Hardening Honeynets against Honeypot-Aware Botnet Attacks

Charles Costarella[1], Sam Chung[1,2], Barbara Endicott-Popovsky[2], David Dittrich[3]
[1]Institute of Technology
University of Washington, Tacoma, WA, USA
{costarec, chungsa}@uw.edu
[2]Center for Information Assurance & Cybersecurity
University of Washington, Seattle, WA, USA
endicott@uw.edu
[3]Applied Physics Laboratory
University of Washington, Seattle, WA, USA
dittrich@u.washington.edu

Abstract

Honeynets are a valuable security tool for gathering data on attackers. The amount of data gathered, and therefore the perceived value of the tool is directly related to the amount of time the attacker spends on the network. The allure quality of a honeynet must balance vulnerability of resources and depth of deception to convince the attacker the services and resources are not decoys. Currently, botnet attacks are common to networks. Botnets are increasing in the numbers of bots found in a given botnet, the number of different types of botnets, and the level of deception awareness that botnets possess. The honeynet-aware botnet attacks have rendered honeynet deception strategies less effective. Botnets use well-known protocols for communication and control among bots. The bot herder can use these protocol channels to check on an infected node to see if that bot is still under his/her control. One of the things that a bot herder can verify is if the node is accepting malicious traffic or commands and passing that traffic outbound on to other bots (or potential bots) in the botnet. The propagation spread of this malicious data is crucial to the botnet's operation and existence. A honeypot is traditionally designed to attract malicious attackers to it-self, then contain that traffic, and not allow it to spread to other nodes. We seek to improve the depth of deception levels of honeynets when defending against honeynet aware botnets. We will do this with additional layers of complexity for the bot herder to interact. We will also add depth to the deception of the honeynet. We will construct a honeynet that can operate as an intrusion deception system, reacting to botnets that are honeypot aware with an intelligent deceptive response based on the kind of botnet.

Keywords: honeynet deception, honeypot, botnet tracking, intrusion deception

Introduction

Botnets are an extremely important part of today's current malware scene [PRIS11]. Malicious actors use them for everything from traditional Distributed Denial of Service (DDoS) attacks, phishing and spam email attacks, personal information stealing, keystroke logging surveillance, password cracking, fraudulent use of online gaming and polling sites, and other illicit uses [PROV07].

Security researchers must formulate an effective strategy to mitigate the malicious effects of botnets. Using honeynet technology has been proven an effective strategy in the fight against botnets as part of an overall intrusion deception system on networks [REF]. Even though researchers have found effective strategies of using honeynets against botnets, subsequently, botnet operators have found counter strategies to the honeynets with so-called honeynet-aware botnets [BAIL12].

In this paper, we explore methods of hardening honeynets against honeynet-aware botnet attacks. To search for solutions in mitigating botnet attacks, a fundamental activity that researchers have taken in the past is to track botnets. Therefore, we start with a more thorough understanding of the current state of botnet tracking.

The strategy, tactics, and information presented in the Niels Provos text [PROV07] has been established as a standard starting point for studying honeynet mitigation of botnets in the field. This is a good foundation for understanding the typical behavior of bot herders and their controlled botnets and applies directly to the historical behavior and activities of the varieties of Agobot, including the variant of Agobot, Phatbot-Ghettobot, that was used in this research [PROV07].

Fundamental Activity for Intrusion Deception Systems: Tracking Botnets

It is critical that the honeynet operator challenges the attacker by hardening the honeypots and increasing the *fidelity* of the honeynet by implementing realistic security controls to make the system appear as if it has value [WATS12]. Honeynet technology is a useful and extremely important part of an overall network security strategy if security professionals and researchers are to know their enemies, and insure that network security keeps pace with the rapid changes in network attacks [SPIT03].
A successful strategy of using honeypots and honeynets relies on them going undetected from external attackers. This desired quality leads us to describe honeypots and honeynets as intrusion *deception* systems. A part of the problem lies in the way that honeynets and honeypots reveal their true purpose through data containment and control [PROV07]. Honeypots are careful to control the propagation of data that is compromised or malicious data, which was generated by an attack [PROV07]. Attackers can observe the data control and data containment and reliably determine whether they are attacking a honeynet. This has rendered honeynets and honeypots less effective for securing networks and learning more about attackers and the malware that the attackers use [PROV07].

A botnet (robot network) attack is extremely common and honeynets regularly encounter botnet attacks [WANG10]. A botnet is designed to propagate traffic from one node in a network to another and build its herd of bot (individual robot node) controlled machines [TANG05]. When the attacker observes the deployment of a botnet and a bot encounters a traditionally deployed honeynet, the bot proliferation ceases because of the honeynet's attempt to control and contain the attack. Whether successful or not, this attempt can be detected by the bot herder through regularly practiced observations. This behavior reveals the honeynet's true purpose, and the attack will either subside or continue under a new disguise using alternative methods [TANG05]. The honeynet is no longer gathering malicious data or recording activities of malicious attackers [PROV07].

The purpose of this research is to create a honeynet that increases the disguise capability by allowing a botnet attack to infect a node and then allowing that node to propagate the botnet attack to subsequent nodes. In this approach, the botnet attack is observed and is mistakenly noted by the attacker to be a legitimate spread of a botnet deployment. This will harden the honeynet against detection by a honeynet-aware botnet. We will use the concepts of a multilayer deception system and the "double honeypot" strategy to add deception to the honeynet. We will propose an improvement to this deception by applying the concept of the BotSniffer design, which is based on behavior categorization, so that our hardened honeynet reacts to a honeynet-aware botnet according to some additional criteria. This additional criteria might be an intelligent algorithm that reacts to a fingerprinting determination of the specific type of botnet that is attacking, or it may be simply a random assignment to an additional node in the honeynet, creating a more difficult workload for the bot herder to check data from his sensors.

Botnet tracking and the effectiveness of intrusion deception systems (such as honeynets) is fundamental to understanding how the hardening of a honeynet is going to be accomplished. An explanation of what botnets are, what they do, and how they work, is critical to understanding the malicious activity of botnets. We will cover some basic examples of tracking methods for botnets and show how this method is tailored to the nature of a specific type of botnet and also how that tracking mechanism may not apply to the future evolution of botnets [BAIL12].

Definitions - bot, botnet, and botherder

A **bot** can be described as a remote control program loaded on a computer. Note that there is no mention of malicious (or other) intent. It is simply software. In fact, historically, as is often the case with malware, the malicious use of the software came about after legitimate uses had been established for which the software was originally developed, with malicious uses simply taking advantage of security holes in either strategy or execution that were left unattended.

A **botnet** is a network of these bots, or "robotic network". When we refer to botnets, we are talking about collections of bots or robotically controlled nodes [PROV07].

There are primarily 3 attributes for a bot: 1) a remote control mechanism, 2) the implementation of several commands, and 3) some kind of mechanism to spread itself in order to grow the botnet. The remote control mechanism will be a topic that is focused on quite a bit in this report, since it determines much about how the botnet can be tracked and what can be done about the botnet to mitigate or "take it down" [PROV07].

Note that we do not work from a definition of a botnet as a robotic network in the sense that robots can be thought of as autonomous, i.e., thinking and making decisions for themselves. The bots and botnets we deal with today are usually controlled ultimately by a human controller, known as the **"botherder".** A botherder will grow and cultivate his botnet, in a manner similar to amassing any collection of weapons or munitions that are going to be used as a combined task force [PROV07]. The growing of a botnet can be time consuming and difficult work, so the individual bots as well as the botnet itself become valuable property of the malicious actor. Plainly stated, the botherder has much

time and effort invested in the creation and maintenance of the botnet [PROV07]. This is so valuable that hackers often steal or hi-jack botnets from each other [PROV07].

Some examples of popular well-known botnets are Agobot, RBot, SDBot, Phatbot, Zeus, and Nugache [PROV07]. Botnet implementations attain popularity in much the same way that legitimate software package distributions do. The malware writers attain wide distribution and then maintain that customer base by nurturing upgraded features and improvements, bug fixes, and updated search capabilities, as well as normal price and value considerations [AGO06].

The anti-virus company Sophos has claimed that there are (quoted in 2007) over 1500 variants of Agobot Phatbot, making it one of the most prolific bots of all time in terms of the extensibility and flexibility of the design [PROV07].

Researchers distinguish botnets from worms by 2 of the 3 criteria mentioned earlier, specifically, that with bots there is a remote control mechanism so that the bot can receive commands from the bot herder in order to execute specific instructions on the remote system, and the commands themselves constitute a distinction helping to define a botnet [PROV07].

Attack Types

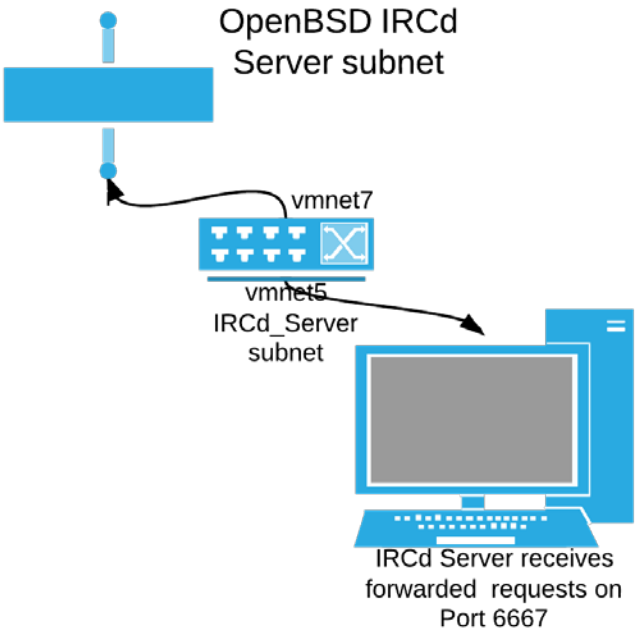
Botnets are used in a variety of different types of attacks. Email **spam** is an extremely common use for a botnet. In fact, early development of botnets was likely without malicious intent at all. Software that would be indefensibly malicious such as a **phishing** email attack would come later. An email phishing attack is often quoted as the most popular method of compromise used to get inside a network by malicious actors. Organizations have an extremely difficult time limiting the use of their network to work-only related actions. With SaaS so prevalent in today's working world, many employees may take a moment to check personal email at work [WATS12]. Even a well-intentioned employee, perhaps checking personal email to communicate with a spouse over shared child-care responsibilities, and using browser based email such as Gmail, Hotmail, or other, can be the target of a phishing scam sent to thousands of mailboxes and containing a malicious link that, if clicked on, will take the user to a drive by malware web server ready to deliver its payload [WATS12]. It is exactly this scenario that compromises many corporate and government networks. Network security specialists such as Mike Hamilton, CISO with the City of Seattle cites a "10%" rule, where he claims that if 1000 such emails are delivered to inboxes at a given organization, 10% of them (100 users) , or a count of 100 will be clicked on. Botnets are also commonly used for DDoS (Distributed Denial of Service) attacks [WATS12]. The mass replication of bots and the remote control mechanisms of botnets make them a perfect design match for a distributed denial of service attack, SYN flooding, and other attacks that are based on strategies of overwhelming servers and overloading resources until they fail [WATS12]. A botnet is also a good tool to use for a mass identity theft operation. Small amounts of data from multiple sources that are valuable to the bot herder but the exfiltration of data is not even noticed by an individual computer operator [WATS12]. Any attack which is based on a strategy of robotic automation and replication is a good match attack for a botnet to be used as the weapon. Agobot Phatbot can be utilized for any of the above mentioned attacks [WATS12].

