AN ANALYSIS OF A BOTNET TOOLKIT AND A FRAMEWORK FOR A DEFAMATION ATTACK

THOMAS ORMEROD

A THESIS

IN

THE CONCORDIA INSTITUTE FOR INFORMATION SYSTEMS ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF APPLIED SCIENCE IN INFORMATION SYSTEMS SECURITY

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

SEPTEMBER 2012

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Thomas Ormerod**

Entitled: **An Analysis of a Botnet Toolkit and a Framework for a Defamation Attack**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science in Information Systems Security

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Chadi Assi ______________________________ Chair

Dr. Khaled Galal ______________________________ Examiner

Dr. Amr Youssef ______________________________ Examiner

Dr. Lingyu Wang ______________________________ Supervisor

Dr. Mourad Debbabi ______________________________ Supervisor

Approved ____

Chair of Department or Graduate Program Director

____________ 20 ______ ______________________________

Dr. Robin A.L. Drew, Dean

Faculty of Engineering and Computer Science

ABSTRACT

An Analysis of a Botnet Toolkit and a Framework for a Defamation Attack

Thomas Ormerod

Botnets continue to undermine the security of the Internet. They have evolved in both their level of sophistication and ability to cause harm. A driving factor for the use of botnets is their ability to generate a profit using traditional malware attacks on a large scale. Attacks of particular concern are those that harvest credentials for online financial services and transactions. Even with advances in malware detection, botnets continues to infect systems with relative ease, obtaining victims' sensitive information for the attacker's gain.

In this thesis, we first analyze a popular toolkit, Zeus, used to create botnets. Zeus is primarily used to steal victims' sensitive information and it employs obfuscation techniques, making it a challenging and interesting case study. The analysis illustrates the added difficulty of obfuscation and the revealed botnet architecture once removed. In addition, we describe methods to extract the encryption key and communications. We also present a conceptual framework for defaming botnets through supplying falsified credentials to the operators of botnets. This framework targets botnet toolkits by diluting stolen credentials with false ones, and providing an opportunity for law enforcement to apprehend end-users. The overarching goal of our framework is to defame the toolkit used to perform the attack;

making it an unprofitable tool. Furthermore, we analyze the affect our framework will have on the Internet black market and illustrate the defamation approach using Zeus as a case study. Finally, we test behavioural methodologies aimed at identifying key information to be used in the defamation framework.

Acknowledgments

I would first like to thank my supervisors, Dr. Lingyu Wang and Dr. Mourad Debbabi, for their guidance and support throughout this research. They were instrumental in my success for which I am very appreciative.

I would also like to thank the other researchers at the National Cyber-Forensics Training Alliance (NCFTA) Canada and the other students in my research lab, all of whom were a great help with my studies; always ready to answer any questions I may have and provide helpful advice.

Finally, I would like to thank my family for their support while I pursued this degree across the country and especially my partner, Anne, whose tireless patience and support got me through the inevitable challenges that arose throughout my degree.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet is arguably one of the greatest achievements of mankind. There is little doubt of its importance for a business and the public's reliance on it to perform a number of tasks. From its infancy as a mechanism to share information between researchers, to its current form as a global platform used for email, advertising, commerce, communication, and gaming, it has become a vital resource to our daily lives. Unfortunately, this fact has been also recognized by criminals, who see the Internet as a less risky (than traditional crimes) and highly profitable mechanism to conduct illicit activities. Some examples of this type of crime include spamming, identity theft, and distributed denial of service (DDoS) [29] attacks.

Of particular concern is the ease with which identity theft can occur. Much of our daily activities involve online services such as email, Internet banking, and e-commerce. An attacker needs only steal the credentials used to access these services in order to assume a user's identity. The credentials used to access each of these online services is equally sensitive. With email credentials one gains knowledge of many services registered with that email address and, since many services offer password reset through email, attackers may gain access to those services as well. Theft of Internet banking credentials can lead to unauthorized transfers, and theft of credit card information can lead to traditional carding schemes and fraudulent purchases.

Botnets [30] are the primary platform used to perform these attacks. Botnets can perform a variety of Internet crimes at much larger volumes and have introduced a new business model where infected systems may be rented to third parties to conduct further illicit activities.

The credentials stolen by botnets may be sold in the black market of the Internet to those who specialize in extracting funds from these kinds of accounts. Sales generally take place on underground forums and Internet Relay Chat (IRC) servers. In the case of credentials for financial services, the purchaser makes use of money mules to siphon the funds. This scarce resource is often attracted through fake job offers where the *employee* is tasked with receiving the funds into their accounts, withdrawing the funds and transferring the funds to their employer, minus a commission, often through wire transfers.

Another traditional method used to perform identity theft is to direct users to phishing sites. A phishing site is usually a replicated version of a website, controlled by an attacker, whose sole purpose is to have a user enter login credentials into the, falsely represented, site and record those credentials. The attacker may then login to the real website and perform tasks as the user whose credentials were stolen. A user is usually directed to these phishing sites through links in emails. These emails employ social engineering to encourage the user to follow the link. Phishing sites continue to be an active problem; however, with advances in phishing site detection and alert-services, such as Google Safe Browsing [21] and Microsoft's SmartScreen Filter [40], users may better be able to protect their identities from these attacks. A second, more stealthy method to perform identity theft, is to infect victims' machines with malware, which has shown more resilience to countermeasures.

Further compounding the problem is the advent of botnet toolkits. A botnet toolkit is a suite of programs to build and operate a botnet. This adds new dynamics to the threat landscape. Where previously an attacker required some strong technical know-how to design and propagate a botnet, attackers can now purchase toolkits that are user-friendly, and require little technical knowledge from the user. One of the most infamous botnet toolkits is Zeus, which has become the number one botnet threat in America with 3.6 million compromised machines in the U.S. alone [37]. The Zeus botnet and its toolkit have become a favourite tool for Internet criminals due to its user friendly interface, its competitive price and its ability to easily facilitate identity theft activities. This toolkit allows a user to create customized malware for the purpose of stealing credit card numbers, Internet banking credentials and many other forms of identity information to be sold in the black market [25].

1.1 Contributions

Our work presents an in-depth reverse engineering analysis of the Zeus crimeware toolkit [6] where we illustrate the steps required to analyze this malware. We first present an overview of the toolkit. Next, we analyze communications between the command-and-control (C&C) server and the infected hosts. The sequence of communications is illustrated, revealing the periodic nature of transmissions. Following the network analysis, we present the reverse engineering steps to deobfuscate the malware and provide a description of the installation steps the malware takes upon infection of the host. We also extract configuration information from the binary and present its structure. We then describe a method to extract the encryption key which is used to decrypt communications and decrypt the local storage of both stolen information and additional configuration information. Finally, we breakdown the structure of a message sent from the bot to the C&C server where we identify a vulnerability with the implementation of the encryption algorithm Zeus employs.

The success of botnets is partly due to a strong economic incentive underlying every level of the botnet business model. We address the problem of botnets, specifically those created with toolkits, by exploring an economic attack against every level of the *food chain*:

botnet toolkits sold by their authors allow any layman to generate his/her own customized botnet and become a botmaster; botnet services sold by botmasters allow any criminal to steal identities and credit card information; stolen credentials are then sold to those who would make unauthorized transactions, transferring funds to money mules; and finally, end-users wire-transfer funds, minus a commission, to their employers. Although some preliminary efforts exist on botmaster trace-back, a countermeasure meets significantly more difficulties when the objective is to capture the botmaster or authors of a toolkit. Those at the highest levels of the food chain are the most technology-savvy and elusive. We address this concern by proposing a bottom-up approach to defaming botnet toolkits, causing the business of making and selling toolkits an unprofitable endeavour [45].

1.2 Thesis Organization

The remainder of this thesis is organized as follows. In Chapter 2, we provide background on botnets, related attacks, the Internet black market and provide a review of related work. In Chapter 3, we describe our reverse engineering efforts on the Zeus crimeware toolkit. In Chapter 4, we describe our novel approach to defaming botnet toolkits. Finally, in Chapter 5, we conclude the research and suggest future work.

Chapter 2

Related Work

In this chapter, we describe botnets and related attacks, in particular those that are financially motivated. Next, we highlight research efforts to detect, mitigate and analyze the botnet threat. We then describe the Internet black market economy and finally, highlight research that studies this phenomenon and proposes countermeasures.

2.1 Botnets

A botnet is usually defined as a network of compromised machines, individually known as bots, that are used to perform coordinated attacks as commanded by their owner, known as the botmaster. This definition distinguishes bots from other forms of malware, which do not communicate nor perform coordinated attacks. Another unique feature of botnets is that they primarily perform attacks that generate their botmaster a profit.

As a member of a botnet, a bot has a life cycle that can be generalized as follows:

1. A system is infected using any number of malware propagation methods;

2. The infected system initiates contact with the botnet;

3. The infected system receives commands and executes attacks; and,

4. The infected system may report attack results.

Bots may propagate themselves using any number of methods. For example, they may use tradition worm propagation by scanning the network for vulnerable systems, or use the botnet itself to email out copies of itself. Understanding how a botnet communicates is very important in developing any mitigation strategy. Botnet communication structures can be divided into two broad categories: centralized and decentralized.

- *Centralized*: Traditionally, botnets have a centralized network topology. They consist of a C&C server that a botmaster uses to send commands to their bots. The first C&C servers were IRC channels. These were employed by Agabot, SDBot, and SpyBot [4]. Upon infection, a bot would join a specific IRC channel on a specific server and wait for specific messages that invoke programmed actions. A botmaster will issue a command by posting a message to this channel. The C&C server *pushes* the command to bots which then invoke the command with a relatively low latency. As long as the infected machine is on and has an active Internet connection, it will remain connected to the C&C server awaiting commands. The simplicity of creating an IRC botnet led to its initial success; however, in recent years their popularity has declined [5] due to the emergence of other types of botnets.

  In contrast, Hyper Text Transfer Protocol (HTTP)-bots communicate with their C&C server through port 80, the standard Internet port. Just like a user visiting a website, a bot visits a server and communicates using the standard `GET` and `POST` commands a web browser would use. This represents a *pull* style for communicating commands; the bot itself must decide when to *phone* home and check for further instructions. A bot may communicate with the server periodically, randomly or using a predefined custom protocol. Depending on the frequency of such communications, latency between a botmaster commanding an action and the botnet performing the action may be relatively high. In spite of the issue of high latency, botnets such as BlackEnergy [43], Rustock [7], and clickbot.A [11] chose to communicate over HTTP, hiding their communications among all other Internet traffic. Another advantage of HTTP-botnets are, that unlike IRC-based botnets where many networks may block ports associated with IRC traffic, networks rarely block port 80; and therefore, the commands are more likely to reach the bot.

Centralized architectures, or topologies, are widely used for their effectiveness and simplicity. The main weakness with this architecture is that if the C&C server is taken offline, no further commands or updates may be issued to the botnet. Centralized topologies can mitigate this risk through use of a bullet-proof hosting service [55], which ignore website take-down requests, or have redundant C&C servers. This ongoing battle between botmasters and those trying to secure the Internet has no clear winner yet.

- *Decentralized*: In a decentralized network topology, there is no single C&C server that is known to all botnet members. Peer-to-peer (P2P) protocols, usually based on distributed hash tables (DHT) [1], are used to implement this architecture. In P2P, each node will know some subset of the other nodes in order to propagate files that contain commands. This makes the network resilient to take-down attempts. There is no single node that, if removed, will significantly impact membership. In addition, some protocols are capable of reorganizing themselves to better handle the loss of key nodes. The infamous Storm botnet [19] makes use of the Overnet Protocol, an implementation of the DHT Kademlia, to organize its peers.

  Decentralized botnets have a significant advantage over a centralized architecture as they are more difficult to dismantle since there are no clear C&C servers. However, it has been shown that they are vulnerable in other ways. A case study on Storm [26] involved researchers infiltrating the botnet. They were able to dismantle Storm by exploiting the P2P bootstrapping process, which relies on a static list of Internet Protocol (IP) addresses or domain names contacted by new nodes wishing to join the network. Controlling the IP addresses or domain names from the list prevents new membership. This highlights a weakness in using some P2P systems, as there is no authentication on new clients, making it easy to populate the botnet with machines controlled by investigators and not controlled by the botmaster. In addition, a P2P botnet can be prevented from growing by removing peers whose IP addresses are used in the bootstrapping process. However, even with the limitations described, future botnets may choose to adopt a P2P network topology, fostered by the success of P2P botnets such as Storm and the increased complexity needed to dismantle this style of botnet.

2.1.1 Botnet Attacks

As briefly mentioned in Chapter 1, botnets are used to perform a variety of, in general, financially motivated attacks. The first attack we describe is very familiar to anyone who uses email. Email spam is an unsolicited email that is sent, in bulk, to a large number of accounts. Typical email spam may be advertising products, directing users to phishing sites or distributing malware. Much of the email spam is sent by botnets. The botmasters will rent out their botnets to perform a spam campaign. Email spam from botnets may be more difficult to prevent than tradition email spam due to the number of different computers and email accounts being used in the attack.

A classic botnet attack is a DDoS. A denial-of-service (DoS) typically targets a system's resources such as their central processing unit (CPU), memory or network connections [42]. One common resource attack is to overload the capacity of a network device by sending a large number of packets. This prevents legitimate communications from occurring. A second common attack, which affects a victim's CPU, is a SYN flood [57], where Transmission Control Protocol (TCP) SYN packets are continuously sent to a system. Each packet requires a lookup by the CPU for existing connection information and the creation of an entry if none is found. These incoming connection attempts are queued and can quickly prevent other legitimate connection attempts. A DDoS is an extension of a DoS attack that originates from many different systems simultaneously (*i.e.*, distributed) and, which increases the effectiveness of the attack. Botnets are a perfect vessel to perform DDoS attacks. Botmasters make money from this type of attack either through extortion or as a paid service to a competing organization. Extortion may occur by threatening a company with shutting their servers down over a busy period such as the holiday season, which could result in a huge loss to online sales. A company may choose to pay the botmaster rather than risk greater financial loss through reduced online sales and customer dissatisfaction.

Another form of botnet attacks are click fraud attacks which target the pay per impression (*i.e.*, click) advertising market. A botmaster commands bots to click on advertisement banners, which pay a small amount of money per click to an account controlled by the botmaster. An individual system will not generate large revenues, however, with a botnet, numerous systems in the order of hundreds, thousands or more can bring in a sizeable profit.

Finally, information theft and phishing are two attacks with the same goal of stealing users' confidential information either with the objective of selling the stolen information or using it directly:

- Information theft can be performed in a number of ways: key loggers record and send a user's keystrokes to an attacker's database; and, form grabbers record information entered into online forms, such as credentials and credit cards, during a user's browsing activity. Some security measures involve graphical keyboards; however, this type of malware often comes with screen shot capture capabilities; and, malware may also steal information from protected storage such as cookies and password files.

- Phishing is the process of directing a user, often by email or instant message, to a fake website that is meant to look like a real service the user uses. When they log in or perform transactions they are inadvertently giving away their credentials and private information. Botnets are used in phishing attacks in a few ways. They may be used to spam phishing emails, host phishing sites, or an infected system may control a user's browsing experience such that they are redirected to a phishing site. This last attack is particularly problematic as what the user sees is under the complete control of the botmaster.

2.2 Botnet Detection

Botnet detection is one promising area of research to aid in mitigating the potential damage a botnet can cause. The numerous proposed and implemented methods to detect botnets are divided into two broad categories: host-based and network-based detection.

2.2.1 Host-based Detection

Since botnets are simply a collection of infected machines designed to receive commands and perform coordinated attacks, traditional malware detection mechanisms can be used to detect infections. Malware signatures have long been used by antivirus software to detect infected systems. However, with increased use of various obfuscation techniques such as encryption, polymorphism and metamorphism [56], traditional signature based approaches have been less successful.

Host-based behavioural detection identifies bot infections by observing systems behaviour and state. Detection occurs when an active system deviates from a normal systems behaviour and state. Stinson and Mitchell [54] proposed a system called *BotSwat*, which is based on the theory that a system process is being externally or remotely controlled when it uses network data (“an untrusted source”) as a system call argument (“a trusted sink”); this is referred to, in literature, as taint-propagation analysis. This method is computationally expensive and, like other anomaly-based detection methods, suffers from false positives. Another method based on taint analysis is proposed by Yin *et al.* [59]. *Panorama* performs “whole-system, fine-grained taint tracking” on user input such as text entered by a keyboard. The data is marked as tainted and monitored to see if it is transmitted via the network. This propagation creates a signature that can be used to detect entire families of malware. This is again a computationally expensive task and is designed to be performed off-line.

Martignoni *et al.* [36] created an approach that utilizes data-flow analysis of program execution and system calls. They model a set of actions associated with malware such as ‘proxying’, ‘keystroke logging’, ‘data leaking’, and ‘downloading and executing a program’ using combinations of system calls. A strong work in misuse detection, it may still be susceptible to evasion; malware can specifically alter the order of these events or use non-standard techniques. Similarly, Liu *et al.* [34] proposed *BotTracer*, a program which monitors system activities to detect three features modelled by the program, as critical for a functioning bot: startup of a bot is automatic; a C&C channel must be established; and, a bot will eventually perform attacks. This approach uses a combination of mechanisms to detect each of these features. During an operating system’s (OS) startup phase, a whitelist is created for services and other applications started by the OS. Any remaining processes that exhibit outward bound traffic are monitored. *BotTracer* monitors all inbound and outbound traffic on these suspicious processes and identifies C&C servers using known characteristics. It further monitor system calls associated with information harvesting and dispersion. The system additionally filters out those processes that are started by user activity which may lead to false negatives.

