Honeypot-Aware Advanced Botnet Construction and Maintenance

Cliff C. Zou Ryan Cunningham
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816-2362
{czou,rcunning}@cs.ucf.edu

Abstract

Because "botnets" can be used for illicit financial gain, they have become quite popular in recent Internet attacks. "Honeypots" have been successfully deployed in many defense systems. Thus, attackers constructing and maintaining botnets will be forced to find ways to avoid honeypot traps. In this paper, we present a hardware and software independent honeypot detection methodology based on the following assumption: security professionals deploying honeypots have liability constraints such that they cannot allow their honeypots to participate in real (or too many real) attacks. Based on this assumption, attackers can detect honeypots in their botnet by checking whether the compromised machines in the botnet can successfully send out unmodified malicious traffic to attackers' sensors or whether the bot controller in their botnet can successfully relay potential attack commands. In addition, we present a novel "two-stage reconnaissance" worm that can automatically construct a peer-to-peer structured botnet and detect and remove infected honeypots during its propagation stage. Finally, we discuss some guidelines for defending against the general honeypot-aware attacks.

1 Introduction

In the last ten years, Internet users have been attacked unremittingly by widespread email viruses and Internet-scanning worms. Some of the more devastating email viruses include Melissa in 1999, Love Letter in 2000, W32/Sircam in 2001, and MyDoom, Netsky and Bagle in 2004, etc. Similiarly damaging internet-scanning worms include Code Red in 2001, Slammer, Blaster in 2003, Witty and Sassar in 2004 [6]. Strangely, we have not seen a major virus or worm outbreak since the Sassar worm incident in May 2004. This is probably not because the Internet is much more secure, but more likely because attackers no longer focus on infecting a large number of computers just to attract media attention. Instead, their attention has shifted to compromising and controlling victim computers, an attack scheme which provides more potential for personal profit.

This new and lucrative attack trend has produced a large number of botnets in the current Internet. A "*botnet*" is a network of computers that are compromised and controlled by an attacker [21]. Each computer is infected with a malicious program called a "bot", which actively communicates with other bots in the botnet or with several "*bot controllers*" to receive commands from the botnet owner. Attackers maintain complete control of their botnets, and can conduct Distributed Denial-of-Service (DDoS) attacks, email spamming, keylogging, abusing online advertisements, spreading new malware, etc [21].

Turning our focus now to recent trends in computer security, honeypot is a general and effective attack detection technique. A "*honeypot*" is a special constructed computer or network trap designed to attract and detect malicious attacks. In recent years, honeypots have become popular, and security researchers have generated many successful honeypot-based attack analysis and detection systems (such as [9, 22, 13, 3, 29, 17].) As more people begin to use honeypots in monitoring and defense systems, attackers constructing and maintaining botnets will try to find ways to avoid honeypot traps.

In this paper, we show how attackers might attempt to remove honeypot traps when constructing and maintaining botnets. This knowledge is useful for security professionals wanting to prepare for more advanced botnet attacks. Unlike hardware or software specific honeypot detection methods [24, 8, 2], the honeypot detection methodology presented here is based on a general principle that is hardware and software independent: honeypot owners have liability constraints such that they cannot allow their honeypots to send out real attacks (or send out too many real attacks.) As laws are developed to combat cybercrime in the coming years, security experts deploying honeypots will probably incur more liability than they have today, because they knowingly allow their honeypots to be compromised by attackers. If they fail to perform due dilligence by securing their honeypot against damaging other machines, they will be considered negligent. To our knowledge, this is the first paper to systematically study honeypot detection based on such a general methodology.

Based on this principle, attackers can command their botnets to actively send out malicious traffic (or *faked* malicious traffic) to one or several other compromised computers. These computers behave as "sensors". Attackers can then determine whether a bot is actually a honeypot or a verified vulnerable victim machine based on whether or not the sensors observe the complete and correct attack traffic transmitted from this bot.

To detect a hijacked bot controller in a hierarchical botnet, attackers can issue a test command via the bot controller under inspection that causes botnet members to send trivial traffic to the attackers' "sensors". The hijacked controller can then easily be detected if the command is not carried out or is not carried out correctly. In addition, attackers can detect bot controllers hijacked via DNS redirection [10] by checking whether the IP addresses resolved by DNS queries match the real IP addresses of their bot controllers.

Compared with the currently popular hierarchical botnets, a P2P botnet is much harder for the security community to monitor and eliminate. In this paper, we present a simple but effective P2P botnet construction technique via a novel "*two-stage reconnaissance*" Internet worm attack, which is also capable of detecting and removing infected honeypots during the worm propagation stage.

The honeypot avoidance technique presented in this paper is not specific to botnets but applicable for detection of general honeypots. It can be conducted after a remote honeypot is compromised. Attackers can use it when they manually compromise remote computers. In the area of large-scale automatic attacks, it is not effective to use this methodology in a traditional worm, since a honeypot has already obtained the worm binary code once it is compromised. On the other hand, the actions taken after the machine is compromised are more important to a botnet attacker than the compromising bot code itself. These actions cannot be predicted or obtained just based on the initial bot code captured by a honeypot. For this reason, we believe botnets will probably be the first major battlefield between attackers and security professionals using honeypots. Because of this, we decided to focus on botnets in this paper rather than other types of attacks.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the honeypot detection methods for current hierarchical botnets. Section 4 introduces an advanced honeypot-aware worm that can construct a P2P botnet. In Section 5 we discuss several guidelines to counterattack honeypot-aware attacks from the security professional's perspective. In the final Section 6 we summarize our conclusions.