Honeynet Aware Botnets

Honeynets are valuable security tools for gathering data on attackers. The amount of data gathered, and therefore the perceived value of the tool is directly related to the amount of time the attacker spends on the network. This quality is designated as the allure of a honeypot and must balance the vulnerability of the resources offered to the attacker and the depth of deception to convince the attacker the services and resources are not decoys [PROVO07]. Botnet attacks are common to networks [WATS12]. Botnets are on the rise as attack threats in several ways. They are increasing in the numbers of bots found in a given botnet; they are increasing in the number of different types of botnets; and they are increasing in the level of deception awareness that botnets possess [WATS12]. Honeynet-aware botnets have rendered honeynet deception strategies less effective. Botnets use well-known protocols for communication and control among bots. The bot herder can use these protocol channels to check on an infected node to see if that bot is still under his/her control [PROV07]. One of the things that a bot herder can verify is if the node is accepting malicious traffic or commands and passing that traffic outbound on to other bots (or potential bots) in the botnet [WANG10]. The propagation spread of this malicious data is crucial to the botnet's operation and existence. It is, in fact, one of the fundamental characterizations of malware that causes us to call it a botnet in the first place [WATS12]. A honeypot is traditionally designed to attract malicious attackers to itself, then contain that traffic, and not allow it to spread to other nodes [WATS12]. We seek to improve the depth of deception levels of honeynets when defending against honeynet aware botnets. We will do this with additional layers of complexity for the bot herder to interact. We will construct a honeynet that can operate as an intrusion deception system, reacting to botnets that are honeypot aware with an intelligent deceptive response based on the kind of botnet.

Command and Control of Botnets

At the heart of the remote control mechanism and the command set that a bot responds to is the communication system setup from the bots in a botnet to the bot herder who is controlling it. This is known as the C&C (Command and Control) infrastructure. Traditionally, botnets have used IRC (Internet Relay Chat) as defined in RFC 1810 in the late 1980s as the main channel of communication for messaging botnets. IRC has been preferred because it is relatively easy to setup, easy for the botherder to administrate the botnet, and easy for the botherder to compromise the IRC channels [PROV07]. The anonymous, non-recorded, non-archived nature of chat communications make IRC an attractive mechanism, especially in for the earlier bots. The popularity of IRC for the Command and Control of botnets continued until recently, when anti-botnet technology countered the botnet attacks by disrupting and otherwise compromising the IRC communication links. Botnets have existed on the Internet that have grown to several hundred thousand individual nodes [PROV07]. Each node represents a compromised computer that has remained compromised for a given period of time [PROV07]. Some instances of botnets, namely the botnet Nugache, have been known to grow to over 20,000 nodes in some instances of the botnet [PROV07].



Even though as a traditional IRC communication mode botnet must often use more than a single IRC channel for communication as the botnet grows in the number of bots, the tracking of the botnet is still focused on the IRC server or servers being used for C&C and the tracking strategy scales well to the size of the problem for the tracker [PROV07]. The ability to take down a botnet of such a large number of compromised computers by tracking the communications through a "single point of failure" has led to botnets migrating to other, more sophisticated means of communications.

Newer designs of botnets use different communication channels and protocols, but just as the case with computer languages such as C, FORTRAN, and BASIC, malware that has been released in the wild and has established a black market rarely completely disappears while it can still be used effectively [OLLM09]. This is even more true of botnets because the nature of their use is to compromise multiple nodes on a network and this feature establishes a connection between effectiveness and the quantity of compromised bot nodes [OLLM09].

The Phatbot-Ghettobot variant of Agobot, used in this research uses IRC as the C&C mechanism for communication. IRC allows the bot herder to communicate in real time with the bots in the botnet and all types of IRC channels are used. More recent examples of botnets are using more sophisticated C&C mechanisms such as Nugache, which employs a P2P (peer to peer) network for communication of C&C. This scheme of using P2P offers the same benefit to the bot herder that makes P2P popular in the first place, it eliminates the single point of failure that the IRC server creates in the C&C communication [OLLM09]. So, a botherder can lose a node in her botnet, but the rest of the botnet is still in communication, still receiving messages, and still operational. Many of the takedowns of botnets have been carried out by destroying the communication of the IRC C&C server [WATS12]. This is a classic and basic military strategy, to disrupt communications [SPIT06].

Even though bots can be programmed to carry out some basic malicious activity autonomously, such as the way a worm spreads blindly to additional nodes, this, much noisier, network activity is more likely to be noticed and mitigated in some way. This has led to the popularity of newer botnet types such as Nugache which use a distributed C&C mechanism via P2P protocols [PROV07].

Some botnets have used even more esoteric communications, including lighting different pixels in GIFs, JPGs, or other image files that are posted on a social media network [WATS12]. This could be the ultimate steganography based approach to securing communications from the bot herder. It would be very difficult to prove that communication was being passed this way, and even more difficult for network security admins to effectively block all traffic from this kind of an approach [WATS12]. HTTP is also used as a C&C channel over the standard ports 80 or 8080, so again disrupting communication becomes extremely problematic when done over a very commonly used and permitted channel [BAIL12]. Bots can also use covert channels such as a DNS tunnel, or the bot herder can encode requests to the bots inside of a regular HTTP request object. C&C of botnets has even been analyzed as having taken place inside of DNS TXT records [PROV07]. It is the anonymous nature of the IRC culture that attracts botherders to use this type of communication and that is likely to keep IRC C&C communication linked with botnet operation for some time to come [WATS12].

We will be using UnrealIRC as our IRC server for establishing C&C communication from botherder to the bot nodes. UnrealIRC is a full featured IRC open source software distribution that has a rich feature set, the bulk of which we will not need and will not use. UnrealIRC remains a popular IRC software package however, and many botherders have used compromised UnrealIRCd installations and therefore UnrealIRC was an appropriate choice as part of the study for this research [WATS12]. Ago himself, warns users of Agobot to not attempt to configure UnrealIRC [AGO00].

IRC and other Command and Control Specifics

We are using Phatbot-Ghettobot which is a common variant of Agobot released sometime in the 2000 to 2004 timeframe. The name is usually stylized in print, including its own documentation, using "Leet speak", i.e., "1337-speak" as "Phatbot-Gh3tt0b0t". Phatbot-Ghettotbot uses IRC C&C channels to communicate. We feel that much is written in academia and the press currently about the continued use of IRC for botnet Command and Control and these issues need to be addressed to prove the importance of this research.

There are decentralized schemes and traditional C&C schemes, which could be considered as simply non-decentralized [OLLM09]. The popular C&C schemes are the traditional IRC chat servers, web pages, and internet messenger software. The popular decentralized types are peer to peer networks and trust rings. Combined schemes are possible and will make an effort to combine the best parts of various technologies [OLLM09]. IRC was traditionally used because it is simple and fast to deploy the communication network in the wild. Typically, the botherder wants to find an IRCd server somewhere (not on her own network) where she can operate anonymously. For small amounts of bots, only a single server is needed, which further simplifies the botherder's tasks [OLLM09]. This type of C&C channel can be grown by joining several servers. Many of the most current botnets released now do not use IRC C&C communication. They have more sophisticated methods but they are much more complicated to set up

as well, using such techniques as port knocking to evade detection and also increasing the complexity of the botherder's tasks [OLLM09].

Because both bots and IRC were developed in the 1980s to allow users to share legitimate services and share communications in real time, the IRC mechanism was a perfect match for the C&C requirements of botnets. This could almost be thought of as an early form of a mash-up: The combination of two existing software services for a completely new purpose, albeit a malicious one [PROV07].

Once the IRC server machine has been selected by the botherder, she can direct all or some of the bots to join the chat on a specific channel on the IRC server software. The IRC server can be secured using authentication such as passwords or other methods, and some use SSL-encrypted communications to further mask the intentions of the bots in secrecy. All messages are interpreted as commands in the vocabulary of the particular bot [PROV07]. A comprehensive listing of all botnet commands for operating Phatbot-Ghettobot are included in the accompanying documentation that was apparently distributed with the software [AGO00]. There are various html and readme documents along with a complete table listing the possible commands available to direct the bots supplied by Ago himself. There are, additionally, some immature threatening messages concerning the consequences of software piracy of the botnet's code and especially the scanner classes letting Ago's criminal clients know what he intended to do to them should they illegally copy the illegal software designed to steal from others! So, we could assume that the documentation command listing was what was actually delivered to criminal clients when a blackhat hacker purchased Phatbot-Ghettobot. As if this were not incredulous enough, the botnet included essentially a copied version of the GPL. It is interesting from the standpoint of the history of network security and the role that social engineering plays in hacking that the mindset of someone who was engaged in this criminal activity doesn't appear to be able to see forward to a time when there would be any attribution of their criminal actions assigned to them whatsoever [AGO00].

Attribution is a key issue for the choice of C&C channel to communicate from the botherder to the infected bots. In a realistic scenario one might find in the wild, the server chosen would not be on the same subnet as the botherder [WATS12]. The clandestine nature of the botherder means that she will likely choose a public IRC chat server to use for communication to her bots to avoid attribution of malicious and criminal activity back on to herself [WATS12]. This is reflected in the configuration of the virtual network in the experiment.

C&C via a web server is similar in complexity to the setup involved in IRC [OLLM09]. We server communication allows for more possibilities of control scheme variations than IRC and although it is easy to deploy and control, it is also easy for the botherder to lose control of the bots communicated to in such a scheme [OLLM09]. Conficker utilized a scheme of random domain name creation to set up C&C servers with domain names that were only valid for an extremely short period of time [OLLM09]. Internet Messenger has been used as a control mechanism for botnet operations, but it has not been a popular replacement for IRC due to the difficulty in creating the communication network. Registration for a single user is simple, but automation of the task becomes problematic considering large numbers of bots and automation is botnets in the thousands and tens of thousands [OLLM09]. Schemes such as

peer to peer and trust ring are considered even more complex and difficult to develop, but because of their decentralized nature, they also considered harder to take down and destroy [OLLM09].

To summarize the IRC C&C position, the strong argument for IRC as opposed to more complex types is that IRC allows the botherder interactive control of the bot [OLLM09]. This means that a botherder can send arbitrary specific commands to a specific bot [OLLM09]. This gives the botherder a very high degree of control using IRC along with comparatively easy setup and effort [OLLM09]. Achieving the same level of granularity with a method such as website communication would require customized server software [OLLM09]. IRC communications could also be considered more thoroughly tested, and the availability of free software implementations that can simply be used is a factor leading to their continued popularity as well [OLLM09].

