An Advanced Hybrid Peer-to-Peer Botnet

Ping Wang, Sherri Sparks, and Cliff C. Zou *Member, IEEE,*

Abstract—A “botnet” consists of a network of compromised computers controlled by an attacker (“botmaster”). Recently botnets have become the root cause of many Internet attacks. To be well prepared for future attacks, it is not enough to study how to detect and defend against the botnets that have appeared in the past. More importantly, we should study advanced botnet designs that could be developed by botmasters in the near future. In this paper, we present the design of an advanced hybrid peer-to-peer botnet. Compared with current botnets, the proposed botnet is harder to be shut down, monitored, and hijacked. It provides robust network connectivity, individualized encryption and control traffic dispersion, limited botnet exposure by each bot, and easy monitoring and recovery by its botmaster. In the end, we suggest and analyze several possible defenses against this advanced botnet.

Index Terms—Botnet, peer-to-peer, robustness, honeypot

1 Introduction

In the last several years, Internet malware attacks have evolved into better organized and more profit-centered endeavors. Email spam, extortion through denial-of-service attacks [1], and click fraud [2] represent a few examples of this emerging trend. “Botnets” are a root cause of these problems [3], [4], [5]. A “botnet” consists of a network of compromised computers (“bots”) connected to the Internet that is controlled by a remote attacker (“botmaster”) [5], [6]. Since a botmaster could scatter attack tasks over hundreds or even tens of thousands of computers distributed across the Internet, the enormous cumulative bandwidth and large number of attack sources make botnet-based attacks extremely dangerous and hard to defend against.

Compared to other Internet malware, the unique feature of a botnet lies in its control communication network. Most botnets that have appeared until now have had a common centralized architecture. That is, bots in the botnet connect directly to some special hosts (called “*command-and-control*” servers, or “C&C” servers). These C&C servers receive commands from their botmaster and forward them to the other bots in the network. From now on we will call a botnet with such a control communication architecture a “C&C botnet”. Fig. 1 shows the basic control communication architecture for a typical C&C botnet (in reality, a C&C botnet usually has more than two C&C servers). Arrows represent the directions of network connections.

As botnet-based attacks become popular and dangerous, security researchers have studied how to detect, monitor, and defend against them [1], [3], [4], [5], [6], [7]. Most of the current research has focused upon the C&C botnets that have appeared in the past, especially Internet Relay Chat (IRC) based botnets. It is necessary to conduct such research in order to deal with the threat we are facing today. However, it is equally important to conduct research on advanced botnet designs that could be developed by attackers in the near future. Otherwise, we will remain susceptible to the next generation of internet malware attacks.

From a botmaster’s perspective, the C&C servers are the fundamental weak points in current botnet architectures. First, a botmaster will lose control of her botnet once the limited number of C&C servers are shut down by defenders. Second, defenders could easily obtain the identities (e.g., IP addresses) of all C&C servers based on their service traffic to a large number of bots [7], or simply from one single captured bot (which contains the list of C&C servers). Third, an entire botnet may be exposed once a C&C server in the botnet is hijacked or captured by defenders [4]. As network security practitioners put more resources and effort into defending against botnet attacks, hackers will develop and deploy the next generation of botnets with a different control architecture.

1.1 Current P2P Botnets and Their Weaknesses

Considering the above weaknesses inherent to the centralized architecture of current C&C botnets, it is a natural strategy for botmasters to design a peer-to-peer (P2P) control mechanism into their botnets. In the last several years, botnets such as Slapper [8], Sinit [9], Phatbot [10] and Nugache [11] have implemented different kinds of P2P control architectures. They have shown several advanced designs. For example, some of them have removed the “bootstrap” process used in common P2P protocols[1]. Sinit uses public key cryptography for update authentication [9]. Nugache attempts to thwart detection by implementing an encrypted/obsfucated control channel [11].

Nevertheless, simply migrating available P2P protocols will not generate a sound botnet, and the P2P designs used by several botnets in the past are not mature and have many weaknesses. To remove bootstrap procedure, a Sinit bot uses random probing to find other Sinit bots to communicate with. This results in poor connectivity for the constructed botnet and

• *Ping Wang, Sherri Sparks, and Cliff C. Zou are with School of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL 32816*
E-mail: {pwang, ssparks, czou}@cs.ucf.edu

1. Many peer-to-peer networks have a central server or a seed list of peers who can be contacted in order for a new peer joining the network. This bootstrap process is not a major problem for normal P2P networks, but it poses a single point of failure for a P2P botnet.

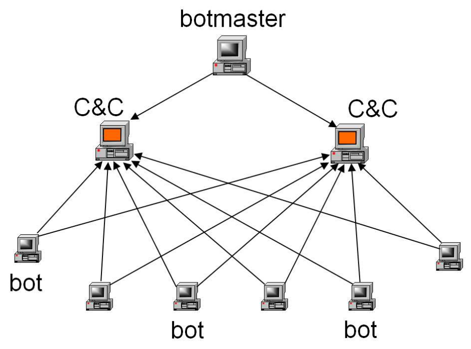


Fig. 1. Command and control architecture of a C&C botnet

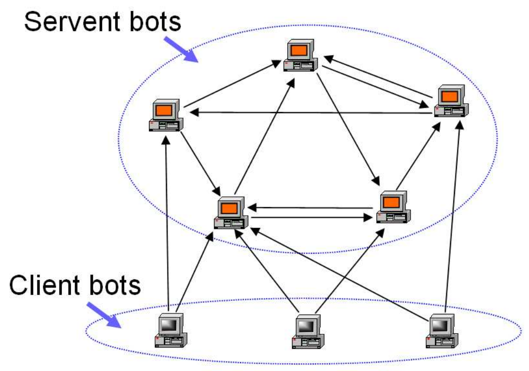


Fig. 2. Command and control architecture of the proposed hybrid P2P botnet

easy detection due to the extensive probing traffic [9]. Phatbot utilizes Gnutella cache servers for its bootstrap process. This botnet can be easily shut down if security community set up filter on those Gnutella cache servers, or block any traffic to and from those cache servers. In addition, its underlying WASTE peer-to-peer protocol is not scalable across a large network [10]. Nugache's weakness lies in its reliance on a seed list of 22 IP addresses during its bootstrap process [11]. Slapper fails to implement encryption and command authentication enabling it to be easily hijacked by others. In addition, its list of known bots contains all (or almost all) members of the botnet. Thus, one single captured bot would expose the entire botnet to defenders [8]. Furthermore, its complicated communication mechanism generates a large amount of traffic, rendering it susceptible to monitoring via network flow analysis.

Some other available distributed systems include "censorship-resistant" system and "anonymous" P2P system. However, their design goal is different from a botnet. For example, these distributed systems try to hide the source node of a message within a crowd of nodes. However, they do not bother to hide the identities of this crowd. On the other hand, a botnet needs to try its best to hide IP addresses of all bots in it.

1.2 Proposed Hybrid P2P Botnet

Considering the problems encountered by C&C botnets and previous P2P botnets, the design of an advanced botnet, from our understanding, should consider the following practical challenges faced by botmasters: (1). How to generate a robust botnet capable of maintaining control of its remaining bots even after a substantial portion of the botnet population has been removed by defenders? (2). How to prevent significant exposure of the network topology when some bots are captured by defenders? (3). How to easily monitor and obtain the complete information of a botnet by its botmaster? (4). How to prevent (or make it harder for) defenders from detecting bots via their communication traffic patterns? In addition, the design should also consider many network related issues such as dynamic or private IP addresses and the diurnal online/offline property of bots [4].

