ABSTRACT

Title of Document: Advanced Honeypot Architecture for Network Threats Quantification

Robin Berthier, Ph.D., 2009

Directed By: Associate Professor Michel Cukier, Reliability Engineering Program

Today's world is increasingly relying on computer networks. The increase in the use of network resources is followed by a rising volume of security problems. New threats and vulnerabilities are discovered everyday and affect users and companies at critical levels, from privacy issues to financial losses. Monitoring network activity is a mandatory step for researchers and security analysts to understand these threats and to build better protections. Honeypots were introduced to monitor unused IP spaces to learn about attackers. The advantage of honeypots over other monitoring solutions is to collect only suspicious activity. However, current honeypots are expensive to deploy and complex to administrate especially in the context of large organization networks.

This study addresses the challenge of improving the scalability and flexibility of honeypots by introducing a novel hybrid honeypot architecture. This architecture is based on a Decision Engine and a Redirection Engine that automatically filter attacks and save resources by reducing the size of the attack data collection and allow

researchers to actively specify the type of attack they want to collect. For a better integration into the organization network, this architecture was combined with network flows collected at the border of the production network. By offering an exhaustive view of all communications between internal and external hosts of the organization, network flows can 1) assist the configuration of honeypots, and 2) extend the scope of honeypot data analysis by providing a comprehensive profile of network activity to track attackers in the organization network. These capabilities were made possible through the development of a passive scanner and server discovery algorithm working on top of network flows. This algorithm and the hybrid honeypot architecture were deployed and evaluated at the University of Maryland, which represents a network of 40,000 computers.

This study marks a major step toward leveraging honeypots into a powerful security solution. The contributions of this study will enable security analysts and network operators to make a precise assessment of the malicious activity targeting their network.

ADVANCED HONEYPOT ARCHITECTURE FOR NETWORK THREATS QUANTIFICATION

By

Robin G. Berthier

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2009

Advisory Committee:
Associate Professor Michel Cukier, Chair
Associate Professor Jeff Foster
Associate Professor Atif Memon
Professor Ali Mosleh
Associate Professor Gang Qu

Acknowledgements

I would first like to thank my doctoral committee: Jeff Foster, Atif Memon, Ali Mosleh, and Gang Qu for their valuable feedback.

I would like then to thank my family for their constant support and encouragement. Moving to a new continent could have been challenging but you greatly contributed to make it an amazing and always joyful experience. At a professional level, I would like to thank my brother, Yann, for introducing me to the security community and for the numerous fruitful discussions we had that helped me make the right decisions.

I would like to thank the friends I met during graduate school, and the students with whom I worked during the past four years. I am grateful to Jorge Arjona, Pierre-Yves Dion, Thibaut Gadiolet, Jesus Molina, Susmit Panjwani, Daniel Ramsbrock, and Stephanie Tan for their help and involvement through out the numerous honeypot experiments we conducted. I would also like to acknowledge Jeannie Boeffel, Virginie Klein and Bruno Muscolino for their contributions with Netflow. I am also deeply grateful for the work achieved by Thomas Coquelin, and Julien Vehent on Honeybrid. Finally, I would like to thank Bertrand Sobesto for his great assistance in running the lab, and Keith Jarrin for his contagious enthusiasm and for his strong support before the defense.

I would like to express my gratitude to the people from OIT, especially Gerry Sneeringer from Security, Tim Shortall, and Karl Reuss from NTS. I would also like

also to thank Jeff McKinney from ISR. Without your help and the resources you allowed me to access, this research would simply not have been possible.

I am also very grateful to the researchers from AT&T Labs Research, including Gregg Vesonder, Matti Hiltunen, Dave Kormann, Dan Sheleheda, and John Wright. Being able to work at Florham Park and to collaborate with you over the past two years was an amazing opportunity.

I wish to thank my advisor, Michel Cukier, with whom I was honored to work during the past four years. I am sincerely grateful for your guidance and for having provided me so many opportunities. Working with you has been a unique experience and I am looking forward to pursuing our collaboration.

Finally, and most importantly, I want to thank Danielle for her unfailing support. Being able to share this journey with you at both a professional and personal levels has been an incredible blessing.

Table of Contents

List of Tables

List of Figures

CHAPTER I

INTRODUCTION

1. Introduction

Today's world increasingly relies on computer networks. The use of network resources is growing and network infrastructures are gaining in size and complexity. This increase is followed by a rising volume of security problems. New threats and vulnerabilities are found everyday, and computers are far from being secure. In the first half of 2008, 3,534 vulnerabilities were disclosed by vendors, researchers and independants [42]. Between 8 and 16% of these vulnerabilities were exploited the day they were released by malicious programs [42]. The consequences affect users and companies at critical levels, from privacy issues to financial losses [68].

To address this concern, network operators and security researchers have developed and deployed a variety of solutions. The goal of these solutions is two-fold: first to monitor, and second to protect network assets. Monitoring allows researchers to understand the different threats. Data are being collected to better characterize and quantify malicious activity. The goal of this dissertation is to introduce an innovative framework to better measure malicious threats in the organization network. The framework is based on a flexible hybrid honeypot architecture that we integrate with the organization network using network flows.

2. Background

2.1. Network Security

Network malicious activity can be quantified and characterized through two distinct approaches: the first is to monitor production networks, where live hosts and devices are actually used by people; the second is to monitor an unused address space that nobody uses. The advantage of the second approach over the first is that there is no user traffic to filter out. Indeed, the traffic received by unused addresses falls into three categories: malicious activity, misconfiguration, and backscatter from spoofed addresses [55]. On the other hand, the disadvantage of the second approach is to rely on the assumption that malicious activity destined to unused addresses is similar to the one targeting production machines.

Tools used in these two different approaches can be divided into two groups: passive and active tools. When monitoring production networks, passive security tools include intrusion detection systems (IDSs) such as Snort [70], and network traffic sniffers such as Tcpdump [80]tc or Netflow [58]. Active tools include firewalls such as Netfilter [57], intrusion prevention systems (IPSs) such as Snort Inline [74], and vulnerability scanners such as Nessus [30]. When monitoring an unused address space, passive tools are similar, but active tools are specific sensors developed with the only goal of better investigating the malicious activity received. Historically, unused address spaces were only passively monitored. Then researchers had the idea of actively replying to the traffic received to discover the exact threat behind each connection attempt. To understand the research challenges introduced with this new idea, we will now describe the different existing types of active sensors.

2.2. Honeypots

2.2.1. Definitions

The following vocabulary is important to understand the remaining parts of this dissertation:

- We define a **network sensor** as an unused IP address instrumented to collect information about suspicious traffic. We separate sensors into two categories: *passive sensors,* which simply collect data without any interaction with the source of traffic; and *active sensors,* which can interact with the source of traffic to collect additional information.
- We define a **honeypot** as a network device that provides a mechanism for completing network connections not normally provided on a system and logging those connection attempts [19]. We note that honeypot and active network sensor are synonyms.
- We define a **darknet** as a network of passive sensors.
- Similarly, we define a **honeynet** as a network of honeypots.
- By **honeypot architecture**, we mean a specific combination of software solutions to administrate a honeynet.
- Finally by **honeypot framework**, we mean the combination of a honeypot architecture and a data processing solution to analyze malicious network activity.

2.2.2. Honeypot Attributes and Classification

The main goal of honeypots is to provide information about network attacks. A large variety of honeypots have been proposed by researchers to collect various types of security threats. These honeypots can be organized according to three main attributes:

- *Fidelity*: honeypots have different levels of interaction, whether they offer emulated or real services to attackers. The more interactions a honeypot has with an attacker, the more knowledge is gained about the attack. Hence, three different levels of interaction are defined:

    1. A *high-interaction honeypot* is a conventional network resource, such as a computer or router, with no active user and no specific task other than getting attacked. From an attacker's point of view, this type of honeypot can hardly be differentiated from another production machine. The advantage is to gain as much information as possible about the attack. Of course, with such a genuine exposure, the risk of being effectively compromised is real. Consequently, these honeypots should be closely monitored and data control mechanisms, such as a reverse firewall, should be configured to prevent an attacker from using the honeypot to damage other production resources. The Honeynet Project [40] provides tools and documentation to deploy and administrate this type of honeypot.

    2. A *low-interaction honeypot* provides limited interaction with the attacker by emulating a set of services. The goal of low-interaction honeypots is to gather information about the first steps of an attack. Information about the motivation of the threats received is rarely captured because the level of interaction is too low for the honeypot to be effectively compromised. A well-known implementation of a low-interaction honeypot is Honeyd [62].

    3. A *zero-interaction sensor* is no longer a honeypot but a passive sensor that does not respond to attackers. Such sensors are called darknets and are

nonetheless able to collect important information about how attackers probe networks and what services they target. Relevant darknet projects are [6] and [88].

- *Scalability*: the level of interaction of honeypots affects the number of IP addresses on which the honeypots can be deployed as well as the maximum bandwidth they can sustain. Indeed, a darknet is more scalable than a set of high-interaction honeypots because, from a resource perspective, passively monitoring thousands of network addresses is less demanding than deploying and administrating a few high-interaction honeypots. As a result, current honeypot architectures offer either large scalability or high interaction but not both.
- *Security*: as explained for high interaction honeypots, deploying honeypots to actively collect malicious traffic is not a safe activity. Honeypots can be compromised and so several protection systems currently exist to avoid attackers from using honeypots to relay malicious activity.

Honeypots are governed by these three contending attributes: *scalability*, *fidelity* and *security*. Researchers have to balance these attributes according to their needs and their resources. They can either study the first steps of an attack by deploying a large number of low interaction honeypots. Or they can study the full attack process by deploying high interaction honeypots. This last option requires constant monitoring and important software and hardware resources.

2.2.3. Honeypots and Network Attack Processes

To better understand how honeypots can be used, it is important to first describe how network attacks proceed to spread and to compromise computers. For this purpose, we

define an attack process as a sequence of network communications between an attacker and a victim, with a malicious purpose. We divide the network attack process in three phases:

1. The first phase is to reach a victim, which means to send a communication attempt to a specific service hosted on a network device. For example, a technique for an attacker to discover a large number of victims is to scan incrementally all network addresses within a specific subnet. The attacker goes to the next phase only if the victim replies and the service is open to the attacker.

2. The second phase is to exploit the service found on the victim's machine by launching an attack payload. There is not always a clear boundary between the first and second phase, because some attacks are made of a single network packet [54], so communication attempts and attack payload overlap. Moreover, attackers often use the connection initialized during the scan to send the attack payload. The attacker goes to the next phase only if the service is successfully compromised by the attack launched.

3. The third phase is to use the newly corrupted victim's machine. The attacker can be someone who wants to gain access to a specific resource, or a worm that is simply spreading from one vulnerable machine to another. In such case, the worm installed on the newly corrupted machine will start probing for other victims and will create a new attack process starting with phase one again.

From this model we can map the different phases of network attack with the different types of honeypots. Figure 1 details this mapping and explains how honeypots attributes are related.

The probing phase of an attack can be detected by all types of sensors (zero-interaction sensors, low and high-interaction honeypots). However, to gather significant statistical results about scanning techniques and services targeted, one need to monitor large address space. Therefore, darknets are the most suitable solution to study this attack phase because of their high scalability.

The second phase of an attack can be detected only by sensors which can reply to probes. The reason is that an attack payload can be sent by the attacker to the sensor only if a network connection is correctly established between the two peers. As we just saw when explaining the second phase of an attack, we can find some exceptions to this requirement, because some attacks are made of a single network packet that does not need first an acknowledgment from the victim to be sent. Low interaction honeypots are well suited to gather exploits sent during the second phase of an attack, because they are scalable and provide enough interaction for the attacker to send its attack payload. However, emulated scripts hosted by low interaction honeypots will not always satisfy the level of interaction required by complex attacks. This threshold between simple and complex attacks is represented by the level of emulation on Figure 1. Furthermore, low interaction honeypots cannot be compromised by attackers, so the third phase of the attack process is never collected by this type of architecture.

As a result, the full attack process requires high interaction honeypots to be analyzed in detail. High interaction honeypots are not only able to collect complex exploits, they can also collect the third phase of an attack, which is how the attacker will use the compromised resource. This phase gives information regarding the motivation of the attack. For example, attacks can lead to the installation of a rogue software to provide

illicit services to the attacker community, such as a botnet client, illegal file sharing or hidden remote control. Of course, high interaction honeypots should be closely monitored to learn enough of the attacker's actions while staying under control. The risk is to have an attacker being able to use the honeypot to attack external production resources. Thus, the amount of information gathered on the attack will depend on the level of control deployed in the honeypot architecture. This level of control is represented on Figure 1 as the boundary between high interaction honeypots and vanilla systems such as live hosts. This requirement to closely monitor and control honeypots directly reduces the scalability. Moreover, high interaction honeypots, even if ran on virtual machines such as VMWare [85], need important hardware resources.

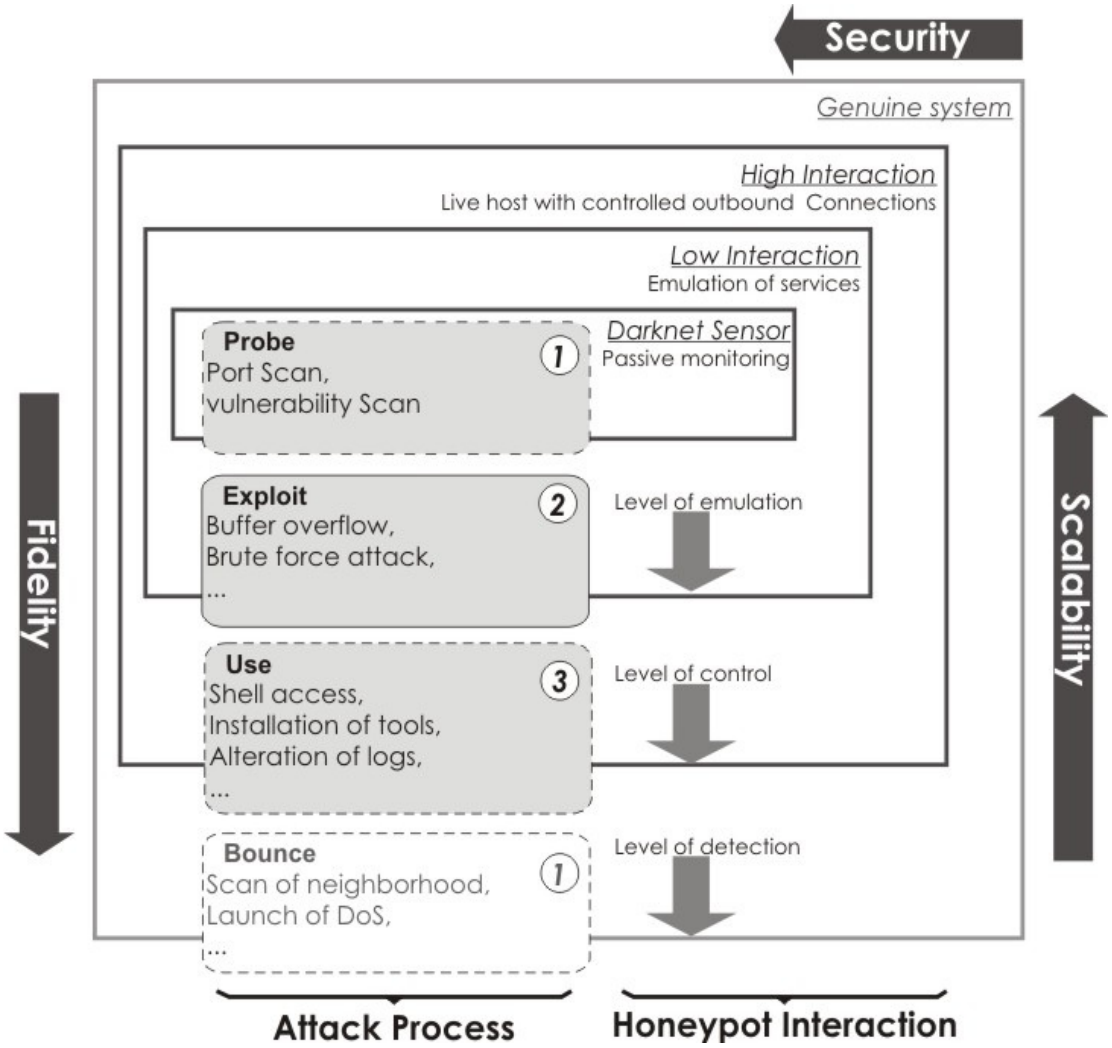

Figure 1: Different honeypot architectures to collect different phases of network attack process

3. Related Work

This section provides a broad overview of relevant solutions to collect attack processes using passive sensors and honeypots. A more specific review of related literature per topic is given at the beginning of Chapters 2 to 5.

3.1. Darknets

The idea of passively monitoring unused IP space to learn about suspicious traffic has spread through several research projects with various names. First, the Network

Telescope [53] [56] has pioneered the use of darknets to learn about denial of service attacks [55]. Another project called Blackholes [75] was able to collect traffic from 16.8 million IP addresses (1/256th of the Internet) to study global trends of worm activity. The Darknet project [29] provides a full guide to learn how to configure a darknet and start monitoring malware traffic.

While darknets offer the greatest scalability of all monitoring solutions, they are greatly limited by the lack of active responders. The Internet Motion Sensor [5] and iSink [88] are two research projects that implemented stateless active responders to darknets. As a result, they were able to keep a high scalability while capturing the first attack payloads sent by attackers. These two architectures provided important discoveries on the attributes of unused IP space to understand the differences in traffic collected. They are at the transition between passive darknets and active honeypots.

3.2. Low Interaction Honeypots

The most widely used low interaction honeypot is Honeyd [62]. Honeyd can create a population of virtual hosts on a network using unassigned IP addresses. Each host can be configured with a set of emulated services and a specific operating system behavior. The simplicity and flexibility of Honeyd makes it a relevant solution to host a complete low interaction honeynet. However, attacks collected depend on the interaction provided by the emulated services, and developing these services is often a difficult challenge.

Another well-known low interaction honeypot is Nepenthes [2]. Nepenthes was designed to automatically capture malware that spread from one computer to another. It consists of a set of emulated vulnerabilities that give enough interaction to capture the

infection attempt of malware. Then Nepenthes examines the attack payload and tries to download the remaining part of the malware.

3.3. High Interaction Honeypots

The Honeynet Project [40] has developed a variety of tools to help researchers deploying their honeynet and analyzing suspicious network traffic. One of these tools called Honeywall [41] was especially designed to administrate high interaction honeypots. It provides a web interface to monitor the data collection, and a reverse firewall to control outgoing connections from potentially compromised honeypots. Honeywall also integrates system monitoring capabilities through the Sebek kernel module [71].

The more cost efficient solution to host high interaction honeypots is to use virtual machines. Compared to genuine systems, virtual environments have the important advantage of being easier to monitor, to save and to clean after a successful compromise. The current virtual machine solutions are: VMWare [85], VirtualBox [83], Qemu [13], User Mode Linux [33], Xen [11] and Virtual PC [82] from Microsoft.

A recent project based on Qemu and called Argos [59] allows memory tainting to follow network data inside the virtualized environment. As a result, attacks based on buffer overflows can be immediately detected, analyzed and associated to a remote network attacker.

3.4. Hybrid Honeypots

The need to collect detailed attack processes on large IP spaces has pushed researchers to invent more scalable and intelligent architectures. These projects fall into the category of hybrid honeypot architecture. We provide a detailed comparison between our approach

and existing solutions at the beginning of Chapter 4, when we introduce our Honeybrid architecture.

Collapsar [43] simplifies the deployment and administration of high interaction honeypots on large IP spaces by using GRE tunnels to route traffic from distributed networks into a centralized farm of honeypots. The limitation of Collapsar is to not provide any filtering mechanism that can prevent high interaction honeypots from being overloaded.

Another project called Potemkin [86] is based on the idea that idle high interaction honeypots do not even need to run. As a result, the architecture saves resources by starting a new virtual machine for each active IP address. As soon as an IP address becomes inactive, the virtual machine is destroyed to save physical memory and CPU resources. Such a system allows hundreds of virtual machines to run on a single physical host.

4. Publications

We ran different experiments at the University of Maryland using honeypots to quantify and analyze malicious threats targeting the campus network. The results of these experiments have been released in the following publications:

- In [28] we investigated which network characteristics would be the most relevant to automatically separate network attacks collected by high interaction honeypots. During four months, we deployed two high interaction honeypots running Windows 2000 with a set of 25 vulnerabilities. We discovered that 78% of the 4,707 attacks collected tried to compromise the Microsoft Netbios service. Thus, we focused on investigating how attackers exploited Netbios vulnerabilities and

we defined a methodology based on the K-Means clustering algorithm to automatically classify the different attacks received. We presented our findings at the International Conference on Dependable Systems and Networks 2006 (DSN '06, 24% acceptance rate). In [16] that was published in the International Journal of Security and Networks, we extended our analysis of automatic attack classification by running the K-Means algorithm on combinations of characteristics.

For these two experiments, we collected attacks using high interaction honeypots because we did not want emulated services from low interaction honeypots to limit attackers. We closely monitored and controlled outbound connections and we re-initialized the honeypots when we detected that they were compromised. Thus, we collected little to no information about the third phase of attack processes (when attackers can use the compromised system).

- This third phase of the attack process was the focus of our next experiment, where we decided to study the motivation of attackers trying to compromise SSH servers. We deployed a set of four high interaction honeypots running Linux and SSH servers with simple passwords. During a first experiment of 24 days, we collected on average 2,805 connection attempts per computer per day. Out of these, 824 logged in successfully. From this significant number of successful compromised sessions, we were able to draw a state machine of the different actions taken by attackers to use the corrupted honeypots. For example, we discovered that most of the attackers started by changing the password of the compromised account, then checking the computer configuration, to finally

download and install a rogue software. The results of this analysis were published in [65] and presented at the International Conference on Dependable Systems and Networks 2007 (DSN '07, 25% acceptance rate).

- We then ran a second experiment for a longer period of 8 months, during which we recorded a total of 1,171 attack sessions. In these sessions, attackers typed a total of 20,335 commands that we could more precisely categorize in 24 specific actions. These actions were then analyzed based on the type of rogue software installed by attackers. This experiment is described in detail in Chapter 2 to illustrate the significance of high interaction honeypots to learn about attackers. Our findings have been published in [14] and will be presented at the International Conference on Dependable Systems and Networks 2009 (DSN '09) next June.

- To understand the global or local characteristics of network attacks, we then deployed two identical high interaction honeynets in two different locations: an academic network (at the University of Maryland) and a corporate network (at AT&T Labs Research). We correlated the volumes and the sources of attack collected with a global IDS and a globally distributed honeynet. We found that only 30% of attack traffic could be globally correlated. The remaining 70% were specific to the local network. This experiment showed the importance of honeypots inside organization network to provide an accurate view of network threats. We published our results in [17] and we presented our findings at the IEEE Symposium on High Assurance System Engineering 2008 (HASE '08, 22% acceptance rate).

We learned from these different experiments that honeypots provided unique insights about malicious threats. They offered an invaluable solution to understand attacks and quantify network security, without the problem faced by IDSs of filtering user traffic. However, the administrative burden required by high interaction architectures was heavily time and resource consuming. Moreover, it took us several months after the end of each experiment to analyze and extract insightful results from the large amount of data collected. Moreover, we were able to collect attacks using only few honeypot IP addresses. These conclusions are the direct consequence of the poor scalability of high interaction honeypots. With such architecture, offering a high level of interaction to attackers could not be expanded to a large panel of services and to a large IP space. For example, we know that the University of Maryland network is made of two /16s subnets. This represents a total of 131,074 allocatable IP addresses. Quantifying malicious activity on such space would require deploying honeypots on more than a handful of IP addresses.

- To precisely assess this problem of coverage (the number of honeypots that need to be deployed to collect a representative sample of the malicious activity that targets the organization), we collected during six weeks the complete scanning activity occurring on campus. We used network flow collectors deployed at the edge of the campus network to store all the network flows destined to unused addresses. We discovered that a single honeypot was collecting traffic from only 3% of the overall campus malicious activity. We also assessed that to reach a coverage of 50% of this overall activity, we needed to deploy more than 500 honeypots. The results of this analysis are presented in Chapter 3 and were

published in [15]. Our findings were presented the IEEE Symposium on High Assurance System Engineering 2008 (HASE '08, 22% acceptance rate).

These publications outline our experience in assessing the limitations of current honeypot technologies. We summarize these limitations and we detail our approach in the next two sections.

5. Problem Statement

When deploying honeypots, researchers have to precisely define three elements: a location, an architecture, and a configuration. Data collected by honeypots is critically affected by these three keys. Therefore, they need to be carefully selected. We will now detail the different problems related to each of these elements.

The **location** is the set of IP addresses used by honeypots to receive and collect network traffic. The current addressing protocol deployed on the Internet is IPv4 [60], which is made of 4.3 billions unique addresses. The volume and the nature of attacks can greatly change from one IP address to another. Some attack threats such as the Slammer worm [54] are globally distributed, while others such as Denials of Service [55] target precise locations. So the location of honeypots can greatly affect the data it will receive. Recent studies started to compare attack data from different locations [61] and defined network characteristics such as reachability or proximity to production networks [23] that could partially explain the differences observed. Moreover, not only the location but the size of the network of honeypots is important to collect significant attack results. We detail further in Chapter 3 the relation between number of honeypots and malicious activity coverage.

The honeypot **architecture** refers to the type of honeypot. We saw in the previous section that the different types of honeypots were governed by three attributes: fidelity, scalability and security. There is currently no solution available that offers both scalability and a high level of interaction [63]. As a result, researchers and network operators who want to deploy honeypots cannot collect and analyze datasets which have both detailed attack processes and large network space coverage. We introduce in Chapter 4 a hybrid honeypot architecture to precisely address this challenge.

The **configuration** defines the set of services offered to attackers and thus the behavior of the honeypot. By set of services we mean the set of opened ports and software listening for network connections on the honeypot. These services can be emulated or real. They can be host-specific resources or vulnerabilities to study specific categories of attack. The problem when deploying honeypots in a large organization network is that there is a very large number of possible configurations to choose from. There is currently no solution to determine whether the configuration of a network of honeypots is optimal to collect malicious threats; and to make sure that the fingerprint of the network of honeypots is small enough to prevent attackers from detecting it. We present in Chapters 5 and 6 a method based on network flows to precisely address this challenge.

The last major issue of current honeypots is that even if they actively reply to attackers with more or less interaction, they do not allow researchers to select the type of attack they want to study. This means that because honeypots collect attacks randomly, the information collected is not often the information researchers were really looking to analyze. From such point of view, existing honeypots are collecting attack traffic

passively. We believe that if honeypots adopt a more active approach when receiving illegitimate connections, they could 1) provide better results on the exact threat expected to be studied, and 2) reduce the resources spent to analyze and filter data collected.

6. Approach

The purpose of our study is to develop efficient solutions to overcome current honeypot limitations. We addressed the issue of the size and the location of honeynets by correlating network flows with darknet data. We solved the problem of scalability of high interaction honeypot by implementing an advanced hybrid honeypot architecture called Honeybrid. We solved the problem of configuring honeynets in large organization network by using a server and scanner discovery program based on network flows. Finally we addressed the challenge of cost effectively analyzing large volumes of malicious data by implementing an aggregation process that integrates network flows and honeypot data. These solutions are integrated into a complete framework to facilitate honeypot deployment and attack data analysis. The different software solutions of this framework are represented on Figure 2. The overall goals are 1) to provide to the security community an advanced honeypot solution that can be better integrated into the landscape of security tools used by researchers and network operators, and 2) to deploy such architecture at the University of Maryland to better quantify malicious activity occurring on the campus network. The cornerstone of this architecture is a hybrid gateway that offers both advantages of high and low interaction honeypots: fidelity and scalability.

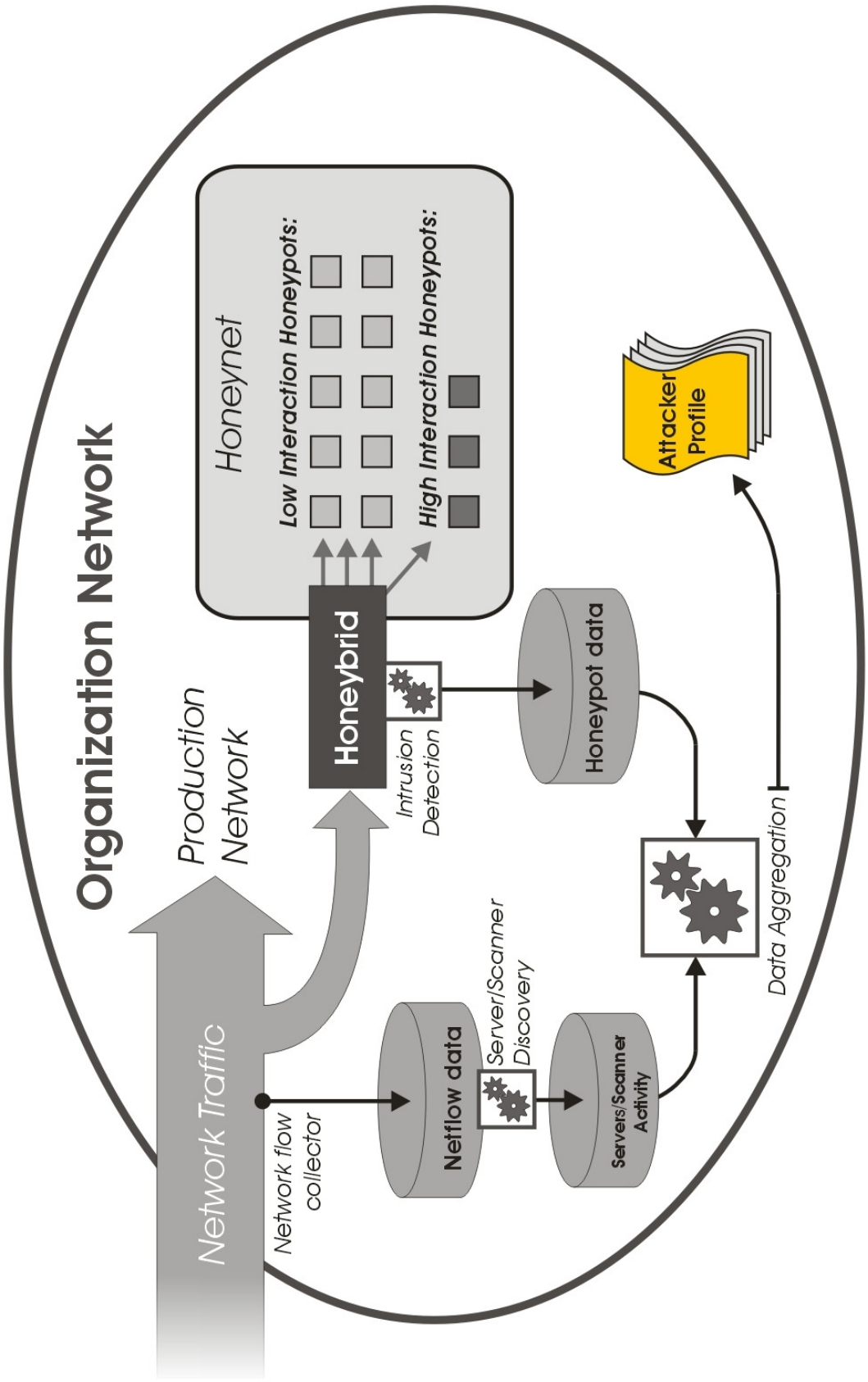

Figure 2: Overview of our malicious traffic analysis framework

7. Contributions

The contributions of this work are:

- *Honeypot classification and mapping with attack process*: we provide a detailed classification of current honeypot solutions and we link this classification with the different phases of attack collected. We outline the different properties and limitations of honeypots.