Botnet Spreading mechanism

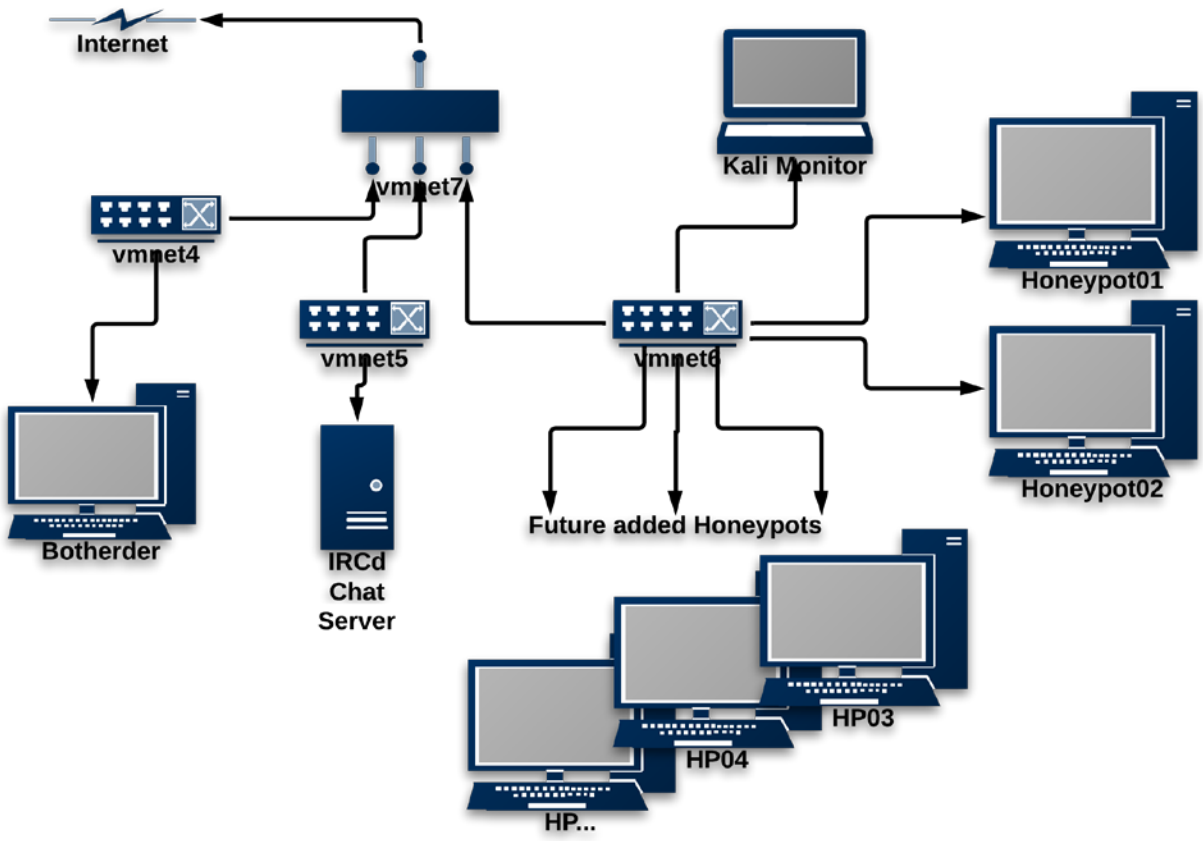
Frequently, the various different bot designs focus on using a specific group of exploits against specific well known ports for the purposes of spreading and growing the botnet. Botnets aggressively try to compromise more machines and they will be generating a noticeable amount of network traffic as they do this [WATS08]. Now that this history of compromise has been established previously, new bot variants will begin to attempt to spread by trying these same well-known exploits again. This pattern could be compared to criminal activity in the physical world as burglars will return to a previously compromised property or armed robbers will target a chain of stores that share the same cash handling and payroll distribution activities. This also gives security researchers a well-known list of places to look for bot activity and attempted compromise of a network [WATS08]. Following is a partial list of the basic Microsoft file and resource sharing ports used and the services typically using those ports:

Common Port Vulnerabilities Table

PORT ##	Description
135	Microsoft RPC (remote procedure call)
137	Microsoft NetBIOS UDP
139	Microsoft NetBIOS TCP
445	Microsoft DS (File sharing and resource sharing services)
42-	WINS (Host Name Server)
80, 8080	http or www vulnerabilities in IIS (Internet Information Server) 4 & 5 or Apache
903	NetDevil backdoor
1025	MS Remote Procedure Call service and Windows Messenger port
1433	Microsoft SQL Server
2745	Backdoor of Bagle worm (mass mailing worm)

3127	Backdoor of MyDoom worm (mass mailing worm)
3306	MySQL UDF Weakness
5000	upnp (Universal Plug and Play)

Information on common botnet targets in table from [PROV07] and from [WATS12].



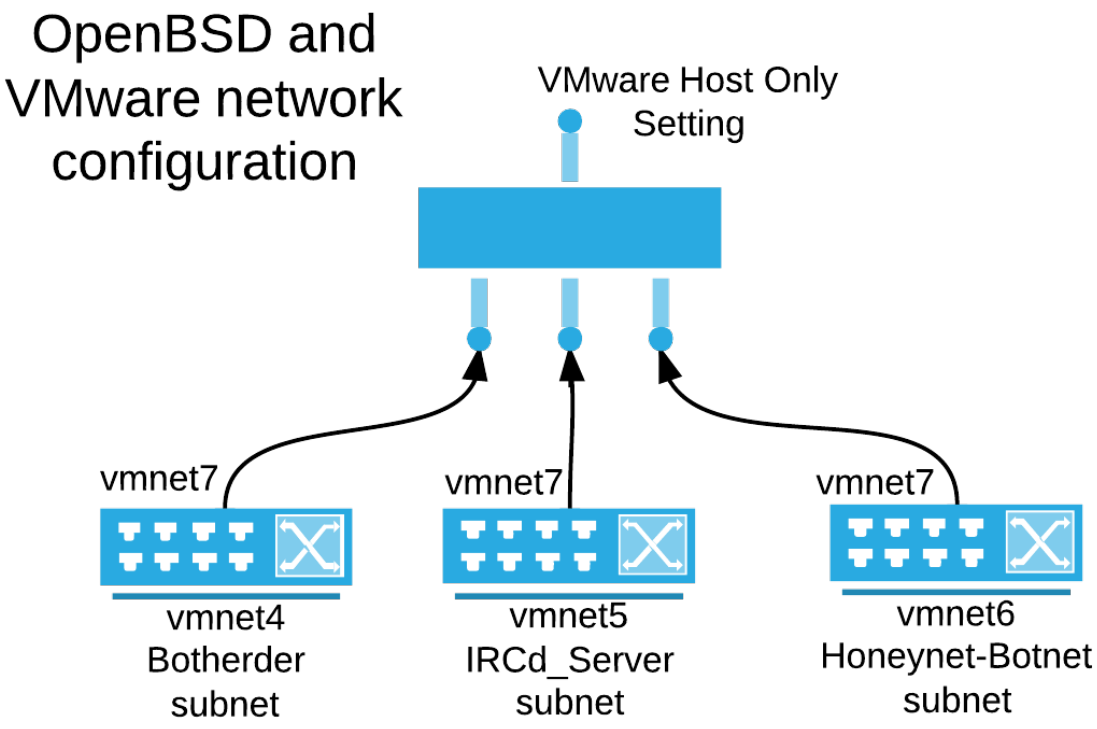
Network Laboratory Configuration

The virtualized network is at the heart of the experiment. Following is a complete description of the network that was employed for the lab experiment setup.

- There are 3 routers that make up the network infrastructure to emulate three separate subnets, one for the bot herder, one for the Honeynet, and one for the IRC Chat Server functioning as the bot network's Command and Control channel.
- The virtualization hardware platform was a Macintosh MacBook Pro 15 inch from late 2011 running OS X version 10.8.4 (Mountain Lion) with a 2.4 GHz Intel Core i7. The MacBook was loaded with 16 GB 1333 MHz DDR3.
- The virtualization software used is VMware Fusion Professional 5.0.3.
- The 3 routers are built using OpenBSD 5.3 configured with 256 MB RAM and 3 GB HDD each.
    - tcpdump is the only software used on the routers, and comes pre-installed with the OpenBSD distribution
- Behind each of the 3 OpenBSD 5.3 routers above are the following subnets:
    - The **Botherder** machine, behind the router designated in VMware as vmnet4
        - Windows XP Professional SP 2, 2 core processor with 1GB RAM and 40 GB HDD.
        - Visual C++ 6.0 (this is the last VC release prior to .NET)
        - VC6 Service Pack 5
        - The Platform SDK (from MSDN subscription CD dated March 2003)

- Visual C++ Processor Pack - The VCPP referred to in the Agobot documentation was not installed, and was not needed to compile the Phatbot-Ghettobot variant.
- Phatbot - Ghettobot - A variant of Agobot likely released circa 2000 - 2003
- XChat 2.8.9 - Chat client for testing communication to the infected bots outside of the botnet communication channel
- WinDump - Win32 version of tcpdump with Winpcap.dll driver
- Google Chrome (current)- Stock install of IE with XP Pro SP2 was unusable

- An **IRC Chat Server**, behind the router designated in VMware as vmnet5
  - IRC Server platform is a Windows Server 2003 R2 Enterprise Edition Service Pack 2 installed running on a 2 core processor using 2 GB RAM and 40 GB HDD.
  - Added software for IRC Server
    - UnrealIRC 3.2.10.1 - Internet Relay Chat server software to host the IRCd service used by Botherder for Command and Control of the botnet
    - XChat 2.8.9 - Chat client for monitoring the chat traffic on the IRC server
    - WinDump - Win32 version of tcpdump with Winpcap.dll driver
    - Google Chrome - Stock install of IE with XP Pro SP2 was unusable
- The **Honeynet** component, behind router designated in VMware as vmnet6
  - Honeypot nodes are configured with Windows XP Professional with Service Pack 2, 512 MB RAM with a 10 GB HDD.
  - WinDump - Win32 version of tcpdump with Winpcap.dll driver
  - Google Chrome - The installed version of IE with XP Pro SP2 was unusable
  - Additional identical honeypots can be added for future work
- The **Kali Linux Honeynet Monitoring** Station Component
  - A component of the honeynet subnet, behind router designated vmnet6
  - Debian Release Kali Linux 1.0 32-bit Kernel Linux 3.7-trunk-686-pae
  - GNOME 3.4.2 desktop
  - Single core processor with 768 MB RAM and a 30 GB HDD
  - Executed an ***apt-get install update*** after installation
  - Primarily used a combination of nmap, tcpdump, and Wireshark



The Routers

The OpenBSD routers are the foundation of the network. Each router has 2 network interfaces, an inward facing adapter, and an outward facing adapter. The outward facing adapter (which we considered to be facing the simulated "internet") is designated in VMware as vmnet7 for all 3 subnets. This is a Host only configuration in VMware and so adds a measure of safety when the live malware is executing. The router configuration enables the project to have the botherder, the IRC Server, and the Honeynet/botnet nodes each on their own separate subnet. We feel this is an extremely realistic simulation of what the configuration of a live botnet might look like [PROV07].

