Honeypot detection in advanced botnet attacks

Ping Wang, Lei Wu, Ryan Cunningham
and Cliff C. Zou*

School of Electrical Engineering
and Computer Science
University of Central Florida
Orlando, FL 32816–2362, USA
E-mail: pwang@eecs.ucf.edu
E-mail: lwu@eecs.ucf.edu
E-mail: rcunning@eecs.ucf.edu
E-mail: czou@eecs.ucf.edu
*Corresponding author

Abstract: *Botnets* have become one of the major attacks in the internet today due to their illicit profitable financial gain. Meanwhile, *honeypots* have been successfully deployed in many computer security defence systems. Since honeypots set up by security defenders can attract botnet compromises and become spies in exposing botnet membership and botnet attacker behaviours, they are widely used by security defenders in botnet defence. Therefore, 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 a liability constraint such that they cannot allow their honeypots to participate in real attacks that could cause damage to others, while attackers do not need to follow this constraint. Attackers could detect honeypots in their botnets by checking whether compromised machines in a botnet can successfully send out unmodified malicious traffic. Based on this basic detection principle, we present honeypot detection techniques to be used in both centralised botnets and Peer-to-Peer (P2P) structured botnets. Experiments show that current standard honeypots and honeynet programs are vulnerable to the proposed honeypot detection techniques. At the end, we discuss some guidelines for defending against general honeypot-aware attacks.

Keywords: liability; honeypots; botnets; peer-to-peer; P2P; modelling.

Reference to this paper should be made as follows: Wang, P., Wu, L., Cunningham, R. and Zou, C.C. (2010) 'Honeypot detection in advanced botnet attacks', *Int. J. Information and Computer Security*, Vol. 4, No. 1, pp.30–51.

Biographical notes: Ping Wang received her BS and MS Degrees in Computer Science from Beijing University of Aeronautics and Astronauts, China, in 2001 and 2004, respectively. Currently she is working towards a PhD Degree in the School of Electrical Engineering and Computer Science at the University of Central Florida, USA. Her research interests include computer and network security.

Lei Wu received his BS in Software Engineering and MS in Computer Science from Nanjing University, China, in 2005 and 2008, respectively. Currently he is working towards a PhD Degree in the School of Electrical Engineering and Computer Science at the University of Central Florida, USA. His research interests include computer and network security.

Ryan Cunningham received his BS and MS Degree in Computer Science from the University of Central Florida, USA, in 2005 and 2007, respectively. Currently he is working towards a PhD Degree in the Department of Computer Science at the University of Illinois at Urbana-Champaign.

Cliff C. Zou received his BS and MS Degrees from the University of Science and Technology of China in 1996 and 1999, respectively. Then he received a PhD Degree in the Department of Electrical and Computer Engineering from the University of Massachusetts, Amherst, MA, USA in 2005. Currently he is an Assistant Professor in the School of Electrical Engineering and Computer Science, University of Central Florida. His research interests include computer and network security, network modelling and wireless networking.

1 Introduction

In the last ten years, internet users have been attacked unremittingly by widespread e-mail viruses and worms. The famous wide-spreading e-mail viruses include Melissa in 1999, Love Letter in 2000, W32/Sircam in 2001, MyDoom, Netsky and Bagle in 2004. Similarly famous wide-spreading scanning worms include Code Red and Code Red II in 2001, Slammer and Blaster in 2003, Witty and Sassar in 2004 (CERT, 2001, 2003, 2004). However, we have not seen a major virus or worm outbreak in the similar scale after the Sassar worm incident in May 2004. This is not because the internet is much more secure, but more likely because attackers have shifted their attention to compromising and controlling victim computers, an attack scheme which provides more potential for personal profit and attack capability.

This lucrative attack theme 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 (Bächer *et al.*, 2008). Each compromised computer is installed 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, or called '*botmaster*'. Botmasters maintain complete control of their botnets, and can conduct Distributed Denial-of-Service (DDoS) attacks, e-mail spamming, keylogging, abusing online advertisements, spreading new malware, *etc.* (Bächer *et al.*, 2008; Dagon *et al.*, 2006).

Turning our focus now to the defense side in computer security. A '*honeypot*' is a special constructed computer or network trap designed to attract and detect malicious attacks (Even, 2000). In recent years, honeypots have become popular, and security researchers have generated many successful honeypot-based attack analysis and detection systems (such as Anagnostakis *et al.*, 2005; Dagon *et al.*, 2004; Jiang and Xu, 2004; Levine *et al.*, 2003; Provos, 2004; Rajab *et al.*, 2007; Vrable *et al.*, 2005). As more people begin to use honeypots in monitoring and defense systems, botmasters constructing and maintaining botnets will sooner or later try to find ways to avoid honeypot traps.

In this paper, we present how botmasters might attempt to remove honeypot traps when constructing and maintaining their botnets. This knowledge is useful for security professionals to better prepared for more advanced botnet attacks in the near future. Unlike hardware or software specific honeypot detection methods (Corey, 2004; Honeyd, 2004; Seifried, 2002), the honeypot detection methodology presented here is based on a general principle that is hardware and software independent: security defenders who set up honeypots have liability constraint such that they cannot allow their honeypots to send out real attacks to cause damage to others, while botmasters do not need to follow this constraint. As laws are developed to combat cybercrime in the coming years, security experts deploying honeypots will probably incur more liability constraint than they have today, because they knowingly allow their honeypots to be compromised by attackers. If they fail to perform due diligence by securing their honeypot from damaging other machines, they will be considered negligent.

To our knowledge, this is the first paper to systematically study honeypot detection that could be deployed by attackers based on the above general principle. Although Lance Spitzner (2003) and Richard Salgado (2005) addressed the basic potential legal issues of honeypots, their discussion was general and did not provide details of what attackers might exploit the legal issues and how to deal with such exploits.

Based on this principle, we present a novel honeypot detection technique, which is simple but effective. Botmasters could command their botnets to actively send out malicious traffic (or *counterfeit* malicious traffic) to one or several other compromised computers. These computers behave as ‘sensors’. Botmasters 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. Simulation experiments show that current standard honeypot and honeynet programs are vulnerable to the proposed attack.

The above honeypot detection technique will also falsely treat some normal computers as honeypots: these computers are subject to security egress filtering such that their outgoing malicious traffic are blocked. This false positive in honeypot detection, however, does not matter to botmasters. If a bot computer cannot send out malicious traffic, it is better be removed from a botnet, no matter whether it is a honeypot or a well managed normal computer.