- *Hybrid architecture*: we describe an innovative honeypot solution that provides both a high scalability and a high level of interaction. We also introduce the concept of an attack event, to differentiate network attacks worth of analysis from the noise of malicious traffic. Our architecture is designed to be able to harvest large IP spaces while actively filtering attack events from attack traffic for detailed focused analysis.

- *Dynamic configuration engine*: we address the problem of honeypot configuration by combining network flows and automated honeypot management. From an exhaustive monitoring of the existing attack patterns targeting the organization network, we infer the required honeypot configuration to assess the malicious activity.

- *Architecture integration*: we provide the first open source implementation of an hybrid honeypot architecture. We integrate this architecture with network flows to provide a complete attack assessment framework. Finally, we deploy this framework at the University of Maryland and we show how it can be used to accurately detect compromised computers inside the organization network.

8. Structure

In this dissertation we present our research into developing case studies based on data collected on high interaction honeypots and designing an advanced architecture to better characterize malicious activity on large IP space such as the University of Maryland network. We organize this research as follow: in Chapter 2, we illustrate the usefulness of high interaction honeypots by presenting an experiment based on determining attacker profiles. We also present the difficulties to expand such experiment to cover larger IP spaces. In Chapter 3, we motivate further the need for an adaptive honeypot architecture by assessing the required number of honeypots to cover a significant volume of attack traffic. This study leads to Chapter 4, where we present the complete design of our innovative honeypot architecture. We first give the specifications of the hybrid honeypot technology that addresses the problem of scalability and level of interaction. We then detail our implementation, and we evaluate its functionalities on a large IP space. In Chapter 5, we introduce a heuristic-based algorithm to detect scanners and servers from network flows. We integrate this algorithm with the hybrid architecture in Chapter 6, to offer a complete network attack assessment framework for the organization network.

CHAPTER 2

HIGH INTERACTION HONEYPOTS AT WORK

1. Introduction

This chapter illustrates the depth of understanding that can be gained from high interaction honeypots. As we mentioned in Chapter 1, high interaction honeypots are well suited for collecting information on the third phase of attack processes, which is how attackers use the computer resources they successfully took over. We therefore designed an experiment with the maximum level of interaction to gather more information about this attack phase.

Our first observation of the network of the University of Maryland showed us a constant scanning activity by attackers looking for SSH servers to compromise. We also found in the literature that most security analysis experiments focus on methods for keeping attackers out of target systems, but do little to address their behavior after a remote compromise. We therefore started an experiment to understand attackers' motivation by focusing exclusively on post-compromise attacker behavior. We configured four high interaction honeypots with SSH servers running on Linux. These servers were configured with simple passwords to attract attackers. This chapter presents our findings from eight-month of data collection.

The chapter is structured as follows. Section 2 discusses the related work. Section 3 describes the experimental setup used to collect the attack data. In Section 4, we discuss the analyses on the attack data, we present the attackers' action, and the rogue software

that was downloaded, installed and run on the target computers. Finally, we summarize our findings in Section 5.

2. Related Work

The experiment described in this chapter has been published in [14]. A previous study based on a smaller data collection period of 24 days was published in [65].

In [66], the authors performed an in-depth forensic analysis of post-compromise attacker behavior. Their primary focus was on investigating the actions of more sophisticated attackers. The main difference between their project and our experiment was that we focused on a larger set of less sophisticated attackers and gathered aggregate statistics about their actions rather than investigating individual incidents in detail.

In [72], the author described the login attempts on a single honeypot over a 22-day period. A modified SSH server was used to collect password attempts, and most of the article was dedicated to the analysis of these attempts. During a period of seven days where Sebek was also installed on the honeypot, the author recorded one successful login attempt, providing some insight into attacker behavior.

The project which is the most similar to our study is [1], in which the authors collected SSH intrusions during six months from a total of 35 attackers. By comparing IP addresses of attackers with the ones collected using a large distributed low interaction Honeynet, the authors determined that intruders and scanners were two distinct sets of attackers. They also came to the same conclusion that attackers targeting weakly secured SSH servers were low skilled. Our work differs with [1] due to our larger data collection (we collected attacks from 305 distinct attackers), which allowed a more precise quantification of actions performed by attackers and rogue software downloaded.

3. Experimental Setup

To collect our data, we used a set of four high-interaction Linux honeypot computers on a sealed-off network that allowed all incoming connections, but severely limited outgoing connections to minimize damage by the attackers. The IP addresses of these honeypots were never advertised. The four honeypots all ran on an identical Linux disk image: a slimmed-down installation of Fedora Core 3. To monitor attacker activity, we used the following tools: a modified OpenSSH server to collect password attempts, syslog-ng [79] to remotely log important system events, including logins and password changes, strace [78] to record all system calls made by incoming SSH connections, and the Honeynet Project's Sebek tool [71] to secretly collect all keystrokes on incoming SSH connections. As described in [65], Each honeypot had one privileged root account plus five non-privileged user accounts. To get an idea about commonly tried usernames, we ran some initial experiments. Based on these results, we decided to use the following usernames: *admin*, *mysql*, *oracle*, *sarah*, and *louise*. These experiments also revealed that the most commonly tried passwords were '(username)', '(username)123', 'password', and '123456', where (username) represents the username being tried. We rotated among these four passwords for each username as follows: after a compromise, we re-deployed the honeypot and moved on to the next password in the list. To ensure quick turnaround after a compromise, we used a pre-built disk image and automated scripts to manage the deployment of the honeypots. We monitored the syslog messages coming from each honeypot at least every 24 hours to check for logins and password changes. In this context, we defined a compromise as an unauthorized login followed by a password change, rather than using the traditional definition of an unauthorized login only. Password changes typically happened every day. Following a password change, we

waited at least one hour before we copied the disk image back onto the honeypot, re-ran the deployment script, and continued monitoring the live syslog data.

In order to encourage attackers to enter the non-privileged user accounts instead of the root account, two of the honeypots were set up with strong root passwords. The other two honeypots had root accounts which rotated among the four passwords 'root', 'root123', 'password', and '123456'.

4. Analyzing the Attacks

The first step of our analysis was to divide the information we collected into different attacks and sessions. We assumed that each IP address over a time window of 24 hours corresponded to an attacker. We defined an attack to be all interactions between an attacker and a honeypot. We then defined a session as a single SSH interaction between an attacker and a honeypot. We extracted a total of 1,171 different sessions. Differentiating attacks from sessions is important because we discovered that some attackers used several SSH sessions in parallel to perform their attack. Our results showed that 47% of attackers used a single session, and the remaining 53% of attackers used at least two parallel sessions. Table 1 provides the number of sessions associated to the four honeypots. After several months of collecting data, we changed the set of honeypot IP addresses (March 28 - August 18, 2007 we used one set of IP addresses and August 19 - December 4, 2007 we used a second set). The change was a request from the network administrator on which our testbed was located.

Table 1: Number of sessions collected by each honeypot

Hosts	No. of Sessions (First set of IPs)	No. of Sessions (Second set of IPs)
Honeypot A	91	231
Honeypot B	155	262
Honeypot C	122	127
Honeypot D	101	81

4.1. Analyzing Attackers' Actions

In this section, we refined the analysis on the different sessions by introducing the concept of actions. An action is defined as a set of commands run by an attacker to reach a goal, for example, gathering information or installing software. Table 2 provides the detail of each action with examples of commands and overall statistics. We can see from Table 2 that typical attack sessions consist of three steps:

- First, attackers check the system configuration, by reviewing network settings, user accounts, processes running and software installed;
- Second, attackers change the system configuration by adding user accounts, modifying passwords and altering software settings; and
- Third, attackers download, unpack, install and run rogue software.

From Table 2, typical actions appear to include getting information related to users (present in 66% of the sessions), getting information related to the system (39%), and getting information related to other parts (38%). Passwords were changed in 37% of the sessions and system files were modified in 34%. We observed that files were downloaded and unpacked in more than 41% of the sessions. A total of 2,437 commands (12% of the 20,335 commands) could not be identified. Most of these unidentified commands were discovered to be typos that did not correspond to actual commands.

Besides these typical sessions, some attackers took other interesting actions such as trying to hide their intrusion. In 26% of the sessions, attackers deleted log entries and the command history. A few attackers recreated deleted files to make sure their intrusion would remain undetected. Another practice to hide intrusions was to obscure the name of the folder in which the rogue software was installed. The most popular folder names were: " ", "...", ".. ", ". " and "..". Finding one of these folders is an unmistakable sign that an attack occurred. We also found that some attackers were compulsively using the Unix command "w" or "who" to make sure no other legitimate user could connect while they were attacking the system.

Most of the attack sessions were short and lasted less than one minute. Some lasted a few hours because attackers would sometimes launch rogue software such as a network sniffer and would return to check the output. The distribution of session durations is provided in Figure 3.

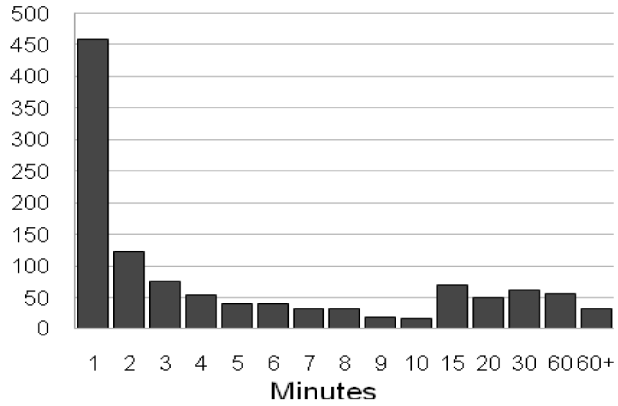

Figure 3: Number of attack sessions by duration

Table 2: Grouping of attacker's actions and statistics on the number of commands

Group	Action	Commands	Commands		Sessions	
			No.	%	No.	%
Get information	get information related to users	`w, whoami, who, last, id, finger, lastlog`	1,464	7.20%	768	65.64%
	get information related to the system	`uptime, uname, if-config, netstat, locate, php -v, hostname, whereis, nmap, cat /etc/<system file>`	894	4.40%	454	38.80%
	get other type of information	`ps -a, ps -ax, history, cat <file>`	876	4.31%	445	38.03%
Change configuration	add one or multiple users and set passwords	`adduser, useradd, passwd`	116	0.57%	48	4.10%
	gain root privileges	`su, sudo`	85	0.42%	40	3.42%
	change the password	`passwd`	569	2.80%	432	36.92%
	modify system files	`cp, mkdir, mv, rm`	1,030	5.07%	400	34.19%
	change system configuration	`path, userdel, iptables, ln, export, chown, chmod, rhsnd`	610	3.00%	236	20.00%
Edit files	read system files	`nano, pico, vi <lastlog, wtmp, bash_history, /etc/passwd>`	139	0.68%	13	1.11%
	read other files	`nano, pico, vi`	39	0.19%	86	7.35%
	edit system files	`nano, pico, vi`	430	2.11%	63	5.38%
Hide intrusion	tamper with system files or user variables to hide traces of the intrusion	`unset <ENV_VAR>, rm [-rf] <file>, export <ENV_VAR>=/dev/null, cat /dev/null > <file>`	859	4.22%	313	26.75%
Restore deleted files	restore deleted files to hide traces	`touch`	95	0.47%	27	2.31%
Kill process	terminate the execution of processes	`kill, cat <file-name>.pid, ps`	362	1.78%	109	9.32%
Fetch rogue software	download a file from a remote host and unpack it	`lwp-download, scp, curl, ftp, wget, unzip, tar`	2,339	11.50%	484	41.37%
Deploy rogue software	run a rogue software	`perl, ./<command>`	1,065	5.24%	459	39.23%
	install a rogue software	`./configure, make, make install, ./setup, gcc`	25	0.12%	11	0.94%
Tool cmd	Unix tool commands	`cd, ls, pwd, clear`	5,700	28.03%	892	76.24%
Other actions	use SSH	`ssh, "yes"`	47	0.23%	23	1.97%
	launch a new console	`sh, bash`	79	0.39%	69	5.90%
	launch a new console	`screen`	68	0.33%	30	2.56%
	failed attempt of getting files using SCP	`winscp unsuccessful attempt`	611	3.00%	8	0.68%
	chat with other users	`wall`	56	0.28%	6	0.51%
Exit	exit the session	`exit`	340	1.67%	291	24.85%
Total		**Identified commands**	**17,898**	**88.02%**	**1171**	
		Unidentified commands	**2,437**	**11.98%**		
		Total commands	**20,335**	**100.00%**		

4.2. Analyzing Rogue Software

The honeypots were instrumented to record all network traffic. We analyzed the incoming traffic using tcpdump [80] and chaosreader [20] to extract files downloaded on the honeypots. We extracted 250 files from 379 attack sessions where attackers downloaded files. We could not extract files in the other 129 sessions because the attacker file download had failed mainly due to incorrect URLs or missing files. This result let us believe that attackers were neither skilled nor organized. Here is an example of an attack session where the attackers did not succeed downloading a remote file. We found that the attacker tried to download several files, was unsuccessful and then left.

```
9:53:31 w
9:53:33 uname    -a
9:53:34 uptime
9:53:45 cat /cpuproc/cpuinfo
9:53:48 cd /tmp
9:53:49 ls -al
9:53:53 mkcd #
9:53:54 ls -al
9:53:58 mkdir   " "
9:54:00 cd  " "
9:54:03 ls wget
9:54:07 /sbin/ifconfig
9:54:31 wget
9:55:09 ftp
9:55:47 o
9:55:54 ftplynx www.almerimur.com/capella/linux.tar.gz
9:56:08 wget www.geocities.com/capella99_2000/linux.tar.gz
9:56:18 ftp 207.150.179.22
9:57:18 wget  www.almerimur.com/capella/linux.tar.gz
9:57:26 history -c -d offset
```

To analyze the nature of the files we extracted, we submitted them to the VirusTotal web service [84]. 50% of the files could be identified using VirusTotal. The remaining 50% were identified manually using their source code. Figure 4 provides the categories and the volume of files downloaded by attackers during the data collection period. Figure 4 indicates that the popularity of our honeypots grew over time. The gap in August 2007 is due to the change in IP addresses. Figure 4 also shows that the main interest of

attackers was to install "**IRCBots**" (most were Mech-based IRCBots [51]). The motivation to deploy IRCBots is to make the compromised computer part of a botnet. An army of several thousand bots can be turned into profit by attackers who sell computer resources on the black market [35]. Here is an example of IRCbot based attack session:

```
13:46:06 cd /var/tmp
13:46:11 wget http://www.shaq.profesor.info/like/error.tar.gz
13:46:16 tar xzvf error.tar.gz
13:46:18 rm –rf error.tar.gz
13:46:19 cd error
13:46:21 chmod +x*
13:46:22 ./x
13:46:28 exit
```

The second most popular software installed was "**Bouncers**", which are programs used to relay network connections, much like a proxy. Attackers often use this type of software to hide their source IP address and hostname. Most of the bouncers we collected were based on Psybnc [64]. Under "**Attack Tools**", we grouped various programs used by attackers to compromise the computer. These tools included non-malicious software, such as rogue SSH servers, and malicious programs, such as john-the-ripper [44], log cleaners, process hiders and network sniffers. In this category, we included rogue web servers installed by attackers to setup phishing websites [31]. The "**Rootkits**" type included system exploits and rogue binaries used to gain root privileges on the compromised computer. "**Network Scanner**" contained software to automatically probe for listening SSH servers or to perform port scans. The "**Flooder**" type consisted of network applications built to launch denial-of-service attacks [55] against a given target. "**Backdoor**" included programs to stealthily and remotely control the compromise computer. "**Files**" contained non-malicious files, such as movie trailers, computer drivers or even Windows update patches. Attackers who downloaded these files were simply

using the compromised computer for storage. We even found an attacker who attempted to turn the compromised computer into a CounterStrike game server.

The final step in our analysis was to label each session with the type of file downloaded. We discovered that in 58 out of the 379 sessions, attackers downloaded more than one type of file. This number increases to 158 if we aggregate files per attack instead of session. This is because attackers often used auxiliary sessions to download other files or performed other tasks in parallel. Main and auxiliary sessions were linked using the source IP and a time window of 24 hours.

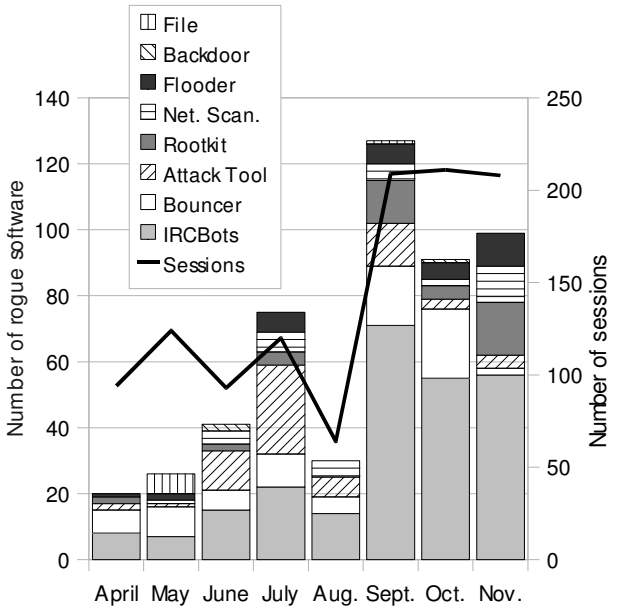

Figure 4: Number of files downloaded by attackers over time, sorted by type

Table 3 indicates that for all categories of rogue software, approximately half of the attackers used auxiliary sessions. These auxiliary sessions had, on average, fewer command lines compared to main sessions and tended to be shorter, except for the "**Flooder**" and "**File**" types. We investigated the delay between the first command typed by attackers and the time when they deployed their rogue software, in order to get insight

on the detection time required by a security tool before damage occurred. On average, for main sessions, it took between 2 minutes 14 seconds and 30 minutes for attackers to deploy their rogue software.

Table 3: Comparison between attack sequences based on the type of rogue software

Main rogue software:	**IRCBots**		**Bouncer**		**Attack Tool**		**Rootkit**		**Net. Scanner**		**Flooder**		**Backdoor**		**File**		**None**	
Other rogue software often associated:	Bouncer (12) Rootkit (11) Flooder (11) Net. Scan. (9)		IRCBots (5) Att. Tool (6) Rootkit (4) Backdoor (2)		IRCBots (7) File (4) Bouncer (4) Net. Scan. (4) Rootkit (1)		IRCBots (4) File (4) Net. Scan. (3) Att. Tool (3) File (2)		Rootkit (2) Att. Tool (2) IRCBots (1) Flooder (1)		Rootkit (3) Att. Tool (2) File (1)		Rootkit (1) Att. Tool (1)		File (6)			
Number of attackers:	**83**		**29**		**24**		**22**		**17**		**18**		**4**		**4**		**182**	
Attackers using aux. sessions:	*53*		*15*		*11*		*9*		*9*		*8*		*0*		*1*		*89*	
Type of session:	**main**	*aux.*	**main**	*aux.*	**main**	*aux.*	**main**	*aux.*	**main**	*aux.*	**main**	*aux.*	**main**	*aux.*	**main**	*aux.*	**main**	*aux.*
Number of sessions:	**144**	*136*	**40**	*48*	**29**	*45*	**24**	*24*	**22**	*17*	**21**	*17*	**7**		**6**	*3*	**262**	*326*
Avg. session duration (s):	**685**	*253*	**761**	*306*	**980**	*252*	**936**	*541*	**1,094**	*192*	**680**	*1,700*	**563**		**742**	*755*	**543**	*854*
Min. session duration (s):	**4**	*0*	**33**	*0*	**88**	*0*	**60**	*0*	**45**	*14*	**19**	*0*	**155**		**366**	*0*	**0**	*0*
Max session duration (s):	**11,470**	*3,339*	**3,645**	*2,611*	**2,750**	*2,664*	**4,183**	*4,582*	**4,512**	*998*	**2,819**	*19,672*	**1,227**		**1,976**	*2,264*	**7,647**	*76,629*
Avg. number of lines:	**28**	*17*	**31**	*15*	**212**	*30*	**41**	*18*	**36**	*10*	**21**	*16*	**28**		**18**	*4*	**15**	*13*
Min. number of lines:	**2**	*1*	**3**	*1*	**15**	*1*	**9**	*1*	**4**	*2*	**3**	*1*	**12**		**6**	*1*	**1**	*1*
Max. number of lines:	**676**	*702*	**93**	*88*	**914**	*859*	**116**	*95*	**261**	*26*	**98**	*59*	**47**		**39**	*9*	**217**	*237*
Avg. delay before exploit. (s):	**1,030**	*204*	**939**	*371*	**715**	*94*	**1,891**	*987*	**2,105**	*74*	**1,119**	*21*	**358**		**96**	*0*	**581**	*432*
Percentages of attackers for each group of actions:																		
Get info.	**87%**	*94%*	**90%**	*100%*	**96%**	*91%*	**91%**	*89%*	**94%**	*100%*	**89%**	*88%*	**100%**		**100%**	*100%*	**80%**	*87%*
Edit files	**17%**	*11%*	**31%**	*33%*	**33%**	*9%*	**18%**	*11%*	**29%**		**17%**	*25%*			**25%**		**18%**	*27%*
Change conf.	**80%**	*87%*	**83%**	*80%*	**88%**	*82%*	**96%**	*67%*	**59%**	*67%*	**72%**	*75%*	**100%**		**75%**	*100%*	**62%**	*81%*
Fetch rogue software	**100%**	*55%*	**100%**	*53%*	**100%**	*82%*	**100%**	*78%*	**100%**	*67%*	**100%**	*38%*	**100%**		**100%**	*100%*	**50%**	*60%*
Deploy rogue software	**76%**	*40%*	**69%**	*33%*	**83%**	*36%*	**82%**	*44%*	**77%**	*44%*	**67%**	*25%*	**25%**		**25%**		**41%**	*51%*
Kill process	**16%**	*21%*	**24%**	*7%*	**17%**	*9%*	**14%**	*11%*	**18%**		**11%**	*25%*					**7%**	*23%*
Hide intrusion	**30%**	*34%*	**45%**	*40%*	**38%**	*46%*	**36%**	*11%*	**29%**	*11%*	**28%**	*38%*	**25%**		**25%**		**26%**	*30%*
Restore deleted files	**1%**		**7%**	*7%*	**4%**		**9%**	*11%*				*13%*	**25%**				**5%**	*5%*
Other actions	**17%**	*13%*	**28%**	*27%*	**33%**	*46%*	**14%**	*33%*	**29%**	*11%*	**6%**		**25%**				**12%**	*20%*
Tool commands	**98%**	*72%*	**97%**	*87%*	**100%**	*82%*	**100%**	*67%*	**94%**	*67%*	**94%**	*75%*	**100%**		**100%**	*100%*	**72%**	*84.00%*

This delay is on average shorter for auxiliary sessions, and can even reach 0 seconds for the "**Backdoor**" and "**File**" types, because auxiliary sessions were sometimes used

only to deploy the rogue software previously downloaded in the main session. This explanation is confirmed if we consider the breakdown of attacker actions per type of session and category of rogue software. We can see that all attackers used a main session to fetch rogue software, while using auxiliary sessions to obtain information about the system or to change program settings. We can also see that for all categories except for "**Rootkit**" and "**Network Scanner**", attackers primarily used auxiliary sessions to hide their intrusion and restore deleted files.

4.3. Discussion

The data we analyzed provided evidence that attackers targeting weakly secured SSH servers tend to be low skilled humans. The large number of typos in recorded commands and the timing between commands indicate that attacks are rarely from automated scripts, but from human beings who used interactive terminals. For scalability reasons, automated attacks are predominantly on the Internet. However, our dataset shows that this rule does not apply for the specific service we opened on our honeypots (SSH on port tcp/22). We believe that human attackers use automated scripts to scan and find SSH servers to compromise. Once successfully logged onto the machine they proceed to manually install rogue software.

Further evidence of the low skill level is found from the relatively low percentage of attackers who attempted to hide their intrusions as well as the large volume of attackers who were not able to complete their attacks. We found a number of attackers did not complete their attacks because some system tools were missing on the honeypot, or because the URL from which they tried to download rogue software was invalid. These findings confirm the conclusions of [1] and [66].

5. Summary

We found in this experiment that a typical attack session consisted of: 1) checking the system configuration, 2) changing the system configuration, and 3) downloading, installing and running rogue software. In about 25% of the cases, attackers will try to hide their intrusion.

We identified 250 rogue software files of various types. The most popular were IRC bots, bouncers, attack tools, root kits, network scanners, flooders and back door programs. We also found that attackers often launched more than one attack session at a time. We compared the main session with the auxiliary ones. We also found that in 27% of the sessions, attackers did download some software which was never used. This is an indication that we might have not given attackers enough time before redeploying the honeypot.

We also learned from this experiment that high interaction honeypots required important computer and human resources. Our hardware configuration consisted of six machines to collect data from only four IP addresses. We spent several months to build our installation and then to analyze the data collected. Moreover, the honeypots had to be closely monitored on a daily basis. To increase the scope of our study and to collect a larger variety of attacks from a greater number of IP addresses would have been very challenging. In order to assess the required size of a honeynet to collect attacks from a majority of attackers targeting the University of Maryland, we started correlating large scale attack information from network flows with small scale attack information from honeypot data. The design of our correlation process and the related results are detailed in the next chapter.

CHAPTER 3

DARKNET SCALE IN THE ORGANIZATION NETWORK

1. Introduction

We presented in Chapter 2 an experiment based on high interaction honeypots. The substantial findings of this experiment demonstrate the advantage of honeypots to better understand malicious threats. We then wanted to know whether these findings were collected from a significant proportion of attackers targeting the organization network of the University of Maryland. To precisely understand the relationship between the size of a honeynet and the volume of attackers captured in a given organization network such as the University of Maryland's network, we have to analyze malicious activity for the entire organization. By providing an exhaustive view of all communications between internal and external hosts, network flows can precisely address this challenge. The network flows collected at the University of Maryland are obtained with Cisco's Netflow [58]. Our methodology consists in correlating the data collected for a given set of unused IP addresses, or darknet, with the scanning activity detected in network flows from collectors deployed at the edge of the University of Maryland's network. This correlation was performed in two ways. First, by comparing the source IP addresses of attackers hitting the darknet with the list of attackers scanning the network, we were able to assess the nature of data collected by the darknet. Second, by comparing the list of destination IP addresses scanned with a simulated set of unused IP addresses, we were able to assess

the darknet size needed to reach a significant coverage of the overall scanning activity that occurred at the University of Maryland's network.

The results of this experiment address the problem of location and size of honeynets deployed within organization networks. Figure 5 indicates which components of our framework are involved in this chapter.

