FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

BOTNET RESILIENT HONEYNETS

By

YAROSLAV KECHKIN

A Project submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Master of Science

Degree Awarded:
Fall Semester, 2015

Yaroslav Kechkin defended this project on November 19th, 2015.
The members of the supervisory committee were:

Professor

Mike Burmester

Committee Member 1

Xiuwen Liu

Committee Member 2

Committee Member

The Graduate School has verified and approved the above-named committee members, and certifies that the project has been approved in accordance with university requirements.

ACKNOWLEDGMENTS

I would like to thank the following individuals and organizations for their outstanding contributions to this research:

National Science Foundation, Scholarship for Service, Cybercorp, for the generous funding, mentoring, networking opportunities which have given me the opportunity to study computer science. Specifically my co-principal investigators, Mike Burmester and Xiuwen Liu

Dr. Mike Burmester, for mentoring and help with research and paper writing

Zach Yannes, for his LaTeX template and help

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

API - **A**pplication **P**rogram **I**nterface
AV - **A**nti-**V**irus
C2 - **C**ommand-and-**C**ontrol
DDoS - **D**istributed **D**enial-**o**f-**S**ervice
DNS - **D**omain **N**ame **S**ystem
ECPA - The **E**lectronic **C**communication **P**rivacy **A**ct
FTP - **F**ile **T**ransfer **P**rotocol
HTTP - **H**yper **T**ext **T**ransfer **P**rotocol
IDS - **I**ntrusion **D**etection **S**ystem
IIS - **I**nternet **I**nformation **S**ervices
IPS - **I**ntrusion **P**revention **S**ystem
IP - **I**nternet **P**rotocol
IRC - **I**nternet **R**elay **C**hat
P2P - **P**eer-to-**Peer**
PIN - **P**ersonal **I**dentification **N**umber
URL - **U**niform **R**esource **I**dentifier
VM - **V**irtual **M**achine

ABSTRACT

One of the greatest challenges the security community faces is lack of information on the enemy. Questions like who is the threat, why do they attack, how do they attack, what are their tools, and when will they strike again often cannot be answered. Traditionally, security professionals have learned about attackers by studying the tools which they use. When a system is compromised, security administrators will often find the attacker's tools left on the attacked system. A variety of assumptions are then made about the attackers based on these captured tools. As effective as this technique can be, there is still so much information that can be learned about attackers. Instead of learning only about the tools that attackers use, it makes sense to identify attackers, determine how well organized they are, and learn their methods. A honeynet is designed to attract attacks to itself, then contain and learn as much as possible about an attacker. A honeynet is usually designed to attract attackers and then contain the attack in itself, without letting the botnet spread to other nodes. With this, attackers explore new techniques in creating honeynet-aware botnets, that use well established protocols to communicate between bots and check if the malicious traffic is spreading or have been contained, thus identifying a honeynet. In this project, we seek ways to improve the deception levels of honeynets when defending against a botnet. In addition, we will discuss several methods that attackers use to detect suspicious environments, like honeynets.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Today's cyber world faces many security threats such as viruses, trojans, botnets, etc. Botnets are an important part of current malware world, and provide the biggest challenge for security researchers. [56] They are used by attackers for many types of malicious activities that range from Distributed Denial of Service (DDoS) attacks, to stealing financial information and fraudulent use of on-line services [52][56].

With botnet threat security researchers must develop an effective strategy that would help mitigate the malicious effects of botnets. One such strategy is honeynet, as it has been proven to be an effective solution in detecting and defending against botnets as part of an intrusion deception system on the network [4][30][52]. Even though using honeynets has proven to be an effective strategy for detecting and defending against botnets, malware developers are not sitting idle either. They have found several strategies to counter and avoid honeynets with so-called honeynet-aware botnets [13][32][67][53][5]. Honeynet-aware botnets have rendered the deception levels of honeynets less effective.

With botnet defenses evolving, many researchers started search for solutions which would help honeynets to deceit honeynet-aware botnets, as honeynets still remain the most viable way of detecting and analyzing the new botnets. To search for solutions, we must first understand the current state of botnet detection, and the research by Niels Provos [52] is a great starting point for studying techniques on botnet mitigation.

1.2 Problem statement

With the constant evolution of botnets and their defense mechanisms, it is essential that honeynet operators implement new ways of deceiving an attacker by hardening the honeynet. Hardening of honeynet against botnet defenses can be accomplished by introducing the standard network and business services to it, and making the honeynet appear as it has a production value from an

attacker's point of view. A successful strategy of using honeypots and honeynets relies on them going undetected from external attackers. This desired quality leads us to describe honeypots and honeynets as intrusion deception systems. "A part of the problem lies in the way that honeynets and honeypots reveal their true purpose through data containment and control" [52]. Botnet's spreading mechanisms rely on the malicious data to propagate on the network, and honeypots carefully control the data propagation. This limits the propagation of the botnet on the network, and attackers can detect it through simple observations, and reliably determine that the infected system is a honeypot, thus stopping the attack. This limitation of deception might render honeynets less effective for learning about attacker's techniques and malware they use, thus defeating the initial purpose of the honeypot [52].

Honeypots regularly encounter botnet attacks [67]. Botnets designed to propagate traffic from one machine on a network to another and build its bot army from infected machines [63]. When the bot encounters a honeypot, its propagation on the network stops, because the honeypot is designed to control and contain the network traffic. The bot master, who oversees the creation of its bot army, can detect such attempt by a honeypot, and either stop the attack, or use alternative methods to disguise the propagation [63]. Thus the purpose of the honeypot is defeated, and its no longer gathering any information about an attacker, nor recording any malicious activity [52].

A honeypot is competent tool designed to help in the discovery of attacker's malicious techniques so that security researches may create and implement defenses against them. With the rising success of honeynets at catching malicious activities of cybercriminals [58], they become a new target of an attack, because cybercriminals want to avoid the discovery of their methods. If an attacker would execute a previously unknown attack on a honeypot, the knowledge of this attack would reach analysts much faster than if they would attack a regular system. This value of a honeynet lies in the ability to gather information when all other techniques fail. The main focus is on tools, tactics, and techniques that cybercriminals use. The ability of a honeynet to collect information on such attackers is in reverse proportion to the ability of a attacker to find out that he is interacting with a honeypot and not with "the real thing" [16]. Because of this, cybercriminals develop new techniques to detect if the targeted system is a honeypot or other suspicious environment (like jail [53]), and, in some cases, to break their defense and use honeypots for malicious activities [29].

1.3 Contribution

The main contribution is to develop technique that help harden honeynets against honeynet-aware botnets. We decided on a two step approach. First, we will implement a solution to demonstrate hardening of the honeynet against a honeynet-aware botnets. Then, we will show how propagation of the botnet from one honeypot to another can be a sufficient deception mechanism. The first step is to implement the system proposed by Wang et al. [68] to show that the botnets can check the compromised bots and see if the nodes are still a part of the botnet. If a particular bot does not obey the commands or acts in a suspicious manner, then the bot master may suspect a suspicious environment like a honeypot and direct the malicious activity away from the honeypot. The compromised bots are also tested by the bot master to verify if the bot is obeying commands received through the C2 server from the master and carrying out those commands [68].

We will setup our virtual network for a botnet attack using Zeus Bot, which can be deployed for several attacks, such as stealing credentials and sending phishing e-mails [33]. Zeus uses a drive-by download type of spreading mechanism, which relies on users to click a malicious link in the e-mail [33].

To show that our approach is valid, two scenarios were created. First (Figure 5.9), we show how honeypot, that controls the traffic, is detected and the attack stops. Second (Figure 5.10), we show that with the use of 2 or more honeypots we can trick the attacker to think that the network is legitimate and to continue the communication with his botnet.

CHAPTER 2

BACKGROUND AND RELATED WORK

Criteria	Costarella, Chung, and Dittrich [13]	Ping, Wang, Lei, Wu, Cunningham, and Zhou [67]	Yong Tang, Shigang, and Chen [63]	Gu, Zhang, and Lee [26]
Hardening honeynet	Yes. Allows malicious traffic to propagate without being modified.	Yes. Suggested methods as further research but not implemented.	Yes. Uses extra steps of "double honeypot" propagation idea against worm attack.	No. Botnet detection technique is not applied to honeynet or honeypot.
Defenses against botnet attacks.	Yes. By controlling the propagation of the botnet.	No. Shows how botnets use sensors to check propagation of unmodified traffic between controlled bots.	No.	Yes. By detecting botnet command and control channels and intercepting communication back to bot master.
Defenses against worm attacks	No.	No.	Yes. Introduces "double honeypot" idea which forces attackers to take extra steps to determine deception.	No. Targets botnet communication channels which are not used in worm attack.
Increases deception levels within honeynet	Yes. Increase deception by allowing the botnet to propagate the attack.	No, but illustrates how botnet can measure propagation of unmodified malicious data.	No.	No. Deception is used only to spoof communication back to bot herder, not to propagate the attack within the network.

Table 2.1: A Comparison of Previous Work

The value of this research rests on the background works performed by Wang et al. [67] in the area of honeynet aware botnets, Tang et al. [63] in the area of utilizing a double honeypot deception approach against a worm attack, Gu et al. [26], who developed the tool BotSniffer to detect botnet Command and Control communication in network traffic, as well as Costarella et al. [13], who proposed a solution to harden honeynets by allowing malicious traffic to propagate between nodes of the honeynet. Table 2.1 shows a comparison of these approaches in terms of four criteria:

- hardening of a honeynet
- ability of the honeynet to defend against botnets
- ability of the honeynet to defend against worms
- increasing the deception levels of honeynets

In a typical botnet, the bot master is capable of sending the commands to the bots under his command instructing an arbitrary bot to broadcast malicious traffic on the network in an attempt to spread the infection ot to perform other malicious actions. The bot master uses sensors to monitor the incoming transmissions from compromised nodes and sniff traffic around them to send reports back to the master with verification of whether modes are still compromised and under her control [67]. The bot master is able to check whether malicious traffic is being contained within a bot, or modified coming from a bot so that it is no longer malicious. This allows the master to identify if the bot is on a regular system that has been fully compromised or the bot is in suspicious environments like honeynets, or jailed environments [68].

Several researchers introduced different approaches to increasing the complexity of deception systems and to harden honeynets. Tang et al. [63] explores additional complications imposed on malware attempting to detect honeypots. The researchers do this by adding additional layers of complexity to the deception by introducing the concept of "double honeypot," which is a conceptual deception system against worm attack [63]. The design of the "double honeypot," proposed by them, relies on adding extra layers of complexity to the detection methodology employed by the malware or the malicious actors deploying it. This complexity adds more work for bot master to detect the honeypot environment. If the honeynet operator can propagate the attack from the first honeypot to an additional honeypot, and yet retain the malicious code within the honeynet,

without allowing it to get out, then the hardening of the honeynet has taken place. The additional work for the attackers adds additional layers of deception and additional layers of hardening against the ability of an attacker to detect a honeypot.

The bot master has to have some mechanism to communicate with the bots in his botnet. These C2 channels are well-known protocols such as IRC and HTTP [32] and many bot masters use well-known Internet locations to network their communication systems [26]. Botnets differ from other forms of malware such as worms in that they utilize these C2 channels [32]. These channels represent the critical communication link for bot masters to send commands to the bots comprising their bot army [26]. Gu et al. present a variety of different types of communication protocols that attackers use as a communication medium between C2 and the bot army [26]. Two of the most popular protocols, researchers argue, are IRC and DNS [26]. Gu et al., in their work, developed a tool BotSniffer to recognize and track the botnet communication in network traffic [26]. "BotSniffer is an anomaly based detection system that looks for anomalies in what would be considered "normal" network traffic."[26] BotSniffer utilizes the theory that one bot master controls all the bots on the network, and these bots will exhibit similar behavior on the network. In our research we will be using tcpdump, Wireshark, and the logging capabilities of the routers to look at the traffic in the communication channels.

Using virtualization techniques is one of the most used method to create honeynets [50]. Researchers Costarella et al. used VMware products in their research to build a honeynet comprising of a several honeypot nodes to mimic the behavior of a regular network that might have value for attackers [13]. Even though virtual environments are not hard to detect, as shown by Dornseif et al. [16] and Holz and Raynal [53], researchers Piva and de Geus [50] argue that using virtual machines can help increase security and levels of deception that honeypots provide.

CHAPTER 3

HONEYNETS

3.1 What is a Honeypot?

According to Lance Spitzner, a honeypot is a system designed to learn how "black-hats" probe for and exploit weaknesses in an IT system [60]. It can also be defined as "an information system resource whose value lies in unauthorised or illicit use of that resource" [59]. In other words, a honeypot is a trap, put out on a network as a bait to lure attackers. Honeypots are typically virtual machines, built to imitate real machines, resembling the appearance of workstations or servers running full services and applications, with open ports that are found on a typical machine on a network. When an attacker attempts to compromise a honeypot, attack-related information, such as the IP address of the attacker, will be collected. The attack provides valuable information and analysis on attacking techniques, allowing system administrators to "trace back" to the source if required.

Honeypots are different from most security tools in that they can take on different appearances. Honeypots are a highly versatile tool that can be applied to a variety of situations. The definition of honeypots may seem vague at first, because they can be used to achieve many different goals and come in a variety of forms. For example, honeypots can be used to deter attacks, a goal shared with firewalls. They also can be used to detect attacks, similar to the Intrusion Detection System. Another way to use honeypots is to capture and analyze automated attacks, such as botnets and worms, or act as early indication of an attack and warning sensors.

3.2 Types of Honeypots

Honeypots can be used for production or research purposes [9]. Production honeypots are what most people typically think of as a honeypot. They add value to the security of a specific organization and help mitigate risk. You implement these honeypots within your organization because they help secure your environment, such as detecting attacks. Production honeypots are the law enforcement of honeypot technologies. Their job is to deal with the bad guys [59]. Commercial

organizations often use production honeypots to mitigate the risk of attackers. Production honeypots usually are easier to build and deploy than research honeypots because they require less functionality. Because of their relative simplicity, they generally have less risk. It's much more difficult to use a production honeypot to attack and harm other systems [60]. However, they also generally give us less information about the attacks or the attackers than research honeypots do. We may learn which systems attackers are coming from or what exploits they launch, but we will most likely not learn how they communicate among each other or how they develop their tools.

Research honeypots are designed to gain information about the blackhat community [59]. Their primary mission is to research the threats organizations may face, such as who the attackers are, how they are organized, what kind of tools they use to attack other systems, and where they obtained those tools. If production honeypots are similar to law enforcement, then you can think of research honeypots as counterintelligence, used to gain information on the bad guys [59]. This information lets us better understand who our threats are and how they operate. Armed with this knowledge, we can then better protect against them. Research honeypots help the security community secure its resources indirectly rather than directly. Research organizations such as universities and security research companies often deploy research honeypots. Also, organizations such as military or government agencies may use research honeypots.