To detect a hijacked bot controller in a hierarchical botnet, botmasters can issue a test command via the bot controller under inspection that causes botnet members to send trivial traffic to the botmasters’ ‘sensors’. The hijacked controller can then easily be detected if the command is not carried out or is not carried out correctly. In addition, botmasters can detect bot controllers hijacked via DNS redirection (Dagon *et al.*, 2006) 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 Peer-to-Peer (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.

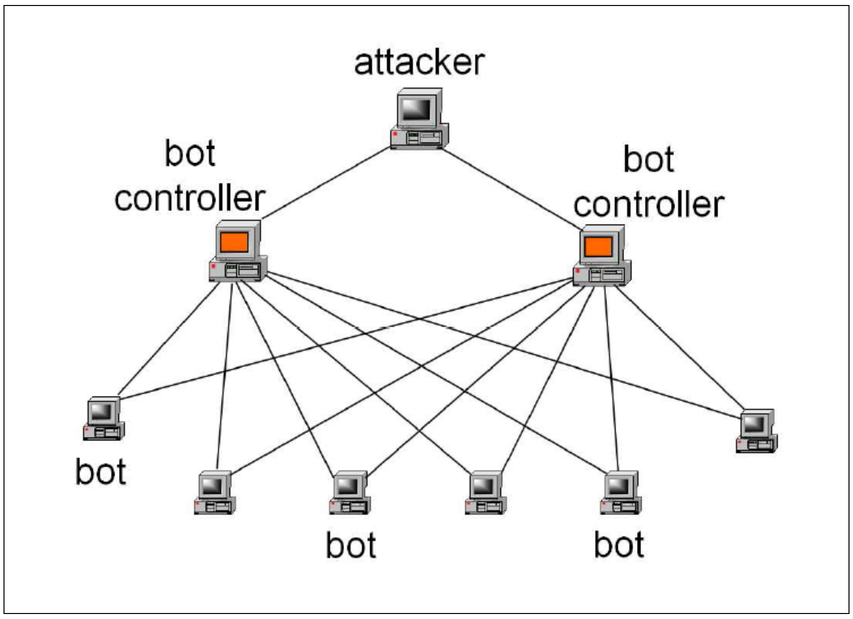
The rest of this paper is organised as follows. Section 2 presents the honeypot detection methods for current hierarchical botnets. In Section 3, we conduct simulation experiments by using the current GenII honeynet (Honeynet Project, 2005) program, showing that current honeypots are vulnerable to the proposed attack. 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. Section 6 discusses related work. In the end we summarise our conclusions in Section 7.

2 Honeypot detection in hierarchical botnets

2.1 Hierarchical botnets introduction

Most botnets currently known in the internet are controlled by botmasters via a hierarchical network structure. Figure 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 botmaster's real identity) to all bot controllers. To prevent defenders from shutting down the command and control channel, botmasters usually use multiple redundant bot controllers in their botnets.

Figure 1 Illustration of a hierarchical botnet (see online version for colours)



To set up bot controllers flexibly, botmasters usually hard-code bot controllers' domain names rather than their IP addresses in all bots (Dagon *et al.*, 2006). Botmasters also try to keep their bot controllers mobile by using Dynamic DNS (DDNS) (Vixie *et al.*, 1997), 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, botmasters can simply create another bot controller on a new compromised machine and update the DDNS entry to point to the new controller.

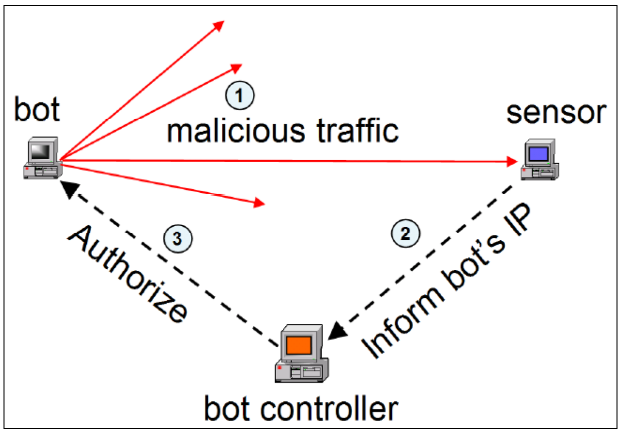
In the rest of this section, we introduce how botmasters can thwart the two botnet trapping techniques presented in the beginning of Section 6, respectively.

2.2 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 'counterfeit' malicious traffic to one or several remote computers that are actually controlled by the botmaster. These remote computers behave as 'sensors' for the botmaster. 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 botmaster's sensors and which ones might be innocent computers, they cannot defend against this honeypot detection technique without incurring the risk of attacking innocent computers.

This honeypot detection procedure is illustrated in Figure 2. A newly infected computer cannot join a botnet before it is verified. This potential bot machine must first send out malicious traffic to many targets, including the botmaster's secret sensor (unknown to the newly infected machine). When the botmaster'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 authorises the checked bot so that the bot can join the botnet. To prevent the possibility of a single point of failure, a botmaster could set up multiple sensors for this test.

Figure 2 Illustration of the procedure in detecting honeypot bots in a hierarchical botnet (see online version for colours)



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 authorisation mechanism. The botmaster (or the botnet controller) uploads the authorisation key to the host and allows it to join the botnet only after the host passes the honeypot detection test. In addition, botmasters may perform the honeypot detection periodically on botnets to discover additional honeypot bots. This could be done whenever botmasters renew their bots' authorisation keys or encryption keys, or update the botnet software.

This honeypot detection scheme relies on the report of sensors deployed by botmasters. Therefore, botmasters must first ensure that sensor machines themselves are not honeypots. This is not hard to be done since only a few sensor machines are needed – botmasters can manually investigate these machines thoroughly beforehand.

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

2.2.1 Detection through infection

When a computer is compromised and a bot program is installed, some 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, a botmaster could let compromised computers send malicious infection traffic to her sensors.

Some honeypots, such as the GenII honeynets (Honeynet Project, 2005), have Network Intrusion Prevention System (NIPS) that can modify outbound malicious traffic to disable the exploit. To detect such honeypots, botmasters' 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 e-mail, the sensors' e-mail addresses could be put at a random point in the outgoing e-mail address list. This procedure will delay the newly infected computer's participation in the botnet, but a botmaster would be willing to incur this slight delay to secure their botnet, because a botnet has long term use to its botmaster.