2 Related Work

Botnet is a new trend in Internet attacks. In 2003, Puri from SANS Institute [23] presented an overview of bots and botnets; McCarty [18] discussed how to use a honeynet to monitor botnets. Currently, there are two techniques to monitor botnet activities. The first technique is to allow honeypots or honeynets to be compromised and join in a botnet [7, 12, 21]. Behaving as normal "bots" in the botnet, these honeypot spies provide valuable information of the monitored botnet activities. With the help from Dynamic DNS service providers, the second technique is to hijack bot controllers in botnets to monitor the command and control communications in botnets [10]. This was accomplished by redirecting the bot controllers' DNS mapping to a botnet monitor.

Honeypots and honeynets are effective detection and defense techniques, and hence there has been much recent research in this area. Provos [22] presented "honeyd," a honeypot software package that makes large-scale honeynet monitoring possible. Dagon *et al.* [9] presented the "honeyStat" system to use coordinated honeypots to detect worm infections in local networks. Jiang and Xu [13] presented a virtual honeynet system that has a distributed presence and centralized operation. Bailey *et al.* [4] presented a globally distributed, hybrid, honeypot-based monitoring architecture which deploys low-interaction honeypots as the frontend content filters and high-interaction honeypots to capture detailed attack traffic. Vrable *et al.* [29] presented several effective methods to design large-scale honeynet systems capable of obtaining high-fidelity attack data, which they called "Potemkin". Tang and Chen [27] presented a novel "double-honeypot" detection system to effectively detect Internet worm attacks. Anagnostakis *et al.* [3] presented a way to use a "shadow honeypot" to conduct real-time host-based attack detection and defense.

There has been some research in discovering and concealing honeypots. Provos [22] discussed how to vividly simulate the routing topology and services of a virtual network by tailoring honeyd's response. GenII honeynets [20] allow a limited number of packets to be sent out from an infected honeynet. From the attacker's perspective, some hardware or software specific means have always been available to detect infected honeypots (e.g. by detecting VMware or another emulated virtual environment [24, 8], or by detecting the honeypot program's faulty responses [2].) However, there has been no systematic research on honeypot detection based on a general methodology.

Krawetz [15] introduced the commercial anti-honeypot spamming tool, "Send-Safe's Honeypot Hunter". On a spammer's computer, the tool is used to detect honeypot open proxies by testing whether the remote open proxy can send email back to the spammer. This anti-honeypot tool uses the similar approach presented in this paper. It can be treated as a special realization of the methodology presented here, but it is only effective for detecting open proxy honeypots.

Bethencourt *et al.* [5] presented a method for attackers to use intelligent probings to detect the location of Internet security sensors (including honeypots) based on their public report statistics. In this paper, we present a general honeypot detection approach that does not require a honeypot to publish its monitored statistics. Lance Spitzner [25] addressed the basic legal issues of honeypots, explaining that honeypots have potential problems in terms of privacy and liability. In this paper, we only consider honeypot liability.

3 Hierarchically Structured Honeypot-Aware Botnets

3.1 Hierarchical botnets introduction

Most botnets currently known in the Internet are controlled by attackers via a hierarchical network structure. Fig. 1 shows the basic network structure of a typical botnet (for simplicity, we only show a botnet with two bot controllers). All compromised computers in a botnet are called "*bots*". They frequently attempt to connect with one or several "*bot controllers*" to retrieve commands from the botnet attacker for further actions. These commands are usually issued from another compromised computer (to hide attacker's real identity) to all bot controllers. To prevent defenders from shutting down the command and control channel, attackers usually use multiple redundant bot controllers in their botnets.

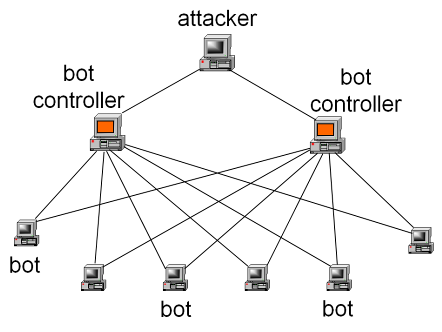

Figure 1. Illustration of a hierarchical botnet

To set up bot controllers flexibly, attackers usually hard-code bot controllers' domain names rather than their IP addresses in all bots [10]. Attackers also try to keep their bot controllers mobile by using dynamic DNS (DDNS) [28], a resolution service that facilitates frequent updates and changes in machine location. Each time a bot controller machine is detected and shut down by its user, attackers can simply create another bot controller on a new compromised machine and update the DDNS entry to point to the new controller.

In the rest of this section, we introduce how attackers can thwart the two botnet trapping techniques presented in the the beginning of Section 2, respectively.

3.2 Detection of honeypot bots

First, we introduce a method to detect honeypots that are infected and acting as bots in a botnet. The general principle is to have an infected computer send out certain malicious or "faked" malicious traffic to one or several remote computers that are actually controlled by the botnet attacker. These remote computers behave as "sensors" for the attacker. If the sensors receive the "complete" and "correct" traffic from the infected host, then the host is considered "trusted" and is treated as a normal bot instead of a honeypot. Since honeypot administrators do not know which remote computers contacted are the attacker's sensors and which ones might be innocent computers, they cannot defend against this honeypot detection technique without incuring the risk of attacking innocent computers.