2.2.2 Network-based Detection

Network-based detection moves the detection mechanism off the host and onto a network device, which captures network traffic for one or more hosts. This type of detection loses some information only available on a host, such as intercepting network bound data before it is encrypted by the host and, relevant system call information. Host-based detection gives access to unencrypted information for identifying botnets and can use signatures from system call traces; whereas network-based detection allows for monitoring numerous systems simultaneously and may produce evidence of coordination amongst hosts, which is potentially indicative of botnets. It can also be used to detect several network traffic anomalies related to traffic volume, latency, unusual port use and Domain Name System (DNS) query activity.

DNS activity can be used in a variety of ways to detect botnets. Ramachandran *et al.* [47] propose using DNS blacklist counter-intelligence to determine botnet membership. This is where a botmaster performs a lookup to ascertain if their bots are blacklisted. Counter-intelligence watches for these queries to identify potential spam bots. However, this is limited to certain categories of spam botnets. Perdisci *et al.* [46] propose a system to analyze recursive DNS (RDNS) queries to detect fast-flux service networks (FFSNs). FFSNs are similar to content delivery networks (CDNs) used for load-balancing and ensuring high availability; however, they are designed to be used for malicious purposes, such as resolving IPs for phishing domains and, are generally comprised of infected hosts in a botnet. They monitor DNS query traffic, filter benign queries and cluster suspicious queries based on traffic statistics. Finally, they classify the clusters as legitimate CDNs or as malicious FFSNs using further statistics such as total number of distinct resolved IPs and average time-to-live (TTL).

Another method that monitors DNS traffic is proposed by Choi *et al.* [8]. In this method, they detect botnet membership based on DNS queries made by bots occurring simultaneously. They use anomaly detection based on anomalous query rates and may be susceptible to mimicry attacks.

Yen and Reiter [58] proposed a *Traffic Aggregation for Malware Detection* (*TAMD*) system, which uses three characteristics (destination, payload, and platform type) to aggregate hosts and determine infected systems. These systems may be members of a botnet or may be infected with other stealthy malware. They perform a horizontal correlation analysis that looks for similarities in the three characteristics across a large enterprise network. Horizontal correlation requires multiple infections by the same malware on the monitored network. In a similar work, Gu *et al.* [22] propose *BotMiner*, which also correlates network traffic, although it uses different characteristics than *TAMD*. Specifically, they are interested in "who is talking to who" (communication activities) and "who is doing what" (malicious activities). They independently cluster these activities and then perform cross-cluster correlation to determine those hosts partaking in both activities.

Karasaridis *et al.* [27] developed a methodology to track and detect IRC botnets on large Tier-1 Internet Service Provider (ISP) networks. Their work is similar to *BotMiner* in that they detect communication activities; however, they match these activity flows to known IRC botnet traffic profiles. The main difficulty with these approaches is that botnets are constantly changing their behaviour to circumvent such detection.

BotHunter, also proposed by Gu *et al.* [23], passively detects individual bots by correlating intrusion detection system (IDS) alerts to a predefined infection model. They perform vertical correlation, that is, correlating events that occur on a single host. Individual infection stages are detected including inbound scanning, exploit usage, egg downloading, outbound bot coordination dialogue, and outbound attack propagation. Two detection rules are used based on the proposed infection model that consist of a partially ordered dialogue exchanged between the host and the external network. If these events occur within a sufficient temporal window, the host is deemed infected. There is a risk of false negatives if key alerts are not raised by the IDS, or if there is a sufficient delay between alerts.

2.3 Internet Black Market

The Internet black market plays host to numerous sales of illicit electronic goods such as credit card information, banking and other online credentials, hacking services, botnets for rent, and malware toolkits. In this section, we discuss the monetary benefits of the market and proposed countermeasures.

To illustrate the Internet black market, Kshetri [29] assesses the cost-benefit structure of cybercriminals. They propose a framework that outlines how the characteristics of the three groups involved in cybercrime: law enforcement agencies, victims, and the cybercriminals, shape the cybercrime landscape. There are four factors that increase the success of cybercrimes and the confidence of cybercriminals: a lack of confidence in law enforcement, weak defence mechanisms, low reporting rates, and businesses complying with demands (*i.e.*, extortion). They also propose an economic formula to represent the cost-benefit analysis for a cybercrime where a cybercrime occurs if:

$$M_b + P_b > O_{cp} + O_{cm} P_a P_c \tag{1}$$

the monetary benefits, M_b, plus the psychological benefits, P_b, are greater than the psychological cost, O_{cp}, plus the opportunity cost of conviction, O_{cm}, times the probability of arrest, P_a, times the probability of conviction, P_c.

2.3.1 Countermeasures

Franklin *et al.* [18] presented two attacks on the trust and rating system of the Internet black market. In the first attack, they create numerous identities that are then used to build up each other's reputation as reliable black market sellers. When a real offer is made on their advertised goods, they accept the money; however, they do not provide the purchased item. This affects the overall reputation of the market, reducing real black market activity. In the second attack, they slander actual sellers to lower their reputation. These two attacks show promise; however, their approach does not handle the case where credentials are not sold in the black market.

Ford and Gordon *et al.* [15] present an attack on the ad revenue stream generated by botnets. In this method, a distributed set of clients joins a botnet used for advertising. These clients then artificially view or click ads. Although, initially this drives up profits for botmasters, the rate they are paid per impression or click is dependant on the number of actual sales occurring. If clicks are not translated into purchases, the advertising client will react by lowering the rate paid per click. Over time, the profits for the botmaster will diminish to the point where operating the botnet is no longer lucrative. This is an early work that addresses the viability of an attack against a botnet's business model.

Li *et al.* [33] present a work that utilizes honeypots to create uncertainty in the necessary rental size of a botnet for a DDoS attack. They extend their work to generalize the botnet-for-rent market as a whole [32]. In this attack, they intend to affect the profit margins of botmasters by reducing the effectiveness of their services. They first create two equations to calculate maximum profits.

For the attacker:

$$max\left(Profit\right) = M - P \times n \tag{2}$$

For the botmaster:

$$max\left(Profit\right) = P \times n - m \times k - a\left(N\right) \tag{3}$$

M is the payment received by the attacker for successfully disabling the target site. The rental price per bot is denoted by P and n is the minimum number of bots needed to perform the attack. The cost to maintain bots is calculated as the cost to maintain a C&C channel, m, times the number of C&C channels necessary to operate these bots, k. N is the size of a typical botnet needed to support an active population n and $a(N)$ is a penalty function representing economic losses suffered from being detected and arrested.

They then demonstrate that having “virtual” machines join a botnet and not perform attacks when commanded, decreases the overall market gains for running attacks. As there is now greater uncertainty in any given bot contributing to the attack, an increasing number of bots is required to successfully perform the attack. Since there are now additional bots required in a botnet, there are new incurred costs associated with the added C&C channels needed to operate the botnet. These have a direct affect in the penalty function as the botnet has grown.

Although this attack shows promise for disrupting the market surrounding DDoS attacks, it has not addressed a fundamental parameter in the existence of botnets which are the malware authors who have created them. The approach described in Chapter 4, extends the economic attack to lower the author’s profits by targeting the botnet toolkits they sell.

Li and Schmitz [31] propose a framework that utilizes different kinds of honeypots to facilitate attracting phishing activities and submitting fake credentials. Four types of honeypots are used:

- A honeyed e-banking system as a phoneypot;
- A number of phoneytokens as fake credentials supported by the honeyed e-banking

system;

- A number of spamtraps for attracting phishing emails and submitting phoneytokens to phishing sites; and,
- A number of phoneybots for submitting phoneytokens to pharmers and phishing malware.

Phoneytokens are fake credentials created and used by e-banking systems for this process. These tokens provide access to pseudo-real accounts that can be used in transactions. When the spamtraps at the financial institution receives a phishing email, they submit these phoneytokens to the phishing universal resource locator (URL). The phisher will eventually attempt to steal money from phoneytoken accounts. When a transfer occurs, the destination account is flagged as suspicious and is used to flag future transactions within that account, requiring the sending account to verify the transfer. This provides protection to real user's accounts. Throughout the process, the financial institution reports phoneytoken activities to authorities for investigative purposes. Their intent is to prevent phishing on real user accounts once destination phishing accounts have been identified.

Chapter 3

Analysis of the Zeus Crimeware Toolkit

3.1 Introduction

Zeus is a prevalent malware toolkit purchased with the intent of creating a Zeus botnet. It is a favourite tool of hackers and presents an interesting challenge to analyze. In this Chapter, we present a case study on an analysis of Zeus through reverse engineering. The purpose of this analysis is to gain insight into its capabilities and sophistication. We first describe the toolkit by analyzing the included files. Following this analysis, we present the results from constructing a Zeus botnet, using the analyzed files, in a virtual machine environment. This allows us to understand its capabilities and analyze the communications that occur between a bot and the C&C server. Next, we describe our efforts to reverse engineer two aspects of the toolkit: a builder program used to generate an infection binary, and the resultant binary used to infect systems with Zeus. Furthermore, this analysis exposes the obfuscation layers that Zeus employs to conceal its mechanisms and evade signature detection. We also describe outcomes of our reverse engineering analysis; a tool to automate the recovery of the encryption key used for communications; and, a sample of a decrypted communication session.

Chapter Organization: In section 3.2, we describe the toolkit and highlight its features and configuration details. We follow this discussion, in section 3.3, with a breakdown of the Zeus botnet communication structure, and observe the network traffic between a bot and the C&C server. In section 3.4, we present a detailed reverse engineering analysis of the Zeus crimeware toolkit. The next section, section 3.5, describes a tool to automate the recovery of the encryption key used for the bot communication, as well as the extraction of the configuration information from the binary bot executables. In section 3.6, we present a sample of a decrypted communication session between an infected machine and a C&C server. Finally, we summarize the analysis in section 3.7.

3.2 Description of the Zeus Crimeware Toolkit

The toolkit employs a number of programs and scripts to create and operate a Zeus botnet. It can be used to create many smaller botnets, each managed from a different C&C server, or one large botnet. The attacker is only limited by how many different networks of infected machines they wish to manage. A Zeus botnet operates using HTTP and is generally operated over the Internet. It can be used for any number of nefarious reasons, but, it is most commonly used to control machines for the purpose of stealing credentials for financial gain. A Zeus bot has the ability to log user input as well as to capture and alter data that is displayed to the user in their browser [25]. Stolen data may contain email addresses, passwords, online banking credentials, credit card numbers, and TANs. However, since a Zeus bot can be instructed to execute arbitrary code, the botnet can also act as a platform to launch any number of other attacks such as, DDoS or spam.

In our analysis, we examine the Zeus crimeware toolkit v.1.2.4.2, which at the time of analysis, was considered the latest stable publicly available version in the underground community. Newer versions of Zeus are available; however, the core functionality remains much the same. The core files contained in the Zeus crimeware toolkit can be grouped into three categories (see Figure 1):

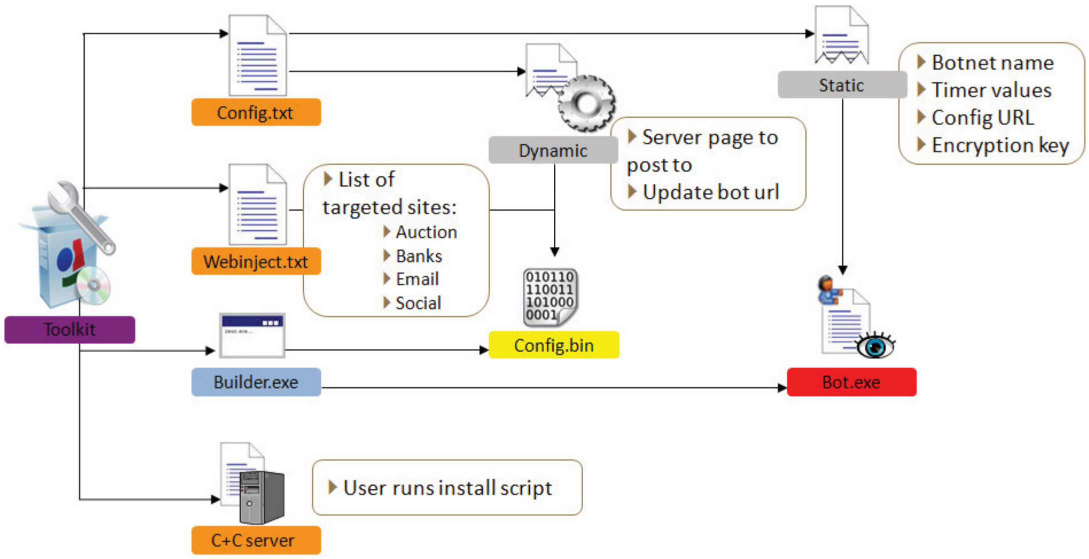


Figure 1: Configuration Files of Zeus

1. A C&C Server with installation scripts (C+C server);
2. A builder program with configuration files (Builder.exe, Config.txt and Webinjects.txt); and,
3. A generated bot executable and configuration binary (Bot.exe and Config.bin).

Each category is described to highlight the capabilities of the Zeus crimeware toolkit and is meant as a reference to facilitate analysis.

3.2.1 C&C Server with Installation Scripts

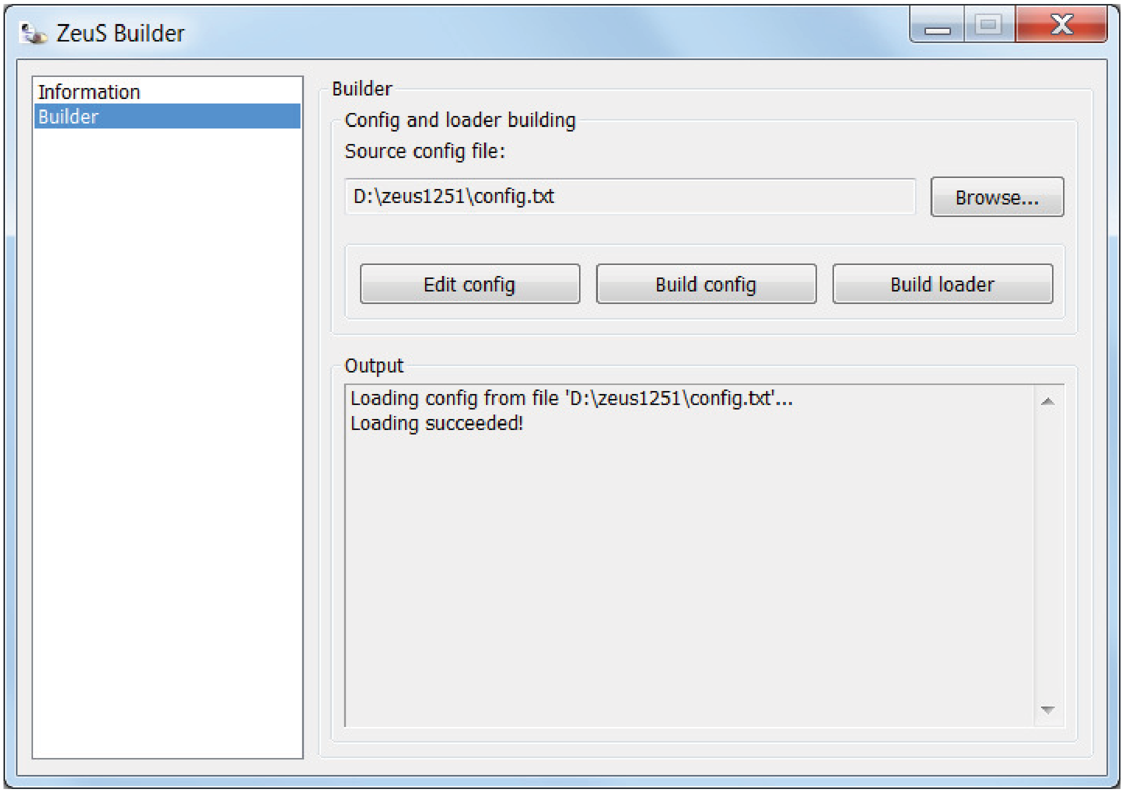
In the first category the crimeware toolkit has a set of simple installation scripts to configure the C&C server. The only pre-requisite is that a web server and a MySQL [10] database are installed. The user browses to the web path `install/index.php` and follows the resulting installation dialog. The installation script configures the C&C server with the botnet name and the password necessary to communicate with the bots. It also creates a database, cpdb, to store information on the botnet and the information the botnet harvests. The database is used to store related information about the botnet and any updated reports from the bots. For a bot to submit information to the C&C Server, and ultimately the database, it must POST the information to gate.php. These updates contain stolen information that is gathered by the bots from the infected machines. The botherder may then view the information through the control panel by navigating to cp.php. This control panel provides a user friendly interface to display the contents of the database as well as to send updates and other commands to the botnet (see Table 3). We observed information being submitted to gate.php and ultimately to the database, cpdb. The breakdown of the communicated information is described in section 3.6.

3.2.2 Builder Program with Configuration Files

The builder program is used to generate the bot executable and the dynamic configuration binary necessary for installing a bot, harvesting information and communicating with the C&C server. When generating the files, the builder program uses information from two files: config.txt and webinjects.txt. The builder program is a fully-fledged Microsoft Windows application with a user interface (see Figure 2).

