STATE OF THE ART BOTNET-CENTRIC HONEYNET DESIGN

A Thesis

by

JOHN SYERS III

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2009

Major Subject: Computer Science

STATE OF THE ART BOTNET-CENTRIC HONEYNET DESIGN

A Thesis

by

JOHN SYERS III

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Udo W. Pooch
Committee Members,	William M. Lively
	Alexander Sprintson
Head of Department,	Valerie E. Taylor

May 2009

Major Subject: Computer Science

ABSTRACT

State of the Art Botnet-Centric Honeynet Design. (May 2009)

John Syers III, B.S., University of Houston–Downtown

Chair of Advisory Committee: Dr. Udo W. Pooch

The problem of malware has escalated at a rate that security professionals and researchers have been unable to deal with. Attackers savage the information technology (IT) infrastructure of corporations and governments with impunity. Of particular significance is the rise of botnets within the past ten years. In response, honeypots and honeynets were developed to gain critical intelligence on attackers and ultimately to neutralize their threats. Unfortunately, the malware community has adapted, and strategies used in the early half of the decade have diminished significantly in their effectiveness. This thesis explores the design characteristics necessary to create a honeynet capable of reversing the current trend and defeating botnet countermeasures. This thesis finds that anti-virtual machine detection techniques along with appropriate failsafes are essential to analyze modern botnet binaries.

To Carolina

ACKNOWLEDGMENTS

I would first like to thank my advisor, Dr. Udo Pooch, for his guidance through this process. Thanks also go to the other members of my committee: Drs. William “Mac” Lively and Alex Sprintson. I am also grateful for the assistance of Dr. Guofei Gu for providing his expertise in honeynets, and to Dr. Jyh-Charn “Steve” Liu, for being kind enough to substitute for Dr. Lively during my defense.

I benefited greatly from the assistance of several of my colleagues, especially Jeremy Kelley, David Collins, and Jeff Scappara. Finally, my wife Carolina Syers deserves my deepest thanks for her continued support during this process. Without her this would not have been possible.

TABLE OF CONTENTS

LIST OF TABLES

TABLE Page

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

The problem of security is relatively new, but it has grown at an exponential rate. It has been just over twenty years since the Morris Worm was unleashed on an unsuspecting Internet community. At that time, the worm was unprecedented, and the idea that someone would intentionally unleash such a devastating program was unthinkable. At a certain point malware became commonplace. It is expected, indeed, it is inevitable. Ironically, for the common user, malware has become less disruptive, yet at the same time much more malevolent.

Malicious programs have developed at a rate greater than the security community can keep pace with. What can be done? The answer to this question, like the related field of information assurance, is broad, and could not possibly be answered in a single paper. Instead, my goal is to focus on a specific subset of malware, namely botnets, and analyze current security tools, focusing on honeynets, in order to develop a viable countermeasure.

A. Botnets–The Killer Web App

In 2007 Syngress released a book called "Botnets: The Killer Web App."[56] This title faithfully captures the essence of the rise of botnets. Much like email did in the previous decade, botnets have made a tremendous impact on the World Wide Web. The major difference is, while email is a highly visible application, botnets for all practical intents and purposes are invisible. This next step in the malware evolutionary scale is distinctly different from its predecessors in that botnet writers

The journal model is *IEEE Transactions on Automatic Control.*

take great pains to keep the presence of this malware masked. And for the most part, they have been successful.

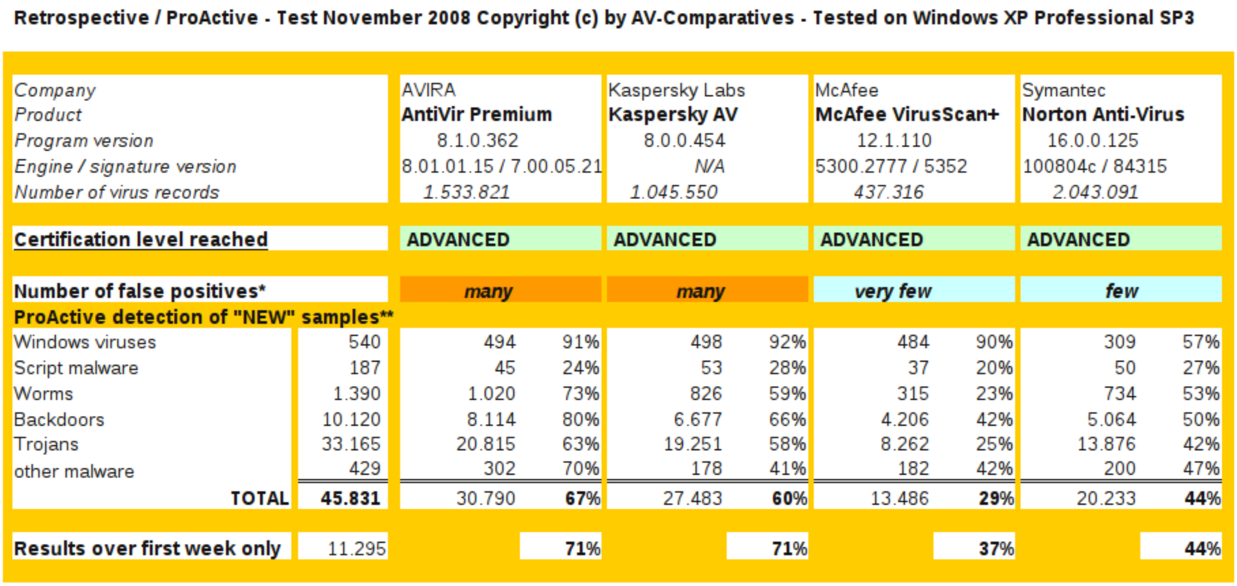
How bad is it? The Federal Bureau of Investigations (FBI) placed cybercrime as their third highest priority, behind terrorism and corporate espionage [29]. They instituted operation BOT ROAST in 2005. By 2007, the FBI reported that it had identified over one million botnet infected computers in the United States [18]. Interestingly enough, botmasters found a way to use the FBI to actually further their goals. Instead of using trite social engineering email subjects like "I love you," a recent malware push involved emails titled, "FBI may strike Facebook." [48] The power of the FBI and the popularity of Facebook made a powerful recruiting tool for the Storm Worm/Botnet.

A 2007 report to Congress on Cybercrime reveals that the U.S. lost $62.7 billion to cybercrime in 2005. Furthermore, it was reported that terrorists have the ability to conduct cyberattacks that could harm the nation's critical infrastructure, shutting down power plants and disrupting air traffic control [70].

The report to congress goes on to cite a particular example of damage caused by a botnet. A California man operated a botnet from 2004 to 2005. During this time the botnet performed more than 2 million infections. Victims of the botnet included hundreds of Department of Defense (DOD) computers in military installations both in the United States and in our installations overseas. Fortunately, this individual was apprehended, but one of the distinct problems of the botnet problem is that the number of botnet operators that are operating unhindered are unknown.

What defenses do we have today against this threat? (Specifically, the threat of having one's systems become part of a botnet. There is also the threat of being attacked by a botnet, but this is a different, and arguably harder problem.) The average consumer has the standard anti-virus programs and internet security suites, but these programs are limited in their protective capability. Table I shows a portion of the November 2008 Retrospective/ProActive anti-virus test from AV-Comparatives [3]. Major anti-virus programs were tested in November against malware samples that were collected by AV-Comparatives three months prior. The program that demonstrated the best performance only detected 71% of the malware, while McAfee VirusScan+, produced by the second largest anti-virus company in the world, detected a dismal 37%.

Table I. November 2008 AV-Comparatives Retrospective/ProActive Test



Retrospective / ProActive - Test November 2008 Copyright (c) by AV-Comparatives - Tested on Windows XP Professional SP3

Company		AVIRA		Kaspersky Labs		McAfee		Symantec	
Product		**AntiVir Premium**		**Kaspersky AV**		**McAfee VirusScan+**		**Norton Anti-Virus**	
Program version		8.1.0.362		8.0.0.454		12.1.110		16.0.0.125	
Engine / signature version		8.01.01.15 / 7.00.05.21		*N/A*		5300.2777 / 5352		100804c / 84315	
Number of virus records		*1.533.821*		*1.045.550*		*437.316*		*2.043.091*	
Certification level reached		**ADVANCED**		**ADVANCED**		**ADVANCED**		**ADVANCED**	
Number of false positives*		***many***		***many***		***very few***		***few***	
ProActive detection of "NEW" samples**									
Windows viruses	540	494	91%	498	92%	484	90%	309	57%
Script malware	187	45	24%	53	28%	37	20%	50	27%
Worms	1.390	1.020	73%	826	59%	315	23%	734	53%
Backdoors	10.120	8.114	80%	6.677	66%	4.206	42%	5.064	50%
Trojans	33.165	20.815	63%	19.251	58%	8.262	25%	13.876	42%
other malware	429	302	70%	178	41%	182	42%	200	47%
TOTAL	**45.831**	30.790	**67%**	27.483	**60%**	13.486	**29%**	20.233	**44%**
Results over first week only	11.295		**71%**		**71%**		**37%**		**44%**

Anti-virus programs are only effective against known botnet infections. A recent article in Information Week illustrated the weaknesses in typical anti-virus programs. In the article a sysadmin discovered an infection on a university computer. The computer was called in and disinfected, but as soon as the machine was placed back on the network, was discovered that the botnet infection remained [30]. The administrator possessed the tools to detect anomalous activity on the network, but not the tools to excise the malware from the infected machine.