By considering all the challenges listed above, in this paper, we present our research on the possible design of an advanced hybrid P2P botnet. The proposed hybrid P2P botnet has the following features:

- The botnet requires no bootstrap procedure.
- The botnet communicates via the peer list contained in each bot. However, unlike Slapper [8], each bot has a fixed and limited size peer list and does not reveal its peer list to other bots. In this way, when a bot is captured by defenders, only the limited number of bots in its peer list are exposed.
- A botmaster could easily monitor the entire botnet by issuing a *report* command. This command instructs all (or partial) bots to report to a compromised machine (which is called a *sensor host*) that is controlled by the botmaster. The IP address of the sensor host, which is specified in the report command, will change every time a report command is issued to prevent defenders from capturing or blocking the sensor host beforehand.
- After collecting information about the botnet through the above report command, a botmaster, if she thinks necessary, could issue an *update* command to actively let all bots contact a sensor host to update their peer lists. This effectively updates the botnet topology such that it has a balanced and robust connectivity, and/or reconnects a broken botnet.
- Only bots with static global IP addresses that are accessible from the Internet are candidates for being in peer lists (they are called *servent bots* according to P2P terminologies [12] since they behave with both client and server features). This design ensures that the peer list in each bot has a long lifetime.
- Each servent bot listens on a self-determined service port for incoming connections from other bots and uses a self-generated symmetric encryption key for incoming traffic. This individualized encryption and individualized service port design makes it very hard for the botnet to be detected through network flow analysis of the botnet communication traffic.

1.3 Paper Organization

The rest of the paper is organized as follows. Section 2 introduces related studies. Section 3 introduces the control communication architecture of the proposed botnet. Section 4 discusses the designs to ensure authentication, security, and traffic dispersion of command communication. In Section 5, we present how a botmaster is able to monitor her botnet reliably and easily. We present how to construct the proposed botnet in Section 6 and study its robustness against defense in Section 7. In Section 8, we present possible defenses against the botnet, and provide simulation studies and performance analytical models of several defense themes. We give a few discussions in Section 9 and finally conclude the paper in Section 10.

2 Related Work

Botnets are an active research topic in recent years. In 2003, Puri [13] presented an overview of bots and botnets, and McCarty [14] discussed how to use a honeynet to monitor botnets. Arce and Levy presented a good analysis of how the Slapper worm built its P2P botnet. Barford and Yegneswaran [15] gave a detailed and systematic dissection of many well-known botnets that have appeared in the past.

Current research on botnets is mainly focused on monitoring and detection. [3], [6], [16], [17] presented comprehensive studies on using honeypots to join botnets in order to monitor botnet activities in the Internet. With the help from Dynamic DNS service providers, [4] presented a botnet monitoring system by redirecting the DNS mapping of a C&C server to a botnet monitor. Ramachandran et al. [5] presented how to *passively* detect botnets by finding botmasters' queries to spam DNS-based blackhole list servers (DNSBL).

Since most botnets nowadays use Internet Relay Chat (IRC) for their C&C servers, many people have studied how to detect them by detecting their IRC channels or traffic. Binkley and Singh [7] attempted to detect them through abnormal IRC channels. Strayer [18] used machine-learning techniques to detect botnet IRC-based control traffic and tested the system on trace-driven network data. Chen [19] presented a system to detect botnet IRC traffic on high-speed network routers.

Nevertheless, few people have studied how botmasters might improve their attack techniques. [8], [9], [10], [11], [15] only introduced the attack techniques already implemented in several botnets appearing in the past. Zou and Cunningham [20] studied how botmasters might improve their botnets to avoid being monitored by a honeypot. Our research presented in this paper belongs to this category.

Our research is conducted at the same time and independent with the work done by Vogt et al. [21]. In [21], the authors presented a "super-botnet", which is a super-size botnet by inter-connecting many small botnets together in a peer-to-peer fashion. However, [21] largely ignored two important practical issues, both of which have been addressed in our work: (1). The majority of compromised computers cannot be used as C&C servers since they are either behind firewall, behind NAT, or have dynamic IP addresses; (2). The robust botnet topology cannot be set up solely through reinfection mechanism, if a botnet does not have substantive reinfections during its build-up, which is the case for most botnets in reality.

3 Proposed Hybrid P2P Botnet Architecture

3.1 Two Classes of Bots

The bots in the proposed P2P botnet are classified into two groups. The first group contains bots that have static, non-private IP addresses and are accessible from the global Internet. Bots in the first group are called *servent bots* since they behave as both clients and servers[2]. The second group contains the remaining bots, including: (1). Bots with dynamically allocated IP addresses; (2). Bots with private IP addresses; (3). Bots behind firewalls such that they cannot be connected from the global Internet. The second group of bots are called *client bots* since they will not accept incoming connections.

Only servent bots are candidates in peer lists. All bots, including both client bots and servent bots, actively contact the servent bots in their peer lists to retrieve commands. Because servent bots normally do not change their IP addresses, this design increases the network stability of a botnet. This bot classification will become more important in the future as a larger proportion of computers will sit behind firewall, or use DHCP or private IP addresses due to shortage of IP space.

A bot could easily determine the type of IP address used by its host machine. For example, on a Windows machine, a bot could run the command "`ipconfig /all`". Not all bots with static global IP addresses are qualified to be servent bots—some of them may stay behind firewall, inaccessible from the global Internet. A botmaster could rely on the collaboration between bots to determine such bots. For example, a bot runs its server program and requests the servent bots in its peer list to initiate connections to its service port. If the bot could receive such test connections, it labels itself as a servent bot. Otherwise, it labels itself as a client bot.

3.2 Botnet Command and Control Architecture

Fig. 2 illustrates the command and control architecture of the proposed botnet. The illustrative botnet shown in this figure has 5 servent bots and 3 client bots. The peer list size is 2 (i.e. each bot's peer list contains the IP addresses of 2 servent bots). An arrow from bot A to bot B represents bot A initiating a connection to bot B. This figure shows that a big cloud of servent bots interconnect with each other—they form the backbone of the control communication network of a botnet.

A botmaster injects her commands through any bot(s) in the botnet. Both client and servent bots periodically connect to the servent bots in their peer lists in order to retrieve commands issued by their botmaster. When a bot receives a new command that it has never seen before (e.g., each command has a unique ID), it immediately forwards the command to all servent bots in its peer list. In addition, if itself is a servent bot, it will also forward the command to any bots connecting to it.

2. In a traditional peer-to-peer file sharing system, all hosts behave both as clients and servers and are called "servents" [22].

This description of command communication means that, in terms of command forwarding, the proposed botnet has an undirected graph topology. A botmaster's command could pass via the links shown in Fig. 2 in both directions. If the size of the botnet peer list is denoted by M, then this design makes sure that each bot has at least M venues to receive commands.

3.3 Relationship Between Traditional C&C Botnets and the Proposed Botnet

Compared to a C&C botnet (see Fig. 1), it is easy to see that the proposed hybrid P2P botnet shown in Fig. 2 is actually an extension of a C&C botnet. The hybrid P2P botnet is equivalent to a C&C botnet where servent bots take the role of C&C servers: the number of C&C servers (servent bots) is greatly enlarged, and they interconnect with each other. Indeed, the large number of servent bots is the primary reason why the proposed hybrid P2P botnet is very hard to be shut down. We will explain these properties in detail later in Section 6 and Section 7.

4 Botnet Command and Control

The essential component of a botnet is its command and control communication. Compared to a C&C botnet, the proposed botnet has a more robust and complex communication architecture. The major design challenge is to generate a botnet that is difficult to be shut down, or monitored by defenders or other attackers.

4.1 Command Authentication

Compared with a C&C botnet, because bots in the proposed botnet do not receive commands from predefined places, it is especially important to implement a strong command authentication. A standard public-key authentication would be sufficient. A botmaster generates a pair of public/private keys, $\langle K^+, K^- \rangle$, and hard codes the public key K^+ into the bot program before releasing and building the botnet. There is no need for key distribution because the public key is hard-coded in bot program. Later, the command messages sent from the botmaster could be digitally signed by the private key K^- to ensure their authentication and integrity.

This public-key based authentication could also be readily deployed by current C&C botnets. So botnet hijacking is not a major issue.

4.2 Individualized Encryption Key

A botmaster may also wish to encrypt her command messages to prevent being eavesdropped by defenders or other attackers. The peer-list based architecture of the proposed P2P botnet makes it easy to implement a strong encryption.

In the proposed botnet, each servent bot i randomly generates its symmetric encryption key K_i. Suppose the peer list on bot A is denoted by L_A. It will not only contain the IP addresses of M servent bots, but also the symmetric keys used by these servent bots. Thus, the peer list on bot A is:

$$L_A = \{(\text{IP}_{i_1}, K_{i_1}), (\text{IP}_{i_2}, K_{i_2}), \cdots (\text{IP}_{i_M}, K_{i_M})\} \qquad (1)$$

where $(\text{IP}_{i_j}, K_{i_j})$ are the IP address and symmetric key used by servent bot i_j. With such a peer list design, each servent bot uses its own symmetric key for incoming connections from any other bot. This is applicable because if bot B connects to a servent bot A, bot B must have (IP_A, K_A) in its peer list.