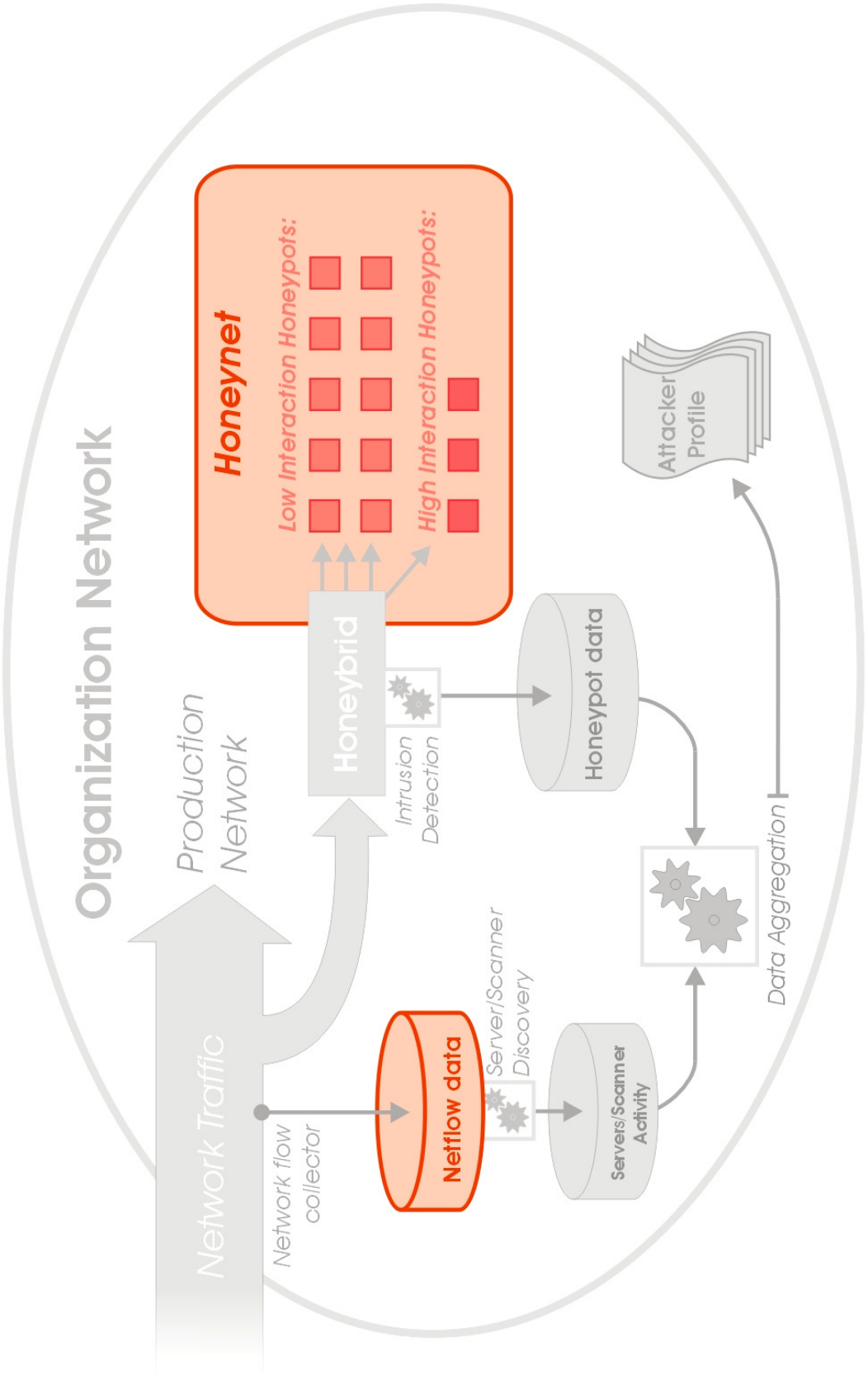

Figure 5: Overview of the components of our framework involved in this Chapter

This chapter is organized as follows. The related publications are reviewed in Section 2. In Section 3, we present an investigation of the different types of activity received by unused IP addresses. In Section 4, we provide statistics about the scanning activity at the University of Maryland's network level, which leads to Section 5 where we explain how we could use this scanning activity to improve darknet coverage. Finally, in Section 6, we identify future work and summarize our findings.

2. Related Work

This study is based on two areas of research: darknets and port scan detection. We review the related publications for each of these areas in the next two sub-sections.

2.1. Darknets

The idea of monitoring unused address space to detect malicious activity has led to several research projects including [29], [75], [56] and [3]. Bailey et al. provided practical solutions on darknet measurement in [4] and analyzed further in [24] the impact of darknet location by comparing traffic collected using 10 distributed darknets that ranged in size from /25 to /8 networks. They presented evidence that very different traffic activities were observed among distributed address blocks. They also built a list of sensor properties, such as reachability, visibility, and local scanning preferences, to explain the differences and to better understand the implication of sensor location. They also introduced in [6] a source-based filtering algorithm to reduce datasets collected by darknets. Our work differs from their study because we focused on assessing the size of a single darknet deployed within a given organization's network by comparing data from both the darknet and the production network. We believe that tracking attackers who

visited dark IP addresses within the production network can greatly help understanding the different attack strategies.

Sensor size was also analyzed by [55] but from the perspective of detection time, whereas in this study, we were motivated by the problem of the volume of malicious activity covered.

2.2. Port scan detection

We defined a port scan as the activity of an attacker that probes a set of IP addresses at a site looking for vulnerable servers [45]. Algorithms to detect such patterns fall into three categories:

- The simplest solution is to identify attackers generating more than N network events within a given time window T [39]. This method is currently used by the Snort [70] IDS. The drawback of this solution is to rely on parameters that have a great impact on performance [69].
- To reduce this problem, a probabilistic distribution model was suggested by [47]. The idea is to rank remote source IP addresses as normal or attacker by computing for each of their local destination addresses a measure of unusual access. This metric is calculated using an access probability distribution based on access history.
- To address the problem of fast detection speed, sequential hypothesis testing was introduced by [45] with an algorithm called Threshold Random Walk (TRW). This algorithm computes a series of updates of a likelihood variable based on the assumption that attackers will generate many more failed connections than normal hosts.

In this study we did not have the constraint of detection time and we had only access to a dataset of failed TCP connections. We also had the motivation of focusing solely on scanners and thus limit the impact of false positive. Therefore we decided to use the simple threshold methodology with a conservative value of more than 30 local source addresses targeted in less than 24 hours. We justify these thresholds in Section 4. We leave the possibility to implement a more complex algorithm for a future extension of this work.

3. Assessing Darknet Activity

In this section, we first define and then quantify the different types of traffic received by unused IP addresses on our organization's network. We limit the scope of this study to external TCP traffic, where external means traffic coming from network addresses that do not belong to the organization's network, because we will correlate darknet traffic with Netflow information collected at the edge of the organization's network. We limit the study to TCP traffic because we will use the protocol flag information to more precisely filter the different types of traffic. We will provide the volumes of UDP and ICMP traffic collected to show that TCP traffic is a significant amount of the overall traffic received. We define an attacker, or source, as a tuple {day; source address; destination port; protocol}. This means that a single source IP address sending TCP packets at two different days to two different ports each day will be seen as four distinct sources. The decision to include the destination port in our definition of a source was made to better identify attacks that are linked to a specific network service.

3.1. Defining categories of TCP traffic received by unused IP addresses

By definition, unused IP addresses do not receive user traffic. So the remaining types of traffic they are susceptible to receive include:

1. Misconfiguration traffic: when a source is trying to establish a connection with an incorrectly configured destination, which happens to be an unused IP address;
2. Backscatter traffic: when the victim of a DoS attack is trying to reply to a spoofed network address and;
3. Malicious traffic: when an attacker is trying to compromise a network resource.

We can further divide this last category into:

- Random attacks: when an attacker uses scanning techniques to discover victims and;
- Targeted attacks: when an attacker has knowledge of the location of a specific network resource to compromise. The victim is therefore a target of choice, and the attacker does not usually need to scan multiple network addresses to find it.

From a source point of view, these categories have specific characteristics that could help us to differentiate them. If we take the characteristic "number of destination addresses", then scans can be characterized by a single source address sending packets to a large number of destination addresses. On the contrary, traffic due to misconfiguration or directed attacks will often be characterized by a single source address sending packets to few destination addresses (the ones that are incorrectly configured or the ones that are specifically targeted). So, by correctly defining what low and high numbers of destination

addresses represent, we can use this characteristic to differentiate between scans and misconfiguration traffic or directed attacks.

In the context of the TCP protocol, another characteristic we can use are the TCP flags. For example, usual backscatter from TCP DoS attacks are characterized by packets with the flags SYN and ACK enabled (because the victim of the DoS attack is trying to acknowledge the TCP handshake initiated by the spoofed addresses). TCP scans, misconfigurations and directed attacks are trying to initialize TCP connections. Therefore, these categories are characterized by first packets that have only the flag SYN enabled. Finally, unused IP addresses can be targeted by other types of TCP scans besides SYN-scans. This category will therefore be characterized by first packets that have other combinations of TCP flags than a single SYN.

According to these characteristics, we can differentiate three groups using only two characteristics: the number of destination addresses and the flags. To distinguish between low and high number of destination sources, we used a threshold N and a time window of one day. If more than N destination addresses are contacted by a single source during a day, then we characterize this source as having a high number of destination addresses. In Section 4 we will empirically assess the value of N. Table 4 summarizes the filters we developed to separate the three groups according to these two characteristics.

Table 4: Groups of traffic and related filters

Groups of traffic	Filters
- Misconfiguration - Directed attacks	SYN flag and less than N destination addresses per day
- SYN-scan	SYN flag and more than N destination addresses per day
- Other scanning techniques - Backscatter	Other flag combinations than a single SYN

3.2. Quantifying traffic received by unused IP addresses

Our methodology to quantify the categories of external TCP traffic collected by unused IP addresses was to correlate Netflow information collected at the edge of the University of Maryland's network with darknet traffic. This correlation provides the required information on the total number of destination addresses per source and per day, to separate sources with low and high numbers of destination addresses.

The empirical results are based on a darknet of 77 network addresses, deployed in one of two /16 university networks. The 77 monitored addresses are spread over a single /23 subnet where other live hosts are in use.

Netflow traffic was collected at the edge of the university's network using the Nfsen/Nfdump architecture [38]. The university's network is protected by a firewall and the traffic rejected is not part of our data collection. Therefore, only the traffic that went through the firewall was recorded. We used a filter to store all the traffic towards the darknet on a daily basis. We wrote a script to extract, on a daily basis, all the university's network activity coming from the sources identified in the traffic towards the darknet. Finally, we correlated the Netflow and darknet datasets to classify each source sending packets to the darknet in the categories defined in Table 4. We collected data during a period of six weeks, from September 21 to November 4 2007. During the time of the experiment, we recorded a total of 18,778 external sources. 36% of these sources sent TCP packets, 45% sent UDP packets and 19% sent ICMP packets. After running the filters on the TCP data collection with a value of N=30 for the threshold between low and high number of destination addresses (we justify the threshold value of N=30 in the next section), 66% of sources were classified in the group of SYN-scans, 21% in the group of

backscatter and other scanning techniques, 13% in the group of misconfiguration and directed attacks. Figure 6 shows the evolution of the percentage of distinct sources per day and per group.

We can see from Figure 6 that the category "SYN-scans" prevails. The purpose of using darknets is to monitor malicious behavior, and potentially to understand attack processes using interactive sensors: how attackers are trying to compromise machines and for what purpose. The problem is that the current darknet provides only a sample of the scanning activity that occurs over the entire organization's network. Consequently, the next phase of our study is to know how significant this sample is. By understanding the actual coverage of our darknet of 77 network addresses among the two /16 networks in our organization, we can better decide if we need to monitor additional unused IP addresses to acquire more statistically significant results.

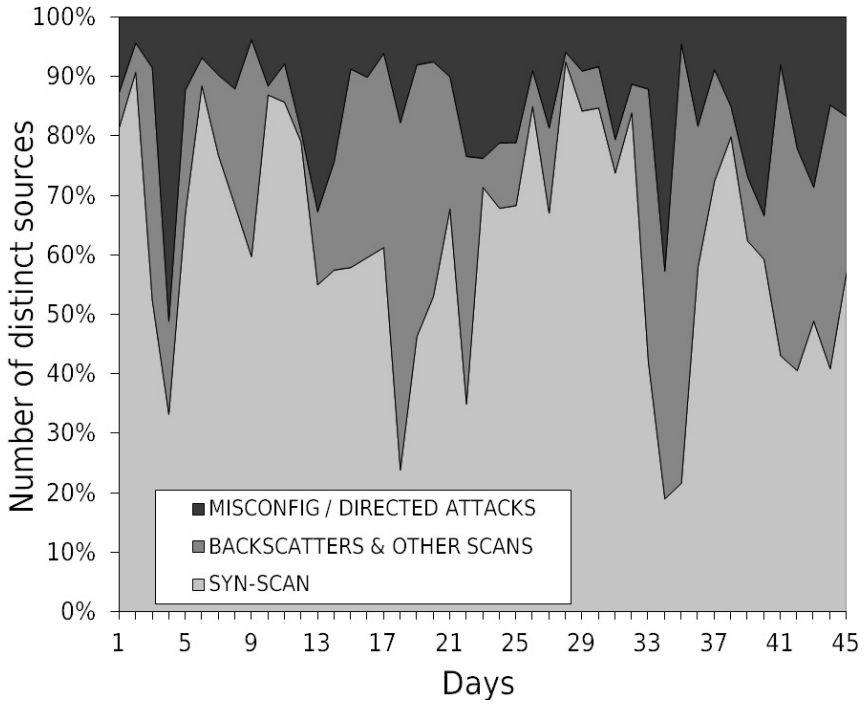

Figure 6: Darknet external TCP activity per group (Normalized)

This question is addressed in the two following sections: in Section 4 we measured the scanning activity at the organization level, and in Section 5 we correlated the destination addresses of these scans with various simulated darknets.

4. Measuring University Scanning Activity

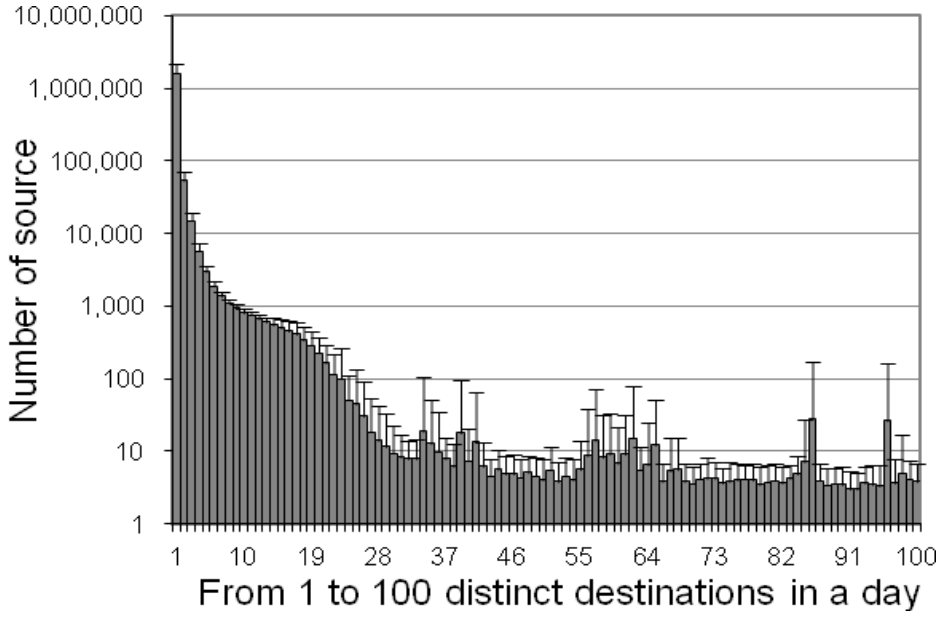

Figure 7: Daily number of sources per number of distinct destination addresses (average on 45 days)

In Section 3, we found that more than 66% of the external TCP sources collected by our set of 77 unused addresses were performing SYN-scans. In this section, we measure the overall SYN-scan activity at the organization's level. Our data collection is based on the assumption that an attacker scanning the organization network using the TCP protocol will most likely generate failed connections. We are confident in this assumption because we know that the ratio between the number of network devices that could successfully reply to a given probe, and the ones that could not due to filtering or because the network address is not used, is close to zero. For example, for port 22 we detected that 870 destination addresses in the university's network would accept incoming connections

from an external source. Compared to the 131,072 addresses available in the two /16 networks in our organization, the probability that a scan hits an open port 22 is below 0.7%. Therefore, we are using noise of failed TCP connections to identify sources scanning the network. Flows generated from failed TCP connections are easily characterized because they only have the SYN flag enabled. Consequently, our methodology to record the scanning activity consists in collecting daily all ingress flows that only have the SYN flag enabled. Then, we aggregate these flows per source to find the number of destination addresses each source was trying to reach. As mentioned, we define a source as a tuple {day; source address; destination port; protocol}. Finally, we separate sources into two categories: user traffic and malicious traffic. We assume that the difference between these two categories is based on the number of destination addresses attempted in a day. Our dataset only consists of failed initializations of TCP connections. This means that the destination addresses did not send any packets back to the source. On large networks, failed connections can occur for various reasons:

- At the network level: packets can be dropped because of a network outage or network policy, such as failures in the network infrastructure, packet shaping or packet filtering and;
- At the host level: the connection can fail because the destination address is not allocated, or the host is shut down or the service is not hosted and/or filtered.

Table 5: Daily number of sources for a range of destination addresses (average over 45 days)

Ranges of destination addresses	Average number of sources per day	Categories
1	1,631,703	Legitimate failed connections
2-16	86,768	
17-30	1,818	
31-100	411	Scans
101-1,000	362	
1,001-10,000	53	
10,001-100,000	43	
100,001-131,000	2	

With a time window of one day, we assume that failed connections from a single source is not characterized by a large number of distinct destination addresses, except if the source performed a scan. So we used a threshold of more than N destination addresses per day to separate sources generating legitimate failed connections from sources scanning the network. Figure 7 provides the distribution of the daily average number of sources that unsuccessfully tried to contact between 1 and 100 distinct destination addresses. The error bars in Figure 7 indicate the standard deviation, and the daily average of 460 sources that contacted more than 100 destination addresses on campus are not shown due to space. The logarithmic scale of the number of sources in Figure 7 highlights two different modes in the distribution. From 1 to 30 destination addresses, we see an important volume of sources, which quickly decreases from more than one million to ten. After 30 destinations addresses, the number of sources steadily decreases, except for a few spikes, which are likely to reveal specific scanning software that use similar algorithms to find victims on the organization network. To select a threshold between legitimate failed connections and scans, we collected all failed connections generated by two computers under standard usage, with various network applications including P2P

software during 17 days. The average number of distinct destinations from failed TCP connections per computer and per day was 16 with a standard deviation of 13. Based on this information and the distribution given by Figure 7, we selected a value of N=30 for the threshold separating legitimate failed connections and scans. There might be external sources scanning less than 30 destinations in our organization's network, however we believe that a threshold of 30 is a conservative value to use to filter out most of the legitimate failed connections from the scanning activity. Table 5 gives the average number of sources for different ranges of destination addresses.

5. Darknet Coverage

The previous section focused on the distribution of sources that launched TCP scans against the organization's network. In this section, we investigate the distribution of the destination addresses scanned. The goal is to discover if these destination addresses are uniformly distributed, or if they are concentrated in specific regions within the organization's network. This information will help us understand where the darknet should be located to cover a large malicious activity targeting the organization's network.

We first quantified the malicious activity coverage of the current darknet of 77 addresses. Then, we analyzed the distributions of destination addresses scanned to find the best deployment strategy. It is important to note that we are working with a dataset of failed TCP connections. This means that from all destination addresses scanned by external sources, we did not include in our statistical results the destination addresses that replied to the attacker. We do not think that this alters our findings because, as we show with an empirical example in Section 4, the number of unused addresses is much higher than the number of live hosts. Therefore, the ratio between the number of destination

addresses replying to SYN-scans and the total number of destination addresses probed is likely to be close to zero.

5.1. Current darknet coverage

As mentioned in Section 3, we monitored 77 unused addresses on a single /23 subnet where other production machines are used. Our organization's network is made up of two /16 networks that we will call Network A and Network B. The darknet is currently deployed on Network B. During the six-week experiment, 66% of the 6,693 distinct TCP sources collected by the darknet also scanned the organization's network. These 4,398 sources represent an average of 14% of the overall dataset of sources scanning the organization's network during the same period of time. This percentage increases to 26% if we simply consider sources that scanned Network B. Figure 8 provides the daily evolution of the number of sources collected by Network A, Network B and the darknet over the six-week period. We can see from this evolution that the daily number of external sources scanning the network greatly fluctuated, from 200 sources to more than 2,000 in only a few days. However, the darknet activity always shows a strong correlation with the activity occurring in Network B, which is confirmed by a Spearman correlation coefficient of 0.91 between these two datasets.

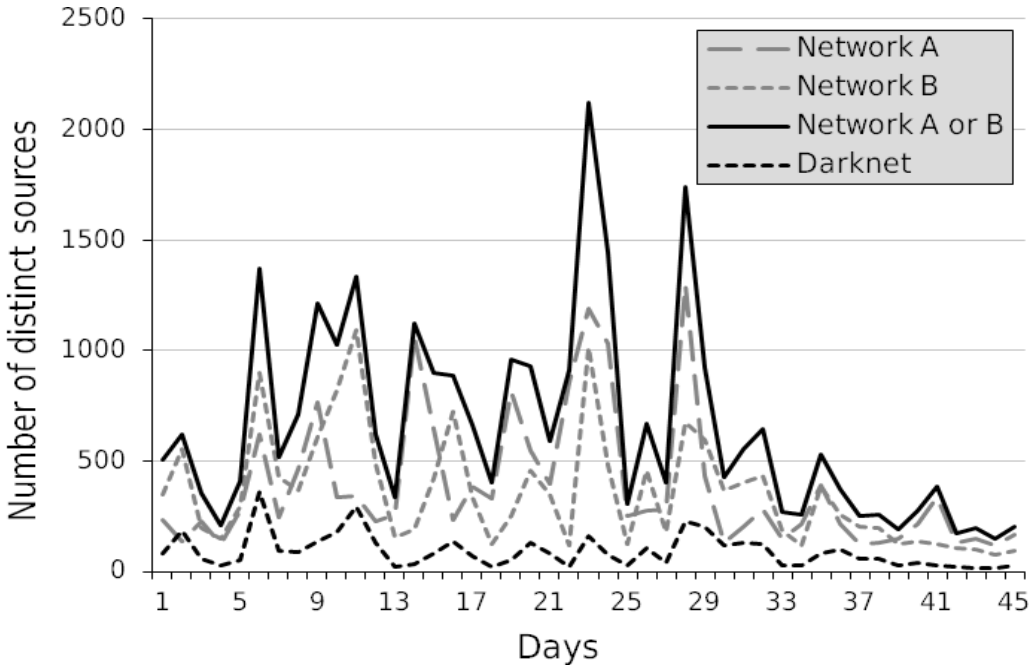

Figure 8: Daily evolution of the number of sources scanning the campus networks

The conclusion of this analysis is that our darknet, on average, does not collect information from 84% of the sources scanning the organization's network. In the remaining part of this section, we will determine the required darknet size and location to increase this coverage.

5.2. Distribution of destination addresses scanned

Our methodology to increase the darknet coverage consists of understanding the different strategies used by attackers to scan the organization's network. For this purpose, we applied on each source an entropy-based metric that detects clusters or uniform distributions in a two-dimensional space [37]. Our dataset can be seen as a two-dimensional space because the first two bytes of all destination IP addresses belong to the only two prefixes that uniquely characterize the two /16 networks: Network A and

Network B. Therefore, if we focus on one of these two networks, only the last two bytes will change.

The metric computes two scores for each source: the third byte entropy and the fourth byte entropy. This entropy is defined as the weighted sum of densities of destination addresses per third or fourth byte ranges. For example, for the third byte in Network A, we define $X_A(i)$ as the number of destination addresses in Network A that are scanned by the external source i. We define $X_{A[j]}(i)$ as the number of destination addresses scanned in Network A by the source i that have a third byte equal to j (this means that all destination addresses counted in $X_{A[j]}(i)$ belong to the same /24 subnet). Then, the entropy of the external source i for the third byte in Network A is defined by equation (1):

$$H_A(i) = \frac{-\sum_{j=0}^{255} \frac{X_{A[j]}(i)}{X_A(i)} \cdot \log\left(\frac{X_{A[j]}(i)}{X_A(i)}\right)}{\log(X_A(i))} \quad (1)$$

The result of this metric is a number between 0 and 1 that quantifies how concentrated or uniformly spread the scanned destination addresses are within the /16 network. For example, if a single source scanned 200 destination addresses in the same /24 subnet, then the third byte entropy will be equal to 0. This indicates the existence of a high-density cluster, because all targets are in the same range. On the contrary, if the 200 destination addresses were scanned on 200 distinct /24 subnet ranges, then the third byte entropy would be equal to 1. This indicates a random uniformity, because all targets are spread across all ranges. An entropy of 0.5 is the result of an equal combination between uniformity and clustering. For example, an exhaustive scan of all destination addresses of a /16 network reunites both uniformity and high concentration. Therefore, it will produce an entropy of 0.5. Figure 9 displays, for each of the 39,225 sources recorded during the

six-weeks' data collection, the distributions of the total number of destination addresses and the percentage of sources for the third and fourth byte entropy of destination addresses scanned in Network A and B.

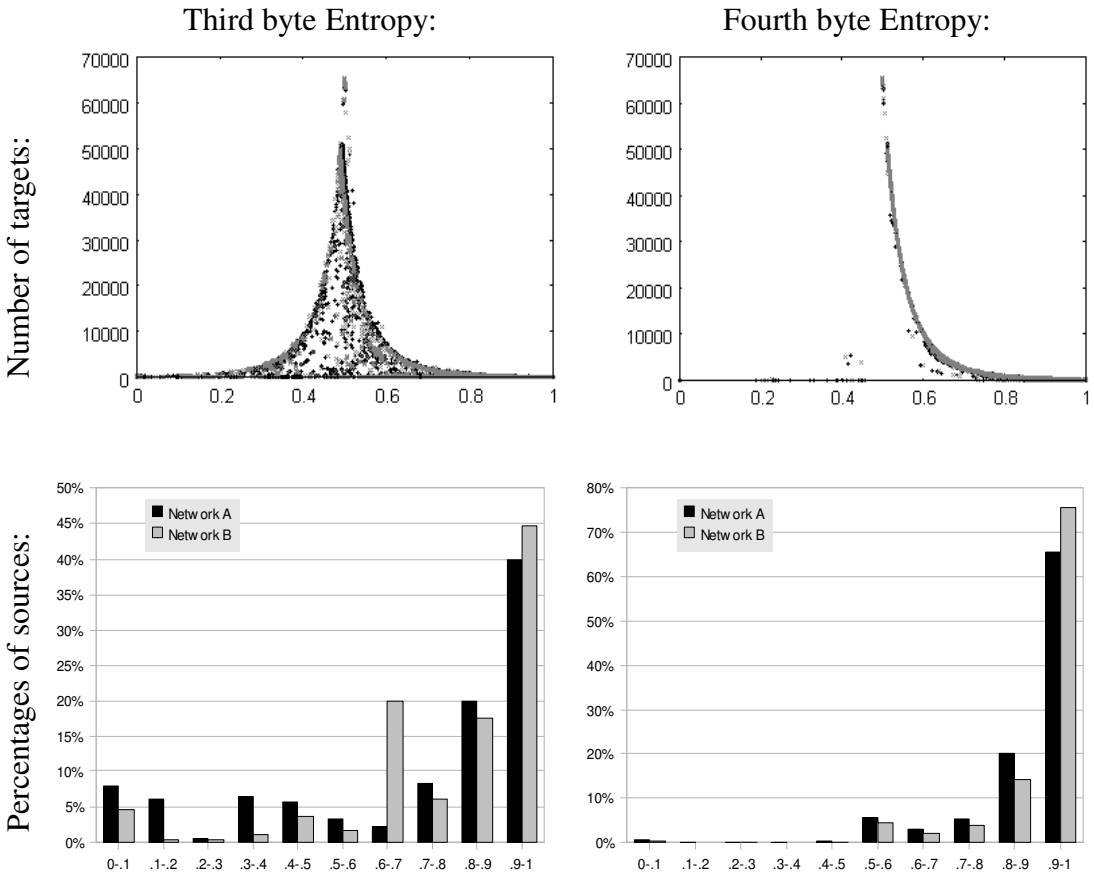

Figure 9: Distributions of the number of targets and the percentage of sources for the 3rd and 4th byte entropies of destination addresses scanned in Network A and B.

Two important remarks can be made based on Figure 9:

- On both Network A and Network B, 99.5% of the sources have a fourth byte entropy above 0.5, and 88% are above 0.8. This is a strong evidence that almost no source is targeting specific fourth bytes across multiple /24 subnets. This profile is clearly highlighted by the distribution of the number of targets for the fourth byte entropy displayed in Figure 9

- The third byte entropy reveals a similar result with 82% of the sources above 0.5 and 61% above 0.8. This means that only 18% of the sources are targeting specific ranges across the /16 networks.

These results show that the overall scanning activity for more than 80% of the sources is not to target specific /24 subnet ranges across the /16 networks.

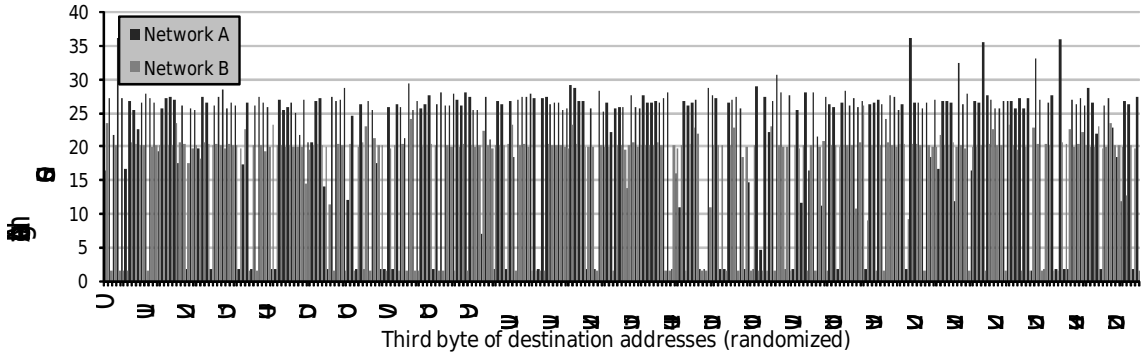

Figure 10: Average number of sources for the third byte of all destination addresses scanned

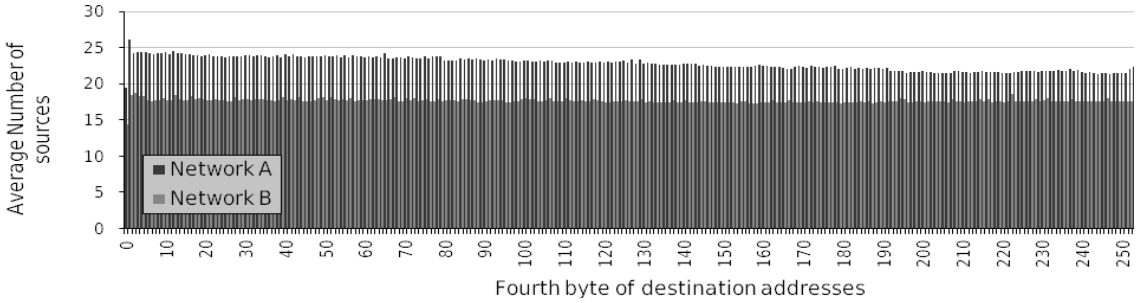

Figure 11: Average number of sources for the fourth byte of all destination addresses scanned

This conclusion is confirmed by Figures 10 and 11 that provide the average number of sources for the third and fourth byte, respectively. Both graphs show strong linear trends for the average number of sources per byte for Network A and B. More precisely, if we remove the gaps in Figure 10 (where the average is below 2 sources per byte), which are due to unallocated subnets where most of the traffic is not routed, we obtain for the third byte an average of 25.4 sources per byte for Network A with a standard deviation of 4.5,

and an average of 20.2 for Network B with a standard deviation of 2.0. The linear trend is even stronger for the fourth byte, where we found an average of 22.8 sources per byte for Network A with a standard deviation of 1.1, and an average of 17.6 sources per byte for Network B with a standard deviation of 0.6. These low standard deviations indicate that the overall scanning activity is uniformly distributed across the two /16 networks. Nonetheless, we notice in Figure 11 that the trend is slightly decreasing for Network A, from an average of 24 distinct sources per day at the beginning of the range to 22 at the end. This means that on Network A, for a given /24 subnet, IP addresses at the beginning of the range are receiving scans from 8% more sources on average than those at the end.