Botnets for which no signature has been released can function with impunity. The Storm worm/botnet was discovered more than two years ago, at its apex boasted over one million bots and showed no signs of abating [20]. It was rumored to be capable of disrupting small countries [22]. Fortunately, this botnet was crippled mostly due to a patch from Microsoft and was abandoned in September of 2008, though some claim it was divided and sold [63][39]. What resources does the system administrator have? Intrusion Detection Systems, for one, but the host-based systems are unpopular because they create a significant drain on performance, and network-based systems are subject to limitations similar to those of anti-virus programs.

B. Objective

The primary objective of my research is to develop a design for a honeynet capable of dealing decisively with the current botnet threat. More specifically, my objective is to

- Review the most effective characteristics of the latest botnets
- Determine necessary honeynet requirements based on botnet anti-honeypot techniques
- Create a botnet design architecture based on these requirements

CHAPTER II

BOTNET EVOLUTION

A botnet is one of the newer variants of malware. The botnet shares many characteristics with other forms of malware, often spreading from device to device on the Internet through various sources, including email and the web. What distinguishes a botnet from other forms of malware is that instead of solely executing a pre-defined set of instructions, a bot is also designed to accept instructions remotely from an individual, who is referred to as a botmaster or botherder. As the bot program propagates throughout the internet, the collective group of bots form a botnet, providing the botmaster with an arbitrary number of machines to do his bidding.

A. Botnet History

The botnet finds its origins in Internet Relay Chat (IRC), which was created in 1988 at the University of Finland [51][53][45]. The creation of bots, which is short for robots, followed shortly after. The first known bot was written by Bill Wisner and called Bartender [37]. Its sole function was to serve drinks to channel users. A short time later Greg Lindahl created an IRC game manager that allowed users to play Hunt the Wumpus on a channel [38]. These simple programs paved the way for IRC operators to write programs to perform simple functions, such as keeping channels open. Bots grew in scope to the point where the bots were the de facto operators of IRC; all administrative tasks were run through them.

In 1999, the first malicious IRC bot was discovered: PrettyPark. This was a worm that installed its own IRC client to a system upon infection. Once installed, the IRC client contacted an IRC server, and was capable of initiating file transfers of privileged data, such as passwords and credit card numbers [21].

The next year, a new IRC-related malware program was released, the Global Threat (GT) bot. This bot was based on the newly updated mIRC [1], a still-popular IRC client. mIRC had been endowed with a powerful scripting language that, combined with socket capability, made a target for hackers. As the name implies, the GT Bot included a set of commands geared toward conducting attacks. It was capable of conducting Denial of Service (DoS) attacks using the Internet Control Message Protocol (ICMP) [45], User Datagram Protocol (UDP) [49], as well as Smurf and Shiver attacks [6]. The GT Bot was not self-propagating, however.

In 2002, the SDBot surfaced. This bot combined the attack threat capability of the GT Bot with the self-propagating characteristic of PrettyPark. Unlike the hacked GT Bot, SDBot was an original work, written in C++ [6]. Another of its distinctive characteristics was the fact it was a single, trim 40k executable.

SDBot was followed by Agobot. Because code from SDBot was made widely available, it is believed that Agobot borrowed liberally from SDBot's code [56]. This bot was given a three-part modular design, and was made to be easy both to utilize and to update. The Agobot code base is the code upon which the majority of today's botnets are based [6].

As malware has progressed from a method of demonstrating one's superior hacking skills or showing proof of concept to the means of illicit gain or other more sinister clandestine activities, the botnet has emerged as the tool of choice. Botmasters sell their services to the highest bidder, offering the ability to deliver spam, harvest confidential information, provide web hosting, all for the right price. Malware development is also a lucrative business, with price tags on botnets ranging into thousands of dollars [6].

Even more disturbing is the fact that botnet activities are not limited to simple money making ventures. With the use of the DoS, botnets can be used to attack organizations, and extort them. These attacks have even moved into the political arena. In 2008, the Russian-Georgian conflict was characterized not only by Russian tanks, but also by DoS attacks against Georgian government networks [24].

B. Basic Botnet Characteristics and Capabilities

Classifying botnets is a non-trivial task. A traditional hierarchical taxonomy is insufficient for the task because there are several categories of defining characteristics which overlap and are constantly changing. The initial starting point for the classification comes from [26], which outlines the botnet infection process as a sequence of five transactions that may be observed and subsequently used as a means of detection. Conceptually, however, there are other defining characteristics of botnets that must be included, subsequently what follows bears only a small resemblance to the dialog flow approach of [26].

1. Propagation

Methods of propagation are closely related to the different forms of malware, therefore some formal definitions are necessary. A *virus* is a malware program that, like a biological virus, is dependent upon the 'life' of a host in order to reproduce. Computer viruses do not exist and function independently, rather they attach themselves to other executable programs, and are run when those programs are run. A *worm* is not dependent upon a host program in order to replicate. It may or may not be self-propagating; methods of worm propagation will be discussed shortly. A *trojan horse* or *trojan* is a program that appears to perform a certain benign function, but surreptitiously perform certain malicious activities upon execution. Regrettably, there are certain forms of malware that display characteristics of more than one of these forms, making them hard to classify. These may be referred to as *hybrids*. To further add to the confusion, because viruses were the first and most well known form of malware, the term virus is sometimes used generically to refer to malware. That said, none of the botnets studied used the classical virus method for propagation, nor is this expected to change.

Another issue that deserves mention is the role of social engineering in propagation. The term social engineering itself is merely a buzzword meaning deception, and most forms of malware–botnets included–employ deceiving the user as a means of gaining entrance to their system.

a. Worms

Methods of worm propagation fall into two broad categories. The first are worms that are capable of total self-propagation. This means that the worm requires no direct assistance from the computer user in order to spread. Generally, these worms perform network scans for hosts with exploitable vulnerabilities, and then copy themselves to new hosts and activate themselves through the exploit. Worms started possessing this capability with SDBot in 2002.

The second category of worms require some type of user-intervention to help them spread. A common ploy is to send out fake emails to try to get an unsuspecting user to download and execute a program from a website or open a malicious attachment. Some worms use the web to spread by exploiting browser vulnerabilities. Rather than seeking out vulnerable machines, the attacker lets the vulnerable machines come to him. This approach may be slower, but it really depends on the number and popularity of the websites used for propagation. MPack [40] is an example of a tool used to accomplish this. By using an iframe attack [50], legitimate websites can be used for malware propagation. Because the websites themselves are inadvertently spreading malware, worms propagated in this fashion are being referred to as trojans.

b. Trojans

Trojans only propagate by social engineering. While email can be used to spread trojans, they are principally found on false websites; email (spam) may be used to draw attention to the sites. A common method for spreading Trojans is to place them on game or pornography websites. Sometimes these sites are set up solely for the purpose of spreading malware.

2. Control Structure

The authors of [10] proposed three possible command and control (C&C) models for botnets: centralized, decentralized, and random. During the course of this study, no botnets that use a random control structure were found, and given the drawbacks of such a system, it is unlikely that such a model will be developed in the near future [10][69]. It is acknowledged that control structures and communication protocols are very closely related; however in this work they will be discussed independently of each other.

a. Centralized

Centralized botnets are those that utilize a single host to communicate with botnet clients. The majority of known botnets use a centralized control structure. In April 2008, Arbor Networks posited that 95% of known botnets were centralized [42]. Advantages for this approach include simplicity and maintainability. The major disadvantage is that this creates a single point of failure in the system.

b. Decentralized

The decentralized control structure is in itself a countermeasure. Peer-to-peer (P2P) systems were first utilized when security expert and law enforcement officials began actively targeting botnets and botmasters. In the decentralized model, there is no central point of communication. Bot clients are responsible to send messages to their peers, making the botnet more difficult to disable. The first P2P botnet to be discovered was Sinit, in 2003 [60]. Phatbot, a direct descendant of Agobot, abandoned the former's C&C model for a P2P based one [61].