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 botmaster's sensors. A botmaster can make the code analysis even harder by or encrypting sensors' IP addresses in the code.

2.2.2 Detection through other illicit activities

Based on our general honeypot detection principle, botmasters 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 (Honeynet Project, 2005) limits the number of outbound connection rate.

  Some normal computers are configured (*e.g.*, installed a firewall, or a worm detection software such as Kreibich *et al.*, 2005) to limit outgoing connection rate as well. To avoid mislabelling such computers as honeypots, and also to avoid possible detection by users, botmasters should let their bots conduct a very low rate stealthy port-scan for honeypot detection.

- *E-mail spamming*. A botmaster could also have a bot send out spam e-mail to one or several target e-mail addresses owned by the botmaster. These e-mail addresses behave as the honeypot detection sensors. Outgoing e-mail spam, such as 'phishing' e-mail (Drake *et al.*, 2004), could make honeypot security professionals liable for substantial financial losses if they reach real users.

2.3 Detection of hijacked bot controllers

Now we introduce techniques to detect hijacked bot controllers. With the help from DDNS providers, Dagon *et al.* (2006) 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 a botmaster'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, botmasters will do everything they can to eliminate a hijacked bot controller from their botnets. In this section, we present two different techniques that botmasters might use to achieve this goal.

2.3.1 Bot controller DNS query check

When a bot controller is hijacked by the DNS redirection method presented in Dagon *et al.* (2006), 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 botmaster and are running the botmaster's bot controlling program.

Therefore, a botmaster can keep an up-to-date DNS mapping table of all bot controllers. Using one compromised computer as a sensor, the botmaster 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 botmaster 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 botmaster 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.

2.3.2 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, a botmaster can use the same honeypot detection principle we described before. The botnet owner checks whether or not a bot controller passes the botmaster's commands to bots. The monitor presented in Dagon *et al.* (2006) is called 'sinkhole' because it does not pass any botmaster'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 a botmaster'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 that a botnet implements its unique command system. In this case, a known trivial command based on previous experience could possibly conduct harmful task such as deletes files on all of the compromised computers, or launches DDoS attacks against risky targets.

Based on this, a botmaster 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 botmaster'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 botmaster knows that the bot controller has been hijacked (or is at least not working as required).

2.4 Discussions

Honeypot detection procedure will make an infected computer waiting for a while before it can join a botnet. However, this time delay does not affect the infection speed of computers by a botnet. Because most botnets are used by bot-masters as long-time attacking tools, the time delay caused by honeypot detection for infected computers joining a botnet, even as long as several hours or a day, is usually not an issue for botmasters. Therefore, it is not very important for botmasters to consider the trade-off issue between time and accuracy in detecting honeypots.

When deploying a sensor to detect honeypot bots in a botnet, the sensor machine must not be overwhelmed by the testing traffic sent by bots. When conducting honeypot detection, each tested bot only needs to send *one* piece of testing traffic (such as a connection, an e-mail spam, a copy of infection code) to the sensor. Thus a sensor machine is not likely to be overloaded by honeypot detection traffic in most cases, unless bots in a large-size botnet send their test traffic to the sensor at exactly the same time.

To prevent such a rare overload event happens in a large-size botnet, the botmaster could command bots to randomly choose their report time within a time period in order to spread the test traffic load.

When using continuous DNS queries to test whether a bot controller is hijacked by defenders or not, the sensor machine sending out these DNS queries might be detected by defenders by the abnormal DNS queries. Botmasters could prevent this exposure by:

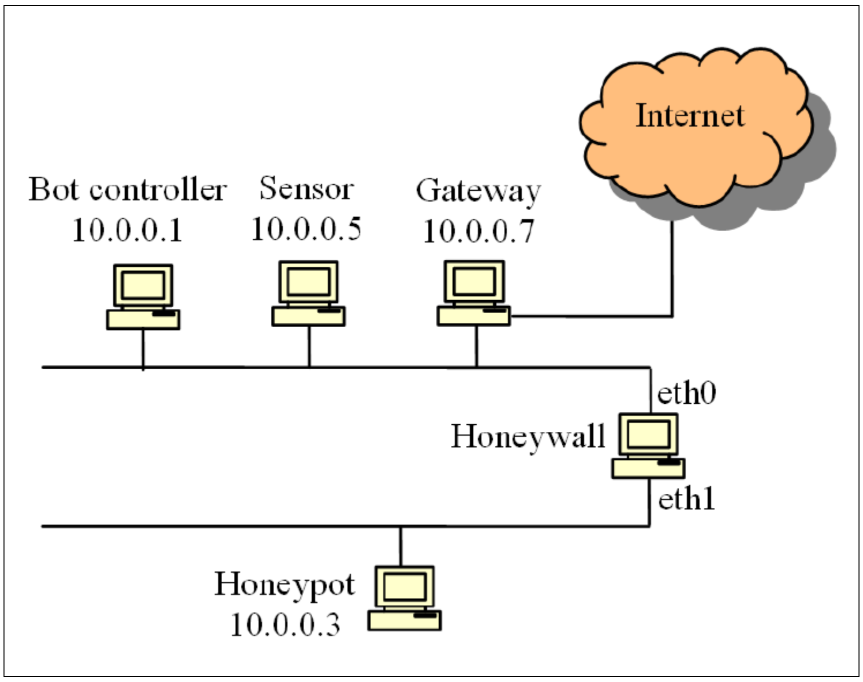
- slowing down the query frequency
- using multiple sensors conducting this query
- sending out DNS queries to many internet DNS servers.

3 Experiment evaluation

3.1 Experiment network system introduction

We conduct experiments to demonstrate the practicability of the proposed honeypot detection methodology by deploying a standard "GenII honeynet" (Honeynet Project, 2005). Figure 3 illustrates the network deployment in our experiments. In this network, there are five computers, and they are functioning as botnet controller, botnet sensor (used for honeypot detection), network gateway, honeywall, and honeypot respectively.

Figure 3 Network deployment for honeypot detection experiments (see online version for colours)



Currently there is no real internet botnet that has implemented the proposed honeypot detection technique. Therefore, in our experiment we extend the Code Red II worm (CodeRedII, 2001) to generate a basic functioning botnet that incorporates the proposed honeypot detection technique.