Routers, Switches, Hubs, and time frames

An issue that we wish to address here is the historical timeframe for the experiment. The technology of Agobot, as the variant is used in this project is from approximately the timeframe from 2000 to 2004. Ago (alias Wonk), was a German hacker who developed Agobot and its variants, including Phatbot-Ghettobot [AGO00]. Ago was arrested in 2002 and went to prison in 2004 for computer crimes [PROV07]. Certainly the development and marketing of the Agobot botnet and its variants were a part of this criminal activity [PROV07]. This makes the dating process extremely difficult, as the documentation of this individual's actions are more than a little off the grid and did not cease with his arrest. What dating we did do relies heavily on the documented use of software, especially in the area of the legitimate development tools that are specified in the documentation [AGO00]. The use of Visual C++ 6.0, its Service Pack 5, and the (rarely mentioned elsewhere) VCPP Processor Pack put the development of some of these variants in a much more narrow time frame, according to the Agobot documentation

itself [AGO00]. The Platform SDK is helpful in dating the software and activity because Microsoft produced this development kit for a time prior to the advent of the Microsoft .NET Common Language Runtime, and Ago prominently discusses the use of the Platform SDK in the documentation of how to build Agobot and its variants, with not just words, but screen shots of the Visual C++ IDE dialog boxes [AGO00]. The Platform SDK we used to build Phatbot-Ghettobot was obtained from MSDN subscription CDs dated 2001. All of the Microsoft development tools used were from this same MSDN subscription from 2001.

Where this background on the tools used to build the botnet become important to the network topology is in the area of the migration of networking practice over the years from a proliferation of hubs to one where more switches and routers are used [TANN12].

If one of the infected bots is commanded to put the VM's network adapter into promiscuous mode, and that node is attached to a simple hub, then network traffic intended for all ports can be sniffed and analyzed [TANN12]. This is not botnet specific behavior but applies to any topology involving simple hubs as opposed to switches [TANN12]. In other words, the botherder will have access to a much more inclusive collision domain of information about what is taking place on that segment of the network if the network device is a hub [TANN12]. This scenario might likely have been the case in 2001, 2002, 2003, or 2004 [TANN12]. If, however, we look at network subnetting using switches, as it is commonly practiced today, that scenario might look a little different. It would be common practice today for those hubs to become switches, and with the same network adapter in promiscuous mode, the network traffic that an infected bot will be able to reach will only be the traffic intended for that specific collision domain [TANN12]. The intelligence built into most modern switches will allow two ports (on the switch) to communicate across the switch without broadcasting that traffic to other ports on the switch [TANN12]. This is the configuration setup for the lab experiment presented in this research.

This decision could mean that even though a more realistic setup for a botherder operating an Agobot variant today would have the benefit of chance encounters of more routers and switches and fewer hubs, it also means that the results of this research are more relevant to today's network subnetting configurations [WATS12]. We felt the trade-off was worth it, as we wanted the research to be as relevant as possible.

Honeynet

The Honeypots and resulting honeynet that make up this experiment are some of the most basic in design that could be used. This decision was important to the design of the overall experiment. Although several well-known Honeypot implementations were considered, in the end they were all discarded in favor of controlling and reducing complexity. We did not need to practice real deception in order to produce our result for this deception system [WATS12]. This system is designed to simulate a botnet attack and track network communication from the botherder through the chat server to the infected bots/honeypots and finally, to observe what communication attempts are made back to the botherder or what information the botherder can obtain from the operation of her bot sniffing the traffic around the infected machine. This sniffing activity and the effort that it requires are critical to the

revealing of the infected node as a honeypot [WATS12]. The actual activity of the honeypot and what it does to the malware is not central to the research presented here.

Therefore, the honeypot implementation that we selected was simply a Windows XP machine with SP2 installed and little else in the way of additional software to add any complexity to the configuration. Additional software to the XP machines is limited to an XChat client for testing communication through the Chat server outside of the botnet's capability, WinDump (the Win32 version of tcpdump) to provide a direct window into the tcp traffic at that VM's connection, and Google's Chrome browser simply for facilitating other software downloads etc. Windows XP is perhaps the most widely attacked computer platform both currently and historically, and it is often used as an attack target platform. Working with a fresh install of Windows XP, even with the application of Service Pack 2, still leaves the researcher with an Internet Explorer browser that has so many problems interacting with modern web sites that it is rendered essentially useless.

Command and Control - IRC Chat Server

The Chat Server used for Command and Control in the project is of special significance for all the reasons previously described for the Command and Control choices of a botherder [OLLM09].

Kali Linux Honeynet Monitoring Station

In addition to the internal honeypot monitoring that we set up for the honeypots to monitor the botnet activity, we also provided another monitoring station on the honeynet subnet. This node was an instance of Kali Linux. This machine was used for more in depth monitoring of the tcp packet data. We did some static analysis by using the combination of nmap to stimulate noise on the network and simultaneously using tcpdump to write that network traffic to pcap files for later analysis using Wireshark. We also used Wireshark in its real time capture mode in order to be able to look at the results of commands being issued to the bot nodes.

Agobot Creation

Agobot was written originally around 2000 by a West German hacker named Ago (alias) Wonk. Ago later was convicted of computer crimes and went to prison for his activities sometime around 2004. The vocabulary of the command listings is built from C++ classes that are well-designed, and easily modified using normal C++ mechanisms (inheritance and overloading). Most security researchers agree that Agobot is a well written piece of object oriented designed software that is highly modular and loosely coupled. In other words, it was written by a very good programmer [PROV07]. It allows the owner to add or subtract features so that the targeted use of the botnet can be modified to suit a particular client or group of clients over a long period of usefulness. This last feature is important, because the malware author can leverage maximum profit from his effort due to the product's ability to be modified [PROV07]. An important note is that the customers of this product are hackers themselves. In reading the documentation, it is obvious that Ago experienced the same software piracy issues that legitimate software businesses have [AGO00]. Agobot is a successful design of malicious software and it is also successful business product strategically aimed at multiple criminal enterprises [PROV07].

Agobot Phatbot-Ghettobot Implementation

The Command reference included in the software shows the modularity of the C++ class design and the relatively high skill level of the botnet's author (or authors). **commandref.html** is the included command reference documentation that was delivered with the bot. It clearly shows a modular, user friendly approach to operating the botnet from the bot herder's perspective [PROV07].

Here are some examples of the clean, high level of the design and the modularity of the software, compared to other bots that had been captured and were extremely popular such as Rbot, SDbot and variants of those bots [PROV07].

The file **harvest_cdkeys.cpp** illustrates the clean Object Oriented design of the original code. This class file refers to a specific attack that is designed to capture Microsoft (and other vendors) CD Product ID Keys so that the bot herder can sell the pirated software or use the keys illegally himself.

httpflood.cpp is an example of an attack type (an HTTP Flood attack) that is modularly added to the bot. Due to the clean OO nature of the design it is easy to see how future evolving attacks can easily be updated to the bot so that the original human and social engineering effort can be leveraged by the black hat to keep current with anti-hacking measures.

hook.cpp shows the modular approach to the DLL hooking mechanism which is critical to any botnet performing as a man in the middle approach to controlling the remote operating system. The Windows API system call is hooked by the botnet's running code and the botnet can redirect the call to do any arbitrary thing that he/she has designed it to do. The bot is operating as a kind of rootkit kernel in front of the operating system.

irc.cpp illustrates the modularity of attaching the bot to the command and control channels of the IRC server.

installer.cpp is the C++ class that controls adding or deleting entries into the compromised systems autostart registry sections. This allows the bot herder to insure that the bot and any other designated programs execute automatically at boot of the compromised machine.

Types of Attacks

There are a variety of different types of attacks that can be carried out by botnets. Even though there are literally every type of attack possible with botnets because of the remote capabilities of the malware type. Common attacks include carrying out DDoS attacks using either SYN flood or UDP flood attacks. Flood and DoS attacks can be extended into spidering attacks, a type of attack that recursively traverses down through a tree against a website or URL and so more completely attempts to overload the server.

In fact, there really aren't very many limits to the types of attacks that can be carried out using botnets because of the nature of the malware [WATS12]. Remember that a botnet is nothing more than a mechanism to remotely control a software program operating on a computer with admin privileges on the remote computer [PROV07]. Botnets were not originally developed with malicious intentions attached [PROV07]. Since most botnets include a mechanism that allows the bot herder to update the bot by downloading arbitrary updated executable files to the bots, this allows the bot to keep up to date with lists of scanners and update the list of attacks that the bot is capable of [PROV07]. Another way to think of this is that the bot can do bad stuff and it can also update so that it can do more new bad stuff.

Example Tracking Strategy

Following an example of a tracking strategy for a botnet, this experiment will copy the example that [PROV07] used in 3 basic steps. First, a honeypot is used to collect the samples of malware in the monitored network. WE will only be simulating the collection of malware, as that is not a part of this research. WE will monitor the activity of the botnet through monitoring the botnet's communications only. Next, a specifically purposed Honeypot design such as Cuckoo would be employed to automatically classify and analyze the malware, and again, we will only be simulating this part, since we are not actually collecting the malware. We will be relying on the Kali Linux monitoring station and the communications of the botherder to the bots to verify the malicious traffic. Finally, a tool such as **Botspy** might be used to continue the observation of the malware [BAIL12]. This part of the tracking will also be simulated with only observation of the traffic using tcpdump, WinDump, and Wireshark. One of the main goals of the monitoring and intelligence gathering effort is to be able to rtead the communications specifics of the botherder well enough to be able to shut down the C&C channels that the botnet is using. This experiment again simulates and "proves" the ability to shut down the IRC C&C communication step by simply monitoring and observing the communications.

The basic strategy can be summarized as:

1. *Collect malware*
2. *Extract information*
3. *Observe Command & Control Server*
4. *Shutdown C&C network*

Specifically, the information we will be able to use to stop the attack includes the DNS location and the IP address of the IRC server as well as the ports being used. If there is a password to authenticate to the server, we may be able to obtain that, depending on if there is encryption being used and the strength of it. Individual bots in a botnet have nicknames as nodes in a legitimate network do. This information is useful as well. Bots have what is known as an IDENT structure, a table of attributes and values that contain all the critical information about the bot's network identity, including the name of the IRC channel being used and any authentication needed for joining [PROV07].