This *individualized* encryption guarantees that if defenders capture one bot, they only obtain keys used by M servent bots in the captured bot's peer list. Thus the encryption among the remaining botnet will not be compromised.

4.3 Individualized Service Port

The peer-list based architecture also enables the proposed botnet to disperse its communication traffic in terms of service port. Since a servent bot needs to accept connections from other bots, it must run a server process listening on a service port. The service port number on servent bot i, denoted by P_i, could be picked by the bot, either randomly or selectively. Considering this, a peer list needs to contain the service port information as well. For example, the peer list on bot A is:

$$L_A = \{(\text{IP}_{i_1}, K_{i_1}, P_{i_1}), \cdots, (\text{IP}_{i_M}, K_{i_M}, P_{i_M})\} \qquad (2)$$

With the new peer list L_A shown above, bot A can connect to any servent bot in its peer list using the correct service port without any difficulty.

This individualized service port design has two benefits for botmasters:

- **Dispersed network traffic:** Since service port is a critical parameter in classifying network traffic, this individualized port design makes it extremely hard for defenders to detect a botnet based on monitored network traffic. When combined with the individualized encryption design, a P2P botnet has a strong resistance against most (if not all) network traffic flow based detection systems, such as the ones introduced in [18], [19].
- **Secret backdoor:** The individualized port design also ensures that servent bots in a P2P botnet keep their backdoors "secret". Otherwise, defenders could scan the specific port used by a botnet to detect potential servent bots, or monitor network traffic targeting this service port to facilitate their botnet detection.

As we mentioned above, the service port number can be chosen either randomly or selectively by each bot. A randomly-generated service port may not be good for botnets since network traffic going to a rarely used port is abnormal. Thus a more realistic approach is that each servent bot selectively picks its service port by choosing one standard encryption port, such as port 22 (SSH), 443 (HTTPS), 993 (IMAPS), to facilitate the encrypted botnet communication traffic and also masquerade as normal traffic. Furthermore, a sophisticated botmaster could even program bot code to mimic the protocol used on a standard service port. This is not difficult as it has already been implemented in the open source program "honeyd" [23].

The individualized service port makes a botnet communication harder to detect, but it does not mean that a servent bot cannot be detected based on its botnet traffic. If a local network's firewall or its security administrator keeps track of what services are provided by each local machine, a bot-infected computer that does not host any standard encryption service can be easily detected. In addition, if the network firewall or the security administrator tracks the client pool profile for each service, even if a local machine does host some encryption services, the bot on this machine can still be detected when the client pool significantly deviates from its normal profile.

5 Botnet Monitoring by its Botmaster

Another major challenge in botnet design is making sure that a botnet is difficult to monitor by defenders, but at the same time, easily monitored by its botmaster. With detailed botnet information, a botmaster could (1). Conduct attacks more effectively according to the bot population, distribution, on/off status, IP address types, etc; (2). Keep tighter control over the botnet when facing various counterattacks from defenders. In this section, we present a simple but effective way for botmasters to monitor their botnets whenever they want, and at the same time, resist being monitored by others.

5.1 Monitoring Via a Dynamically Changeable Sensor

To monitor the proposed hybrid P2P botnet, a botmaster issues a special command, called a *report* command, to the botnet thereby instructing every bot to send its information to a specified machine that is compromised and controlled by the botmaster. This data collection machine is called a *sensor*.

The IP address (or domain name) of the centralized sensor host is specified in the report command. Every round of report command issued by a botmaster could potentially utilize a different sensor host. This would prevent defenders from knowing the identity of the sensor host before seeing the actual report command. After a report command has been sent out by a botmaster, it is possible that defenders could quickly know the identity of the sensor host (e.g., through honeypot joining the botnet [3], [6]), and then either shut it down or monitor the sensor host. To deal with this threat, a botmaster may implement any of the following procedures:

- Use a popular Internet service, such as HTTP or Email, for report to a sensor. The sensor is chosen such that it normally provides such a service to avoid exhibiting abnormal network traffic.
- Use several sensor machines instead of a single sensor.
- Select sensor hosts that are harder to be shut down or monitored, for example, compromised machines in other countries with minimum Internet security and International collaboration.
- Manually verify the selected sensor machines are not honeypots (see further discussion in Section 9).
- Wipe out the hard drive on a sensor host immediately after retrieving the report data.
- Specify expiration time in report command to prevent any bot exposing itself after that time.
- Issue another command to the botnet to cancel the previous report command once the botmaster knows that the sensor host has been captured by defenders.

If a botmaster simply wants to know the current size of a botnet, a probabilistic report would be preferred: each bot uses a small probability p specified in a report command to decide whether to report. Then the botnet has roughly X/p bots if X bots report. Such a probabilistic report could minimize the telltale traffic to the report sensor.

Each bot could use the public key K^+ (hard-coded in the bot program) to ensure the confidentiality of its report data—only the botmaster can read the report by using the corresponding private key K^-. In addition, a botmaster could use several compromised machines as stepping stones when retrieving data from sensors. These are standard practices so we will not explain more.

5.2 Additional Monitoring Information

A botmaster not only wants to know a botnet size and topology, she may also want to know other information in order to conduct efficient attacks.

5.2.1 IP address type

Internet computers are identified by their IP addresses. But the widely-deployed "Dynamic Host Configuration Protocol" (DHCP) and "Network Address Translation" (NAT) have made IP-based identification difficult and error-prone. A botmaster may implement an ID-based identification (such as [24]) to track bots in a botnet: when compromising a computer, the bot program on the computer randomly generates its unique ID. When bots send their report to a sensor host, they report their IDs as well.

This ID-based identification facilitates the measurement of NAT encountered by a botnet. Based on a round of report sent from bots, a botmaster is able to know whether several bots are behind a single NAT — these bots have different IDs but send reports from a single source IP address.

This ID-based identification eliminates the measurement trouble caused by DHCP addresses. A more useful measurement is to know how frequently DHCP-based bots in a botnet change their IP addresses. For this purpose, each bot with DHCP address keeps recording when its IP address changes, and then report this information to its botmaster's sensor host.

This information is particularly useful for botmasters in sending email spam. As pointed out by [25], 80% of current spam is listed in some DNS blacklists (DNSBLs), which are used "to track IP addresses that originate spam, so that future emails sent from these IP addresses can be rejected." [5] This means that if a botnet sends out spam, many bots in the botnet will lose their capability to send out spam again. To counterattack this defense, a botmaster may only use DHCP bots, which change their IP addresses, for example, at least once per day, to send out email spam. In this way, defenders need to blacklist a much larger number of IPs to effectively block spam, and it becomes much harder for defenders to determine a good timeout value for blocked IPs.

Because a botmaster knows how many bots satisfy this requirement based on botnet report, the botmaster can strike a good tradeoff between sending out enough spam and avoiding being blocked by DNSBLs.

The ID-based identification has another benefit: if a botmaster wants to let a selected set of bots to send out an attack, the botmaster may issue a command to the botnet with a list of these attack bots' IDs — a bot will launch attack if it finds its ID in this list. In this way, if the list is captured by defenders, it will not reveal the identities of these attack bots since only the botmaster knows the mapping between IP addresses and IDs.

5.2.2 Diurnal dynamics

As pointed out in [4], the online population of every botnet has a clear "diurnal" dynamics due to many users shutting down their computers at night. In one time zone, the peak online population of a botnet could be as much as four times of the bottom level online population. The significance of diurnal dynamics is further verified by [26], which showed that only about 20% of computers are always online.

To maximize a botnet attack power, a botmaster may want to know the diurnal dynamics of her botnet. For example, a botmaster can launch a denial-of-service attack at the right time when the botnet online population reaches its peak level, or spread a new malware at the optimal release time to increase its propagation speed as introduced in [4].

The diurnal dynamics of each bot is not hard to obtain since a bot is in fact a spyware. For example, at the beginning of each hour, a running bot program appends the current time to a data file, which is then reported to its botmaster. Based on such report data, a botmaster can derive the accurate diurnal dynamics of each bot.