Research honeypots are far more involved than production honeypots [59]. To learn about attackers, you need to give them real operating systems and applications with which to interact. This gives us far more information than simply what applications are being probed. We can potentially learn who the attackers are, how they communicate, or how they develop or acquire their tools. However, this increased functionality has its disadvantages. Research honeypots are more complex, have greater risk, and require more time and effort to administer. In fact, research honeypots could potentially reduce the security of an organization, since they require extensive resources and maintenance [59].

3.3 Advantages and Disadvantages of Honeypots

3.3.1 Advantages

One of the challenges the security community faces is gaining value from data. Organizations collect vast amounts of data every day, including firewall logs, system logs, and Intrusion Detection

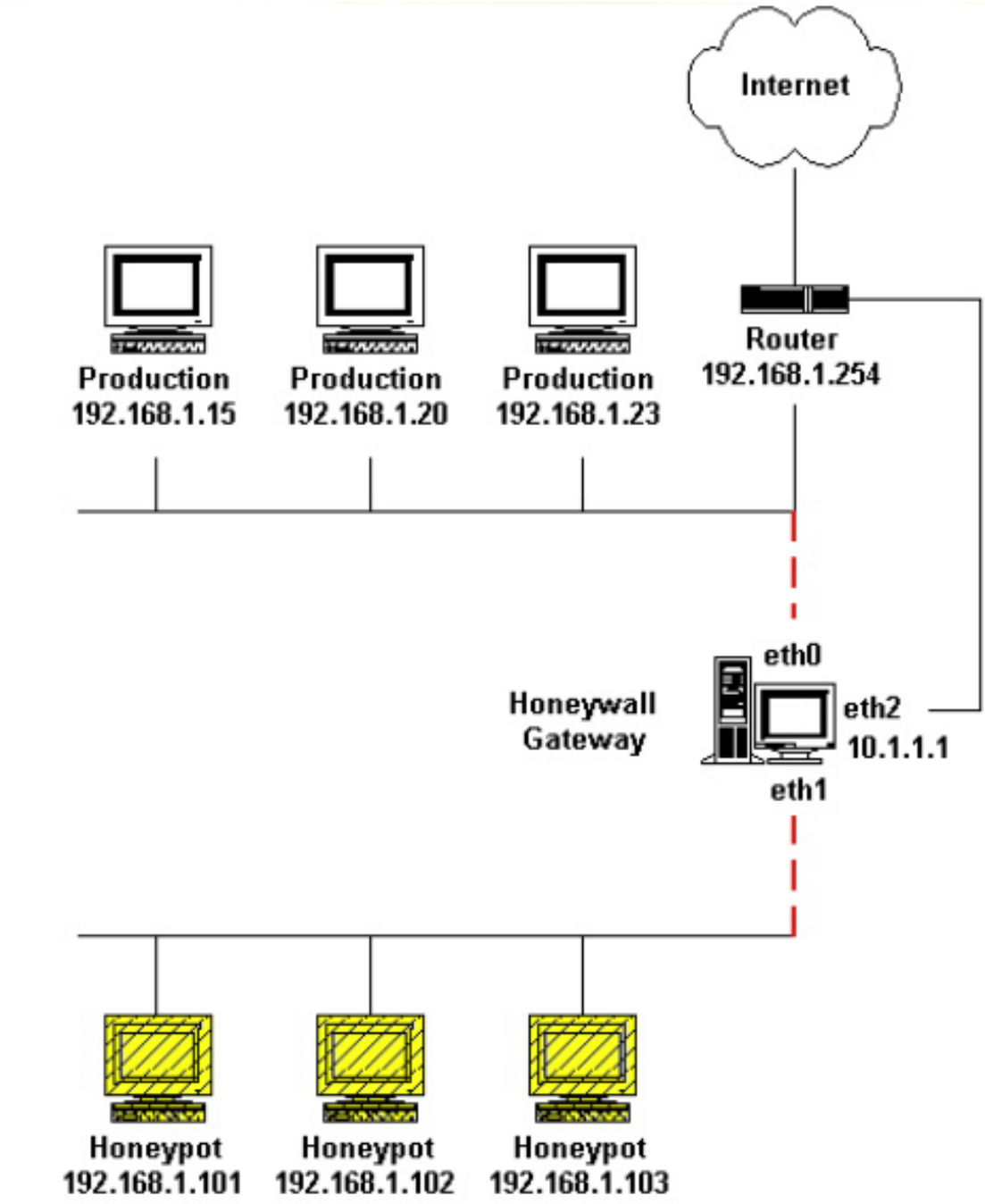

Figure 3.1: GenII Honeynet [61]

alerts. The sheer amount of information can be overwhelming, making it extremely difficult to derive any value from the data. Honeypots, on the other hand, collect very little data, but what

they do collect is normally of high value. The honeypot concept of no expected production activity dramatically reduces the noise level. Instead of logging gigabytes of data every day, most honeypots collect several megabytes of data per day, if even that much. Any data that is logged is most likely a scan, probe, or attack-information of high value. Honeypots can give you the precise information you need in a quick and easy-to-understand format. This makes analysis much easier and reaction time much quicker. For example, the Honeynet Project [61], a group researching honeypots, collects on average less than 1MB of data per day. Even though this is a very small amount of data, it contains primarily malicious activity. This data can then be used for statistical modeling, trend analysis, detecting attacks, or even researching attackers. This is similar to a microscope effect. Whatever data you capture is placed under a microscope for detailed scrutiny.

Another challenge most security mechanisms face is resource limitations, or even resource exhaustion. Resource exhaustion is when a security resource can no longer continue to function because its resources are overwhelmed. For example, a firewall may fail because its connections table is full, it has run out of resources, or it can no longer monitor connections [59]. This forces the firewall to block all connections instead of just blocking unauthorized activity. An Intrusion Detection System may have too much network activity to monitor, perhaps hundreds of megabytes of data per second. When this happens, the IDS sensor's buffers become full, and it begins dropping packets. Its resources have been exhausted, and it can no longer effectively monitor network activity, potentially missing attacks. Another example is centralized log servers. They may not be able to collect all the events from remote systems, potentially dropping and failing to log critical events. Because they capture and monitor little activity, honeypots typically do not have problems of resource exhaustion [59]. As a point of contrast, most IDS sensors have difficulty monitoring networks that have gigabits speed [61]. The speed and volume of the traffic are simply too great for the sensor to analyze every packet. As a result, traffic is dropped and potential attacks are missed. A honeypot deployed on the same network does not share this problem [61]. The honeypot only captures activities directed at itself, so the system is not overwhelmed by the traffic. Where the IDS sensor may fail because of resource exhaustion, the honeypot is not likely to have a problem. A side benefit of the limited resource requirements of a honeypot is that you do not have to invest a great deal of money in hardware for a honeypot. Honeypots, in contrast to many security mechanisms such as firewalls or IDS sensors, do not require the latest cutting-edge technology, vast amounts of

RAM or chip speed, or large disk drives. This means that not only can a honeypot be deployed on a gigabit network but it can be a relatively cheap computer [59].

The last, but not least advantage of honeypots is their simplicity [59]. There are no fancy algorithms to develop, no signature databases to maintain, no rulebases to misconfigure. Just take the honeypot, drop it somewhere in organization, and sit back and wait. While some honeypots, especially research honeypots, can be more complex, they all operate on the same simple premise: If somebody or someone connects to the honeypot, check it out.

3.3.2 Disadvantages

With all of these wonderful advantages, one would think that honeypots would be the ultimate security solution. Unfortunately, that is not the case. They have several disadvantages. It is because of these disadvantages that honeypots do not replace any security mechanisms; they only work with and enhance overall security architecture.

The greatest disadvantage of honeypots is they have a narrow field of view: they only see what activity is directed against them [59]. If an attacker breaks into your network and attacks a variety of systems, your honeypot will be blissfully unaware of the activity unless it is attacked directly. If the attacker has identified your honeypot for what it is, she can now avoid that system and infiltrate your organization, with the honeypot never knowing she got in. As noted earlier, honeypots have a microscope effect on the value of the data you collect, enabling you to focus closely on data of known value. However, like a microscope, the honeypot's very limited field of view can exclude events happening all around it.

Another disadvantage of honeypots, especially many commercial versions, is fingerprinting [59]. Fingerprinting is when an attacker can identify the true identity of a honeypot because it has certain expected characteristics or behaviors. An incorrectly implemented honeypot can also identify itself. For example, a honeypot may be designed to emulate an NT IIS Web server, but the honeypot also has certain characteristics that identify it as a Unix Solaris server [57]. These contradictory identities can act as a signature for a honeypot. There are a variety of other methods to fingerprint a honeypot [53]. Fingerprinting is an even greater risk for research honeypots. A system designed to gain intelligence can be devastated if detected. An attacker can feed bad information to a research honeypot as opposed to avoiding detection.

The third disadvantage of honeypots is risk: they can introduce risk to any environment [59]. A honeypot, once attacked, can be used to attack, infiltrate, or harm other systems or organizations. Different honeypots have different levels of risk. Some introduce very little risk, while others give the attacker entire platforms from which to launch new attacks. The simpler the honeypot, the less the risk. For example, a honeypot that merely emulates a few services is difficult to compromise and use to attack other systems. In contrast, a honeypot that creates a jail gives an attacker an actual operating system with which to interact. An attacker might be able to break out of such jail and then use the honeypot to launch passive or active attacks against other systems or organizations. Risk is variable, depending on how one builds and deploys the honeypot.

Because of their disadvantages, honeypots cannot replace other security mechanisms such as firewalls and intrusion detection systems. Rather, they add value by working with existing security mechanisms. They play a part in overall defenses.

3.4 The Role of Honeypots in Security

Honeypots are systems that help mitigate risk in organization or environment. They provide specific value to securing systems and networks. Their job is to take care of the bad guys. How do they accomplish this? To answer that question, we are going to break down security into three categories and then review how honeypots can or cannot add value to each one of them. The three categories we will discuss are those defined by Bruce Schneier in "Secrets and Lies" [54]. Specifically, Schneier breaks security into prevention, detection, and response.

3.4.1 Prevention

Prevention means keeping the bad guys out. The security community uses a variety of tools to prevent unauthorized activity. Examples include firewalls that control what traffic can enter or leave a network or authentication, such as strong passwords, digital certificates, or two-factor authentication that requires individuals or resources to properly identify themselves [68]. Based on this authentication, you can determine who is authorized to access resources. Mechanisms such as encryption prevent attackers from reading or accessing critical information, such as passwords or confidential documents.

What role do honeypots play here? How do honeypots keep out the bad guys? Honeypots add little value to prevention, since they do not deter the enemy [59]. In fact, if incorrectly implemented,

a honeypot may introduce risk, providing an attacker a window into an organization. What will keep the bad guys out is best practices, such as disabling unneeded or insecure services, patching vulnerable services or operating systems, and using strong authentication mechanisms. The time and resources involved in deploying honeypots for preventing attacks, especially prevention based on deception or deterrence, is time better spent on security best practices. As long as you have vulnerable systems, you will be hacked. No honeypot can prevent that.

3.4.2 Detection

The second tier of security is detection, the act of detecting and alerting unauthorized activity [54]. Within the world of information security, we have the same challenge. Sooner or later, prevention will fail, and the attacker will get in. There are a variety of reasons why this failure can happen a firewall rule base may be misconfigured, an employee uses an easy-to-guess password, a new vulnerability is discovered in an application. There are numerous methods for penetrating an organization. Prevention can only mitigate risk; it will never eliminate it. Within the security community there are several technologies designed for detection. One example is Network Intrusion Detection Systems, a solution designed to monitor networks and detect any malicious activity [68]. There are also programs designed to monitor system logs that look for unauthorized activity. These solutions do not keep out the bad guys, but they alert us if someone is trying to get in and if they are successful.

How do honeypots help detect unauthorized or suspicious activity? While honeypots add limited value to prevention, they add extensive value to detection. Due to their simplicity, honeypots effectively address the three challenges of detection: false positives, false negatives, and data aggregation [59]. Most honeypots have no production traffic, so there is little activity to generate false positives. The only time a false positive occurs is when a mistake happens, such as when a DNS server is misconfigured. Otherwise, honeypots generate valid alerts, greatly reducing false positives [54]. Honeypots address false negatives because they are not easily evaded or defeated by new exploits. In fact, one of their primary benefits is they can detect a new attack by virtue of system activity, not signatures [54]. A honeypot does not use a signature database. It works on the concept that anything sent its way is suspect and malicious. Additionally, honeypots do not require updated signature databases to stay current with new threats or attacks. Honeypots happily capture any attacks thrown their way. The simplicity of honeypots also addresses the third

issue: data aggregation. Honeypots address this issue by creating very little data [54]. There is no valid production traffic to be logged, collected, or aggregated. Honeypots generate only several megabytes of data a day, most of which is of high value. This makes it extremely easy to diagnose useful information from honeypots [59].

3.4.3 Response

Once we detect a successful attack, we need the ability to respond. The challenge that organizations face when reacting to an incident is evidence collection that is, figuring out what happened and when [54]. This is critical not only if an organization wants to prosecute an attacker but also when it comes to defending against an attack. Once compromised, organizations must determine if the attacker hacked into other systems, created any back doors or logic bombs, modified or captured any valuable information such as user accounts [14]. When an attacker breaks into a system, their actions leave evidence, evidence that can be used to determine how the attacker got in, what she did once she gained control of the system, and who she is [51]. It is this evidence that is critical to capture. Without it, organizations cannot effectively respond to the incident. Even if the attackers take steps to hide their actions, such as modifying system log files, these actions can still be traced.

Honeypots can help address these challenges to reaction capability. Honeypot has no production activity, so this helps the problem of data pollution [59]. When a honeypot is compromised, the only real activity on the system is the activity of the attacker, helping to maintain its integrity [54]. Honeypots can also easily be taken offline for further analysis. Since honeypots provide no production services, organizations can easily take them down for analysis without impacting business activity [59]. However, honeypots are not the single solution for incident response, they are only a tool to assist. The most critical step any organization can take is preparing before an incident. Examples include having a documented response plan, taking images of critical files for future analysis, and having the technical tools to quickly recover evidence. It is these best practices that will ensure effective incident responses. However, honeypots can be a powerful tool to complement your reaction capabilities by capturing details on how the attacker got in and what they did.