The conclusions of this analysis are that 1) we observed differences between the two /16 networks of our organization's network, and 2) the location of monitored IP addresses within a /16 network will have almost no impact on the overall coverage for this single /16 network. The goal of the next section is to estimate this coverage for a single /16 network as a function of the size of the darknet.

5.3. Coverage for different darknet sizes

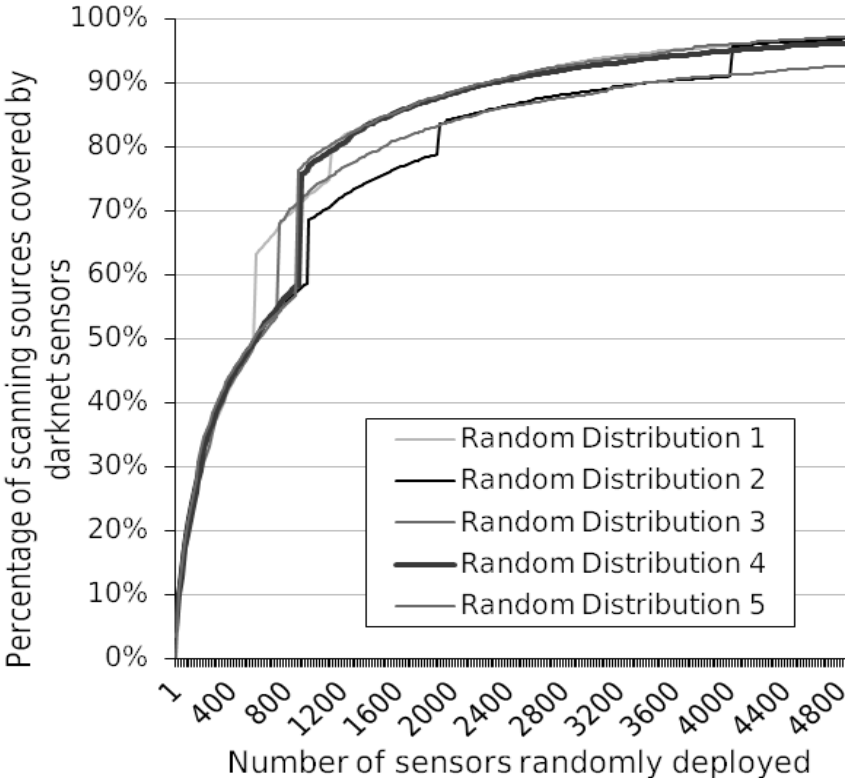

Figure 12: Scanning activity coverage for different darknet sizes within Network B

The methodology to assess the coverage reached by different darknet sizes consists of correlating the scanning data collected during our six-week experiment with a given set of unused IP addresses. We focused on only one of the two /16 networks, so the unused IP addresses were randomly selected within Network B only. To make sure that the random selection was not affecting the results, we ran five different simulations using five distinct random seeds. The simulation consisted in increasing the size of the darknet by adding twenty new randomly located monitored IP addresses at each step. After each growth, we correlated the number of external scanning sources that our set of simulated unused IP addresses would have covered during the six weeks. We stopped the simulation after 250 steps leading to a final darknet of 5,000 IP addresses. Figure 12

provides the overall results for the five simulations. The coverage is given in percentages based on the total number of 21,837 external sources that we collected in Network B during six weeks. Figure 12 shows that the coverage for a growing darknet size follows a logarithmic function, except for few increments where we observe jumps of up to 17% in the percentage of attackers covered. After investigation, we discovered that these jumps were because of the presence in the data of "vertical" port scans. A vertical port scan occurs when a single attacker probes a large number of ports of a single targeted host. From our definition in Section 3, we counted one new source for each new port probed, even if the probes came from a single source IP address. Consequently, when the target of a vertical port scan was added as part of our simulated darknet, the number of new sources covered jumped by several thousands.

We note also from Figure 12 that a single unused IP address randomly located is covering on average 717 sources, which represents 3.3% of the total number of sources. A set of 560 monitored IP addresses randomly located covers 10,729 sources, which represents almost 50% of the total number of sources.

Using a logarithmic regression, we estimated on average, based on the empirical data displayed in Figure 12, that the equation followed by the coverage C of a single /16 network in function of the number N of unused IP is given by (2):

$$X = 0.18 \cdot \ln(N) - 0.52 \qquad (2)$$

The average error distance between the data collected and this logarithmic equation is only 3%.

The next step to learn more about attackers targeting our organization's network is to increase the size of our darknet. The empirical assessment provided in Figure 12 allows

our team of security researchers and network operators to determine the malicious activity coverage they would prefer to study using darknet sensors. After setting this goal, another important information to take into consideration is the technology requirements for collecting network traffic from several hundreds or thousands of unused IP addresses. [86], [88], [43], [62] and [7] detailed innovative ideas to deploy such architecture.

6. Summary

We investigated the significance of data collected by darknet sensors deployed at a the University of Maryland's network. We showed, based on empirical data collected over six weeks on two /16 networks, that 66% of the external TCP traffic collected by unused IP addresses were from sources scanning the university's network. We quantified the daily scanning activity that occurred on the university's network, and we demonstrated that the destination addresses targeted by external scans were uniformly distributed within each of the two /16 networks. In particular, we measured that less than 18% of attackers scanned specific clusters of destination addresses. Finally, we assessed the coverage of the overall scanning activity reached by a given darknet size by correlating the empirical data with multiple sets of 1 to 5,000 monitored IP addresses randomly located. The conclusion of this assessment was that a single unused IP covers 3.3% of the scanning activity, but we can extend this coverage to 50% by increasing the number of network addresses monitored to 560.

The next step toward precisely assessing malicious activity at the scale of the University of Maryland's network is to increase the size of our honeynet and to instrument it with high interaction honeypots. The problem is that with current honeypot technologies, deploying a large number of high interaction honeypots requires an important amount of

hardware and human resources. The solution is to develop a hybrid honeypot architecture in order to increase the scalability of high interaction honeypots while using only limited resources. This was our motivation to start building an advanced hybrid honeypot architecture. We introduce the design and the evaluation of this architecture called Honeybrid in the next chapter.

CHAPTER 4

HONEYBRID: HYBRID HONEYPOT ARCHITECTURE

1. Introduction

Honeypots provide high quality attack datasets that help measuring and understanding network threats. However, honeypots are either expensive to administer and poorly scalable (high-interaction honeypots) or based on emulated resources that limit the level of detail they can collect about attacks (low-interaction honeypots). We showed in the previous chapter that for a large organization such as the University of Maryland, we needed to deploy honeypots on 560 IP addresses in order to collect malicious activity from 50% of attackers.

To reach such scalability while minimizing the cost of deploying honeypots, we introduce in this chapter Honeybrid, a hybrid honeypot architecture that offers high flexibility through an active network redirection mechanism. This flexibility manifests through the capabilities to 1) write custom filtering and redirection policies, and 2) plug existing low and high-interaction honeypots to the architecture.

Figure 13 indicates which components of our framework are involved in this chapter.

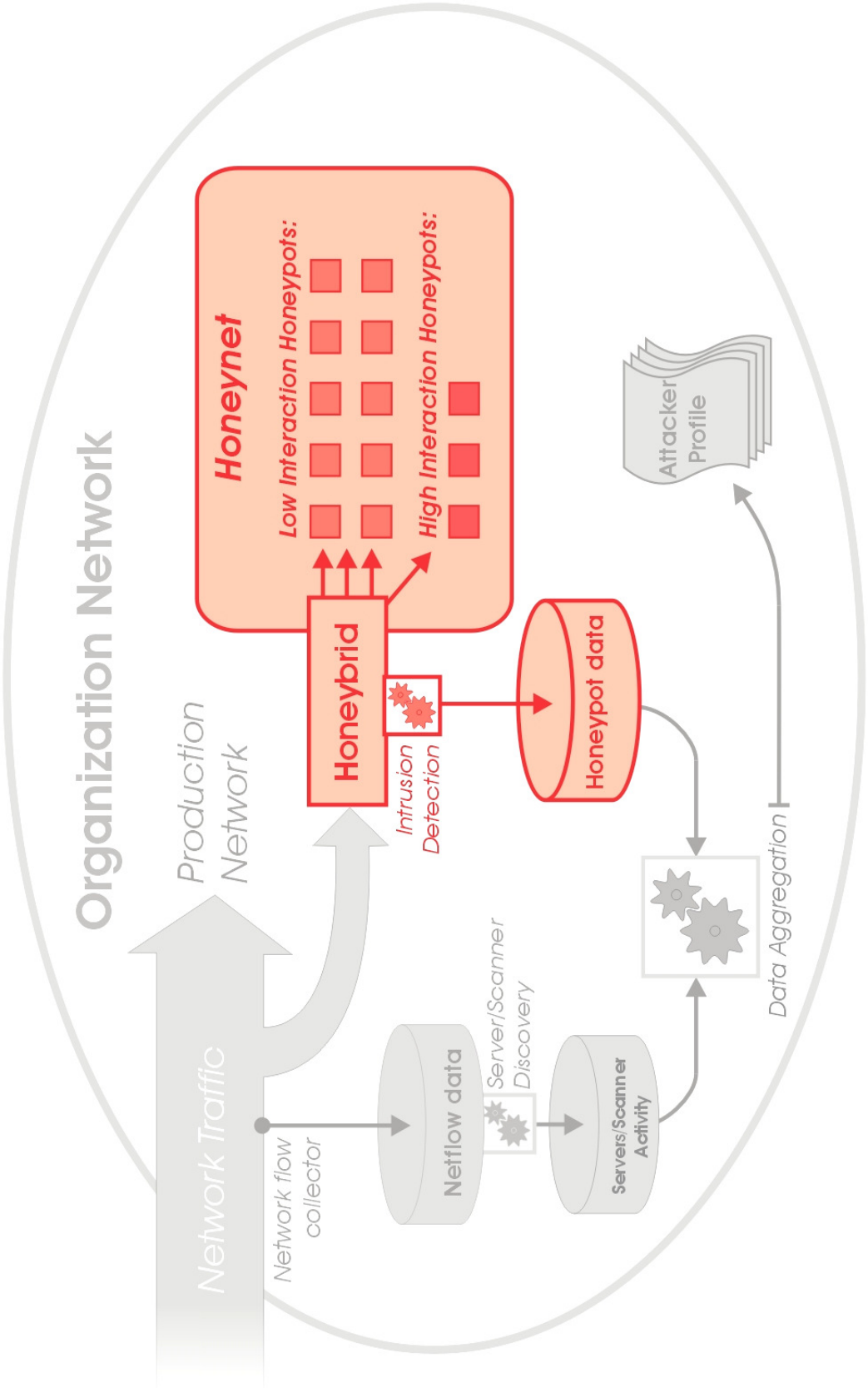

Figure 13: Overview of the components of our framework involved in Honeybrid

This chapter is organized as follow: Section 2 is dedicated to review the related work on hybrid honeypot architectures. In Section 3 we introduce the architecture of Honeybrid and we detail the functionalities of both the Redirection Engine and the Decision Engine. In Section 4 we describe possible applications of Honeybrid. In Section 5 we provide the results of our evaluation. In Section 6 we discuss the current limitations and related future work. Finally in Section 7 we provide a summary of this chapter.

2. Related Work

The idea of extending the scalability of high-interaction honeypots has spread through several research projects.

The simplest solution to achieve this goal is to use a network address translation firewall (NAT), as described in [63], that consists in translating a large set of IP addresses into a small set of high interaction honeypots by modifying in real time destinations addresses of incoming attacks. The problem with such solution is that it lacks the capability to filter attacks, and high interaction honeypots can easily become overwhelmed by a large volume of incoming probes. As a result, more advanced solutions have emerged. So far we can distinguish three areas of improvement:

1. Solutions to enhance resource management at the high-interaction level;
2. Architectures centralized on a smart gateway to filter out uninteresting traffic;
3. Solutions to generate automatically network responders at the low-interaction level.

The focus of Honeybrid is the second area: using a smart gateway to better scale high-interaction honeypot deployments. We review the projects related to all three of these research areas in the following sub-sections.

2.1. Enhancing Resource Management

The concept of Collapsar [43] is to collect detailed traffic from multiple remote networks and to minimize the cost and the risk of deploying high-interaction honeypots. The authors used a farm of virtual machines to centralize high-interaction honeypots in a uniform architecture, and deployed a set of Generic Routing Encapsulation (GRE) tunnels to forward traffic from different locations. Thus, to attackers, honeypots appear in different networks, but the centralized physical location has the advantages of reducing constraints of deployment, monitoring and analysis. For example, such architecture uses a central tarpitting module for mitigating outgoing attacks from all honeypots. This module slows down outgoing connections and alters exploit payloads sent by honeypots that match harmful attack signatures. Collapsar has also a central correlation module that provides data mining functionalities over threats spanning on multiple networks. A limitation of this architecture is that there is a one-to-one relation between the number of IP addresses configured to collect traffic and the number of high-interaction honeypots.

Another project called Potemkin [86] was built to collect traffic from large IP spaces using high-interaction honeypots only. The idea of the project is based on the fact that honeypots have no production value: when they do not receive malicious traffic they do not have to run at all. Consequently, the architecture creates a new virtual machine for each active IP address. When an IP becomes inactive, the virtual machine is destroyed to save physical memory and CPU resources. Such system allows hundreds of virtual

machines to run on a single physical host. Another interesting innovation of Potemkin is the containment module that prevents corrupted honeypots from attacking the outside world. When an outgoing connection is detected from a compromised virtual machine, a system called the reflector creates a new virtual honeypot with the destination address corresponding to the denied outbound packet. Thus, the whole Internet can be virtualized and researchers can observe for example the propagation behavior of a worm that tries to spread. Contrary to Collapsar, the scalability Potemkin was tested on a /16 network to collect traffic from 65,000 IP addresses. On average, 58 active virtual machines were required, but during peak activity, over 10,000 virtual machines were running concurrently.

2.2. Using a Smart Gateway

In [7], the authors implemented a redirection mechanism similar to ours, where they increased the exposure of high-interaction honeypots by employing low-interaction honeypots as front-end content filter. They showed based on five months of darknet data that 50% of packets collected did not have payloads, and 95% of the payloads received had been seen before. Thus, they established that redirecting only filtered connections to high-interaction honeypots was a scalable solution even for very large IP spaces. This work differs from ours in two ways: first the authors did not provide any detail on the design of their implementation, second their framework does not offer other redirection policy than sampled payload digest or source-based filtering.

2.3. Generating Low-interaction Responders

Another solution built with the motivation of extending the scalability of high interaction honeypots to collect new worms is the GQ architecture [26]. This project is

the most similar to our architecture since it features both a honeypot independent framework and a dynamic replay mechanism. The replay mechanism is built on top of the RolePlayer project [27] and allows real time updates of the database of known attacks for maximum filtering efficiency. RolePlayer is able to build client/server scripts from only two well selected instances of the same attack, without any knowledge about the underlying application protocol. Thus, the RolePlayer-based proxy of GQ can dynamically select a redirection threshold when an incoming attack deviates from a previously collected sample. However, the authors were not yet able to develop a truly scalable implementation of their solution. They evaluated their approach from a functionality point of view and they show that it can handle a maximum of 100 incoming connections per second. As we will see in Section 5, our implementation is able to handle more than 250 incoming connections per second.

A project similar to GQ is SGNET [49], which is built on top of ScriptGen [50]. SGNET is a distributed honeypot system that reduces the load on high-interaction honeypots by generating automatically low-interaction honeypot scripts. These scripts are built to provide a high-depth view of the attack with a limited resource consumption. The main contribution of SGNET is to be able to automatically learn the behavior of a given network protocol and to translate it into a script. This learning takes place by using high interaction honeypots as oracles to extract information from attackers. As a result, the scalability of the SGNET architecture is increasing over time. At the beginning, all connections are forwarded to high-interaction honeypots and the system is poorly scalable. But then after few weeks, most of the connections are handled by the generated scripts and so the load on the high interaction honeypots is greatly reduced.

The GQ and SGNET architectures are hybrid architectures that not only use a filtering gateway but also a system to automatically generate low-interaction responders from empirical data. Compared to our project, the two advantages of these approaches are that 1) it removes the dependency on hand-built low interaction scripts to serve as the front-end responder; and 2) it allows a very accurate filtering policy to filter out known attacks from the data collection and save back-end resources. However, these advanced systems are rigid architecture with the only scope of collecting worm attacks. The advantages of our architecture are its flexibility and scalability. Our highly modular filtering policy allows researchers to deploy and combine filters for a large variety of experiments, including the large scale analysis of worm attacks. Moreover, Honeybrid was built without any requirement on the low and high interaction honeypots deployed. Thus, protocol reverse engineering algorithms such as those implemented in GQ and SGNET could be easily plugged to our architecture.

Two other interesting protocol reverse engineering systems are Discoverer [25] from the authors of GQ, and [73], which focuses on extending web-honeypot capabilities. This research project consists of 1) a dynamic content generation system that build a corpus of vulnerable web requests/responses from training data; and 2) an online-classification module able to deduce the response that is most similar to the incoming request. This system was deployed live during two months. The uniqueness of this project was to be able to reference the honeypots in search engine indexes in order to collect targeted attacks. This technique greatly paid off with 368,000 attacks targeting several hundred distinct webapps, including 0-day threats recorded in only two months.

3. Architecture

Honeybrid's architecture is shown in Figure 14. Honeybrid is divided into three parts: a gateway, a set of low-interaction honeypots and a set of high-interaction honeypots. The idea of the architecture is to monitor a large number of IP addresses using the scalable low-interaction honeypots. By default, all incoming traffic is routed to the low-interaction honeypots. These low-interaction honeypots provide the necessary interaction for: 1) the attacker to establish a network session, and 2) the gateway to detect interesting attack processes among the network sessions established. Information such as source IP address, destination port or payload, is gathered to decide what sessions are worthy of further investigation. As soon as an interesting attack is detected, the gateway is in charge of transparently redirecting the flow of the interesting attack from the low-interaction honeypot to the high-interaction honeypot. Finally, the role of the high-interaction honeypot is to offer full interaction to the attacker, in order to record detailed information about the attack process that was flagged as interesting.

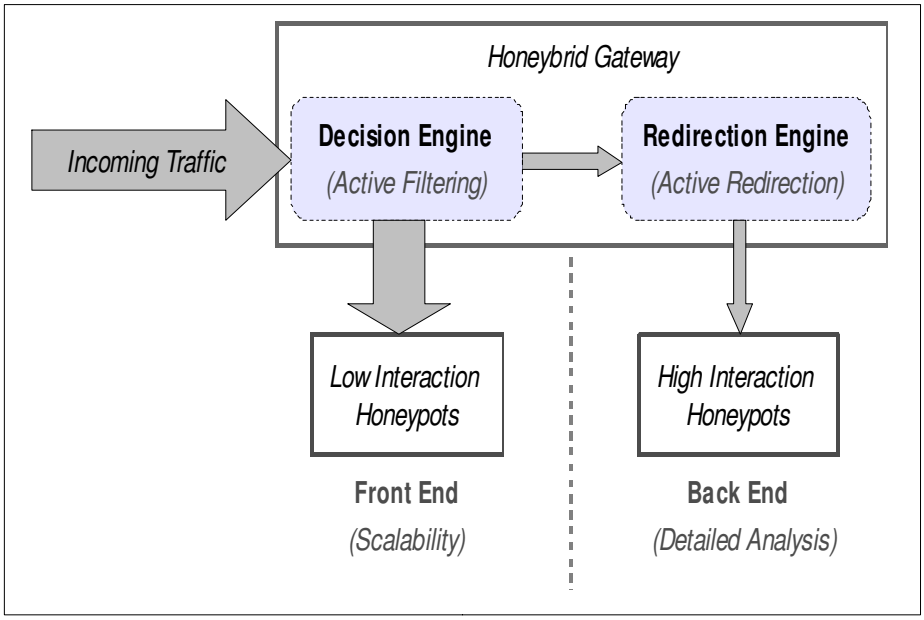

Figure 14: Overview of the architecture

Therefore, the central part of the architecture is the Honeybrid gateway, because it is in charge of orchestrating the filtering and the redirection between the front-end of low-interaction honeypots, and the back-end of high-interaction honeypots. This gateway hosts the Redirection Engine and the Decision Engine. The Decision Engine filters network traffic, which means it selects network sessions worthy of analysis from the overall traffic received. The Redirection Engine handles selected network sessions by transparently changing the destination of selected sessions from low-interaction honeypots to the farm of high-interaction honeypots. The design of these two engines is discussed in the next two sub-sections.

For a better flexibility, the Honeybrid gateway is connected to the low and high-interaction honeypots through the TCP/IP network and is fully responsible for logging and recording the network traffic collected. As a result, there is no requirement regarding the low and high-interaction honeypots besides replying to incoming attack traffic. This

means that once the gateway is installed and configured, any low and high-interaction honeypots can be plugged in the architecture, even remotely. For our evaluation, we used Honeyd [62] as front-end and we tested both Qemu images [13] and Nepenthes [2] as back-end.

3.1. Decision Engine

The first question that we had to address to design the Decision Engine was: how to define an interesting attack? In other words, what does an attack worthy of further investigation exactly mean? The problem with this question is that the answer is subjective and depends on the type of experiment security researchers want to conduct. For example, here is a list of potential answers:

1. Attacks matching a specific fingerprint;
2. Attacks presenting an original content that was never seen before [32];
3. Attacks sending commands that are not implemented in the low-interaction honeypots.

Definitions 1 could be used to investigate all attacks targeting a specific vulnerability, definition 2 could be used to track 0-day attacks, and definition 3 could be used to help improving scripts that emulate services. This list is not exhaustive and we believe that a large number of criteria could be applied to classify a network attack as interesting. For this reason, we developed our filtering solution in a modular fashion, where each module is the implementation of a specific criterion. Criteria are applied on network sessions with a packet-level granularity. For example, we coded a module called HASH that implements the criteria: “the connection presents original content”. This module simply computes a

message digest of the payload of each packet received using the SHA1 algorithm, and then compares this digest against a database of known payloads. If a match is found, it means that the payload has already been seen before and therefore the connection is not redirected. Otherwise, the payload is recognized as original content and the Decision Engine sends a signal to the Redirection Engine to redirect the connection to a high-interaction honeypots. To allow multiple criteria to be applied on the same network session, the Decision Engine can combine them using a Boolean equation. For example, the rules to redirect a network session could be:

1. "The session is destined to port TCP/25" *and*
2. "The session sends original content" *or*
3. "The interaction limit of the low interaction script is reached"

This set of rules means that all TCP connections targeting port 25 and matching rules 2. or 3. will be redirected to the high-interaction farm of honeypots. Rules 2. and 3. are implemented as independent modules in the Decision Engine. As mentioned earlier, these modules process each packet received by the architecture and return a Boolean value to the Decision Engine. This Boolean value indicates whether the criterion is met (*true*) or not (*false*). These answers are computed according to the Boolean equation and the Decision Engine labels a session as worth redirecting if the result of the Boolean equation for the sequence of packets received is *true*. The Decision Engine reads these Boolean equations using a configuration file organized per honeypot and per port. As a result, one can define a complete redirection policy for each IP address monitored by the architecture and for each service opened. For example, the Boolean equation that we

presented, applied to IP address 192.168.0.3, would be written in the configuration file with the following command line:

```
192.168.0.3:25 ->      HASH(db_payload) OR LIMIT(smtp_script)
```

where HASH is the module in charge of detecting unknown payloads, db_payload is the database of known payloads against which HASH should compare newly received payloads, LIMIT is the module in charge of checking that the interaction limit of the low-interaction honeypot that replies to the connection is not reached. Finally, smtp_script is the script that emulates the SMTP service on the low-interaction honeypot that replies to connections on port 25. We provide additional examples of modules and use cases in Section 4.

The next step after flagging that a network session was *interesting* is to redirect it. This task is handled by the Redirection Engine that we present in the following section.

3.2. Redirection Engine

The Redirection Engine offers the possibility to forward an active network session between the low-interaction honeypots and the high-interaction honeypots. To guarantee such functionality, we first have to address the following question: how to *redirect* and *preserve* an active network session?

The different TCP/IP protocols involved in network communications were built to transfer information from a source *S* to a destination *D* over one or multiple networks. However, the protocols were not conceived to transfer information from a source *S* to a first destination *D1*, and then to another destination *D2* within the same network communication. This *S*-to-*D1*-and-then-*D2* relationship is the main functional

requirement of Honeybrid. The second implicit requirement is that the source *S* must not detect that the destination *D1* has been changed to *D2*.

The key problem to meet these requirements is the presence of *states* in the communication. When a source and a destination communicate, they both evolve in specific states that manifest at different layers of the communication model. For our redirection system to work correctly, we need to guaranty that *S*, *D1* and *D2* all move to coherent states. For instance, if *S* starts a communication with *D1*, then after few packets exchanged, *D1* will have moved to a different state that *D2* needs to reach before being able to take over the communication with *S*. In the five-layer TCP/IP model [77], these states appear at the transport layer and at the application layer. The network layer is stateless, therefore, the only task of the Redirection Engine at this layer is to replace the IP address of *D1* by the IP address of *D2* in the redirected communication, like a standard NATing device. Of course, this replacement is performed only between the Honeybrid gateway and *D2*, because *S* should not see the address of *D2* in the packets it sends or receives.

At the transport layer, the situation becomes more complex because the communication can be stateful. Honeybrid handles both UDP and TCP protocols. UDP's stateless nature implies that the Redirection Engine only needs to update the checksum of the UDP headers in redirected UDP packets. TCP's stateful nature implies that more than the checksum field needs to be updated in the TCP headers of redirected TCP packets. TCP was built to provide a reliable stream of packets between the source and the destination of the communication. This reliability is guaranteed through several mechanisms that prevent packets to be lost, or to be received at the application layer in

the wrong order. These mechanisms include a sequence number randomly generated by the source and the destination, then acknowledged and incremented by each network stack to guaranty that all the packets are correctly transmitted. Therefore, when the Redirection Engine takes over a TCP connection established between *S* and *D1* and redirects it to *D2*, it needs to update the sequence and acknowledgment numbers of both packets coming from *S* and packets coming from *D2* to preserve the characteristics of the header that were established between *S* and *D1*. These characteristics also include the size of the TCP window and the TCP options.

At the application layer, the only requirement is to prepare *D2* to accept the communication already started between *S* and *D1*. The Redirection Engine performs this task using a replay mechanism. This means that all packets already sent by *S* are stored in memory, in order to be replayed to *D2* if the communication is redirected.

Figure 15 summarizes the different mechanisms involved in the redirection. The Redirection Engine works in three phases:

- *Initialization phase:* incoming packets from *S* are forwarded to *D1* while being inspected by the Decision Engine;
- *Replay phase:* if the Decision Engine flags the connection as worth redirecting, it sends a signal to the Redirection Engine that starts replaying the connection to *D2*;
- *Forwarding phase:* when *D2* is ready to take over, packets are proxied between *S* and *D2*. Thanks to the update of the TCP headers of all packets proxied during this phase, *S* believes it still communicates with *D1* from the connection

initialized in phase 1, while *D2* believes it still communicates with the gateway from the connection initialized in phase 2.

It is important to notice that while most of the network attacks can be redirected through this mechanism, some specific attack processes are out of the scope of our current architecture. These processes include connections based on cryptographic protocols such as SSH or HTTPS. However, in such cases, Honeybrid can still be configured for specific services and IP addresses to skip the low interaction front-end and to forward directly the connections to the farm of high-interaction honeypots.

Another functionality of Honeybrid is to record in two different PCAP files the non-redirected and the redirected traffic. This last file collection allows researchers to directly work on a reduce dataset containing only interesting network events.