6 Botnet Construction

Unlike a traditional C&C-based botnet, the proposed hybrid P2P botnet does not have a pre-fixed communication architecture. Its network connectivity is solely determined by the peer list in each bot. We will introduce the peer list construction procedure in this section.

Botnets utilize many different infection mechanisms, such as vulnerability exploitation, email viruses, traditional file-based viruses, network share, etc. The botnet construction procedure introduced in this paper is applicable to all infection mechanisms.

6.1 Basic construction procedure

A natural way to build peer lists is to construct them as a botnet propagates. To make sure that a constructed botnet is connected, the initial set of bots should contain some servent bots whose IP addresses are in the peer list in every initial bot. Suppose the size of peer list in each bot is configured to be M. As a bot program propagates, the peer list in each bot is constructed according to the following procedure:

- **New infection:** Bot A passes its peer list to a vulnerable host B when compromising it. if A is a servent bot, B adds A into its peer list (by randomly replacing one entry if its peer list is full). If A knows that B is a servent bot (A may not be aware of B's identity, for example, when B is compromised by an email virus sent from A), A adds B into its peer list in the same way.
- **Reinfection:** If reinfection is possible and bot A reinfects bot B, bot B will then replace R ($R \leq M - 1$) randomly-selected bots in its peer list with R bots from the peer list provided by A. Again, bot A and B will add each other into their respective peer lists if the other one is a servent bot as explained in the above "new infection" procedure.

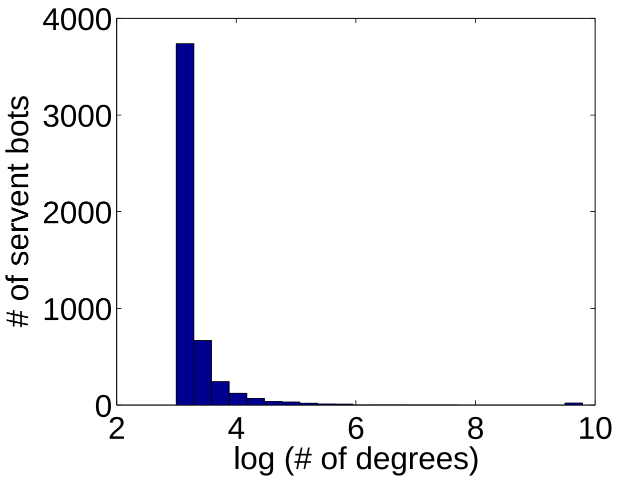
When reinfection happens frequently, the reinfection procedure can effectively interconnect different infection paths together, making a botnet evenly connected. In addition, this procedure makes it hard for defenders to infer the infection time order ("traceback") among bots based on captured peer lists.

In the reinfection procedure, a bot does not provide its peer list to those who reinfect it. This is important, because, if not, defenders could *recursively* infect (and monitor) all servent bots in a botnet based on a captured bot in their honeypot in the following way: Defenders use a firewall redirecting the outgoing infection attempts from captured bot A to reinfect the servent bots in A's peer list; then subsequently get the peer lists from these servent bots and reinfect servent bots in these peer lists in turn.

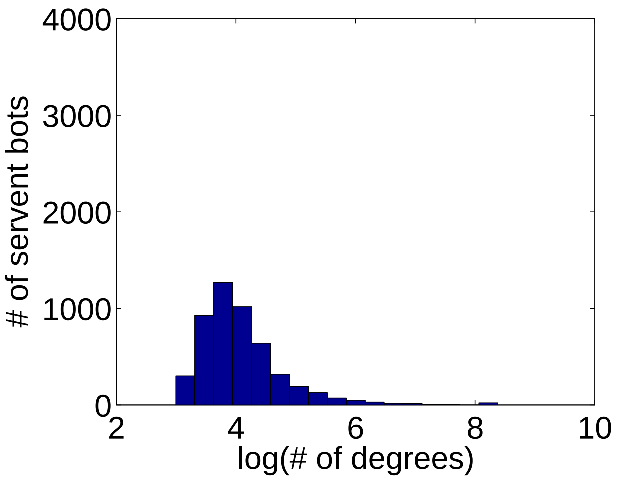
In order to study a constructed botnet topology and its robustness via simulations, we first need to determine simulation settings. First, Bhagwan et al. [24] studied P2P file sharing systems and observed that around 50% of computers change their IP addresses within four to five days. So we expect the fraction of bots with dynamic addresses is around the similar range. In addition, some other bots are behind firewalls or NAT boxes so that they cannot accept Internet connections. We cannot find a good source specifying this statistics, so in this paper we assume that 25% of bots are servent bots.

Second, as pointed out in [27], [28], botnets in recent years have dropped their sizes to an average of 20,000, even though the potential vulnerable population is much larger. Thus we assume a botnet has a potential vulnerable population of 500,000, but stops growing after it reaches the size of 20,000. In addition, we assume that the peer list has a size of $M = 20$ and that there are 21 initial servent hosts to start the spread of the botnet. In this way, the peer list on every bot is always full.

Because scanning and vulnerability exploit is the dominant infection mechanism used by current botnets, in this paper we simulate the construction of a botnet by assuming that the bot code finds and compromises vulnerable computers in the similar way as what a scanning worm does. Fig. 3(a) shows the degree distribution for servent bots (client bots always have a degree M, equal to the size of peer list) after the botnet has accumulated 20,000 members. Because the botnet stops growing when it reaches the size of 20,000, the reinfection events rarely happen (only around 600). For this reason, connections to servent bots are extremely unbalanced: more than 80% (4000) of servent bots have degrees less than 30, while each of the 21 initial servent bots have a degree between 14,000 and 17,500 (the last tiny bar at the bottom right corner of the figure close to X-axis value of 10 represents these 21 servent bots). This is not an ideal botnet. The constructed hybrid P2P botnet is approximately degraded to a C&C botnet where the initial set of servent bots behave as C&C servers.



(a). Vulnerable population $500,000$



(b). Vulnerable population $20,000$

Fig. 3. Servent bot degree distribution (construct botnet via “new infection” and “reinfection” procedure only)

Vogt et al. [21] constructed a super-botnet only with the algorithms that are similar to the “new infection” and “re-infection” procedure presented above. Although authors in [21] showed that their constructed super-botnet is robust, they have an implicit assumption that the super-botnet will have abundant reinfections during its construction period. We believe this assumption is incorrect in a real world scenario—botmasters would want their botnets generating as few as possible reinfections to avoid wasting infection power and being detected by defenders.

To illustrate this argument, we have simulated another botnet scenario where the potential vulnerable population is 20,000 instead of 500,000 used in the previous simulation. The botnet stops propagation after all vulnerable hosts have been infected. Fig. 3(b) shows the degree distribution for servent bots in this scenario. When the botnet stops infection process, overall around 210,000 reinfection events happened. This time, because there are plenty of reinfections, the constructed botnet has a well-balanced connectivity—the degree distribution of all servent bots roughly follows normal distribution, and 80% of servent bots have degrees between 30 and 150.

These simulation experiments show that if a botnet does not have a lot of reinfections, or cannot have reinfection (for example, when the bot program blocks the vulnerable service on an infected host), the aforementioned basic construction procedure is not effective. A botmaster must come up with an additional botnet construction procedure, which is introduced in the following.

6.2 Advanced construction procedure

One intuitive way to improve the network connectivity would be letting bots keep exchanging and updating their peer lists frequently. However, such a design makes it very easy for defenders to obtain the identities of all servent bots, if one or several bots are captured by defenders.

As introduced in Section 5, a botmaster could monitor her botnet easily whenever she wants by issuing a report command. With the detailed botnet information, a botmaster could easily update the peer list in each bot to have a strong and balanced connectivity. The added new procedure is:

- **Peer-list updating:** After a botnet spreads out for a while, a botmaster issues a report command to obtain the information of all currently available servent bots. These servent bots are called *peer-list updating servent bots*. Then, the botmaster issues another command, called *update* command, enabling all bots to obtain an updated peer list from a specified sensor host. The sensor host randomly chooses M servent bots to compose an updated peer list, then sends it back to each requested bot.