Using this methodology for tracking the botnet, one can see that the use of the traditional IRC standing server is a weak point in the botnet's operation strategy and for this reason, communication disruption or blocking of some of the non-IRC mechanisms are much more difficult. Couple this added difficulty with the fact that disruption of a single bot node does effectively nothing to stop the overall attack footprint of a botnet and in fact, may lead to the smuggled honeypot alerting the bot herder to its presence and revealing its true purpose (honeypot) and we have a tracking method that will no longer be effective. Many botnet tracking methods stem from strategies that are covert in nature and are based off of specific implementations of the botnets and for that reason are individually tailored to the type of botnet and focus on a specific aspect of the bot either implementation, deployment, design, configuration, and even location [PROV07]. Knowledge of details of the specific network that has been compromised can also sometimes be a factor in successfully tracking a botnet. Botnets are often operating in the wild as connected networks.

Social Engineering

Even though the experiment is a simulation of a Honeynet being attacked by a botnet, social engineering is an issue in botnet operations that must be addressed. The consensus among botnet experts is that IRC C&C bots will often use unusual port numbers for communications channels [PROV07]. Historically, botnet operators (and in fact, criminals in general) like to talk and brag about their exploits and criminal gains [PROV07]. Botnet operators will discuss their activities on the IRC channels that they are using to control their bots. Lance Spitzner has drawn a parallel between the level of expertise or professionalism (if it can be called that) of a given bot herder, and the amount of time spent present on the network in the IRC chat talking not just to his/her botnet but to other bot herders. [SPIT06] observed a correlation with the amount of time spent by bot herders in the chat channels. The greater the amount of time the botnet operator spent "hanging out" and chatting, the smaller the corresponding criminal enterprise usually was and the less experienced the botherder [PROV07]. The most skilled professionals spent the least amount of time issuing commands [PROV07]. This strategy can also be mapped to military behavior. The less exposure a target allows, the smaller the target appears to an enemy, and the better the chances for successful defense of that target [SPIT04].

Human intelligence experts have also agreed that from a social engineering standpoint, however, this doesn't mean that the time a white hat social engineer spends chatting with black hats is of low value [PROV07]. It was determined that the bragging and talking going on among the less experienced black hats often led to excellent human intelligence on all the bot herders using IRC, both low skilled amateurs and high skilled professionals [PROV07]. Even though social engineering is a big part of tracking any malicious activity, there is no real deception in this experiment, only the simulation of deception.

Approach

In order to prove the viability of the project as a honeynet hardening technique, we decided on a 3 step approach, and then later modified to a 2 step approach with the third step being reassigned to future work. Each step has an associated Use Case scenario that we will follow to build on a step by step basis.

1. We will implement a solution in the test environment to demonstrate hardening of the honeynet against a honeynet-aware botnet attack.
2. We will answer the question posed by [WANG10] of why propagation from a single honeypot to one other honeypot can be sufficient deception, and show this in the test environment.
3. The plan to modify the "double honeypot" solution proposed by [TANG05] against a worm attack to work effectively against a honeynet-aware botnet attack will be moved to future research.

The first step is to implement the system that was proposed by [WANG11] and others to show that Honeynet aware botnets can sniff traffic and check the compromised bots to see if the nodes are currently (on an ongoing basis) compromised and still a part of the botnet [WANG11]. If the sniffed traffic indicates that a particular bot is no longer obeying the commands of the botherder or acting in

some suspicious manner (such as a Honeypot might if it was attempting to carry out its mandate for data capture and containment of malware) then the botherder may suspect she has encountered a Honeypot and cease the malicious activity or direct efforts elsewhere [WANG11].

The compromised bots are also tested by the botherder to verify if that bot is obeying the commands received via C&C communication from the botherder and carrying out those malicious commands [WANG11].

Background

The importance of this research rests on key background work performed by [WANG10] in the area of honeynet aware botnets and [TANG05] in the area of utilizing a double honeypot deception approach against a worm attack, as well as [GU08] who developed the tool BotSniffer to detect botnet Command and Control Channels in network traffic. Table 1 shows a comparison of these approaches in terms of four criteria – 1) hardening a honeypot or honeynet, 2) defending the honeypot or honeynet against botnet attack, 3) defending the honeypot or honeynet against work attack, and 4) increasing deception by propagating attacks.

Table 1. A Comparison of Previous Work with 4 Criteria.

Criteria	**Ping, Wang, Lei, Wu, Cunningham, and Zhou, Honeynet Detection in Advanced Botnet Attacks [WANG10]**	**Yong Tang, Shigang Chen, Defending Against Internet Worms A Signature based Approach [TANG05]**	**Gu, Zhang, & Lee, BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic [GU08]**
Hardening honeypot/honeynet?	**Yes.** Suggested methods as further research but not implemented	**Yes.** Uses complicated extra steps of "double honeypot" propagation idea against worm attack. Not employed against botnet attack.	**No.** Botnet detection technique is not applied to honeynet or honeypot
Defends against botnet attack?	**No.** Illustrates how botnet attackers can use software and hardware sensors to check propagation of malicious unmodified data being passed through botnet to known IP nodes they control.	**No.**	**Yes.** By detecting botnet command and control channels and intercepting communication back to bot herder.
Defends against Worm	**No.**	**Yes.** Uses	**No.** Targets botnet

attack?		complicated "double honeypot" idea which forces attackers to take extra complicated steps to determine deception.	communication channels which are not used in worm attack.
Increases deception by propagating attack (within Honeynet control)?	**No,** but illustrates how botnet can measure propagation of unmodified malicious data.	**No**	**No.** Deception is used only to spoof communication back to botherder, not to propagate the attack within the network.

Based upon the comparison of the above previous works, to begin we discuss how honeynet detection will take place with a honeynet aware botnet. In [WANG10], the bot herder is capable of sending signals to the bots under his/her command and instructing a new bot that malicious traffic be passed on to additional nodes on the network to attempt infection or other malicious actions.

Additionally, the bot herder has the capability of using sensors. The purpose of these sensors is to monitor incoming transmissions from compromised nodes and sniff traffic around them to send reports back to the bot herder with verification of whether modes are still compromised and under her control. The botherder is able to check whether malicious traffic is being contained within a bot, or modified coming from a bot so that it is no longer malicious. In this way, the bot herder is able to make determinations whether or not the bot she is adding to her botnet is a honeypot or a fully compromised node that is listening to her commands [WANG11].

There are different approaches of increasing the complexity of deception systems to harden honeynets. The work of [TANG05] researches additional complications imposed on malware attempting to detect honeypots. The researchers do this by adding additional layers of complexity to the deception scheme. In the simplest next step, they introduce a “double honeypot” deception system against a worm attack [WANG05]. The design of the “double honeypot” configuration simply adds layers of complexity to the detection methodology employed by the malware or the malicious actors deploying it. Simply speaking, it is more work for the bot herder, and therefore harder, to check an additional node for presence of a honeypot than to simply check the first node and be done. If the honeynet operator can propagate the attack from the first honeypot to an additional honeypot, and yet retain the malicious code within the honeywall, then hardening of the honeynet has taken place. The additional manual work for the attackers adds additional layers of deception and additional layers of hardening against detection as a honeypot.

The bot herder has to have some mechanism to communicate with the bots in his botnet. These C&C Channels are, in some cases, well-known protocols such as IRC and many bot herders use well-known internet locations to network their communication systems [GU08]. Botnets differ from other forms of malware such as worms in that they utilize these C&C channels [KIM10]. These channels are the critical communication link for bot herders (those that control the botnets) to send commands to the bots comprising their botnet [GU08]. There are a variety of different types of communication that bot herders use for command and control of their botnets [GU08]. Two of the most popular protocols used are Internet Relay Chat (IRC), Domain Name System (DNS) tracking (whether actual DNS names or directly using IP addressing) [GU08]. Our experiment uses UnrealIRC IRC communication.

In response to this large problem caused by botnets, there have been tools such as BotSniffer developed specifically for tracking botnets [GU08]. BotSniffer is an anomaly based detection system that looks for anomalies in what would be considered "normal" network traffic [GU08]. BotSniffer uses the strategic theory that the same bot herder controls bots in a botnet and the bots run the same malicious program. This means that they are likely to respond similarly to the bot herder's commands and conduct similar activities in a similar manner [GU08]. We will be using tcpdump, WinDump, Wireshark, and the logging capabilities of the OpenBSD routers to look at the traffic in the communication channels.

Approach

We propose our approach that increases deception of the honeynet through botnet attack propagation from the attacked node. We harden the honeynet by allowing malicious traffic to propagate unmodified to other honeypot nodes, strengthening the impression of botnet propagation. Controlling the spread of the botnet to other honeypot nodes in the honeynet will increase the perceived deception of the botnet while isolating the malicious activity from any production network machines. We will also increase the deception by allowing the botnet to spread to additional nodes.

User Scenario (Use Cases)

We will setup our virtual network for a botnet attack using Phatbot-Ghettobot, a variant of Agobot (one of the 2 most well-known botnets), which can be deployed for several different kinds of well-known attacks, such as a DDoS (Distributed Denial of Service).

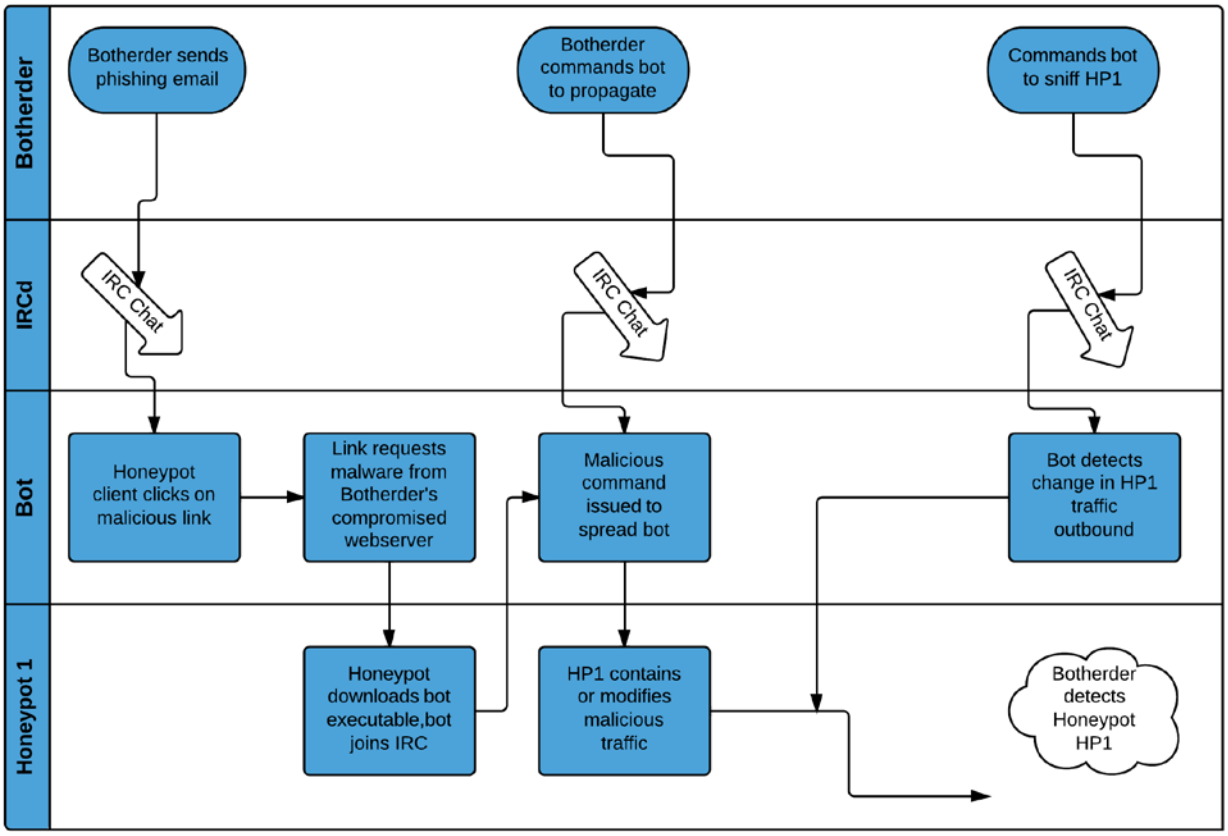
As a previously assumed setup configuration before beginning each Use Case, the following steps will be assumed to reach the point of compromise on a victim computer within the test honeynet. The following list of actions are a preliminary setup for each of the 3 Use Cases we planned on running in this project. We setup the experiment in a virtualized environment and ran the first 2 Use Cases. We were not able to run the final Use Case, and we will leave it for future research.