3. Protocol

The protocol is the standard that defines the syntax and structure of data in communication between two hosts. This is significant to defenders because captured data must be interpreted in order to be understood, and knowing the protocol is a significant part of that.

a. IRC

IRC was the original botnet protocol, and though there are signs that malware writers are exploring other protocols, the majority of botnets currently in operation still use IRC. The IRC protocol is tied to the centralized C&C model, this has been the main motivator for botmasters to seek alternatives. Other disadvantage is the fact that IRC traffic is not common for many networks and thus is easy for administrators to pinpoint and block.

b. Hyper Text Transfer Protocol (HTTP)

The HTTP protocol [19] is also tied to the centralized C&C model, but it has one important advantage over IRC: the majority of internet traffic on networks is HTTP, which makes it much more difficult for administrators to isolate malicious traffic.

c. P2P

While it has been suggested that P2P development is increasing [25], Arbor Networks estimated that at present P2P botnets only comprise 5% of the current botnet population [42]. The reality is that there is insufficient data on botnets currently in the wild to make an accurate determination. What is known is that several P2P botnets have been discovered in the past five years, including Phatbot [61][57], Nugache [44], SpamThru [62] and Peacomm [25].

4. Features

Before bots became modularized, most bots possessed a comprehensive command list encompassing all of the features discussed in this section. Now bots are being designed to be more specialized, a family of bots typically exists to serve a certain function and possesses a commensurate command list. One feature common to all the following functions is the ability to report results back to the botmaster after successful or unsuccessful execution of commands.

a. Distributed Denial of Service (DDoS)

The DDoS attack demonstrates the malicious power of a botnet by crippling an organization's web services by consuming their bandwidth on bogus requests. As of yet there is no solid defense against DDoS attacks. Any botnet whose aggregate bandwidth exceeds that of its target poses a decisive threat. The following attack methods are most common:

- TCP SYN Flood [73]
- HTTP GET Flood [46]
- UDP Flood [35]
- ICMP Flood (Smurf Attack) [73]

b. Spam

One of the major obstacles in curtailing spam is that spamming has become one of the main functions of botnets. Blacklisting has become ineffective because instead of a handful of mail servers there are hundreds of thousands of zombified clients, and the list of bots is always changing. If a botmaster commands a large botnet, he can send out small amounts of spam from each client, thus avoiding creating an anomaly that might be detected.

c. Server

Bots can be used to provide services to the botmaster and his clientele. A bot might be set up as an HTTP server to serve web pages in a phishing scheme. Bots have also been used as FTP servers to store illicit information (such as child pornography).

d. Proxy

Botnets run HTTP and SOCKS (Secured Over Credential-based Kerberos Services [36]) proxies, providing anonymity to botmasters.

e. Extortion

Two methods of botnet extortion exist. In the first, the botnet performs a DDoS attack on an organization, then demands money from the organization in exchange for a cessation of attacks. In the second, the bot client encrypts data on the host machine. A message is then sent to the victim, demanding payment in exchange for the key to unencrypt the data.

f. Data Collection

Any information that is determined to be of value can be retrieved by a bot client. Some botnets utilize simple keylogger programs. These programs generate large amounts of data which must be analyzed, thus botmasters have begun to target specific data. The data itself becomes a commodity to botmasters, often selling off data to third parties. Data collected includes credit card numbers, CD keys, digital certificates, and authentication information for computers, email, Outlook, Paypal, etc. Some data collection is tailored specifically for businesses. A variant of the Prg botnet has been developed targeting the banking sector, allowing attackers to access banking accounts without a username or password [28].

g. Self Preservation

One of the first items in a bot script is to disable anti-virus on the host machine in order to avoid detection. Some systems simply turn off the anti-virus protection. Others subvert it by tampering with the alerting system. It has become common for botnets to patch vulnerabilities in the host system after infection so that the system cannot be exploited by a rival botnet. Botnets also have the ability to receive updates to the botnet software.

h. Self Destruction

If a botmaster suspects a bot has been compromised, he can remotely shut down the bot and erase all data pertaining to the bot.

CHAPTER III

THE HONEYPOT RESPONSE

The number of tools at a security administrator's disposal in his defense against botnets is not large. Honeypots have been recognized as being one of the most useful. In his book *Tracking Hackers*, Lance Spitzner gives a comprehensive definition of what a honeypot is: "A honeypot is a security resource whose value lies in being probed, attacked or compromised." [59]

A. Honeypot History

In 1988, an astronomer named Clifford Stoll published a paper entitled "Stalking the Wily Hacker," outlining his experience of tracing an intruder in the network he managed at Lawrence Berkley Laboratory [64]. The following year, Stoll followed up with a book called *The Cuckoo's Egg* [65]. The method that Stoll used did not include a honeypot; Stoll allowed the hacker to have an almost free run of the lab's production network. His reasons for doing this, namely to be able to track the hacker, captured the interest of the sysadmin community and paved the way for the first honeypots to be designed.

In 1992, *An Evening with Berferd* was released. This whitepaper was much closer to the modern definition of a honeypot. In the whitepaper a system administrator narrates his surreptitious interactions with a hacker. After the attacker falls into one of the traps the administrator set up, the administrator built an environment and crafted responses in real-time in order to study his behavior. There were other goals, the chief of which was to gather enough information about the attacker to be able to ascertain his identity, so that he could be apprehended by the proper authorities. Also prominent was the ability to be able to alert other administrators to the intruder and his practices.[8]

Despite the interest that these documents raised, more than half a decade passed with no further development, at least development that was made known to the public. The first widely available honeypot was the Deception Toolkit (DTK), which was released in 1998 by computer virus expert Dr. Fred Cohen [9]. The toolkit was a series of Perl scripts and code written in C. These scripts were designed to run on a Unix system, and causes the system to appear to have a large number of vulnerabilities. The concept here is that an attacker will be attracted to the system running DTK through routine vulnerability scans, and will subsequently try to break into the system. Because the vulnerabilities presented are manufactured, the attacker will waste time and become frustrated. The toolkit also logs all communications made with the pseudo-vulnerable ports, which allows the user to gain information on the tactics and techniques of the attacker.

Several commercial products followed. CyberCop Sting was the first, produced by Network Associates (now McAfee) in 1998. This product was notable for being the first developed for Windows based system, as opposed to Unix. Around this time Verizon, then known as GTE, developed its own honeypot system called NetFacade, which was capable of simulating a class C network on a single host [71]. Also released that year was BackOfficer Friendly, by NFR Security. It was designed to spoof as a BackOrifice server [15], a known exploit at the time [27]. It ran originally on Windows systems, but was also expanded to Unix systems. In the same vein as DTK, BackOfficer Friendly was also freely available [66].

In 1999, the Honeynet Project was formed[72]. This was designed to be a collaboration between security professionals with the aim of sharing research with the goal of combatting the growing malware problem. Established procedures were published in a series of papers called “Know Your Enemy.” The term Honeynet is used to denote a network of honeypots. The Honeynet project also made its tools freely available. With this collaboration in place, researchers and administrators began deploying honeypots in larger numbers and collecting data. In 2002, a honeypot captured an exploit of the Unix Common Desktop Environment (CDE) Subprocess Control Service (dtspcd), a network daemon that accepts requests from clients to execute commands and launch applications remotely [7, 59]. The vulnerability was known, but it was believed that this weakness was not being exploited; examination of the honeypot showed this to be false. This was the first time that a honeypot was known to discover an unknown exploit.

B. Honeypot Classification

Current publications classify honeypots as either production or research [59]. Production honeypots are those used by organizations as a part of their security plan. Research honeypots are used by universities and other research groups solely to learn more about malware and the malware community. While this is a significant distinction, the differences appear to be in use only, not in actual function. During the course of this research only one honeypot was found that was described specifically as a production honeypot. Incidentally, a 'research' version of this same honeypot was also released, but the only significant differences between the two systems was the amount of support provided and the price [43].

The amount of interaction a honeypot provides to the attacker is the only other viable classifier. Again, previously published information, notably [59], identify three classes of honeypot interaction: low, medium, and high; however, there is not a clear distinction between low and medium. Low-interaction honeypots emulate services and have “limited interaction capabilities;” medium-interaction honeypots also emulate services, but they are said to provide 'more interaction' than low-interaction honeypots. Because there is no clear method to define the difference between low and medium, this paper disregards medium-interaction and refers to all honeypots that emulate services as low.