The botnet controller and the sensor apply the proposed honeypot detection method, cooperating with each other to detect honeypots in the entire botnet. The honeypot is a vulnerable host compromised by the botnet – in experiments, we let the bot controller to send malicious traffic compromising the honeypot. As the botnet continues to spread, the bot program on the compromised honeypot generates a set of random IP addresses, and sends out packets containing malicious code to hosts with those IP addresses. In addition, the simulated bot program on the honeypot sends out predefined packet to the sensor as well, and this action could happen at any time during the propagation phase.

Honeywall is an important component of a honeynet. It is a gateway device that separates honeypots from the rest of the world. Any traffic going to or from the honeypot must go through the honeywall (Honeynet Project, 2005). We use Honeywall CDROM (version roo) to configure the honeywall, which is working under the bridge mode.

Two data control and capture tools running on honeywall: IPTables and Snort_inline[1] IPTables behaves as a firewall. It limits the rate of the outbound (from a honeypot to the internet) traffic, but allows all inbound (from the internet to a honeypot) traffic. Snort_inline is an Intrusion Detection System (IDS), monitoring outbound traffic instead of inbound traffic in our experiments. If a packet is not dropped by IPTables, it will be forwarded to Snort_inline, and Snort_inline will analyse it for matches against a user-defined rule set and perform actions such as dropping, modifying, or rejecting the packet, *etc.* (Snort, 2006).

The gateway in Figure 3 is supposed to connect to the internet, accepting and forwarding any packets targeting all other IP addresses. To prevent malicious traffic going out, in our experiment the gateway does not connect to the internet and drops all packets going out.

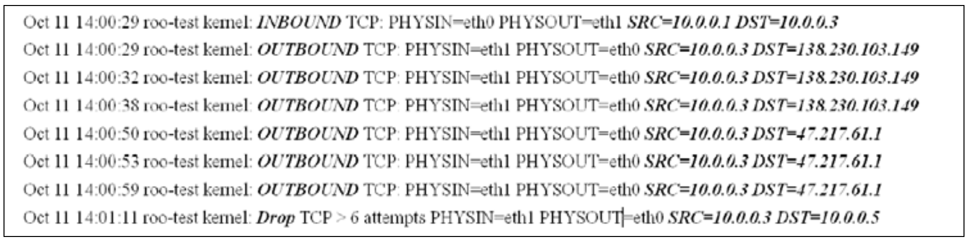
We conduct two types of experiments to show that both data control mechanisms-IPTables and Snort_inline–could cause a honeypot being detected by the proposed attack. In our experiments, instead of using real botnet codes, we emulate botnet infection and bot scanning process.

3.2 Honeypot detection – IPTables

In this experiment, we evaluate the honeypot detection due to the usage of IPTables on honeywall. In our experiment, IPTables is configured to allow six outbound TCP connection attempts per hour, and Snort_inline is not running. The abridged log file of IPTables in this experiment is shown below:

The first entry in Figure 4 is an inbound traffic from the bot controller (10.0.0.1) to the honeypot (10.0.0.3). This traffic causes the honeypot to be infected and installed with the bot program. The next six entries are all outbound traffic from the honeypot to different destinations, which implies the honeypot has been infected and starts to propagate by sending out probing traffic to randomly generated IP addresses. Only two IP addresses have been scanned because MS Windows will try to connect to each IP address three times. After these six outbound connection attempts, the IPTables has reached its blocking threshold. The last entry shows that, the outbound packet, which goes to the botnet sensor (10.0.0.5), is dropped by IPTables because the limit of outbound traffic has been reached. In this experiment, the botnet sensor is able to detect the honeypot due to the blocked traffic that originally targets the sensor.

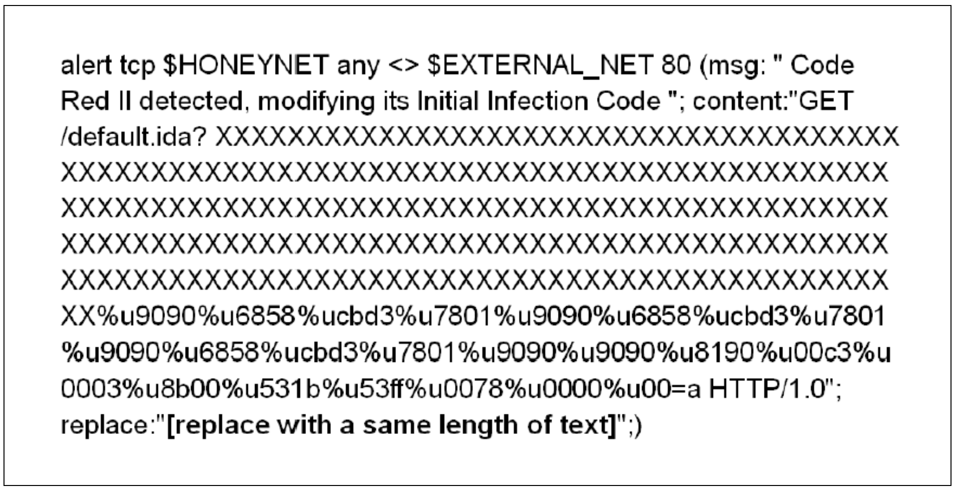
Figure 4 Abridged log file of IPTables on the honeywall



```
Oct 11 14:00:29 roo-test kernel: INBOUND TCP: PHYSIN=eth0 PHYSOUT=eth1 SRC=10.0.0.1 DST=10.0.0.3
Oct 11 14:00:29 roo-test kernel: OUTBOUND TCP: PHYSIN=eth1 PHYSOUT=eth0 SRC=10.0.0.3 DST=138.230.103.149
Oct 11 14:00:32 roo-test kernel: OUTBOUND TCP: PHYSIN=eth1 PHYSOUT=eth0 SRC=10.0.0.3 DST=138.230.103.149
Oct 11 14:00:38 roo-test kernel: OUTBOUND TCP: PHYSIN=eth1 PHYSOUT=eth0 SRC=10.0.0.3 DST=138.230.103.149
Oct 11 14:00:50 roo-test kernel: OUTBOUND TCP: PHYSIN=eth1 PHYSOUT=eth0 SRC=10.0.0.3 DST=47.217.61.1
Oct 11 14:00:53 roo-test kernel: OUTBOUND TCP: PHYSIN=eth1 PHYSOUT=eth0 SRC=10.0.0.3 DST=47.217.61.1
Oct 11 14:00:59 roo-test kernel: OUTBOUND TCP: PHYSIN=eth1 PHYSOUT=eth0 SRC=10.0.0.3 DST=47.217.61.1
Oct 11 14:01:11 roo-test kernel: Drop TCP > 6 attempts PHYSIN=eth1 PHYSOUT=eth0 SRC=10.0.0.3 DST=10.0.0.5
```

If the packet sending to the botnet sensor is among the first six TCP connections, our experiment shows that the packet will successfully arrive the botnet sensor, which could cause the honeypot undetected by the botnet. Therefore, botmasters would mostly likely make their bots connecting to sensors after sending out a number of scans.