- An assumed strategy is for Phatbot-Ghettobot to attack a Windows XP workstation
    - (this would be **Honeypot01** in the diagrams)
- User of **Honeypot01** has clicked on an active link in a phishing email that executes a callback
    - The callback will request a malware download of the bot executable

- The above download would have been unknown to the user of the machine
- The above **Honeypot01** is now an infected bot.

Use Case #1:

- **Honeypot01** will now respond to the botherder with a list of commands it can carry out
- **Honeypot01** acts on its botherder's commands and attempts to spread the malware
- The malicious traffic ordered by the botherder in the above step is not carried out
- **Honeypot02** will not receive any malicious traffic from **Honeypot01**
- Botherder sniffs traffic outgoing from **Honeypot01** to see if malicious traffic is being passed
- Botherder sees data containment at **Honeypot01**
- Botherder suspects **Honeypot01** is a Honeypot
- Attack ceases



Discussion of Use Case #1

Use Case #1 implements the experiment described in [WANG11] with no alterations, and no hardening of the Honeynet. The botnet was spread to the Honeynet and the botherder was able to detect that the infected bot was not obeying the commands form the reference and therefore, the botherder detected a Honeypot [WANG11].

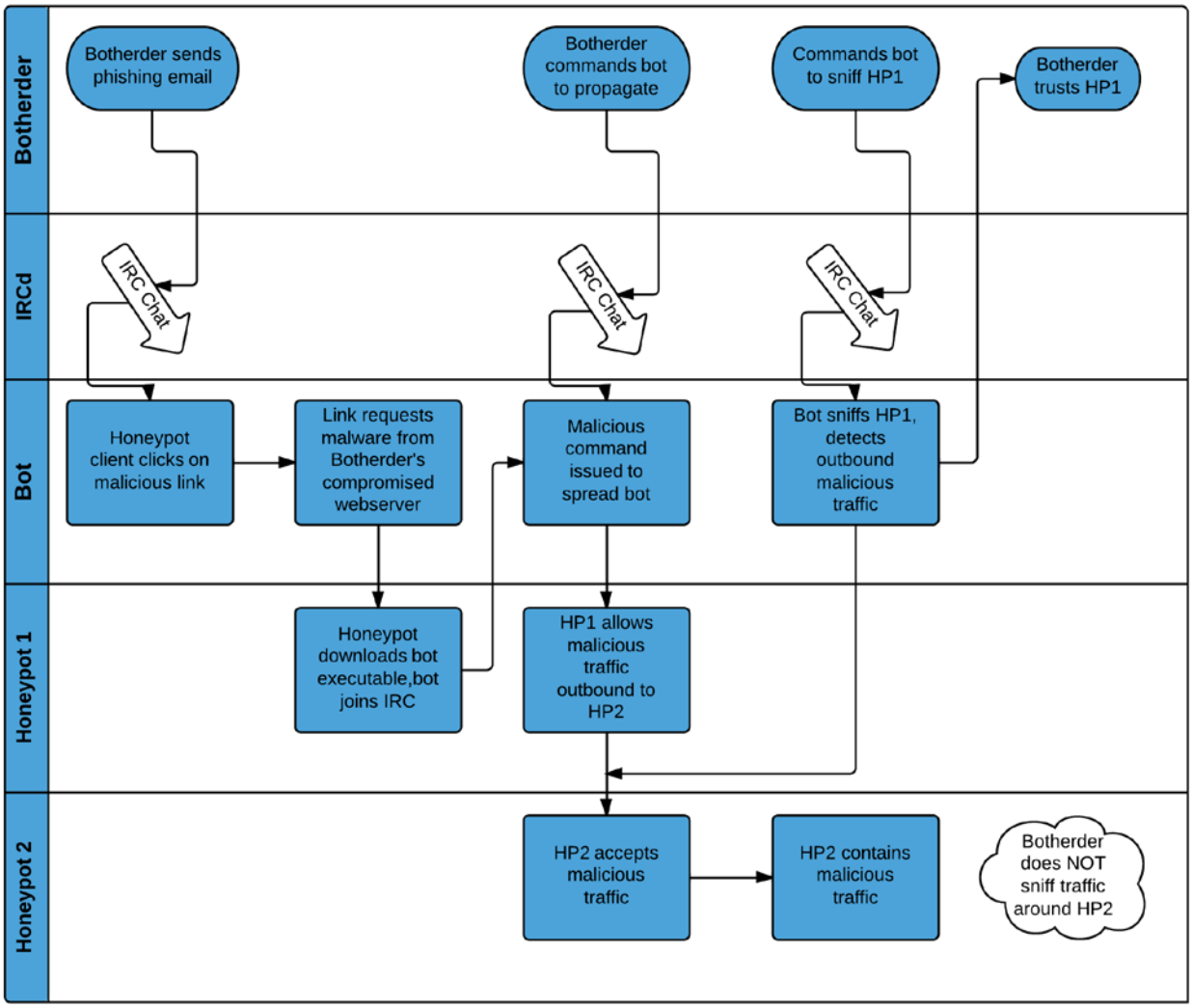
Honeypot01 is attacked by the botnet and there is no additional deception system in place to attempt to hide the honeypot from the botherder. The botherder will attempt to direct malicious traffic to **Honeypot02** and send instructions to **Honeypot01** directing it to pass the malicious traffic to another node in the subnet, designated as **Honeypot02**. Without any specific additional deception system in place, **Honeypot01** will implement the normal data containment many honeypot designs and not allow the malicious data to pass outbound to node **Honeypot02**. The botherder will note that **Honeypot01** has practiced data containment, or at least that something is altering her malicious traffic and so identifies **Honeypot01** as a honeypot. At this point the botherder has little invested in the particular network subnet (a single user clicked on a phishing email sent as spam) and the botherder will likely shift her attention to another network [WANG11]. The opportunity to perform further capture and data containment of, malicious traffic, as well as any opportunity to capture a zero day attack will be lost due to the compromise of the Honeypot. Because the botherder has very little invested so far in this subnet, she may choose to completely ignore it or mark it as a hostile environment [WATS12]. Even though many network security officers may count this as a win (after all, the botherder is not attacking any longer), the realistic purpose of a Honeynet and Honeynet research is to learn as much as possible about the ways and means of the enemy and share the lessons learned [SPIT06].

Use Case #2:

- **Honeypot01** will now respond to the botherder with a list of commands it can carry out
- **Honeypot01** acts on its botherder's commands and attempts to spread the malware
- Contact is made from **Honeypot01** to **Honeypot02** on the subnet
- **Honeypot02** will now receive the malicious traffic from **Honeypot01**
- Botherder sniffs traffic outgoing from **Honeypot01** to see if malicious traffic is being passed
- Botherder sees data is being passed from **Honeypot01** outbound and unaltered
    - There are 2 possibilities from this point:
        - ■ - Botherder may not suspect **Honeypot01** is a Honeypot and attack continues
            - More information about enemy and malware is gathered
        - ■ - Botherder may suspect **Honeypot1** of being a Honeypot
            - Botherder must sniff traffic outbound from **Honeypot02**
            - Botherder has already sniffed traffic outbound from **Honeypot01**
            - Botherder's work has increased factor of 2
            - Detection is more difficult

Discussion of Use Case #2

The second use case has the same preliminary steps as Use Case #1 for setup. The virtualized network topology is identical. The additional Honeypot, designated Honeypot02 is simply not used in Use Case #1.



Use Case #2 is an extension of the experiment described in [WANG11] with some additional alterations at the end, and a resulting hardening of the Honeynet by adding the manual work the botherder must carry out in order to monitor the bots in her botnet. The botnet is spread to the Honeynet and the botherder detects that the infected bot is obeying the commands from the botherder and passing along malicious traffic to a subsequent Honeypot/bot node (Honeypot 02). Therefore, the botherder did not detect Honeypot 01 as a Honeypot [TANG05].

Honeypot01 is attacked by the botnet and there is an additional deception system in place to attempt to hide the honeypot from the botherder. The botherder will attempt to direct malicious traffic to **Honeypot02** and send instructions to **Honeypot01** directing it to pass the malicious traffic to another node in the subnet, designated as **Honeypot02**. With the additional deception steps in place, **Honeypot01** will not implement the normal data containment of many honeypot designs and **Honeypot01** will allow the malicious data to pass outbound to node **Honeypot02**. The botherder will note that **Honeypot01** has not practiced data containment, and will observe that nothing is altering her malicious traffic and so identifies **Honeypot01** as not being a honeypot. At this point the botherder has

an invested interest in this particular network subnet and the botherder will likely remain focused on this subnet to see if her infected bot compromises further nodes in the subnet [TANG05]. The opportunity to perform further capture and data containment of, malicious traffic, as well as any opportunity to capture a zero day attack is not lost due to the deception of **Honeypot01**. Because the botherder has an investment of time and has been deceived into thinking she has gained at least one infected bot with the opportunity to use that bot as a foothold to grow her botnet with additional nodes, she may likely choose to continue working with this subnet and not identify it as a hostile network, but rather a friendly and vulnerable network [WATS12]. This must be considered a win because it represents an opportunity for a Honeynet to learn as much as possible about the ways and means of the enemy and share the lessons learned [SPIT06].

Details of Phatbot-Ghettobot

Phatbot-Ghettobot and most variants of Agobot use the packet sniffing library libpcap, which enables the bots to sniff network traffic near the intended victim machine to collect further information about the network, the target victim node, and any personal or sensitive information about the users that might be found [AGO00]. Agobot has stealth capabilities that enable hiding its presence by containing execution of processes and created files in a rootkit area [PROV07]. Reverse engineering Agobot variants is difficult because the bot command list includes functions to detect debuggers and virtual machines, and the bot encrypts its own configuration within the binary executable. None of these features or specifications concerning Phatbot-Ghettobot are factors in this research experiment due to the simulation.