3.5 Legal Issues with Honeypots

Honeypots is a new and becoming technology in the security community. As security professionals begin to understand the value of honeypots, the legal community is trying to understand them as well. The purpose of this section is to address the most common legal issues that may apply with the use of honeypots. There is no single answer to whether honeypots are legal or not. There are many variable at play - the country you reside may have different laws and regulations and differ in how they apply different aspects of honeypot applications. Even in the United States, each state can have different laws that might limit the applications of honeypots. The legality of the implemented honeypot might depend on the type of information being collected and the intended use for that information. It is better to contact an attorney and legal representatives regarding the issue of deploying honeypots. In addition to the limitations from federal or state government, honeypots might be individually prohibited by the organization in which one might want to implement it. It is best to contact the legal representative within the organization about issues concerning the implementation of honeypot technologies, because even if state and federal laws might permit the use of honeypots, the organization might prohibit the use of honeynets on the premise of civil liability issues.

3.5.1 Privacy

Privacy has a controversial history in the United States, as it hasn't been codified until late 1960's [46][47][45]. This means that legal issues concerning privacy can be complicated regarding the use of the honeypots, because they record all the activity that is occurring on them. To make the issue of privacy challenging, each state has it's own laws about privacy, with some states supplementing the Federal law, like California [46]. Worse case scenario is when intruder is from another country. Which legal statutes to apply in this case?

The closest legal statute in United States that applies to honeypot technologies is The Federal Wiretap Act [47], under which it is illegal to capture the communications of an individual in real time without their permission. There are two factors, concerning honeypot technologies, that affect the legal implications: what information is being collected and what, the collected information, is being used for.

First, how the honeypot is being used might affect the privacy issues. The intended use of honeypot might fall under the Service Provider Protection exemption from The Electronic Communications Privacy Act [27], and thus the definition of the intended use is important. ECPA enforces limitations on accessing, collecting, and disclosing the stored data to others without an authorization. It also restricts public service providers from disclosing information to government without legal process or authorization [59]. Under the exemption, security technologies can collect information as long as this technology is used for protection of the environment in which it is being implemented [27]. Honeypots fall under this category indirectly, because they don't provide a physical protection to the organization (like firewalls, IDS, IPS), but the information they gather can be used to defend against threats. Although, the less the honeypot is used for protection purposes, the less likely it will fall under the Service Provider Protection exemption.

A second important factor is the type of the information being collected by the honeypot. While ECPA covers the disclosure of information, The Wiretap Act [47] and Pen/Trap Statute [45] are the primary sources of federal law regarding the real-time interception of electronic data on the network [59]. There are two types of data - transactions and content. The transactional data is information about a transmission - IP address, port number, IP header, date, etc. Pen/Trap statute defines transactions as "dailing, routing, addressing, and signaling information" [45]. The content is the actual data of the communication - contents of a file, e-mail, chats, etc. As both transactional data and content are defined in Wiretap Act and Pen/Trap Statute respectively, the content has more privacy issues as ECPA also applies to it. For example, low-interaction honeypots, defined earlier, are used to capture only transactional data, with low or none of content data being collected. This type of honeypot falls under the Pen/Trap Statute as it makes illegal the collection of data in real-time, but might be exempt under the Service Provider Protection, discussed earlier, as long as it provides the protection of the service provider. Unlike low-interaction honeypots, high-interaction honeypots gather vast amount of data about an intruder. Honeynet is an example of such honeypot, as it is designed to gather information on a variety of threats. Such honeypot falls under ECPA, Wiretap Act, and Pen/Trap Statute. The exception for Service Provider Protection still applies to this type of honeypot, because it gathers information to defend against threats.

"Searching and Seizing Computers and Obtaining Electronic Evidence in Criminal Investigations" [27] is a great source of information about the aspects of gathering electronic data. It is

frequently updated, and contains the information about ECPA, Wiretap Act, and Pen/Trap Statute and other "dynamic areas of the law" [27].

3.5.2 Entrapment

Another legal issue that raises a concern is "entrapment." Some individuals and organizations might feel that honeypot technologies might imply the entrapment and, as such, cannot be deployed privately or within an organization. These issues arise with honeypots because they are deigned to attract hackers.

Entrapment is defined by law dictionary as "a law-enforcement officer's or government agent's inducement of a person to commit a crime, by means of fraud or undue persuasion, in an attempt to later bring a criminal prosecution against that person" [34]. Thus, entrapment, as a valid concern for honeynet owners as other legal issues presented earlier in this section, is actually a very narrow legal defense. Because private honeynet owner is not a law enforcement agent, and thus an attacker cannot use entrapment defense when she broke into a host machine that happened to be a honeypot. Entrapment defense can only apply in a criminal case in which the law enforcement agent caused the defendant to commit a crime.

3.5.3 Liability

Although previously stated legal issues are a large concern for owners of honeynets, the biggest issue is downstream liability [59]. Because the purpose of honeypots is to be compromised, a badly configured honeypot might allow the hacker to launch further attacks from compromised honeypot. In this case, who is liable for such attacks - the owner of misconfigured honeypot or the attacker? Although the harm is still inflicted by an attacker, if the owner of the honeypot would configure it correctly, the intruder would not be able to use it to launch an attack.

The main difficulty involving downstream liability is that it is state based and not federal based. Thus, a damaged party in New York can bring the suit under New York tort law even though the honeypot from which an attack was launched was located in California. This makes deciding if the owner of the honeypot is liable hard to predict. There has yet to be a case about downstream liability of honeynet owners, but it is best to properly secure and monitor the honeypot to prevent downstream attacks.

CHAPTER 4

BOTNETS

Malicious software, or malware for short, is playing an important role in our modern day high-tech life. In the past, malware has limited local impact, but, with the success of the Internet, the impact has grown into widespread infection, affecting millions of systems world wide. With this, there has been a significant shift in motivation for creating malware - if, at first, the goal of the attackers was vandalism, or establishing the reputation among the community of hackers, then now, with the growth of on-line shopping and banking, it shifted to target financial institutions and personal information.

As a result of such switch, today's economies around the world suffer from a wide range of malware attacks. An ITU report, giving an overview of financial studies about malware, presented numbers ranging from $US13.2 billion for the global economy in 2006 to $US67.2 billion for US businesses alone in 2005. Consumer reports estimated direct costs to US citizens of malware and spam at $US7.1 billion in 2007. The cost of click fraud in 2007 in the US was estimated to be $US1 billion [7]. While the number of malware samples has been increasing at an exponential rate over the last few years [3], Computer Economics has measured a declining worldwide impact of malware attacks on businesses, with financial costs of $US17.5 billion in 2004, $US14.2 billion in 2005 and $US13.3 billion in 2006 [17]. In their report [17], Computer Economics argue, that, while corporations see a decline in costs associated with malware attacks, hackers shift towards attacking private users for their credentials and financial information.

With the shift of motivation for creating malware, the diversity of malware has grown almost exponentially, from several thousand in 2007 to over 400,000 of total malware samples in 2015, with over 120,000 new malware samples in 2015 alone [3]. The need for sophistication and variation in malware is driven by increase in security solutions available today on the market, and their effectiveness. Symantec states that, "in 2009, a total of 2,895,802 new signatures for the detection of malware were created, 51% of all the signatures ever created by them" [20]. Kaspersky Labs

identified about 15 million unique samples of malware in 2008, which means that one unknown sample was discovered roughly every 2 seconds [24].

4.1 What are Botnets?

Botnets, as defined by McAfee, are networks of compromised, remotely controlled computer systems [70]. Plohmann et al. defined a botnet as "...a concept of advanced malicious software that incorporates usually one or more aspects of the techniques introduced by viruses, worms, Trojan horses and rootkits for propagation and hostile integration into a foreign system, providing the functionality of the compromised system to the attacker" [51]. Today more and more botnets consist of Trojan Horses, like notorious malware Zeus Botnet, or ZBot [33].

What mainly defines a botnet is it's most important part - command-and-control (C2) server, which is used to control all systems infected with the bot malware. Several types of communications are used by C2 servers to talk to the bot army - IRC, P2P, and HTTP [70]. IRC communication was used in first C2 servers to control bots. This "old-school" communication protocol is still common among botnets, but suffers from a single point of failure - the IRC server, needed to provide communication between C2 and bot army [10]. Once the server is down, or blocked by firewall, the bot master can no longer communicated with infected systems. With the use of IRC communication, all bots in the botnet receive commands from C2 immediately and have high reaction time [5]. In P2P communication, links between bots allow them to communicate with other bots in the botnet. The knowledge about participating bots in the botnet is propagating throughout the botnet itself, without the need for C2 to oversee the entire botnet. This type of communication has it's own flaws, because the information about the whole botnet cannot be accessed directly, the bot master must provide commands to any arbitrary infected machine, which will relay the message from bot master further in the botnet. This means a low reaction time for the botnet. However, because commands originate from arbitrary machine on the botnet, it makes determination of the bot master almost impossible. Another advantage is that, unlike IRC (and HTTP, discussed later) C2 server, there is no central server in P2P, and thus it cannot be blocked to stop an attack [70]. HTTP is the successor of IRC protocol used by C2 servers. According to Wostowsky et al. C2 servers utilizing HTTP protocol are recent, and switch has been made only 7-8 years ago [70]. HTTP is used to deliver data over the Internet; it transports human readable data

like contents of web pages, images, binary data, and, because of these features, HTTP is rarely filtered, which makes it a viable protocol to serve in botnet communications. As IRC protocol gets older, more various generations of botnets show the use of HTTP and P2P as their main communication protocols [66][25][14].

HTTP is the successor of IRC, used as a main protocol for botnet communication. The switch from IRC to HTTP began with advances in "malware kits," developed mainly by Russian cyber-criminals [70]. These kits allow hackers to install web server, configure settings for malware to suit their needs, and build bots to distribute to victims. The server contains the links to the malware, and, if user clicks on any links that attackers send in spam emails, the malware is downloaded to the user machine, infecting it. Once on victim's machine, the kit determines which exploit to use depending on the region, operating system, browser version, and even multiple client application versions installed on the victim's machine [70].

Zeus botnet is one of the HTTP botnets, and will be closely examined later in this paper. It uses kit to build and deploy the malware. The package it comes in contains a web server and a builder kit to build a client malware [18]. It existed since 2007 and has evolved over time, with the current available version being 2.0.8.9. The following example shows the window of the Zeus Builder for current version:

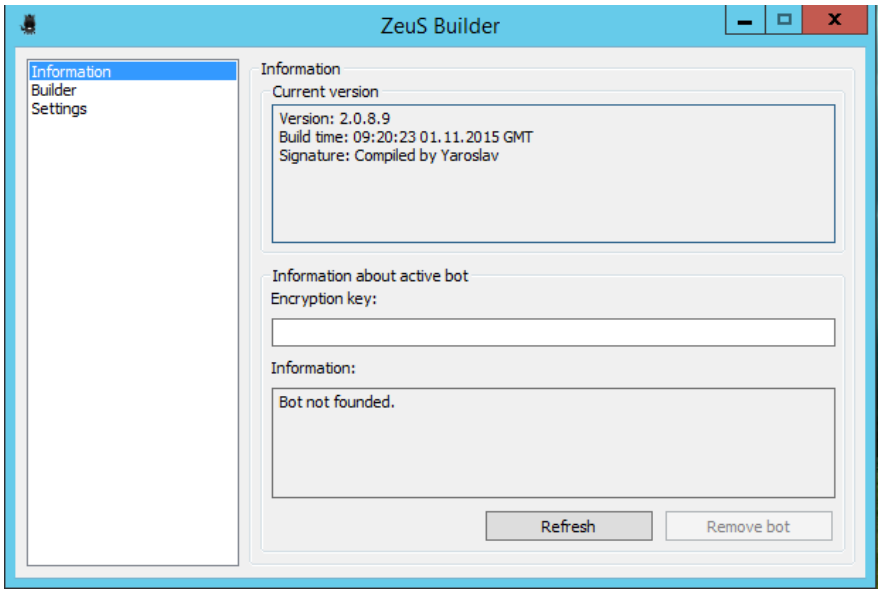

Figure 4.1: Zeus Kit Information Tab

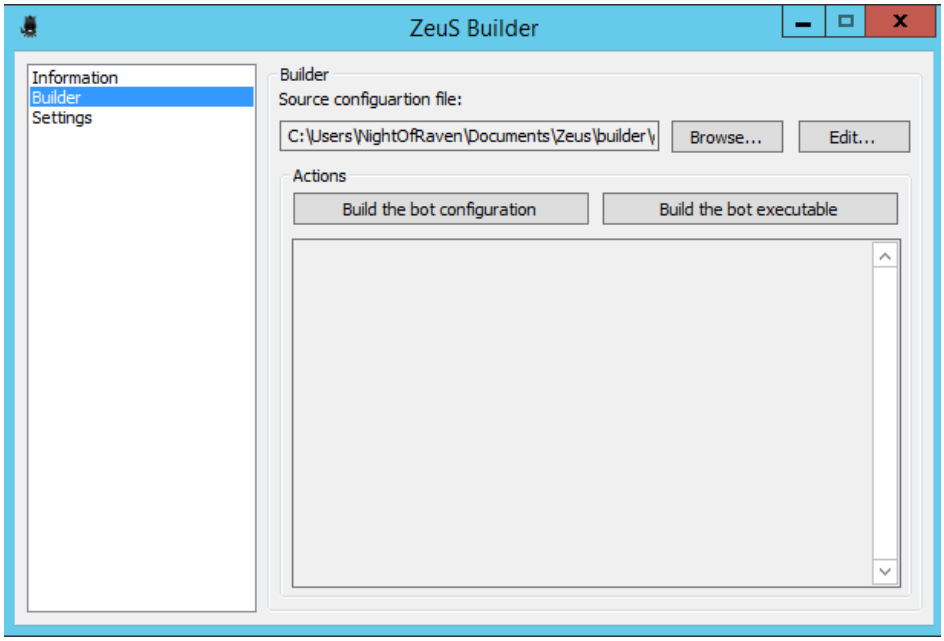

Figure 4.2: Zeus Kit Builder Tab (shown before building config.bin and bot.exe)

The left pane shows 3 tabs: Information, Builder, and Settings. Settings tab contains only language selection, with available languages being English and Russian, which indicates that malware might have originated from Russia. The Information tab shows current version (2.0.8.9), time of the build (can be changed in config.txt file), and the signature of the builder. In this example i used my signature since i was the one compiling the source code. It also will search for active bots if the encryption key, used to generate the bot, is provided.

The Builder tab helps to build a new bot using 2 input files: config.txt and webinjects.txt. In the next chapter we will describe the building process in more detail.

4.2 What are Botnets Used For?

A botnet is a tool, which cybercriminals use for different purposes. The botmaster can have botnets in upwards of sever hundred thousand bots under their command. What are they using them for? The following table [56] outlines famous botnets and types of attacks they can be used for.