Config.txt

The botnet configuration information is composed of two parts: a static section and a dynamic section. When the builder program generates the bot executable, the static section is embedded within the file. We observe the static structure during the analysis of the static configuration structure at the end of section 3.4.2. The dynamic configuration file is the dynamic section compressed and encrypted by the builder program, and then written to file. The configuration file allows for customizing the bot as shown in Figure 3. The following configuration options are available:



```
Loading config from file 'D:\zeus1251\config.txt'...
Loading succeeded!
```

Figure 2: Zeus Builder Interface

StaticConfig

- botnet - The name of the botnet that the bot should join.
- timer_config - Periodic timer variables for the frequency with which to download updates to the dynamic configuration file. There are two numbers; the first is how long to wait, in minutes, between downloading the configuration file, and the second is how long to wait, in minutes, if there is an error before re-attempting the download.
- timer_logs - Periodic timer variables for the frequency with which to upload stolen information to the C&C server. There are two numbers; the first is how long to wait, in minutes, between uploading the log, and the second is how long to wait, in minutes, if there is an error before re-attempting the upload.
- timer_stats - A periodic timing variable for the frequency with which to upload statistical information on the bot to the C&C server. There are two numbers; the first is how long to wait, in minutes, between uploading the statistics, and the second is how long to wait, in minutes, if there is an error before re-attempting the upload.
- url_config - The web location of the dynamic configuration binary.
- url_compip - An external website whose page contents contain the bot's externally visible IP address, as well as the amount of kilobytes required to be retrieved from the site to read the IP address.
- encryption_key - The key to decrypt the dynamic configuration binary and encrypt transmissions to the C&C server.
- blacklist_languages - A list of languages which are not supported by this botnet.

DynamicConfig

- url_loader - The web location to download updated versions of the bot binary.

- url_server - The web location to `POST` stolen information to.

- file_webinjects - The file name of the WebInjects configuration file used when the builder generates the binaries.

- AdvancedConfigs - Additional web locations to download backups of the dynamic configuration file.

- WebFilters - A set of URLs that by default direct the bot to log information, or to not log information by prefixing the URL with a ‘!’ symbol, or to direct the bot to take screenshots of the page by prefixing an ‘@’ symbol on the URL. Screenshots may be used for acquiring passwords entered through virtual keyboards in the web page.

- WebDataFilters - A list of pairs comprised of a URL and a string values that are used to specify what information is logged.

- WebFakes - A list of pairs of URLs used for a phishing attack, where the first URL in the pair would be a website the user intends to navigate to and the second URL in the pair is the website that the malware actually navigates the user to. The malware does so without affecting the URL shown in the location bar of the web browser.

- TANGrabber - A set of rules to extract transaction authentication numbers (TANs) used by some banks.

- DnsMap - Entries for the Windows hosts file. This can be used to effectively override DNS lookups as the hosts file is checked before a DNS query is made.

```
;Build time:   14:15:23 10.04.2009 GMT
;Version:      1.2.4.2

entry "StaticConfig"
  ;botnet "[BOTNET NAME]"
  timer_config 60 1
  timer_logs 1 1
  timer_stats 20 1
  url_config "http://[WEB DOMAIN]/config.bin"
  url_compip "http://[WEB DOMAIN]/ip.php" 1024
  encryption_key "[SECRET KEY]"
  ;blacklist_languages 1049
end

entry "DynamicConfig"
  url_loader "http://[WEB DOMAIN]/bot.exe"
  url_server "http://[WEB DOMAIN]/gate.php"
  file_webinjects "webinjects.txt"
  entry "AdvancedConfigs"
    ;"http://[WEB DOMAIN]/cfg1.bin"
  end
  entry "WebFilters"
    "!*.[WEB DOMAIN]/*"
    "!http://*[WEB DOMAIN]*"
    "https://[WEB DOMAIN]/*"
    "!http://*[WEB DOMAIN]/*"
    "!http://[WEB DOMAIN]/*"
    "@*/[WEB DOMAIN]/*"
    "@*/[WEB DOMAIN]/*"
  end
  entry "WebDataFilters"
    ;"http://[WEB DOMAIN]/*" "passw;login"
  end
  entry "WebFakes"
    ;"http://[WEB DOMAIN]" "http://[PHISING SITE]" "GP" "" ""
  end
  entry "TANGrabber"
    "https://[WEB DOMAIN]" "" "TOKEN=*" "*"
  end
  entry "DnsMap"
    ;127.0.0.1 [WEB DOMAIN]
  end
end
```

Figure 3: Configuration File `config.txt`

WebInjects.txt

The webinjects.txt serves two purposes: firstly, it directs the bot to log specific information from a retrieved web page, and, secondly, instructs the bot to insert additional HyperText Markup Language (HTML) code into retrieved pages. Figure 4 shows an example of this file with the following explanation:

- set_url - The first portion is a string pattern to match against web locations and the second portion is a set of symbols. The entry set_url, indicates a start of a rule. In the first rule, it is a web page that contains an account balance. In the second rule it is a login form. The symbols for set_url are:

  G - Invoke this rule when the page is retrieved with a GET command.

  P - Invoke this rule when the page is retrieved with a POST command.

  L - This represents the switch to log additional information as shown in the first rule or, when L is not present, inject information as shown in the second rule.

- data_before - A string pattern to identify at what location, in the retrieved HTML, to either begin logging information or injecting new HTML code.

- data_inject - Either a label for the extracted piece of information or the new HTML code to inject.

- data_after - A string pattern to identify at what location, in the retrieved HTML, to either stop logging information or to stop overwriting the retrieved HTML with the injected HTML code.

Some examples of how this file can be used include: retrieving account balances, modifying the displayed balance, retrieving usernames or other personal information, and injecting new fields into login forms such as a Social Insurance Number (SIN) or Debit Card Personal Identification Number (PIN) so the user divulges additional information.

```
set_url */[WEB ACCOUNT PATH]* GPL
data_before
  Account balance:<span>
data_end
data_inject
  balance:
data_end
data_after
  </span>
data_end

set_url */[WEB LOGIN PATH]* GP
data_before
  Password: <input type="password" name="pwd" /><br />
data_end
data_inject
  Social Insurance Number: <input type="text" name="sin" /><br />
data_end
data_after
data_end
```

Figure 4: Web Injection File `webinjects.txt`

3.2.3 Generated Bot Executable and Configuration Binary

The final step of the builder program is to generate the bot executable and configuration binary. By performing a simple binary comparison, it can be determined that each instance of the generated bot executable is different, indicating that some form of binary obfuscation is employed.

Before describing these files, there are two additional steps we took in this analysis: we perform a network communication analysis in section 3.3, and we de-obfuscate the binaries at the start of section 3.4, the reverse engineering analysis. The network analysis provides us with insight during the reverse engineering stages, and de-obfuscating the binaries is necessary before we can reverse engineer their functionality.

3.3 Zeus Botnet Network Analysis

We now describe the network communication that occurs between the C&C server and a newly infected machine. The purpose of this analysis is to aid in the creation of detection mechanisms. The test environment is as follows:

1. Two Windows machines were created in a virtual network environment.

2. The first machine was setup as the C&C server. We configured a web server and executed the Zeus installation scripts. This server hosts all the resources necessary to operate the botnet which include the control panel (`cp.php`), the drop location (`gate.php`), the MySQL database (`cpdb`) and the dynamic configuration file (`config.bin`).

3. In the second machine, we configured the malware by modifying `config.txt` to have it communicate with our C&C server and then infected the machine with Zeus.

4. We created fake websites to reflect real scenarios of botnet attacks. All necessary entries of the configuration file as well as the web injects scripts are modified to target the fake websites.

Using Wireshark [2], we collected network traces between the two machines for one day. On the infected machine we navigated to the fake websites and then used login credentials, personal information, and credit card information for testing purposes.

Network communications between the C&C server and the infected machine were then analyzed. From this analysis we were able to confirm that Zeus uses HTTP to communicate between infected clients and the C&C server. From these network traces, we observed that the bot periodically checks the C&C server for a configuration binary and also a bot executable. The analysis also highlighted that the communications between both machines

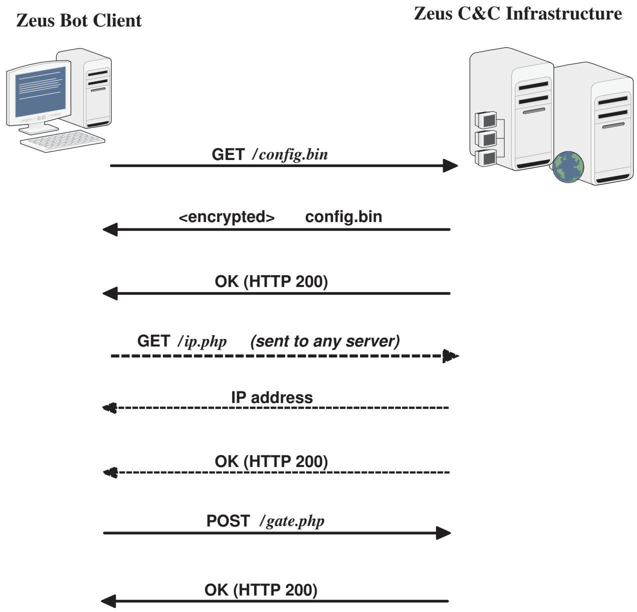


Figure 5: Communication Pattern of Zeus

are encrypted. We have created a sequence of events to illustrate the communication pattern that occurs between the C&C server and the infected machine:

1. The Zeus bot first sends a request message `GET /config.bin` to initiate a download of the dynamic configuration file from the location specified in the static section of `config.txt`. This location can be any web server. For our purposes we used the C&C server;

2. The contacted server then responds with the configuration file `config.bin`. This file can be decrypted by the infected machine using the configuration key embedded in the binary;

3. The infected machine contacts a URL specified in the static portion of the configuration file for the purpose of determining its externally facing IP address;

4. The server then responds with the IP address in plain text; and,

5. The bot posts status information to the C&C server at a configurable web site. For our purposes the message was `POST/gate.php`.

The main findings of this analysis are shown in Figure 5, which depicts the sequence of communications that occur between the two machines. Some communications are repeated periodically. The bot will check for an updated configuration file, send new status information and upload stolen information in a periodic fashion based on the intervals specified in the static portion of the configuration file used to generate the bot.

3.4 Reverse Engineering Analysis

We now describe the reverse engineering analysis we performed on the Zeus crimeware toolkit components. We specifically analyzed two binaries: the Zeus builder and Zeus bot executable. The analysis provides insight into the changes that occur on a user's system, such as modification to the files that run during the operating system's (OS) startup, new network traffic, changes to the registry, new files created and the infection of existing processes.

We use *IDAPro* [50] to analyze the binaries. *IDAPro* is the industry leading disassembler which interprets a string of bytes as mnemonics of machine instructions.

3.4.1 The Zeus Builder Program Analysis

Zeus builder is the first component of the crimeware toolkit. It is used to configure a Zeus botnet and generate the bot executables. It makes use of two configuration files as inputs to customize its functionality. In Figure 2, we can see the functionality provided through the graphical user interface (GUI).

Before analysis of the functionality could take place, we were required to remove an obfuscation layer. Using *PEiD* [53] we observed that the builder is packed using *UPX* [44]

File	**Description**
C:/WINDOWS/system32/sdra64.exe	A copy of the bot executable.
C:/WINDOWS/system32/lowsec/local.ds	A local copy of the generated dynamic configuration file.
C:/WINDOWS/system32/lowsec/user.ds	A data file used to store the users' personal information logged by the bot.

Table 1: Description of the Files Created During the Bot Infection

and must first be unpacked using *UPX*. After this was completed, we used the *PaiMei* reverse engineering framework [3], for its code coverage tracking and data flow tracking capabilities, to see which functions within the Zeus builder are invoked by a specific action within the GUI. This immensely aids in reverse engineering, as it allows us to focus on a few key subroutines at a time. The following summarizes the reverse engineering analysis of the builder program.

- Configuration Binary: This functionality generates the configuration binary. It reads in the dynamic section of `config.txt` and `webinjects.txt` and creates a binary structure of the information. Next it compresses and encrypts the entire structure using *RC4* [48] and the encryption key in `config.txt`.

- Zeus Bot Executable: This functionality generates the Zeus bot executable. It reads in the static section of `config.txt` and creates a portable executable (PE). Much of the compiled code and data structures are obfuscated using unique keys that are generated randomly each time the function is invoked. The encryption key to decrypt the configuration binary is also embedded in the PE.

- Zeus Infection Removal: This functionality is useful if you accidentally infect your system with Zeus. It detects the existence of registry entries and file locations on your system (see Table 1). If detected, the program removes the registry entries, instructs the bot to shutdown, and then deletes the stored Zeus binary, and other related files, from the system.

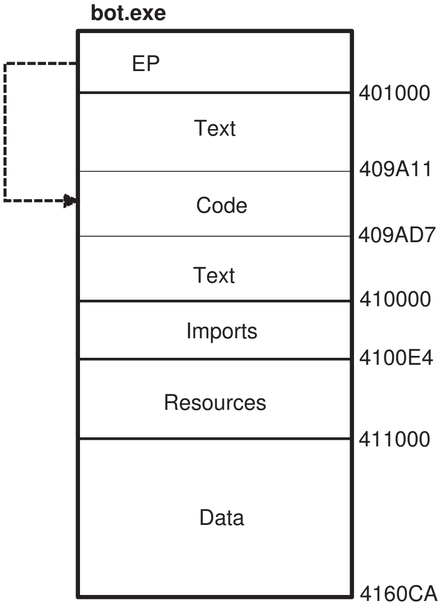


Figure 6: Segments of the `bot.exe` Binary File

3.4.2 Zeus Bot Binary Analysis

As shown in Figure 6, the bot binary file contains four segments: a *text/code* segment, an *imports* segment, a *resources* segment, and a *data* segment. We began our analysis at the malware entry point (EP) that resided in the *text/code* segment. The initial analysis of the disassembly showed that only a small part of the *text/code* block is valid computer instructions. This indicated that most of the binary is obfuscated, meaning that the computer cannot use this code directly and must first use procedures in the remaining valid code blocks of the binary to de-obfuscate these sections.

De-obfuscation Process

Using the *IDAPro* debugger, we were able to debug the malware and step through the instructions to analyze and understand the logic of the de-obfuscation routines. Each routine

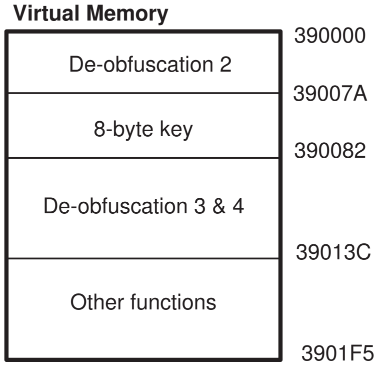
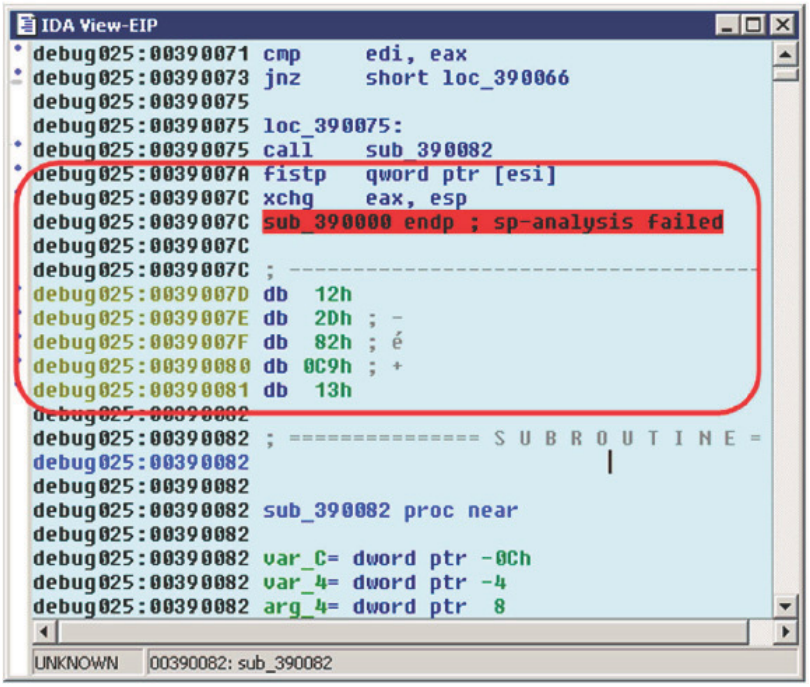


Figure 7: De-obfuscated Code in Virtual Memory

revealed some information which is used by the other routines until all obfuscation layers are removed.

The first de-obfuscation routine contained a decryption key, 4-bytes in length, and a one-byte long seed value. These two values are used to decrypt a block of data from the *text/code* segment one byte at a time, rotating through the decryption key one byte at a time, and performing a logical `AND` operation. The resulting decrypted data is then written into virtual memory.