With this change in place, a low-interaction honeypot is a honeypot which emulates certain services, while a high-interaction honeypot provides a complete operating environment. The latter is accomplished by either using an entire computer, or a virtual machine.

Low-interaction honeypots are usually preconfigured programs that are easy to deploy and monitor. The drawbacks to such systems is that since they are limited in the interaction that they allow an attacker, their ability to collect data is also limited. These honeypots are also more easily detected. Because services are merely emulated, there is less of a chance that an attacker could use the honeypot to attack other systems.

Because a high-interaction honeypot is a complete system, there are no prepackaged setups. Data detection and maintenance methods have to be created by the maintainer, although there are some external tools that can aid with this. Most importantly, the attacker will be able to use the honeypot to attack other systems unless safeguards are put in place. Firewalls are usually erected to keep the honeypot from sending out malicious traffic but this is a telltale sign to an attacker that the machine is a honeypot, or at the least a system of limited value.

CHAPTER IV

ANTI-HONEYPOT TECHNIQUES

Because of the success that defenders have had against the malware community with honeypots, code has been developed to circumvent them. Deception is a key part of a honeypot's effectiveness; it has no value if an attacker knows which systems on a network are honeypots. Most detection methods involve *fingerprinting*, finding a distinct result in which a honeypot differs from its non-honeypot counterpart.

An early example of this is Honeypot Hunter [58], a tool developed by Send Safe in 2002. The program determines whether a discovered mail relay is legitimate or a honeypot by running its own mail server and sending a test message to itself via the mail relay [33]. If the message fails, then the relay is bad; if the message fails but server responds that the message was successfully sent, the relay is likely a honeypot.

A. VM Detection

Virtualization has become the platform of choice for deploying honeypots. Moreover, virtual platforms are also used for malware analysis. For this reason, malware developers have developed routines that check specifically to see if code is being executed in a virtual environment. Because of the efficiency and availability of VMware, it specifically has become a target, although User Mode Linux (UML) has also been used for virtual honeypots.

1. Branding

One of the most obvious ways to determine a virtual machine environment are the virtual devices. A query of hardware information under VMware will show the model of both CDROM and hard drives as VMware, clearly showing virtualization. Compare the two hardware listings:

Listing 1:

```
RealCPU:~# lshw

RealCPU
  description: Portable Computer
  product: Latitude D830
  vendor: Dell Inc.
  serial: B1234E6
...
    *-display:0 UNCLAIMED
      description: VGA compatible controller
      product: Mobile GM965/GL960 Integrated Graphics Controller
      vendor: Intel Corporation
...
    *-cdrom
      description: DVD writer
      product: DVD+-RW TS-L632H
...
    *-disk
      description: ATA Disk
      product: TOSHIBA MK3252GS
      vendor: Toshiba
```

Listing 2:

```
VMCPU:~# lshw

VMCPU
  description: Computer
  product: VMware Virtual Platform
  vendor: VMware, Inc.
  version: None
  serial: VMware-ff 4d b7 9a 2c 7c 9c 62-44 c8 bd f4 e9 f7 4f ff
...
    *-cdrom
      product: VMware Virtual IDE CDROM Drive
...
    *-display
      description: VGA compatible controller
      product: [VMware SVGA II] PCI Display Adapter
      vendor: VMware Inc
...
    *-disk
      description: SCSI Disk
      product: Virtual disk
      vendor: VMware
```

2. System

More differences can be found by looking at low-level functions. One prominent example is that of the Interrupt Descriptor Table (IDT). Virtual machines emulate the table, but it is located in a higher location in memory in a virtual environment. This discrepancy was found by legitimate researchers [31][54], but has also been utilized and improved by the malware community [12].

3. Timing

Another characteristic of virtual machines is that instructions take longer to execute on a virtual machine than a regular system, even though the additional time may be imperceptible to a user. This is because there are additional instructions that take place, either due to emulation because the actual instruction doesn't exist in the virtual environment, or because of the time that it takes to pass the instruction through the virtualization layer. Several successful timing methods have been developed [16], including one which counts individual instructions [55].

B. Sebek

Sebek [67] is a tool produced by the Honeynet Project. It has a client-server architecture. The client is located on the honeypot and functions as a kernel module, detecting syscalls in order to capture keystrokes and other attacker session data. The server, typically the Honeywall gateway, collects and stores the data. Transmissions between client and server are done via UDP and are not easily detectable via packet sniffers. However, the presence of the Sebek kernel module causes the memory locations of the sys_read and sys_write system calls to be much farther apart than they normally would be, constituting a fingerprint [11].

C. Outbound Connection Validation and Initiation

One of the most challenging issues in running a high-interaction honeypot is dealing with outbound connections. Inbound connections are always allowed so that the honeypot can be attacked, but allowing compromised honeypots to become a platform for attacking other machines has ethical and possibly legal repercussions that are best avoided. Defenders expect botnets to establish a connection with C&C servers, so usually a small number of connections are allowed. Still, it is a simple matter for a bot developer to program his bots to attempt to contact the outside world. One hacker admonished his fellows to simply contact 20 prominent websites [11].

Along these same lines, an attacker could create an initiation procedure for the botnet, not allowing a new bot to join the botnet until it completed a certain task, presumably one that required an outbound connection. The attacker could set up multiple authorization servers that each passed on part of a key to the bot client, which would be required by the C&C server for authentication. This method would require the botmaster to maintain multiple servers, possibly employing redundancy, or risk having the botnet initiation process fail and limit the size of the botnet as a result.

Attackers are also aware of the use of Snort_inline on outbound connections to render malicious commands benign[11]. A presumable response could be to initiate a bot by requiring it to infect another host. The secondary host would then report the infection to the C&C server which would pass on an authorization key, which would then be passed back to the original infected bot. A very similar process is outlined in [74].

CHAPTER V

REQUIREMENTS AND ARCHITECTURE

Many of the building blocks necessary to deal with these botnet threats already exist. Most of them are part of a current honeypot solution, some concepts are taken from other security systems. At present, none of them exist altogether in one framework.

A. Requirements

1. Modular and Upgradable

Taking a page from the botnet developer's handbook, a honeynet should be as modular and easy to upgrade as Agobot and all its children. In a manner of speaking, many botnet components are modular. The Honeynet Project has developed a large number of tools that work together. Because these tools are open-source, they are an excellent foundation from which to start to build a customized honeypot solution. This compatibility must be maintained.

2. Utilizes Virtualization

Despite the challenges posed by virtual machine detection, the benefits of virtualization are too great to be ignored. Virtualization allows multiple honeypots to be run on the same system; a server with a quad-core 2.5 MHz processor and 8 GB of memory could run up to twenty honeypots. These systems can all be powered on and off, erased and reimaged by remote control. Another valid question is whether the trend towards honeypots self-destructing in virtual machine environments will continue, as there are non-honeypot virtual machine servers that are high-value targets for botmasters. Commercial visualization is not marketed primarily towards researchers, but businesses who are looking to make their IT expenditures more efficient. It would seem that it would be counterproductive for a botmaster to ignore these potential resources.

3. Accept Input From External Networks

The botnet needs to be modular/flexible enough to deal with inbound traffic from multiple networks. This means that the honeynet's address space needs to be independent of any external addresses. It should be capable of receiving traffic from individual Internet Protocol (IP) addresses scattered throughout subnet or a block of address space. It should also accommodate addresses from different networks, such as traffic redirected from a virtual private network (VPN) [23], generic routing encapsulation (GRE) [17], or internet telescopes [5][41][47].

4. Does Not Allow External Attacks

Despite the fact that this requirement is present in all viable honeypots, it is nontrivial. A honeypot that allows too few connections will be ineffective; one that allows too many could inadvertently cause those seeking to catch lawbreakers to become lawbreakers.

5. Automated Analysis and Infiltration

The system should be able to perform simple analysis without human intervention. It should also be able to extract a network fingerprint and other artifacts from the botnet binary, allowing a clean client to join the botnet and gain intelligence without the danger that the botnet client itself would pose, namely infecting other systems.

6. Allows Secondary Infections

Honeypots should be allowed to infect each other. This allows any initiation requirement to be fulfilled.

7. Masks Branding

References to virtual hardware should be concealed or changed.

8. Only Capture Unique Malware

Agreeing with the reasoning of [52], the stated goal of malware collection is to collect as many bot binaries as possible. However, receiving the same binaries over and over again is not desired. Unfortunately, this is a highly probable occurrence during the outbreak of a new worm–victims are assimilated at a high rate, and each new victim is performing its own scans for vulnerable hosts. Therefore, an amendment is in order: the stated goal of malware collection is to collect as many *unique* bot binaries as possible.