Name	DDoS Attacks	Spam E-Mail	Identity Theft
Confiker	-	-	-
Kraken	-	Yes	-
Rustock	Yes	Yes	Steal sensitive information
Storm	Yes	Yes	-
TDL4	Yes	Yes	Yes
Torping	-	-	Yes
Waledac	Yes	Yes	Capture login information
Zeus	-	Yes	Steal passwords, credentials, and banking information
Zeus(P2P)	Yes	-	Steal crypto certificates, steal cookies, and steal banking information

Table 4.1: Botnet Attacks

4.2.1 Distributed Denial-of-Service

Botnets are often used for DDoS attacks, because of the vast amount of bots that some botnets incorporate [6]. A DDoS is an attack on a network or a web server for the purpose of disrupting the operation of the attacked system and causing a loss of service to users. Each botnet malware can include several different possibilities to carry out a DDoS attack, with most commonly used being TCP SYN and UDP flood attacks [4]. Many companies depend on web services for their every day operations, like on-line banking and web stores. For such companies, the downtime cause by DDoS attacks means loss of business and loss of profits. A popular target for such attacks are gambling websites during major events like Super Bowl [48]. Attacks don't necessary focus on web services - any service on the Internet is vulnerable to such type of attack.

The Shadowserver Foundation [21] recently found a new botnet called "Darkness" [36]. Darkness is generally comparable to the BlackEnergy bot [42], which also specializes in DDoS attacks, but is claimed to be more effective. It has been already actively used in some DDoS attacks and is advertised as being able to take down even large websites with just a few thousand bots [36].

4.2.2 Spam E-Mail

Some botnets, like the ones listed in Table 4.1, have capabilities to execute nefarious attack called spamming, which, in simple terms, is unsolicited mass sending of massive amount of e-mails. With the primary intention of such attack being advertisement, some bots use spamming to send

either payloads with bots or links to malicious websites, from which bots can be downloaded. This is one of the ways of botnet propagation. Spamming is also used for social engineering attacks called "phishing," or password fishing, which is used for identity theft purposes [12]. Estimates of spam e-mails share of all e-mails over the past few years ranges from 50-80% [55]. The figure below shows the origins of the spam e-mails, with Vietnam being the highest with 24.6% [55]. "The ability of botnets to use bots IP addresses to hide the true originator of the spam email complicates countermeasures such as the blacklisting of suspicious IP addresses" [51].

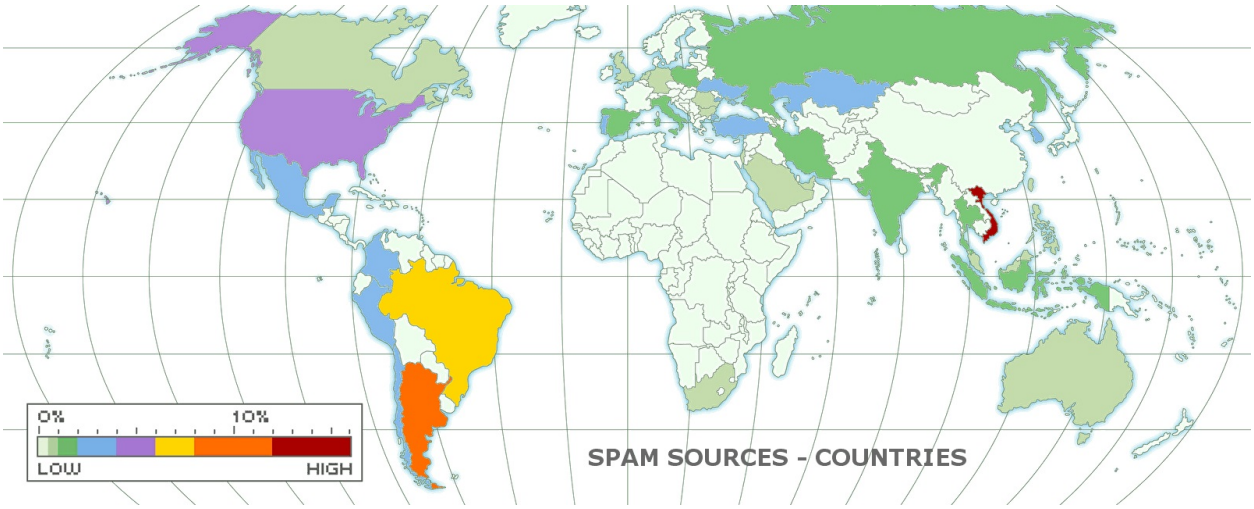

Figure 4.3: Spam Sources by Country [55]

4.2.3 Click Fraud and Pay-Per Install

Another way that botnets can be used for financial gain is through click fraud. An attacker must set up an account with a company who pays for visits or clicking banners or advertisements. One of such programs is Google's AdSense [23] program: AdSense offers companies the possibility to display Google advertisements on their own website and earn money for each click on the add by visitors. An attacker can use his botnet to visit such pages and generate clicks on target advertisements, which artificially increments the click counter. An attacker can use his botnet to directly generate profit for himself, or rent his botnet to a company, who will use it to generate it's own profits. Two companies, Click Forensics and Anchor Intelligence, who specialize in analysis of

traffic, reported an increased growth in click fraud for on-line advertising. Both companies report a continuous growth in click fraud from 14% in 2009 to almost 30% in 2010 [2][11].

4.2.4 Identity Theft

Identity theft is a major use of botnets. The intent behind it is the financial gain of the attackers. The combination of different functionality described above is often used for extracting user credentials and data [69]. Key targets include passwords for various services like e-mail accounts, web shops, banking platforms or social networking platforms [28]. Attackers use social engineering techniques, such as spam e-mails that pretend to be legitimate (banking, game, university e-mails) ask users to go online and submit their private information. Other techniques, like keylogging and website injections, are used by attackers to gain private information [1]. Some botnets, like Zeus, can host multiple fake websites pretending to be Ebay, PayPal, or a bank, and harvest personal information. Zeus also uses a web injection mechanisms to add extra fields to login forms, such as PIN number, or date of birth.

4.2.5 Political Reasons

In addition to economic interests, botnets may also be used for political interests. One such example happened in Estonia in April 2007, when, for two weeks, federal, banking and news websites were under DDoS attack. The attack originated from Russia, and was intended as a protest against the moving of a Russian war monument in Tallin, Estonia [43]. This was considered to be the first politically motivated cyber attacks of this size. The DDoS attack had a significant impact on the Estonian population [64].

Two new botnets were explored in depth in 2009 and 2010, GhostNet [15] and the Shadow Network [41]. These botnets are believed to be used for espionage. The investigators of GhostNet discovered 1295 infected machines in 103 countries, with around 30% of the infected machines considered as high-value, because they were located in various embassies, ministries, and commissions [15]. Network traces, captured from these machines, showed that C2 servers for these botnets were situated in China, and confirmed the extraction of sensitive documents [41].

Stuxnet is a more recent example of malware used for espionage and sabotage of high value properties located in the Middle East [31]. At 500-kilobites, the Stuxnet is considered one of the most complex pieces of malware ever identified [31]. Although it is debatable whether Stuxnet is

a botnet, it contains many typical botnet features, and, as of 2010, it infected over 100,000 hosts from over 155 countries, with majority of infections being in Iran [19]. It was the first malware to target critical infrastructure, as it's intended purpose was to sabotage the Uranium Enrichment Facility at Nazan, Iran [38].

CHAPTER 5

CONTRIBUTIONS

5.1 Choice of Botnet

For this research, we selected the notorious banking botnet - Zeus. Zeus is designed to perform man-in-the-browser attack by injecting the web pages of legitimate sites with additional fields. For example banking web site might have an extra field called PIN or Date of Birth added on login page to gather more information about the user. The bot is fully based on user level WinAPI and doesn't require any drivers or downloads. This allows the bot to successfully collect information even on Guest accounts in Windows systems.

5.1.1 Building the Bot

Zeus' source code, version 2.0.8.9, was leaked in 2011, and this is the version that we are using in this research [65]. This version of Zeus Botnet was written in Visual C++ version 10 with no additional libraries being used. Although to compile the bot, Service Pack 1 for Visual Studio 2010 had to be installed, because the linker file, used in original, was outdated. The source code had several errors in it, and had to be debugged extensively before the successful built. Before compiling the code, some changes to the make_full.bat file Figure 5.1, changed the signature to "Compiled by Yaroslav." The Zeus bot requires the Ultimate Packer for eXecutables [37], and 7z file archiver [49] to compress the Zeus Builder kit and to archive the server implementation. Latest versions of each software were downloaded and used for compiling the source code.

After the source code has been compiled, the following files and folders are created:

The folder builder contains the Zeus Builder kit executable, config.txt file, and webinjects.txt file. Folder other doesn't contain anything interesting, but 2 files redir.php and sockslist.php, which are used by the backdoor to find out what sockets are online. Server folder contains the backdoor BackConnect (zsbcs.exe), which is a shell hosted on the victim's machine and connects back to the attacker without sitting and always listening for connections. Server[php] folder is the heart of the Zeus botnet. It contains the control panel, which is used to track the state of botnets and to

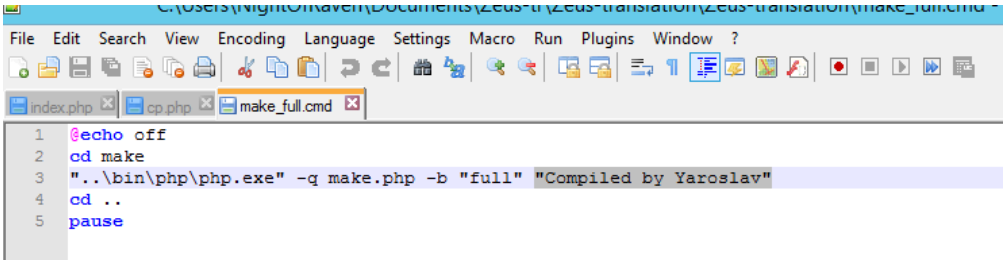

Figure 5.1: make_full.bat file

Name	Date modified	Type	Size
builder	11/1/2015 3:28 AM	File folder	
other	11/1/2015 3:28 AM	File folder	
server	11/1/2015 3:28 AM	File folder	
server[php]	11/1/2015 3:28 AM	File folder	
client32.bin	11/1/2015 3:20 AM	BIN File	138 KB
config	11/1/2015 3:20 AM	File	1 KB
ZS_2.0.8.9.7z	11/1/2015 3:20 AM	7Z File	623 KB
ZS_2.0.8.9.password	11/1/2015 3:20 AM	Text Document	1 KB

Figure 5.2: Output files created by compiler

collect the information that they send. File ZS_2.0.8.9.password.txt contains randomly generated password for the 7z archive ZS_2.0.8.9.7z. The archive, once extracted, contains the same files as server[php] folder. The purpose of this archive is to be sent and setup on the remote server through FTP.

5.1.2 Setting up Zeus Control Panel

Before we begin building our bots, we must first setup the server and Zeus Control Panel. The Control Panel is an open source PHP application, which can be run on IIS (version 6 or higher) or Apache (version 2.2 or higher) web servers. The Control Panel requires also MySQL server to store all the data on the botnet, which includes information about bots, and the data they collect.

For this installation, we used XAMPP, which is an Apache distribution, containing PHP, Perl, MySQL Server, PHP MyAdmin, and FileZilla FTP client [22]. Figure 5.3 shows XAMPP running

on Windows Server 2012 R2, with Apache and MySQL servers started on their respective ports.

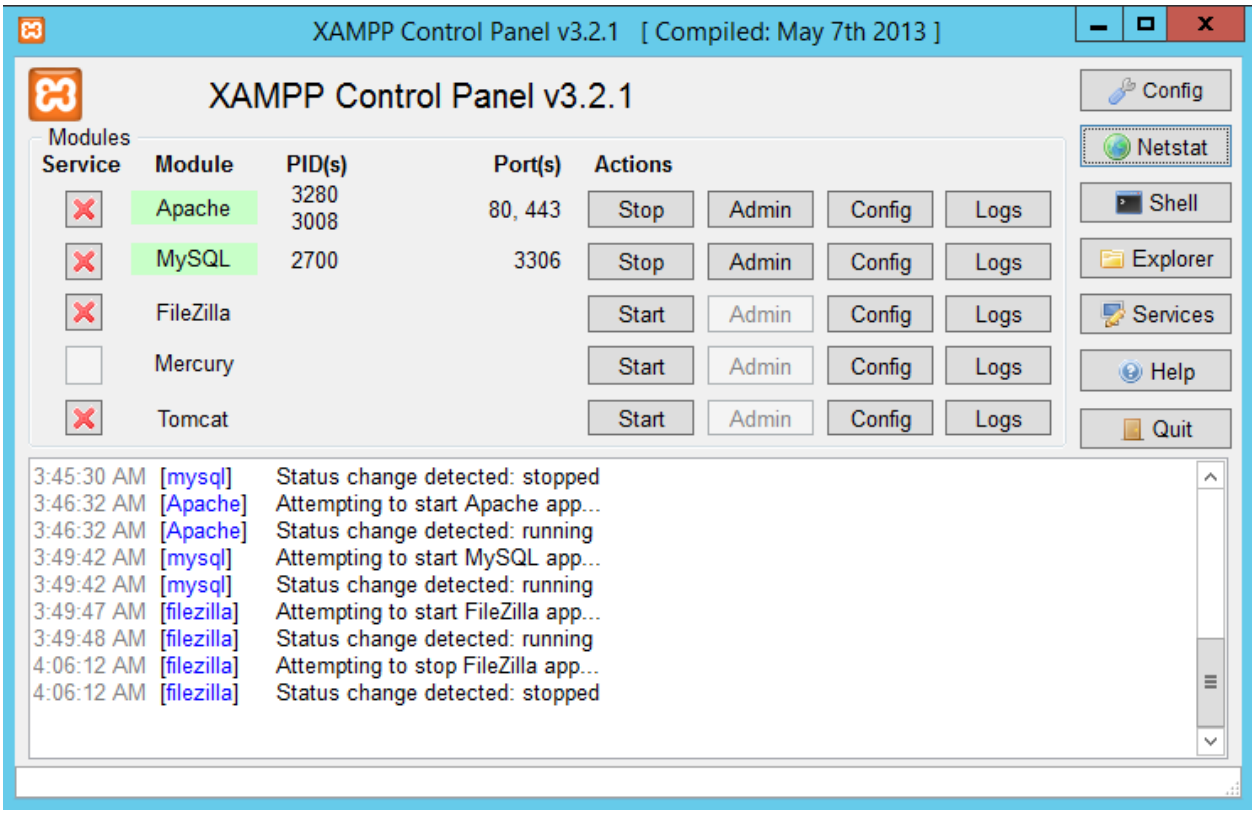

Figure 5.3: XAMPP Control Panel

Once the services were started, the setup of Zeus Control Panel has begun. First, we created a folder named zeus, and copied the contents of the server[php] folder. After, we logged in to our localhost from the browser, and, using phpMyAdmin, created the new database called zeusdb to be used by the botnet. After the database was created, we logged in to localhost/zeus/install to start installing the Control Panel. We get the prompt, like on Figure 5.4. We create a root user leaving the "User name" field default, creating a password, which we will use to connect to Control Panel. In the "Database" field, we enter the name of the database we created in previous step. "Encryption key" field is optional, but we provided a string "qwerty" as an example. This string will be later used in the configuration file for the Zeus Builder. After all necessary fields are complete, we press Install button, and it will take anywhere from 30 seconds to 2 minutes for the

installer to prepare the databases, add users, and create configuration file. After the installation is complete we can login to our Control Panel using http://localhost/zeus/cp.php address.