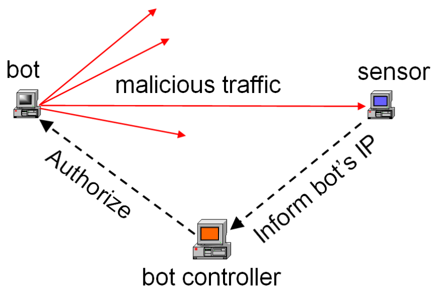

Figure 2. Illustration of the procedure in detecting honeypot bots in a hierarchical botnet

This honeypot detection procedure is illustrated in Fig. 2. A bot sends out malicious traffic to many targets, including the attacker's sensor. When the attacker's sensor receives the traffic and verifies the correctness of the traffic (ensuring that it was not modified by a honeypot), the sensor informs the bot controller of the bot's IP address. The bot controller then sends the authorization key to the checked bot so that the bot can join the botnet. To prevent the possibility of a single point of failure, an attacker could set up multiple sensors for this test.

This honeypot detection procedure can be performed on a newly infected computer before it is allowed to join a botnet. Such a botnet has a built-in authorization mechanism. The attacker (or the botnet controller) uploads the authorization key to the host and allows it joining into the botnet only after the host passes the honeypot detection test. In addition, attackers may perform the honeypot detection periodically on botnets to discover additional honeypot bots. This could be done whenever attackers renew their bots' authorization keys or encryption keys, or update the botnet software.

Next, we will introduce several illicit activities attackers might utilize to detect honeypot bots in their hierarchical botnets.

3.2.1 Detection through infection

When a computer is compromised and a bot program is installed, most bot programs will continuously try to infect other computers in the Internet. In this case, a honeypot must modify or block the outgoing malicious traffic to prevent infecting others. Based on this liability constraint imposed on honeypot security professionals, an attacker could let compromised computers send malicious infection traffic to the attacker's sensors.

Some honeypots, such as the GenII honeynets [20], have Network Intrusion Prevention System (NIPS) that can modify outbound malicious traffic to disable the exploit. To detect such honeypots, attackers' sensors need to verify that the traffic sent from bots are not altered (e.g., using MD5 signature).

It is also important that a newly compromised bot does not send malicious traffic to the sensors alone after the initial compromise. It must hide the honeypot checking procedure to prevent defenders from allowing the initial honeypot detection traffic going out. To hide the sensor's identity, a bot could put the sensors' IP addresses at a random point in the IP address list to be scanned. For a bot that infects via email, the sensors' email addresses could be put at a random point in the outgoing email address list. This procedure will delay the newly infected computer's participation in the botnet, but an attacker would be willing to incur this slight delay to secure their botnet, because botnets have long term use to the attacker.

This honeypot detection technique is difficult for honeypot defenders to deal with. Honeypot defenders cannot block or even modify the outgoing infection traffic. Without accurate binary code analysis, honeypot defenders will not be able to know which target IPs belong to the attacker's sensors. An attacker can make the code analysis even harder by obfusticating or encrypting sensors' IP addresses in the code. Furthermore, this honeypot detection technique is also natural to a botnet during its the construction time since a newly infected bot will try to infect others whether or not it is using this honeypot detection procedure.

3.2.2 Detection through other illicit activities

Based on our general honeypot detection principle, attackers can have their botnets send out other types of illicit traffic to sensors for honeypot detection. These illicit activities include:

Low rate port scanning. By hiding sensors' IP addresses in the port-scan IP address list, a bot can detect whether or not it is in a honeypot that limits outgoing connection requests. For example, GenII honeynet [20] limits the number of outbound connection rate.

Some normal computers are configured (e.g., installed a firwall, or a worm detection software such as [16]) to limit outgoing connection rate as well. To avoid mislabeling such computers as honeypots, and also to avoid possible detection by users, attackers should let their bots conduct a very low rate stealthy port-scan for honeypot detection.

Web requests. An attacker could also have a bot continuously request content from a web server that is actually compromised and controlled by the attacker. From the network administrator's perspective of one machine, this activity looks just like a normal user's web requests, however honeypot defenders will have hard time distinguishing whether or not this is an attack, because this bot could actually be conducting a real DDoS attack together with thousands of other computers in the same botnet. If this is a real DDoS attack but not blocked by honeypot defenders, they will be liable for possible damage caused by the attack.

Email spamming. An attacker could also have a bot send out spam email to one or several target email addresses owned by the attacker. These e-mail addresses behave as the honeypot detection sensors. Outgoing email spam, such as "phishing" email [11], could make honeypot security professionals liable for substantial financial losses if they reach real users.

3.3 Detection of hijacked bot controllers

Now we introduce techniques to detect hijacked bot controllers. With the help from Dynamic DNS providers, Dagon *et al.* [10] presented an effective botnet sinkhole that can change the domain name mapping of a detected bot controller to point to a monitoring machine. This way, the monitor receives connection requests from most (if not all) bots in the botnet. Conceptually speaking, the monitor becomes a hijacked bot controller, which is similar to a honeypot in term of functionality.

From an attacker's perspective, the botnet monitor is very dangerous, because security professionals can learn most of the IP addresses of bots in a botnet — the monitor owners can easily provide a "black-list" of these IP addresses to the security community or potential victims. For this reason, botnet attackers will do everything they can to eliminate a hijacked bot controller from their botnets. In this section, we present two different techniques that attackers might use to achieve this goal.