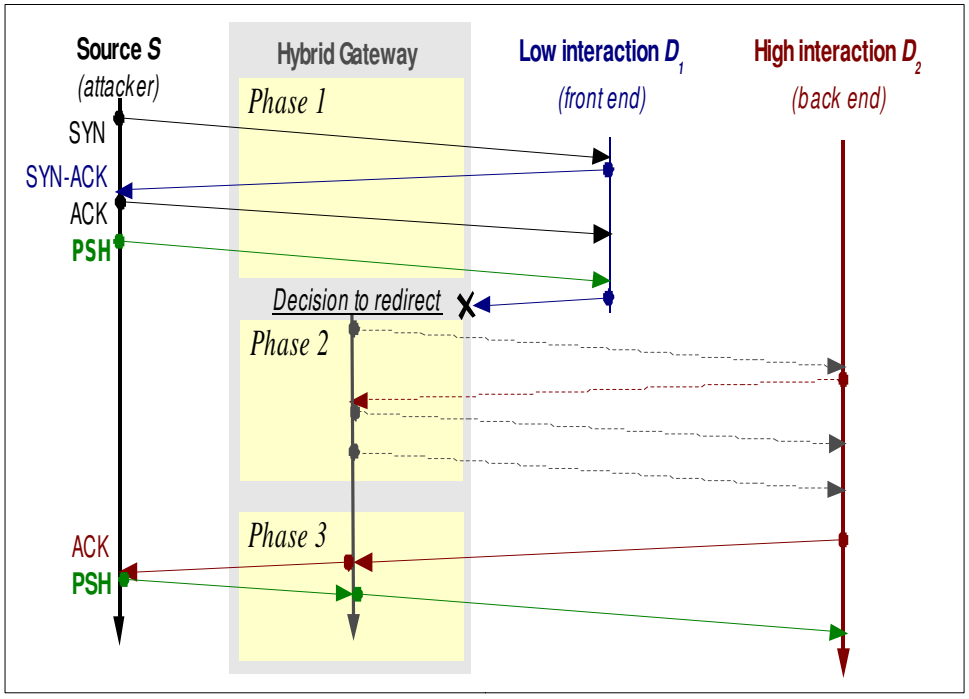

Figure 15: Illustration of the Redirection Mechanism for a TCP Connection

4. Application

Honeybrid was built to provide a high level of flexibility in order to suit a large diversity of honeypot experiments. This flexibility is mainly expressed through the Decision Engine and its set of modules. We described in Section 3.A the following two modules:

- HASH: to redirect attacks presenting an original content that was never seen before (content-based filtering);
- LIMIT: to redirect attacks made of commands that are not implemented in the low-interaction honeypots.

We also implemented a set of modules specifically built to simplify large scale honeypot deployment:

- SOURCE: to redirect only the first connection of each IP address (source-based filtering). This module works similarly to the HASH module, by maintaining a table of known items. In the case of SOURCE, these items are source IP addresses, that are automatically expired after a given period of time;
- RANDOM: to randomly redirect one out of any given number of connections. For example RANDOM(100) would randomly redirect 1% of incoming connections;
- SAMPLE: to sequentially redirect one out of any given number of connections. For instance, SAMPLE(40) would redirect one every 40 connections;
- LOAD: to redirect connections up to a given high interaction load. For example, LOAD(10) would stop redirecting after 10 connections per high interaction honeypot per second;
- COUNTER: to redirect connections after a given number of packets.

We note that these modules are extremely simple to add to the architecture. The smallest one is SAMPLE with 8 lines of C code, and the largest one so far is the HASH module with less than 150 lines of code. The fact that they can be combined greatly leverage the capabilities of Honeybrid. For example, it takes only one line in the configuration file of Honeybrid to collect unique attacks targeting web servers on a large network of 65,535 IP addresses (/16) with a single high-interaction honeypot:

```
192.168.0.0/16:80 -> HASH(web_db) AND LOAD(10)
```

The HASH module will filter out non-unique attack requests, and the LOAD module will guarantee that the high-interaction honeypot is never overwhelmed. By adding the RANDOM module to the equation, we can collect additional attack requests that would look identical after the first payload, but that could then potentially differ:

```
192.168.0.0/16:80 -> (HASH(web_db) OR RANDOM(10)) AND LOAD(10)
```

Thanks to the versatility of our solution, another possible usage of Honeybrid is be to deploy it in front of a legitimate server and dynamically redirect suspicious connections to a high interaction honeypot configured exactly like the server. Benefits would be to 1) prevent the legitimate server from being potentially compromised, and 2) study the suspicious connection in a safe and controlled environment. This functionality would be similar to the one provided by the BaitnSwitch project [10] that has now been implemented into Snort [9].

5. Evaluation

We evaluated Honeybrid in two ways: first, we measured the impact of the replay mechanism on the duration of a redirected connection. Then, we empirically evaluated the performance of the architecture configured with the HASH module. The first part addresses the problem of how a remote attacker could potentially fingerprint the architecture. The second part validates the functionalities of the architecture in a live environment.

5.1. Evaluation and impact of the replay mechanism

We studied in this section the latency added to a TCP connection by the replay mechanism for an increasing number of packets replayed. The experiment consisted of

four computers: an attacker running Metasploit, two identical virtual machines running Windows 2000 using Qemu, and the Honeybrid gateway configured to redirect TCP connections after a given number of packets from the first virtual machine to the second one. We note that all the measurements part of these evaluations were repeated three times, and so each numerical result presented in Figures 16 and 17 is an average from three consecutive measurements.

The attacker was configured to send a buffer overflow exploit attempt via the Microsoft Netbios service on port TCP/445. This exploit is launched through a single TCP connection for a total of 40 packets. We note that this overflow is based on the MS03-49 vulnerability for which the Windows 2000 virtual machines were not vulnerable. We could therefore repeat the experiment without compromising the virtual honeypots but still checking using the output of Metasploit that the exploit attempt was correctly carried out while being transparently redirected.

We measured the duration of the three phases of the redirection from the attacker perspective:

- The *initializing phase*, in which the attacker communicates with the first virtual machine,
- The *replaying phase*, in which Honeybrid replays the initialization phases with the second virtual machine,
- The *forwarding phase*, in which the attacker communicates with the second virtual machine.

The boundary among these three phases was given by the precise attack packet that triggered the redirection. We changed the redirection rule incrementally to redirect connections on port 445 after up to 20 packets from the attack. We note that only packets carrying a payload and coming from the attacker were counted as *attacker packets*. So the TCP handshake was not counted in the number of attacker packets replayed. We can see from the results provided on Figure 16 that the duration of the replaying phase (difference between the end of the replaying phase and the end of the initialization phase) increases according to the number of attacker packets to replay. The duration of the initialization phase also increases because Honeybrid waits longer before triggering the redirection. Finally the total duration of the connection increases because of the overhead of the replaying phase. This total duration can be compared with the first measurement (number of packets equal to zero) that corresponds to a non-redirected connection.

Figure 17 shows in more detail the exact duration of the replaying phase. It is interesting to see that redirecting between 1 and 5 attack packets takes less than 40ms, which is approximately equal to the maximum RTT delay measured for a non-redirected connection. This indicates that up to 5 packets, the redirection mechanism would be totally invisible to our local attacker. After 5 packets, the duration increases from 60ms up to 120ms. Such delays are commonly found on the Internet and indicate that the redirection mechanism would also be invisible for a remote attacker even up to 20 packets replayed.

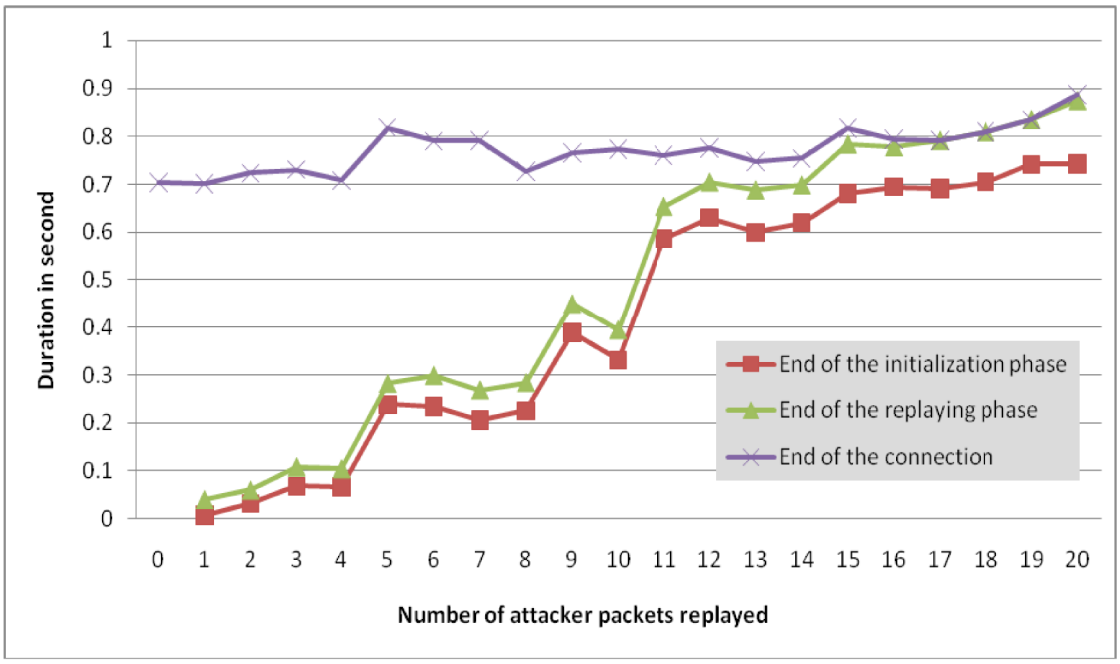

Figure 16: Durations of the initialization, replaying and forwarding phases (end of the connection) for different number of packets replayed

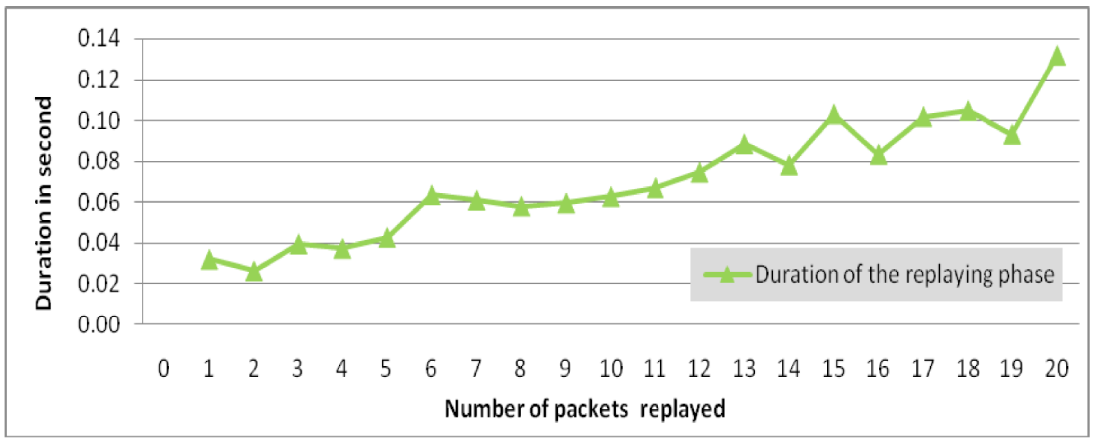

Figure 17: Duration of the replaying phase for an increasing number of packets replayed

5.2. Empirical results

To validate the functionality of Honeybrid, we deployed it on five /24 networks during 48 hours. These networks represent a total of 1,275 IP addresses. Honeyd was deployed as a low-interaction front-end and Nepenthes was deployed as a high-interaction back-end to offer vulnerabilities and collect exploits. A collection of 29 TCP ports were

opened by Nepenthes. Honeyd was configured to reply to these 29 ports to initiate TCP connections with attackers. All other ports were closed. Honeybrid was configured with the HASH module on each of these 29 ports. An hourly graph of the number of connections handled by Honeybrid is depicted on Figure 18. This graph shows that an average of 2,400 connections were received every hour, ranging between a minimum of 1,016 and a maximum of 7,250 connections. The peak on hour 22 is because of a massive attack targeting web servers (port TCP/80). This event is interesting because we recorded that Honeybrid was able to correctly handle up to 257 connections per second. We also recorded that the Decision Engine sustained a rate of 155 connections per second and the Redirection Engine a rate of 48 connections per second. These values give an idea of the traffic rate that can be received when 1,275 IP addresses are instrumented. We plan as future work to deploy Honeybrid on a much larger network in order to study the maximum traffic rate it can handle.

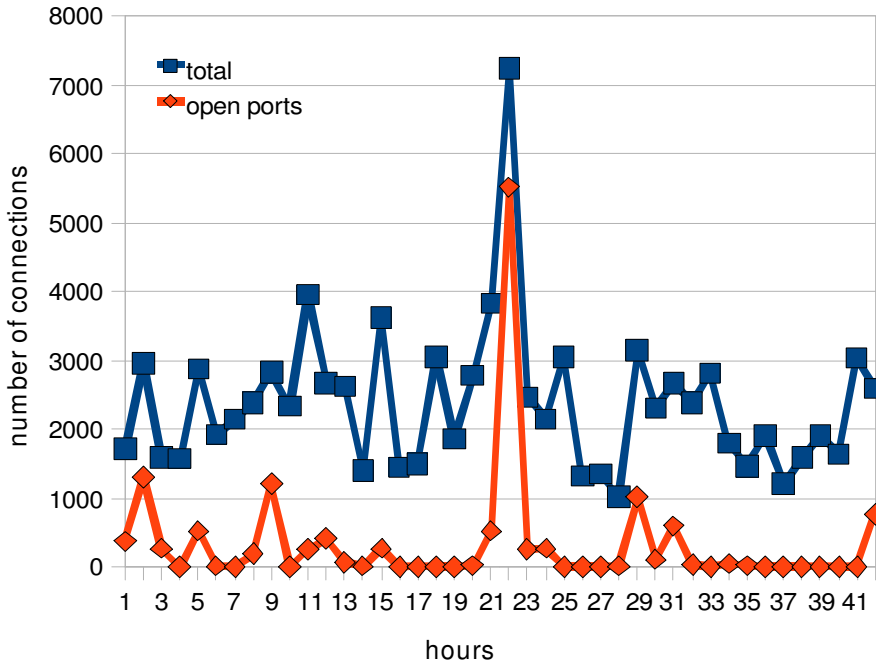

Figure 18: Hourly number of connections handled by Honeybrid, for both all ports and open ports.

To investigate if the Decision Engine and the HASH module correctly worked, we extracted data from hour 22 for port TCP/80 related to the massive web attack that lasted 12 minutes. The results are detailled on Figure 19. A total of 5,522 connections were handled by Honeybrid. The series *redirected* represent the 2,550 connections that had a unique payload which was not yet recorded in the HASH database. These connections were thus replayed and then forwarded to Nepenthes. 113 connections had also a unique payload and started to be replayed, but the replay process was not completed because Nepenthes was too busy and could apparently not reply to all of Honeybrid's requests. These connections are represented by the series *interrupted* and represent 4.24% of the 2,663 connections that carried a new payload. Finally the series *not redirected* represent the 2,858 connections that had a payload already recorded in the HASH database. Theses connections were handled only by Honeyd and they were not recorded as part of the

Nepenthes traffic. After four minutes of attack, we clearly see on Figure 19 that the number of redirected connections drops while the number of non redirected connections raises. This indicates that the HASH table has caught up and connections with known payloads are automatically discarded by the Decision Engine. Each redirected connection represents one unique attack payload that was added to the HASH table. After four minutes, the table made of 2,663 hashes was used to discard a total of 2,858 known attacks. As a result, traffic sent to Nepenthes was divided by a factor of two, as well as the size of the network data collection. These results show that Honeybrid and the HASH module worked as expected even under heavy load. The only issue was due to Nepenthes which was slightly overwhelmed at the beginning of the attack, when the HASH table was almost empty. We note that adding the module LOAD to the configuration of Honeybrid would have prevented this issue but would also have decreased the number of unique hashes collected. A possible solution is to use multiple backends in order to share the load of connections handled by high interaction honeypots while not affecting the number of unique hashes collected.

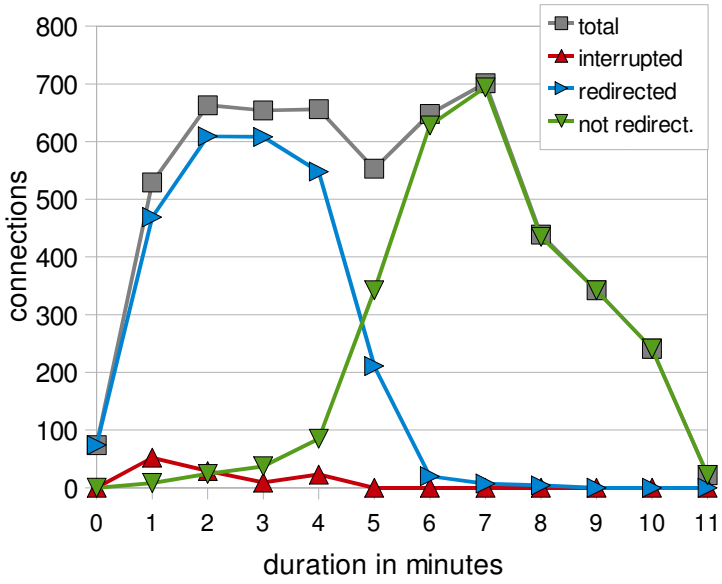

Figure 19: Evolution of the data collection handled by Honeybrid for a sustained attack targeting web servers.

Over the 48 hours of the experiment, Honeybrid collected a total of 3,491 unique payload hashes targeting web servers. After investigation, we were able to classify these hashes in 31 unique web requests, ranging from inoffensive "GET HTTP/1.1" to exploit attempts such as "GET /user/soapCaller.bs". We found that each web request had a large number of different hashes because the destination IP address targeted by the attacker was often included in the request (through the "Host" field of the HTTP header). This prompted us to modify the `HASH` module to replace any occurrence of the destination IP address with a generic string before calculating the hash. This small hack reveals that Honeybrid and its current set of module has a limited understanding of the attack processes, compared to architectures such as GQ [26] or SGNET [49]. But Honeybrid was built with the goal of simplicity and robustness, and attacks intelligence has to be handled through the system of modules. As we showed in this evaluation, Honeybrid offers a robust and flexible architecture where one can easily add more advanced

functionalities without sustaining the expensive cost usually inherent to honeypot experiments.

6. Limitations and Future Work

Our study contributes to the field of honeypot technologies by detailing the complete design of a hybrid architecture. The result is a robust and flexible implementation that offers a simple and scalable framework for honeypot researchers. As a result, the main advantages of Honeybrid over other honeypot implementations are:

- *Versatililty*: Honeybrid can be used as a front-end to handle traffic toward any low and high interaction honeypots;
- *Flexibility*: modules can be added to the Decision Engine of Honeybrid and combined to write highly customized filtering policy;
- *Scalability*: Honeybrid was built to process several hundred connections per second with a low resource consumption;
- *Simplicity*: Honeybrid is based on two robust engines, each supporting a single functionality. Modules can be easily added to the architecture to handle more advanced functionalities.

The current limitations of honeybrid compared to other advanced honeypot architectures are:

- *Multi-stage attacks*: Honeybrid works with a network session granularity, where network sessions are identified by protocol, IP addresses, and ports. Consequently, an attack made of multiple network sessions might be incorrectly spread out to different low or high interaction honeypots;

- *Attack intelligence*: Honeybrid relies on existing low-interaction scripts to extract information to make a decision. Unlike the GQ or SGNET architectures, Honeybrid lacks the capability to automatically learn and adapt itself to better collect network threats.

It is part of our future work to address these limitations. We are also working on 1) deploying Honeybrid on larger networks to precisely assess the impact of the filtering policy on the scalability of the architecture; and 2) implementing additional modules, including one to use the output of an Intrusion Detection System such as Snort to decide on network sessions worth redirecting.

7. Summary

In this chapter we presented Honeybrid, a hybrid honeypot architecture that offers two majors capabilities: 1) to combine the scalability of low-interaction honeypots with the advantage of high-interaction honeypots in order to collect detailed attack processes over large IP spaces using inexpensive resources; and 2) to offer a highly flexible framework on which security researchers can apply customized filtering and redirection policies, using the low and the high interaction solutions they need.

Honeybrid is based on a central gateway that hosts a Decision Engine and a Redirection Engine. The Decision Engine filters and selects network sessions worthy of analysis from the overall traffic received. The Redirection Engine handles selected network sessions by transparently changing their destination from the default low-interaction front-end to the back-end of high-interaction honeypots.

We evaluated these two engines by deploying Honeybrid during 48 hours on a network of 1,275 IP addresses. Honeybrid successfully sustained more than 250

connections per second and collected 3,491 unique hashes representing 31 unique probes or attacks targeting web servers. We also evaluated the latency of the replay mechanism, by showing that replaying 20 packets sent by an attacker would take less than 120ms.

Honeybrid is the central component of our attack assessment framework. The scalability offered by Honeybrid addresses the challenge of easily deploying a large honeynet with limited resources. To integrate Honeybrid in the organization network and to continue the implementation of our framework, we now need to include network flows. By providing an exhaustive view on all communications between internal and external hosts of the organization network, network flows can greatly assist the configuration of Honeybrid and the data analysis of malicious traffic. The goal of the next chapter is to introduce a scanner and server discovery application that aggregate network flows to generate a relevant dataset for our framework.

CHAPTER 5

NETWORK VISIBILITY THROUGH NETWORK FLOWS

1. Introduction

We studied in Chapter 3 how network flows could be used to assess the required size of a honeynet. We learned from this study that network flows have a great potential to improve the understanding of attacks collected by honeypots. In order to better integrate network flows in our honeynet architecture, we designed an algorithm to automatically extract scanner and server information. This information can not only contribute to better configure honeypots but can also help tracking attackers within the organization network.

The goal of this chapter is to introduce the server and scanner detection algorithm. We will study in the next chapter how we can combine this information with honeypot data. Figure 20 indicates which components of our framework are involved in this chapter.

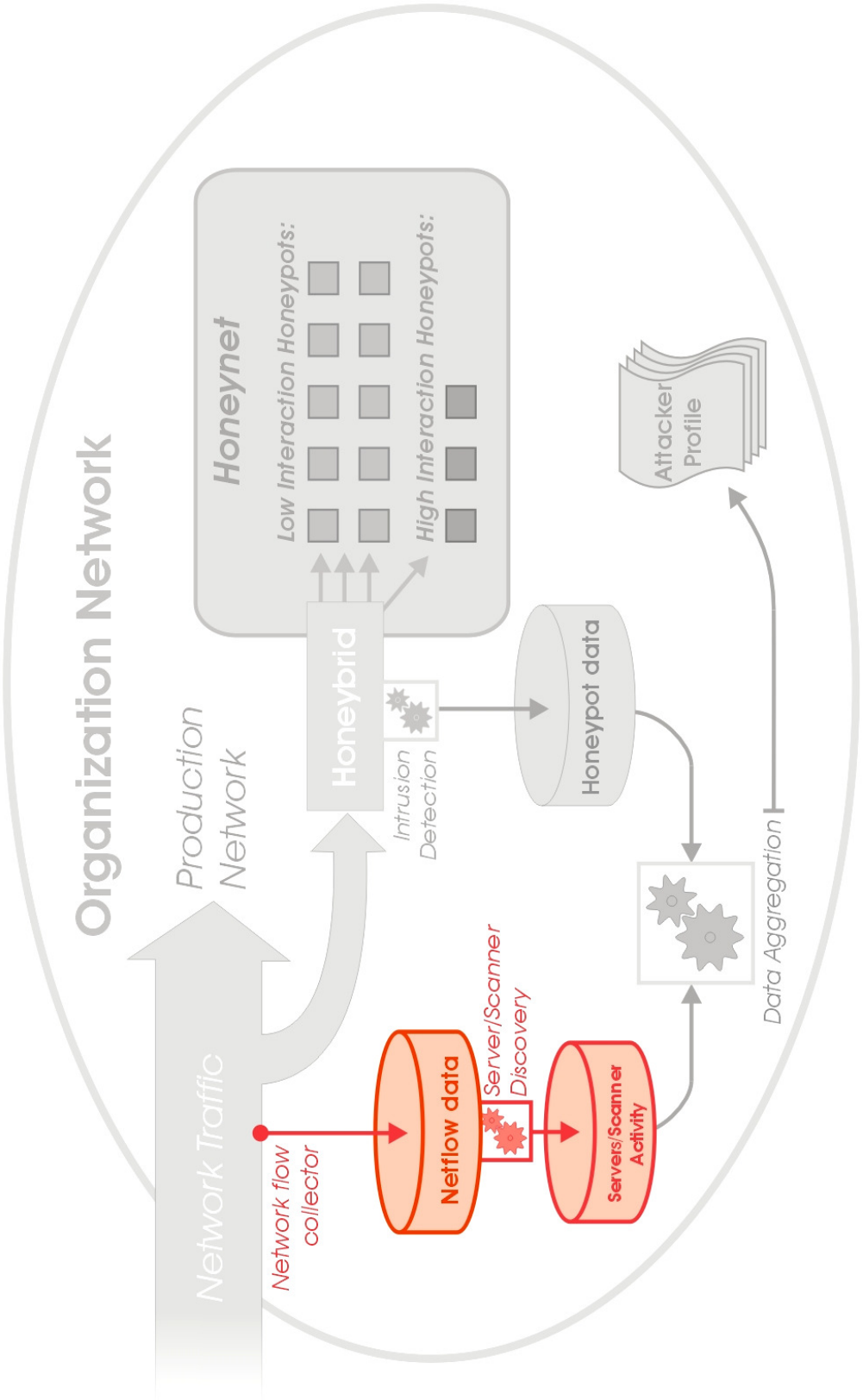

Figure 20: Overview of the components of our framework involved in the scanner and server discovery application

This chapter is organized as follow: background about service discovery is provided in Section 2. Our approach is presented in Section 3. Then the architecture of our solution is described in Section 4. The evaluation on a large campus network is detailed in Section 5. Section 5 also includes a discussion about the limitations of our approach, such as flow sampling, and details future work. Section 6 summarizes the study.

2. Related Work

Learning about which servers are deployed in the organization network and who is scanning the network falls in the research area of network visibility. Existing solutions to gain network visibility are divided into two categories: active and passive techniques.

Active sensors involve sending network probes to a set of targets to check for the presence of a listening service. Passive sensors extract information about services from network sniffing devices. The advantage of active solutions is accuracy and completeness. However, the drawbacks of active techniques are 1) they provide only a snapshot of the network in time, 2) they cannot detect services protected by firewalls, 3) they are intrusive and not scalable, and 4) aggressive scanning may also cause system and network disruptions or outages. On the other hand, passive solutions offer a continuous view of the network, their results are not impacted by firewalls and they are completely nonintrusive for the network. The main disadvantage of the passive approach is that it detects only active services. Unused services for which there is no incoming traffic cannot be discovered. Moreover, some services which are rarely used will take days or even weeks to be detected by a passive technique. [87] and [12] provides extensive results about the pros and cons of active scanning and passive detection. They show that the two techniques can be efficiently combined to mitigate each other's disadvantages.

On large networks, the role of active scanning is limited, supplanted by the more scalable passive approach. Well known commercial solutions include Cisco MARS [21], Real-Time Network Awareness (RNA [76]) from Sourcefire and a Passive Vulnerability Scanner (PVS [81]) from Tenable. The limitation of these solutions is that they often use payload information. Storing and analyzing packet payloads is very expensive [52], which makes such solutions less scalable than flow based architectures.

To our knowledge, the only existing tool that was built to merge unidirectional flows into connection-oriented bidirectional flows is *rwmatch* from SiLK [36]. But this tool has a single functionality (merging flows) and was not built to accurately classify end points like we wanted to.

3. Approach

This chapter focuses on the use of passive service discovery and historical comparison to assess the population of rogue servers and compromised computers. Our motivation is to provide a simple yet effective method to continuously and accurately detect the entire population of servers of a given network, from which network and security administrators can validate legitimate services and be alerted when suspicious services appear.

Our solution is built on top of Netflow from Cisco [22]. The design choice of using Netflow is motivated by three factors:

1. a wide majority of networks are already instrumented with Netflow;
2. unlike active probing, information gathered by Netflow is continuously updated;
3. Netflow is scalable and non-intrusive.

The problem with Netflow is that it was primary conceived for accountability and not for security. This translates into three main limitations:

- Netflow provides only header information. As a result, servers detected from Netflow data can be categorized at layer 4 (transport) but not at layer 7 (application). This means that applications listening to non-standard ports cannot be recognized at the application level. It is important to note that applications increasingly use encrypted protocols, so even having access to payload data would not help categorizing them;
- Netflow collects unidirectional flows. Consequently, information about the orientation of network connections is not collected. This means differentiating a server from a client based solely on unidirectional flows requires heuristics.
- Traffic that is confined to a local area network segment is typically not collected as it does not cross a Netflow-instrumented device such as a router.

The first listed limitation on identifying applications using Netflow is the focus of ongoing research [46] and is out of the scope of this chapter. This study addresses specifically the second issue regarding the limitation of unidirectional flows. More precisely, the main contributions of this study are:

1. To provide an innovative and accurate method to reassemble unidirectional flows into connection-oriented bidirectional flows, and therefore to leverage Netflow into a security monitoring solution;

2. To provide an implementation of this method that is built with the goals of high efficiency and simplicity (this implementation is under review to be released as an open source application);

3. To evaluate this implementation on a campus network of about 40,000 computers.

4. Architecture

The architecture of our Netflow-based server discovery tool is made of a back end module designed to process newly received flow files and to extract server-related information. This section details how this module works.

4.1. Netflow concepts

Netflow is a network protocol developed by Cisco and implemented in most routers to collect traffic information using packet headers. The two main versions currently in use are Netflow version 5 and version 9 [22]. Netflow is proprietary but an industry standard called IPFIX [48] based on Netflow version 9 is soon to be released.

The concept behind network flows is to collect summarized information about network traffic by grouping packets that share similar source and destination information. More precisely, a network flow is defined as a unidirectional sequence of packets sharing all of the following five parameters:

- *source IP address*;
- *source port* (for TCP or UDP, 0 for other protocols);
- *IP protocol*;

- *destination IP* address;
- *destination port* (for TCP or UDP, 0 for other protocols).

For each packet received on one of its Netflow-enabled interfaces, a router will try to find in its cache an existing flow that shares the five parameter values listed above from the packet's headers. If no match is found, a new flow record is created. Otherwise, an existing flow record is updated. A netflow record carries a wide variety of network-related information including: timestamp of the first packets received, duration, total number of packets and bytes, input and output interfaces, IP address of the next hop, source and destination IP masks and cumulative TCP flags in the case of TCP flows.

After being created and updated, a network flow has to be expired and exported. There are three possible expiration rules:

- For TCP flows, a valid sequence of FIN packets is observed, or a RESET packet is received;
- No new packet has been received and the flow has not been updated since a given period of time (usually 5 minutes);
- The flow reaches a maximum age limit (usually 15 minutes).