Figure 5 Snort_inline rule to modify Code Red II outgoing packets



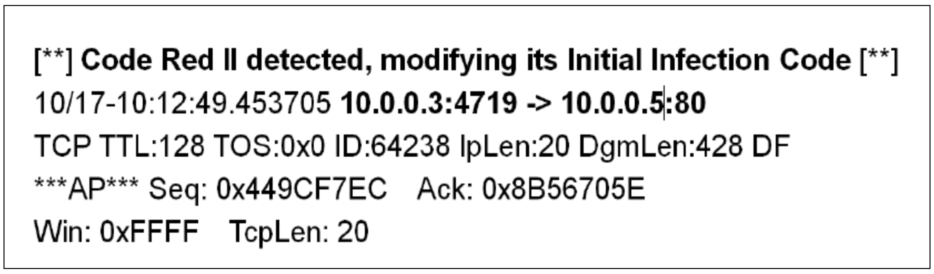
```
alert tcp $HONEYNET any <> $EXTERNAL_NET 80 (msg: " Code
Red II detected, modifying its Initial Infection Code "; content:"GET
/default.ida? XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XX%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801
%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3%u
0003%u8b00%u531b%u53ff%u0078%u0000%u00=a HTTP/1.0";
replace:"[replace with a same length of text]";)
```

3.3 Honeypot detection–Snort_inline

In this experiment, the Snort_inline is activated. One primary function of Snort_inline in honeywall is to detect outgoing malicious traffic, and then either drop it or modify it to be unmalicious. In this experiment, Snort_inline contains the rule shown in Figure 5 to change a *Code Red II* (CodeRedII, 2001) packet to become unmalicious.

In the experiment, the honeypot sends out a packet containing the Code Red II worm to the sensor, Snort_inline captures the packet and modifies its content, as shown in Figure 5, to a same length of unharmful text. Figure 6 shows the related entry in Snort_inline's log file. Even though the sensor (10.0.0.5) can receive this packet, the botnet still detects the honeypot because the content of the required packet has been changed.

Figure 6 Related Snort_inline log entry on honeywall



```
[**] Code Red II detected, modifying its Initial Infection Code [**]
10/17-10:12:49.453705 10.0.0.3:4719 -> 10.0.0.5:80
TCP TTL:128 TOS:0x0 ID:64238 IpLen:20 DgmLen:428 DF
***AP*** Seq: 0x449CF7EC   Ack: 0x8B56705E
Win: 0xFFFF   TcpLen: 20
```

4 Honeypot detection in P2P-structured botnets

Cooke *et al.* (2005) discussed three different botnet communication topologies and their properties: centralised, P2P, and random. In a random topological botnet, a bot knows no more than one other bot (Cooke *et al.*, 2005). 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 centralised topology discussed in Cooke *et al.* (2005)) introduced in the previous Section 2.1. To increase the availability of the command&control channel in a hierarchical botnet, a botmaster has to increase the number of bot controllers in the botnet. This will increase the financial cost of maintaining the botnet, since the botmaster will need to purchase more DDNS 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 Dagon *et al.* (2006).

To botmasters, changing a botnet's control architecture to be P2P is a natural way to make a botnet harder to be shut down by defenders. In recent years, botmasters have tested and implemented different kinds of preliminary P2P botnets such as Slapper (Arce and Levy, 2003), Sinit (Stewart, 2003), Phatbot (Stewart, 2004) and Nugache (Lemos, 2006). Some researchers have studied P2P botnet designs (Vogt *et al.*, 2007; Wang *et al.*, 2007). Therefore, we believe more P2P botnets will be created in the near future.

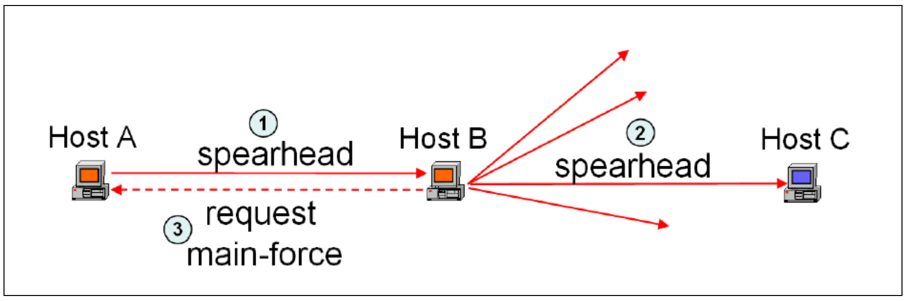
Botmasters will need to come up with a new honeypot detection technique for a P2P botnet. In a P2P botnet, each bot contains a list of IP addresses of other bots that it can connect with, which is called "peer list" (Wang *et al.*, 2007). Because there are no centralised bot controllers to provide authentication in a P2P botnet, each bot must make its own decision, or collaborate with its peers, to decide whether its hosted machine is a honeypot or not. In this paper, we present a simple but effective advanced worm, called '*two-stage reconnaissance worm*', that can be used to distributively detect honeypots as it propagates.

4.1 Two-stage reconnaissance worm

A two-stage reconnaissance 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 authorisation component allowing the infected host to join 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. Figure 7 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 peer 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 peer list that has the main-force component. The main-force code lets the worm join the constructed botnet via the authorisation key contained in the main-force (*e.g.*, the authorisation key could be a private public key).

Figure 7 Illustration of the propagation procedure of a two-stage reconnaissance worm (see online version for colours)



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 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 botmaster 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 (CERT). 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.

The reconnaissance worm described above needs a separate procedure (Procedure 3 as shown in Figure 7) to obtain the complete bot code. This could be a problem for a botnet since the original Host A might be unaccessible from others, or Host A has changed its IP address when Host Be tries to get the main-force worm code. To deal with this issue, the worm could combine the main-force code together with the spearhead code, but first deactivate and possibly encrypt the main-force code at the beginning. After the spearhead code verifies that a hosted machine is not honeypot, it will unpack and execute the main-force code. One drawback of this approach is that honeypot defenders can easily obtain the main-force code even when their honeypots are not able to join the botnet.