3.3.1 Bot controller DNS query check

When a bot controller is hijacked by the DNS redirection method presented in [10], the IP address of the bot controller returned by DNS query will not be the IP address of the real bot controller. Although bots in a botnet know the domain names instead of the actual IP addresses of bot controllers, the botnet owner can easily learn all the IP addresses of the botnet's controllers, because these computers are compromised by the attacker and are running the attacker's bot controlling program.

Therefore, a botnet attacker can keep an up-to-date DNS mapping table of all bot controllers. Using one compromised computer as a sensor, the attacker can have this sensor continuously send DNS queries to resolve the name and IP mapping of all bot controllers in the botnet and then compare the results with the real domain name mapping table. Besides the short time period right after the attacker changes the bot controller's IP address, this continuous DNS query procedure is always able to detect whether or not a hijacked bot controller is present in the botnet. If a hijacked controller is detected, the attacker can immediately use other bot controllers to issue a command to update the domain names in all bots, thus obviating further compromise from the hijacked controller.

3.3.2 Bot controller command channel check

The above DNS query check is an effective way to detect DNS redirection of bot controllers. However, it is possible for security defenders to conduct a more stealthy monitoring by actually capturing and monitoring a bot controller machine. In this case, the DNS query check will not work.

To detect such a physically hijacked bot controller, an attacker can use the same honeypot detection principle we described before. The botnet owner checks whether or not a bot controller passes the attacker's commands to bots. The monitor presented in [10] is called "sinkhole" because it does not pass any attacker's commands to bots. In fact, a hijacked bot controller puts a much more serious liability burden on security defenders than a normal compromised honeypot. If it passes an attacker's command to bots in a botnet, the defender could potentially be liable for attacks sent out by thousands of computers in the botnet. For this reason, security defenders do not dare to let a hijacked bot controller send out a single command. Even if the command seems harmless from previous experience, it is always possible the botnet software has been modified such that a previous known trivial command actually deletes files on all of the compromised computers or launches DDoS attacks against risky targets.

Based on this, an attacker can issue a trivial command to the bot controller under inspection without passing the command to other bot controllers. The trivial command orders a small number of bots to send a specific service request to the attacker's sensor (e.g., a compromised web server). Bots will not be exposed by this action since they simply send out some normal service requests. If the sensor does not receive the corresponding service requests, the attacker knows that the bot controller has been hijacked (or is at least not working as required).

4 P2P-Structured Honeypot-Aware Botnets

Cook *et al.* [7] discussed three different botnet communication topologies and their properties: centralized, peer-to-peer (P2P), and random. In a random topological botnet, a bot knows no more than one other bot [7]. Since such a botnet has extreme high latency in communication with no guarantee of delivery, we will not consider this topology in botnet study.

Most current botnets in the Internet use the hierarchical structure (or the centralized topology discussed in [7]) introduced in the previous Section 3.1. To increase the availability of the command&control channel in a hierarchical botnet, an attacker has to increase the number of bot controllers in the botnet. This will increase the financial cost of maintaining the botnet, since the attacker will need to purchase more Dynamic DNS domain names. In addition, the botnet is susceptible to bot controller hijacking, which exposes the identity of the entire botnet to security professionals, as was illustrated in [10].

On the other hand, a P2P botnet is much harder for security professionals to track. There is no centralized bot controller they can monitor. A honeypot bot in the botnet can only monitor a very small portion of the entire botnet. Attackers also do not need to spend money buying Dynamic DNS services. Therefore, we believe more P2P botnets will be created in the near future (if they do not already exist.)

4.1 Constructing P2P botnets based on "buddy list"

We present a simple but effective worm that can build a P2P botnet as it spreads. Basically, each worm-infected computer has a "*buddy list*" containing IP addresses of n other infected hosts. As the worm spreads, it gradually builds a P2P botnet structured by the buddy list (which could be described by a directed graph in which every node can pass botnet commands to n neighbors.)

At the beginning, the worm can spread through a small botnet that has $n + 1$ bots — each bot contains the IP addresses of the other n members in the botnet. Or, the worm could simply spread through several infected hosts where the other IP addresses in the buddy list are empty.

The P2P botnet buddy list can be built in the following way: When an infected host A infects a new victim B, its buddy list is passed to the victim. Host A chooses with a probability whether or not to replace one IP address in its own buddy list with host B's IP address. If host B has already been infected before, host B updates a part of its own buddy list with the new one sent from host A. This buddy list constructing procedure mixes the relationship of infected computers in a buddy list so security professionals cannot infer (or "traceback") the infection links when they capture some infected computers.

Besides the IP addresses, the buddy list can contain additional information to facilitate the propagation and maintenance of a botnet. For example, the buddy list on a host can contain the rough estimate of the bandwidth between this host and its neighboring bots. In this way, when the host needs to download the updated version of the bot program, it can select the host in the buddy list with the highest bandwidth from which to download the update.

4.2 Command and control in P2P botnets

Before introducing honeypot detection techniques, let us first discuss how to maintain the command and control in a P2P botnet constructed with the buddy list technique above.

To issue a command to a P2P botnet, the botnet attacker can inject the command to several bots in the botnet that he knows, (e.g. the several initial infected computers for used to spread the worm.) These bots will pass the commands to all bots listed in their buddy lists, and so on. In this way, commands will flood the botnet via the buddy-list topology. Since each bot sends the same command only once to hosts in its buddy list, this command flooding will stop naturally. The botnet attacker can increase the controllability and response speed of his botnet by increasing the size of the buddy list.