The result of the first de-obfuscation routine revealed some new code segments. These segments contain three de-obfuscation routines as shown in Figure 7. During our analysis, the initial offset address of the memory for the code segments was `0x390000`. This offset location may change upon subsequent executions of the binary as it is determined by the OS at runtime. Immediately following the second de-obfuscation routine there was an 8-byte key that the *IDAPro* dissembler incorrectly identified as machine language instructions, or opcodes. Figure 8 illustrates the location of the 8-byte key and shows that the bytes are incorrectly interpreted by the disassembler as opcodes and presented as assembly code mnemonics. This further complicates the analysis as the reverse engineer must identify when these errors occur. In the following sequence of events, we explain the main logic of the second de-obfuscation routine.



```
debug025:00390071 cmp     edi, eax
debug025:00390073 jnz     short loc_390066
debug025:00390075
debug025:00390075 loc_390075:
debug025:00390075 call    sub_390082
debug025:0039007A fistp   qword ptr [esi]
debug025:0039007C xchg    eax, esp
debug025:0039007C sub_390000 endp ; sp-analysis failed
debug025:0039007C
debug025:0039007C ; ------------------------------------
debug025:0039007D db  12h
debug025:0039007E db  2Dh ; -
debug025:0039007F db  82h ; é
debug025:00390080 db 0C9h ; +
debug025:00390081 db  13h
debug025:00390082
debug025:00390082 ; =============== S U B R O U T I N E =
debug025:00390082
debug025:00390082
debug025:00390082 sub_390082 proc near
debug025:00390082
debug025:00390082 var_C= dword ptr -0Ch
debug025:00390082 var_4= dword ptr -4
debug025:00390082 arg_4= dword ptr  8
```

Figure 8: The 8-byte Key

1. First, it copies two binary blocks from the *text/code* segment, concatenates them together, and then writes them into the virtual memory. The first text block contains data with many zero value bytes that will be filled by the next text block as shown in Figure 9.

2. The routine scans every byte of the first text block and, when it encounters a hole (zero byte), it will overwrite the zero-byte with the next available byte in the filler block. This is repeated until all holes are filled (see Figure 10).

The filled text segment is the main output of the second de-obfuscation routine. However, this text segment is still not fully de-obfuscated and, therefore, not interpreted as valid computer instructions. By utilizing the 8-byte key, in Figure 8, the third de-obfuscation routine starts decrypting the output of the second de-obfuscation. Similar to the first de-obfuscation routine, this routine utilizes an 8-byte key and performs an `eXclusive-OR` (`XOR`) operation rather than a logical `AND` operation, again rotating through the key one

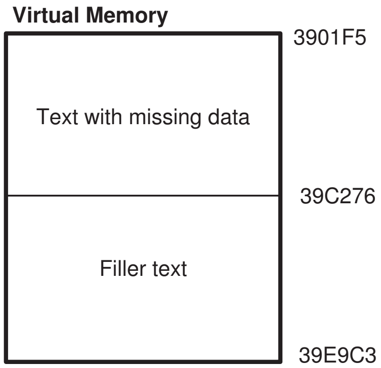


Figure 9: Virtual Memory Used by the Second De-obfuscation Routine

byte at a time, and decrypting the text one byte at a time.

Finally, the fourth de-obfuscation layer contains heavy computations to initialize and prepare some parameters for the rest of the malware operations. It uses the decrypted bytes revealed by the previous routines to modify the rest of the *text/code* segment. This routine was not fully analyzed. After this routine completes, we can observe the *real* starting point of Zeus.

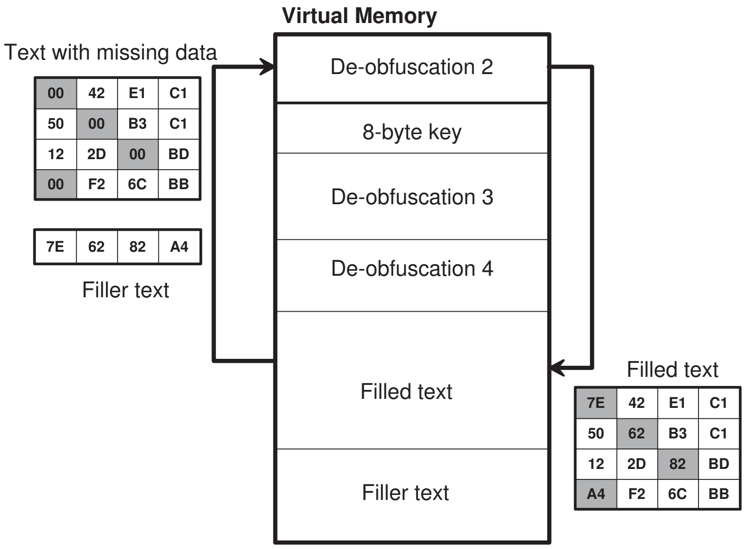


Figure 10: Result of the Second De-obfuscation Routine

Even though the *text/code* segment is now valid, the Zeus bot employs two additional layers of obfuscation. These two layers are de-obfuscated during the installation procedure. They consist of logical loops that transform arbitrarily long strings into clear text. The first routine is performed on a set of strings that the malware uses to load the dynamic link libraries (DLLs), retrieve function names, and for other purposes during the installation process. Similarly, the second routine is used to decrypt the URLs that were previously embedded in the binary, from the static section of the configuration file, when the binary was generated by Zeus builder. The main logic of these two routines are described in Algorithm 3.4.1 and Algorithm 3.4.2.

Algorithm 3.4.1: DECRYPT_STRING(enc_string)

```
seed = 0xBA;
String new_string = new String(enc_string.length());
for $i = 0$ to enc_string.length()
    do { new_string[i] = (enc_string[i] + seed) %256;
         seed = (seed + 2);
return ($new\_string$)
```

Algorithm 3.4.2: DECRYPT_URL(enc_url)

```
String new_url = new String(enc_url.length());
for $i = 0$ to enc_url.length()
    do {
        if $(i\%2 == 0)$
          then
        new_url[i] = (enc_url[i] + 0xF6 - i * 2) %256;

          else
        new_url[i] = (enc_url[i] + 0x7 + i * 2)%256;
    }
return $(new\_url)$
```

Bot Installation Process

After the first four de-obfuscation routines are executed, the malware begins the installation process. The installation process aims at preparing and then launching the malicious activities of the malware. We now explain the main procedure of the installation process:

1. Zeus dynamically loads the `LoadLibrary` and the `GetProcAddress` methods from `Kernel32.dll` library;

2. It decrypts a set of strings, which become DLL methods names, into the virtual memory according to Algorithm 3.4.1. Table 2 lists all of the Windows DLLs that Zeus employs and the number of methods that are loaded from each one of them;

3. The `LoadLibrary` and the `GetProcAddress` methods are then used to load further methods, as decrypted in step 2, from the Windows DLLs;

4. Zeus enumerates the current process table looking for targeted processes such as the main process, `outpost.exe`, for the Outpost [51] personal firewall application

Windows DLL	# of dynamically loaded methods
Kernel32	32
Shell32	2
Ntdll	9
Shlwapi	16
Psapi	1
Wininet	28
Advapi32	25
Ws2_32	23
Wsock32	3
User32	24
Ole32	3
Crypt32	8

Table 2: List of Windows DLLs Dynamically Loaded by the Bot Binary

from Agnitum Security and the main process, `zlclient.exe`, for the personal firewall [35] from CheckPoint. If any of these processes are found, then Zeus aborts the installation process;

5. Zeus appends the path `C:/Windows/System32/sdra64.exe` to the registry key: `HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/WindowsNT/CurrentVersion/Winlogon/Userinit`. This entry enables Zeus to initiate its installation process again during Windows startup;

6. Finally, it injects the de-obfuscated Zeus logic from the memory address `0x400000` to `0x417000` into virtual memory allocated within `winlogon.exe`. Following this step, Zeus passes control of this process by creating a new user thread, which is immediately executed.

As mentioned in step 5, the bot follows these steps when the machine is restarted to reinfect it. There are a few additional steps that are only performed during the initial Zeus installation process, or when a new version of Zeus is installed. The routine creates a local copy of the malware and stores it on the infected system to be activated anytime the system is restarted. Below, we list the process to create a local copy of the malware.

1. Zeus searches for any existing copies of a previous Zeus infection file, `sdra64.exe`, and then erases it from the machine. This behaviour would occur when the Zeus binary file is being updated with a newer version of the malware.

2. It makes an exact copy of itself and then saves it to `C:/Windows/System32/sdra64.exe`. To evade signature-based detection systems, it appends some randomly generated bytes to the end of the file to ensure any future attempts to hash the file will result in a different hash value.

3. In order to hide itself, the bot copies the modification, access, and creation (MAC) time information from the `Ntdll.dll` file, and applies them to `sdra64.exe`. The intent of this is to make `sdra64.exe` appear as a Windows OS file.

4. To further hide the file, it sets the `sdra64.exe` file attributes to `system` and `hidden`, so that the user cannot see the file using the standard file explorer settings.

The installation procedure is continued by the user thread that was started in the `winlogon.exe` process as described in step 6 of the Bot Installation Process. Once this hand-off occurs, the currently running bot exits and leaves control to the injected thread in `winlogon.exe`. The Zeus thread injects itself into another process, `svchost.exe`. This injected thread initiates a communication channel with the C&C server to download the latest updates of the configuration file and the malware itself. The malware is now ready to harvest the user's personal information. During the malware update process, the following changes were observed on the file system:

1. A new folder is created at the path `C:/Windows/System32/lowsec`. Hiding techniques similar to those applied to the `sdra64.exe` are also applied to the created folder.

Command	Purpose	Return Value
1	Retrieve Zeus version number	4 bytes in a buffer
2	Retrieve name of the botnet	Ascii string in buffer
3	Uninstall bot	n/a
4	Open the `local.ds` file or create it if it does not exist	n/a
5	Close the `local.ds` file	n/a
6	Open the `user.ds` or create it if it does not exist	n/a
7	Close the `user.ds` file	n/a
8	Close the `sdra64.exe` file	n/a
9	Open the `sdra64.exe` file	n/a
10	Retrieve the `sdra64.exe` file path	Wide character string
11	Retrieve the `local.ds` file path	Wide character string
12	Retrieve the `user.ds` file path	Wide character string
13	Crash the `winlogon.exe` process	n/a

Table 3: List of Zeus Commands

2. Two new files, `local.ds` and `user.ds`, are created and placed in the new folder. The file, `local.ds`, stores the dynamic configuration file, and the file, `user.ds`, logs the stolen information until Zeus is ready to send it to the drop location.

The Zeus thread that resides in the `winlogon.exe` process contains the main functionality, *i.e.*, the brains, for a Zeus bot. It communicates and coordinates all the infected processes using the named pipe `_AVIRA_2109`. Table 3 shows the list of the commands that are supported by Zeus.

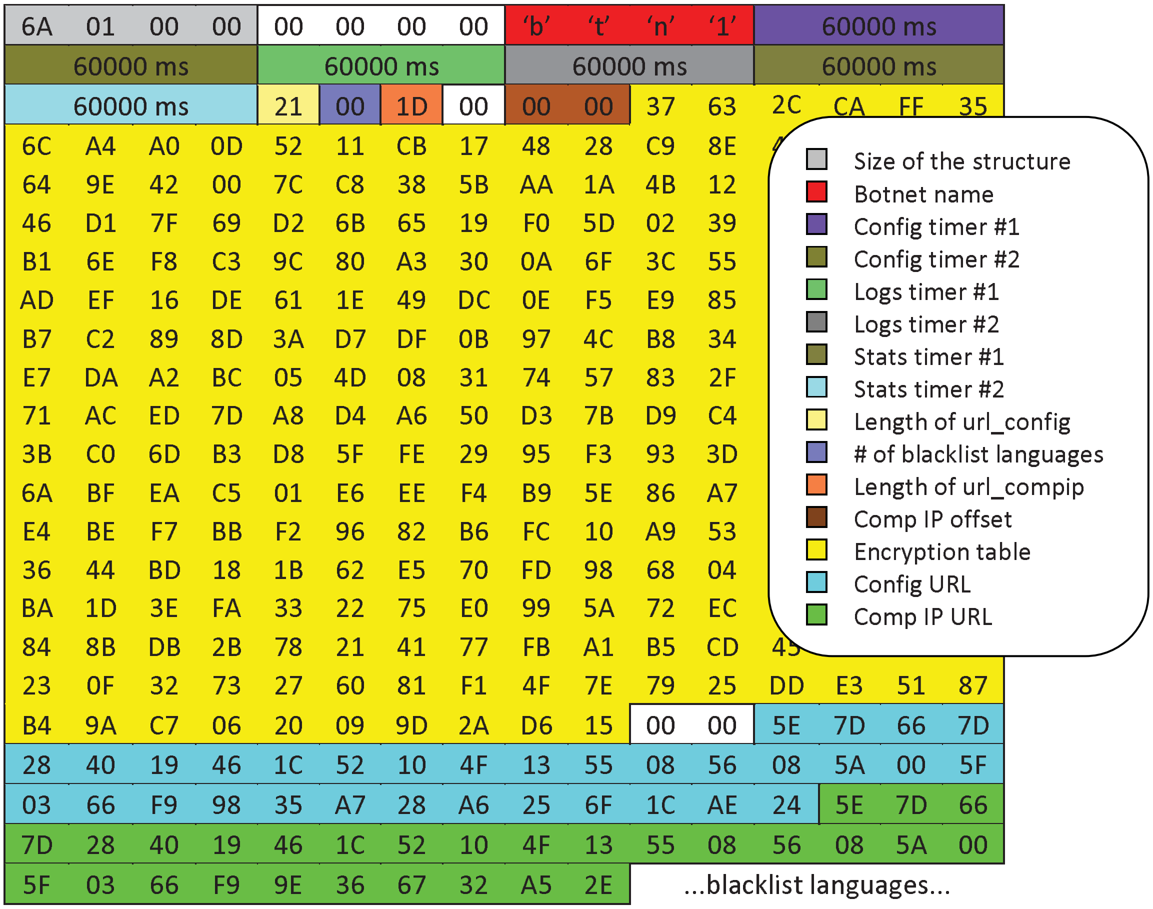
We now describe the remaining functions of Zeus which are aimed at hiding the three files created during installation: `local.ds`, `user.ds` and `sdra64.exe`; and, controlling the user's browsing experience including, but not limited to, harvesting the form input fields submitted during browsing activities. Zeus achieves this by altering the functionality of key Windows application programming interface (API) calls through function call hooking [24].

Zeus is able to effectively hide files by intercepting Windows API calls, such as FindFirstFile and FindNextFile [39], and modifying what results are returned to the process that initiated the call. For example, when Explorer (`explorer.exe`) is displaying files under the path `C:/Windows/System32/` it would display all files to the user as this is what is generally returned from the API calls; however, Zeus intercepts these calls and hides `sdra64.exe` by intercepting and modifying the output. This effectively hides the files from the list that is displayed to the user. This modification occurs for every process in the process table and makes `sdra64.exe` invisible to all processes that use Windows API calls to access files on the computer. `Sdra64.exe` could still be viewed by processes that do not make use of the file system functions; instead, they directly access the disk's file system structures. Unlike Explorer, RootkitRevealer [9], a program designed to reveal stealthy activity in a system, is able to see that these files do in fact exist by performing a raw scan of the disk instead of using the Windows API.

Zeus is able to modify the user's browsing experience and steal sensitive information by intercepting calls to Windows API Internet functions. These functions are used when the browser communicates with the Internet. All data, such as form input fields being sent to a website, or HTML received from a website, can be intercepted, recorded and modified. This is how Zeus intercepts and records the form data and modifies the page that the user sees. These functions may be used by programs other than Internet explorer which would then also be vulnerable. We take advantage of this later in this thesis in section 4.4.1.

Static Configuration Structure

As mentioned in section 3.2, the Zeus botnet uses a configuration file that contains static information. Specifically, this part of the configuration is stored inside the malware binary file in a specific structure. During the de-obfuscation processes, this structure is recovered and placed in virtual memory, `0x416000` in our analysis. All information in the structure is completely de-obfuscated except for two URLs: `url_compip` and `url_config`. These URLs can be de-obfuscated using Algorithm 3.4.2. The URL, `url_compip`, is the



6A	01	00	00	00	00	00	00	‘b’	‘t’	‘n’	‘1’	60000 ms			
60000 ms				60000 ms				60000 ms				60000 ms			
60000 ms				21	00	1D	00	00	00	37	63	2C	CA	FF	35
6C	A4	A0	0D	52	11	CB	17	48	28	C9	8E	4			
64	9E	42	00	7C	C8	38	5B	AA	1A	4B	12				
46	D1	7F	69	D2	6B	65	19	F0	5D	02	39				
B1	6E	F8	C3	9C	80	A3	30	0A	6F	3C	55				
AD	EF	16	DE	61	1E	49	DC	0E	F5	E9	85				
B7	C2	89	8D	3A	D7	DF	0B	97	4C	B8	34				
E7	DA	A2	BC	05	4D	08	31	74	57	83	2F				
71	AC	ED	7D	A8	D4	A6	50	D3	7B	D9	C4				
3B	C0	6D	B3	D8	5F	FE	29	95	F3	93	3D				
6A	BF	EA	C5	01	E6	EE	F4	B9	5E	86	A7				
E4	BE	F7	BB	F2	96	82	B6	FC	10	A9	53				
36	44	BD	18	1B	62	E5	70	FD	98	68	04				
BA	1D	3E	FA	33	22	75	E0	99	5A	72	EC				
84	8B	DB	2B	78	21	41	77	FB	A1	B5	CD	45			
23	0F	32	73	27	60	81	F1	4F	7E	79	25	DD	E3	51	87
B4	9A	C7	06	20	09	9D	2A	D6	15	00	00	5E	7D	66	7D
28	40	19	46	1C	52	10	4F	13	55	08	56	08	5A	00	5F
03	66	F9	98	35	A7	28	A6	25	6F	1C	AE	24	5E	7D	66
7D	28	40	19	46	1C	52	10	4F	13	55	08	56	08	5A	00
5F	03	66	F9	9E	36	67	32	A5	2E	...blacklist languages...					