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

Tang and Chen (2005) 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

2 a dual honeypot can emulate any outside remote host with arbitrary IP address.

To detect such a dual-honeypot defense system, botmasters can design an even more advanced two-stage reconnaissance worm that propagates as following (illustrated in Figure 8):

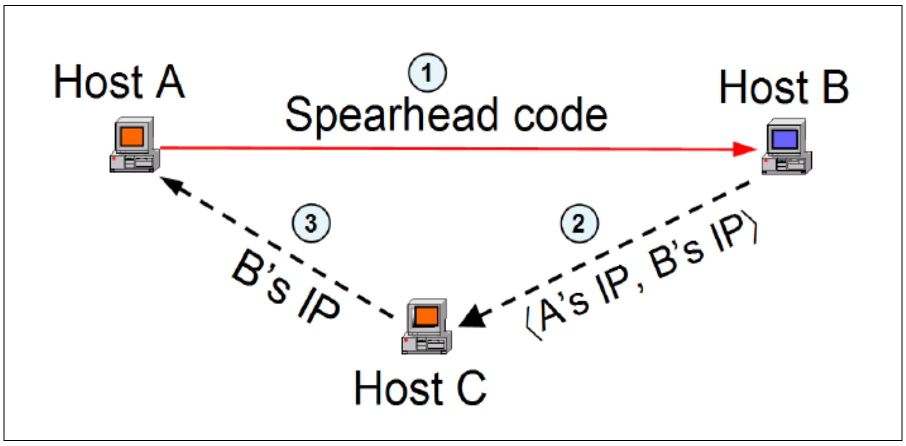
- When an infected host A finds and infects a target host B, it records host B's IP address. Host A continues finding and infecting others in this way. Host A has a peer list for connecting neighboring bots. The peer list can be constructed according to the P2P botnet designs presented in Vogt *et al.* (2007) and Wang *et al.* (2007).
- Host B sets up a TCP connection with every host in host A's peer 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 peer list. At the time when it receives the tuple ⟨A's IP, B's IP⟩, host C may or may not be a fully-fledged member of the botnet. Host C records this tuple if the incoming TCP connection is really from the claimed host B's IP address.
- After host C obtains the main-force of the worm (host C passes the honeypot detection test earlier than host A), it informs host A of host B's IP address, digitally signed by the private authorisation key. This report can be done only by hosts having the main-force code because the authorisation 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 infected a real host. After host A finds out that it has successfully infected *m* real hosts, the honeypot detection procedure is over. And then host A tries to download (and execute) the main-force code from any host in its peer list that has the complete code.

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

- 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

- When host B informs hosts in host A's peer list of the address tuple ⟨A's IP,B's IP⟩, it cannot cheat about its IP address due to the TCP connection
- 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 Figure 8 to fool the spearhead in host A).

Figure 8 The procedure in counterattacking dual-honeypot defense by an advanced two-stage reconnaissance worm (see online version for colours)



Notes: Host A is the bot under inspection.
Host C is in A's peer list that has the main-force code.

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 (Zou *et al.*, 2006), the worm can conduct the infection report shown in Figure 8 for global infection only, *i.e.*, host B is required to be far away from host A.

4.3 Honeypot detection time delay modelling and analysis

As described above, an infected host joins in a botnet only after it has executed the main-force code of the reconnaissance worm. Thus the botnet grows a step behind the propagation of the worm's spearhead. This time delay affects when the botmaster can use her botnet to conduct attacks, or when she can upgrade her botnet code. Thus botmasters may be interested in knowing this time delay. In addition, the time delay also affects the attack strength by a new-born botnet (some compromised machines have not joined in the botnet yet due to the honeypot detection procedure), thus security defenders may also be interested in knowing the delay time. In this section, we study the time delay caused by honeypot detection procedure. We present an analytical model for modelling the growth of a botnet as the two-stage reconnaissance worm spreads.

The modelling presented here tries to show that a two-stage worm will not slow down botnet construction, even though it adds a delay. The modelling results, as presented below, show that all infected computers (not including detected honey-pots) will join 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 modelled by the popular epidemic model as used in Nicol and Liljenstam (2004), Staniford *et al.* (2002), Zou *et al.* (2003), *etc.* Since worm modelling 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 Kesidis *et al.* (2005) and Zou *et al.* (2006) have presented modelling 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 modelled as Staniford *et al.* (2002) and Zou *et al.* (2003; 2006):

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

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

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

$$\frac{dI(t)}{dt} = \frac{\eta}{\Omega} I(t)[N - I(t)]. \tag{4}$$

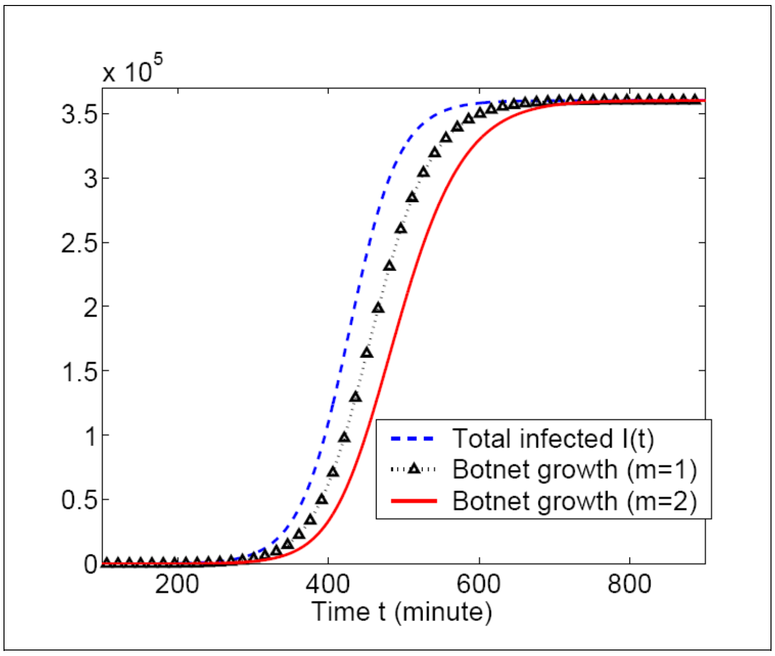
For a general two-stage reconnaissance worm that has $m > 1$, we can derive the botnet growth model in the similar way. For example, if $m = 2$, then we need to add an intermediate variable $I_1(t)$ to represent the number of spearhead-only infected hosts at time t – each of them has infected exactly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

$$\begin{aligned}
\frac{d\bar{I}(t)}{dt} &= \frac{\eta}{\Omega} I_1(t)N \\
\frac{dI_1(t)}{dt} &= \frac{\eta}{\Omega}[I(t) - I_1(t) - \bar{I}(t)]N - \frac{d\bar{I}(t)}{dt} \\
\frac{dI(t)}{dt} &= \frac{\eta}{\Omega} I(t)[N - I(t)].
\end{aligned} \tag{5}$$