In a peer-to-peer network, some computers are not able to receive connection requests since they are behind Network Address Translation (NAT) boxes or firewalls; some computers have changeable IPs since they are using Dynamic Host Configuration Protocol (DHCP). For a P2P botnet, an attacker can implement many techniques to solve the above issues, as people have studied the communication issues in general P2P networking for a long time. For example, a bot behind a NAT can actively contact bots in its buddy list instead of waiting for a command. For another example, whenever its address changes, a bot with a DHCP-assigned IP address immediately informs all bots in its buddy list. These bots must put the new IP address in their buddy lists upon receiving such information. In this way, a DHCP-assigned bot can make sure that at least bots in its buddy list can pass the attacker's command to it.

Of course, to prevent a botnet from being hijacked by other hackers or security professionals, a botnet needs to implement authentication and encryption mechanisms in its communication.

The P2P botnet constructed as introduced above is easy for attackers to control and robust when facing monitoring and defense from security defenders. First, an attacker can easily learn how many zombie machines have been collected in the botnet and their IP addresses. The attacker can connect to several known infected computers, asking them to issue a command to let all bots sending a specific service request to the attacker's sensor (the similar method as presented in Section 3.3.2). On the other hand, security professionals cannot use this technique for monitoring, even if they know how to send such a command, due to their liability constraint. Second, an attacker can randomly choose any one or several bots to infill commands into the botnet — it is very hard for security defenders to cut of the control channel unless they hijack the botnet and take control of it by themselves. Such an active defense requires security professionals to issue commands to the botnet and update bot code on all (potentially hundreds or even thousands) compromised computers, which clearly puts a heavy liability burden on security professionals. Third, suppose security professionals remove many infected computers in a botnet. The attacker still has control over the remaining P2P botnet, even if the remaining botnet is broken into many separated smaller ones.

4.3 Two-stage reconnaissance worm to detect honeypots in constructing P2P botnets

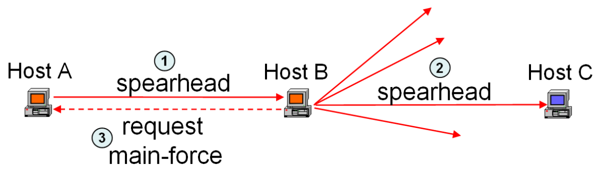

Figure 3. Illustration of the propagation procedure of a two-stage reconnaissance worm

We have introduced a means for attackers to construct a P2P botnet using a worm with buddy list. Now we introduce a way for the worm to detect and remove infected honeypots while propagating. To achieve this goal, the worm is designed to have two parts: the first part compromises a vulnerable computer and then decides whether this newly infected machine is a honeypot or not; the second part contains the major payload and also the authorization component allowing the infected host to join in the constructed P2P botnet. Due to the different roles in a worm propagation, we call the first part the "*spearhead*", the second part the "*main-force*" of the worm.

A simple way to verify whether a newly compromised host is a honeypot or not is to check whether or not the worm on it can infect other hosts in the Internet. Fig. 3 illustrates the propagation procedure of a two-stage reconnaissance worm in infecting host B and checking whether it is a honeypot or not. First, the vulnerable host B is infected by the spearhead of the worm, which contains the exploiting code and the buddy list. Second, the spearhead on host B keeps scanning the Internet to find targets (such as host C) to infect with the spearhead code. Third, after the spearhead on host B successfully compromises m hosts (include both vulnerable and already-infected ones), it tries to download the main-force of the worm from any host in its buddy list that has the main-force component. The main-force code lets the worm join the constructed botnet via the authorization key contained in the main-force.

By deploying such a two-stage reconnaissance worm, the botnet is constructed with a certain time delay as the worm spreads. This means that some infected hosts will not be able to join in the botnet, since they could be cleaned before the main-force is downloaded. However, this does not affect the botnet, since it makes no difference to the attacker whether or not the botnet contains bots that will be quickly removed by security defenders.

In fact, it is not a new idea to spread a worm in two stages. Blaster worm and Sasser worm used a basic FTP service to transfer the main code of the worm after compromising a remote vulnerable host [6]. The two-stage reconnaissance worm presented here can be treated as an advanced two-stage worm by adding the honeypot detection functionality into the first-stage exploit code.

4.4 Advanced two-stage reconnaissance worm in response to "double honeypot" defense

Tang and Chen [27] presented a "double-honeypot" system where all the outgoing traffic from the first honeypot is redirected to a dual honeypot. If the dual honeypot is set up to emulate a remote vulnerable host, then the dual honeypot can fool the above two-stage reconnaissance worm into believing that the first honeypot is a real infected host.

This vulnerability of the two-stage reconnaissance worm is due to: (1) a spearhead makes the decision by itself whether a remote host is infected or not; and (2) a dual honeypot can emulate any outside remote host with arbitrary IP address. To detect such a dual-honeypot defense system, attackers can design an even more advanced two-stage re-

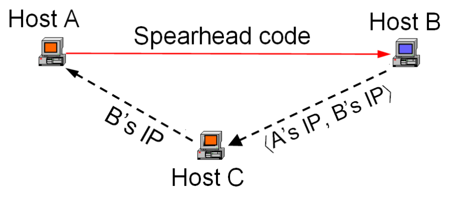

Figure 4. The procedure in counterattacking dual-honeypot defense by an advanced two-stage reconnaissance worm (host A is the bot under inspection; host C is in A's buddy list that has the main-force code)

connaissance worm that propagates as following (illustrated in Fig. 4):

- When an infected host A finds and infects a target host B, it records down host B's IP address. Host A continues finding and infecting others in this way.
- Host B sets up a TCP connection with every host in host A's buddy list, telling them the tuple ⟨A's IP, B's IP⟩, which means "host A has sent the correct exploiting code to host B". Suppose host C is one of the hosts in A's buddy list.
- When host C obtains the main-force of the worm, it informs host A of host B's IP address. This report can be done only by hosts having the main-force code (e.g., the authorization key is in the main-force code).
- If host A finds B's IP address in its recorded infection IP list, it knows that it has really infected another host. When host A has successfully infected m hosts, it passes the honeypot detection procedure and then tries to download the main-force code from any host in its buddy list that has the complete code.