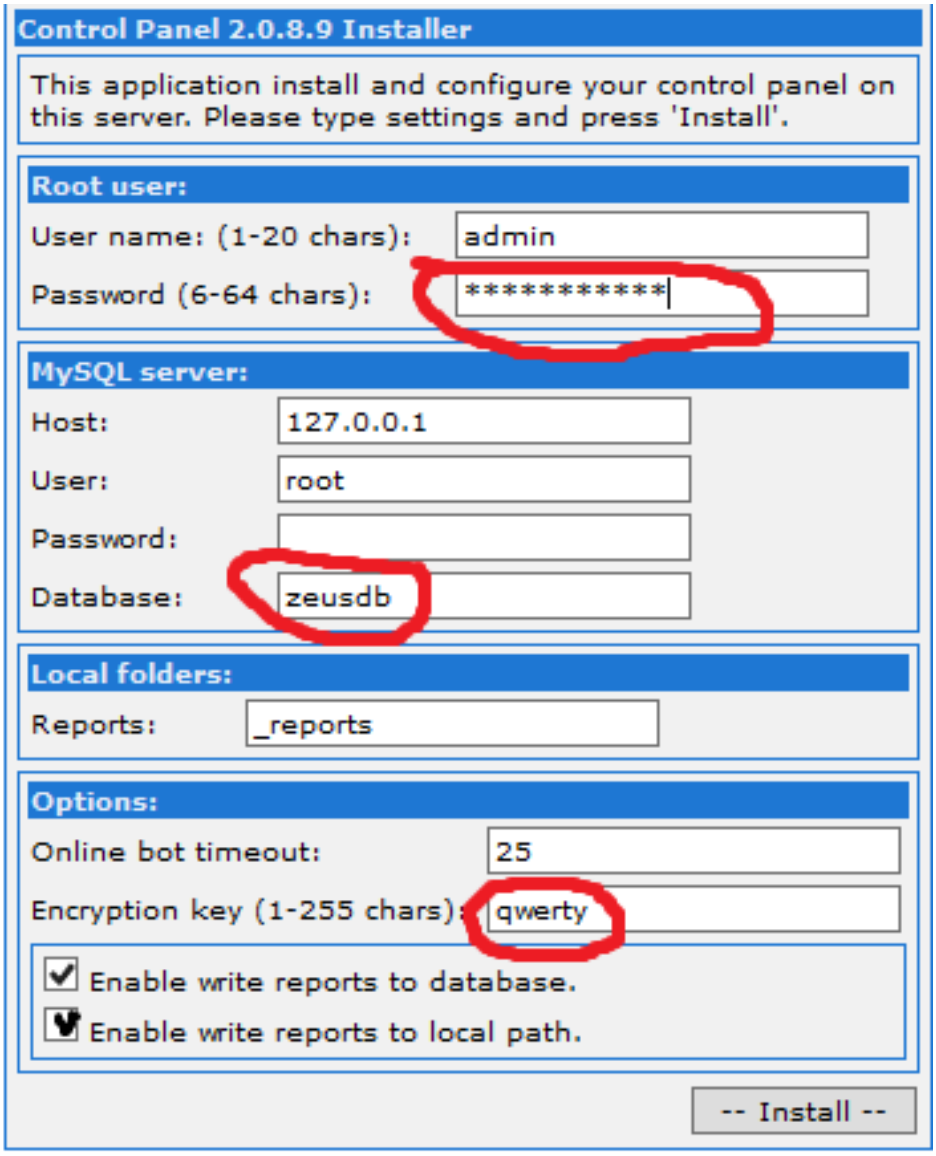

Figure 5.4: Zeus Control Panel Installer

First it will ask us for user name and password that we entered in the Installer. After we logged in to our Control Panel, the initial page with Summary statistics is displayed (Figure 5.5). On the left side, there is a menu where various pages can be accessed. On the right - the summary

of information about the botnet. Now, after we have our Control Panel running, we need to start building bots. The next section describes this process.

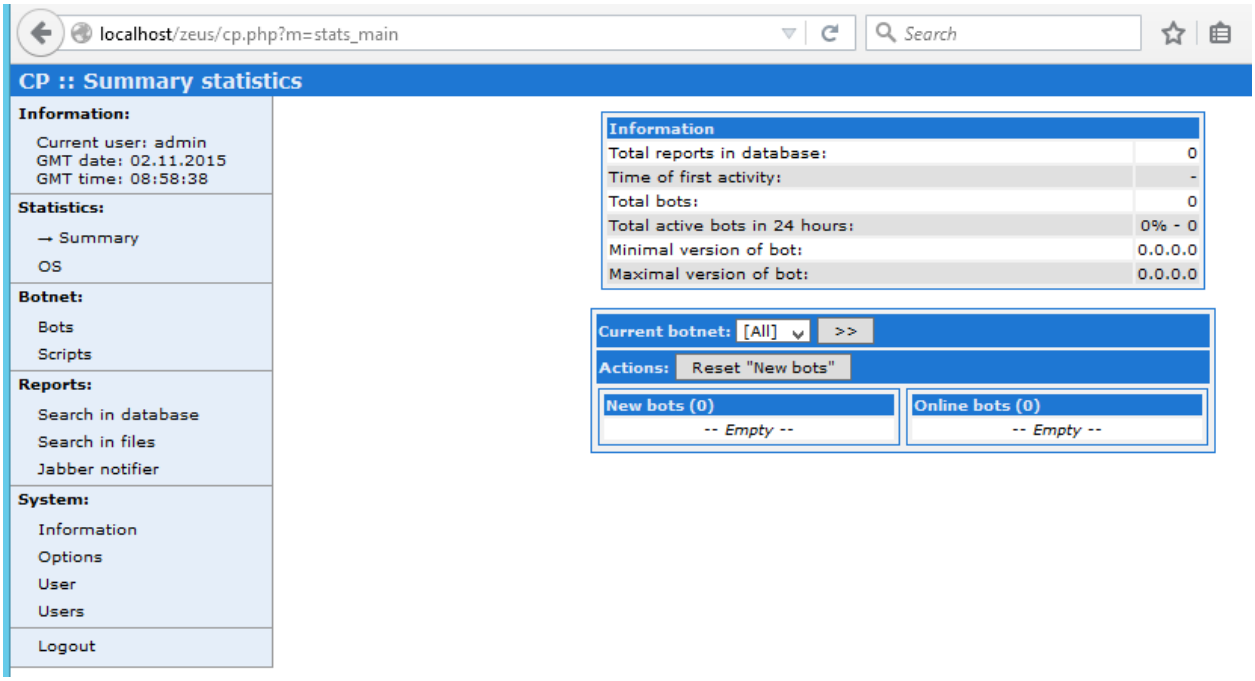

Figure 5.5: Zeus Control Panel

5.1.3 Building the Zeus Bot

Before we start building our bot, we must first edit the configuration file shown in Appendix A. The file has 2 main sections: StaicConfig and DynamicConfig. The StaticConfig is compiled into the bot by the Builder tool. It contains the information that the bot will need when it is first executed. The available settings are:

- botnet: The name of the botnet that this bot belongs to. The name field is commented out.
- timer_config: The amount of time to wait between dynamic configuration file downloads.
- timer_logs: The time interval between uploads of logs and statistical information to the server.
- timer_stats: The time interval between uploads of statistics to the server (includes open ports, sockets, services, etc.).
- url_config: The URL where the bot can get the dynamic configuration file (config.bin).

- The encryption key that is used to hide information transmitted within the botnet.
- blacklist_languages: A language ID list that tells the bot to go into a dormant state if the infected computers language is on the list.

The DynamicConfig is downloaded by the bot immediately after it is installed on a victim's computer. This file is downloaded at timed intervals by the bot, and can be used to change the behavior of the botnet. Available settings include:

- url_loader: A URL where the bot can use to update itself.
- urs_server: The URL of the server where logs, statistics and files will be uploaded and stored.
- file_webinjects: Information used to inject additional fields into web pages.
- Section AdvancedConfigs: A list of URLs which hold the backup configuration file.
- Section WebFilter: A set of URL masks used to cause or prevent logging of information.
- Section WebDataFilters: A set of URL masks to indicate that a screen image should be saved if the left mouse button is clicked.
- Section WebFakes: A list of pairs of URLs that are used to cause redirection from the first URL to the second.
- A set of URL masks used to collect TAN (Transaction Authentication) numbers used by some banks for online authentication.
- A list of IP/URL pairs that are inserted into the infected computer's hosts file to override DNS lookups.

Using this information, we substitute default values with our values, including the previously generated encrypton key "qwerty." At this point we are ready to build our first bot. We open the Zeus Builder tool and navigate to Builder tab. First, we are building the bot configuration. It will compile the configuration file into its encrypted form, the config.bin, which is populated with URLs for web injections (Figure 5.6). When it is ready, it is placed on the server, where all bots on botnet will look for the DynamicConfig.

The second stage, is to build the bot. Using the Zeus Builder, the beginner bot master can click "Build the bot executable," and, if successful, the distributable form of the bot will be assembled

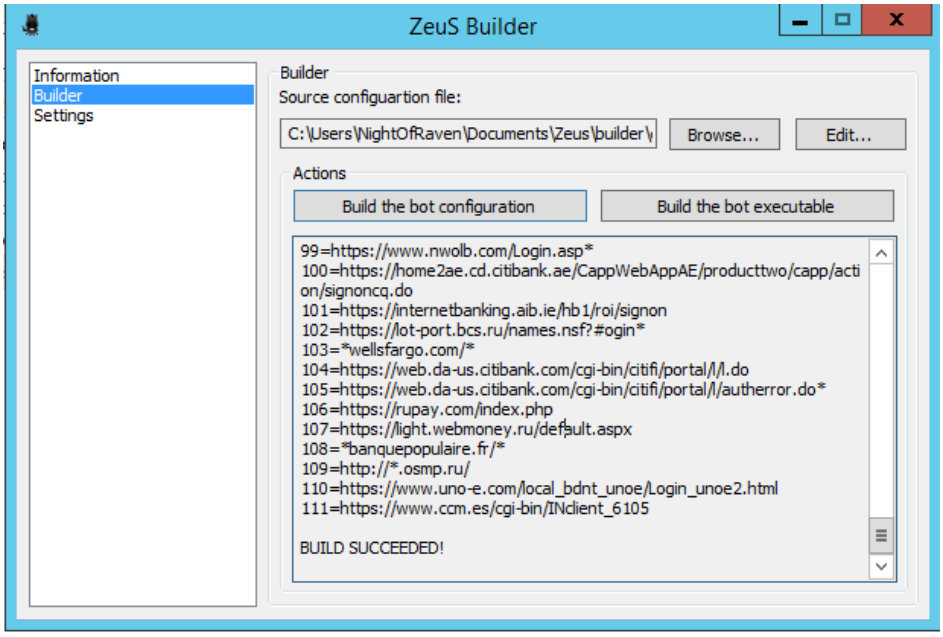

Figure 5.6: Building Configuration for Zeus Bot

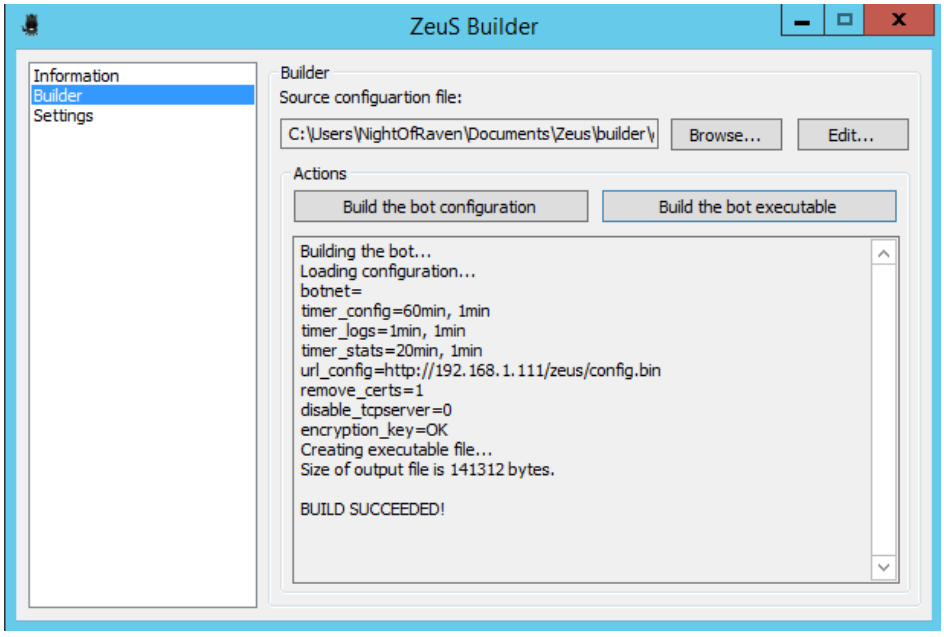

Figure 5.7: Building Zeus Bot

and saved. We generated bot.exe, as also mentioned in the configuration file in Appendix A. Each new bot generated will be assembled with new encryption and identification.

At this point we have set up our server and built our bot, and can start the propagation of our botnet.