9. Virtualization Failsafe

Because virtual machine detection is a big part of anti-honeypot techniques, some botnets might avoid collection despite best efforts. The honeynet should be able to compensate for this.

10. Automated Centralized Malware and Data Collection

Even in a moderately sized honeynet, there will be much data to be collected. Network statistics and malware are arguably most important. The honeynet should have the ability to capture data from multiple sources and store it in an organized fashion.

B. Architecture

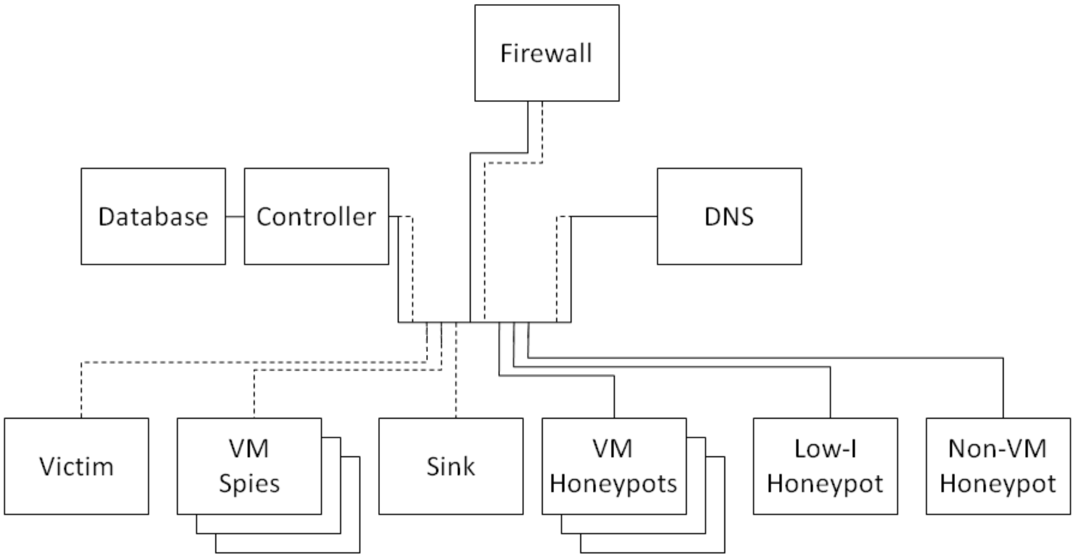


Fig. 1. Honeynet Architecture

The architecture for the honeynet is shown in Figure 1. It consists of three main sections: collection, testing, and control.

1. Collection

The collection section is the heart of the honeynet. It contains three different types of honeypots: VM high-interaction, low-interaction, and non-VM high-interaction. The variance in platform provides a way to capture binaries that have virtual machine detection that has not yet been defeated. Because low-interaction honeypots require less maintenance, they provide a collection platform that can be run continuously.

Nepenthes [4] is chosen for the low-interaction honeypot because of its ease of use and its compatibility with other Honeynet Project tools. Because Microsoft Windows is more frequently targeted by bots, Windows XP with service pack 2 is the operating system of choice for both the non-virtual honeypot and the majority of the virtual honeypots. The remainder of the operating systems consist of an even split of Linux Debian based and RPM based systems.

The virtual honeypots utilize an existing patch that changes the branding in VMware to more common device and driver names [32]. Assuming that botnet code would be looking for specific values (”VMware”), making simple changes should be sufficient to thwart bot tests, where a flesh-and-blood attacker might not be so easily fooled.

2. Testing

The testing environment is isolated on a separate virtual local area network (VLAN) to avoid secondary infections from honeypots. A virtual machine with multiple guests, designated as spies, is the principal system for testing. A fresh operating system image can be loaded into the spy, along with a copy of the botnet binary. Two other systems exist in the testing VLAN, one a victim that exists solely to be infected, and the other plays the role of the command and control server. The server runs both an IRC and a HTTP server in order to interact with the bot client. One assumption here is that the botnet utilizes either the IRC or HTTP protocol. As the bot begins to make contacts, traffic is redirected to either the server or the victim based on protocol.

If no meaningful communication is established, the controller will erase the test machine and start over, remembering the previous traffic redirections and reversing them. The goal here is two fold, to learn more about the infection process, including any initiation procedures, and to learn how the bot interacts with the C&C server. This is similar to the approach used in [52]; the difference is that their test environment does not include a victim host. The server will attempt to communicate with the bot based on information stored in the database. If the bot self-destructs, the controller will reset the test machine and the server will try different responses until its database is exhausted.

3. Control

The two prominent parts of the control section are the controller and the firewall. The controller is responsible for managing the honeynet as a whole. The firewall controls the flow of data both in and out of the honeynet.

The controller is responsible for data collection, one of the most important parts of the honeynet. Because there are three different types of honeypots, multiple collection methods are needed to automatically extract bot binaries from each different type of system. Sebek is used for collection purposes on both virtualized and non-virtualized honeypots. Because of Sebek detection however, one virtual honeypot utilizes a different collection scheme.

In this case, the virtual machine is shut down at a specified interval. The virtual machine image is then compared to a clean image, utilizing a whitelist to capture common system log changes. The remaining files are collected for testing. With the low interaction honeypots, a simple script captures malware at scheduled intervals. Finally, data is also collected from the firewall. All data is routed to the database, running MySQL.

The controller is also responsible for the general maintenance of all systems. Both virtual and non-virtual machines are rebooted at timed intervals and are reimaged with clean copies of their operating systems. With non-virtual systems this is accomplished with the use of an administrative operating system on a separate partition.

The firewall performs several tasks, most of which deal with filtering and blocking traffic. The firewall also redirects all domain name server (DNS) traffic to the internal

DNS. It is the firewall that produces the efficiency of the honeynet.

In [52], the uniqueness question was dealt with by disabling the download capacity of the low-interaction honeypot Nepenthes, and using an independent download station. The experiment also included virtual high-interaction honeypots, although Nepenthes was the primary malware collection platform. Unfortunately, the issues of reoccurring downloads in the virtual honeypots was never addressed.

The honeyfarm GQ [13] deals with this problem by utilizing two different types of filtering. The first filter blocks packets that can positively be identified as having been seen before by the first packet. The second involved using a replay proxy [14] to continue communication with inbound traffic where more packets are needed to determine whether the traffic is new. Unique traffic is then played back to a honeypot and the session is transferred.

While GQ's solution is novel, it is much more than what is required for a honeynet designed to handle a class C network. However, a simplification of the idea would be productive, monitoring incoming traffic to a honeypot and resetting the connection if it can be determined that the attack has already been stored. While the idea of dropping traffic is hardly novel, traditionally in honeynets the inbound traffic is unhindered to allow the greatest amount of infection, while constraints are placed on outbound packets.

The firewall also applies constraints to outbound traffic, but the principal means of contraint is not connection counting like Honeywall. Snort_inline is the principal means of constraining traffic, converting known malicious traffic to non-malicious. The firewall does utilize its own form of connection counting: if the firewall counts the number of outbound connections per minute exceeding a certain threshold, the number of connections will be throttled.

CHAPTER VI

PROPOSED IMPLEMENTATION

The proposed implementation of the work takes place in four phases. The phased implementation is designed to allow incremental testing. In the first phase, the honeynets are deployed with a preconfigured controller and firewall. The customized controller is developed in the second phase. Phase three adds the the testing platform. In phase four, the system is modified to accept noncontiguous and external network connections. A proof of concept was completed that contains a proof of concept that contains partial elements of the first and second phases. See Appendix A.

CHAPTER VII

FUTURE WORK

At the time of writing, phase one has been implemented but not tested. The next step is to test phase one and continue through with development and test through the proposed implementation. There are also two additional tools that currently have not been evaluated but their descriptions look promising. The first is CWSandbox. This is a commercial tool that creates a simulated malware analysis environment. The second is botsnoopd, a daemon that spies on botnets. The tool utilizes IRC, HTTP, and P2P via WASTE. Note that this tool is not publicly available, but can be obtained by those who have sufficient credentials in the security community.

CHAPTER VIII

CONCLUSION

In order for a honeynet to be able to retain efficiency and deal with modern anti-honeypot techniques, it must utilize virtualization. Therefore, it must be able to defeat virtual machine detection. In order to deal with all contingencies, it should have a non-virtual honeypot within the network. Sebek is not a must for collection management, but if it is used, steps must be taken to ensure that it is not detected. Outbound connections must be flexible enough to allow connection to C&C servers and fulfill any simple connection verification tests while still disallowing external attacks.