This reconnaissance worm will not be fooled by a dual-honeypot system because:

(1) The spearhead in host A chooses IP addresses to scan by itself, thus the real IP address of a dual honeypot has a negligible probability to actually be scanned by host A without IP address redirection;

(2) When host B informs hosts in host A's buddy list of the address tuple ⟨A's IP, B's IP⟩, it cannot cheat about its IP address due to the TCP connection;

(3) Only an infected host that is not a honeypot will have the main-force code, and hence, host A can trust that host C is not a honeypot (without this trusted host C, security defenders could use honeypots for all three hosts in Fig. 4 to fool the spearhead in host A).

In summary, the advanced reconnaissance worm works because host B cannot lie about its IP address and host C is trusted.

Security defenders in a local network could use a honeynet to cover a large number of local IP addresses. To prevent the spearhead in host A from actually scanning and infecting a local IP address occupied by a honeypot (especially if the worm deploys the "local preference" scans [32]), the worm can conduct the infection report shown in Fig. 4 for global infection only, i.e., host B is required to be far away from host A.

4.5 Modeling of the constructed P2P botnet growth

In this section, we present an analytical model for modeling the growth of the botnet as the two-stage reconnaissance worm spreads. As explained above, an infected host joins in the botnet only when it has downloaded and executed the main-force of the worm. Thus the botnet grows a step behind the propagation of the worm's spearhead.

The modeling presented here tries to show that a two-stage worm will not slow down botnet construction, even though it adds a delay. The modeling results, as presented below, show that all infected computers (not including detected honeypots) will join in the botnet in the end, and the machines will join the botnet shortly after the initial infection.

The spearhead of a two-stage reconnaissance worm propagates in way similar to that of a traditional worm, thus it can be modeled by the popular epidemic model as used in [19, 26, 30], etc. Since worm modeling is not the focus of this paper, we present a simple model, where the two-stage reconnaissance worm uniformly scans the IP space. Papers such as [14, 32] have presented modeling of local preference scanning, bandwidth-limited spread, and other worm scanning strategies. The model presented here can be extended based on the models in those papers for various non-uniform scanning strategies.

Let $I(t)$ denote the total number of infected hosts at time t — whether a host is infected only by the spearhead or by the full worm; $\bar{I}(t)$ denotes the number of infected hosts that have joined in the botnet by time t, i.e., they have the main-force of the worm. The propagation of the spearhead can be modeled as [26, 30, 32]:

$$\frac{dI(t)}{dt} = \frac{\eta}{\Omega} I(t)[N - I(t)] \tag{1}$$

where N is the total vulnerable population, η is the worm's average scan rate per infected host, Ω is the size of the IP space scanned by the worm.

First, we derive the propagation model of $\bar{I}(t)$ via "infinitesimal analysis" for the two-stage reconnaissance worm with $m = 1$, i.e., a spearhead-infected host downloads the main-force right after it sends out the spearhead and compromises another host. At time t, there are $I(t)$ infected hosts, among them $[I(t) - \bar{I}(t)]$ are infected only by the spearhead — they have not infected others yet. At the next small time interval δ, each spearhead-only infected host will have the probability $p = \eta\delta N/\Omega$ to infect another host since there are N targets to infect (a target host that has already been infected still counts). Therefore, on average $[I(t) - \bar{I}(t)]p$ spearhead-only infected hosts will infect others and download the main-force of the worm during the small time interval δ. Thus we have,

$$\bar{I}(t+\delta) - \bar{I}(t) = [I(t) - \bar{I}(t)] \cdot p = \frac{\eta}{\Omega}[I(t) - \bar{I}(t)]N \cdot \delta \tag{2}$$

Taking $\delta \rightarrow 0$ yields the botnet growth model ($m = 1$):

$$\begin{aligned} \frac{d\bar{I}(t)}{dt} &= \frac{\eta}{\Omega}[I(t) - \bar{I}(t)]N \\ \frac{dI(t)}{dt} &= \frac{\eta}{\Omega} I(t)[N - I(t)] \end{aligned} \tag{3}$$

For a general two-stage reconnaissance worm that has $m > 1$, we can derive the botnet growth model in the similar way. For example, if $m = 2$, then we need to add an intermediate variable $I_1(t)$ to represent the number of spearhead-only infected hosts at time t — each of them has infected exactly one host at time t. Using the similar infinitesimal analysis as illustrated above, we can derive the botnet growth model ($m = 2$):

$$\begin{aligned} \frac{d\bar{I}(t)}{dt} &= \frac{\eta}{\Omega} I_1(t) N \\ \frac{dI_1(t)}{dt} &= \frac{\eta}{\Omega}[I(t) - I_1(t) - \bar{I}(t)]N - \frac{d\bar{I}(t)}{dt} \\ \frac{dI(t)}{dt} &= \frac{\eta}{\Omega} I(t)[N - I(t)] \end{aligned} \tag{4}$$

The above two models assume that the spearhead in host A can download the main-force immediately after it infects m target hosts, which means we assume that at least one of the hosts in A's buddy list contains the main-force when A wants to download the main-force. If the size of the buddy list is not too small, this assumption is probably accurate for modeling purposes.