5.2 Network Laboratory Configuration

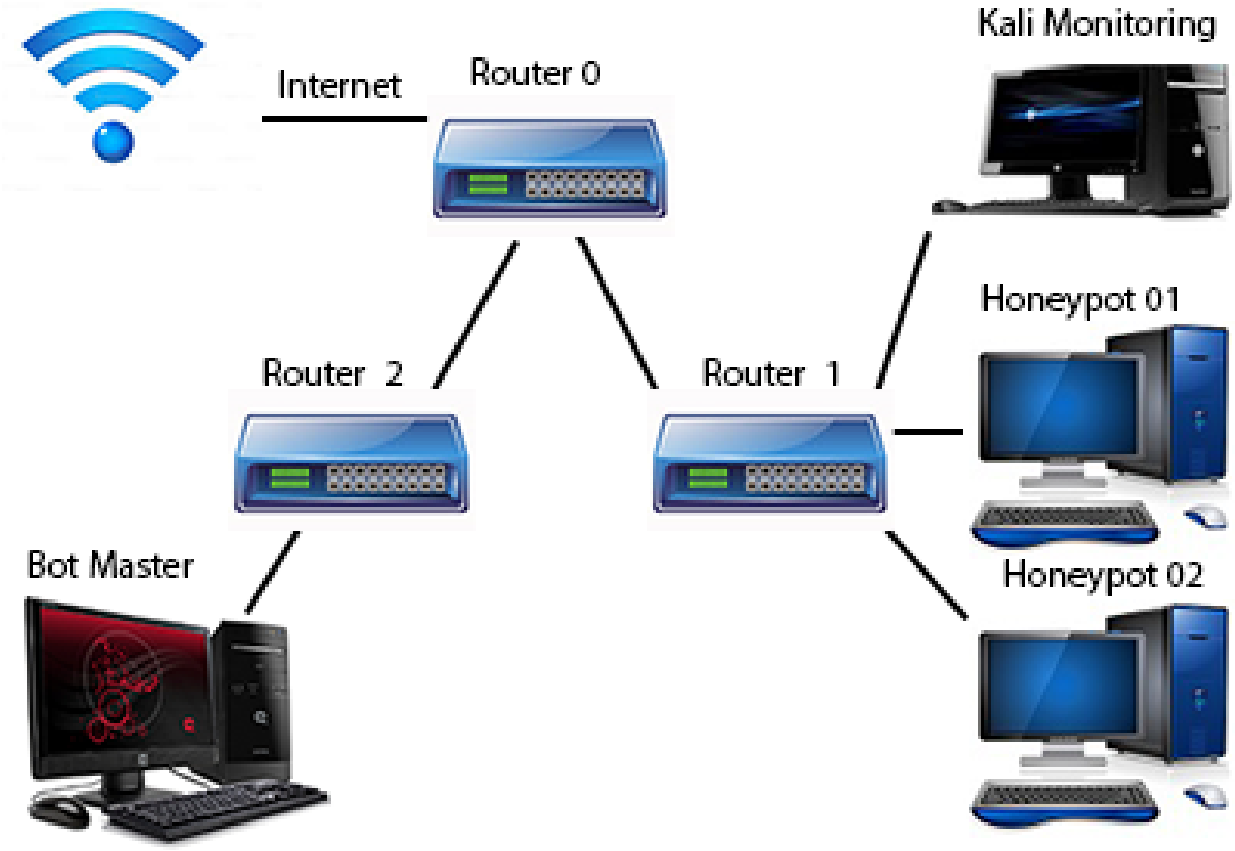

Figure 5.8: Network Configuration

The virtual network is the heart of the experiment. Following is the complete description of the network that was setup for the lab experiment.

- There are 2 routers that make up the network infrastructure to emulate 2 separate subnets, one for the bot master (attacker), and one for the Honeynet.

- The virtualization hardware platform was an Asus ROG G750JH running Microsoft Windows 8.1 Pro. The laptop was loaded with Intel Core I7 2.4 GHz (3.2 GHz Boost) and 24 GB 1600 MHz DDR3 memory.
- The virtualization software used is VMware Workstation 12.0 Pro.
- The volatility framework was used to analyze the state of honeypots before and after an infection.
- The 2 routers are built using PfSense 2.2.5 configured with 256 MB RAM and 2 GB HDD each.
- Behind each of the routers above are the following subnets:

  The **Bot Master** machine, behind the router designated as Router 2

  Microsoft Windows Server 2012 R2 , 2 core processor with 4 GB RAM and 60 GB HDD

  Visual Studio 2010 Ultimate

  Visual C++ 9.0 with Service Pack 1

  The Platform SDK

  Mozilla Firefox (current version), Google Chrome (current version)

  XAMPP for Windows v5.6.14 with PHP 5.6.14, MySQL, PHP my Admin, and Apache

  Zeus source code version 2.0.8.9

  The **Kali Linux Honeynet Monitoring**

  A component of honeynet subnet behind the Router 1

  Debian Release Kali Linux 2.0 64-bit

  Single core processor with 2 GB RAM and 25 GB HDD

  Executed an apt-get install update after installation

  Primarily used a combination of Wireshark, nmap, and tcpdump

  The **Honeypot 02**, a component of the Honeynet behind the Router 1

  Microsoft Windows 10 Pro 64-bit with all updates installed

  Configured with 2 core processor, 4 GB RAM, and 100 GB HDD

  Mozilla Firefox (current version), Google Chrome (current version)

  The **Honeypot 01**, a component of the Honeynet behind the Router 1

  Microsoft Windows Seven Pro 64-bit with all updates installed

  Configured with 2 core processor, 2 GB RAM, and 60 GB HDD

Additional honeypot (not on the network)

Microsoft Windows XP Pro with Service Pack 3 and all updates installed

Single core processor, 1 GB RAM, and 20 GB HDD

Mozilla Firefox (current version), Google Chrome (current version)

Additional identical honeypots can be added to the network as needed

5.2.1 Routers

The PfSense routers are the foundation of the network. Each router has 2 network interfaces - an inward facing adapter, and an outward facing adapter. The outward facing adapter is facing Router 0 on the diagram, which is designed to simulate the Internet). All adapters are created using Host Only configuration in VMware Workstation as this adds a measure of safety when the live malware is executing. The router configuration enables the project to have a bot master and the Honeynet (bots) nodes each on their own separate subnet. This is a realistic simulation of what a configuration of a life botnet would look like in real world.

There is one issue that we want to address in this section - thee time frame for the experiment. The technology of Zeus, as the variant is used in this project is from approximate time frame from 2007 to 2009. Since the source code was released in 2011 [65], There were many variations built off released version. The documentation, released with the source code, has not be updated since 2011, but, unlike some other kits, the source code can be built using any version of Visual C++, starting with version 9. The platform SDK is helpful in dating the software and activity, because Microsoft produced the development kit necessary to build the software with Windows Seven, and using newer versions of Windows (Server 2012 R2, Windows 8, 8.1, 10) proved to be inefficient in building the source code. The platform SDK we used to build the Zeus was obtained from Windows Seven distribution. All of the Microsoft development tools used in out experiment were from the same distribution of Windows.

Where this background on the tools used to build the botnet become important to the network configuration is in the area of the migration of networking practices over the years from use of hubs to one where more switches and routers are used [62].

If one of the bots is commanded to put the VMs network adapter into promiscuous mode, and that node is attached to a simple hub, then network traffic intended for all ports can be sniffed and analyzed [62]. This particular behavior might not apply to most botnets, but applies to any

network topology that uses hubs as opposed to switches [62]. Such behavior might allow the bot master to have access to a more inclusive collision domain on the network.Using switches changes this scenario to a completely different one, as, even if network adapter is put into promiscuous mode, the network traffic, that bot master can sniff, will be localized to one collision domain of the infected node. "The intelligence built into most modern switches will allow two ports (on the switch) to communicate across the switch without broadcasting that traffic to other ports on the switch" [62]. We will be using this configuration setup in the lab experiment presented in this research. This decision means a more realistic setup for a bot master operating Zeus today, and would make this research a more relevant to today's networks.

5.2.2 Honeynet

The honeypots and resulting honeynets that make up the network are some of the most basic in design that could be used. This decision wa important to the overall design of the experiment. Although several popular honeypot implementations were considered (such as Specter, jailing, and Honeyd [59]), in the end they were discarded in favor of reducing the complexity and increasing the amount of control we have over the network. We did not to practice real deception to produce the results for our deception system [57]. The system is designed to simulate a botnet attack and track network communication to and from the bot master, and to observe what communication attempts are made back to the botmaster and what information bot master can obtain from the infected nodes. We also observe the infection behavior of the Zeus botnet and what changes it does to the system. The actual value and activity of the honeypot are not critical to the research.

The honeypot implementation that we selected was simple and contained 3 different operating systems to test the validity of the Zeus botnet in today's world. First operating system we selected was Windows 7 Pro 64-bit, which is the most popular operating system today [44]. All updates up to the beginning date of the experiment were installed. Additional software to this machine was limited to 2 current browsers - Firefox and Chrome and WinDump (the Windows version of tcpdump) to provide a direct view of the network traffic on VM's network interface. Second operating system used in our experiment was Windows 10 Pro 64-bit, which is a new operating system released by Microsoft on July 29th 2015 [40]. This operating system is gaining popularity among the users very quickly, and, by November 2015, it already has 8% market share of all Operating Systems, behind Windows 7 with 55% and Windows XP with 11% [44]. The system

is gaining popularity and therefore was an important part of this research. Additional software installed on this machine was the same as Windows 7. Third and last operating system used in this project was Windows XP. Although this operating system is still surprisingly popular, it was not a part of the network setup as we had a limited amount of time for the research.

In addition to the internal monitoring of honeypots we set up Kali Linux to monitor the activity of the botnets and bot master on the network. This system was used for more in depth monitoring of the data and packet inspection. The noise was introduced into the network traffic to simulate the activity of a real network using nmap, while at the same time writing all network traffic using tcpdump into pcap files for further analysis by the Wireshark. We also used Wireshark in live capture mode to monitor the results of the commands sent by bot master to nodes and inspect the information bots send back.

5.2.3 Tracking Strategy

The experiment relies on three basic steps introduced by Provos and Holz [52]. First, a honeypot was used to collect samples of malware, which only be simulated, as this is not a part of the research. We did, however, monitor the activity of the botnet by monitoring the communication channels. Since we were not collecting the malware, there was no need for specific programs to analyze the collected malware sample. We used volatility framework to monitor the effects that malware has on the operating system. We relied on Kali Linux monitoring tools and the communications between bot master and infected systems to verify the malicious traffic. Further observation of malware network activity will be monitored using WinDump, tcpdump, and Wireshark. The main goal of monitoring and gathering the information about the malware and its communications is to be able to shut down the C2 server that botnet is using.

This strategy can be summarized as:

- Collect malware
- Observe behavior
- Extract information
- Observe communication channels
- Shutdown the C2 server

The information we will be able to gather to shut down the C2 includes DNS location and the IP addresses of the server, as well as ports used. According to Holz and Provos [52], individual bots on the botnet have identification names just as nodes on the network do. This information will be useful as bots contain a structure with the information about communication channels and any authentication information that might be used [52].

Using this methodology, we can observe that disruption or blocking of some communications channels, such as HTTP and P2P, can be proven very difficult. Pairing this with the fact that disruption of one bot does nothing to stop the overall attack of the botnet and may in fact alert the bot master about the presence of a honeypot and revealing its true purpose, thus leaving us with a no longer viable tracking method. In fact, many botnet tracking methods based on strategies that are covert and originate from specific botnet implementations and, therefore, individually tailored to the type of a botnet and focus on a specific aspect of the bot, which are either implementation, design, configuration, deployment, and location [52].

Even though the experiment is a simulation of a Honeynet being attacked by a botnet, we must address the issue of social engineering, which is the primary method for spreading the Zeus Bot. According to Lance Spitzner [35][60], there is a correlation between the experience of a bot master and the time spent by him communicating with his botnet - experienced bot masters would spend less time issuing commands to their botnets. This strategy mimics the military behavior - less communication means less exposure of a botnet, which increases the chances of successful integration of a botnet and hiding it on the network [35].

5.3 Approach

To prove the validity of the project as a honeynet hardening technique, we decided on a two step approach. First, we will implement a solution to demonstrate hardening of the honeynet against a honeynet-aware botnets. Then, we will show how propagation of the botnet from one honeypot to another can be a sufficient deception mechanism. The first step is to implement the system proposed by Wang et al. [68] to show that the botnets can check the compromised bots and see if the nodes are still a part of the botnet. If a particular bot does not obey the commands or acting in a suspicious manner, then the bot master may suspect a suspicious environment like a honeypot and direct the malicious activity away from the honeypot. The compromised bots are also tested

by the bot master to verify if the bot is obeying commands received through C2 from the master and carrying out those commands [68].

We will setup our virtual network for a botnet attack using Zeus Bot, which can be deployed for several attacks, such as stealing credentials and sending phishing e-mails [33]. Zeus uses a drive-by download type of spreading mechanism, which relies on users to click a malicious link in the e-mail [33].

The following list of actions are a preliminary setup to use in this project:

- An assumed strategy is for Zeus to attack a Windows 7 workstation

  Honeypot 01 on a diagram

- User of Honeypot 01 has clicked on a malicious link in a phishing e-mail that executes a call back

  The callback will request a download of the bot executable

  The above download is unknown to the user

  The user executes the bot thinking it's a legitimate file

  The Honeypot 01 is now infected.

To show that our approach is valid, two scenarios were created. First (Figure 5.9), we show how honeypot, that controls the traffic, is detected and the attack stops. Second (Figure 5.10), we show that with the use of 2 or more honeypots we can trick the attacker to think that the network is legitimate and to continue the communication with his botnet.

5.3.1 Discussion of Case 1

- **Honeypot 01** responds to the commands from Bot Master
- **Honeypot 01** acts on Bot Master's commands and attempts to spread the malicious traffic
- The malicious traffic is contained by the **Honeypot 01**
- **Honeypot 02** does not receive any malicious traffic
- Bot Master detects that malicious traffic does not propagate on the network
- Bot Master detects suspects that the infected machine is a honeypot
- Bot Master ceases the attack

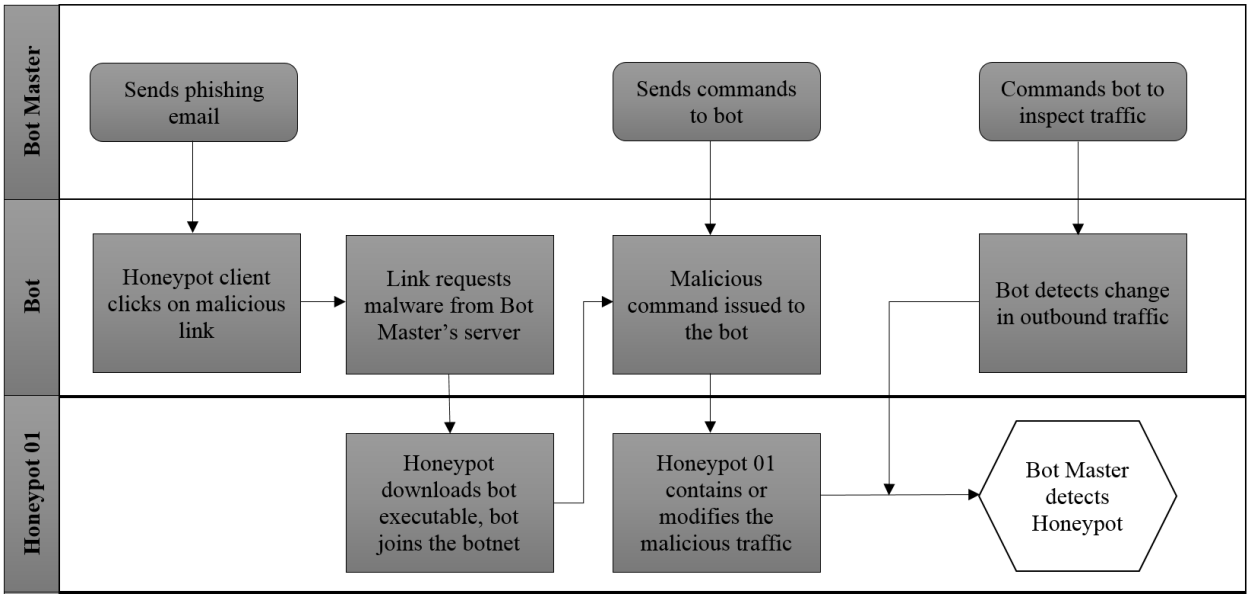

Figure 5.9: Case 1

The Case 1 implements the attack on the honeynet without hardening. The Honeynet was infected by the botnet, but, since honeypot was designed to contain the traffic, the bot master was able to detect that the infected machine is a honeypot and remove the bot [67].