A botmaster could run this procedure once or a few times during or after botnet propagation stage. After each run of this procedure, all current bots will have uniform and balanced connections to peer-list updating servent bots.

From a botmaster’s point of view, when and how often should this peer-list updating procedure be run? First, this procedure should be executed once shortly after the release of a botnet to prevent defenders from removing all initial servent bots—before the first peer-list updating procedure, the P2P botnet is as vulnerable as current C&C-based botnets. Second, as a botnet spreads out, each round of this updating procedure makes the constructed botnet have a stronger and more balanced connectivity, but at the same time, it incurs an increasing risk of exposing the botnet to defenders. It is therefore up to a botmaster to strike a comfortable balance. In addition, a botmaster could run this procedure to conveniently update the topology of a botnet, or reconnect a broken botnet.

Fig. 4 shows the degree distribution for servent bots (client bots always have a degree of M) when a botnet uses all three construction methods. We assume the peer-list updating procedure is executed just once when 1,000 (25% of) servent bots have been infected. This figure shows that in terms of network topology, the servent bots in the botnet can be classified into two groups: the first 1000 servent bots used in peer-list updating have large and balanced connection degrees ranging from 300 to 500 (they are represented by the several bars around the X-axis value of 6). They form the robust

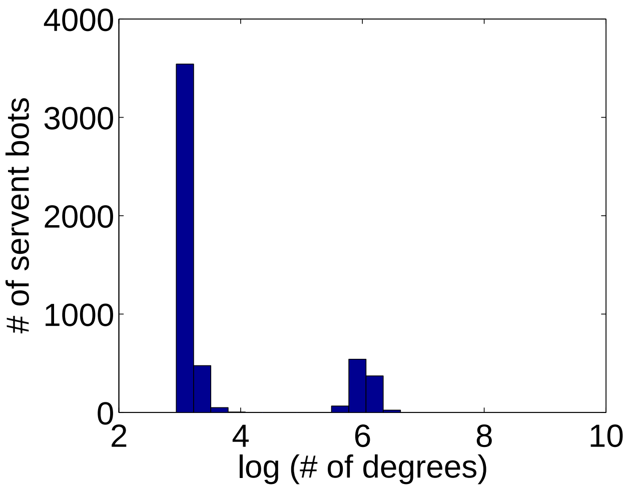


Fig. 4. Servent bot degree distribution (constructed via infection and peer-list updating)

backbone, connecting the hybrid P2P botnet tightly together. On the other hand, the remaining 4000 servent bots infected after the peer-list updating procedure have connection degrees only around 20 to 30.

6.3 Botnet command initiation

Comparing Fig. 2 with Fig. 1, we can see that the proposed P2P botnet does not show how its botmaster contacts the botnet to issue commands. Will the flexible service ports on servent bots prevent its botmaster from contacting them? We can answer this question now after introducing the botnet construction procedure.

As used in the simulation experiment, a botnet propagates based on a set of initial servent bots. The botmaster sets their service ports beforehand and thus knows their service ports. After the botnet is released, the botmaster could inject commands through these initial servent bots. After the botmaster issues a report command and gets the first report with the information of service ports of all current servent bots, the botmaster can inject commands through an arbitrarily chosen set of servent bots.

7 Botnet Robustness Study

Next, we study the robustness property of a constructed hybrid P2P botnet. Two factors affect the connectivity of a botnet: (1). Some bots are removed by defenders; and (2). Some bots are off-line (for example, due to the diurnal phenomenon [4]). These two factors, even though completely different, have the same impact on botnet connectivity when the botnet is used by its botmaster at a specific time. For this reason, we do not distinguish them in the following study.

7.1 Botnet robustness based on two metric functions

Since servent bots, especially the servent bots used in peer-list updating procedure, are the backbone connecting a botnet together, we study botnet connectivity when a certain fraction of peer-list updating servent bots are removed (that is to say, either removed by defenders or off-line).

We present two metric functions to measure robustness. Let $C(p)$ denote the *connected ratio* and $D(p)$ denote the *degree ratio* after removing top p fraction of mostly-connected bots among those peer-list updating servent bots—this is the most efficient and aggressive defense that could be done when defenders have the complete knowledge (topology, bot IP addresses ...) of the botnet. $C(p)$ and $D(p)$ are defined as:

$$C(p) = \frac{\text{\# of bots in the largest connected graph}}{\text{\# of remaining bots}} \qquad (3)$$

$$D(p) = \frac{\text{Average degree of the largest connected graph}}{\text{Average degree of the original botnet}} \qquad (4)$$

These two metric functions have clear physical meanings. The metric $C(p)$ shows how well a botnet survives a defense action by keeping the remaining members connected together. The metric $D(p)$ shows how densely the remaining botnet is connected together—it exhibits the ability of the remaining botnet to survive a further removal.

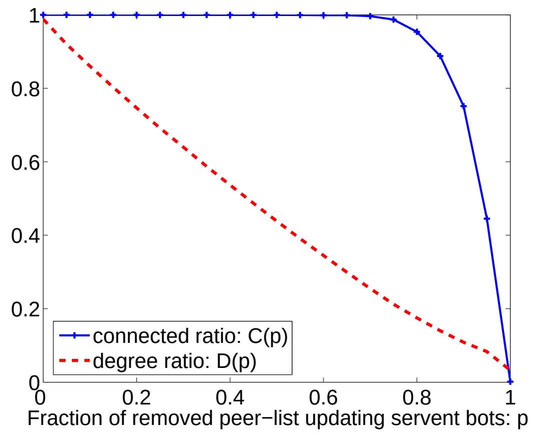


Fig. 5. Botnet robustness study

Fig. 5 shows the robustness of the proposed P2P botnet. The botnet is the one shown in Fig. 4 that has a vulnerable population of 500,000 and runs the peer-list updating procedure only once when 1,000 servent bots are infected. As shown in this figure, if all 1000 peer-list updating servent bots are removed, the botnet will be completely broken. This result shows the importance of the peer-list updating procedure. The botnet will largely stay connected ($C(p) > 95\%$) if less than 700 of those 1000 peer-list updating servent bots are removed, although it has a gradually decreasing connectivity as removal goes on (as exhibited by $D(p)$). This experiment shows the strong resistance of the proposed botnet against defense, even if defenders know the identities of all bots and the complete botnet topology.

7.2 Peer-list updating procedure

If a botmaster runs the peer-list updating procedure soon after releasing a botnet, she will shrink the time window for defenders to shut down the initial servent bots; however, the constructed botnet will rely upon fewer peer-list updating servent bots for its connectivity. Therefore, it is necessary to study how the number of peer-list updating servent bots affects the robustness of a botnet. Fig. 6 shows the simulation results in terms of $C(p)$ by varying the number of servent bots used in the peer-list updating procedure from 100 to 2000.

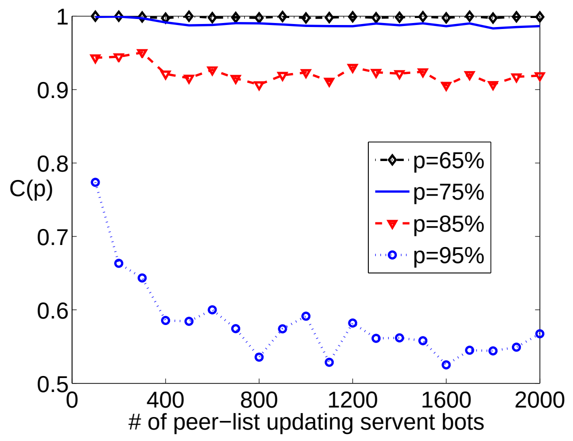


Fig. 6. Botnet robustness when the peer-list updating procedure runs once with different number of servent bots

This figure shows that as long as a small fraction of peer-list updating servant bots remain, botnet robustness does not change much when the number of peer-list updating servent bots varies. Of course, if the peer-list updating procedure contains fewer servent bots, it will likewise be easier for defenders to remove most of them, and hence, shut down the botnet.

7.3 Robustness mathematical analysis

We provide a simple analytical study of the botnet robustness. Assume that each peer list contains M servent bots. It is hard to provide a formula when removing the top p fraction of mostly-connected nodes. However, we could provide the formula of $C(p)$ when *randomly* removing p fraction of peer-list updating servent bots.