An efficient-botnet centric honeynet solution is certainly attainable. Unfortunately, attackers and defenders are caught up in a continually escalating arms race, which means that any viable solution is only effective for a short period before means are found to overcome it. One of the works referenced in this paper is a fake issue of Phrack magazine, which contained not only hacking, but some pontificating [11], claiming that the Honeynet Project was built upon flawed premises:

1. HoneyPot Technology may be openly shared and remain effective.

2. HoneyPot Technology may be deployed in a hostile environment, and remain undetected.

3. Even if detected, Attackers will not target the honeypot or its operators for further exploitation.

While it is näive to think that security professionals are unaware of these issues or are unprepared to deal with them, the first one specifically seems to be a significant disadvantage. In many works, including this one, there is much speculation about what the malware community is doing because the malware community does not widely disseminate its doings. And then, because of the lack of information, research create their own scenarios about what hackers may be doing, and in the process, provide them with additional resources. This is a non-trivial problem, but it reminds me of why the government has different security clearances and some government information is classified. Sometimes the public does not need to know.

REFERENCES

[1] mIRC: Internet relay chat client. [Online]. Available: http://www.mirc.com/. Accessed on 20 March 2009.

[2] Offensive computing. [Online]. Available: http://www.offensivecomputing.net/. Accessed on 19 April 2009.

[3] AV-Comparatives. Retrospective / proactive - test November 2008. [Online]. Available: http://www.av-comparatives.org/seiten/ergebnisse_2008_11.php.

[4] P. Baecher, M. Koetter, M. Dornseif, and F. Freiling, “The nepenthes platform: An efficient approach to collect malware,” in *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID).* Springer, 2006, pp. 165–184.

[5] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson, “The internet motion sensor: A distributed blackhole monitoring system,” in *Proceedings of Network and Distributed System Security Symposium (NDSS 05)*, 2005, pp. 167–179, [Online]. Available: http://www.isoc.org/isoc/conferences/ndss/05/proceedings/papers/ims-ndss05.pdf.

[6] J. Canavan, “The evolution of malicious IRC bots,” in *Proceedings of Virus Bulletin Conference 2005*, October 2005, pp. 104–114, [Online]. Available: http://www.symantec.com/avcenter/reference/the.evolution.of.malicious.irc.bots.pdf.

[7] CERT/CC. (2001, November) Cert advisory CA-2001-31 buffer overflow in CDE subprocess control service. CERT. [Online]. Available: http://www.cert.org/advisories/CA-2001-31.html.

[8] W. Cheswick, *An Evening with Berferd.* New York: USA: ACM Press/Addison-Wesley Publishing Co., 1998, ch. 7, pp. 103–116, [Online]. Available: http://www.cheswick.com/ches/papers/berferd.pdf.

[9] F. Cohen. Deception toolkit. [Online]. Available: http://all.net/dtk/index.html. Accessed on February 2009.

[10] E. Cooke, F. Jahanian, and D. McPherson, “The Zombie roundup: Understanding, detecting, and disrupting botnets,” in *SRUTI’05: Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop.* Berkeley, CA, USA: USENIX Association, 2005, pp. 6–6, [Online]. Available: http://www.eecs.umich.edu/~emcooke/pubs/botnets-sruti05.pdf.

[11] J. Corey, “Local honeypot identification,” *Phrack*, September 2003, [Online]. Available: http://www.phrack.nl/site/phrack/phrack62/p62-0x07.txt.

[12] Cthulhu. (2009, January) Malware refuses to run properly on VMware. [Online]. Available: http://www.woodmann.com/forum/showthread.php?t=12359.

[13] W. Cui, V. Paxson, and N. C. Weaver. (2006, September) Gq: Realizing a system to catch worms in a quarter million places. Berkeley, CA. [Online]. Available: http://research.microsoft.com/en-us/um/people/wdcui/papers/gq-techreport.pdf.

[14] W. Cui, V. Paxson, N. C. Weaver, and R. H. Katz, “Protocol-independent adaptive replay of application dialog,” in *The 13th Annual Network and Distributed System Security Symposium (NDSS)*, 2006, [Online]. Available: http://research.microsoft.com/en-us/um/people/wdcui/papers/roleplayer-ndss06.pdf.

[15] Cult of the Dead Cow. Back orifice 2000. [Online]. Available: http://bo2k.sourceforge.net/.

[16] G. Delalleau, "Mesure locale des temps d'execution: Application au controle d'integrite et au fingerprinting," in *Proceedings of SSTIC 2004*, 2004, [Online]. Available: http://actes.sstic.org/SSTIC04/Fingerprinting_integrite_par_timing/SSTIC04-article-Delalleau-Fingerprinting_integrite_par_timing.pdf.

[17] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. (2000, March) Generic routing encapsulation (gre). Internet Engineering Task Force. [Online]. Available: http://tools.ietf.org/html/rfc2784.

[18] Federal Bureau of Investigation. (2007, June) Over 1 million potential victims of botnet cyber crime. Federal Bureau of Investigation. [Online]. Available: http://www.fbi.gov/pressrel/pressrel07/botnet061307.htm.

[19] R. Fielding, J. Gettys, J. Mogol, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. (1999, June) Hypertext transfer protocol – HTTP/1.1. Internet Engineering Task Force. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt.

[20] D. Fisher, "Experts predict storm trojan's reign to continue," *SearchSecurity.com*, 2007, [Online]. Available: http://searchsecurity.techtarget.com/news/article/0,289142,sid14_gci1278241,00.html.

[21] D. Fowler, "Netnews," *netWorker*, vol. 3, no. 3, pp. 7–12, 1999.

[22] S. Gaudin, "Storm worm botnet attacks anti-spam firms," *InformationWeek*, September 2007, [Online]. Available: http://www.informationweek.com/news/internet/showArticle.jhtml?articleID=201807222.

[23] B. Gleeson, A. Lin, J. Heinanen, T. Finland, G. Armitage, and A. Malis. (2000, February) A framework for ip based virtual private networks. Internet Engineering Task Force. [Online]. Available: http://www.ietf.org/rfc/rfc2764.txt.

[24] J. Goodwin, "Russian "hacktivists" used Turkish botnets to attack Georgia," *Government Security News*, vol. 6, no. 10, p. 9, Oct 2008, [Online]. Available: http://www.gsnmagazine.com/cms/features/news-analysis/1042.html.

[25] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon, "Peer-to-peer botnets: Overview and case study," in *HotBots'07: Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets.* Berkeley, CA, USA: USENIX Association, 2007, pp. 1–1, [Online]. Available: http://www.usenix.org/events/hotbots07/tech/full_papers/grizzard/grizzard_html/.

[26] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "Bothunter: Detecting malware infection through IDS-driven dialog correlation," in *Proceedings of the 16th USENIX Security Symposium (Security'07)*, Boston, MA, August 2007, pp. 167–182, [Online]. Available: http://www.usenix.org/events/sec07/tech/gu.html.

[27] Internet Security Systems. ISS X-Force white paper: Back orifice 2000 backdoor program. [Online]. Available: http://www.isskk.co.jp/customer_care/resource_center/whitepapers/bo2k.pdf. Accessed on February 2009.

[28] D. Jackson. (2007, December) Hackers use stealthy, new prg banking trojan to attack commercial banking clients in four countries. [Online]. Available: http://www.secureworks.com/research/threats/bankingprg/.

[29] K. C. Jones, “Cybercrime high on FBI priority list; help wanted,” *InformationWeek*, October 2006, [Online]. Available: http://www.informationweek.com/news/security/government/showArticle.jhtml?articleID=193402056.

[30] D. Kaplan, “Busting bots: Defending against botnets,” *SC Magazine*, February 2009, [Online]. Available: http://www.scmagazineus.com/Busting-bots-Defending-against-botnets/article/126919/.

[31] T. Klein. (2008) Scoopyng - the VMware detection tool. [Online]. Available: http://www.trapkit.de/research/vmm/scoopyng/index.html.

[32] K. Kortchinsky. (2004, January) Counter measures to VMware fingerprinting. [Online]. Available: http://seclists.org/honeypots/2004/q1/0015.html.

[33] N. Krawetz, “Anti-honeypot technology,” *IEEE Security and Privacy*, vol. 2, no. 1, pp. 76–79, 2004.

[34] C. Kreibich and J. Crowcroft, “Honeycomb: Creating intrusion detection signatures using honeypots,” *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, pp. 51–56, 2004.

