeyclient project. `https://github.com/dkindlund/honeyclient`. Last accessed December 2015.

[105] Kathy Wang. Using honeyclients to detect new attacks. In *RECON Conference (June 2005)*, 2005.

[106] Christian Seifert honeynet project and Ramon Steenson of the New Zealand Chapter. Capture-hpc client honeypot. `https://projects.honeynet.org/capture-hpc`. Last accessed December 2015.

[107] Microsoft Research Team. Strider honeymonkey exploit detection. `http://research.microsoft.com/en-us/um/redmond/projects/strider/honeymonkey/`. Last accessed January 2016.

[108] Yi-Min Wang, Doug Beck, Xuxian Jiang, Roussi Roussev, Chad Verbowski, Shuo Chen, and Sam King. Automated web patrol with strider honeymonkeys. In *Proceedings of the 2006 Network and Distributed System Security Symposium*, pages 35–49, 2006.

[109] Australian Honeynet Project. Trigona. `http://honeynet.org.au/?q=node/63`. Last accessed on December 2015 using the web archive.

[110] National Cyber Security Centre (Netherlands) NASK/CERT Polska. Honeyspider network. `http://www.honeyspider.net/`. Last accessed on December 2015.

[111] Herbert Bos Joan Robert Rocaspana at Vrije Universiteit Amsterdam. Shelia: a client-side honeypot for attack detection. `https://www.cs.vu.nl/~herbertb/misc/shelia/`. Last accessed on January 2016.

[112] Alexander Moshchuk, Tanya Bragin, Steven D Gribble, and Henry M Levy. A crawler-based study of spyware in the web. In *NDSS*, 2006.

[113] Stuttgart Thomas Müller Benjamin Mack Mehmet Arziman, Hochschule der Medien (HdM). Web exploit finder. `http://www.xnos.org/security/web-exploit-finder.html`. Last accessed January 2016.

[114] Mengjun Xie, Zhenyu Wu, and Haining Wang. Honeyim: Fast detection and suppression of instant messaging malware in enterprise-like networks. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 64–73. IEEE, 2007.

[115] TOBIAS JARMUZEK Shahriyar Jalayeri. Pwnypot, high interaction client honeypot. `https://github.com/shjalayeri/pwnypot`. Last accessed January 2016.

[116] Team Violating. Bait and switch honeypot. `http://baitnswitch.sourceforge.net/`. Last accessed on December 2015.

[117] Inc. Activeworx. Honeynet security console. `http://seclists.org/focus-ids/2004/May/74`. Last accessed on December 2015.

[118] Google Summer of Code GSOC 2009. Honeyweb. `https://code.google.com/p/gsoc-honeyweb/`. Last accessed on January 2016.

[119] Kevin Timm. Honeyweb - http honeypot. `http://www.citi.umich.edu/u/provos/honeyd/contrib/ktimm/`. Last accessed on December 2015.

[120] The Honeynet Project. Honeysnap. `https://projects.honeynet.org/honeysnap/`. Last accessed on January 2016.

[121] Carnivore Project (Tillmann Werner). Pe hunter. `http://src.carnivore.it/pehunter/`.

[122] Portugese Chapter Honeynet Project. Honeymole. `http://www.honeynet.org.pt/index.php/HoneyMole`. Last accessed on January 2016 using the web archive.

[123] C(yber) Brian. Dionaea honeypot obfuscation. `https://www.cyberbrian.net/2014/09/dionaea-honeypot-obfuscation/`. Last accessed January 2016.

[124] Andrew Morris. Detecting kippo ssh honeypots, bypassing patches, and all that jazz. `http://morris.guru/detecting-kippo-ssh-honeypots/`. Last accessed December 2015.

[125] Xinwen Fu, Wei Yu, Dan Cheng, Xuejun Tan, Kevin Streff, and Steve Graham. On recognizing virtual honeypots and countermeasures. In *Dependable, Autonomic and Secure Computing, 2nd IEEE International Symposium on*, pages 211–218. IEEE, 2006.

[126] Thorsten Holz and Frederic Raynal. Detecting honeypots and other suspicious environments. In *Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC*, pages 29–36. IEEE, 2005.

[127] S Mukkamala, K Yendrapalli, R Basnet, MK Shankarapani, and AH Sung. Detection

of virtual environments and low interaction honeypots. In *Information Assurance and Security Workshop, 2007. IAW'07. IEEE SMC*, pages 92–98. IEEE, 2007.

[128] Cliff C Zou and Ryan Cunningham. Honeypot-aware advanced botnet construction and maintenance. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pages 199–208. IEEE, 2006.

[129] Ping Wang, Lei Wu, Ryan Cunningham, and Cliff C Zou. Honeypot detection in advanced botnet attacks. *International Journal of Information and Computer Security*, 4(1):30–51, 2010.