- □ Size of the structure
- 🟥 Botnet name
- 🟪 Config timer #1
- 🟫 Config timer #2
- 🟩 Logs timer #1
- ⬛ Logs timer #2
- 🟫 Stats timer #1
- 🟦 Stats timer #2
- 🟨 Length of url_config
- 🟪 # of blacklist languages
- 🟧 Length of url_compip
- 🟫 Comp IP offset
- 🟨 Encryption table
- 🟦 Config URL
- 🟩 Comp IP URL

Figure 11: Static Configuration Structure

web location to determine the IP address of the infected host, and `url_config` is the web location to download the configuration file for the botnet. The static configuration structure also contains an RC4 substitution table that is generated by the encryption key specified in the configuration file. Throughout our analysis, we noticed that the substitution table was generated by RC4’s key-scheduling algorithm and then we verified that the encryption employed by Zeus is done by the RC4 algorithm. See Figure 11 for a breakdown of the static configuration structure. This structure is of particular relevance because it provides much of the botnet’s configuration details including the timer variables, the configuration URL and the expanded RC4 substitution table; the timer variables can be used to write network detection routines for this botnet's activities, and the configuration URL identifies a key server in the botnet where the bots retrieve the rest of the configuration details.

3.4.3 Summary of Reverse Engineering Analysis

The analysis of Zeus showed the core functionality of the builder program and identified the files created during an infection by observing its infection removal functionality. The analysis also showed the layers of obfuscation employed by Zeus to frustrate analysis and evade signature detection. Furthermore, the installation steps of the malware were illustrated. This provides researchers valuable insight into how malware authors make use of operating system APIs to subvert control of active processes. The research also shows both the basic functionality to hide its presence as well as the rootkit functionality employed to make the malware's files virtually invisible to all processes and ultimately the victim. Finally, the steps to extract the static configuration information and the analysis of its structure were presented. This information allows researchers to begin to identify the networked infrastructure supporting the botnet. Combined with information extracted from the dynamic configuration file, researchers now have a thorough understanding of all the botnet components.

3.5 Automatic Key Extraction

The recovered static configuration can be used in different ways to gain some control over the botnet. One of the most valuable pieces of information is the substitution table which can be used to decrypt all the communications of the Zeus botnet. Moreover, it can be used to decrypt the dynamic configuration file as well as the stolen information. In order to recover the static configuration structure described above, we have to go through all the de-obfuscation phases discussed in section 3.4.2. This requires executing the malware until it finishes all the de-obfuscation layers.

Using the Python [16] scripting language along with the *IDAPython* plugin [13], we were able to emulate all the de-obfuscation routines and extract the substitution table from the static configuration structure. This substitution table allows for decrypting the botnet communication traffic and the encrypted files. Our experimental results show that any sub-version of Zeus (v.1.2.x.x) can be fully analyzed using our methodology because it holds the same logical blocks. We now explain the overall procedure to extract the substitution table that resides in the static configuration structure:

- Emulate the initial de-obfuscation routine:
    1. Locate the EP and use it as a reference point for other memory locations;
    2. Use a byte offset from the EP to determine the location of the first de-obfuscation routine's 4-byte key;
    3. Use a byte offset from the EP to determine the hard coded seed, in the case of Zeus it is `0xE1`;
    4. Use a byte offset from the EP to determine the obfuscated code's location;
    5. Create a new memory segment at an address outside the program's allocated memory (e.g. 0x420000); and,
    6. Perform the first de-obfuscation routine as described in section 3.4.2 using the seed value and the 4-byte key, and then write the result into the new segment. The length of the result is always `0x1F5`.
- Emulate the second de-obfuscation routine:
    1. Use a byte offset from the start of the new segment (`0x420000`) to locate the:
        (a) Missing data text block starting address;
        (b) Missing data text block size;

        (c) Filler text block starting address; and,

        (d) Filler text block size.

    2. Create a new segment in an unused memory location (`0x422000`); and,
    3. Perform the second de-obfuscation routine as described in section 3.4.2 and write the results into the new segment.

- Emulate the third de-obfuscation routine:

    1. Use a byte offset from the memory reference `0x420000` to locate the 8-byte key; and,
    2. Perform the third de-obfuscation routine as described in section 3.4.2.

- Emulate the fourth de-obfuscation routine:

    1. Use the previous de-obfuscation results to modify the rest of the *text/code* and *data* segments.

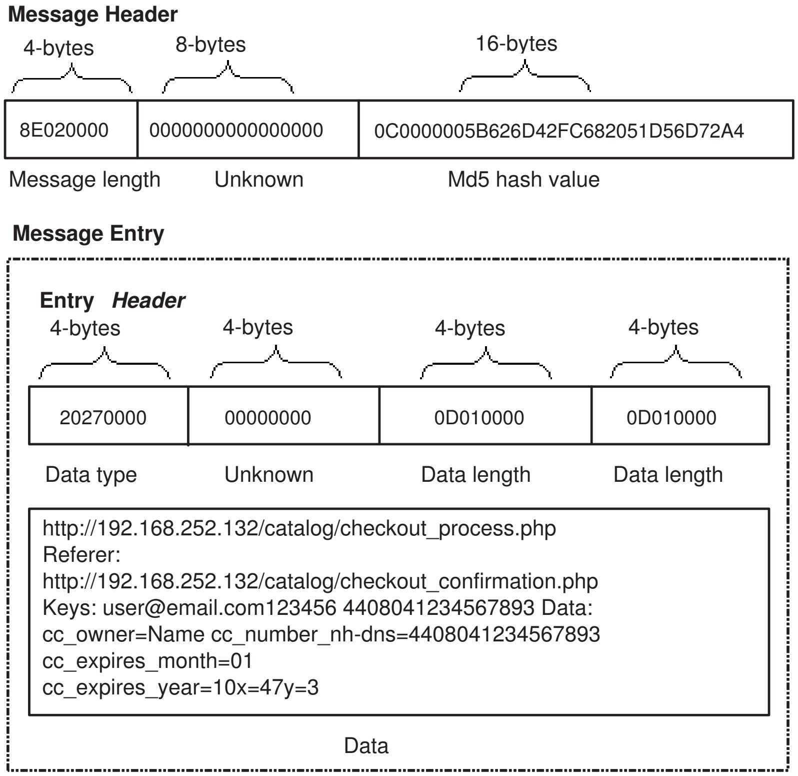
- Key extraction:

    1. Extract the 256-byte substitution table located at memory address `0x41602A`. This key will allow for decrypting the botnet's communication traffic and all the encrypted files.

As mentioned previously, this also enables the extraction of any information from the static configuration structure, such as the URL for the dynamic configuration file, exposing one of the key servers in the botnet's infrastructure. If one were to gain control of this server and, in possession of the substitution table, they could reconfigure the bots. By creating a new dynamic configuration file, this would cause the bots to communicate with a new C&C server under their control and effectively hijack control of the Zeus botnet. This is just one approach, of many, to subvert a botmasters control. This approach was not pursued in this research.

3.6 Packet Decryption

In this section we reveal the structure of a communication sent by the bot to the C&C server via a POST to `/gate.php`. We first extracted a message from the network trace we took in section 3.3. With the RC4 encryption key we extracted in section 3.5, we decrypted the payload of this message and performed an analysis of the structure of the message. In Figure 12, we illustrate the structure of the message as follows:



8E020000	0000000000000000	0C0000005B626D42FC682051D56D72A4
Message length	Unknown	Md5 hash value

20270000	00000000	0D010000	0D010000
Data type	Unknown	Data length	Data length

```
http://192.168.252.132/catalog/checkout_process.php
Referer:
http://192.168.252.132/catalog/checkout_confirmation.php
Keys: user@email.com123456 4408041234567893 Data:
cc_owner=Name cc_number_nh-dns=4408041234567893
cc_expires_month=01
cc_expires_year=10x=47y=3
```

Figure 12: A Decrypted Sample Message

1. The first 28-bytes is the message header. The first 4-bytes is the message length and the last 16-bytes is an MD5 hash of rest of the message.

2. The body of the message may contain any number of message entries consisting of:

(a) A 16-byte entry header where the first 4-bytes determine the type of the entry. This information is used by the control panel to determine in which table to save the information. The length of the entry is determined by the bytes beginning at byte eight; and,

(b) The stolen information or status information of the bot.

A key observation resulting from our analysis is that the encrypted communications of the Zeus botnet are vulnerable to the RC4 keystream reuse attack. Because a new initialization vector (IV) is not setup in every session, *i.e.*, the same RC4 keystream is reused to encrypt all messages. This vulnerability is similar to the attacks on the Wired Equivalency Privacy (WEP) protocol [14], which would allow an interested party to break the encryption of intercepted packets with relative ease.

3.7 Summary

The Zeus crimeware toolkit is an advanced tool used to generate very effective botnets that facilitate criminal activities for financial gain. The generated bot executables employ stealthy rootkit technology to evade detection of the malware on the host. In addition, the use of encrypted HTTP messages for communication makes it difficult to detect at the network level. Finally, the many layers of malware obfuscation further frustrate analysis efforts and make writing signatures more difficult.

In this chapter, we:

1. Provided an overview of the functionality of the Zeus crimeware toolkit;

2. Analyzed communications between the command-and-control (C&C) server and the infected hosts. The sequence of communications was illustrated, revealing the periodic nature of transmissions.

3. Presented a detailed reverse engineering analysis of the Zeus crimeware toolkit to unveil its underlying architecture and aid in mitigation efforts. This analysis illustrated the obfuscation Zeus employs and described the installation steps the malware takes upon infection of the host. We also revealed the static configuration structure from the binary;
4. Designed a tool to automate the recovery of the encryption key and the extraction of the configuration information from the bot executable; and,
5. Provided a breakdown for the structure of the Zeus botnet network messages where we identified a vulnerability.

The analysis of the C&C communications indicates that the RC4 algorithm is implemented insecurely making it vulnerable to a keystream reuse attack. In addition to the knowledge of the messaging structure, one can launch active countermeasures by interacting with the botnet's servers using the extracted encryption key. For example, by injecting falsified information into botnet communications for various purposes, such as defaming the botnet business model, reduces the effectiveness of the botnet. This attack is later described in Chapter 4. Future work could be to use the mechanism to extract encryption keys and the dynamic configuration file to identify Zeus' C&C servers.

Chapter 4

Defaming Botnet Toolkits

We have introduced the botnet toolkit problem and described the capabilities of the financially motivated Zeus crimeware toolkit. The Zeus analysis showed the difficulty in reverse engineering due to the sophistication of its obfuscation and the use of encryption. This motivated us to pursue an approach that could make use of the information learned from reverse engineering and address the problem of botnet toolkits as a whole. In this chapter, we introduce a botnet toolkit defamation process to target the profit model of these toolkits.

4.1 Introduction

Inspired by the bottom-up control of a biological food chain [52], we propose an approach of defeating a botnet toolkit through financial discouragement or prosecution of its end-users in order to defame and ultimately affect the top of the food chain, the malware authors. We first describe the Internet black market model for stolen credentials. Next, we present a generic framework describing the proposed approach and its two variations that defame a botnet toolkit from the security and profitability perspectives. To make the concepts more concrete, we then present a case study of the approach used on the Zeus crimeware toolkit analyzed in Chapter 3. For extracting necessary information for the defaming approach, we demonstrate two methodologies, namely, reverse engineering and behavioural analysis in the case study.

Chapter Organization: In section 4.2 we describe the Internet black market model our framework targets. Section 4.3, gives an overview of the defamation approach and its effects on the Internet black market. In section 4.4, we present a case study of how to apply this framework to the Zeus botnet toolkit. The following section, section 4.4.1, describes behavioural methodologies we tested on Zeus to invoke the framework. And finally, in section 4.5, we discuss several practical considerations with respect to our framework and summarize the chapter.

4.2 Internet Black Market Model

In creating the black market model we adapt equations 2 and 3 we reviewed in section 2.3.1, to address other stolen sensitive information such as credit card numbers and online financial institution credentials. We assume a profit model with four players: the malware authors who design and sell botnet toolkits; the botmasters who purchase the toolkit and use it to steal credentials; the buyers who purchase the stolen credentials in the hopes of making a profit; and, the money mules who eventually withdraw funds and then transfer, minus a commission, the funds back to the buyers.

The parameters of this profit model are:

1. A, the average amount stolen per credential. This amount should stay constant;
2. n, the number of credentials;
3. C_p, the commission, as a percentage of the amount stolen, that the money mule keeps from each transaction;
4. S, the sale price per credential in the black market;

5. M, the purchase price of the botnet toolkit in the black market; and,

6. N, the number of sales of the botnet toolkit.

For each player we describe the profit model that will be affected by the proposed framework:

The profit model for the money mule is:

$$max\,(Profit\_mule) = A \times n \times C_p \tag{4}$$

The profit model for the buyer of the credentials is:

$$max\,(Profit\_buyer) = A \times n \times (1 - C_p) - S \times n \tag{5}$$

The profit model for the botmaster is:

$$max\,(Profit\_botmaster) = S \times n - M \tag{6}$$

The frequency with which the botmaster must pay M is dependant on how often new releases of the toolkit are purchased.

Finally, the profit model for the toolkit author is:

$$max\,(Profit\_author) = M \times N \tag{7}$$

4.3 Framework

The goal of the proposed framework is to discredit a botnet toolkit and ultimately reduce its sales to those who wish to create botnets. We attack the toolkits credibility on two fronts: profitability of the toolkit in respect to the use and sale of credit card information, online

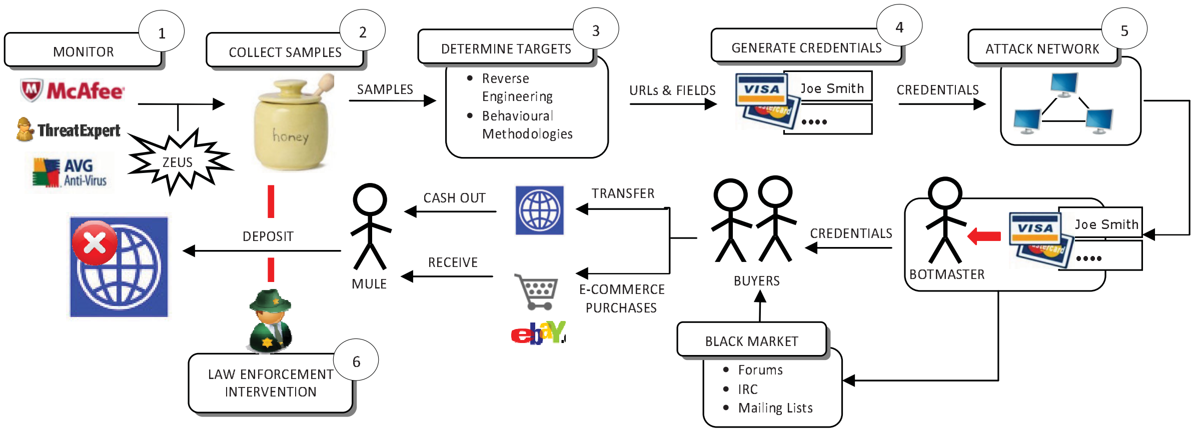


Figure 13: Botnet Toolkit Defamation Process

financial services credentials, and other sensitive information; and, security of the toolkit in respect to its ability to protect its users from prosecution. We propose two variants to this approach: reducing a toolkit's profitability by flooding it with false information, and making the toolkit insecure by submitting honeytokens to the botnets that aid in the arrest and prosecution of the end-users of this stolen information. The framework is illustrated in Figure 13, which shows the two variants and the surrounding process, which will be explained in details in the following.

4.3.1 First Variant

The intent of the first variant is to reduce the profitability of using a Zeus botnet for credential stealing by diluting the stolen identity information with specially crafted, false credentials. It works as follows:

1. Monitor current botnet toolkit trends and select the leading toolkit used for identity theft and fraudulent transactions as our target;

2. Acquire samples of botnet binaries from honeypots, antivirus vendors, financial institutions and security forums;

3. Determine targeted websites and targeted fields by analyzing malware samples;

4. Generate false identity information; and,

5. Submit identity information to the botmasters through the attack network.

The fraudulent credentials, along with legitimate credentials, are then sold on the black market. When the purchaser attempts to use the credentials, they will discover that many of them are invalid. Diluting the credentials affects the profit model by reducing the number of valid credentials used to gain a profit. This introduces a new variable into the equation, P_f, the probability that a credential is fake. The money mule was only ever paid for funds that were transferred to their account and would likely not be aware of fake credentials. The profit model for the buyer will be affected as follows:

$$max\,(Profit\_buyer) = A \times n \times (1 - P_f) \times (1 - C_p) - S \times n \tag{8}$$

The result of diluting the information sold, has a cascading effect; each subsequent party discredits the next party in the chain due to the loss in profits. Firstly, the purchased credentials generate less than expected profits for the buyer as many of the credentials are false. This is reflected in the new formula through a reduction of profits resulting from $1 - P_f$. Next, the buyer will discredit the seller, who is also the botmaster, in the Internet black market where the purchase was made because many of the credentials purchased were false. This will affect future profits by either reducing the sale price per credential, S, and/or reducing total sales, n, as this seller's wares are less reliable. Finally, the sellers, or botmasters, will attribute the problem to the toolkit used to create their botnet and discredit this toolkit in the black market due to its poor performance. In addition, botmasters will no longer purchase toolkit updates or future releases of that specific brand or author. This will affect future sales of the toolkit, by its author, resulting from the reduced sale price of the toolkit, M, and/or the reduced total sales, N.