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

We use Matlab Simulink (Simulink, 2009) to derive the numerical solutions of model (3) and model (5). We use the Code Red worm parameters (Zou *et al.*, 2002), $N = 360\,000$, $\eta = 358$/min, $\Omega = 2^{32}$ in the calculation and assume one initially infected host. Figure 9 shows the worm propagation and the botnet growth over time. The propagation speed relationship would be similar for any other set of worm parameters. 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 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.

Figure 9 Worm propagation and the constructed botnet growth (see online version for colours)



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 botnet introduced in this paper relies 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, therefore, is to invalidate this principle. For example, some national organisations or major security companies could set up limited-scale honeypot-based detection systems that are authorised 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; hence, some researchers or security defenders have deployed honeypots that freely send out malicious attacks. However, such honeypot defense practices are negligent and unethical. It will become illegal as the laws regarding cyberspace security and liability gradually mature.

The current popular GenII honeynet (Honeynet Project, 2005) has considered preventing attack traffic from being sent, but it does not implement this as strictly as the assumption used in the paper. First, it limits outgoing connection rate, thus it is possible that some early honeypot detection traffic could be sent out. Second, it can block or modify only *detected* outgoing malicious traffic, thus unknown malicious packets are possibly being sent out by honeypots. For this reason, the GenII honeynet might be able to avoid the honeypot detection conducted by attackers (as explained in Section 3); 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.

When botmasters deploy sensors to detect honeypots by checking test traffic, they rely on the fact that the identities of sensor machines are secret to honeypot defenders. Therefore, if security defenders could quickly figure out the identities of sensors before botmasters change their sensor machines (such as through binary code analysis), defenders' honeypots could avoid detection by allowing test traffic going out to those sensors.

A promising defense against honeypot-aware attacks is the "double-honeypot" idea (Tang and Chen, 2005). From Section 4.2, we can see that attackers need to take complicated 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 network traffic coming in from the 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 2).

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 botnet exposure to security professionals. If a botmaster 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 emphasise 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. Honeypot is a great tool to detect the infection vector and the source of internet attacks. It also provides an easy way in capturing attacking code to facilitate security analysis and signature generation.

6 Related work

Botnet is one of the major internet threats nowadays. There have been some systematic studies on general bots and botnets, such as Barford and Yegneswaran (2006), Dagon *et al.* (2007), Puri (2003), Trend Micro (2006) and Zhu *et al.* (2008). McCarty (2003) 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 a botnet (Bächer *et al.*, 2008; Cooke *et al.*, 2005; Freiling *et al.*, 2005). Behaving as normal 'bots' in the botnet, these honeypot spies provide valuable information of the monitored botnet activities. With the help from DDNS service providers, the second technique is to hijack bot controllers in botnets to monitor the command and control communications in botnets (Dagon *et al.*, 2006). 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 (2004) presented "honeyd", a honeypot software package that makes large-scale honeynet monitoring possible. Dagon *et al.* (2004) presented the "HoneyStat" system to use coordinated honeypots to detect worm infections in local networks. Jiang and Xu (2004) presented a virtual honeynet system that has a distributed presence and centralised operation. Bailey *et al.* (2004) 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.* (2005) presented several effective methods to design large-scale honeynet systems capable of obtaining high-fidelity attack data, which they called "Potemkin". Tang and Chen (2005) presented a novel "double-honeypot" detection system to effectively detect internet worm attacks. Anagnostakis *et al.* (2005) 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 (2004) discussed how to vividly simulate the routing topology and services of a virtual network by tailoring honeyd's response. GenII honeynets (Honeynet Project, 2005) allow a limited number of packets to be sent out from an infected honeynet. From the botmaster's perspective, some hardware or software specific means have always been available to detect infected honeypots (*e.g.* by detecting VMware (Hintz, 2002)) or another emulated virtual environment (Corey, 2004; Seifried, 2002), or by detecting the honeypot program's faulty responses (Honeyd, 2004). However, there has been no systematic research on honeypot detection based on a general methodology.

Krawetz (2004) 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 e-mail back to the spammer. This anti-honeypot tool uses the similar approach presented in this paper. It can be treated as a special realisation of the methodology presented here, but it is only effective for detecting open proxy honeypots.

Bethencourt *et al.* (2005) 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.

7 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, botmasters will find ways to avoid honeypot traps in their botnets. Botmasters can use software or hardware specific codes to detect the honeypot virtual environment (Corey, 2004; Honeyd, 2004; Seifried, 2002), 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 botmasters 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.

Acknowledgement

This work was supported by NSF Cyber Trust Grant CNS-0627318 and Intel Research Fund.

References

Anagnostakis, K., Sidiroglou, S., Akritidis, P., Xinidis, K., Markatos, E. and Keromytis, A. (2005) 'Detecting targeted attacks using shadow honeypots', *Proceedings of the 14th USENIX Security Symposium.*

Arce, I. and Levy, E. (2003) 'An analysis of the slapper worm', *IEEE Security & Privacy Magazine*, January–February.

Bächer, P., Holz, T., Kötter, M. and Wicherski, G. (2008) 'Know your enemy: tracking botnets', http://www.honeynet.org/papers/bots/.

Bailey, M., Cooke, E., Watson, D., Jahanian, F. and Provos, N. (2004) 'A hybrid honeypot architecture for scalable network monitoring', Tech. Rep. CSE-TR-499-04, U. Michigan.

Barford, P. and Yegneswaran, V. (2006) 'An inside look at botnets', *Malware Detection*, *in Series: Advances in Information Security*, pp.171–191.

Bethencourt, J., Franklin, J. and Vernon, M. (2005) 'Mapping internet sensors with probe response attacks', *Proceedings of the USENIX Security Symposium*, pp.193–208.

CERT (2001, 2003, 2004) 'CERT/CC advisories', http://www.cert.org/advisories/.