The scanner capabilities that are included and occasionally updated in Phatbot-Ghettobot are extremely important to the criminal enterprise [AGO00]. The scanner is fundamental to the botnet's ability to find new vulnerabilities [PROV07]. The scanner module in variants of Agobot can be thought of as the botnet's automated version of a tool such as Metasploit with the purpose of finding vulnerabilities [PROV07]. This is the basic module of the malware that keeps up with the changes in botnet mitigation and network defense [WATS12]. Ago writes, in the documentation supplied, that purchasers of Agobot are not allowed to share or exchange scanners with other botherders and Ago expects this "honor among thieves" request to be followed by his criminal customers [AGO00]. There are several customized scanner classes included in the C++ source code that comes with Agobot and they reveal specific targets and also a generalized expansion of the botnet's overall target locating strategy [AGO00].

Operation of Phatbot-Ghettobot details

The bots connect to the server at a predefined port and join a specific channel [PROV07]. The attacker can issue commands in this channel, and these commands are sent via the C&C server to the individual bots, which then execute these commands. In the following example we can see an attacker issuing a command for all bots listening to attack a specific server with a DDoS attack. All bots responding send as many packets as possible to the victim, effectively prohibiting the normal service

[PROV07]. All bots in the botnet may not be in the same chat channel. In the case of larger sized botnets, they likely will not all be in the same IRC channel at the same time, so the botherder will either issue commands on several channels or decide how many bots are required for a given attack.

Hardening Algorithms

In this research, we provided a method to improve botnet mitigation through the use of honeynets that are hardened against honeynet aware botnets that use sensor technology to detect the passing of malicious traffic and code to propagate itself on a network. Our procedure will do this by increasing the amount of manual and time intensive sniffing and evaluation that the botherder must engage in to detect a Honeypot node from an infected bot. We feel it is further significant that this detection or evaluation only comes after the botnet has exposed itself via the infection operation and initial download of the bot executable.

We have further hardened the Honeynet against detection by allowing propagation of the botnet from one Honeypot to another Honeypot and shown how this increases the workload of the botherder by forcing her to do additional time consuming and labor intensive work for each and every bot infected node.

It is further our position that in future work, this principle can be used to continue the propagation from one Honeypot node to another Honeypot node and by some further refinement or additional algorithm concerning the subsequent selection of additional nodes, by some scheme involving an additional strategy such as randomization, categorization of attack, or another strategy, honeynets can be further hardened.

Evaluation

The system will be evaluated based on a field test. To substantiate the claim that the botnet is not detecting the presence of the honeynet, the additional steps required of the bot herder to carry out data checking on the sensor nodes will show hardening. The additional complication of checking additional sensors shown in the work of [TANG05] against worm attacks has been demonstrated in our work against botnet attacks.

Our evaluation of the data was to observe the results of sent communication from the botherder to the bots through the IRC channels. The observations were recorded in several different ways. We used a combination of monitoring software on the routers. The OpenBSD routers provided a simple way of monitoring the traffic on both the internal adapter and the external adapters on the network of each subnet and the simulated internet side of the routers. The OpenBSD routers provided access to the vmnet4, vmnet5, and vmnet 6 subnets as well as providing access to the outward facing adapter vmnet7, so that both sides of each router could be monitored. tcpdump was used both in streaming mode and also to capture packet data for later analysis and comparisons.

The experiments all took the following form of starting up the virtual machines in the following order:

- Start all 3 OpenBSD routers to establish the virtual network communication
    - Start tcpdump on each of the 3 routers, looking at the inward facing adapters
- Start the IRC Chat Server VM
    - Open DOS shell and start WinDump (listening to vmnet5)
    - Launch UnrealIRCd
    - Open UnrealIRCd logging window
- Start the Botherder machine, build or rebuild the botnet if needed
    - Open DOS shell and start WinDump (listening to vmnet4)
    - Login to IRCd using XChat (#Test channel)
    - Launch Phatbot-Ghettobot
- Start the XP Honeypot01, XP Honeypot02 VMs
    - Open DOS shell and start WinDump (listening to vmnet6)
    - Login to IRCd using XChat (#Test channel)
- Start Kali Linux Monitor VM
    - Open shell, start tcpdump (listening to vmnet6)
        - ome runs are live dump to shell
        - ome runs to capture data to pcap files for analysis later
        - ogin to IRCd using XChat (#Test channel)
    - Open Wireshark and either stream data, or read captured pcaps
- [OPTIONAL] use MacBook Pro to look at the “Internet” side of routers
    - Open shell, run tcpdump (listening to vmnet7)
    - Run Wireshark to analyze vmnet7

Criteria 1

The first criteria we used to prove hardening is simply the live streaming from the experiment that shows communication from the botherder to the infected bot. The botherder is able to sniff the traffic around the infected bot and see that commands from the command reference being issued to the bot are not being carried out in the case of propagating malicious traffic to the subsequent node on the network. The screen shots show that the Honeypot is containing the malicious traffic. Just as malicious traffic communication captured in pcap data files will flag anti-virus, even though no live virus or malware is operational, the same presence can be taken as proof that the malicious traffic is not being passed out of the node to the surrounding subnet.

Criteria 2

The next criteria we used to prove hardening is to show that the malicious traffic contained in the pcap data as well as the live streaming data is being passed along from the first Honeypot node to possible subsequent nodes in the subnet. Showing that the pcap data coming out of the honeypot node and being directed elsewhere on the subnet is proof that the honeynet is hardened against botnets

tracking traffic in this method [TANG05]. Screen shots and pcap packet data have this communication in them as well.

Conclusion and Future Work

We think that strong future work could be done in the area of further developing the proposed 3rd Use Case. This might include developing a propagation algorithm to go beyond the "double honeypot" strategy, to possibly a 3rd, 4th, and so on Honeypot design [TANG05]. Certainly, with this kind of an increase in the amount of manual work created for the botherder, she will be forced into the position where she will have to stop the manual monitoring at some reasonable point. If the process of propagation could be automated to a simulated machine that represents multiple IP addresses on a subnet, such as Honeyd, then the authors feel that would be a very worthwhile project for future expansion.

Namely, any strategy to propagate the malicious code to another Honeypot based on some intelligent decision would be worthwhile for future work. The scheme could be a strategy to select a Honeypot design according to the specific attack type of the malware, or the strategy could be something based on an AI algorithm approach, perhaps using an expert system, or a classification algorithm so that a response can be custom tailored to the attack type. To summarize, we feel that the following areas hold strong interest in the security community:

a. Random subsequent node selection
b. Context-aware propagation
c. Other possible strategies

Acknowledgement

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

National Science Foundation, Scholarship for Service, Cybercorp

Dr. Sam Chung, University of Washington, Tacoma, Institute of Technology

Dr. Barbara Endicott-Popovsky, University of Washington, Information School

David Dittrich, University of Washington, Applied Physics Laboratory

Michael Hamilton, CISO, City of Seattle, Washington

Michael P. Schweiger, MSCSS Candidate, University of Washington, Tacoma

The Honeynet Project, Pacific Northwest Chapter

References

[AGO00] Agobot3 Documentation and command reference, Ago (alias Wonk). 2000.

[BAIL12] Botnets.org, Michael Bailey, Jake Czyz, Dave Dittrich, Michael Hamilton, Manish Karir, Available: http://botnets.org

[BRIN12] M. Bringer, C. Chelmecki, and H. Fujinoki, "A survey: Recent advances and future trends in honeypot research," *International Journal*, vol. 4, 2012.

[GU08] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," in *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS08)*. San Diego, CA, 2008.

[HOLZ05] T. Holz and F. Raynal, "Detecting honeypots and other suspicious environments," in *Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC*. IEEE, 2005, pp. 29–36.

[KIM10] W. Kim, O. Jeong, C. Kim, and J. So, "On botnets," in *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services*. ACM, 2010, pp. 5–10.

[LEDE09] T. W. Felix Leder, "Know your enemy: Containing conficker: To tame a malware," web, April 2009. [Online]. Available: http://www.honeynet.org/ files/KYE- Conficker.pdf

[OLLM09] Ollmann, Gunter. "Botnet communication topologies." Damballa white paper (2009);

[PRIS11] PRISEM, Public Regional Information Security Event Management. Available: https://portal.securityondemand.com/Login.aspx

[PROV07] N. Provos and T. Holz, *Virtual honeypots: from botnet tracking to intrusion detection*. Addison-Wesley Professional, 2007.

[SCHN01] B. Schneier, *Secrets and Lies: Digital Security in a Networked World*, kindle edition ed. John Wiley and Sons, March 2001.

[SPIT03] L. Spitzner, "Know your enemy: Defining virtual honeynets," web, January 2003. [Online]. Available: http://old.honeynet.org/papers/virtual/

[SPIT05] L. Spitzner, "Know your enemy: Genii honeynets, easier to deploy, harder to detect, safer to maintain," web, May 2005. [Online]. Available: http://old. honeynet.org/papers/gen2/

[SPIT06] L. Spitzner, “Know your enemy: Honeynets, what a honeynet is, its value, overview of how it works, and risk/issues involved,” web, May 2006. [Online]. Available: http://old.honeynet.org/papers/honeynet/

[STAC13] Stackoverflow.com 2013 [Online]. Available: http://stackoverflow.com/questions/186207/visual-studio-6-processor-pack-compatibility

[TANG05] Y. Tang and S. Chen, “Defending against internet worms: A signature-based approach,” in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 2. IEEE, 2005, pp. 1384–1394.

[TANN12] Tanenbaum, Andrew S; Wetherall, David J. (2012-01-07). Computer Networks (5th Edition). Prentice Hall. Kindle Edition.

[WANG10] P. Wang, L. Wu, R. Cunningham, and C. Zou, “Honeypot detection in advanced botnet attacks,” *International Journal of Information and Computer Security*, vol. 4, no. 1, pp. 30–51, 2010.

[WANG12] W. Wang, J. Bickford, I. Murynets, R. Subbaraman, A. Forte, and G. Singaraju, “Catching the wily hacker: A multilayer deception system,” in *Sarnoff Symposium (SARNOFF), 2012 35th IEEE*. IEEE, 2012, pp. 1–6.