We use Matlab Simulink [1] to derive the numerical solutions of model (3) and model (4). We use the Code Red worm parameters [31], $N = 360,000$, $\eta = 358$/min, $\Omega = 2^{32}$ in the calculation and assume one initially infected host. Fig. 5 shows the worm propagation and the botnet growth over time. This figure shows that the botnet is constructed with a certain time delay (depends on m) as the worm spreads, but in the end all infected hosts will join the

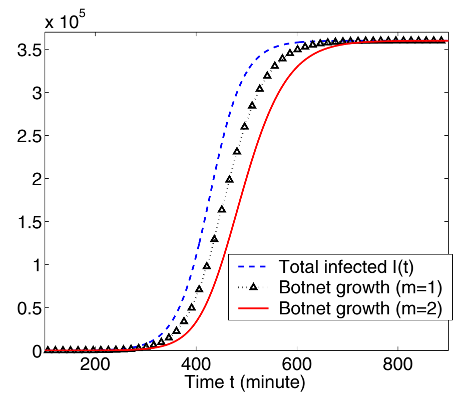

Figure 5. Worm propagation and the constructed botnet growth

P2P botnet. This shows that the method described could potentially produce a viable and large botnet capable of avoiding current botnet monitoring techniques quite rapidly.

5 Defense Against Honeypot-Aware Attacks

In this section, we discuss how to defend against the general honeypot-aware attacks (not just botnets) introduced in previous sections.

The honeypot-aware botnets introduced in this paper rely on the basic principle that security professionals have liability constraints, while attackers do not need to obey such constraints. The fundamental counterattack by security professionals is to somehow invalidate this principle. For example, some national organizations or major security companies could set up limited-scale honeypot-based detection systems that are authorized by legal officials to freely send out malicious traffic.

Of course, the law currently regulating cyberspace security is not mature or defined in many countries, and hence, some researchers or security defenders have deployed honeypots that freely send out malicious attacks. For these honeypots, attackers cannot detect them using the honeypot detection methodology presented in this paper. However, such honeypot defense practices are negligent and unethical, which will probably become illegal as the laws regarding cyberspace security and liability gradually mature.

The current popular GenII honeynet [20] has considered preventing attack traffic from being sent, but it does not implement this as strictly as the assumption used in the paper: it limits outgoing connection rate and can block/modify *detected* outgoing malicious traffic. For this reason, the GenII honeynet might be able to avoid the honeypot detection conducted by attackers (e.g., if a bot only tests if it can send out malicious code to a sensor), but at the same time, it could actually infect other computers as well and thus potentially make the honeynet owners liable for the ensuing damage.

Another promising defense against honeypot-aware attacks is the “double-honeypot” idea [27]. From Section 4.4, we can see that attackers need to take extra steps in order to avoid being fooled by double-honeypot traps. By using dual honeypots, or a distributed honeypot network that can accurately emulate the outside Internet, security defenders can take proactive roles in deceiving honeypot-aware attacks. For example, security defenders can build a large-scale distributed honeynet to cover many blocks of IP space, and allow all malicious traffic to pass freely within this honeypot virtual network. However, this defense will be ineffective if attackers use their own sensors to detect honeypots (as introduced in Section 3).

Security defenders could also try to distinguish which outgoing traffic is for honeypot detection and which outgoing traffic is for a real attack. If this could be done, then honeypots could be configured to allow the honeypot-detection traffic to be sent while blocking all other malicious traffic. For this purpose, security defenders will need to conduct more research on practically implementing automatic binary code analysis in honeypots.

Internet security attack and defense is an endless war. From the attackers’ perspective, there is a trade-off between detecting honeypots in their botnets and avoiding bot removal by security professionas. If an attacker conducts honeypot-aware test on a botnet frequently, honeypots in the botnet can be detected and removed quickly. But at the same time, the bots in the botnet will generate more outgoing traffic, and hence, they have more chance to be detected and removed by their users or security staff.

In the end, we should emphasize that even if attackers can successfully detect and remove honeypots based on the methodology presented in the paper, there is still significant value in honeypot research and deployment for detecting the infection vector and the source of attacks. It may not be possible for honeypots to join the botnet, but the security hole used to facilitate the infection can be quickly discovered and patched.

6 Conclusion

Due to their potential for illicit financial gain, “botnets” have become popular among Internet attackers in recent years. As security defenders build more honeypot-based detection and defense systems, attackers will find ways to avoid honeypot traps in their botnets. Attackers can use software or hardware specific codes to detect the honeypot virtual environment [2, 8, 24], but they can also rely on a more general principle to detect honeypots: security professionals using honeypots have liability constraints such that their honeypots cannot be configured in a way that would allow them to send out real malicious attacks or too many malicious attacks. In this paper, we introduced various means by which attackers could detect honeypots in their constructed botnets based on this principle. Honeypot research and deployment still has significant value for the security community, but we hope this paper will remind honeypot researchers of the importance of studying ways to build covert honeypots, and the limitation in deploying honeypots in security defense. The current popular research focused on finding effective honeypot-based detection and defense approaches will be for naught if honeypots remain as easily detectible as they are presently.