Expired network flows are exported in batch by routers toward a Netflow collector using the UDP protocol. The role of a Netflow collector is to receive and store Netflow feeds sent by routers. We used the Nfsen/Nfdump framework [38] to achieve this task.

Nfsen is an application designed to 1) configure and display the Netflow data collection, and 2) start and stop the nfcapd daemon which is in charge of storing Netflow records into compressed binary files at a regular interval (5 minutes by default). The

nfdump tool can then be used to convert these binary files into human-readable text files. Nfdump can also sort, aggregate and filter flows using the tcpdump filtering syntax [80].

4.2. Backend script and the challenge of flow timing

The backend modules include a Perl script (*backend.pl*) that runs every 5 minutes on the last flow file received and stored by the Nfsen collector. A handler automatically pipes the output of nfdump into the *backend.pl* script using the following command:

```
$ nfdump –o pipe –m –r <last flow file> | ./backend.pl > results.output
```

The role of the *backend.pl* script is to extract server-related information from a given flow file. This process is based on the difficult task of merging unidirectional flows into connection-oriented bidirectional flows. The specifics of this process are explained in the next section but first it is important to understand the mechanism behind it and why it is a challenge.

Unidirectional flows are an aggregate of packets that track from a specific source toward a specific destination during a given period of time. Therefore, a network session between a client and a server will consist of two distinct unidirectional flows: a first flow of packets sent by the client, and a second flow of packets sent by the server. The following example illustrates a network session between the client 10.0.0.1 and the web server 10.0.0.2 listening on port 80:

```
timestamp               source IP:port    dest. IP:port           Proto.
  Packets Flags
2009-01-01 12:34:56.789       10.0.0.1:30323    10.0.0.2:80       TCP
  12      .SAP..
2009-01-01 12:34:56.791       10.0.0.2:80       10.0.0.1:30323    TCP
  11      .SAP..
```

The first flow is called a *request flow* and the second flow is called a *reply flow*. It is relatively easy to merge a request and a reply flow by matching the source and the

destination information. The bidirectional flows that one could extract from the previous example would be:

```
timestamp                    src IP:port       dest. IP:port Proto Pkts
2009-01-01 12:34:56.789      10.0.0.1:30323 → 10.0.0.2:80  TCP   23
```

A direction arrow has been added to indicate which one of the source and destination is the client and which one is the server. Finding this direction in our example is simple because a request occurs always before a reply and according to the timestamp values, we can clearly see that the first packet of the request flow was received by the router 2 milliseconds before the first packet of the reply flow.

A problem arises when the timestamps of a request and a reply are identical. This problem occurs frequently (on average 20% of the times according to our empirical results) because of the relatively low time resolution of netflow records which is the millisecond. We will see in Section 4.2 that the timestamp of a request flow can even be reported by a router few milliseconds *after* the timestamp of the related reply flow. The next section details the different heuristics we developed to address this challenge and accurately detect clients and servers from Netflow records.

4.3. Heuristics

We defined six heuristics to try to correctly merge connection-related flow records. These heuristics were developed to cover a variety of intuitions gathered from network experts. The goal is to evaluate each of them using empirical data to find the best ones, individually or in combinations. For each bidirectional flow processed, a default orientation is selected from the order by which flows were read by the backend script. Then each heuristic is evaluated and can have 3 possible outcomes:

- the heuristic can be in favor of keeping the current direction of the flow;
- the heuristic can be in favor of reversing the direction of the flow;
- the heuristic cannot decide.

These six heuristics are:

- *Flow timing (H.0)*: this heuristic consists in trusting the timestamps of the flows to decide which one is the request and which one is the reply. If both the request and the reply flows have the same timestamp, then this heuristic cannot make a decision.
- *Port number (H.1)*: this heuristic states that servers usually have a lower port number than clients. Therefore when two unidirectional flows are merged, according to this heuristic, the direction of the bidirectional flow will go from the higher port number (source port) to the lower port number (destination port). If both the source and the destination ports are equal, then this heuristic will not make a decision.
- *Port number with threshold at 1024 (H.2)*: this heuristic is similar to the previous one but applies a threshold of 1024 to decide between client and server ports. The value of 1024 corresponds to the limit under which ports are considered privileged and designated for well-known services. This heuristic will not make a decision if both the source and the destination ports are above or below 1024.
- *Port number advertised in /etc/services (H.3)*: this heuristic uses the system file /etc/service that compiles assigned port numbers and registered port numbers [67]. These ports are mostly used by well-known services and should therefore

not be used by clients as source ports. If both the source and the destination port are not in /etc/services, then this heuristic will not make a decision.

- *Number of distinct ports related to a given port (H.4)*: this heuristic compares the numbers of distinct ports that are related to the source port and to the destination port of a bidirectional flow. The port that was involved in network connections with a greater variety of other port numbers is designated as the server port. This heuristic comes from the fact that ports on the client-side are often randomly selected. Therefore ports on the client-side of a connection are less likely to be used in other connections compared to ports on the server-side. If both the source and the destination ports are related to the same number of ports, then this heuristic cannot make a decision.

- *Number of distinct ports related to a given port with a threshold (H.5)*: this heuristic is identical to the previous one but applies a threshold of 5 as the minimum number of ports required to be server. The value of 5 was picked to be conservative. A full range of values will be evaluated. If both the source and the destination ports are related to fewer than 5 other ports, then this heuristic cannot make a decision.

The algorithm to apply these six heuristics on the flow dataset is described in the next section.

4.4. Algorithm

The script *backend.pl* reads flow records from the standard input and then outputs a list of clients and servers when the analysis process is complete. We configured the input

to cover a time window of 5 minutes of flow records, which is the default value used by Nfsen to store a new flow file.

The analysis process consists in trying to combine unidirectional flows into bidirectional flows using the protocol, the source and destination IP addresses and the source and destination ports. In the case of the TCP protocol, the number of packets per flow and the TCP flags are used to discriminate between valid and invalid connections. A valid TCP connection is made of two unidirectional flows with at least two packets per flow and the flags SYN and ACK enabled on each flow. Therefore the algorithm considers three types of flow:

- Unidirectional flows that cannot be paired;
- Valid bidirectional flows;
- Invalid bidirectional flows.

From these three types of flow we can extract four types of end points, where an end point is defined by a 3-tuple {IP address; protocol; port} according the following rules:

1. A *scanner* is the source end point of an unpaired unidirectional flow, or the source end point of an invalid bidirectional flow;
2. A *client* is the source end point of a valid bidirectional flow;
3. A *server* is the destination end point of a valid bidirectional flow;
4. An *invalid* is the destination end point of an unpaired unidirectional flow, or the destination end point of an invalid bidirectional flow.

It is important to note that since the source ports on the client-side of a connection are mostly random, the 3-tuple defining source end points is {source IP address; protocol;

destination port}, whereas the 3-tuple defining destination end points is {source IP address; protocol; source port}. The following example illustrates the three types of flow and the four types of end-points:

```
Id timestamp      source IP:port    dest. IP:port     Proto Pkts  Flags
#1 00:00:00.555   10.0.0.1:3006     10.0.0.2:445      TCP   1     .S....
#2 00:00:01.100   10.0.1.3:4000     10.0.1.4:22       TCP   64    .SAP..
#3 00:00:01.102   10.0.1.4:22       10.0.1.3:4000     TCP   65    .SAP..
#4 00:00:20.000   10.0.2.5:21560    10.0.2.6:8080     TCP   1     .S....
#5 00:00:20.001   10.0.2.6:8080     10.0.2.5:21560    TCP   1     ....R.
```

From these set of 5 flows, the algorithm will extract the following information:

- Flow #1 is a unidirectional flow that cannot be paired with any other unidirectional flow.
    - The source end point {10.0.0.1; TCP; 445} is labeled as *scanner*;
    - The destination end point {10.0.0.2; TCP; 445} is labeled as *invalid*;
- Flows #2 and #3 are part of a valid bidirectional flow;
    - The source end point {10.0.1.3; TCP; 22} is labeled as *client*;
    - The destination end point {10.0.1.4; TCP; 22} is labeled as *server*;
- Flows #4 and #5 are part of an invalid bidirectional flow;
    - The source end point {10.0.2.5; TCP; 8080} is labeled as *scanner*;
    - The destination end point {10.0.2.6; TCP; 8080} is labeled as *invalid*;

Note that the label *scanner* designates end points that generate suspicious traffic activity, which means requests to non-existent or filtered services. Such traffic can be generated because of scanning activity or because of misconfiguration.

The algorithm used by *backend.pl* to analyze flows is the following:

```
 Loop from standard input (one flow record per line piped from Nfdump) {
   Parse line to extract the values of each field of the flow record
   Exclude if the flow is not TCP or not UDP
   Exclude if the source or the dest. IP does not belong to a defined
internal subnet
   Create or update the flow record into memory using a hash LINKS
indexed by the key:
    {source IP; source port; protocol; dest. IP; dest. port}
 }
 Loop on flows from the hash LINKS {
   Search for a flow in the hash LINKS with the mirrored key:
    {dest. IP; dest. port; protocol; source IP; source port}
   If a mirror flow is found {
       If flow and mirror flow are valid {
             Label flow and mirror flow as “Valid Bidirectional Flow”
       } else {
             Label flow and mirror flow as “Invalid Bidirectional Flow”
       }
       For each heuristic {
             Apply the heuristic to decide on the direction of the flow
       }
       For each of the two possible flow directions {
             Create or update end points into memory using a second hash
NODES
              indexed by: {IP address; protocol; port; supporting
heuristics}
             According to the direction and the validity of the flow,
label
              each end points as Scanner or Client or Server or Invalid
       }
   } else {
       Label flow as “Unidirectional Flow”
       Create or update source and destination end points into memory
using the
        second hash NODES indexed by: {IP address; protocol; port}
       Label the source end point as Scanner and the dest. end point as
Invalid
   }
 }
 Loop on end points from the second hash NODES {
   Write information about the end point to standard output
 }
```

4.5. Data format

Every 5 minutes, the script *backend.pl* outputs information about end points detected from the flow file provided in input. This information is stored in a new text file and is organized with one end point per line. Each line carries the following fields:

1. End point identification: {IP address; protocol; source or destination port}
2. Location of the end point: *Internal* or *External* (based on the list of subnets that defines the organization network and provided in the configuration file of *backend.pl*)
3. Type of end point: *Scanner* or *Client* or *Server* or *Invalid*
4. Statistics about the end point: number of related flows, number of packets and number of bytes
5. List of supporting heuristics

End points which have more than one type or which were supported by different combinations of heuristics are spread on multiple lines.

5. Evaluation

The goal of the evaluation is to use empirical data to determine the efficiency of the different heuristics to passively identify servers using Netflow. The efficiency was measured using an active service discovery script running Nmap. Heuristics were evaluated individually and in combinations. Results are discussed according to different network parameters including: timing of request and reply flows, number of flows and number of hosts and ports related to servers detected.

5.1. Presentation of the dataset:

Data were collected over 48 hours at the border of a campus network made of two /16 networks (131,072 distinct IP addresses) and hosting approximately 40,000 computers. Statistics collected by the script *backend.pl* are provided in Table 6. Values in Table 6 are averages calculated from the 576 output files written by the script every 5 minutes during

the 48 hours of data collection. Percentages are derived from the averages and therefore do not always sum up to 100%. For the purpose of the evaluation, the script was configured to report end points detected by any combination of heuristics. This means that statistics about end points in Table 6 include all end points detected by at least one heuristic.

Table 6: Average statistics for 5 minutes of flow processed by the script backend.pl

Processing time	170.47 seconds	
Flows analyzed	442,356	100.00%
Incoming flows	226,831	51.28%
Outoing flows	204,761	46.29%
TCP flows	271,608	61.40%
UDP flows	159,985	36.17%
ICMP flows	10,761	2.43%
Other flows	2	0.00%
Flows discarded	10,764	2.43%
Unique flows extracted	423,794	100.00%
Flows combined into bidirectional flows	191,315	45.14%
Unique end points extracted	449,452	100.00%
Scanner end points detected	20,269	4.51%
Client end points detected	157,441	35.03%
Server end points detected	99,226	22.08%
Invalid end points detected	4,423	0.98%

The processing time provides an idea of the scalability of the backend script. The backend script and the Nfsen framework ran on the same machine which is a dual processor (Intel Xeon 3.80GHz) with 2 GB of memory. The processing time had an average of 170.47 seconds and ranged from 76 seconds (with 203,522 flows processed) up to 296 seconds (with up to 877,801 flows processed). We note again that the script was configured for the purpose of the evaluation to record end points detected from all combinations of heuristics. In a production environment, only accurate combinations of

heuristics would be kept and so the great reduction of false positives compared to our evaluation configuration would decrease the processing time.

A total of 404,615 unique TCP servers inside the campus network were discovered over the 48 hours of data collection. These servers are used as the baseline for the evaluation.

5.2. Classifying results with Nmap

To evaluate the accuracy of the heuristics implemented in the backend script, we checked the status of each internal TCP server discovered by the backend script using an active probing module based on Nmap. Scans originated from a computer inside the campus and therefore were not recorded by the routers collecting Netflow at the border of the network. Three possible outcomes can be returned by Nmap:

- *Open*: if the target of the scan sent back a valid reply;
- *Closed*: if the target of the scan sent back an invalid reply (reset packet);
- *Unknown*: if the target of the scan did not send anything back.

An unknown status occurs either because the target is not a server, or because its access is restricted by a firewall, or because the server was transient and is no longer connected to the network. To have further information about the unknown status, each time a server did not reply we checked from the log files of the backend script if this server communicated with one or multiple end points. If only one end point was involved, we configured the active probing module to send an alternative probe to this end point, in order to check if it was not actually the server. If it replied to the probe, then

we knew that the heuristic responsible for this detection made a wrong decision. Therefore we collected three more outcomes:

- *Source open*: if the end point targeted by the alternative scan sent back a valid reply;
- *Source closed*: if the end point targeted by the alternative scan sent back an invalid reply (reset packet);
- *Source unknown*: if the end point targeted by the alternative scan did not reply.

We configured the active probing script not to scan the same target multiple times in less than 60 minutes. This means that for end points which were constantly detected as server by the backend script over the 48 hours of data collection, we could have up to 48 measurements. We received varying outcomes for only 4,027 end points (0.99%). To decide on the correctness of the detection, we picked the final outcome according to the following priority order: *open* > *closed* > *source closed* > *source open* > *unknown* > *source unknown*. This means that even a minority of *open* led to a final *open* outcome for the 48 hours. The reason behind this choice is that Nmap sent a probe up to 15 minutes after the passive detection of a server. Therefore transient servers that would be shutdown during the delay between the passive detection and the active probing could be incorrectly classified as false positives if the *open* outcome was not leader for the final outcome.

The active probing script was automatically launched on the last set of servers detected. Every 5 minutes, a new set of servers was logged by the backend script and the current active probing script was killed to allow a new instance to be launched. As a result, some servers could not be scanned and the status *timeout* was used to differentiate

them. It occurred for 26.14% of the servers. Several instances of the Nmap script were running concurrently (one per /24 networks). Each instance was then scanning the list of detected server sequentially. This way we managed to have a good tradeoff between reducing the volume of timeout while keeping a reasonable level of aggressiveness to prevent undesired disruption and to avoid possible outages.

Table 7 summarizes the different outcomes, their meaning for the passive detection accuracy and the number of unique end points collected during the 48 hours of data collection. We can see from Table 7 that 4 out of the 7 possible outcomes lead to a precise validation or invalidation of the passive detection accuracy. However the three outcomes *unknown*, *source unknown* and *timeout* are inconclusive.

It is important to understand that our backend script was configured for this evaluation to record detected end points from all combinations of heuristics. This means that for a given bidirectional flow, if heuristics H.0 and H.1 are in favor of reversing the direction of the flow, but heuristics H.2 to H.5 are in favor of keeping the current direction of the flow, then the backend script will accept both decisions and will output two possibilities for each end point. Such configuration leads always to one set of true positives and one other set of true negatives. This explains why the numbers of incorrectly detected end points in Table 7 are so important. In a production environment, the script would have been configured to output only one orientation for every bidirectional flows, and the number of false positives would have been greatly reduced. The purpose of this evaluation is to choose which heuristics to trust for better results.

Table 7: Classification of active scan results and related ground truth for the passive technique

Status from active scan	Possible scenarios	Passive Detection	Distinct end points
Open	Server exists	**Correct**	14,242
Closed	Server does not exist	**Incorrect**	137,407
Unknown	Server exists but is filtered Server exists but was transient Server does not exist	**Correct** **Correct** **Incorrect**	12
Source open	Client is actually a server	**Incorrect**	103,555
Source closed	Client is not a server	**Correct**	34,728
Source unknown	Client is a server but is filtered Client is a server but was transient Client is not a server	**Incorrect** **Incorrect** **Correct**	8,899
Timeout	Scan could not be run	-	105,772
Total			**404,615**

We note that of the 298,843 successfully scanned end points reported during the evaluation, a total of 239,329 end points appeared only once (80.08%). This high volume of one-time servers is likely to indicate a large number of false positives. To find out which heuristic or which combinations of heuristics led to accurate detection or false positives, we investigate in detail the results in the next two sections.

5.3. Results per heuristic

Table 8 provides an overview of the results per heuristic. The second and third column entitled *Undecided* and *Supported* indicate the number of end points for which heuristics could take a decision or not. We defined and labeled the heuristics in Section 3.1.3. As mentioned, heuristics do not always make a decision because, for example, the source and destination ports are above 1024 (H.2) or both ports are equal (H.1). Table 8 shows that heuristics H.2, H.3 and H.5 have almost a perfect detection score with at least 97.71% of correctly classified end points. Heuristic H.4 has a relatively good score with

66.92% of correctly classified end points. Heuristic H.1 shows an opposite result with only 17.91% of correctly classified end points. This low percentage strongly indicates that relying on the intuition that client port numbers are higher than server port numbers is incorrect.

Table 8: Overall accuracy results per heuristic

Heur.	Unique end points		Correct		Incorrect		unknown			Accuracy
	Undecided	*Supported*	*Open*	*Source Closed*	*Closed*	*Source Open*	*Unknown*	*Source unknown*	*Timeout*	
H.0	27,542	89,002	11,608	4,299	55,219	230	7,584	3,188	17,158	**22.29%**
H.1	13	343,868	12,284	31,311	93,786	106,088	144,172	10,389	110,575	**17.91%**
H.2	381,406	9,165	8,966	69	57	9	206	131	981	**99.27%**
H.3	381,678	8,964	8,775	69	51	4	194	125	957	**99.38%**
H.4	217,555	9,566	5,381	141	1,780	950	1,375	270	1,819	**66.92%**
H.5	301,236	2,833	2,675	18	51	12	128	55	374	**97.71%**

Table 8 provides some insight about the timing of the request and reply flows. We can see that heuristic H.0 could not decide for 27,542 end points because they came from bidirectional flows that had identical starting times. We also see that only 22.29% of the successfully scanned end points for which H.0 could decide on the orientation were incorrectly classified. To analyze how heuristics H.1 to H.5 could help make a decision with or against H.0, we divided the empirical results in three parts:

- End points created from flows that have identical request and reply timestamps;
- End points created from flows that have a request recorded *before* the reply;
- End points created from flows that have a request recorded *after* the reply.

Table 9 gives the detailed results for heuristic H.1 to H.5 according to these three groups of end points. Results indicate first that when timestamps are identical, heuristics H.2 to H.5 have excellent detection accuracies in providing a correct orientation to request and reply flows. The second conclusion from Table 9 is that heuristic H.1 is

responsible for the large majority of end points classified as servers against the timing convention of requests occurring before replies. From the low score of only 15.66% of correctly identified servers, we note that, decisions from heuristic H.1 are again mostly incorrect. On the other hand, Table 9 shows that heuristics H.2, H.3 and H.5 should be trusted even if the timing looks reversed. They offer at least 94.86% of correctness when requests are recorded after replies.

Table 9: Results detailed according to the timing of request and reply flows

		Identical timestamp	**Request before reply**	**Request after reply**
Total		**31,525**	**97,313**	**338,439**
H.1	*correct*	6,983	10,741	31,782
	incorrect	13,787	15,967	171,170
	%	***33.62%***	***40.22%***	***15.66%***
H.2	*correct*	5,113	8,217	818
	incorrect	21	19	30
	%	***99.59%***	***99.77%***	***96.46%***
H.3	*correct*	4,960	8,043	804
	incorrect	18	17	23
	%	***99.64%***	***99.79%***	***97.22%***
H.4	*correct*	3,815	3,811	886
	incorrect	420	268	2,047
	%	***90.08%***	***93.43%***	***30.21%***
H.5	*correct*	2,391	1,229	702
	incorrect	12	15	38
	%	***99.50%***	***98.79%***	***94.86%***

5.4. Results and flow parameters

We saw in the previous section that requests from clients could sometime have a timestamp *posterior* to replies from valid servers. To investigate further this issue and to get insight about flow timing and server detection, we plotted on Figure 21 the time difference between request and reply flows against the active probe result for the first half of the data collection (24 hours). Each dot on Figure 21 is a scan triggered by the correct or incorrect orientation of a bidirectional flow. Jittering on the Y axis has been applied to

have a better idea of the density of points for a given time difference. The range -10 to 10 seconds represents 99.95% of bidirectional flows recorded. 1,882 end points have been detected outside of the range -5 to 5 seconds, among which only 32 were valid servers. All the correctly classified servers with a request timestamp recorded after the reply timestamp have been detected thanks to heuristic H.2. This indicates again that H.2 should have the priority over H.0 to decide on the correct orientation of a bidirectional flow.

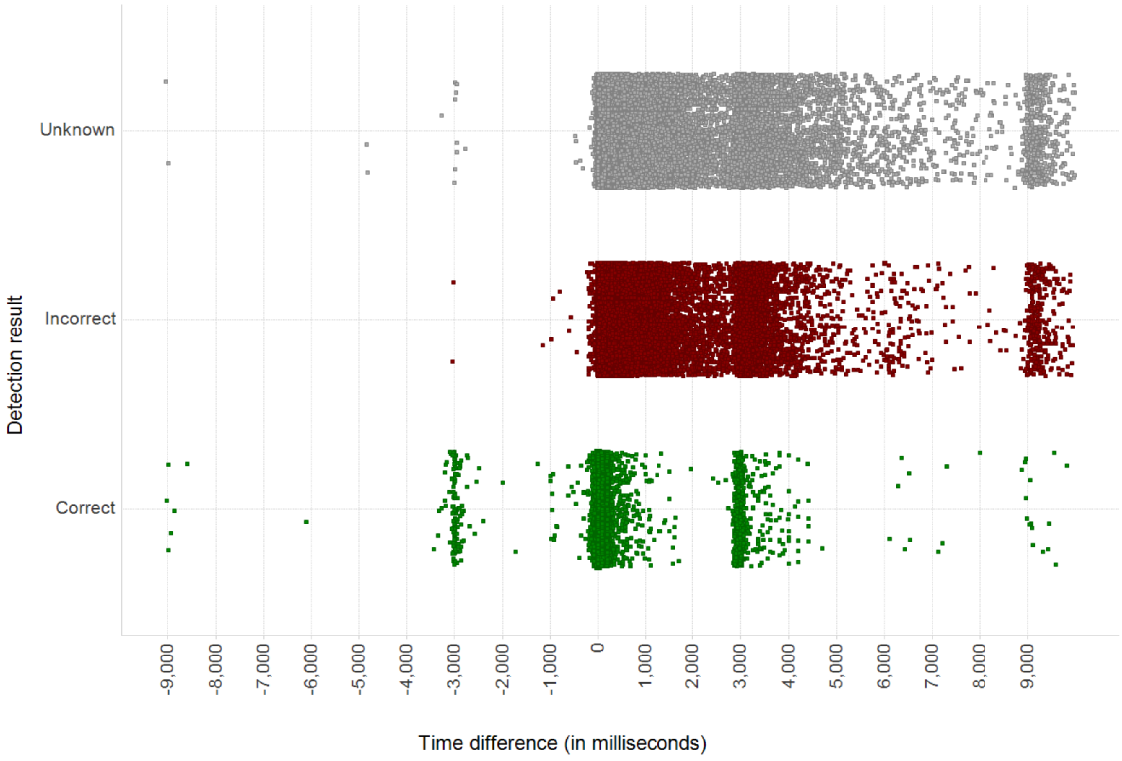

Figure 21: Time difference between request and reply flows categorized per detection results

We can clearly see some clusters on Figure 21 with the bulk of request and replies flows having less than 400 milliseconds of delay. 92.67% of the end points are from flows in the range -400 to 400 milliseconds. This range represents also 97.91% of the valid servers detected. This means that most of the valid servers have a short delay

between request and replies. We believe that this value is specific to the organization network but could be implemented as a parameterized heuristic in our architecture to improve the accuracy of the detection. We note also that we detect two clusters at -3 and 3 seconds, and two other clusters at -9 and 9 seconds. We are still investigating from a network topology point of view why these values.

Other parameters that we investigated to improve the heuristics are the number of flows per connection, the number of unique hosts and the number of unique ports related to a given end point. If a web server listening on port 80 is contacted by two clients, and each of these clients makes two connections using the random source ports 2000, 2001, 4000 and 4001, then the end point {web server; TCP; 80} will have the following parameter values:

- Number of flows: 8 (2 per connections);
- Number of related hosts: 2 (the two clients);
- Number of related ports: 4 (the four random client ports).

Figure 22 shows the distributions of these three network parameters according to the correctness of the passive detection.

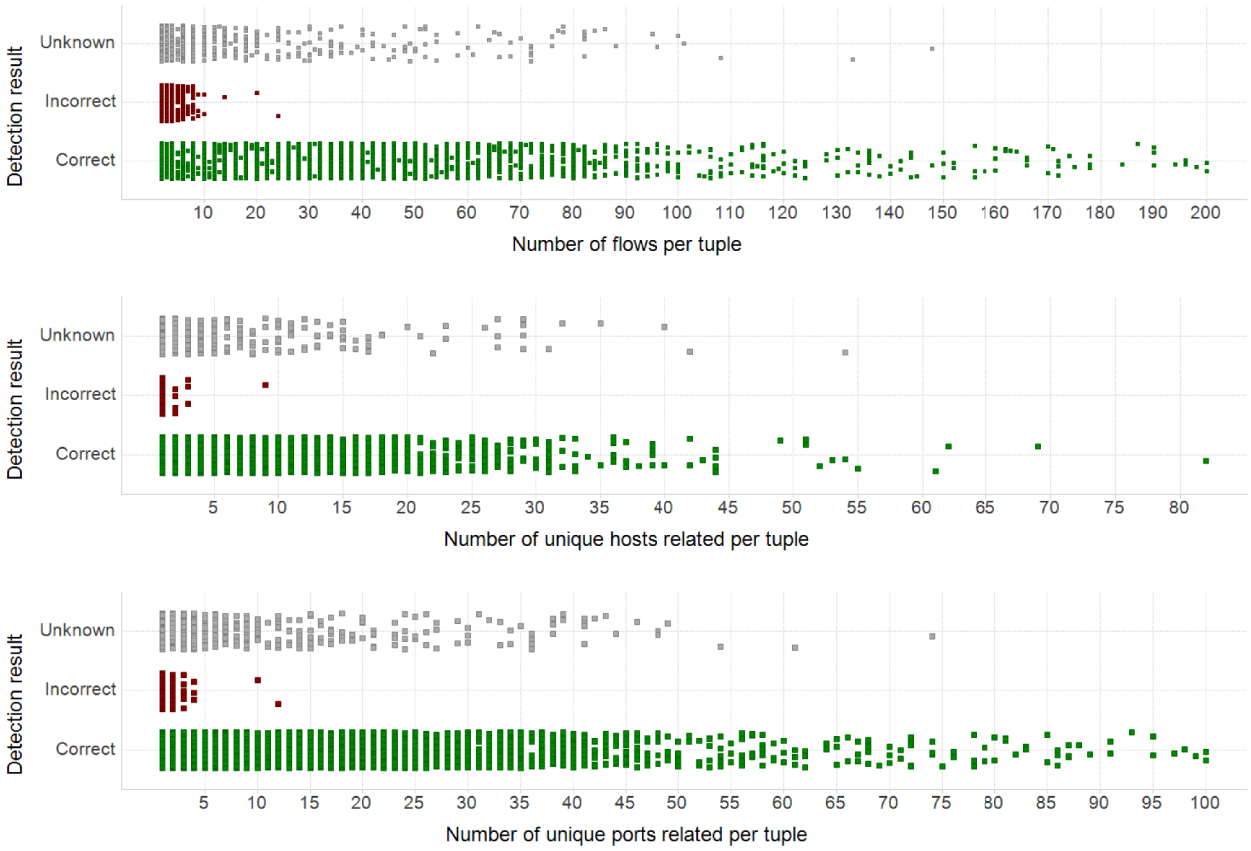

Figure 22: Distributions of (a) the number of flows, (b) the number of unique hosts and (c) the number of unique ports related to end points detected by the architecture.

We see from Figure 22 that almost all of the incorrectly classified end points have the following parameter values:

- A number of flows below 10;
- A population of related hosts lower or equal to 3;
- A number of related ports lower or equal to 4.

The number 4 of related ports explains why heuristic H.5 that used a threshold of 5 had such a good accuracy (98.01% of correctly detected servers).

The problem with these parameters is that once combined, they match only 350 of the 38,662 servers correctly detected. If we take them independently:

- 1,101 correctly classified servers (2.84%) have a number of flows above or equal to 10;
- 396 correctly identified servers (1.02%) have a number of related hosts above or equal to 4;
- 1,100 correctly classified servers (2.84%) have a number of related ports above or equal to 5.

As a result, these parameters offer a strong accuracy but are limited to a small population of servers.

5.5. Results per combination of heuristics

For each bidirectional flow, each heuristic can have 3 possible outcomes:

- Heuristic is in favor of reversing the direction of the flow (labeled with "+" in this section);
- Heuristic is in favor of keeping the direction of the flow (labeled with "-" in this section);
- Heuristic cannot make a decision (labeled with "=" in this section).

Therefore for our 6 heuristics, a total of $6^3 = 216$ combinations can be used. Over the 48 hours of evaluation, 83 combinations have been recorded in the dataset. Our goal is to rank each combination according to its accuracy in order to eliminate combinations that lead to incorrect detection.