[35] F. Lau, S. Rubin, M. Smith, and L. Trajkovic, “Distributed denial of service attacks,” vol. 3, 2000, pp. 2275–2280.

[36] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. (1996, March) Socks protocol version 5. Internet Engineering Task Force. [Online]. Available: http://tools.ietf.org/html/rfc1928.

[37] G. Lindahl. (1991, July) Re: Interesting idea! alt.irc. [Online]. Available: http://groups.google.com/group/alt.irc/msg/753c065b9ee6ab62. USENET.

[38] ——. (1992, January) Re: Welcome to alt.irc.recovery! alt.irc.recovery. [Online]. Available: http://groups.google.com/group/alt.irc.recovery/msg/3235a21c369c69f4. USENET.

[39] Marshal8e6. (2008, October) Storm botnet fades away to nothing. [Online]. Available: http://www.marshal8e6.com/newsitem.asp?article=788.

[40] V. Martínez. (2007, May) Pandalabs report: Mpack uncovered. [Online]. Available: http://blogs.pandasoftware.com/blogs/images/PandaLabs/2007/05/11/MPack.pdf.

[41] D. Moore, C. Shannon, G. M. Voelker, and S. Savage. (2004) Network telescopes: Technical report. [Online]. Available: http://www.caida.org/publications/papers/2004/tr-2004-04/tr-2004-04.pdf.

[42] J. Nazario. (2008, April) Bot and botnet taxonomy. Computer Security Institute. Computer Security Institute Security Exchange 2008 (CSI SX 08). [Online]. Available: http://monkey.org/~jose/presentations/csisx2008.d/CSI%20SX%202008%20Nazario%20Botnet%20Taxonomy.pdf.

[43] Network Security Software. (2008) Specter intrusion detection system. [Online]. Available: http://www.specter.com/default50.htm.

[44] S.-K. Noh, J.-H. Oh, J.-S. Lee, B.-N. Noh, and H.-C. Jeong, “Detecting P2P botnets using a multi-phased flow model,” *Digital Society, 2009. ICDS ’09. Third International Conference on*, pp. 247–253, Feb. 2009.

[45] J. Oikarinen and D. Reed. (1993, May) Internet relay chat protocol. Internet Engineering Task Force. [Online]. Available: http://tools.ietf.org/html/rfc1459.

[46] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the dos and ddos problems," *ACM Comput. Surv.*, vol. 39, no. 1, p. 3, 2007.

[47] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the internet," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 59–64, 2003.

[48] S. M. Poremba, "Storm worm leverages FBI and facebook in new attack," *SC Magazine*, July 2008, [Online]. Available: http://www.scmagazineus.com/Storm-Worm-leverages-FBI-and-Facebook-in-new-attack/article/113071/.

[49] J. Postel. (1980, August) User datagram protocol. Internet Engineering Task Force. [Online]. Available: http://tools.ietf.org/html/rfc768.

[50] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose, "All your iFRAMEs point to us," in *SS'08: Proceedings of the 17th Conference on Security Symposium.* Berkeley, CA, USA: USENIX Association, 2008, pp. 1–15, [Online]. Available: http://research.google.com/archive/provos-2008a.pdf.

[51] R. Puri. (2003, August) Bots & botnet: An overview. SANS Institute. [Online]. Available: http://www.sans.org/reading_room/whitepapers/malicious/bots_and_botnet_an_overview_1299?show=1299.php\&cat=malicious.

[52] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in *IMC '06: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement.* New York, NY, USA: ACM, 2006, pp. 41–52, [Online]. Available: http://www.imconf.net/imc-2006/papers/p4-rajab.pdf.

[53] D. Reed. (1992, May) A discussion on computer network conferencing. Internet Engineering Task Force. [Online]. Available: http://tools.ietf.org/html/rfc1324.

[54] J. Rutkowska. (2004, November) Red Pill... or how to detect VMM using (almost) one CPU instruction. [Online]. Available: http://www.invisiblethings.org/papers/redpill.html.

[55] J. K. Rutkowski, "Execution path analysis: Finding kernel based rootkits," *Phrack*, July 2002, [Online]. Available: http://www.textfiles.com/magazines/PHRACK/PHRACK59.

[56] C. A. Schiller, J. Binkley, D. Harley, G. Evron, T. Bradley, C. Willems, and M. Cross, *Botnets: The Killer Web App.* Rockland, MA: Syngress, January 2007.

[57] R. Schoof and R. Koning. (2007, February) Detecting peer-to-peer botnets. System and Network Engineering master student projects 2006 - 2007. [Online]. Available: http://staff.science.uva.nl/~delaat/sne-2006-2007/index.html.

[58] Send-Safe. (2006, September) Send-safe honeypot hunter. [Online]. Available: http://www.send-safe.com/honeypot-hunter.html.

[59] L. Spitzner, *Honeypots: Tracking Hackers.* Boston, MA: Addison-Wesley, 2003, [Online]. Available: http://www.tracking-hackers.com/book/.

[60] J. Stewart. (2003, Dec.) Sinit P2P trojan analysis. SecureWorks. [Online]. Available: http://www.secureworks.com/research/threats/sinit/.

[61] ——. (2004, March) Phatbot trojan analysis. SecureWorks. [Online]. Available: http://www.secureworks.com/research/threats/phatbot/.

[62] ——. (2006, October) Spamthru trojan analysis. SecureWorks. [Online]. Available: http://www.secureworks.com/research/threats/spamthru/.

[63] ——. (2009, January) Spam botnets to watch in 2009. SecureWorks. [Online]. Available: http://www.secureworks.com/research/threats/botnets2009/.

[64] C. Stoll, "Stalking the wily hacker," *Communications of the ACM*, vol. 31, no. 5, pp. 484–497, 1988.

[65] ——, *The Cuckoo's Egg: Tracking a Spy through the Maze of Computer Espionage*. New York: Doubleday, 1989.

[66] W. Summers. (2007, Oct) A review of 5 honeypots. CSU Center for Information Assurance Education. [Online]. Available: http://csc.colstate.edu/CAE-IA/Honeypot_Report/Honeypots/honey/Honey-7-1.html.

[67] The Honeynet Project. (2003, November) Know your enemy: Sebek. [Online]. Available: http://old.honeynet.org/papers/sebek.pdf.

[68] ——. (2005, August) Know your enemy: Honeywall cdrom roo. [Online]. Available: http://old.honeynet.org/papers/cdrom/roo/index.html.

[69] Trend Micro. (2006, November) Taxonomy of botnet threats. [Online]. Available: http://us.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibrary/botnettaxonomywhitepapernovember2006.pdf.

[70] United States Government Accountability Office. (2007, June) Cybercrime: Public and private entities face challenges in addressing cyber threats. [Online]. Available: http://www.gao.gov/new.items/d07705.pdf.

[71] Verizon. Netfacade intrusion detection. [Online]. Available: http://www22.verizon.com/fns/solutions/netsec/netsec_netfacade.html.

[72] D. Watson and J. Riden, "The honeynet project: Data collection tools, infrastructure, archives and analysis," *Information Security Threats Data Collection and Sharing, 2008. WISTDCS '08. WOMBAT Workshop on*, pp. 24–30, April 2008.

[73] D. Yuan and J. Zhong, "A lab implementation of SYN flood attack and defense," in *SIGITE '08: Proceedings of the 9th ACM SIGITE Conference on Information Technology Education*. New York, NY, USA: ACM, 2008, pp. 57–58.

[74] C. C. Zou and R. Cunningham, "Honeypot-aware advanced botnet construction and maintenance," in *DSN '06: Proceedings of the International Conference on Dependable Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 199–208.

APPENDIX A

PROOF OF CONCEPT

The experiment was executed in order to validate the viability of the design. There are two main contributions to honeynets in this design. The first is in terms of efficiency, and the second in productivity. Specifically, the experiment was designed to show that honeynets can be more efficient by being selective in allowing incoming traffic. Allowing all incoming traffic produces redundancy and is therefore inefficient. It is also designed to show that virtualization failsafes are a necessary part of honeynets today.

The test consisted on creating a closed network in which a 'victim' machne was subjected to various forms of malware as shown in Figure 2. The victim machine was connected to two separate honeynets, one using 'standard' configuration, and one with enhancements based on the proposed design. Data was collected from the honeynet, upon which conclusions were based.