The botnet attacked Honeypot 01, which doesn't contain any deception system and does not hide the fact that it is a honeypot from the bot master. Naturally, the bot master attempts to propagate the malicious traffic on the network by sending the commands to the infected node. Without any deception systems in place, the Honeypot 01 will act as a honeypot should and will not allow any traffic to propagate from it. At this point the bot master will notice a suspicious behavior of the infected node, and, with only little effort invested, will detect a honeypot, and shift his attack away from our network. When this happens, we lose every opportunity to gather any data about communication channels, malicious activity, as well as to analyze the method used by the bot master to infiltrate the network. This defeats the true purpose of the honeypot - to learn as much as possible about the ways and means of the enemy [60].

5.3.2 Discussion of Case 2

- **Honeypot 01** responds to the commands from Bot Master
- **Honeypot 01** acts on Bot Master's commands and attempts to spread the malicious traffic

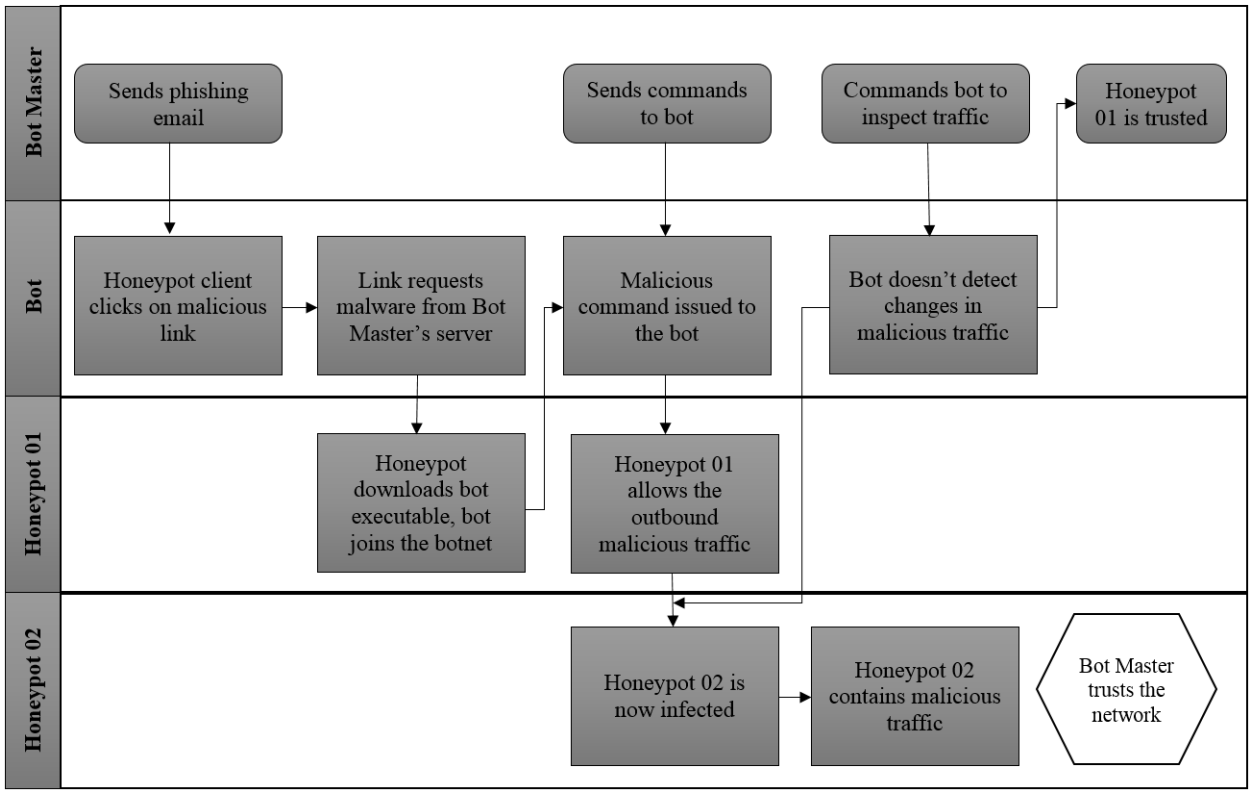

Figure 5.10: Case 2

- The malicious traffic is propagated on the network by the **Honeypot 01**
- **Honeypot 02** now able to receive the commands from Bot Master
- Bot Master now inspects the traffic from botnet on **Honeypot 01**
- **Honeypot 01** allows outbound and unaltered traffic on the network
- At this point there are 2 possibilities to consider:

  Bot Master does not suspect the honeypot and the attack continues

  More information is gathered about the attacker and the vector of attack

  Bot Master suspects Honeypot 01

  The Bot Master must inspect outbound traffic from **Honeypot 02**

  Bot Master's work to detect suspicious environment has increased by the factor of 2

  Detection is more difficult

The second case has the same initial steps as Case 1 for the setup. The network setup is identical. The additional honeypot, used in Case 2, was not physically used in Case 1.

Case 2 is an extension of the Case 1 experiment with some additional alterations at the end, and resulting hardening of the Honeynet by increasing the amount of manual work the bot master must perform in order to monitor bot in his botnet. The botnet is spread on the honeynet and the bot master watches his bots are obeying the commands and the malicious traffic is spreading without modification, thus not detecting honeypots [63].

The botnet attacks Honeypot 01, which now has an additional deception system in place to attempt to hide the honeypot from the bot master. The bot master will attempt to monitor the outbound traffic on the first infected honeypot, and, with additional deception in place, the honeypot will not implement the data containment of many honeypot designs and will allow the malicious traffic to pass through on honeynet. By observing the outbound traffic, the bot master will note that the infected environment is not suspicious and behaving as a normal system would, thus will not suspect that the infected node is a honeypot. At this point the bot master would have an invested interest in the network and will likely remain focused on it [63]. With this, the opportunity to capture more malicious data, malicious traffic, as well as the means of attack is not lost due to the deception provided by the first honeypot. Because the bot master has invested time on the network and has been deceived into thinking that infected bots provide the opportunity for collecting user data, he will likely to continue to work on an infected honeynet, which was not identified as one, but rather identified as a friendly and vulnerable network [68]. Unlike Case 1, Case 2 is considered a winning situation for us, because it represents an opportunity for a honeynet to learn as much as possible about the ways and means of the attacker [60].

5.3.3 Operation of Zeus details

After infection, bots connect to the server's Control Panel at a predefined port. The attacker can issue commands through control panel, as well as upload additional scripts to perform extra operations, like network sniffing, bot removal, remote administration of the infected machine with the ability to reboot or shutdown. Out of the box capabilities of the Zeus involve web injections, keylogging, ability to take screen shots at any time or through specified time interval, grabbing web browser cookies, gathering windows version, language, location (if not used locally), and time zone.

The Control Panel has extensive list of functions to work with botnets. Statistics displayed by the Control Panel includes number of infected computers, number of bots on-line, number of new bots, daily activity of the bots, statistics by the country and by version of the operating system. When working with bots Control Panel provides several functionalities:

- Filtering the list by country, botnets, IP-addresses, NAT-status, etc
- Displaying desktop screenshots in real time
- Mass inspection of the Socks-servers state
- Displays detailed information about the bots. Of the most important here are:

  Windows version, user language and time zone

  Location and computer IP-address (not for local)

  Internet connection speed (measured by calculating the load time of a predetermined HTTP-resource)

  The first and last time of communication with the server

  Time in online
- Ability to set comment for each bot

Each bot sends reports to the server that are stored in the database, which can be filtered by bots, botnets, or content. The list goes on, but shows the extensibility of the malware.

CHAPTER 6

CONCLUSIONS

6.1 Application

We have shown an example of honeynet deployment and what results we were able to collect from our research. We provided a method to improve a botnet mitigation through the use of the hardened honeynet. Our procedure accomplishes this by increasing the amount of manual and time intensive labor needed by the attacker to detect a honeypot from a regular infected system. It is important to note that this detection and evaluation is possible after the botnet has exposed itself through infection and initial download of the executable.

Further hardening of the honeynet against detection by the botnet was accomplished by allowing the propagation of the malicious traffic unmodified and without restrictions. We showed how this step increases the workload of the bot master.

6.2 Future Work

The malware community is already developing techniques that allow the bot master to easily detect honeypots and any virtualization techniques, and it seems reasonable to expand our project to create extra levels of deception for an attacker. We feel that a lot of future work can be done in the area of further developing the hardening techniques for honeypots. In our future work, we plan to adopt the solution proposed by Wang et al., the "double honeypot" [63], and expand this technique to work effectively against honeypot-aware botnets, which might include developing an intelligent algorithm to allow the malicious traffic to propagate on the honeynet without modification. We feel that this technique is a viable approach for developing a new deception levels for honeynets.

Since virtualization software was not designed to be a security tool, attackers are able to easily detect virtualization techniques [53]. With this in mind, we plan to expand the project and move from using virtual machines to using physical nodes, which, we believe, would further increase the deception levels of honeynets.

APPENDIX A

ZEUS BUILDER CONFIGURATION FILE

```
;Build time:    09:20:23 01.11.2015 GMT
;Version:       2.0.8.9

entry "StaticConfig"
        ;botnet "btn1"
        timer_config 60 1
        timer_logs 1 1
        timer_stats 20 1
        url_config "http://192.168.1.111/zeus/config.bin"
        remove_certs 1
        disable_tcpserver 0
        encryption_key "qwerty"
end

entry "DynamicConfig"
        url_loader "http://192.168.1.111/zeus/bot.exe"
        url_server "http://192.168.1.111/zeus/gate.php"
        file_webinjects "C:\Users\NightOfRaven\Documents\Zeus\builder\webinjects.txt"
        entry "AdvancedConfigs"
                ;"http://advdomain/cfg1.bin"
        end
        entry "WebFilters"
                "!*.microsoft.com/*"
                "!http://*myspace.com*"
                "https://www.gruposantander.es/*"
```

```
            "!http://*odnoklassniki.ru/*"
            "!http://vkontakte.ru/*"
            "@*/login.osmp.ru/*"
           "@*/atl.osmp.ru/*"
        end
        entry "WebDataFilters"
            ;"http://mail.rambler.ru/*" "passw;login"
        end
        entry "WebFakes"
            ;"http://www.google.com" "http://www.yahoo.com" "GP" "" ""
        end
end
```

APPENDIX B

TECHNICAL DIFFICULTIES ENCOUNTERED DURING THE EXPERIMENT

The experiment proved difficult to implement in practice. Several problems provided barriers to getting the implementation up and running, and as with any technical challenge, i personally found my greatest amount of knowledge gain in this area.

The network infrastructure provided a challenge. We could have used more than one physical machine and separate routers, etc., but combining all the VMs on one machine made the early stages of the experiment easier in terms of portability. As the experiment grew in the size and number of VMs we were using, there was no compelling reason to move to another type of deployment, and the whole implementation was left on the ASUS ROG laptop.

We began the project using Hyper-V for the virtualization software, and then switched to VMware Workstation professional for the completion of the project. The increase in performance of VMware compared to the use of Hyper-V was remarkable, but it only became apparent what the performance capabilities of the two systems were after we had tried to run all the nodes of the system together. This performance evaluation of the two software packages was a result of the gradual refinement of the individual requirements that each node in the system needed in terms of the resources.

The use of Hyper-V at the outset of the configuration of the lab and experiments has proven to be a time consuming task due to a few hiccups in the software. Mid-project, the switch was made to VMware Workstation and the remainder of the project went smoother, implementation-wise, than the first part that relied on Hyper-V.

The problem encountered using Hyper-V were all but eliminated by switching to VMware Workstation. This change was made in order to facilitate the completion of the project because of the technical difficulties with networking, but a side benefit was the increase in performance of VMware emulated machines. The VMware software performs faster and better on the ASUS laptop than Hyper-V.

Any machine placed on a network with no production purpose and for the reason of deception is considered a honeypot. There are no false positives and any traffic to and from that node is considered suspicious. We considered to use different honeypot designs, like Honeyd and Cuckoo, but abandoned it because our implementation did not rely on the specific design of a honeypot. Instead, we decided to build a honeynet with real systems (Windows XP, 7, 10), which allowed us reduce the complexity of the design and focus more on the scope of the project.

The variant of the Zeus Botnet we had access to is designed as Zeus 2.0.8.9. The build instructions were provided with the documentation and required Visual C++ 9.0 to compile. At first, we tried using Visual Studio 2013 Ultimate to compile the code, but we were missing several files from SDK that were designed for Visual C++ 9.0. We acquired the copy of Visual Studio 2010 Ultimate with Service Pack 1 from Microsoft. Service Pack 1 was required, because the linker used in Visual Studio 2010 was outdated. We compiled the source code on the Windows 7 machine, because SDKs used in newer version of Windows were incompatible with the source code. Finding the correct software and SKDs proved more difficult that we had anticipated. We did not realize that using an older compiler and linker would be necessary. The source code itself provided several barriers as it had multiple errors in it. We had to use Visual Studio debugger to find and fix those errors, which took a lot of time of reading and analyzing the code. After compiling the code, we had a working version of a Zeus Botnet Builder tool on our hands.