As we mentioned in Section 4.2, whenever heuristics disagree on the orientation of a bidirectional flow, we configured our backend script in this evaluation to output both orientations.

On the 83 combinations that we collected, 39 had disagreeing heuristics and so we have accuracy results for both orientations of these combinations. Some of these combinations offer an obvious indication about which orientation lead to more accurate results. For example, the combination *(H0-)(H1+)(H2=)(H3=)(H4-)(H5-)* has the following accuracy results:

- Orientation kept: 97.22% of correctly classified servers;
- Orientation reversed: 0.18% of correctly classified servers.

We note that this result concords with the conclusion of Section 4.3 where heuristics were studied individually: *(H0-)(H1+)(H2=)(H3=)(H4-)(H5-)* means that H.1 is in favor of reversing the orientation of the flow (0.18% of correctness) and H.0, H.4 and H.5 are in favor of keeping the orientation of the flow (97.22% of correctness).

But for other combinations, the results are not so obvious. For instance, the combination *(H0-)(H1-)(H2=)(H3=)(H4+)(H5=)* has the following accuracy results:

- Orientation kept: 7.50% of correctly classified servers;
- Orientation reversed: 2.56% of correctly classified servers.

Such results indicate that our current set of heuristics is not sufficient to correctly decide on the orientation of flows matching the conditions of *(H0-)(H1-)(H2=)(H3=)(H4+)(H5=)*. Implementing and evaluating additional heuristics to improve the performance of our tool is part of the future work.

Table 10 and 11 provides the top 5 and worst 5 combinations evaluated during 48 hours. Results are sorted per accuracy and number of servers reported. The letter "*R*" in front of heuristics indicates that the orientation of the flow was reversed, and the letter

"*K*" indicates the orientation of the flow was kept (according to the original orientation at which the flow was received by the script).

Table 10: Top 5 combinations of heuristics having classified more than 1,000 tuples

Orient.	Heuristics	Total	Open	Closed	Timeout	Unk.	Src. op.	Src. cl.	Src. unk	Classified	Uncl.	Correct	Incorrect
K	H0=H1-H2-H3-H4=H5=	2908	2829	7	2	0	28	22	134	2838	184	99.93%	0.07%
K	H0-H1-H2-H3-H4=H5=	5639	5470	23	6	0	48	29	337	5499	414	99.89%	0.11%
R	H0+H1+H2+H3+H4=H5=	5596	5412	17	6	0	47	32	345	5435	424	99.89%	0.11%
R	H0+H1+H2+H3+H4+H5=	2097	2041	9	3	0	25	15	126	2053	166	99.85%	0.15%
R	H0=H1+H2+H3+H4+H5=	1197	1171	1	2	0	8	6	46	1174	60	99.83%	0.17%

Table 11: Worst 5 combinations of heuristics having classified more than 1,000 tuples

Orient.	Heuristics	Total	Open	Closed	Timeout	Unk.	Src. op.	Src. cl.	Src. unk	Classified	Uncl.	Correct	Incorrect
K	H0+H1-H2=H3=H4+H5+	57826	10	43	9770	24588	25248	766	24599	34411	50613	0.15%	99.85%
R	H0-H1+H2=H3=H4-H5-	58112	14	50	9895	24832	25562	809	24484	34791	50855	0.18%	99.82%
R	H0-H1+H2=H3=H4-H5=	54550	12	795	14892	19958	22148	1765	18971	35657	42884	2.26%	97.74%
K	H0+H1-H2=H3=H4+H5=	54798	14	838	14682	20179	22453	1822	19104	35713	43379	2.39%	97.61%
K	H0-H1+H2=H3=H4=H5=	23917	890	27	19587	0	566	539	3149	20504	4254	4.47%	95.53%

6. Discussion and future work

Heuristics were evaluated with 48 hours of network traffic collected on a campus network of 40,000 computers. The results of this evaluation show that:

- Relying on the timing of request and reply flows (heuristic H.0) is not accurate to identify clients and servers;
- Relying on ports numbers (heuristic H.1) when both source and destination ports are above 1024 is highly inaccurate (82% of false positives);
- However, relying on port numbers when one of the port is below 1024 (heuristics H.2), or advertised by /etc/services (heuristic H.3), or linked to at least five other ports (heuristic H.5) offers an almost perfect accuracy, with at least 97% of correctly classified servers;

A first limitation of our tool is that the current set of heuristics is not sufficient to correctly classify end points from Netflow. But adding new heuristics to our backend script is easy and we are currently investigating the following idea:

- *Number of detections during a given time window*: if the orientation of a bidirectional flow cannot be decided, we could keep an history of it over few hours and use this history to help future decision.
- *Port number*: instead of using /etc/services as a white list of server ports, we can build our own list based on empirical evaluations.

A second limitation that we did not yet investigate is flow sampling. Our tool was evaluated without any sampling and it would be interesting as part of the future work to study the effect of sampling on the detection accuracy of the different heuristics.

7. Summary

We introduced in this chapter a passive server discovery architecture based on network flows. We presented the design of this tool driven by the motivation to provide a simple and efficient solution to gain visibility in organization networks. Simplicity comes from the fact that we use only network flows. Efficiency comes from a set of heuristics that addresses the challenge of accurately combining unidirectional flows into connection-oriented flows. We evaluated these heuristics during 48 hours on the University of Maryland network which is made of 40,000 computers. The results of this evaluation were used to 1) show the scalability of our implementation, 2) discard inaccurate heuristics and approve accurate heuristics and 3) suggest network parameters to be included in additional heuristics.

The server and scanner dataset provided by this application can be directly used to assist the configuration of our honeynet and the data analysis of malicious traffic. The integration of the scanner and server discovery application into our malicious activity assessment framework is the topic of the next chapter.

CHAPTER 6

COMBINING HONEYPOT DATA AND NETWORK FLOWS

1. Introduction

The final phase of our project to integrate our hybrid honeypot architecture in the organization' set of security solutions is to combine it with network flows. We explained in the first chapter that honeypots data could give a detailed view of network threats but on a restricted IP space. By providing a high level but exhaustive view of the network traffic, network flows can adequately balance the limitations of honeypots. More precisely, network flows can assist our hybrid honeypot architecture in two important ways:

- To improve the configuration of honeypot sensors in order to extract more information from network threats;
- To extend the understanding of attacker's actions by providing a complete history of their communications in the organization network.

This chapter is dedicated to study these two issues in detail by presenting a complete architecture that integrates network flows and honeypots data. The components of our framework involved in this chapter are indicated on Figure 23.

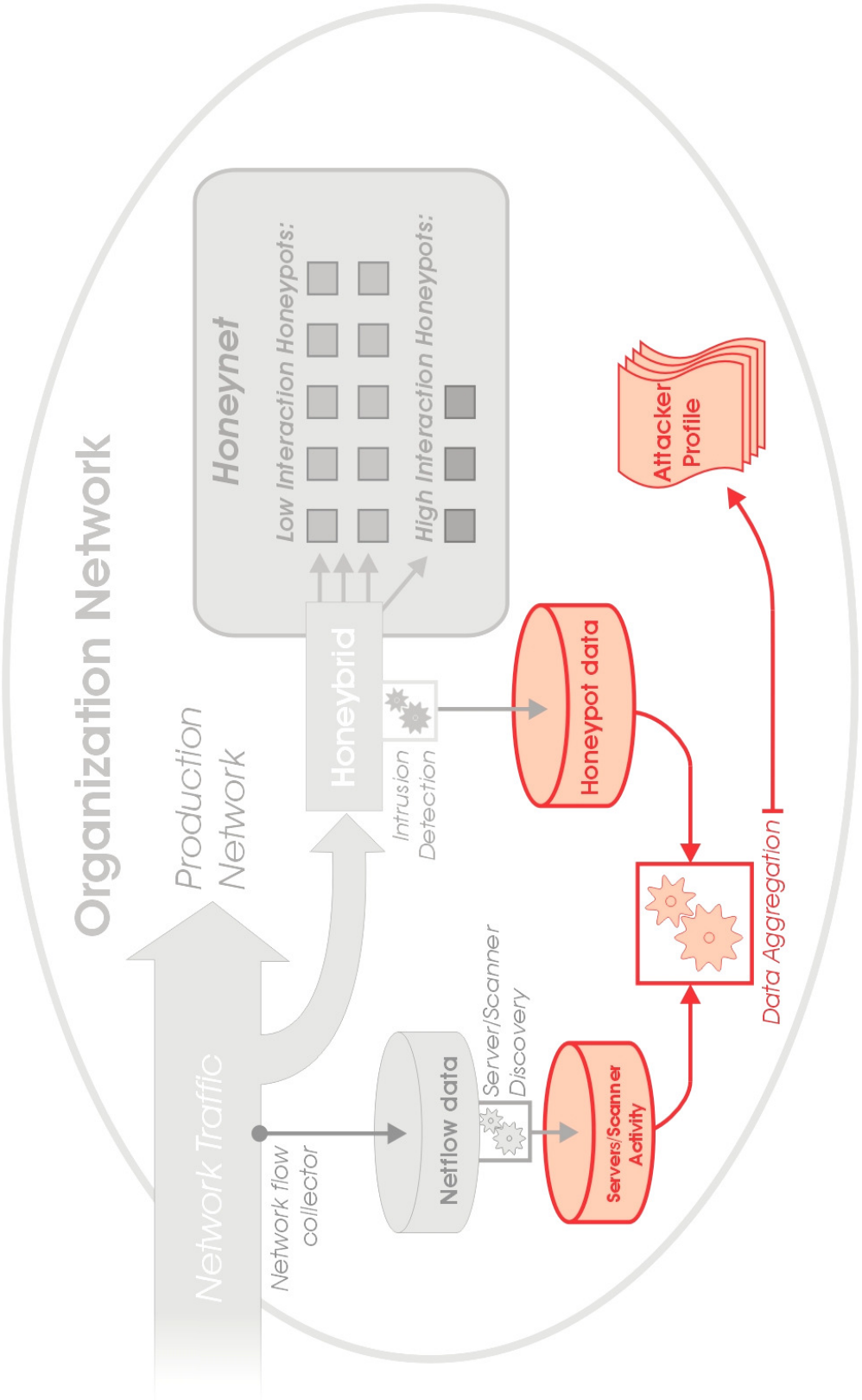

Figure 23: Overview of the components of our framework involved in this chapter

This chapter is organized as follow: in Section 2, we introduce an algorithm that uses flows to automatically configure honeypots. In Section 3, we detail a correlation and aggregation method that combines honeypot data and flows to profile and track attackers. We summarize this chapter in Section 4.

2. Assisted Honeypot Configuration

Attackers hunt for specific vulnerable network services and then attempt to exploit the ones they find. On the defense side, honeypots sensors try to be part of the pool of victims in order to gain intelligence about attacks. The main requirement for this process to work is for honeypots to offer network services that match the ones targeted by attackers. In case of mismatch, attackers and honeypots simply cannot communicate. As a result, the choice of network services deployed critically affects the efficiency of a honeypot architecture. We distinguish two possible motivations behind this choice:

- **Protecting the organization network**. In a production network, the goal of security analysts is to defeat attacks targeting the organization's assets. Consequently, honeypots deployed in such environment should offer to attackers the same type of services running in the production network.
- **Studying the latest attack trends**. In a research environment, the goal of researchers is to assess network threats and learn about unknown or recent vulnerabilities. Therefore, research honeypots should offer to attackers the type of services that are the most targeted.

We will now study how network flows can assist honeypot architectures for both of these motivations.

2.1. Protecting the Organization Network

In an ideal case, honeypots deployed in the organization network would be a perfect duplicate of the production network [8]. In reality, the size of the IP space and the resources allocated to the honeypot architecture force security analysts to build honeypot configurations that are a scale down version of the production network. This task is simple for small organizations with a single network administrator in charge of both the production machines and the honeypots. However, it reaches an impossible level of complexity in the case of large organizations where network administration is spread over different departments, each hosting potentially thousands of possible configurations of network services. This challenge can be precisely addressed by using the server discovery application that we introduced in Chapter 5.

The server discovery application passively returns the entire set of network services running in the production network and communicating with external hosts. From this dataset, we can have a precise assessment of the nature and the volume of services deployed in the organization network. We can directly use this information to automatically generate the network configuration of a honeypot architecture. [8] listed the two key properties of such method to be successful:

- **Proportional representation**. "*The possible vulnerable population on a honeynet should proportionately represent the vulnerable hosts on the network.*"
- **Individual host consistency**. "*To be effective in warding off possible fingerprinting efforts, we would like configurations for each host on the honeynet to be individually consistent.*"

We designed an algorithm that complies with these two properties by automatically generating the network settings of a given set of honeypots from the knowledge of the different services discovered in the production network. This algorithm works in four phases:

1. **Parsing input data**. The algorithm starts by reading the raw dataset of services discovered over a selected period of time by the server discovery application. Services are identified by a tuple {protocol; hosted port}. The ones hosted by honeypots are discarded. The ones hosted by production servers are grouped in combinations per IP address. The algorithm then ranks each combination of services according to their volume of hosting IP. For example, if 15 distinct IP addresses are detected having both the services {TCP; 22} and {TCP; 80} open, then the value "15" wil be used to rank the combination [{TCP; 22}, {TCP;80}] among all combinations. The most popular combinations, i.e., the ones associated to the greatest number of IP addresses, are ranked at the top of the list.

2. **Applying a proportionality factor**. The proportionality factor is calculated by dividing the number of unique IP addresses in the server discovery dataset by the size of the IP space dedicated to the honeypot architecture. This factor is then applied to the number of hosting IP for each combination of tuples. A threshold is used to keep only the most popular services and to discard services which are too infrequent (these services are saved for later review). The result is a list of combinations of services associated with a scaled down number of IP addresses.

3. **Generating a network configuration**. The list generated is then converted to a sequence of firewall "ACCEPT" rules. The set of IP addresses hosting each combination of tuples are selected randomly from the pool of honeypot IP addresses in order to distribute services in the IP space dedicated for the honeynet. This process is done iteratively starting with the most popular combination of services.

4. **Generating a Honeybrid configuration**. The list of ports used to build the firewall configuration is now used to generate automatically the configuration file of Honeybrid to deal with traffic on the ports of interest.

We ran the script on two week days of server discovery data collected at the University of Maryland. We declared a honeynet of five /24 subnets, which represents a total of 1,275 IP addresses. The script found 8,221 IP addresses hosting a total of 1,661 different combinations of services on the campus network. The top combination hosted by 1,504 different IP addresses was made of the single service {TCP; 22}. By dividing the count of 8,221 active IP addresses by 1,275 declared honeypot sensors, the script found a proportionality factor of 6.44. It then generated the firewall rules starting with the top combination and up to combinations hosted by at least 7 IP addresses. A sample of the processing output of the script is provided below:

```
# 1275 honeypots declared
# Generating the combinations of protocol/ports...
# Total ip: 8221, Total honeypot: 1275, Ratio: 6.44
# Distributing combinations to IP addresses in the honeynet...
#  Converted 1504 into 233 for combination "TCP-22"
#  Converted 710 into 110 for combination "TCP-80"
#  Converted 660 into 102 for combination "TCP-23 TCP-80"
#  Converted 423 into 65 for combination "TCP-8081"
#  Converted 343 into 53 for combination "TCP-3389"
#  Converted 338 into 52 for combination "TCP-443 TCP-80"
#  Converted 336 into 52 for combination "TCP-5900"
...
```

```
    #  Converted 7 into 1 for combination "TCP-53 UDP-53"
    #  Converted 7 into 1 for combination "TCP-3306 TCP-80"
    #  Converted 7 into 1 for combination "UDP-7000"
    #  Converted 7 into 1 for combination "UDP-6970"
    # Count is below 1, so we stop
    # 967 IP addresses were assigned in your honeypot (75.85%
covered)
    # Generating Honeybrid config per service...
    # 30 services defined for 5 honeypot networks with module sha1()
```

The script was able to assign the 70 combinations that were hosted by at least 7 IP addresses. On the remaining 1,591 combinations, 1,450 were hosted by only one IP address. As a result, the 70 assigned combinations represent 78.03% of the servers on campus (6,415 IP addresses out of 8,221). A similar calculation can be applied on the number of services: 30 services out of a 65,619 unique services could be assigned. But again, these 30 services are hosted by 78.03% of the servers.

The two configuration files generated by the script can then be automatically installed in iptables and Honeybrid. The result is a honeynet which is a scaled down representation of the production network, where each combination of services hosted by honeypots match the combination of services of a set of production servers.

2.2. Studying the Latest Attack Trends

The goal with research honeypots is to learn about the current and future trends of network threats. For example, catching a 0-day exploit is a proof of success for a honeypot-based experiment. Following the assumption that attackers use a cost-benefit approach to select the services they scan and exploit, the fact that a network service is scanned by a large variety of attackers is an important indication that attacks against this service offer a good return on investment. Researchers should therefore try to collect these attacks. The challenge is to configure honeypots with the network services that are

the most often scanned by attackers. Here again, the server discovery application introduced in Chapter 5 can greatly facilitate this process.

The idea is to build the network configuration of honeypots by using the dataset of scanners targeting the organization network. The algorithm to achieve this task works as follow:

- **Parsing input data**. The algorithm starts by reading the raw dataset of scanners detected over a selected period of time by the server discovery application. Services, which are identified by tuples {protocol; targeted port}, are individually extracted. Then the number of scanners is used to rank each service. Services which were scanned by the largest number of IP addresses are ranked at the top of the list.
- **Selecting the most popular services**. A user-defined threshold is then applied on the list of services to discard the ones scanned by only few attackers.
- **Generating a network configuration**. The list of services generated is then converted to a sequence of rules for Honeybrid. These rules use the equation of modules "SOURCE() or RANDOM()" in order to collect at least one attack sample from all attackers. The module SOURCE() guarantees that the first attempt of each attacker will be collected. Further attempts from the same attackers will be discarded to prevent attackers from fingerprinting the honeypot architecture. The module RANDOM() allows Honeybrid to collect additional samples of attacks without advertising the presence of honeypots. We note that to be even more realistic, the probability given in argument of RANDOM() could be

extracted from the popularity of the service in the organization network. This popularity can be provided once again by the server discovery application.

The main difference with the previous approach is the location of the filtering. In the case of production honeypots, the knowledge of servers was used to generate firewall rules. The Honeybrid gateway was then accepting everything that went through the filtering firewall. In the case of research honeypots, the knowledge of scanners is used to generate Honeybrid rules and to build the largest possible fishnet. The filtering is no longer on the firewall but on Honeybrid itself. The advantage of the first approach is to build a highly realistic honeynet. But if an attacker targets an IP on a service that was not declared open, the attack fails. On the other hand, the second approach has the advantage of building a giant fishnet, able to catch traffic from all attackers. The realistic part of the honeynet is weaker since it relies on the RANDOM() module to sample attacks from the same attacker toward the same service randomly.

We ran this script on 24 hours of scanner data collected by the server discovery application. A total of 2,069 unique services had been scanned on the campus network by 7,604 distinct IP addresses. We configured the script with a threshold of 10 to discard services that were scanned by 9 IP addresses or less. As a result, 41 services were extracted, representing 97.40% of the scanners (7,406 out of 7,604 scanning IP addresses). The top 5 most scanned services are provided below:

- Service UDP-53 scanned by 2,716 IP addresses;
- Service TCP-80 scanned by 2,037 IP addresses;
- Service TCP-25 scanned by 2,021 IP addresses;

- Service TCP-443 scanned by 272 IP addresses;
- Service UDP-389 scanned by 99 IP addresses;

The result is a new configuration file for Honeybrid defined by 41 entries.

2.3. Advantages and Limitations

We presented in this section two techniques to take advantages of network flows to remove the burden of manually handling the network configuration of honeypots. These two techniques cover the two possible motivations behind honeynet deployment: either to protect a production network, or to run research experiments. The contributions of our approach are:

- *Efficiency*: the method is fast and automated;
- *Accuracy*: the configuration is based on precise server and scanner detection;
- *Stealthiness*: the two properties of proportional representation and individual host consistency reduce the fingerprint of the honeynet architecture;
- *Dynamicity*: the script can be run on a daily basis to guarantee that the honeynet follows the latest attack or service trends.

This approach is a major step to solve the issue of honeynet configuration. However, it remains limited to the network configuration of honeypot sensors, and it does not provide a solution for host configuration. This is because our approach is based on network flows, which gives information at the network level (protocol and ports) but not at the application level. For example, when port TCP-80 is configured to be open on a given set of honeypots, we have no information to decide which web server to deploy (Apache or IIS?) and which application to exactly run. We note that a possible solution would be to

configure a dynamic low interaction responder such as [49] or [26] to dynamically infer the type of application from the attacker's requests. Then we would have to implement a new module in Honeybrid to adjust the configuration of the redirection from the output of the responders.

3. Building Attacker Profiles from Multiple Datasets

We studied in the previous section how network flows could help the deployment of honeypot sensors before starting the data collection. In this section, we investigate how network flows can assist security analysts after collecting data. The idea is to use flows as a tracking system to trace attackers in the organization network. Using this information, we can present to security analysts a complete history of communication for each attacker, including scanning activity, client activity and server activity. This information combined with the type of attack identified by the honeypot solution provides a comprehensive attacker profile.

3.1. Aggregation Scheme

The first step to be able to merge various datasets is to define an aggregation process. The concept of aggregation is to build layers of abstraction, in order to present to the end operators only the more relevant information. The amount of information reported by honeypot and network flow sensors can be quickly overwhelming, especially in a large organization network. This is why building an efficient aggregation scheme is critical. We define the following layers of abstraction, from bottom to top:

1. **Raw data**. The lowest layer is the smallest amount of output data provided by the different sensors. For example, in the case of network flow, a single line representing one flow would be seen as raw data.

2. **Event**. An event is the result of the first aggregation operation. Raw data can be combined in similar objects over a given period of time to generate events.

3. **Profile**. Profiles are the key part of the aggregation process, since they regroup events from multiple heterogeneous sources.

4. **Alert**. The top layer is the most abstract. Similarly to a distributed intrusion detection system (IDS), it consists of a set of alerts that a security analyst can quickly review to assess a situation. The alerts are often the starting point for more in-depth analysis. The alerts are triggered by a set of rules defined either by learning algorithms or human operators.

Figure 24 provides a representation of this aggregation scheme in the case of our architecture, where data from network flows collectors and honeypots are combined.

Raw data and events for both honeypots and network flows collectors have already been introduced in Chapters 4 and 5 through the Honeybrid and Server Discovery applications. The contribution of this chapter is the attacker profile. The alert layer is out of the scope of this study but will be discussed at the end of this section.

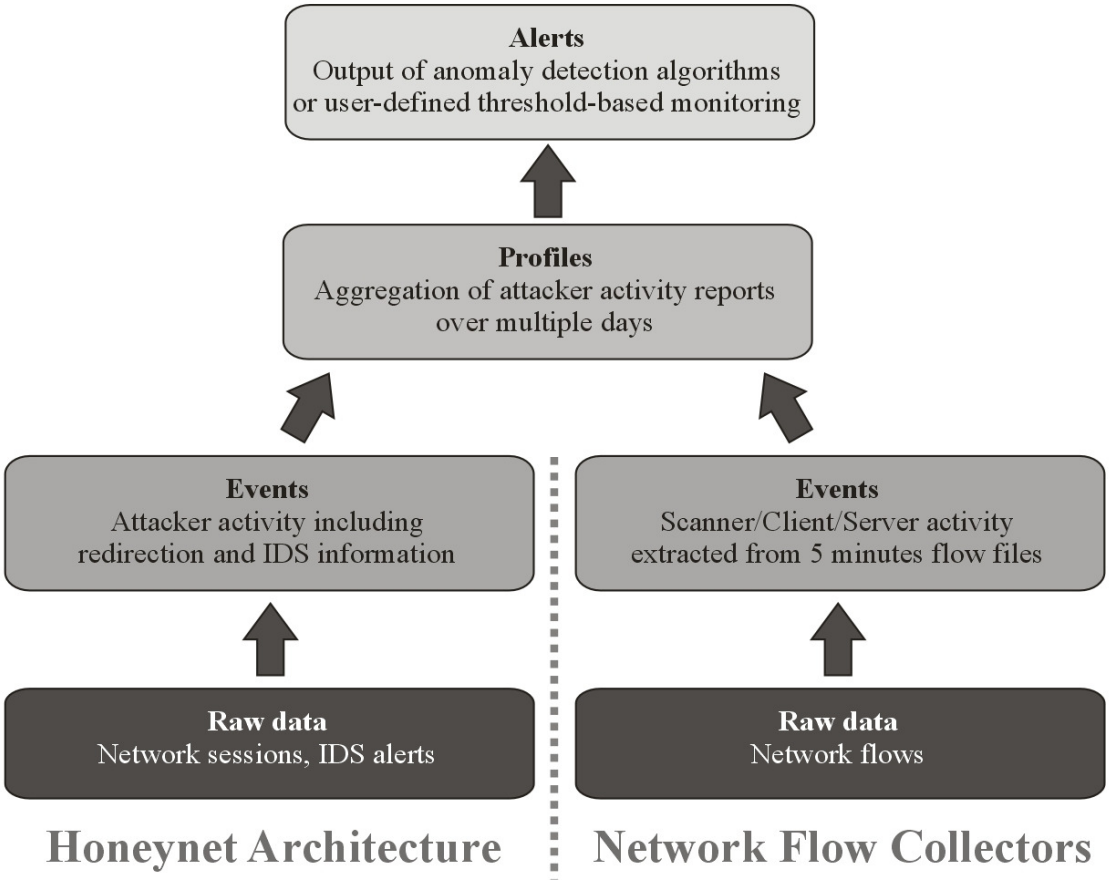

Figure 24: Overview of the aggregation scheme to regroup network flow and honeypot data

3.2. Aggregation Key

Events are automatically imported to a central database where a script is run automatically to build attacker profiles. A profile is a high-level representation of events collected through the organization network flows and through the honeypot architecture. The key used to regroup similar events in the same profile is the IP address. This approach has the following limitations:

- IP addresses can be spoofed by attackers;
- Attackers can bounce through multiple IP addresses before launching their attack;
- Attacks can be distributed over multiple IP addresses.

The first issue has a limited impact because 1) it concerns only single packet attacks, since packet replies sent back by victims to the spoofed sources will be lost; and 2) Internet service providers are more and more controlling outbound communications to automatically drop those coming from IP addresses out of their designated IP space [18]. Moreover, there is on-going research on the concept of backtracking to be able to traceback the source of spoofed packets [34].

The second issue is not a real limitation since the last IP address used by attackers and recorded in our log still reveals an infected system.

We partially solve the third issue by aggregating profiles not only by attackers but also by victims. Thus, attacks patterns involving multiples attackers will be recorded in the victim profiles. However, attackers who use different source IPs to target different victims might still be diluted in the dataset. A possible solution is to rely on advanced alerting rules at a higher aggregation level to be able to find them.

3.3. Profile Content

The different data fields in a profile were selected to provide a comprehensive summary of an attack. Their definition is generic enough to fit a large variety of attacks, but based on user needs, their aggregation thresholds can be adjusted to highlight more specific attack patterns.

A profile is divided in four sections: key, overview, flow report and honeynet report. These sections have the following fields:

- **Key**: IP address;
- **Overview**:

- Total time range (first date and last date of network activity collected);
- Total number of flows, packets and bytes collected;
- Total number of distinct hosts that communicated with this IP;

- **Flow report(s)**:
  - Key: port(s) or port-range(s) and an activity type (determined by the server discovery application):
    - Scanner;
    - Client;
    - Server;
    - Invalid;
  - Associated time range;
  - Associated number of flows, hosts, packets and bytes;

- **Honeynet report(s)**:
  - Key: port(s) or port-range(s) and an activity type:
    - scan: if no IDS signature is triggered besides scanning;
    - attack: if one or more IDS signature(s) is/are triggered;
  - Associated exploit detected (if any);
  - Associated time range;
  - Associated number of flows, hosts, packets and bytes;

Multiple flow and honeynet reports can be associated to a single key. Each report is an aggregated view of a set of events linked together by three elements:

1. A major key: the IP address of the profile;
2. A minor key: a port or a set of ports and a type;
3. A time range.

The first element is the primary key of the profile and obvious to use as a filter. The second and third elements are calculated using thresholds. The first threshold to be applied is an inactivity timeout between events. It is used as a minimum gap to separate events which are disjoint in time into separate reports. For example, if the three following events are collected:

- 10.0.0.1 is a scanner targeting port tcp/80 between 5:32pm and 5:37pm,
- 10.0.0.1 is a scanner targeting port tcp/80 at 5:57pm,
- 10.0.0.1 is a scanner targeting port tcp/80 between 11:03pm and 11:24pm,

and if we apply a timeout threshold of one hour, then events 1) and 2) would be grouped in a first flow report, since they occurred 20 minutes apart; but event 3) would be associated to a second flow report, since it occurred 5 hours and 6 minutes after event 2).

The second threshold applied concerns the ports. The idea is to produce as few reports as possible while keeping a maximum volume of information. When an IP address is detected having the same type of activity toward a list of different ports over a short period of time (defined by the first threshold), then we group all these events in the same report, and we display the port information either using an exhaustive list of all the ports involved, or using a count of ports and a port range. The problem is to decide the most

relevant solution between these two options: in the first case, we do not lose information, but we take the risk of confusing security analysts with too many displayed elements. In the second case, we lose information but we only have to keep record of two ports (the upper and lower boundaries of the range).

3.4. Aggregation Algorithm

Our aggregation algorithm fills the task of generating profiles from events. Events are added to our central database constantly. The aggregation algorithm loops over events and tries to group them using the keys and thresholds defined in the previous subsection. This process has two main constraints:

- Each new event added to the database has to be analyzed;
- Profiles and reports are constantly updated.

The first constraint is easily solved by using an auto-incremented id on each event and then keeping track of the last id analyzed. To address the second constraint, we divided the data storage of profiles in three tables:

1. A *master* table stores the overall information about the profile;
2. A *flow report* table stores all the reports created from network flow events;
3. A *honeynet report* table stores all the reports created from honeypot events.