References

[1] Mathworks Inc.: Simulink. http://www.mathworks.com/products/simulink.

[2] Honeyd security advisory 2004-001: Remote detection via simple probe packet. http://www.honeyd.org/adv.2004-01.asc, 2004.

[3] K. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. Keromytis. Detecting targeted attacks using shadow honeypots. In *Proceedings of 14th USENIX Security Symposium*, August 2005.

[4] M. Bailey, E. Cooke, D. Watson, F. Jahanian, and N. Provos. A hybrid honeypot architecture for scalable network monitoring. Technical Report CSE-TR-499-04, U. Michigan, October 2004.

[5] J. Bethencourt, J. Franklin, and M. Vernon. Mapping Internet sensors with probe response attacks. In *Proceedings of USENIX Security Symposium*, pages 193–208, August 2005.

[6] CERT. CERT/CC advisories. http://www.cert.org/advisories/.

[7] E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Proceedings of SRUTI: Steps to Reducing Unwanted Traffic on the Internet*, July 2005.

[8] J. Corey. Advanced honey pot identification and exploitation. http://www.phrack.org/fakes/p63/p63-0x09.txt, 2004.

[9] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levin, and H. Owen. Honeystat: Local worm detection using honeypots. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2004.

[10] D. Dagon, C. C. Zou, and W. Lee. Modeling botnet propagation using time zones. In *Proceedings of 13th Annual Network and Distributed System Security Symposium (NDSS)*, pages 235–249, Feburary 2006.

[11] C. Drake, J. Oliver, and E. Koontz. Mailfrontier, Inc. whitepaper: Anatomy of a Phishing email. http://www.ceas.cc/papers-2004/114.pdf, 2004.

[12] F. Freiling, T. Holz, and G. Wicherski. Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. Technical Report AIB-2005-07, CS Dept. of RWTH Aachen University, April 2005.

[13] X. Jiang and D. Xu. Collapsar: A vm-based architecture for network attack detention center. In *Proceedings of 13th USENIX Security Symposium*, August 2004.

[14] G. Kesidis, I. Hamadeh, and S. Jiwasurat. Coupled kermack-mckendrick models for randomly scanning and bandwidth-saturating internet worms. In *Proceedings of 3rd International Workshop on QoS in Multiservice IP Networks (QoS-IP*, pages 101–109, February 2005.

[15] N. Krawetz. Anti-honeypot technology. *IEEE Security & Privacy Magazine*, 2(1), 2004.

[16] C. Kreibich, A. Warfield, J. Crowcroft, S. Hand, and I. Pratt. Using packet symmetry to curtail malicious traffic. In *Proceedings of the Fourth Workshop on Hot Topics in Networks (HotNets-IV)*, November 2005.

[17] J. Levine, R. LaBella, H. Owen, D. Contis, and B. Culver. The use of honeynets to detect exploited systems across large enterprise networks. In *Proceedings of IEEE Workshop on Information Assurance*, June 2003.

[18] B. McCarty. Botnets: Big and bigger. *IEEE Security & Privacy Magazine*, 1(4), July 2003.

[19] D. Nicol and M. Liljenstam. Models of active worm defenses. In *Proceedings of the IPSI Studenica Conference*, June 2004.

[20] H. Project. Know your enemy: GenII honeynets. http://www.honeynet.org/papers/gen2, 2005.

[21] H. Project. Know your enemy: Tracking botnets. http://www.honeynet.org/papers/bots/, 2005.

[22] N. Provos. A virtual honeypot framework. In *Proceedings of 13th USENIX Security Symposium*, August 2004.

[23] R. Puri. Bots & botnet: An overview. http://www.sans.org/rr/whitepapers/malicious/1299.php, 2003.

[24] K. Seifried. Honeypotting with VMware basics. http://www.seifried.org/security/index.php/Honeypotting_With_VMWare_Basics, 2002.

[25] L. Spitzner. Honeypots: Are they illegal? http://www.securityfocus.com/infocus/1703, 2003.

[26] S. Staniford, V. Paxson, and N.Weaver. How to own the Internet in your spare time. In *Proceedings of USENIX Security Symposium*, pages 149–167, August 2002.

[27] Y. Tang and S. Chen. Defending against internet worms: A signature-based approach. In *Proceedings of the IEEE INFOCOM*, May 2005.

[28] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound. Dynamic updates in the domain name system (DNS update). http://www.ietf.org/rfc/rfc2136.txt, 1997.

[29] M. Vrable, J. Ma, J. chen, D. Moore, E. Vandekieft, A. Snoeren, G. Voelker, and S. Savage. Scalability, fidelity and containment in the potemkin virtual honeyfarm. In *Proceedings of the ACM Symposium on Operating System Principles (SOSP)*, October 2005.

[30] C. C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and early warning for Internet worms. In *Proceedings of 10th ACM Conference on Computer and Communications Security (CCS'03)*, pages 190–199, October 2003.

[31] C. C. Zou, W. Gong, and D. Towsley. Code Red worm propagation modeling and analysis. In *Proceedings of 9th ACM Conference on Computer and Communications Security (CCS'02)*, pages 138–147, October 2002.

[32] C. C. Zou, D. Towsley, and W. Gong. On the performance of Internet worm scanning strategies. *Journal of Performance Evaluation*, to appear.