We now establish a negative effect not just on the economy of the Internet black market, but directly on the incomes of the malware authors. If future sales of the toolkit can be sufficiently reduced, then the authors may decide that supporting their toolkit is no longer profitable. We assume that authors of these toolkits are the minority of Internet criminals and if, through these attacks, we can dissuade them from creating botnet toolkits for material gain, we will reduce the amount of innovation these botnets possess in future attacks; therefore, making them obsolete. In addition, this approach may frustrate potential botmasters from purchasing toolkits by continually reducing the effectiveness of the botnets that these toolkits create. This framework repeats, re-evaluating the currently most prevalent and dangerous botnet toolkit and reapplying the approach. As a toolkit loses its popularity, the framework adjusts to tackle the next leading threat.

4.3.2 Second Variant

The second variant of our framework is an extension of the first. The intent of this secondary approach is to harm the sense of security for the end-users, the money mules, of the stolen information. This approach contains one additional step that occurs after the credentials propagate through the Internet black market:

6. Make arrests when the fraudulent accounts are used.

To facilitate this additional step, the fraudulent accounts must be, or appear as, real, active accounts with accessible funds. Coordination between law enforcement and the various financial institutions is paramount for this approach to succeed. The criminals must be convinced that the accounts are valid and the transactions are working. This can be accomplished by providing funds to dummy accounts under the control of law enforcement of financial institutions. Law enforcement's role is to monitor these accounts and attempt to arrest individuals as they try to extract the funds. Funds can be extracted by performing online purchases, transferring funds to other accounts (e.g., financial institutions or Paypal) and through wire-transfers. We postulate that, with foreknowledge of the accounts being used, there is an increased likelihood that law enforcement can successfully apprehend and charge money mules with an offence.

To analyze the impact of apprehension and prosecution of money mules, we operate under the assumption that the players in the identified Internet black market will only continue these activities if it remains profitable for them to do so. We simplify the equation 1, reviewed in section 2.3, to only address monetary benefit, M_b, the opportunity cost of conviction, O_{cm}, the probability of arrest, P_a, and the probability of conviction, P_c, excluding the psychological benefits and opportunity costs. Therefore, a cybercrime occurs if:

$$M_b > O_{cm} P_a P_c \tag{9}$$

We assume a direct correlation between the probability of arrest and conviction, and the number of transfers a money mule performs. Therefore, this correlation does not affect the ratio in equation 9. The increase in probability of conviction resulting from our framework is not offset by an increase in the money mules profits. In order for the money mules to stay profitable, we can assume they will demand a higher commission. In addition, convictions will remove some end-users from further involvement in the black market. Reduction in supply of potential end-users, as well as increased concern over security will raise the cost of hiring money mules; this is reflected in an increased commission, C_p, charged for services. Increased costs, in turn, reduce future profits of the buyers; and, much like the first variant, these buyers will continue to propagate the effects of arrests within their ranks, by attacking the credibility of the source of the identity information they purchased. This impugns the reputations of the sellers of the stolen identity information and drives down the market price for the information. Finally, sellers of stolen identity information, who are also botmasters, will discredit the toolkits they used to create their botnets. With a damaged reputation, the toolkit will lose future sales.

4.3.3 Technical Approach

We now discuss methods to determine the targeted websites and their application in defaming Zeus. The first method of identifying these websites, is to reverse engineer malware samples to determine the targeted sites and, in particular, the identity information they are after. In the case of botnet toolkits, malware samples generated by the same version of the toolkit will have a similar code structure even though they may appear completely different at first glance. This will allow a reverse engineering analysis to be compiled into scripts that may be run on each malware sample to repeat the reverse engineering steps. In section 4.4, we describe our results from such an analysis. In section 4.4.1, we also discuss behavioural methodologies to determine the targeted websites that stemmed from our analysis of Zeus and have roots in side-channel attacks [41]. These methodologies look for anomalous system activity related to the browser process to indicate when web page harvesting occurs. In particular, we are trying to determine when the malware stores identity information on the hard drive prior to sending it to the botmaster.

4.4 Zeus Case Study

We now illustrate the technical details of the framework through examples of how it would be applied to the Zeus botnet toolkit. This section includes the steps needed to infiltrate the botnet and inject fake credentials into the black market, and the behavioural methodologies that could be applied to future toolkits. We limit our analysis of Zeus to extracting the encryption key and the targeted website URLs for use in our framework. With these two pieces of information, we are able to join an existing Zeus botnet, pretend to be an active member, and submit the falsified identity information.

We use scripts written during the reverse engineering analysis to automatically extract the encryption key and the configuration information embedded in the bot binary. This information directs us to a URL, which contains the dynamic configuration for the botnet. This file must first be decrypted using the extracted key. It contains a list of targeted URLs and their respective extraction rules, blacklisted websites and websites that have injected web content.

We begin the process by downloading the dynamic configuration file from the web location `url_config`. Once downloaded, the dynamic configuration file can be decrypted using the RC4 algorithm with the substitution table provided. Next, we extract the URLs that are being targeted from this file. These URLs satisfy step 3 of our framework and are the input to the identity information generator in step 4. We now use a machine, infected with the binary sample from which we extracted the URLs, and submit our generated identity information. The Zeus bot will detect the website, extract the submitted credentials and send them to the botmaster. Figure 13 outlines the rest of the propagation process. According to our model, after the botmaster receives the credentials, they will sell the information in the black market to buyers of the stolen information. Buyers will transfer funds to the money mules, using the stolen credentials. At this stage, law enforcement agencies may perform arrests. By monitoring the use of the credentials injected in the botnet by the framework, law enforcement agencies can target the money mules who receive the funds and perform transfers back to the buyers.

4.4.1 Behavioural Methodologies

Although reverse engineering provided us with exact results for the targeted URLs, it can be a time-consuming process, as malware authors utilize obfuscation techniques to prevent binary snooping. Since we are interested in applying our method to all families of malware, not just Zeus, we pursued behavioural methodologies in the hopes of automating a process to extract targeting rules from malware samples.

Behavioural methodologies may allow us to determine the URLs for any malware family, as they detect the URLs independently of the binary structure and the obfuscation techniques employed by the malware. These techniques may aid in the proposed framework by providing the details of precisely how a website is targeted, including the exact fields and inputs necessary to submit the generated credentials through the framework.

We evaluate various techniques to detect anomalous activity that, if successful, cover a broader range of malware . We test these methodologies on Zeus to check for their validity.

Assumptions

We make the following assumptions in order to create a data stealing model:

1. Data is stolen when a web browser submits information to a website;

2. The malware will be selective in what data it steals either through positive action, by specifying websites from which to harvest, or blacklisting websites whose information is not wanted. We make this assumption as recording all form grabbing malware may be less stealthy than only stealing what is of use. For example, the botmaster may not want web searches performed by the user as they play no part in the profit model.

3. Botmasters may also want to extract supplementary information from websites to determine if the captured form information is valuable. For instance, when a user logs into a banking website and views their account, the malware will record the login credentials and, if instructed, will also record the account balance. The botmaster is then aware of available funds without having to log in to the account and may use this information when they attempt to sell the credentials.

From our assumptions we have created the following data stealing model:

1. Data is recorded when a browser submits a `POST`, with the user input, to a website. Data may also be recorded during a `GET`, when query fields are present in the URL. Although malware may have this capability, it is unlikely that sensitive information is contained within a query string;

2. The malware will only record information for a subset of websites the victim visits;

3. Additional form fields and text/images may be presented to the victim by modifying the page content before it is rendered by the browser. The malware will use a rule set to determine which page to inject the additional information and where on the page to put it. The form fields may be used to gather additional personal information from the victim; and,

4. Data from a rendered page can be extracted with a pattern matching rule set.

Test Methods

From our proposed data stealing model, we are interested in detecting when a webpage's form and page data are extracted, and when additional page content is injected. In the case of Zeus, these behaviours would demonstrate the webpage is not blacklisted. A series of tests is run where a web browser navigates to potentially targeted websites using a system that is in a clean or infected state for the purpose of extracting behavioural characteristics for each website visited. When sufficient difference is found in the measured metrics between the system states, we count it as evidence that the current browsing actions are being targeted. The extraction process would, from a set of websites, determine when a page is blacklisted, when a page's specific details are extracted, or when a page's content is modified. This effectively creates a targeting configuration model of the malware samples analyzed. A list of potentially targeted websites is required to be used during the tests. To reduce effort on compiling this list, we let attackers determine the targeted domains through their phishing efforts which are then compiled into a list of phishing brands. Since the purpose of phishing is to also harvest identity information, then the targeted brands should be very similar, or likely a superset, to those targeted by identity theft botnets. This approach is left as future work.

We test various behavioural methodologies which use anomaly detection of windows performance variables gathered with PerfMon [49] and of processor activity gathered with ProcMon [49]. From PerfMon we test approaches that are based on performance monitoring counters that are specific to the web browser process or to the system as a whole.

- Disk read bytes / second
- Disk write bytes / second
- Processor utilization
- Processor utilization by the web browser
- Web browser read bytes / second
- Web browser write bytes / second

For the purposes of Zeus, and other HTTP botnets, we monitored writes to the filesystem using ProcMon. Anomalous filesystem writes may indicate stolen information being stored for later transmission to the botmaster. For our tests, we utilized a Windows XP virtual machine that is in one of the following five states, referred to as state 1 through 5:

1. Clean: the machine has not been infected with Zeus.
2. Infected w/ no targeting: the machine has been infected with Zeus; however, no specific rules for the test URL have been included.

3. Infected w/ injected content: the machine has been infected with Zeus and rules to inject an additional login form field are included in Zeus' configuration for the test URL.

4. Infected w/ page content extraction rules: the machine has been infected with Zeus, and rules to record specific page content are included in Zeus' configuration for the test URL.

5. Infected w/ URL filtered: the machine has been infected with Zeus and blacklist rules are included in Zeus' configuration to not record form information for the test URL.

Two general tests were run on the virtual machine to evaluate the behavioural methodologies. They are designed to evaluate if a given test URL exhibits anomalous behaviour in one of the monitoring tools, PerfMon or ProcMon, during browsing activities.

Test 1

1. Start the web browser.

2. Start monitoring tool (PerfMon or ProcMon).

3. Navigate to a test URL.

4. Stop monitoring tool.

Test 1 can be performed on states 1, 2, 3 and 4 of the virtual machine and is used to determine if a URL has additional content injected into its page and/or if a URL has page data being recorded by the malware.

Test 2

1. Start the web browser.

2. Navigate to a test URL which contains form fields.

3. Start monitoring tool (PerfMon or ProcMon).

4. Fill out form fields.

5. Submit form.

6. Stop monitoring tool.

Test 2 can be performed on states 1, 2 and 5 of the virtual machine and is used to determine if a URL's form data is being recorded.

We made use of two different websites in the following experiments as we were interested in identifying different behaviours. We used a website with a login form for Experiments 1 and 3, as they require form-field entries; whereas in Experiment 2, we use a website of mostly textual content as we were interested in identifying when page content was extracted. These tests are not correlated to one another; rather, they are used to identify different mechanisms of the data stealing model to further aid in automating extraction of targeting rules from malware samples.

- Experiment 1: Navigate to GMail [20] using state 1, 2 and 3 of the virtual machine. For each state we utilized Test 1 with `mail.google.com` as the URL. We first performed the test using ProcMon as the monitoring tool. We then repeated the experiment using PerfMon as the monitoring tool and repeated this second test three times. In addition, for each of the states we captured the page source.

  In this experiment, we want to test if we can identify when additional page content is injected into the page displayed to the user. For state 3, we modified `webinjects.txt` to insert an additional form field into GMail for the victim's SIN. This is similar to the scenario where an attacker may want to gain additional identity information from a victim.

- Experiment 2: Navigate to Wikinews [17] using state 1, 2 and 4 of the virtual machine. For each state we utilized Test 1 with `en.wikinews.org` as the URL. We first performed the test using ProcMon as the monitoring tool. We then repeated the experiment using PerfMon as the monitoring tool and repeated this second test three times.

  In this experiment, we want to test if we can identify when page content is being extracted by the malware. For state 4, we modified `webinjects.txt` to extract information from the webpage displayed to the victim. This is similar to the scenario where an attacker may want to gain sensitive information about a user's online account that is displayed after authentication (*e.g.*, online bank account total).

- Experiment 3: Log into GMail using states 1, 2 and 5 of the virtual machine. For each state we utilized Test 2 with `mail.google.com` as the URL. We first performed the test using ProcMon as the monitoring tool. We then repeated the experiment using PerfMon as the monitoring tool and performed this second test three times.

  In this experiment, we want to test if we can identify when form data is being extracted by the malware. For state 5, we modified `webinjects.txt` to not extract information from GMail. As Zeus by default extracts all form data, this scenario is testing for blacklisting.

The results of running the experiment on the different states are then compared to the specific configuration rules used in the malware.

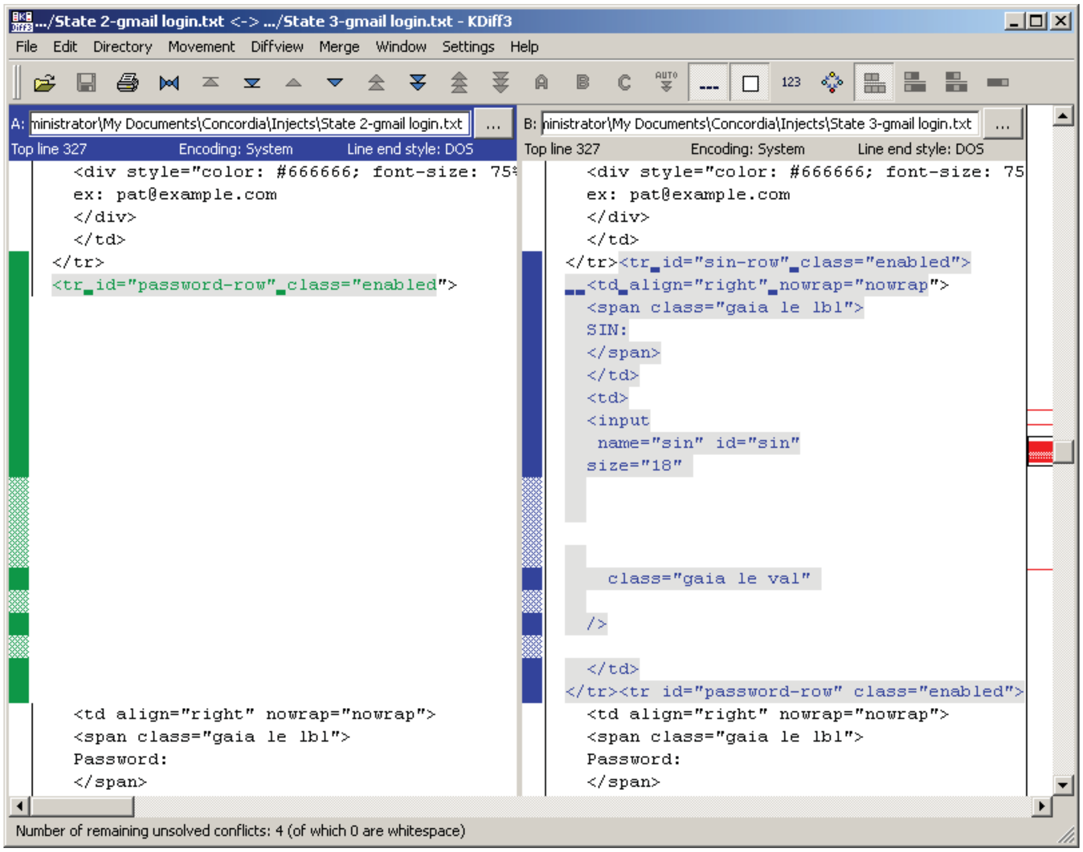
Results

We first examine Experiment 1. The result from running the experiment with ProcMon provides a list of paths where Internet Explorer [38] wrote data. Since Experiment 1 executes a navigation experiment where the machine is either in one of state 1, 2, or 3, it is unsurprising there are no anomalous entries in the path lists between the three states (see tables 4, 5 and 6 in the Appendices). We define what an anomalous path is in the appropriate experiment.

When executing the experiment with PerfMon we are not provided with any concrete conclusions as to whether page content has been injected and displayed to the user. We believe there are two main reasons this experiment failed. First, the recorded metrics are far too coarse for detecting the subtle system changes that injected page content produces. PerfMon is designed as a system performance monitor meant to aid in detecting performance issues. Its detectors give results that are an average over a time period that can only be as short as one second. The potential spikes in system activity we are looking for are likely lost in this timeframe. Second, we observed that system resources as a whole increased when the system is infected by malware. Any change in those metrics that is a result of the malware actively injecting content or extracting information, as is the case of the other experiments, is lost in the increased overall activity produced by the malware's rootkit functionality. It is our belief that this is endemic of all rootkits as they attach themselves to all or most running processes and increase overall system usage.

For this experiment we also recorded the page source for `mail.google.com`. Even though Zeus can inject any type of page content, our main concern is determining when additional form fields have been injected. A simple side-by-side text comparison using KDiff3 [12] shows that, in state 3, an additional field has been added to the login form (see Figure 14).