APPENDIX C

WINDOWS 10 FINDINGS DURING THE EXPERIMENT

Windows 10 was chosen for this experiment as it is a new operating system from Microsoft and gaining popularity quickly.[40][44]

During our experiment the Zeus Botnet was not able to install on Windows 10, we concluded that it is probably due to changes in APIs used in Windows 10, that must be different from the ones used in Windows 7 and earlier versions of Windows. We were able to collect and analyze the network data using Wireshark, and this data revealed the list of different services that Windows 10 sends a large amount of data to. Those were identified as telemetry services of Windows 10, that gather data and send it to Microsoft. The most active service was found to be Cortana - Windows 10 personal assistant.[39] Our analysis revealed the every 30 minutes, Windows connects and sends about 80 Megabytes of collected data to following Microsoft servers:

- oca.telemetry.microsoft.com.nsatc.net
- pre.footprintpredict.com
- reports.wes.df.telemetry.microsoft.com
- vortex.data.microsoft.com
- vortex-win.data.microsoft.com
- telecommand.telemetry.microsoft.com
- telecommand.telemetry.microsoft.com.nsatc.net
- oca.telemetry.microsoft.com
- oca.telemetry.microsoft.com.nsatc.net
- sqm.telemetry.microsoft.com
- sqm.telemetry.microsoft.com.nsatc.net

- df.telemetry.microsoft.com
- reports.wes.df.telemetry.microsoft.com
- cs1.wpc.v0cdn.net
- vortex-sandbox.data.microsoft.com
- pre.footprintpredict.com

We did further research on the matter and found that the list of the Microsoft servers is more extensive than what we have found.[8] Windows 10 sends different type of information to multiple servers. It sent the video file data from the camera that was connected to the virtual machine from the laptop, it sends audio files recorded from communications, text tiles, and other information that it can gather.[8]

We turned off Cortana from settings, but connections to some of these servers continued to appear in our network traffic data. Restoring the virtual machine state to the moment right after the installation helped, and Windows 10 made only several connections to Microsoft websites, which turned out to be Windows Update services. We concluded that Cortana will be disabled and will not send information to Microsoft if the user is not signed in with Microsoft account, but uses local account instead.

We then tried blocking Cortana services using built in Windows Firewall. The list of 55 firewall rules was created using the names of servers from the list[8] to produce the following results in the firewall (Figure C.1).

These rules have helped disable the communications between Cortana and Microsoft, but have disabled almost all functionality of Cortana, because it relies on the Internet access and the information previously sent to Microsoft to provide personalized assistance. After further experimentation with Windows 10 and Cortana, we have concluded that Cortana exhibits similar behavior to that of a botnet.

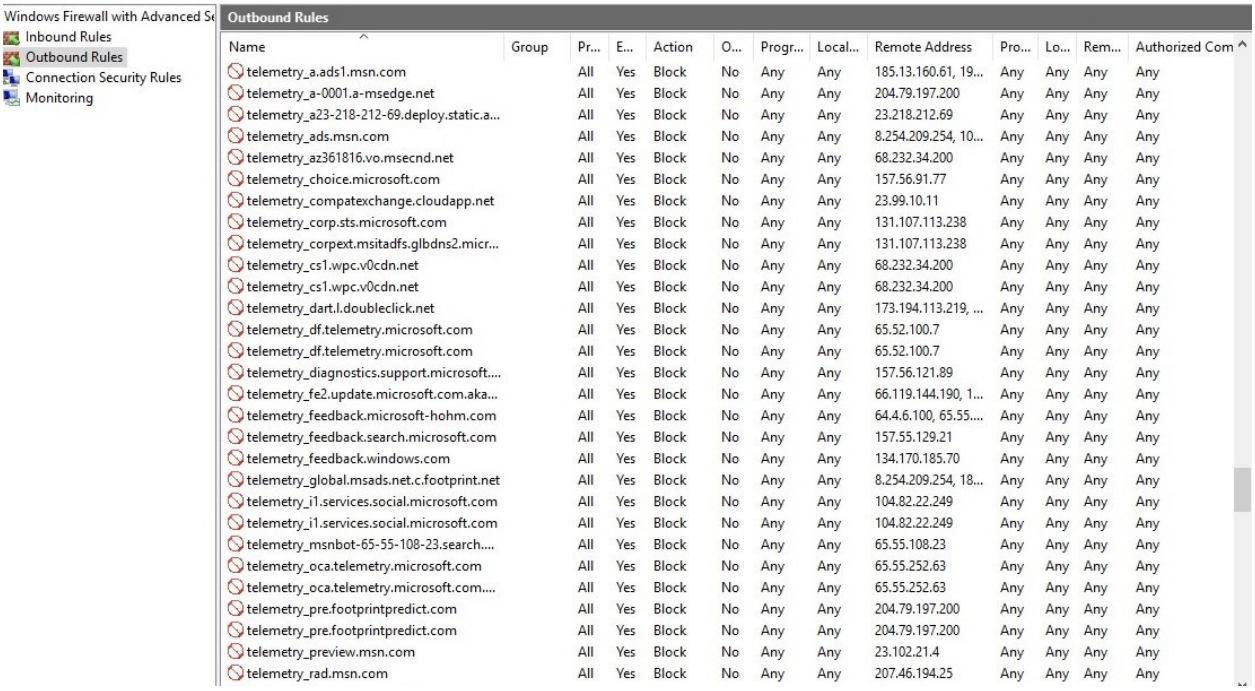

Figure C.1: Firewall Rules

REFERENCES

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

[2] Inc. Anchor Intelligence. `http://www.anchorintelligence.com`.

[3] av test.gov. Year-end malware stats from av-test, 2015. `http://www.av-test.gov`.

[4] Paul Bacher, Thorsten Holz, Markus Kotter, and Georg Wicherski. Know your enemy: Tracking botnets, 2005.

[5] Michael Bailey, Evan Cooke, Farnam Jahanian, Yunjing Xu, and Manish Karir. A survey of botnet technology and defenses. In *Conference For Homeland Security, 2009. CATCH'09. Cybersecurity Applications & Technology*, pages 299–304. IEEE, 2009.

[6] Paul Barford and Vinod Yegneswaran. An inside look at botnets. In *Malware Detection*, pages 171–191. Springer, 2007.

[7] Johannes M Bauer, Michel JG Van Eeten, and T Chattopadhyay. Itu study on the financial aspects of network security: Malware and spam. *ICT Applications and Cybersecurity Division, International Telecommunication Union, Final Report, July*, 2008.

[8] Peter Bright. Even when told not to, windows 10 just cant stop talking to microsoft, 2015. `http://arstechnica.co.uk/information-technology/2015/08/even-when-told-not-to-windows-10-just-cant-stop-talking-to-microsoft/`.

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

[10] Wentao Chang, Aziz Mohaisen, An Wang, and Songqing Chen. Measuring botnets in the wild: Some new trends. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, pages 645–650. ACM, 2015.

[11] Inc. ClickForensics. `www.clickforenics.com`.

[12] Evan Cooke, Farnam Jahanian, and Danny McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Proceedings of the USENIX SRUTI Workshop*, volume 39, page 44, 2005.

[13] Charles Costarella, Sam Chung, Barbara Endicott-Popovsky, and David Dittrich. Hardening honeynets against honeypot-aware botnet attacks.

[14] Neil Daswani and Michael Stoppelman. The anatomy of clickbot. a. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 11–11. USENIX Association, 2007.

[15] Ron Deibert and Rafal Rohozinski. Tracking ghostnet: Investigating a cyber espionage network. *Information Warfare Monitor*, page 6, 2009.

[16] Maximillian Dornseif, Thorsten Holz, and Und Sven Müller. Honeypots and limitations of deception. 2005.

[17] C Economics. Malware report: The economic impact of viruses, spyware, adware, botnets, and other malicious code. Technical report, Tech. Rep., Jun, 2007.

[18] Nicolas Falliere and Eric Chien. Zeus: King of the bots. *Symantec Security Response*, 2009.

[19] Nicolas Falliere, Liam O Murchu, and Eric Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5, 2011.

[20] Marc Fossi, Gerry Egan, Kevin Haley, Eric Johnson, Trevor Mack, Téo Adams, Joseph Blackbird, Mo King Low, Debbie Mazurek, David McKinney, et al. Symantec internet security threat report trends for 2010. *Volume*, 16:20, 2011.

[21] ShadowServer Foundation. `http://www.shadowserver.org`.

[22] Apache Friends. What is xampp?, 2015. `https://www.apachefriends.org/index.html`.

[23] Google. Google adsense. `http://www.google.com/adsense/start/#?modal_active=none`.

[24] Aleksandr Gostev, Oleg Zaitsev, Sergey Golovanov, and Vitaly Kamluk. Kaspersky security bulletin malware evolution 2008. *Kaspersky Lab (April 2009)*, 5, 2009.

[25] Julian B Grizzard, Vikram Sharma, Chris Nunnery, Brent ByungHoon Kang, and David Dagon. Peer-to-peer botnets: Overview and case study. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 1–1, 2007.

[26] Guofei Gu, Junjie Zhang, and Wenke Lee. Botsniffer: Detecting botnet command and control channels in network traffic. 2008.

[27] Ed Hagen Nathan Judish H. Marshall Jarett, Michael W. Bailie. Searching and seizing computers and obtaining electronic evidence in criminal investigations, 2015.

[28] Thorsten Holz, Markus Engelberth, and Felix Freiling. *Learning more about the underground economy: A case-study of keyloggers and dropzones.* Springer, 2009.

[29] Simon Innes and Craig Valli. Honeypots: How do you know when you are inside one? In *Australian Digital Forensics Conference*, page 28, 2006.

[30] Mr Dias Jose. Intrusion detection using honeypots and sniffers. *chance*, 1(1), 2015.

[31] Gregg Keizer. Is stuxnet the 'best' malware ever. *Computerworld, September*, 16, 2010.

[32] Won Kim, Ok-Ran Jeong, Chulyun Kim, and Jungmin So. On botnets. In *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services*, pages 5–10. ACM, 2010.

[33] Kaspersky Lab. Zeus trojan malware, 2015. `https://usa.kaspersky.com/internet-security-center/threats/zeus-trojan-malware-threat`.

[34] law.com. Entrapment, 2015. `http://dictionary.law.com/Default.aspx?selected=637`.

[35] Felix Leder and Tillmann Werner. Know your enemy: Containing conficker. *The Honeynet Project, University of Bonn, Germany, Tech. Rep*, 2009.

[36] DiMino A. M. Blackenergy competitor the 'darkness' ddos bot, 2010. `http://www.shadowserver.org/wiki/pmwiki.php/Calendar/20101205`.

[37] John F. Reiser Markus Franz Xaver Johannes Oberhumer, Lszl Molnr. Ultimate packer for executables, 2013. `http://upx.sourceforge.net/`.

[38] Geoff McDonald, Liam O Murchu, Stephen Doherty, and Eric Chien. Stuxnet 0.5: The missing link. *Symantec Report*, 2013.

[39] Microsoft. What is cortana?, 2015. `http://windows.microsoft.com/en-us/windows-10/getstarted-what-is-cortana`.

[40] Microsoft. Windows 10, July, 2015. `https://www.microsoft.com/en-us/windows/features`.

[41] Information Warfare Monitor and Shadowserver Foundation. Shadows in the cloud: An investigation into cyber espionage 2.0, April 6, 2010.

[42] Jose Nazario. Blackenergy ddos bot analysis. *Arbor Networks*, 2007.

[43] Jose Nazario. Politically motivated denial of service attacks. *The Virtual Battlefield: Perspectives on Cyber Warfare*, pages 163–181, 2009.

[44] NetMarketShare. Desktop operating system market share, October, 2015. `https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0`.

[45] Department of Justice. Pen register and trap and trace statute, 18 usc 3121-27, 2006. https://www.law.cornell.edu/uscode/text/18/3121.

[46] Department of Justice. Electronic communications privacy act, 18 usc 2510-22, 2013. https://it.ojp.gov/PrivacyLiberty/authorities/statutes/1285.

[47] Department of Justice. Wiretap act, 18 usc 2511, 2013. https://it.ojp.gov/PrivacyLiberty/authorities/statutes/1284.

[48] Richard A Paulson and James E Weber. Cyberextortion: an overview of distributed denial of service attacks against online gaming companies. *Issues in Information Systems*, 7(2):52–56, 2006.

[49] Igor Pavlov. 7-zip, 2010. http://www.7-zip.org/.

[50] Eduardo Fernandes Piva and Paulo Lıcio de Geus. Using virtal machines to increase honeypot security. *V Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, 2005.

[51] Daniel Plohmann, Elmar Gerhards-Padilla, and Felix Leder. Botnets: Detection, measurement, disinfection & defence. *The European Network and Information Security Agency (ENISA)*, 2011.

[52] Niels Provos and Thorsten Holz. *Virtual honeypots: from botnet tracking to intrusion detection*. Pearson Education, 2007.

[53] Thorsten Holz Frederic Raynal. Detecting honeypots and other suspicious environments. In *Workshop on Information Assurance and Security*, volume 1, page 1555, 2005.

[54] Bruce Schneier. *Secrets and lies: digital security in a networked world*. John Wiley & Sons, 2011.

[55] M86 Security. Spam statistic. https://www3.trustwave.com/support/labs/spam_statistics.asp.

[56] Somayeh Soltani, Seyed Amin Hosseini Seno, Maryam Nezhadkamali, and Rahmat Budiarto. A survey on real world botnets and detection mechanisms. *International Journal of Information and Network Security (IJINS)*, 3(2):116–127, 2014.

[57] L Spitzner. Know your enemy: Defining virtual honeynets. *web, January*, 2003.

[58] Lance Spitzner. Honeypots: Catching the insider threat. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pages 170–179. IEEE, 2003.

[59] Lance Spitzner. *Honeypots: tracking hackers*, volume 1. Addison-Wesley Reading, 2003.

[60] Lance Spitzner. Know your enemy: Honeynets. *Honeynet Project*, 2006.

[61] Lanz Spitzner. Know your enemy: Genii honeynets. *The Honeynet Alliance*, 2005.

[62] Andrew S Tanenbaum. Computer networks, 5-th edition. *ed: Prentice Hall*, 2012.

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

[64] Eneken Tikk, Kadri Kaska, and Liis Vihul. *International cyber incidents: Legal considerations*, volume 112. Cooperative Cyber Defence Centre of Excellence, 2010.

[65] Visgean. Zeus' source code, 2014. `https://github.com/Visgean/Zeus`.

[66] Ping Wang, Sherri Sparks, and Cliff C Zou. An advanced hybrid peer-to-peer botnet. *Dependable and Secure Computing, IEEE Transactions on*, 7(2):113–127, 2010.

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

[68] Wei Wang, Justin Bickford, Ilona Murynets, Ramesh Subbaraman, Andrea G Forte, and Gokul Singaraju. Catching the wily hacker: A multilayer deception system. In *Sarnoff Symposium (SARNOFF), 2012 35th IEEE*, pages 1–6. IEEE, 2012.

[69] Vinod Yegneswaran, Paul Barford, and Johannes Ullrich. Internet intrusions: global characteristics and prevalence. *ACM SIGMETRICS Performance Evaluation Review*, 31(1):138–147, 2003.

[70] Bu Zheng, Bueno Pedro, Kashyap Rahul, and Wosotowsky Adam. The new era of botnets. Technical report, 2010.

BIOGRAPHICAL SKETCH

Yarosalv Kechkin is a Scholarship for Service recipient studying computer science at Florida State University. His research interests include reverse engineering, networking, malware analysis, cyber security, and their intersections.