Each of these three tables uses the same field “IP address” as a key. Overall information in the master tables are re-calculated after each update of the associated flow and honeynet reports. As such, the removal of old information is automatically taken care of in the master table when records are expired in the flow and honeynet report tables. We discuss further about the aging process in the next subsection.

The flow chart of the aggregation algorithm is described on Figure 25.

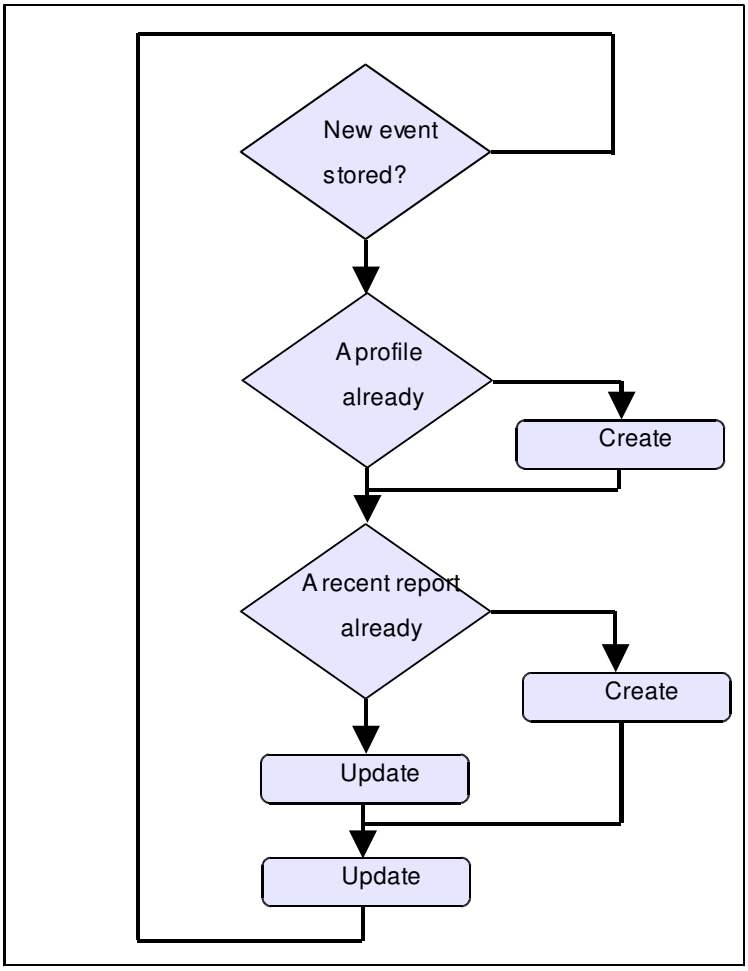

Figure 25: Flow chart of the aggregation algorithm

3.5. Automatic Aging and Backup Process

The volume of events, reports and profiles stored in the database can be very important for a large organization network. As a result, it is critical to develop an automated aging process in order to remove records which are too old, while not removing important information. Each record in the database has a field named "*last visited*", that hold the timestamp of the last time the record was queried by a user. The difference between the current day and the value of this field "*last visited*" is used to expire records. As a result, records which are queried often never expire (the system

assumes that these records are important). Then our aging process uses two threshold for records which have been either never queried or queried a long time ago: a *short* threshold (for example 7 days) is used to expire records with a small volume of network activity; and a *long* threshold (for example a 60 days) is used to expire records with a large volume of network activity. The volume of network activity is computed using both the number of flows collected in the organization network, and the number of attacks detected in the honeynet. Here, the assumption is that a trace of major disruptive activity should be kept during a longer period of time.

3.6. Case Study at the University of Maryland

3.6.1. Overview

We evaluated the aggregation architecture over a period of nine days at the University of Maryland. The campus network is made of two /16s which represent a total of 131,072 IP addresses. 1,275 of these IP were dedicated to the Honeybrid architecture. Honeybrid was configured with the Nepenthes program to handle low interaction traffic, and three high interaction honeypots running Windows 2003. During the nine days of the experiment, the server discovery application reported a total of 5,370,985 events. Honeybrid recorded a total of 772,218 network sessions, from which 34,495 carried an exploit detected by the Snort IDS.

The aggregation algorithm generated 180,768 profiles, 285,718 flow reports and 165,293 honeynet reports. 1,329 profiles generated had both flows and honeypot data associated. 15,362 profiles were for IP addresses internal to the campus network but only 9 profiles had honeypot data. These 9 profiles are likely to reveal compromised internal computers. We investigate further these profiles in the next subsection. On the 165,406 profiles of external IP addresses, 138,327 profiles had at least one honeynet report and

1,324 profiles had both flows and honeypot data associated. The low number of profiles with both flows and honeypot data can be explained by two reasons: 1) we deployed the server/scanner discovery application only on the main Internet providers of the University of Maryland, which is connected through two providers; and 2) we did not record client information in the flows report but only scanners and servers information. So if an attacker targets the honeynet and then only responsive servers in the production network, it will be recorded in the honeynet reports but not in the flow reports.

By aggregating reports per port we can quickly identify the top attack ports on the honeynet and the different threats targeting the organization network. Table 12 details the top 15 ports targeted on the honeynet and the related scanning traffic recorded by network flow collectors. Table 12 also indicate the number of different payloads collected by Honeybrid for the ports where the HASH module was installed. We see from the results that by comparing the number of attackers and scanners, we can divide ports in two categories: 1) ports such as UDP-389 or TCP-2967 with a low activity inside the organization network but a relatively important activity in the honeynet; and 2) ports such as TCP-22, TCP-80 or TCP-25 with a very large volume of traffic inside the organization network and a regular activity in the honeynet. This last set of ports is targetted by a large number of attackers because they are the most popular services hosted at the University of Maryland. These results are important to tune the honeynet architecture and to improve the data collection by focusing on the most important services.

Table 12: Overview of attacks collected by Honeybrid per port and related traffic at the University of Maryland

Ports	Honeynet		Network flows		Attack signatures collected by honeybrid
	Attackers	Flows	Scanners	Flows	
UDP-389	257	456	366	723	*no service deployed*
TCP-3268	217	314	227	309	*no service deployed*
TCP-2967	164	408	140	264	*no service deployed*
TCP-22	109	140	1,128	3,297	*no service deployed*
TCP-1433	97	216	119	184	10,191 brute force attack payloads
TCP-3072	47	131	38	67	*no service deployed*
TCP-80	47	272	8,562	20,298	1,181 attack payloads
TCP-8080	41	91	69	100	*no service deployed*
TCP-1024	39	101	43	77	*no service deployed*
TCP-23	19	37	255	287	*no service deployed*
TCP-3389	19	42	24	27	*no service deployed*
TCP-25	16	55	7,115	22,950	260 attack payloads
TCP-2968	14	17	91	118	*no service deployed*
UDP-2176	14	17	4	4	*no service deployed*
UDP-53	13	63	11,235	31,102	7,633 payloads

3.6.2. Web Attacks Collected

Figure 26 details the type of attacks collected by the `HASH` module of Honeybrid for the port TCP/80 (web servers) over 10 days. This report is useful to undertand the type of threat targeting the organization network, and to be able to make sure that applications deployed in the organization network are not vulnerable to these specific attack payloads.

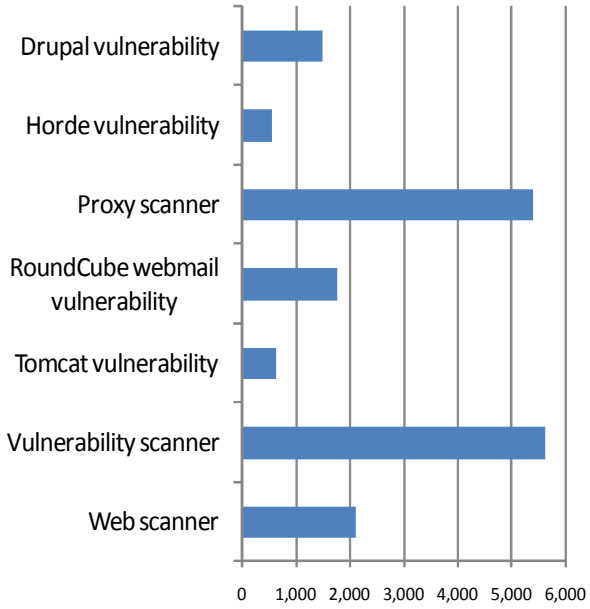

Figure 26: Volume of web attacks collected by Honeybrid and aggregated per type

3.6.3. Finding Internal Compromised Hosts

By filtering profiles to retrieve only those for internal IP addresses with at least one honeynet report, we got a list of nine IP addresses. We were surprised to discover that two of them were security scanners from the OIT department of the University. Five others had only a single hit in the honeynet and no flow report. The two other IP addresses could be immediately identified as compromised computers. Tables 13 provide the detailed information for the profile of one of these two addresses.

The profile represented on Table 13 reveals that *IP address A* belongs to a compromised computer. The slow scanning pattern of this computer (the maximum scanning rate is only six connection attempts per minute to the port TCP-445) makes it relatively undetectable by traditional security solutions. However, the honeynet succeed in capturing and precisely identifying the malicious activity carried by this computer. The aggregated profile offers a comprehensive timeline of events that allows a network operator to quickly identify the problem and take action (for example, by blocking the computer and contacting its owner). We notice that it took only 24 minutes for the honeypot sensors to receive attack probes from the second scanning campaign of IP address A, which started on Tuesday at 4:25am and targeted 1,216 IP during 4 hours.

Table 13: Aggregated activity profile for IP address A

Profile for IP Address A					
Overview:					
First seen:	Monday 4:45am				
Last seen:	Thursday 2:30pm				
Flows:	19,614				
Peers:	19,608				
Honeynet reports: (7)					
Start	*Duration*	*Type*	*Port*	*Flows*	*Peers*
Tues. 4:49am	2h	Scanning	TCP-445	64	64
Wed. 8:52am	1h	Scanning	TCP-445	7	7
Wed. 11:34am	1s	Scanning	TCP-445	2	2
Wed. 3:11pm	1s	Scanning	TCP-445	4	4
Wed. 3:11pm	31s	**Netbios Exploit**	TCP-139	1	1
Wed. 7:14pm	1h	Scanning	TCP-445	16	16
Thur. 11:38am	34min	Scanning	TCP-445	5	5
Flow reports: (9)					
Start	*Duration*	*Type*	*Port*	*Flows*	*Peers*
Mon. 4:45am	5min	Server	UDP-1027	2	1
Mon. 8:45am	5min	Scanner	TCP-445	25	25
Tues. 3:25am	5min	Server	UDP-1027	2	1
Tues. 4:25am	4h	Scanner	TCP-445	1,216	1,216
Tues. 1:40pm	5min	Server	UDP-1027	2	1
Tues. 7:25pm	5min	Server	UDP-1027	2	1
Tues. 9:10pm	2 days	Scanner	TCP-445	18,262	18,262
Thur. 5:10am	5min	Server	UDP-1027	2	1
Tues. 7:55am	5min	Server	UDP-1027	2	1

3.7. Limitation and Future Work

The first limitation of our aggregation framework is that it depends on the location of network flow collectors. In the case of the University of Maryland, network flows collectors are deployed at the border of the organization network. This means that all communications between internal hosts and external hosts are captured, but

communications between internal hosts only remains off the record. So internal attackers who target IP addresses inside the organization network might be detected by the honeynet but they will not leave any trace in the network flow data. This problem can be addressed by deploying additional network flow collectors between subnets of the organization network.

A second limitation is the lack of automated alerting system at the top layer of our aggregation process. Security analysts currently have to manually query for suspicious data. It is part of our future tasks to implement an alerting system that would be automatically triggered when a profile carries an attack pattern. In the case study that we mentioned previously, the attack signature could be: { "repeated scanning to port 445 detected in the flows" + "Netbios exploit detected in the honeynet" }.

4. Summary

In this chapter, we first show how network flows could be integrated with our honeynet architecture in order to generate automatically the network configuration of honeypots. Thanks to the server and scanner discovery application introduced in Chapter 5, we were able to build a program that takes the list of detected production servers to generate a honeynet configuration that accurately represent the production network at a smaller scale. We then presented a second program that takes the list of detected scanners to generate a honeynet configuration optimized to gather as much information as possible from the current population of attackers. These two programs contribute to solve the problem of configuring a large honeynet.

We then presented a second application of network flows to assist the analysis of honeypot data. By aggregating events from network flows and honeypot data into

summarized reports, we were able to build comprehensive attacker profiles that facilitate the work of security analysts to spot malicious activity and to understand attack patterns. We showed how we could detect two internal compromised computers after a few days of deployment at the University of Maryland.

CHAPTER 7

CONCLUSIONS

1. Summary

As our dependence on computers and network constantly increases, comprehensive network security is of tremendous importance. A first requirement to be able to better protect networks assets is to gain a detailed understanding of malicious threats. The concept of honeypot has been precisely invented to fill this task. In this dissertation, we presented a complete architecture to address the current limitations of honeypots deployed in the context of large organization networks. We started by defining what are honeypots and what types of attack they can capture. We then reviewed the three elements that researchers and security analysts need to define when deploying honeypots in a large organization network: a location, an architecture and a configuration. We then introduced different software solutions to help defining these three elements and to greatly reduce the costs associated with the deployment and the administration of honeypots. The cornerstone of our approach is the first open source implementation of a hybrid honeypot architecture that provides scalability and high level of interaction. We then integrated network flows into this architecture through an innovative passive server and scanner discovery application that 1) assists the automated configuration of large honeynet, and 2) extends the scope of honeypot data analysis by providing a comprehensive profile of network activity to track attackers in the organization network. Our final honeypot architecture marks a major step toward leveraging honeypot technologies into a powerful security solution for the organization network.

2. Insights

The first contribution of our work is to provide a detailed implementation of a hybrid honeypot architecture. We developed an advanced Decision Engine and Redirection Engine that offer to honeypot administrators a flexible and scalable solution to collect a large variety of network attacks. We introduced the notion of modular filtering to automatically separate interesting attack events from the noise of background traffic.

The second contribution of our work is to include network flows into our architecture to assist and extend the capabilities of honeypots. We detailed an algorithm to automatically handle the network configuration of honeypots without the need of human intervention. We then show how using network flows and honeypot data could improve the understanding of attacker's activity.

The third contribution of our work is to have deployed and tested our architecture in the large organization network of the University of Maryland. Thanks to the scalability of our honeynet, we are able to automatically collect malicious traffic from 1,275 IP addresses with small computer and human resources. Moreover, we show that the aggregation process we designed could efficiently reduce a large volume of data into a comprehensive set of attacker profiles.

3. Limitations

While the flexibility introduced by our advanced architecture allows broadening and expanding the spectrum of attack type collected, we still do not cover the analysis of targeted attacks. If we classify malicious activity in two categories: targeted attacks, involving skilled human attackers, targeting specific resources using stealthy reconnaissance techniques; and random attacks, involving scan-based threats such as

worms or botnets; then honeypots are mostly able to collect random attacks. The reason is that honeypots are deployed on unused IP spaces; therefore they can hardly be a target of choice for directed attacks. However, the impact of this limitation is reduced because the volume of directed attacks can often be neglected compared to the volume of random attacks. A possible solution is to deploy our Honeybrid architecture as a shadow honeypot in front of production servers instead of unused IP. Legitimate traffic would be sent to the server, but whenever malicious traffic is detected by the Decision Engine, the connection would be replayed by the Redirection Engine toward a honeypot for further analysis.

A second limitation of our architecture is the lack of automatic responder to handle traffic directed to services we did not yet deploy. Our automatic honeynet configuration program takes care of the network configuration of honeypots, but the administrators are still in charge of the host configuration of honeypots. The consequence is that a delay is induced between the discovery of a new vulnerable services and its deployment in the honeynet. To instrument Honeybrid with reverse engineering protocol capabilities in order to create low interaction responders on the fly would suppress this delay.

Finally, a limitation of honeypots that we did not address is the host monitoring functionality. We are using Argos based on Qemu to taint network packets in the virtual machine and detect intrusions, but we do not yet have a fully automated solution to save, analyze and clean our farm of high interaction honeypots.

4. Future Work

The limitations reviewed in the previous section provide some first indications on the future tasks required to improve our architecture: 1) using intelligent responders in place

of traditional low interaction scripts, and 2) implementing a virtual machine handler to detect compromised honeypots and re-image them automatically.

Another area that we did not explore is the user interface of our data aggregation framework. We built a first prototype that allows network operators to display attacker profiles and review the status of the different honeypots, but the functionalities are still limited. We see four possible improvements that could be implemented:

1. Making use of data visualization techniques to help human analysts getting a better understanding of attack processes;
2. Adding an alerting system to be able to define attack signatures from network flows and honeynet reports;
3. Allowing security analysts to share their findings through a collaborative user interface;
4. Adding data anonymization and data export functionalities to enable cross-organization data sharing for a better threat management.

Bibliography

[1] E. Alata, V. Nicomette, M. Kaâniche, M. Dacier, and M. Herrb, "Lessons learned from the deployment of a high-interaction honeypot," *edcc*, 2006, pp. 18-20.

[2] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling, "The nepenthes platform: An efficient approach to collect malware," *Lecture Notes in Computer Science*, vol. 4219, 2006, p. 165.

[3] M. Bailey, E. Cooke, T. Battles, and D. McPherson, "Tracking global threats with the Internet Motion Sensor," *32nd Meeting of the North American Network Operators Group*, 2004.

[4] M. Bailey, E. Cooke, F. Jahanian, A. Myrick, and S. Sinha, "Practical darknet measurement," *Ann Arbor*, vol. 1001, pp. 48109-2122.

[5] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson, "The Internet Motion Sensor: A distributed blackhole monitoring system," *Proceedings of the 12th Annual Network and Distributed System Security Symposium*, 2005.

[6] M. Bailey, E. Cooke, F. Jahanian, N. Provos, K. Rosaen, and D. Watson, "Data reduction for the scalable automated analysis of distributed darknet traffic," *Proceedings of the USENIX/ACM Internet Measurement Conference, New Orleans, LA*, 2005.

[7] M. Bailey, E. Cooke, D. Watson, F. Jahanian, and N. Provos, "A hybrid honeypot architecture for scalable network monitoring," *TechnicalReportCSE-TR-499-04, UniversityofMichigan*, 2004.

[8] M.D. Bailey, "A scalable hybrid network monitoring architecture for measuring, characterizing, and tracking internet threat dynamics," 2006.

[9] BaitnSwitch and Snort, *http://doc.emergingthreats.net/bin/view/Main/BaitnSwitch, 2008*.

[10] BaitnSwitch Project, *http://baitnswitch.sourceforge.net/, 2008*.

[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, 2003, pp. 164-177.

[12] G. Bartlett, J. Heidemann, and C. Papadopoulos, "Understanding passive and active service discovery," *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, ACM New York, NY, USA, 2007, pp. 57-70.

[13] F. Bellard, "QEMU, a fast and portable dynamic translator," *Proceedings of the USENIX Annual Technical Conference, FREENIX Track*, 2005, pp. 41–46.

[14] R. Berthier, J. Arjona, and M. Cukier, "Analyzing the process of installing rogue software," *International Conference on Dependable Systems and Networks*, to appear, 2009.

[15] R. Berthier and M. Cukier, "The Deployment of a Darknet on an Organization-Wide Network: An Empirical Analysis," *11th IEEE High Assurance Systems Engineering Symposium, 2008. HASE 2008*, 2008, pp. 59-68.

[16] R. Berthier and M. Cukier, "An evaluation of connection characteristics for separating network attacks," *International Journal*, vol. 4, 2009, pp. 110-124.

[17] R. Berthier, D. Korman, M. Cukier, M. Hiltunen, G. Vesonder, and D. Sheleheda, "On the Comparison of Network Attack Datasets: An Empirical Analysis," *11th IEEE High Assurance Systems Engineering Symposium, 2008. HASE 2008*, 2008, pp. 39-48.

[18] R. Beverly and S. Bauer, "The spoofer project: Inferring the extent of source address filtering on the Internet," *Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop table of contents*, USENIX Association Berkeley, CA, USA, 2005, pp. 8-8.

[19] G.G. Certification, "Enhancing IDS using, Tiny Honeypot."

[20] Chaosreader, *http://chaosreader.sourceforge.net, 2008*.

[21] Cisco MARS, *http://www.cisco.com/en/US/products/ps6241/, 2009*.

[22] B. Claise, "RFC 3954: Cisco systems NetFlow services export version 9," *Published by Internet Engineering Task Force (IETF). Internet Society (ISOC) RFC Editor. USA. oct*, 2004.

[23] E. Cooke, M. Bailey, Z.M. Mao, D. Watson, F. Jahanian, and D. McPherson, "Toward understanding distributed blackhole placement," *Proceedings of the 2004 ACM workshop on Rapid malcode*, ACM New York, NY, USA, 2004, pp. 54-64.

[24] E. Cooke, M. Bailey, Z.M. Mao, D. Watson, F. Jahanian, and D. McPherson, "Toward understanding distributed blackhole placement," *Proceedings of the 2004 ACM workshop on Rapid malcode*, ACM New York, NY, USA, 2004, pp. 54-64.

[25] W. Cui, J. Kannan, U.C. Berkeley, and H.J. Wang, "Discoverer: Automatic protocol reverse engineering from network traces."

[26] W. Cui, V. Paxson, and N. Weaver, "GQ: realizing a system to catch worms in a quarter million places," 2006.

[27] W. Cui, V. Paxson, N. Weaver, and R.H. Katz, "Protocol-independent adaptive replay of application dialog," *Proceedings of the*.

[28] M. Cukier, R. Berthier, S. Panjwani, and S. Tan, "A Statistical Analysis of Attack Data to Separate Attacks," *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, 2006, pp. 383-392.

[29] T. Cymru, "The darknet project," *Internet: http://www. cymru. com/Darknet*, 2004.

[30] R. Deraison, "Nessus," *The Nessus Project. February*, vol. 24, 2003.

[31] R. Dhamija, J.D. Tygar, and M. Hearst, "Why phishing works," *Proceedings of the SIGCHI conference on Human Factors in computing systems*, ACM New York, NY, USA, 2006, pp. 581-590.

[32] P. Diebold, A. Hess, and G. Schafer, "A honeypot architecture for detecting and analyzing unknown network attacks," *14th Kommunikation in Verteilten Systemen*, 2005.

[33] J. Dike, "User-mode linux," *Proceedings of the 5th annual conference on Linux Showcase & Conference-Volume 5*, USENIX Association Berkeley, CA, USA, 2001, pp. 2-2.

[34] A. Fadlallah, A. Serhrouchni, Y. Begriche, and F. Nait-Abdesselam, "A Hybrid Messaging-Based Scheme for IP Traceback," *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, 2008, pp. 1-6.

[35] N. Friess and J. Aycock, "Black Market Botnets," 2007.

[36] C. Gates, M. Collins, M. Duggan, A. Kompanek, and M. Thomas, "More NetFlow tools: For performance and security."

[37] D. Guo, M. Gahegan, D. Peuquet, and A. MacEachren, "Breaking down dimensionality: an effective feature selection method for high-dimensional clustering," *Workshop on Clustering High Dimensional Data and its Applications, the Third SIAM International Conference on Data Mining, May*, pp. 1-3.

[38] P. Haag, "Netflow Tools NfSen and NFDUMP," *18th Annual FIRST Conference*, 2006.

[39] L.T. Heberlein, G.V. Dias, K.N. Levitt, B. Mukherjee, J. Wood, and D. Wolber, "A network security monitor," *1990 IEEE Computer Society Symposium on Research in Security and Privacy, 1990. Proceedings.*, 1990, pp. 296-304.

[40] Honeynet Project, *http://www.honeynet.org/tools/sebek/, 2008*.

[41] Honeywall, *https://projects.honeynet.org/honeywall, 2008*.

[42] IBM X-Force 2008 Trend Statistics, *http://www.ibm.com/services/us/iss/xforce/midyearreport/xforce-midyear-report-2008.pdf*.

[43] X. Jiang and D. Xu, "Collapsar: A VM-based architecture for network attack detention center."

[44] John-the-ripper, *http://www.openwall.com/john, 2008*.

[45] J. Jung, V. Paxson, A.W. Berger, and H. Balakrishnan, "Fast portscan detection using sequential hypothesis testing," *2004 IEEE Symposium on Security and Privacy, 2004. Proceedings*, 2004, pp. 211-225.

[46] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: multilevel traffic classification in the dark," *ACM SIGCOMM Computer Communication Review*, vol. 35, 2005, pp. 229-240.

[47] C. Leckie and R. Kotagiri, "A probabilistic approach to detecting network scans," *2002 IEEE/IFIP Network Operations and Management Symposium, 2002. NOMS 2002*, 2002, pp. 359-372.

[48] S. Leinen, "RFC 3955: Evaluation of Candidate Protocols for IP Flow Information Export (IPFIX)," *Published by Internet Engineering Task Force (IETF). Internet Society (ISOC) RFC Editor. USA. out*, 2004.

[49] C. Leita and M. Dacier, "SGNET: a worldwide deployable framework to support the analysis of malware threat models," *Dependable Computing Conference, 2008. EDCC 2008. Seventh European*, 2008, pp. 99-109.

[50] C. Leita, K. Mermoud, and M. Dacier, "Scriptgen: an automated script generation tool for honeyd," *Computer Security Applications Conference, 21st Annual*, 2005, p. 12.

[51] Mech IRCBot, *http://www.energymech.net, 2008*.

[52] A. Moore, J. Hall, C. Kreibich, E. Harris, and I. Pratt, "Architecture of a network monitor," *Passive & Active Measurement Workshop 2003 (PAM2003)*, 2003.

[53] D. Moore, "Network telescopes: Observing small or distant security events," *Proceedings of the 11th USENIX security symposium*, 2002.

[54] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the slammer worm," *IEEE Security & Privacy*, vol. 1, 2003, pp. 33-39.

[55] D. Moore, C. Shannon, D.J. Brown, G.M. Voelker, and S. Savage, "Inferring Internet denial-of-service activity," *ACM Transactions on Computer Systems (TOCS)*, vol. 24, 2006, pp. 115-139.

[56] D. Moore, C. Shannon, G. Voelker, and S. Savage, "Network telescopes: Technical report," *CAIDA, April*, 2004.

[57] Netfilter, *http://www.netfilter.org, 2008*.

[58] Netflow, *http://www.cisco.com/go/netflow, 2008*.

[59] G. Portokalidis, A. Slowinska, and H. Bos, "Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation," *Proceedings of the 2006 EuroSys conference*, ACM New York, NY, USA, 2006, pp. 15-27.

[60] J. Postel, *RFC 791: Internet protocol*, September, 1981.

[61] F. Pouget, M. Dacier, and V.H. Pham, "Leurre. com: On the advantages of deploying a large scale distributed honeypot platform," *E-Crime and Computer Evidence Conference (ECCE 2005)*, 2005.

[62] N. Provos, "A virtual honeypot framework," *Proceedings of the 13th USENIX Security Symposium*, 2004, pp. 1–14.

[63] N. Provos and T. Holz, "Virtual honeypots: from botnet tracking to intrusion detection," 2007.

[64] PsyBNC, *http://www.psybnc.at, 2008*.

[65] D. Ramsbrock, R. Berthier, and M. Cukier, "Profiling attacker behavior following SSH compromises," *adm*, vol. 1, p. 0.58.

[66] F. Raynal, Y. Berthier, P. Biondi, and D. Kaminsky, "Honeypot forensics," *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*, 2004, pp. 22-29.

[67] J. Reynolds and J. Postel, "RFC1340: Assigned Numbers," *RFC Editor United States*, 1992.

[68] R.L. Richardson, *CSI Survey 2007: The 12th Annual Computer Crime and Security Survey*, Computer Security Institute, 2007.

[69] S. Robertson, E.V. Siegel, M. Miller, and S.J. Stolfo, "Surveillance detection in high bandwidth environments," *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, 2003.

[70] M. Roesch, "Snort–lightweight intrusion detection for networks."

[71] Sebek, *https://projects.honeynet.org/sebek, 2008*.

[72] C. Seifert and N.Z.H. Alliance, "Malicious SSH Login Attempts," *Adresse Internet: www. securityfocus. com/infocus/1876*.

[73] S. Small, J. Mason, F. Monrose, N. Provos, and A. Stubblefield, "To catch a predator: A natural language approach for eliciting malicious payloads."

[74] Snort Inline, *http://snort-inline.sourceforge.net, 2008*.

[75] D. Song, R. Malan, and R. Stone, "A snapshot of global Internet worm activity," *Arbor Networks, Tech. Rep*, 2001.

[76] SourceFire RNA, *http://www.sourcefire.com/products/3D/rna, 2009*.

[77] W.R. Stevens, *TCP/IP illustrated (vol. 1): the protocols*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1993.

[78] Strace, *http://strace.sf.net, 2008*.

[79] Syslog-ng, *http://www.balabit.com/network-security/syslog-ng/, 2008*.

[80] Tcpdump, *http://www.tcpdump.org, 2008*.

[81] Tenable PVS, *http://www.nessus.org/products/pvs/, 2009*.

[82] Virtual PC, *http://www.microsoft.com/windows/products/winfamily/virtualpc/default.mspx, 2008*.

[83] Virtualbox, *http://www.virtualbox.org, 2008*.

[84] Virustotal, *http://www.virustotal.com, 2008*.

[85] VMWare, *http://www.vmware.com, 2008*.

[86] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A.C. Snoeren, G.M. Voelker, and S. Savage, "Scalability, fidelity, and containment in the potemkin virtual honeyfarm," *ACM SIGOPS Operating Systems Review*, vol. 39, 2005, pp. 148-162.

[87] S. Webster, R. Lippmann, and M. Zissman, "Experience using active and passive mapping for network situational awareness," *Fifth IEEE International Symposium on Network Computing and Applications, 2006. NCA 2006*, pp. 19-26.

[88] V. Yegneswaran, P. Barford, and D. Plonka, "On the design and use of Internet sinks for network abuse monitoring," *Lecture notes in computer science*, 2004, pp. 146-165.