In Experiment 2, we are provided with some useful results from ProcMon (see tables 7, 8 and 9 in the Appendices). The path list for the three states is very similar as the folder structure and filenames have little difference. The one exception is a set of writes to `C:/WINDOWS/system32/lowsec/user.ds`. As mentioned in the Zeus


.../State 2-gmail login.txt <-> .../State 3-gmail login.txt - KDiff3
File Edit Directory Movement Diffview Merge Window Settings Help
A B C
AUTO
123
...
...

A: ministrator\My Documents\Concordia\Injects\State 2-gmail login.txt

Top line 327 Encoding: System Line end style: DOS

```
    <div style="color: #666666; font-size: 75%
    ex: pat@example.com
    </div>
    </td>
  </tr>
  <tr id="password-row" class="enabled">
















    <td align="right" nowrap="nowrap">
    <span class="gaia le lbl">
    Password:
    </span>
```

B: ministrator\My Documents\Concordia\Injects\State 3-gmail login.txt

Top line 327 Encoding: System Line end style: DOS

```
    <div style="color: #666666; font-size: 75
    ex: pat@example.com
    </div>
    </td>
  </tr><tr id="sin-row" class="enabled">
  <td align="right" nowrap="nowrap">
    <span class="gaia le lbl">
    SIN:
    </span>
    </td>
    <td>
    <input
     name="sin" id="sin"
    size="18"




      class="gaia le val"

    />

    </td>
  </tr><tr id="password-row" class="enabled">
    <td align="right" nowrap="nowrap">
    <span class="gaia le lbl">
    Password:
    </span>
```

Number of remaining unsolved conflicts: 4 (of which 0 are whitespace)

Figure 14: KDiff3 Comparison

case study, this is the location where extracted page and form data is stored prior to being transmitted to the botmaster. We can deduce that information from the web location `en.wikinews.org` was in fact extracted and submitted to the botmaster.

Again, PerfMon did not provide useful output for deducing when a page is harvested and the information is stored on the local hard drive. The shear number of writes that occur when viewing a website effectively mask this behaviour. In addition, page caching and new content being sent on subsequent visits adds further ambiguity to the results. Creating a more controlled environment where we cache the web pages content and ensure that each subsequent visit does not receive variable data may help to refine the results; however, it is likely still that these performance metrics will not prove useful.

The purpose of Experiment 3 is to determine when form data is captured. As Zeus captures all form data by default, we are in fact determining when form data is being specifically excluded. Regardless, running the experiment on a set of potentially targeted websites will return a subset of sites that have their data being recorded. As in Experiment 2, we again are provided with a set of writes to `C:/WINDOWS/system32/lowsec/user.ds` (see tables 10, 11 and 12 in the Appendices), indicating that either form information is being recorded; and/or, the page, rendered in the web browser after the form data is submitted, has data being extracted by the malware. Also in state 5, there is no data written to disk confirming that blacklisting `mail.google.com` did indeed cause the expected behaviour. We can safely deduce that form data has been captured during the experiment, while in state 2, as it is unlikely that the botmaster is only interested in information in the rendered page resulting from a log in and not the login credentials themselves. As in the previous experiments, PerfMon was ineffective for the same reasons.

Extraction Process

We can partially automate the process to extract a malware's configuration using the WebSpec API from Web Application Testing in Java (Watij) [28], a cross-platform open source java project that enables easy scripting of popular web browsers. We run the following algorithm multiple times on each sample page and store the results for later comparison.

1. Open web browser.
2. Start PerfMon.
3. Navigate to web location.
4. Stop PerfMon.
5. Repeat steps 2-4 for each web location.
6. Close web browser.

Step 3 of our algorithm is more complicated than simply navigating to a web location. There may be necessary authentication steps that require test credentials provided by a potentially targeted company. Using the Webspec API we can create scripts to successfully navigate these sites and incorporate them into our algorithm. These same scripts would be used during the proposed defaming framework to submit credentials to the botmasters. See Figure 15 for the Java code to log into a PayPal account using a web browser, and recording a system log. The output is then converted into a comma-separated-value (CSV) file for further analysis. We only monitor disk writes that are initiated by Internet Explorer and we specify this configuration in `ProcmonConfiguration.pmc`.

We run this script on each of the specified virtual machine states in our experiments to generate our test data. In addition, after step 3, we can optionally save the page source for use in detecting web injects using the function `source()` exposed by the Webspec API.

```
// 1. Open JExplorer.
WebSpec spec = new WebSpec().ie();

// 2. Start ProcMon.
ProcessBuilder pb_procmon_start =
  new ProcessBuilder("Procmon.exe",
                     "/AcceptEula",
                     "/Quiet",
                     "/LoadConfig",
                     "ProcmonConfiguration.pmc",
                     "/BackingFile",
                     "sample.pml");
pb_procmon_start.start();

// 3. Navigate to website and wait for load to complete.
spec.open("[URL]");
spec.busy();

// Fill out credentials.
spec.find("input").with("name==‘login_email’")
                  .set("value=‘sample@sample.com’");
spec.find("input").with("name==‘login_password’")
                  .set("value=‘sample’");

// Submit login credentials.
spec.find("input").with("type==‘submit’")
                  .with("name==‘submit.x’").click();

// 4. Stop monitoring.
ProcessBuilder pb_procmon_end =
  new ProcessBuilder("Procmon.exe", "/Terminate");
pb_procmon_end.start();

// 5. Close JExplorer
spec.closeAll();

// Convert log into CSV format.
ProcessBuilder pb_procmon_csv =
  new ProcessBuilder("Procmon.exe",
                     "/NoConnect",
                     "/AcceptEula",
                     "/Quiet",
                     "/OpenLog",
                     "sample.pml",
                     "/SaveAs",
                     "sample.csv");
pb_procmon_csv.start();
```

Figure 15: Web Navigation Testing Script

After gathering the test data we can execute a test to determine whether information stealing occurs. The process takes the output from the Webspec executions and searches for the path `C:/WINDOWS/system32/lowsec/user.ds`. This path is specific to the malware samples we analyzed and would need to be configured for each sample. If the path is detected, then an extraction occurred and the test URL is added to the site list for our defamation attack.

To determine web page injections in an automated fashion, a comparison between the source collected from state 1 and 3 is needed. Comparing hash values of the sources collected is sufficient to indicate injected content, however, if the injected content includes a form field we require this additional information to instrument our attack. After confirming the sources are different by a hash check, we perform an additional check for those test sites that include a form submission process. The form used for submission in state 1 and 3 is compared by extracting the form fields from each source and comparing the result set. Additional fields in the state 3 version would then indicate form injection and provide our attack with an additional input requirement. Future work is needed to automate this process and further evaluate its effectiveness as explained in the following section.

4.5 Summary

In this chapter we introduced a framework to combat identity theft toolkits. In it, we performed an economic analysis of the effects of defamation. The technical challenge of this approach is to determine the websites that are targeted by instances of a botnet toolkit. We discussed reverse engineering results from an analysis we performed on Zeus that allows us to automate this process for Zeus binary samples. In addition, behavioural methodologies were proposed and evaluated.

The advantage of this framework is that it targets the weakest point of a botnet food chain (the end-users). The cascading effect will eventually affect the top level of the chain (the toolkit author) by diminishing his/her profits from future sales of the toolkit. In addition, since we are attacking the business model, malware authors would need to change how they do business to circumvent our attack, which, we postulate, is more difficult than modifying the implementation of their toolkits.

A major obstacle with our approach is that it is unclear whether we are diluting many botnets with false credentials from many botmasters, or many botnets from one or a few botmasters. Even if we collect many binaries that use different C&C servers, they all may belong to the same botmaster. Diluting the botnets that belong to our collected samples may only be affecting one or a few botmasters business models while other botmasters that use the same toolkit are unaffected. Currently we rely on an assumption that the most active botnets will likely be represented in our acquired malware set and so, we would have an affect on the most egregious actors and therefore the most impact in the marketplace.

As this approach is an exploration on paper, future work would be needed to provide further details on implementing the attack. For the first step, Internet black market forums and IRC channels can be monitored to identify what the popular toolkits are. For identifying targeted websites, further exploration into behavioural methodologies may provide more useful results. The main finding from our behavioural methodologies was the correlation between file writes to the local data storage and data being harvested from either form fields or page content. Although this provided some success, our findings did not illustrate when, in general, information is harvested. For example, if the malware does not write to disk, but rather immediately transmits to the drop location, then our finding would not apply. One potential method is to look for anomalous behaviour that occurs when operating system APIs that are known to be hooked by malware are called. Finally, further monitoring of black market forums and IRC channels may provide law enforcement with additional information that would be useful during apprehension, such as the geographic region in which the mule operates and which wire-transfer service they will use.

To evaluate the approach and the effectiveness of the attack, two locations may be monitored. The first is the same forums and IRC channels where popular toolkits are identified; the second is the drop locations where the stolen information is sent. By monitoring these two locations, one could determine at what rate false information needs to arrive relative to all information being sent to the drop location, in order to have an effect on the value of harvested credentials and ultimately the value of the botnet toolkit.

Chapter 5

Conclusion

The Internet continues to be plagued by malicious actors using botnets to perform identity theft and other attacks. While many solutions have been proposed, this problem is still prevalent.

In this thesis, we provided background on botnets and the various attacks they can launch. In addition, we looked at the existing research to address this problem. Botnet detection through host-based and network-based detection was also described. Moreover, we discussed the Internet black market and its motivations, given that the modern botnet is financially driven and is involved in the black market at many levels.

One of the main contributions for this thesis was a reverse engineering analysis of the Zeus crimeware toolkit. We described the functionality of a particular botnet, Zeus, and provided an in-depth analysis of its process to infect a machine, harvest a victim's private information and communicate it back to the C&C server. The power and simplicity of operating the toolkit was highlighted, as was the sophistication it employed to evade detection and frustrate analysis efforts. We also presented an approach to extract key configuration information from the malware that reveals the C&C server and other key servers necessary to operate a Zeus botnet. We designed a tool to automate this extraction using emulation techniques that do not require the malware to be running in a system. We also breakdown the structure of the Zeus botnet communications. An advantage of using this tool to unpack a binary and extract information is that, if ported into a standalone application, it can be executed as a batch process. This could be used to quickly identify all encryption keys, drop locations and C&C servers used by malware that has targeted an organization. An organization can use this information to block traffic and determine what information is being stolen. A point of interest is the weak implementation of RC4 that Zeus employs. This shows that while malware authors produce sophisticated code they have vulnerabilities in their programs much like the legitimate software industry. This thesis provides valuable insight into future reverse engineering efforts and combating newer versions of Zeus and its relatives through new means of detection and mitigation.

The Zeus analysis showed the difficulty in reverse engineering due to the sophistication of its obfuscation and the use of encryption. This led us toward the other main contribution of this thesis, a novel approach to defaming botnet toolkits: making use of the information learned from reverse engineering and addressing the problem of botnet toolkits as a whole. Two economic attacks were proposed targeting end-users of stolen credentials to create a cascading effect on the ultimate target, the authors of botnet toolkits. Attacking the Internet black market and ultimately the wallets of malicious actors may prove to be a strong approach to handling these threats since, although, malware functionality can be changed, modifying the business model may prove more difficult. A case study using Zeus was explored and various behavioural methodologies were tested but showed initial success in only a few experiments.

Reverse engineering malware is a time consuming task. Future work is to develop automated techniques that can improve the efficiency of malware analysis. In the Zeus analysis, the tool which automates extraction of configuration information relies on the disassembler, IDAPro. To truly create a batch process, the functionality would need to be migrated into a standalone application. This would require additional logic at the start of the tool to identify the entry point in the binary executable file.

In addition, the knowledge gained from our analysis can be used to interact with a botnet for a variety of purposes including our defamation approach. Falsified information could be submitted directly to the botmaster by crafting packets using the structure we deduced from analysis and the extracted encryption key. Additional research could be performed to explore other economic attacks against the Internet black market and specifically against the malware authors whose wares enable these sophisticated attacks. Evaluating the effectiveness of these attacks remains an outstanding question. Creating tools to automatically monitor and mine information, such as the popularity of a toolkit, would provide valuable input for any evaluation. While the behavioural methodologies only provided some initial success, further exploration into these areas may aid in automated attacks against the botnet financial model.

Bibliography

[1] Distributed hash table. Available at: http://en.wikipedia.org/wiki/Distributed_hash_-table.

[2] Wireshark. Available at: http://www.wireshark.org/.

[3] Pedram Amini. Paimei - reverse engineering framework. Available at: http://code.google.com/p/paimei/.

[4] Paul Barford and Vinod Yegneswaran. An inside look at botnets. In *Malware Detection*, volume 27 of *Advances in Information Security*, pages 171–191. Springer US, 2007.

[5] Ulrich Bayer, Imam Habibi, Davide Balzarotti, Engin Kirda, and Christopher Kruegel. A view on current malware behaviors. In *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, LEET'09, pages 8–8, Berkeley, CA, USA, 2009. USENIX Association.

[6] Hamad Binsalleh, Thomas Ormerod, Amine Boukhtouta, Prosenjit Sinha, Amr Youssef, Mourad Debbabi, and Lingyu Wang. On the analysis of the zeus botnet crimeware toolkit. In *Proceedings of the The Eight International Conference on Privacy, Security and Trust (PST '10)*, Ottawa, Ontario, Canada, August 2010. IEEE Software.

[7] Ken Chiang and Levi Lloyd. A case study of the rustock rootkit and spam bot. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, Berkeley, CA, USA, 2007. USENIX Association.

[8] Hyunsang Choi, Hanwoo Lee, Heejo Lee, and Hyogon Kim. Botnet detection by monitoring group activities in dns traffic. In *Proceedings of the 7th IEEE International Conference on Computer and Information Technology*, pages 715–720, Washington, DC, USA, 2007. IEEE Computer Society.

[9] Bryce Cogswell and Mark E. Russinovich. Rootkitrevealer. Available at: http://technet.microsoft.com/en-us/sysinternals/bb897445.aspx.

[10] Oracle Corporation. Mysql. Available at: http://www.mysql.com/.

[11] Neil Daswani and Michael Stoppelman. The anatomy of clickbot.a. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, Berkeley, CA, USA, 2007. USENIX Association.

[12] Joachim Eibl. Kdiff3. Available at: http://kdiff3.sourceforge.net/.

[13] Gergely Erdelyi, Elias Bachaalany, and Ero Carrera. Idapython: an ida pro plugin. Available at: http://code.google.com/p/idapython/.

[14] Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of rc4. In S. Vaudenay and A.M. Youssef, editors, *Selected Areas in Cryptography*, volume 2259 of Lecture Notes in Computer Science., pages 1–24. Springer, 2001.

[15] Richard Ford and Sarah Gordon. Cent, five cent, ten cent, dollar: Hitting botnets where it really hurts. In *Proceedings of New Security Paradigms Workshop*, pages 3–10, 2006.

[16] Python Software Foundation. Python programming language. Available at: http://www.pythong.org/.

[17] Wikimedia Foundation. Wikinews. Available at: http://en.wikinews.org/.

[18] Jason Franklin, Vern Paxson, Adrian Perrig, and Stefan Savage. An inquiry into the nature and causes of the wealth of internet miscreants. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS'07)*, pages 375–388, 2007.

[19] Dan Goodin. The balkanization of storm worm botnets, October 2007. [Online]. Available at: http://www.theregister.co.uk/2007/10/15/storm_trojan_balkanization/, accessed April, 2011.

[20] Google. Gmail. Available at: http://mail.google.com/.

[21] Google. Google safe browsing. Available at: http://www.google.com.

[22] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the USENIX Security Symposium*, pages 139–154, Berkeley, CA, USA, August 2008. USENIX Association.

[23] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. Bothunter: detecting malware infection through ids-driven dialog correlation. In *SS'07: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, pages 1–16, Berkeley, CA, USA, 2007. USENIX Association.

[24] Greg Hoglund and James Butler. *Rootkits: Subverting the Windows Kernel*. Addison-Wesley, 2006.

[25] Thorsten Holz, Markus Engelberth, and Felix Freiling. Learning more about the underground economy: A case-study of keyloggers and dropzones. *Computer Security ESORICS 2009*, pages 1–18, 2009.

[26] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix Freiling. Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In *LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 1–9, Berkeley, CA, USA, 2008. USENIX Association.

[27] Anestis Karasaridis, Brian Rexroad, and David Hoeflin. Wide-scale botnet detection and characterization. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, Berkeley, CA, USA, 2007. USENIX Association.

[28] Brian Knorr. Web application testing in java (watij) - webspec api. Available at: http://watij.com/webspec-api/.

[29] Nir Kshetri. The simple economics of cybercrimes. *IEEE Security & Privacy*, 4(1):33–39, January 2006.

[30] Wenke Lee, Cliff Wang, and David Dagon, editors. *Botnet Detection: Countering the Largest Security Threat*, volume 36 of *Advances in Information Security*. Springer-Verlag New York, 2008.

[31] Shujun Li and Roland Schmitz. A novel anti-phisihng framework based on honeypots. In *Proceedings of the 4th annual Anti-Phishing Working Groups eCrime Researchers Summit*, September 2009.

[32] Zhen Li, Qi Liao, Andrew Blaich, and Aaron Striegel. Fighting botnets with economic uncertainty. *Journal of Security and Communication Networks*, 2010.