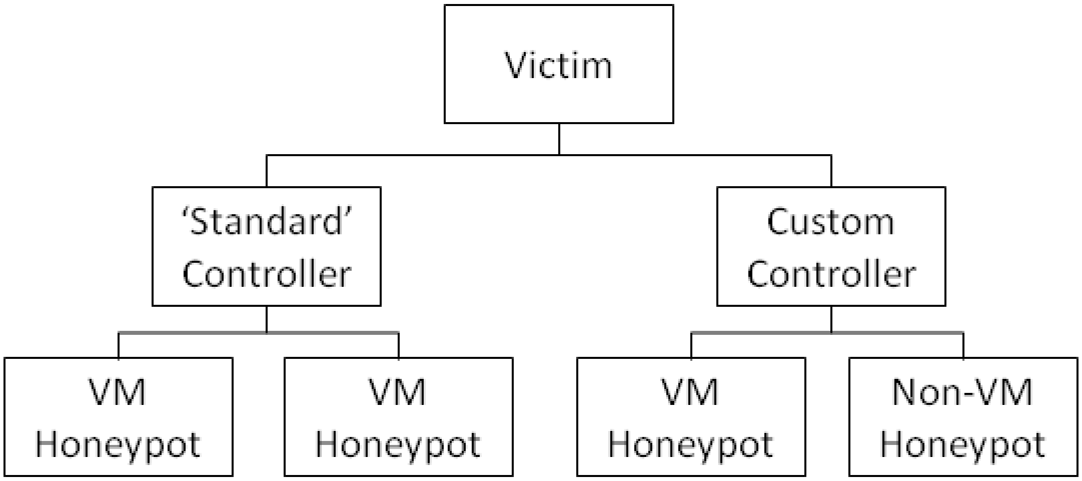


Fig. 2. Test Network Layout

The latest version of Honeywall [68] from the Honeynet Project was used as the 'standard' honeynet controller. Honeywall utilizes Snort_inline with a standard set of rules designed to restrict outbound traffic. It also includes a Sebek server for collecting malware into a MySQL database. By contrast, the customized controller also uses Snort_inline, but emphasis was placed on inbound traffic. For simplicity's sake, outbound traffic filters were omitted.

The custom controller was designed to automatically generate rules based on incoming network traffic, so that a honeynet would only be infected with the same malware once. The initial plan was to use Honeycomb [34], a tool specially designed to create Snort and Bro signatures. Unfortunately, Honeycomb requires exposure to network traffic in order to train itself, which posed a problem. This issue was solved by the creation of a far simpler signature generation tool, *makesig*. The tool uses a libpcap trace to generate signatures for self-propagating malware that copies itself to the host using FTP or TFTP. Packet data is extracted based on port data along with packet sizes, which was based on previous observations. Using a timed interval, makesig checks the trace file for packets that meet the pattern, and if the pattern is matched, new rules are created, the trace file is flushed, and Snort_inline is restarted (See Figure 3).

The standard honeynet uses two virtual high-interaction honeypots. The custom honeynet has both a virtual and a non-virtual high-interaction honeypot. VMWare ESI Server 3i 3.5.0 was the virtualization platform used. All honeypots were running Microsoft Windows SP 2. The malware was obtained from Offensive Computing [2].

Each honeynet was run on a separate IBM eServer xSeries 335, each with two Intel Xeon 2.4 GHz processor, 2 GB of memory and two 73.4 GB hard drives. The victim machine and the non-virtual honeypot were both Dell Optiplexes GX240 with a Intel Pentium 4 1.6 GHz processor, 256 MB of memory and 20 GB of storage space.

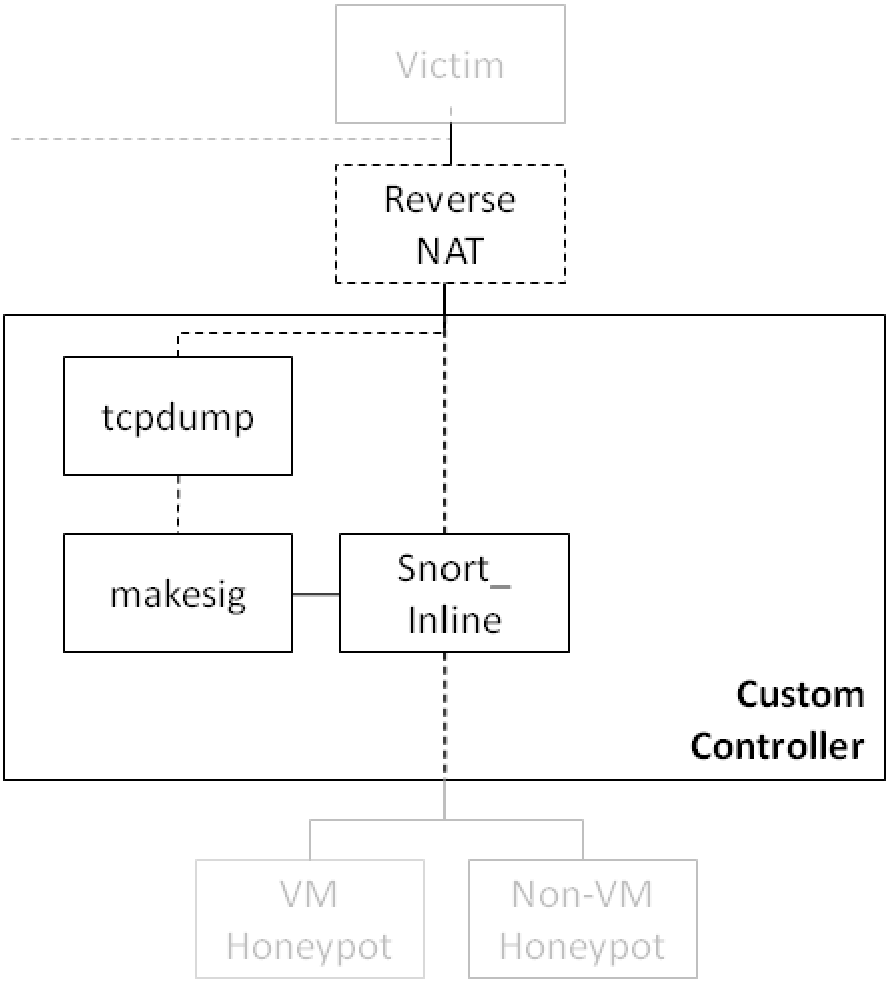


Fig. 3. Expanded Controller View

A Dell Powerconnect 2708 switch was used to connect all machines, utilizing multiple VLANs to provide traffic separation.

During the course of the experiment it was observed that it took a long time for self-propagating malware to actually infect the honeypots because they scanned randomly for IP addresses within the Class A and Class B address space, rather than starting with their local subnet. To speed up the process, a reverse NAT module was added to each honeynet. The module was a short program written in C++ that ingored initial packets based on a specified count, then captured two packets and mapped them to the two internal honeypots, and forwarded them through the controller. All subsequent packets were dropped. Resetting the reverse NAT module also allowed the honeypots to be allowed to be infected multiple times. Each NAT module was reset once, allowing the machine to be potentially infected with each virus twice.

Twelve different malware samples were tested. Of the twelve, one signature was generated, successfully preventing that honeynet from being infected with that malware again. Surprisingly, four of the samples would not run in a virtual environment. In addition, there was no observed behavior at all on three of the samples in the test. These are marked as N/A in Table II.

Table II. Number of Honeynet Infections

	Standard		Custom	
	Honeypot 1	Honeypot 2	Honeypot 1	Honeypot 2
1	2	2	1	1
2	N/A	N/A	N/A	N/A
3	N/A	N/A	N/A	N/A
4	0	0	0	0
5	0	0	0	2
6	N/A	N/A	N/A	N/A
7	2	2	2	2
8	0	0	0	2
9	0	0	0	0
10	2	2	2	2
11	0	0	0	2
12	0	0	0	2

The results show a significant difference between the standard honeypot and the custom one. Ideally, a honeynet should capture each malware sample one time. Multiple collections is inefficient, and a failure to capture is unproductive. Based on the results, efficiency and productivity are calculated using the equations below:

$$\text{efficiency} = \frac{\text{number of unique collections}}{\text{number of total collections}} \tag{A.1}$$

$$\text{productivity} = \frac{\text{number of unique infections}}{\text{number of unique collections}} \tag{A.2}$$

Table III. Honeypot Productivity and Efficiency

Honeynet	Productivity	Efficiency
Standard	25%	33%
Custom	39%	78%

VITA

John Syers III received a Bachelor of Science in Computer Science from the University of Houston-Downtown in May 1999. He received a Master of Science degree in Computer Science from Texas A&M University in May 2009. His interests include computer security, information assurance and operating systems. He can be contacted at: Department of Computer Science, Texas A&M University, College Station, TX 77843-3112; or jsyers@acm.org.

The typist for this thesis was John Syers III.