[WATS08] D. Watson, “The honeynet project gdh - global distributed honeynet,” web, January 2008. [Online]. Available: http://old.honeynet.org/speaking/PacSec07 David Watson Global Distributed Honeynet.pdf

[WATS12] D. Watson, “Hands on with the honeywall and virtual honeynets,” web, March 2012. [Online]. Available: http://www.honeynet.org

Appendix A

Technical problems encountered in setting up the experiment

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

The network infrastructure of the network itself provided a challenge. Originally conceived to be deployed in a cloud environment that was going to be provided by UW Bothell, that resource fell through and we had to reposition to simply setting up the complete virtualization environment on a single laptop computer. WE could have used more than one physical computer and separate routers, etc., but combining all the VMs on one machine made the early stages of the experiment easier for me in terms of portability. As the experiment grew in the size and number of VMs we were using, there was no single point where there was a compelling reason to move to another deployment, so it simply stayed on a single laptop. Once I gained experience working with the VMs and the virtual environment as a whole, scaling back the resources allocated to the individual machines and so on, and I became a better judge of what was required in terms of resources for the individual nodes and their functions, this issue was resolved. WE now have a system that nicely runs all the virtualized nodes simultaneously on a MacBook Pro with an i7 quad core simulating 8 cores and has 16 GB RAM installed.

We began the project using VirtualBox for the virtualization software, and then switched to VMware Fusion Professional for the completion of the project. The increase in performance of VMware compared to the use of VirtualBox was remarkable, but it only became apparent what the performance capabilities of the two systems were after we had tried and failed to run all the nodes of the system together, which we had not solidified early on in the project. This performance evaluation of the two software packages was also a result of the gradual refinement of the individual requirements that each node in the system needed in terms of the resources described previously. So, we only learned what worked better (VMware v. VirtualBox) after we learned what we needed (smaller, more well thought out resource allocation to individual nodes). We still like VirtualBox for simple, individual setups and instructional labs. It is freely available and is a nice introductory tool to virtualization technology.

VirtualBox Problems encountered, switching to VMware (Fusion)

The use of Oracle VirtualBox at the outset of the configuration of the lab and experiments, has proven to be a huge time consumer due to a relatively few hiccups in the software. Mid-project, the switch was made to VMware Fusion and the remainder of the project went far more smoothly, implementation-wise, than the first part that relied on VirtualBox.

VirtualBox is a free software distribution from Oracle that works on both Mac and Windows platforms. VirtualBox is very popular among the hacking and free software crowd due to its free availability and the relative positive performance enhancements that it offers.

The problems encountered using VirtualBox were all but eliminated by switching to VMware Fusion (for Mac hosts). This change was made originally in order to facilitate the completion of the project because of unreliability of the machine images in VirtualBox, but a side benefit which was not a trivial improvement was the increase in performance of VMware emulated machines. The software just simply performs better and faster in all aspects. The boot times are reduced and the loading of images and installation and updating of packages is greatly improved. Even though the VirtualBox machines are highly configurable and the number of different states gives a virtualization researcher much to play with, the VMware hosted machines simply outperform VirtualBox in every way and at every point to point comparison.

Boot times for virtualization

Boot times for virtual machine images are a factor in my environment, even though that may seem surprising. When research with virtualization is being used more extensively, the boot times of the individual machine images can accumulate to make the project's individual experiments take longer and longer run times. Eventually, this performance block is significantly impacting a project.

Running time for computationally intensive tasks

VMware is an alternative for virtualization that has become more or less the standard of the industry for The Honeynet Project and many other security users where research is often focused on virtualization implementations. In our research, the basic use, reliability, and robustness of the images made the switch to VMware one that had a significant impact on the accomplishment of the research. As with any new technology, or change that is introduced once a project has already been started, there was an initial period where the change to VMware slowed everything to a halt and necessitated some aspects of the project to be started over again, but once this initial cost was absorbed, the gains realized by the change were well worth the temporary setbacks caused by the switch. Some of the immediate improvements were a remarkably noticeable improvement in the run time performance of the machine images. Without even performing any kind of measured metrics, the responsiveness, boot times, more efficient (i.e., smaller) size of the executing footprint running in memory are all improvements. VirtualBox processes took much more space in memory to run on the host machine than the correlating VMware images. This savings in memory has an overall compounded effect on the lab environment.

Honeypot Implementation Problems

Some discussion of why we chose a plain vanilla Windows XP Professional honeypot as our Honeypot "implementation". Technically, any machine placed on a network with no production purpose and placed there for the reason of deception is a Honeypot. There are no false positives and any traffic to and from that node is suspicious. Our Windows XP honeypots were scaled back from other well known honeypot designs that we started with.

We attempted to use Cuckoo. When we were at the point in the project where we believed we would have access to cloud resources, Cuckoo seemed like a good choice for the honeypot design. We had access to other research that was utilizing already utilizing Cuckoo and it seemed a natural choice for our Honeypot design choice. However, when we discovered we would not have access to a cloud environment, and we were scaling back the project to be deployable on a laptop, the problems with Cuckoo became a block for our progress. Using a bare metal hypervisor was decided to be not convenient to work with for a self-contained laboratory experiment setup. Cuckoo is designed to be used in one of 2 basic ways, and operate either on a hypervisor with (often) a Windows XP victim machine running in a VM and reporting back to the Cuckoo host (which is also hosting the Windows machine), or doing the same thing on a native platform (such as Debian). We attempted initially to use Cuckoo as our Honeypot implementation choice, but abandoned it because of the problems we had getting it to run in the virtualized environment we had in VMware. With the Cuckoo host running in VMware and that Cuckoo host attempting to host a Windows XP victim machine, we had no success launching the Python process from the host to do the analysis on the malware.

Somewhere in the complexities of attempting to configure this arrangement, we decided that not only did we not know for certain if what we were trying to do was even possible (running the Cuckoo host on something other than a native platform or bare metal hypervisor) but we also decided that the scope of the project just didn't need a specific Honeypot implementation at all.

Several other Honeypot implementations were considered and discarded before settling on the plain Windows XP nodes. These nodes, as it turned out, while much more vulnerable without any Windows updates, had to have the Service Pack 2 applied in order to facilitate interacting with them even minimally (namely to install WinDump, the Windows port of tcpdump). The stock IE Browser would not run without crashing when attempting to go to anything but the most plain html pages, and installing either Chrome or Firefox both required newer DLLs that in turn required the SP2.

Building the botnet code

The variant of Agobot which we had access to, thanks to David Dittrich at the Applied Physics Laboratory at the University of Washington, is designated in the documentation as Agobot 3.0.1.0. There are detailed build instructions included with the software which have options for Visual C++, Borland C++, MingW32 and GCC. Agobot is designed to primarily attack vulnerabilities found in Windows machines and because this version was from circa the 2000 - 2004 timeframe, we could assume that Windows XP was a prime target.

Finding the correct software from the time period proved more difficult than we had anticipated. We did not realize that using an older compiler would be necessary, and others in the Honeynet Project reported that in fact, it was not necessary to use older versions of software. However, the distributed software of a botnet does not have the integrity of a commercial product and we had no way of knowing if the distribution that we were working with was the same as those that other HP members were reporting success with.

We had a distribution CD with over 30 variants of Agobot 3 to work from. Some were corrupted, and most came with GUI utility programs to configure the botnet which were Trojan'ed. We did not

have time to forensically determine what the source of these infections were, which might be an interesting item to follow up on some day. After having several problems compiling some of the variants of Phatbot-stoney, we finally chose Phatbot-Ghettobot because it was a very “clean” example from the collection of malware we had. The word clean here is used in a relative sense, and that is to say that Phatbot-Ghettobot gave us only what we had come to consider the “normal” amount of compile issues and Trojan infections in the included ConfigGUI program. We chose to build Phatbot-Ghettobot with Visual C++ 6. The documentation directed us to add the Platform SDK to the include path, the library, and the compile path in the Visual C++ project file as the top entry. The Platform SDK we obtained was from the same MSDN library subscription from 2001 that we obtained Visual C++ 6, and VC++ 6 Service Pack 5. The documentation also mentioned a VC Processor Pack which we never were able to locate. Stack overflow discussions led us to determine that the Processor Pack, which is not widely known of, was included as an add on to the Service Pack 5 for VC++ 6, and then scheduled for inclusion in Service Pack 6, but removed from Service Pack 6 distribution just prior to the release of Service Pack 6.

The documentation we found specified that there is an interaction observed by some developers between the Processor Pack available for Visual Studio 6 and Service Pack 5 and Service Pack 6. The Visual C++ Processor Pack (VCPP) was removed from Service Pack 6. If you have the VCPP installed on top of SP5, then installing SP6 will remove it from your machine. There were finally, suggestions that If you wish to continue using the VCPP, you need to either stay with SP5 or migrate to Visual Studio 2002 or 2003. The migration forward was recommended, but, for our research, VS 2002 was already into the Microsoft .NET CLR (Common Language Runtime) and so was not an option that we wanted to get involved with. Our preliminary builds of the botnets using Visual Studio 2008 and 2010 were a disaster that we quickly abandoned in favor of pursuing the legacy environment tools.

As far as we could determine, the Processor Pack allowed developers to target individual CPU architectures that were competing in the time frame of 2000 - 2001. Since we had a clean compile and build, and the botnet appeared to operate correctly, we did not follow up on the specific issues of what, if any, problems this might have caused. The issue may have been resolved by XP, since Windows 2000 was more likely the current popular version of Windows at the time the documentation was first written.

This issue of the legacy software was interesting to me as a former developer and current network security student for 2 reasons. First was the obvious impact of how it blocked the progress of the project by coupling the use of certain specific software to the use of other specific software, some of which was more difficult (or impossible) to obtain. So, as exact as the virtualized environment can be create a laboratory environment where the scientific method can be applied, it’s still impacted by the messy real world. The second reason originates from more of an academic perspective. When working with malware and virtualization, you might, or likely will be, required to put the malware into some historical context, and the extent to which that context impacts the outcome of what you are doing has many, many, variables, some of which cannot be accounted for and tied back to specifics about the environment that you are attempting to recreate. Aside from learning about botnets, networking,

subnetting, virtualization, and network traffic monitoring, this lesson of historical context of software and hardware configurations was perhaps the most interesting outcome of the Capstone Project for me.

Final thoughts

I would like to issue a very special thank you to all the individuals who went “above and beyond” to help us obtain the needed software to pull this project together. Dave Dittrich from the UW APL gave us about 30 different variants of Agobot, and lent his expertise in the subject area. Stephen Rondeau from the Institute’s Computer Lab dug into his old subscriptions of MSDN to find the critical build tools when other sources dried up. I would like to thank Mike Schweiger from the Institute and SFS Program for helping with ideas on how to configure the network. I would like to thank all the members of The Honeynet Project for the inspiration to do research in such a fascinating area of study. I would like to thank Mike Hamilton from the City of Seattle for giving me a place to learn and a playground to play in and access to really cool toys to use and lots of bad guys to track. I would like to thank Felix Leder, Dave Dittrich, Ron Dodge, Mike Schweiger, and Guillaume Arcas of The Honeynet Project for really coming through to help me get the software and help I needed. When I asked Dave Dittrich for help and he said to put out an email to the all@ list, I was overwhelmed with the response I got from people I really barely knew, except in our common goal of network security. It feels really special to belong to an organization that has such a strong sense of community spread out across the globe. Finally, I would like to thank Dr. Sam Chung and Dr. Barbara Endicott-Popovsky, who believed that I could go back to school at my age and do this thing.