[33] Zhen Li, Qi Liao, and Aaron Striegel. Botnet economics: Uncertainty matters. In *Proceedings of the 7th Workshop on the Economics of Information Security (WEIS'08)*, 2008.

[34] Lei Liu, Songqing Chen, Guanhua Yan, and Zhao Zhang. Bottracer: Execution-based bot-like malware detection. *Information Security*, pages 97–113, 2008.

[35] CheckPoint Software Technologies LTD. Zonealarm. Available at: http://www.zonealarm.com/.

[36] Lorenzo Martignoni, Elizabeth Stinson, Matt Fredrikson, Somesh Jha, and John Mitchell. A layered architecture for detecting malicious behaviors. *Recent Advances in Intrusion Detection*, pages 78–97, 2008.

[37] Ellen Messmer. America's 10 most wanted botnets, July 2009. Availabe at: http://www.networkworld.com/news/2009/072209-botnets.html, last accessed: February 2011.

[38] Microsoft. Internet explorer. Available at: http://www.microsoft.com/windows/ie/.

[39] Microsoft. Listing the files in a directory. Available at: http://msdn.microsoft.com/en-us/library/windows/desktop/aa365200

[40] Microsoft. Smartscreen filter. Available at: http://www.microsoft.com/.

[41] David Molnar, Matt Piotrowski, David Schultz, and David Wagner. The program counter security model: Automatic detection and removal of control-flow side channel attacks. In *Proceedings of the 8th International Conference on Information Security and Cryptology*, pages 156–168, Seoul, Korea, December 2005.

[42] David Moore, Colleen Shannon, Douglas J. Brown, Geoffrey M. Voelker, and Stefan Savage. Inferring internet denial-of-service activity. *ACM Trans. Comput. Syst.*, 24(2):115–139, 2006.

[43] Jose Nazario. Blackenergy ddos bot analysis. Technical report, Arbor Networks, 2007.

[44] Markus Franz Xaver Johannes Oberhumer, László Molnár, and John F. Reiser. Upx - the ultimate packer for executables. Available: http://upx.sourceforge.net/.

[45] Thomas Ormerod, Lingyu Wang, Mourad Debbabi, Amr Youssef, Hamad Binsalleh, and Prosenjit Sinha. Defaming botnet toolkits: A bottom-up approach to mitigating the threat. In *Proceedings of the International Conference on Emerging Security Information, Systems and Technologies, SECURWARE*, Mestre, Italy, 2010. IEEE Press.

[46] R. Perdisci, I. Corona, D. Dagon, and W. Lee. Detecting malicious flux service networks through passive analysis of recursive dns traces. In *25th Annual Computer Security Applications Conference (ACSAC 2009)*, 2009.

[47] Anirudh Ramachandran, Nick Feamster, and David Dagon. Revealing botnet membership using dnsbl counter-intelligence. In *Proceedings of USENIX SRUTI'06*, pages 49–54, July 2006.

[48] Ron Rivest. Rc4. Available at: http://en.wikipedia.org/wiki/RC4.

[49] Mark E. Russinovich and David A. Solomon. *Microsoft Windows Internals: Windows Server 2003, Windows XP, and Windows 2000*. Microsoft Press, 4th edition, 2004.

[50] Hex-Rays SA. Idapro - multi-processor disassembler and debugger. Available at: http://www.hex-rays.com/idapro/.

[51] Agnitum Security. Outpost firewall. Available at: http://www.agnitum.com/.

[52] J.B. Shurin, D.S. Gruner, and H. Hillebrand. All wet or dried up? real differences between aquatic and terrestrial food webs. *Proceedings of the Royal Society B: Biological Sciences*, 273(1582):1–9, 01 2006.

[53] snaker, Qwerton, Jibz, and xineohP. Pe identifier. Available at: http://www.peid.info/files/PEiD-0.95-20081103.zip.

[54] Elizabeth Stinson and John C. Mitchell. Characterizing bots' remote control behavior. In *4th GI International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, DIMVA 2007, July 12,2007 - July 13*, volume 4579 LNCS, pages 89–108, Lucerne, Switzerland, 2007. Department of Computer Science, Stanford University, Stanford, CA 94305, Springer Verlag.

[55] Brett Stone-Gross, Christopher Kruegel, Kevin Almeroth, Andreas Moser, and Engin Kirda. Fire: Finding rogue networks. In *Proceedings of Computer Security Applications Conference (ACSAC '09)*, December 2009.

[56] P. Szor. *The Art of Computer Virus Research and Defense*. Addison-Wesley, 2005.

[57] Haining Wang, Danlu Zhang, and Kang G. Shin. Detecting syn flooding attacks. In *Proceedings of IEEE Infocom '02*, June 2002.

[58] Ting-Fang Yen and Michael K. Reiter. Traffic aggregation for malware detection. In *Proceedings of the Fifth GI International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'08)*, pages 207–227, 2008 2008.

[59] Heng Yin, Dawn Song, Manuel Egele, Christopher Kruegel, and Engin Kirda. Panorama: capturing system-wide information flow for malware detection and analysis. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 116–127, New York, NY, USA, 2007. ACM.

Appendices

Path
%tempinet%\Content.IE5\S4KTHFSE\ServiceLogin[1].htm
%tempinet%\Content.IE5\S4KTHFSE\ServiceLogin[1].htm
%tempinet%\Content.IE5\S4KTHFSE\ServiceLogin[1].htm
%tempinet%\Content.IE5\S4KTHFSE\ServiceLogin[1].htm
%tempinet%\Content.IE5\S4KTHFSE\ServiceLogin[1].htm
%tempinet%\Content.IE5\S4KTHFSE\ServiceLogin[1].htm
%tempinet%\Content.IE5\S4KTHFSE\ServiceLogin[1].htm
%tempinet%\Content.IE5\RYAFS53V\mail_logo[1].png
%tempinet%\Content.IE5\RYAFS53V\mail_logo[1].png
%tempinet%\Content.IE5\RYAFS53V\mail_logo[1].png
%tempinet%\Content.IE5\RYAFS53V\mail_logo[1].png
%tempinet%\Content.IE5\LOYERILN\ga[1].js
%tempinet%\Content.IE5\LOYERILN\ga[1].js
%tempinet%\Content.IE5\LOYERILN\ga[1].js
%tempinet%\Content.IE5\LOYERILN\ga[1].js
%tempinet%\Content.IE5\LOYERILN\ga[1].js
%tempinet%\Content.IE5\LOYERILN\ga[1].js
%tempinet%\Content.IE5\LOYERILN\ga[1].js
%tempinet%\Content.IE5\LOYERILN\ga[1].js
%tempinet%\Content.IE5\RYAFS53V\sprite_mail_hp[1].png
%tempinet%\Content.IE5\RYAFS53V\sprite_mail_hp[1].png
%userdir%\Cookies\admin@accounts[2].txt
%userdir%\Cookies\admin@accounts[2].txt
%userdir%\Cookies\admin@accounts[2].txt
...

Table 4: State 1, Navigate to GMail File Writes

Path
%tempinet%\Content.IE5\LOYERILN\ServiceLogin[2].htm
%tempinet%\Content.IE5\RYAFS53V\ga[1].js
%tempinet%\Content.IE5\RYAFS53V\ga[1].js
%tempinet%\Content.IE5\RYAFS53V\ga[1].js
%tempinet%\Content.IE5\RYAFS53V\ga[1].js
%tempinet%\Content.IE5\RYAFS53V\ga[1].js
%tempinet%\Content.IE5\RYAFS53V\ga[1].js
%tempinet%\Content.IE5\RYAFS53V\ga[1].js
%tempinet%\Content.IE5\RYAFS53V\ga[1].js
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
...

Table 5: State 2, Navigate to GMail File Writes

Path
%tempinet%\Content.IE5\XV1BKGQZ\ServiceLogin[1].htm
%tempinet%\Content.IE5\LOYERILN\ga[1].js
%tempinet%\Content.IE5\LOYERILN\ga[1].js
%tempinet%\Content.IE5\LOYERILN\ga[1].js
%tempinet%\Content.IE5\LOYERILN\ga[1].js
%tempinet%\Content.IE5\LOYERILN\ga[1].js
%tempinet%\Content.IE5\LOYERILN\ga[1].js
%tempinet%\Content.IE5\LOYERILN\ga[1].js
%tempinet%\Content.IE5\LOYERILN\ga[1].js
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\admin@accounts[1].txt
%userdir%\Cookies\index.dat
%userdir%\Cookies\index.dat
%userdir%\Cookies\admin@accounts[2].txt
...

Table 6: State 3, Navigate to GMail File Writes

Path
%tempinet%\Content.IE5\S4KTHFSE\Main_Page[1]
%tempinet%\Content.IE5\S4KTHFSE\Main_Page[1].htm
%tempinet%\Content.IE5\S4KTHFSE\Main_Page[1]
%tempinet%\Content.IE5\S4KTHFSE\Main_Page[1]
%tempinet%\Content.IE5\S4KTHFSE\Main_Page[1].htm
%tempinet%\Content.IE5\S4KTHFSE\Main_Page[1].htm
%tempinet%\Content.IE5\S4KTHFSE\Main_Page[1]
%tempinet%\Content.IE5\S4KTHFSE\Main_Page[1].htm
%tempinet%\Content.IE5\S4KTHFSE\main-ltr[2].css
%tempinet%\Content.IE5\S4KTHFSE\shared[2].css
%tempinet%\Content.IE5\S4KTHFSE\readerfeedback[1].css
%tempinet%\Content.IE5\S4KTHFSE\readerfeedback[2].css
%tempinet%\Content.IE5\S4KTHFSE\index[1].php
%tempinet%\Content.IE5\RYAFS53V\index[4].css
%tempinet%\Content.IE5\S4KTHFSE\Main_Page[1]
%tempinet%\Content.IE5\S4KTHFSE\Main_Page[1].htm
%tempinet%\Content.IE5\S4KTHFSE\main-ltr[2].css
%tempinet%\Content.IE5\S4KTHFSE\main-ltr[2].css
%tempinet%\Content.IE5\S4KTHFSE\shared[2].css
%tempinet%\Content.IE5\S4KTHFSE\index[1].php
%tempinet%\Content.IE5\S4KTHFSE\index[1].php
%tempinet%\Content.IE5\RYAFS53V\index[4].css
%tempinet%\Content.IE5\RYAFS53V\index[2].php
%tempinet%\Content.IE5\RYAFS53V\index[1].css
...

Table 7: State 1, Navigate to Wikinews File Writes

Path
%tempinet%\Content.IE5\LOYERILN\Main_Page[1]
%tempinet%\Content.IE5\LOYERILN\Main_Page[1]
%tempinet%\Content.IE5\LOYERILN\Main_Page[1]
%tempinet%\Content.IE5\LOYERILN\Main_Page[1].htm
%tempinet%\Content.IE5\LOYERILN\Main_Page[1]
%tempinet%\Content.IE5\LOYERILN\Main_Page[1]
%tempinet%\Content.IE5\LOYERILN\Main_Page[1].htm
%tempinet%\Content.IE5\LOYERILN\Main_Page[1].htm
%tempinet%\Content.IE5\LOYERILN\Main_Page[1]
%tempinet%\Content.IE5\LOYERILN\Main_Page[1]
%tempinet%\Content.IE5\LOYERILN\Main_Page[1].htm
%tempinet%\Content.IE5\LOYERILN\Main_Page[1]
%tempinet%\Content.IE5\LOYERILN\Main_Page[1]
%tempinet%\Content.IE5\LOYERILN\Main_Page[1].htm
%tempinet%\Content.IE5\XV1BKGQZ\index[1].php
%tempinet%\Content.IE5\XV1BKGQZ\index[3].php
%tempinet%\Content.IE5\S4KTHFSE\index[2].php
%tempinet%\Content.IE5\S4KTHFSE\index[1].php

Table 8: State 2, Navigate to Wikinews File Writes

Path
%tempinet%\Content.IE5\LOYERILN\Main_Page[1].htm
%tempinet%\Content.IE5\LOYERILN\Main_Page[1].htm
%tempinet%\Content.IE5\LOYERILN\Main_Page[1].htm
%tempinet%\Content.IE5\LOYERILN\Main_Page[1].htm
%tempinet%\Content.IE5\LOYERILN\Main_Page[1].htm
%tempinet%\Content.IE5\LOYERILN\Main_Page[1].htm
C:\WINDOWS\system32\lowsec\user.ds
C:\WINDOWS\system32\lowsec\user.ds
C:\WINDOWS\system32\lowsec\user.ds
C:\WINDOWS\system32\lowsec\user.ds
%tempinet%\Content.IE5\RYAFS53V\index[2].php
%tempinet%\Content.IE5\RYAFS53V\index[3].php
%tempinet%\Content.IE5\XV1BKGQZ\index[2].php
%tempinet%\Content.IE5\XV1BKGQZ\index[3].php
%tempinet%\Content.IE5\XV1BKGQZ\index[2].php
%tempinet%\Content.IE5\XV1BKGQZ\index[2].php
%tempinet%\Content.IE5\XV1BKGQZ\index[3].php
%tempinet%\Content.IE5\XV1BKGQZ\index[3].php
%tempinet%\Content.IE5\XV1BKGQZ\index[2].php
%tempinet%\Content.IE5\XV1BKGQZ\index[3].php
%tempinet%\Content.IE5\LOYERILN\124px-Mubarak[1].jpg
%tempinet%\Content.IE5\LOYERILN\124px-Mubarak[1].jpg
%tempinet%\Content.IE5\LOYERILN\124px-Mubarak[1].jpg
%tempinet%\Content.IE5\S4KTHFSE\100px-2011februaryblizzard[1].jpg
%tempinet%\Content.IE5\S4KTHFSE\74px-German_Ulloa[1].jpg
...

Table 9: State 4, Navigate to Wikinews File Writes

Path
%tempinet%\Content.IE5\LOYERILN\mail[3]
%tempinet%\Content.IE5\LOYERILN\mail[3]
%tempinet%\Content.IE5\LOYERILN\mail[4]
%tempinet%\Content.IE5\LOYERILN\mail[3]
%tempinet%\Content.IE5\LOYERILN\mail[3]
%tempinet%\Content.IE5\LOYERILN\mail[4]
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\index.dat
%userdir%\Cookies\index.dat
%tempinet%\Content.IE5\RYAFS53V\mail[3].htm
%tempinet%\Content.IE5\RYAFS53V\mail[3].htm
%tempinet%\Content.IE5\RYAFS53V\mail[3].htm
%tempinet%\Content.IE5\RYAFS53V\mail[3].htm
%tempinet%\Content.IE5\RYAFS53V\mail[3].htm
%tempinet%\Content.IE5\XV1BKGQZ\mail[8]
...

Table 10: State 1, Log into GMail File Writes

Path
C:\WINDOWS\system32\lowsec\user.ds
C:\WINDOWS\system32\lowsec\user.ds
C:\WINDOWS\system32\lowsec\user.ds
C:\WINDOWS\system32\lowsec\user.ds
C:\WINDOWS\system32\lowsec\user.ds
%tempinet%\Content.IE5\LOYERILN\ServiceLoginAuth12d212a1[1]
%tempinet%\Content.IE5\LOYERILN\ServiceLoginAuth12d212a1[1]
%tempinet%\Content.IE5\RYAFS53V\ServiceLoginAuth[1].htm
%tempinet%\Content.IE5\LOYERILN\mail[6]
%tempinet%\Content.IE5\LOYERILN\mail[6]
%tempinet%\Content.IE5\LOYERILN\mail[6]
%tempinet%\Content.IE5\LOYERILN\mail[6]
%tempinet%\Content.IE5\LOYERILN\mail[6]
%tempinet%\Content.IE5\LOYERILN\mail[5].htm
%tempinet%\Content.IE5\LOYERILN\mail[6]
%tempinet%\Content.IE5\LOYERILN\mail[5].htm
%tempinet%\Content.IE5\LOYERILN\mail[5].htm
%tempinet%\Content.IE5\LOYERILN\mail[6]
%tempinet%\Content.IE5\LOYERILN\mail[6]
%tempinet%\Content.IE5\LOYERILN\mail[6]
%tempinet%\Content.IE5\LOYERILN\mail[6]
%tempinet%\Content.IE5\LOYERILN\mail[6]
%tempinet%\Content.IE5\LOYERILN\mail[5].htm
%tempinet%\Content.IE5\LOYERILN\mail[5].htm
...

Table 11: State 2, Log into GMail File Writes

Path
%tempinet%\Content.IE5\XV1BKGQZ\mail[12]
%tempinet%\Content.IE5\XV1BKGQZ\mail[12]
%tempinet%\Content.IE5\XV1BKGQZ\mail[11]
%tempinet%\Content.IE5\XV1BKGQZ\mail[11]
%tempinet%\Content.IE5\XV1BKGQZ\mail[11]
%tempinet%\Content.IE5\XV1BKGQZ\mail[11]
%tempinet%\Content.IE5\XV1BKGQZ\mail[11]
%tempinet%\Content.IE5\XV1BKGQZ\mail[11]
%tempinet%\Content.IE5\XV1BKGQZ\mail[12]
%tempinet%\Content.IE5\XV1BKGQZ\mail[11]
%tempinet%\Content.IE5\XV1BKGQZ\mail[11]
%tempinet%\Content.IE5\XV1BKGQZ\mail[12]
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\admin@mail[1].txt
%userdir%\Cookies\index.dat
%userdir%\Cookies\index.dat
...

Table 12: State 5, Log into GMail File Writes