[130] Joseph Corey. Local honeypot identification, phrack (unofficial), volume 0x0b, issue 0x3e. `http://repo.hackerzvoice.net/depot_ouah/p62-0x07.txt`. Last accessed January 2016.

[131] Joanna Rutkowska. Red pill... or how to detect vmm using (almost) one cpu instruction. `http://repo.hackerzvoice.net/depot_ouah/Red_%20Pill.html`. Last accessed January 2016.

[132] Fabien Pouget, Marc Dacier, et al. Honeypot-based forensics. In *AusCERT Asia Pacific Information Technology Security Conference*, 2004.

[133] Jérôme Francois, Olivier Festor, et al. Activity monitoring for large honeynets and network telescopes. *International Journal on Advances in Systems and Measurements*, 1(1):1–13, 2008.

[134] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, Masashi Eto, Daisuke Inoue, and Koji Nakao. Statistical analysis of honeypot data and building of kyoto 2006+ dataset for nids evaluation. In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, pages 29–36. ACM, 2011.

[135] Robert McGrew. Experiences with honeypot systems: Development, deployment, and analysis. In *System Sciences, 2006. HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on*, volume 9, pages 220a–220a. IEEE, 2006.

[136] Aarjav J Trivedi, Paul Q Judge, and Sven Krasser. Analyzing network and content characteristics of spim using honeypots. In *Proceedings of the 3rd USENIX SRUTI*, 2007.

[137] Craig Valli. An analysis of malfeasant activity directed at a voip honeypot. *Proceedings of the 8th Australian Digital Forensics Conference*, 2010.

[138] Mohamed Kaaniche, Yves Deswarte, Eric Alata, Marc Dacier, and Vincent Nicomette. Empirical analysis and statistical modeling of attack processes based on honeypots. *arXiv preprint arXiv:0704.0861*, 2007.

[139] Saleh Almotairi, Andrew Clark, George Mohay, and Jacob Zimmermann. Characterization of attackers' activities in honeypot traffic using principal component analysis. In *Network and Parallel Computing, 2008. NPC 2008. IFIP International Conference on*, pages 147–154. IEEE, 2008.

[140] Olivier Thonnard and Marc Dacier. A framework for attack patterns' discovery in honeynet data. *digital investigation*, 5:S128–S139, 2008.

[141] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Zmap: Fast internet-wide scanning and its security applications. In *Usenix Security*, pages 605–620, 2013.

[142] Anil K Jain and Richard C Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[143] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666, 2010.

[144] Paul Barford, Jeffery Kline, David Plonka, and Amos Ron. A signal analysis of network traffic anomalies. In *Proceedings of the 2Nd ACM SIGCOMM Workshop on Internet Measurment*, IMW '02, pages 71–82, New York, NY, USA, 2002. ACM.

[145] Jake D. Brutlag. Aberrant behavior detection in time series for network monitoring. In *Proceedings of the 14th USENIX Conference on System Administration*, LISA '00, pages 139–146, Berkeley, CA, USA, 2000. USENIX Association.

[146] Anukool Lakhina, Mark Crovella, and Christiphe Diot. Characterization of network-wide anomalies in traffic flows. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04, pages 201–206, New York, NY, USA, 2004. ACM.

[147] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. *SIGCOMM Comput. Commun. Rev.*, 35(4):217–228, August 2005.

[148] Leonid Portnoy, Eleazar Eskin, and Sal Stolfo. Intrusion detection with unlabeled data using clustering. In *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001*. Citeseer, 2001.

[149] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo. A geometric framework for unsupervised anomaly detection. In *Applications of data mining in computer security*, pages 77–101. Springer, 2002.

[150] Philippe Owezarski. A near real-time algorithm for autonomous identification and characterization of honeypot attacks. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 531–542, New York, NY, USA, 2015. ACM.

[151] Sven Krasser, Gregory Conti, Julian Grizzard, Jeff Gribschaw, and Henry Owen. Real-time and forensic network data analysis using animated and coordinated visualization. In *Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC*, pages 42–49. IEEE, 2005.

[152] Alvin C Rencher. *Methods of multivariate analysis*, volume 492. John Wiley & Sons, 2003.

[153] Johan Mazel, Pedro Casas, and Philippe Owezarski. Sub-space clustering and evidence accumulation for unsupervised network anomaly detection. In *Proceedings of the Third International Conference on Traffic Monitoring and Analysis*, TMA'11, pages 15–28, Berlin, Heidelberg, 2011. Springer-Verlag.

[154] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. *SIGMOD Rec.*, 27(2):94–105, June 1998.

[155] André Grégio, Rafael Santos, and Antonio Montes. Evaluation of data mining techniques for suspicious network activity classification using honeypots data. In *Defense and Security Symposium*, pages 657006–657006. International Society for Optics and

Photonics, 2007.

[156] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998.

[157] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[158] Christian Kreibich and Jon Crowcroft. Honeycomb: Creating intrusion detection signatures using honeypots. *SIGCOMM Comput. Commun. Rev.*, 34(1):51–56, January 2004.