As we discussed before, the servent bots not used in peer-list updating procedure have very few extra links besides the M links given by their own peer lists. We simplify the analysis by assuming that each bot in the botnet connects only to peer-list updating servent bots. Then, when we consider removing a fraction of peer-list updating servent bots, more links will be removed compared to the original botnet network. Because of this bias, the analytical formula presented below slightly underestimates $C(p)$ in the case of random removal.

A bot is disconnected from the others when all M servent bots in its peer list have been removed. Because of the random removal, each peer-list updating servent bot has the equal probability p to be removed. Thus, the probability that a bot is disconnected is p^M. Therefore, any remaining bot has the same probability $1-p^M$ to stay connected, i.e., the mean value of $C(p)$ is (in case of random removal):

$$C(p) = 1 - p^M \tag{5}$$

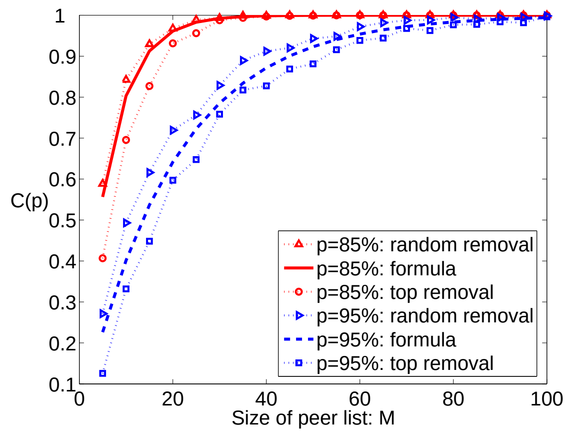


Fig. 7. Comparison of the analytical formula (5) and simulation results

Fig. 7 shows the analytical result from (5), comparing with the simulation result $C(p)$ of the random removal, and the simulation result $C(p)$ of the removal of top p fraction of mostly-connected peer-list updating servent bots. The analytical curve lies between those two simulated robustness metrics. It shows that the analytical formula indeed has a small underestimation bias compared with the random removal. Because removing top p fraction will remove more links from the botnet network than a random removal, the simulation results $C(p)$ from the top removal scenario are slightly lower than the derived results from (5). In summary, this figure shows that, even though the analytical formula (5) is not very accurate, it provides a good first-hand estimate of the robustness of a botnet.

This figure also shows that the proposed botnet does not need a large peer list to achieve a strong robustness.

For comparison, we can come up with a simple robustness model of today's C&C-based botnets. Suppose a C&C-based botnet has R C&C servers. When defenders have the complete knowledge of such a botnet, they will always remove these R bots first. Thus the botnet robustness metric $C(p)$ is:

$$C(p) = \begin{cases} 1, & < R \text{ bots are removed} \\ 0, & \geq R \text{ bots are removed} \end{cases} \tag{6}$$

the botnet will be shut down if all R C&C server bots are removed, which makes it much less robust than the proposed P2P botnet.

The robustness study presented here is a static study and analysis, considering the robustness of a botnet at any specific moment. In this case, we do not need to consider the botnet infection rate and its spreading speed. If we want to study the dynamics of a botnet when bots are removed gradually, or when bots are removed as the botnet spreads, we will need to consider these two important parameters.

8 Defense Against the Proposed Hybrid P2P Botnet

In this section, we discuss how defenders might defend against such an advanced botnet. In addition, we provide simulation studies and mathematical analysis of the performance of botnet monitoring.

8.1 Annihilation

First, the proposed hybrid P2P botnet relies on "servent bots" in constructing its communication network. If the botnet is unable to acquire a large number of servent bots, the botnet will be degraded to a traditional C&C botnet (the relationship of these two botnets is discussed in Section 3.3), which is much easier to shut down. For this reason, defenders should focus their defense effort on computers with static global IP addresses, preventing them from being compromised, or removing compromised ones quickly.

Second, as shown in Section 6, before a botmaster issues an update command for the first time, a botnet is in its most vulnerable state since it is mainly connected through the small set of initial servent bots. Therefore, defenders should develop quick detection and response systems, enabling them to quickly shut down the initial set of servent bots in a newly created botnet before its botmaster issues the first update command.

The third defense method relies on honeypot techniques. If a botnet cannot detect honeypots, defenders could try to *poison* its communication channel. Defenders let their infected honeypots join the botnet and claim to have static global IP addresses (these honeypots are configured to accept connections from other bots), they will be treated as servent bots. As a result, they will occupy many positions in peer lists of many bots, greatly decreasing the number of valid communication channels in the hybrid P2P botnet. In addition, defenders would know the detailed botnet communication structure and its members through those spying honeypots. With the detailed knowledge of the botnet, defenders could effectively shut it down by cutting off its remaining fragile communication channels.

Another controversial defense approach falls in the category of so-called "good worm" defense [29], [30], or the "cyber-immune system" [31]. Defenders program a "good-purpose" code to exploit the same vulnerability used in a botnet. The code will compromise vulnerable machines in the Internet and patch them. When a machine is already infected by a botnet, the good-purpose code obtains the bot's peer list, cleans the bot code, and then reversely compromises and cleans bots in the peer list. If a cleaned host is contacted by any other bots, the good-purpose code could fire back and clean those bots as well. However, this active defense is in fact another form of Internet attack; it would probably cause more harm than good. Thus it may not be a practical defense in the real world.

As discussed in Section 7, the strong robustness of the proposed botnet relies heavily on the peer-list updating procedure. Servent bots used in the peer-list updating procedure form the backbone of the communication network of a botnet. Therefore, the best strategy to disrupt the communication channel of a botnet, if the botnet cannot detect honeypots, is to poison the peer-list updating procedure with the following steps. First, once a honeypot is infected by a bot program, defenders quickly let the bot program infect many other honeypots (for example, by redirecting the bot's outgoing infection traffic to other honeypots). Then, when receiving a report command from the botmaster, all honeypot bots report as servent bots so that they will be used in the peer-list updating procedure. Defenders would achieve better poisoning defense if they have distributed honeypots and a large number of IP addresses.

When defenders conduct the above poisoning defense, a fraction of servent bots can be treated as being removed from the botnet. The botnet robustness studies presented in Section 7 show the effectiveness of such a defense.

8.2 Botnet monitoring based on honeypot techniques

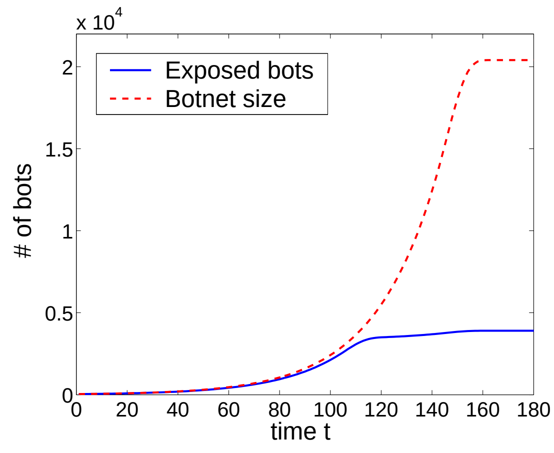
Honeypot is an effective way to trap and spy on malware and malicious activities. Because compromised machines in a botnet need to cooperate and work together, it is particular effective to use honeypot techniques in botnet spying [6], [32], if a botnet cannot detect and get rid off honeypot bots. The third annihilation method introduced above relies on honeypot techniques. In this section, we will introduce botnet monitoring and detection approaches based on honeypot techniques.