CodeRedII (2001) 'eEye digital security: CodeRedII worm analysis', http://www.eeye.com/html/Research/Advisories/AL20010804.html.

Cooke, E., Jahanian, F. and McPherson, D. (2005) 'The zombie roundup: understanding, detecting, and disrupting botnets', *Proceedings of the Steps to Reducing Unwanted Traffic on the Internet Workshop.*

Corey, J. (2004) 'Advanced honey pot identification and exploitation', http://www.phrack.org/fakes/p63/p63-0x09.txt.

Dagon, D., Gu, G., Lee, C. and Lee, W. (2007) 'A taxonomy of botnet structures', *Proceedings of the 23rd Annual Computer Security Applications Conference*, pp.325–339.

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

Dagon, D., Zou, C. and Lee, W. (2006) 'Modeling botnet propagation using time zones', *Proceedings of the 13th Annual Network and Distributed System Security Symposium*, pp.235–249.

Drake, C., Oliver, J. and Koontz, E. (2004) 'Mailfrontier, Inc. whitepaper: anatomy of a phishing email', http://www.ceas.cc/papers-2004/114.pdf.

Even, L.R. (2000) 'Honey pot systems explained', http://www.sans.org/resources/idfaq/honeypot3.php.

Freiling, F., Holz, T. and Wicherski, G. (2005) 'Botnet tracking: exploring a root-cause methodology to prevent distributed denial-of-service attacks', Tech. Rep. AIB-2005-07, CS Dept. of RWTH Aachen University.

Hintz, A. (2002) '4tphi: detecting VMWare', http://seclists.org/honeypots/2002/q4/0029.html.

Honeyd (2004) 'Honeyd security advisory 2004-001: remote detection via simple probe packet', http://www.honeyd.org/adv.2004-01.asc.

Honeynet Project (2005) 'Know your enemy: GenII honeynets', http://www.honeynet.org/papers/gen2.

Jiang, X. and Xu, D. (2004) 'Collapsar: a VM-based architecture for network attack detention center', *Proceedings of the 13th USENIX Security Symposium.*

Kesidis, G., Hamadeh, I. and Jiwasurat, S. (2005) 'Coupled Kermack-McKendrick models for randomly scanning and bandwidth-saturating internet worms', *Proceedings of the 3rd International Workshop on QoS in Multiservice IP Networks*, pp.101–109.

Krawetz, N. (2004) 'Anti-honeypot technology', *IEEE Security & Privacy Magazine*, Vol. 2, No. 1.

Kreibich, C., Warfield, A., Crowcroft, J., Hand, S. and Pratt, I. (2005) 'Using packet symmetry to curtail malicious traffic', *Proceedings of the 4th Workshop on Hot Topics in Networks (HotNets-IV).*

Lemos, R. (2006) 'Bot software looks to improve peerage', http://www.securityfocus.com/news/11390.

Levine, J., LaBella, R., Owen, H., Contis, D. and Culver, B. (2003) 'The use of honeynets to detect exploited systems across large enterprise networks', *Proceedings of the IEEE Workshop on Information Assurance.*

McCarty, B. (2003) 'Botnets: big and bigger', *IEEE Security & Privacy Magazine*, Vol. 1.

Nicol, D. and Liljenstam, M. (2004) 'Models of active worm defenses', *Proceedings of the IPSI Studenica Conference.*

Provos, N. (2004) 'A virtual honeypot framework', *Proceedings of the 13th USENIX Security Symposium.*

Puri, R. (2003) 'Bots & botnet: an overview', http://www.sans.org/rr/whitepapers/malicious/1299.php.

Rajab, M., Zarfoss, J., Monrose, F. and Terzis, A. (2007) 'My botnet is bigger than yours (maybe, better than yours): why size estimates remain challenging', *Proceedings of the USENIX Workshop on Hot Topics in Understanding Botnets.*

Salgado, R. (2005) 'The legal ramifications of operating a honeypot', *IEEE Magazine on Security and Privacy*, Vol. 1.

Seifried, K. (2002) 'Honeypotting with VMware basics', http://www.seifried.org/security/index.php/Honeypotting_With_VMWare_Basics.

Simulink (2009) 'Mathworks Inc.: Simulink', http://www.mathworks. com/products/simulink.

Snort (2006) 'Snort users manual', http://www.snort.org/docs/snort htmanuals/htmanual 260/.

Spitzner, L. (2003) 'Honeypots: are they illegal?', http://www.securityfocus.com/infocus/1703.

Staniford, S., Paxson, V. and Weaver, N. (2002) 'How to own the internet in your spare time', *Proceedings of the USENIX Security Symposium*, pp.149–167.

Stewart, J. (2003) 'Sinit P2P trojan analysis', http://www.secureworks.com/research/threats/sinit/.

Stewart, J. (2004) 'Phatbot trojan analysis', http://www.secureworks.com/research/threats/phatbot/.

Tang, Y. and Chen, S. (2005) 'Defending against internet worms: a signature-based approach', *Proceedings of the IEEE INFOCOM.*

Trend Micro (2006) 'Taxonomy of botnet threats', *Technical report*, Trend Micro White paper, pp.235–249.

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

Vogt, R., Aycock, J. and Jacobson, M. (2007) 'Army of botnets', *Proceedings of the 14th Annual Network and Distributed System Security Symposium.*

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

Wang, P., Sparks, S. and Zou, C. (2007) 'An advanced hybrid peer-to-peer botnet', *Proceedings of the USENIX Workshop on Hot Topics in Understanding Botnets.*

Zhu, Z., Lu, G., Chen, Y., Fu, Z.J., Roberts, P. and Han, K. (2008) 'Modeling botnet propagation using time zones', *Proceedings of the 32nd Annual IEEE International Computer Software and Applications*, pp.967–972.

Zou, C., Gao, L., Gong, W. and Towsley, D. (2003) 'Monitoring and early warning for internet worms', *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pp.190–199.

Zou, C., Gong, W. and Towsley, D. (2002) 'Code red worm propagation modeling and analysis', *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pp.138–147.

Zou, C., Towsley, D. and Gong, W. (2006) 'On the performance of internet worm scanning strategies', *Journal of Performance Evaluation*, Vol. 63.

Note

1 New version of Snort has integrated the functionality of Snort_inline. What is running on a honeywall is actually Snort. But we still use Snort_inline in the paper in order to differentiate Snort_inline from other functionalities of Snort.