[159] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[160] James Newsome, Brad Karp, and Dawn Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Security and Privacy, 2005 IEEE Symposium on*, pages 226–241. IEEE, 2005.

[161] Yong Tang and Shigang Chen. Defending against internet worms: A signature-based approach. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 2, pages 1384–1394. IEEE, 2005.

[162] Mohssen MZE Mohammed, H Anthony Chan, Neco Ventura, Mohsim Hashim, Izzeldin Amin, and Eihab Bashier. Detection of zero-day polymorphic worms using principal component analysis. In *Networking and Services (ICNS), 2010 Sixth International Conference on*, pages 277–281. IEEE, 2010.

[163] Marc Dacier, Fabien Pouget, and Hervé Debar. Attack processes found on the internet. Technical report, DTIC Document, 2004.

[164] Jan Kohlrausch. Experiences with the noah honeynet testbed to detect new internet worms. In *IT Security Incident Management and IT Forensics, 2009. IMF'09. Fifth International Conference on*, pages 13–26. IEEE, 2009.

[165] Eric Alata, Vincent Nicomette, Marc Dacier, Matthieu Herrb, et al. Lessons learned from the deployment of a high-interaction honeypot. *arXiv preprint arXiv:0704.0858*, 2007.

[166] Fabien Pouget, Marc Dacier, and Van-Hau Pham. Understanding threats: a prerequisite to enhance survivability of computing systems. *International Journal of Critical Infrastructures*, 4(1-2):153–171, 2008.

[167] Pele Li, Mehdi Salour, and Xiao Su. A survey of internet worm detection and containment. *Communications Surveys & Tutorials, IEEE*, 10(1):20–35, 2008.

[168] Georgios Portokalidis and Herbert Bos. Sweetbait: Zero-hour worm detection and containment using low-and high-interaction honeypots. *Computer Networks*, 51(5):1256–1274, 2007.

[169] Matthias Wählisch, Sebastian Trapp, Christian Keil, Jochen Schönfelder, Jochen Schiller, et al. First insights from a mobile honeypot. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 305–306. ACM, 2012.

[170] Matthias Wählisch, André Vorbach, Christian Keil, Jochen Schönfelder, Thomas C Schmidt, and Jochen H Schiller. Design, implementation, and operation of a mobile honeypot. *arXiv preprint arXiv:1301.7257*, 2013.

[171] Chris Stucchio. Don't use hadoop - your data isn't that big. `https://www.chrisstucchio.com/blog/2013/hadoop_hatred.html`. Last accessed January 2016.

[172] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[173] Wes McKinney. *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, Inc., 2012.

[174] Hassan Artail, Haidar Safa, Malek Sraj, Iyad Kuwatly, and Zaid Al-Masri. A hybrid honeypot framework for improving intrusion detection systems in protecting organizational networks. *computers & security*, 25(4):274–288, 2006.

[175] John P John, Fang Yu, Yinglian Xie, Arvind Krishnamurthy, and Martín Abadi. Heat-seeking honeypots: design and experience. In *Proceedings of the 20th international conference on World wide web*, pages 207–216. ACM, 2011.

[176] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. A multifaceted approach to understanding the botnet phenomenon. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 41–52. ACM, 2006.

[177] Ioannis Koniaris, Georgios Papadimitriou, and Petros Nicopolitidis. Analysis and visualization of ssh attacks using honeypots. In *EUROCON, 2013 IEEE*, pages 65–72. IEEE, 2013.

[178] Kyumin Lee, James Caverlee, and Steve Webb. The social honeypot project: protecting online communities from spammers. In *Proceedings of the 19th international conference on World wide web*, pages 1139–1140. ACM, 2010.

[179] RIPE Network Coordination Centre. Routing information service (ris), ris raw data. `https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/ris-raw-data`. Last accessed January 2016.

[180] Internet Engineering Task Force (IETF). Rfc 6890, special-purpose ip address registries. `https://tools.ietf.org/html/rfc6890`.

[181] CIDR REPORT. Summary of total route table size for the past 7 days. `http://www.cidr-report.org/as2.0/#General_Status`. Last accessed on November 2012.

[182] Martin Arlitt. Characterizing web user sessions. *ACM SIGMETRICS Performance Evaluation Review*, 28(2):50–63, 2000.

[183] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, volume 22, pages 207–216. ACM, 1993.

[184] George EP Box, Gwilym M Jenkins, and Gregory C Reinsel. *Time series analysis: forecasting and control*, volume 734. John Wiley & Sons, 2011.

[185] Robert T Morris. A weakness in the 4.2 bsd unix tcp/ip software. *Issue 117 of Computing science technical report*, 1985.

[186] IPV6Test. Overall ipv6 and v4 protocol support, monthly statistics. `http://ipv6-test.com/stats/`. Last accessed February 2016.