8.2.1 Botnet monitoring based on spying honeypots

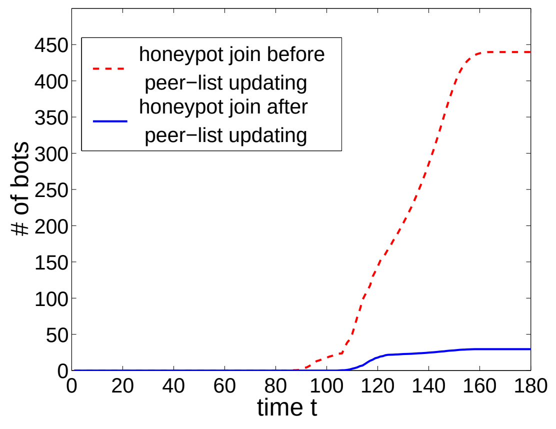
If a botnet cannot effectively detect honeypots, defenders could let their honeypots join botnets and monitor botnet activities. Based on honeypot bots, defenders may be able to obtain the plain text of commands issued by a botmaster. Once the meaning of the commands is understood, defenders are able to: (1). Quickly find the sensor machines used by a botmaster in report commands. If a sensor machine can be captured by defenders before the collected information on it is erased by its botmaster, they might be able to obtain detailed information of the entire botnet; (2). Know the target in an attack command so that they could implement corresponding countermeasures quickly right before (or as soon as) the actual attack begins.

Another honeypot-based monitoring opportunity happens during peer-list updating procedure. First, defenders could let their honeypot bots claim to be servent bots in peer-list updating. By doing this, these honeypots will be connected by many bots in the botnet; and hence, defenders are able to monitor a large fraction of the botnet. Second, during peer-list updating, each honeypot bot could get a fresh peer list, which means the number of bots revealed to each honeypot could be doubled.

A honeypot could be configured to route all its outgoing traffic to other honeypots; at the same time, the trapped malicious code still believes that it has contacted some real machines. This technique has been used before, such as in [33], [34]. Based on the similar technique, defenders could quickly build up a large number of spying honeypot bots by rerouting the infections sent out from already compromised honeypots to other vulnerable honeypots. In this way, defenders have many spying bots at hand, enabling them to monitor



(a). Botnet growth and a honeypot monitoring as one of the initial servent bot



(b). A honeypot monitoring by joining before/after peer-list updating procedure

Fig. 8. Botnet monitoring with one honeypot joining as a servent bot

the botnet effectively. Because all of these spying honeypots run the real bot code, a *remote code authentication*[3] cannot enable a botnet or its peer-list updating sensor to detect these spying honeypots.

Upon receiving a botnet report command, a honeypot could have its special program to send back a large amount of identities of fake bots. The information should not be sent from one single IP address. Instead, the honeypot should send out each fake bot's ID and host characteristics with different IP addresses. This approach falls in the category of "Sybil attack" [36]. It is another way to build up spying honeypot bots, to decrease the number of actual core servent bots or the botnet size (if the botmaster stops its botnet growth upon reaching a predefined size). However, this Sybil defense can be defeated if a botnet has a mechanism to conduct remote code authentication, such as using the Oblivious Hashing method [35].

From the above discussion, we can see that by using remote code authentication, a botnet could prevent a defense honeypot from sending information of a large amount of fake bots to its sensor, but it cannot prevent defenders from generating many spying honeypot bots by using real infected honeypots.

For the simulated botnet shown in Fig. 4, we conduct another set of simulations where we assume one of its servent bots is a defender's honeypot. We simulate three scenarios: the honeypot joining the botnet as one of initial servent bots; joining the botnet as a servent bot halfway before the peer-list updating procedure (when the botnet accumulates 500 servent bots—the peer-list updating procedure happens when the botnet accumulates 1000 servent bots); and joining the botnet right after the peer-list updating procedure. Fig. 8 shows the simulation results averaged over 100 simulation runs. The peer-list updating procedure happens around time t=110 (each bot is assumed to send out 358 scans per unit time to the entire IPv4 space, similar to what Code Red worm did [37]).

3. Remote code authentication is "the problem of verifying the identity of a remote program." [35]

If the honeypot is one of the initial servent bots, Fig. 8(a) shows that before the peer-list updating procedure, the honeypot could monitor most infected computers since it would appear in most bots' peer lists (as explained in Fig. 3). After the peer-list updating procedure, the honeypot could only observe a few more bots because it loses its critical role in the botnet connectivity. Fig. 8(b) shows that if the honeypot joins in the botnet halfway before the peer-list updating procedure, it knows on average around 450 bots in the botnet after the botnet propagation stops; while the honeypot could only know around 30 bots in the botnet if it joins after the peer-list updating procedure.

It could be very hard for a defender's honeypot to be one of initial servent bots, especially botmasters know the risk and select those initial servent bots very carefully. Therefore, we only consider the more realistic monitoring case where honeypots join as servent bots before the botmaster's peer-list updating procedure.

8.2.2 *Simulation and analysis of botnet monitoring by multiple honeypots*

It would be interesting to know how many honeypots defenders should set up in order to have an effective monitoring. To study this, we conduct another set of simulations by varying the number of honeypots joining a botnet before the peer-list updating procedure. Fig. 9 shows the number of exposed bots after the botnet stops growing as it reaches its desired size of 20,000 (the other simulation settings are the same as the experiment shown in Fig. 4). The simulation results are derived by averaging over 100 simulation runs.

We can actually derive an analytical model to estimate the mean value of exposed bots, denoted by $E[N_{exposed}]$, when there are n honeypots joining the botnet before the peer-list updating procedure. Suppose the peer list size is M, the final botnet has I number of bots, and the number of servent bots used in peer-list updating procedure is K. In our simulations, $M = 20, I = 20,000$ and $K = 1,000$.

Before the peer-list updating procedure, because few bots put any of those n honeypots in their peer lists (peer lists

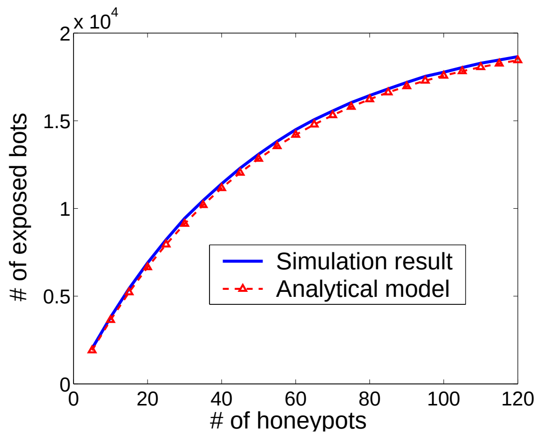


Fig. 9. Honeypot monitoring by joining as servent bots before the peer-list updating procedure

are dominated by initial servent bots), we ignore these small number of bots in our analysis, and hence, only consider how many bots have put at least one honeypot in their peer lists after the peer-list updating procedure.

For a specific bot, its peer list contains M servent bots. Since the vast majority of servent bots in peer lists belong to the group of K bots used in peer-list updating procedure, we can assume with little error that all servent bots in any peer list are picked from those K bots.

Thus the probability that the peer list in a specific bot contains none of those n honeypots is:

$$(1-\frac{n}{K})(1-\frac{n}{K-1})(1-\frac{n}{K-2})\cdots(1-\frac{n}{K-M}) \quad (7)$$

When $K \gg M$, which is the case in our simulations, (7) is approximately equal to:

$$(1-\frac{n}{K})^M \quad (8)$$

which is the probability that this bot will not be exposed to honeypots.

Since the botnet has overall I bots, the average number of exposed bots would be:

$$E[N_{exposed}] = I[1-(1-\frac{n}{K})^M] \quad (9)$$

Fig. 9 also shows the analytical result derived from the above model (9), which matches nicely with the simulation results. Because the analytical model ignores the few bots exposed before the peer-list updating procedure, the analytical estimates are slightly smaller than simulation results as exhibited in this figure.

8.2.3 *Simulation and analysis of botnet monitoring via darknet space*

The honeypot-based defense methods introduced above assume that compromised honeypots can join in a botnet and continue spying on the botnet activities. Sometimes this requirement may not be satisfied, e.g., when a botnet can quickly detect its honeypot members and remove them from its network [20]. For this reason, we introduce another honeypot-based monitoring technique, which only requires that the botnet could be fooled by a honeypot initially to pass its complete code, including its peer list, to the honeypot.

"Darknet space", or called "black hole", "network telescope", is a chunk of IP space that have no real computers. It is well known that darknet is effective in monitoring Internet malicious traffic [37], [38], [39]. By implementing "honeyd" [40], or an advanced honeypot-based darknet monitor (such as Internet Motion Sensor [41]), defenders may be able to trap a large number of botnet infection attempts. If the bot program cannot detect the darknet monitor and its honeypots initially, and passes its peer list in each infection attempt, defenders could get many copies of peer lists, obtaining the identities and important information (IP addresses, encryption key, service port) of many servent bots in a botnet.

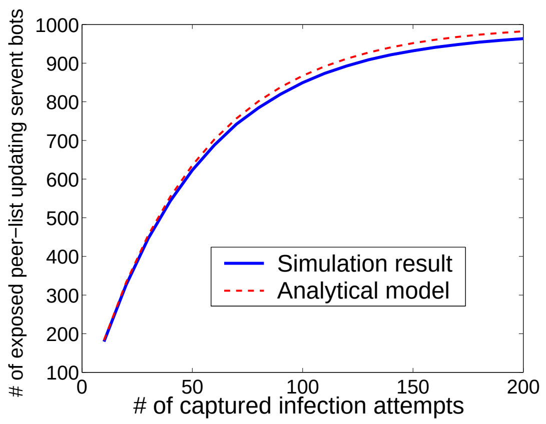


Fig. 10. Darknet monitoring of servent bots used in peer-list updating procedure

Because the servent bots used in peer-list updating procedure form the backbone of a botnet, we are interested in the fraction of these servent bots that are exposed via a darknet space monitoring. Here we present a simple analytical model for such a monitoring system.

Suppose the darknet begins capturing infection attempts after the peer-list updating procedure. Each captured infection attempt provides one peer list containing M servent bots, and there are K servent bots used in this peer-list updating procedure. For a specific servent bot used in peer-list updating procedure, it has probability p_1 to be exposed by one captured peer list where $p_1 < M/K$. Section 7 tells us that most servent bots in the peer list are the ones that have been used in peer list updating procedure. Thus we can estimate p_1 as:

$$p_1 \approx M/K \quad (10)$$

Therefore, if the darknet space captures x infection attempts, the probability that a specific servent bot used in peer-list updating procedure is exposed (through the captured peer lists) is equal to:

$$1-(1-\frac{M}{K})^x \quad (11)$$

Based on (10) and (11), we can derive the average number of servent bots used in peer-list updating procedure, denoted by $E[N'_{exposed}]$, that will be exposed by x captured infection attempts:

$$E[N'_{exposed}] = K[1 - (1 - \frac{M}{K})^x] \quad (12)$$

Fig. 10 shows the simulation results (averaged over 100 simulation runs) and the analytical model results. The analytical results match well with the simulation results but are a bit higher, which is due to the fact that (10) has slightly overestimated the value of p_1.

Fig. 10 shows that if the darknet can capture 200 copies of peer lists, defenders would be able to know more than 95% of servent bots used in peer-list updating procedure. Thus this darknet based monitoring is an effective way to track the proposed botnet. However, this approach still relies on honeypot techniques. If a bot infection is composed by several sequential components (which is the case for most current botnets [42]), and a bot passes its peer list to a newly infected host only after the remote host is verified not being a honeypot, the darknet-based monitoring approach would become invalid.

8.3 Botnet detection and monitoring without honeypots

The previous subsection shows that we can propose many effective botnet monitoring approaches based on honeypot techniques. However, as honeypot-based defense systems gradually become popular and widely deployed, botmasters will inevitably develop their botnets to detect honeypots. For this reason, we propose botnet detection and monitoring approaches that do not rely on honeypots.

8.3.1 *Monitoring traffic to botnet sensor*

A possible weakness point of the proposed botnet is its centralized monitoring sensor. If defenders have set up a good traffic logging system, it is possible that they could capture the traffic to a botnet sensor. We call such a monitoring system as a *botnet sensor monitor*. Even though defenders may not be able to capture a botnet sensor before its botmaster destroys the sensor (after completing botmaster's monitoring task), they still could use the captured traffic log to figure out the IP addresses of bots who contacted the sensor in the past. In this way, defenders could get a relatively complete picture of a botnet.

8.3.2 *Detecting and monitoring servent bots*

In the proposed hybrid P2P botnet, servent bots, especially those used in the peer-list updating procedure, are the backbone of a botnet. Fig. 4 shows that each servent bot used in the peer-list updating will serve 300 to 500 bots. If a non-server host is infected and serves as one of these servent bots, the host is relatively easy to be spotted by defenders due to the huge increase of traffic in and out of this host. When the number of servent bots compared to the total botnet population decreases, each of these servent bots must serve a larger number of bots, and hence, is easier to be detected by defenders.

A simple statistical analysis can show this relationship. Denote the number of client bots served by a servent bot as D. For the proposed botnet, D is a random variable. Suppose the peer-list updating procedure is conducted after a botnet finishes its propagation, then all servent bots are used in the updating procedure; and hence, they have evenly distributed connection degrees. Following the same notations as in previous analysis, K is the number of servent bots in a botnet, I is the botnet size, and M is the peer list size. There are $I - K$ client bots in the botnet, each of which connects to M servent bots. Thus there are $(I - K) \cdot M$ connection links initiated from client bots to servent bots. We can derive the mean value of the number of client bots served by each servent bot as:

$$E[D] = \frac{(I - K) \cdot M}{K} = \frac{I \cdot M}{K} - M \quad (13)$$

It is not hard to derive the distribution of D. In the peer-list updating procedure, each client bot is given a randomly chosen peer list by the updating sensor. For any specific servent bot, each client bot has an equal and small probability M/K to connect to the servent bot. Therefore, the random variable D follows "Binormial distribution" [43] with parameters $(I-K)$ and M/K, i.e.,

$$D \sim B(I - K, M/K) \quad (14)$$

9 Discussions

From the defense discussion in previous section, we see that honeypot plays a critical role in most defense methods against the proposed hybrid P2P botnet. Botmasters might design countermeasures against honeypot defense systems. Such countermeasures might include detecting honeypots based on software or hardware fingerprinting [40], [44], [45], or exploiting the legal and ethical constraints held by honeypot owners [20]. Most of current botnets do not attempt to avoid honeypots—perhaps it is simply because attackers have not felt the threat from honeypot defense yet. As honeypot-based defense becomes popular and being widely deployed, we believe botmasters will eventually add honeypot detection mechanisms in their botnets. The war between honeypot-based defense and honeypot-aware botnet attack will come soon and intensify in the near future.

For botnet defense, current research shows that it is not very hard to monitor Internet botnets [4], [15], [32]. The hard problem is: how to defend against attacks sent from botnets, since it is normally very hard to shut down a botnet's control? Because of legal and ethical reason, we as security defenders cannot actively attack or compromise a remote bot machine or a botnet C&C server, even if we are sure a remote machine is installed with a bot program. For example, the well-known "good worm" approach is not practical in the real world. The current practice of collaborating with the ISPs containing bot-infected machines is slow and resource-consuming. There are still significant challenges in botnet defense research in this aspect.

From the robustness study in Section 7 and the defense study in Section 8, we can see that the proposed hybrid P2P botnet makes a future botnet harder to be monitored, but most importantly, makes a botnet MUCH harder to shut down. By replacing a few isolated C&C servers with a significantly larger amount of interleaved servent bots, the proposed botnet greatly increases its survivability.

The proposed hybrid P2P botnet utilizes centralized sensor hosts. This does not make it as weak as a centralized version of botnets. First, sensor hosts are not responsible for botnet command and control communication—their roles are data collection and peer list distribution. If a sensor host is detected and monitored, the botnet could possibly be fully exposed to defenders. However, the botnet will still have its strong survivability as discussed in Section 7. In other words, the command and control channel of the proposed botnet is mostly peer-to-peer structured and not affected by sensor hosts. Second, sensor hosts are disposable. When a botmaster suspects that her senor host is being monitored, she can simply discard it and pick another compromised machine as the sensor host.

The proposed hybrid P2P botnet represents only a specific P2P botnet design. In reality, botmasters may come up with some other types of P2P botnet designs. However, we believe this research is still meaningful to security community. The proposed design is practical and can be implemented by botmasters with little engineering complexities. Botmasters will come with a similar design sooner or later, and we must be well prepared for such an attack, or a similar attack, before it happens.

10 Conclusion

To be well prepared for future botnet attacks, we should study advanced botnet attack techniques that could be developed by botmasters in the near future. In this paper, we present the design of an advanced hybrid peer-to-peer botnet. Compared with current botnets, the proposed one is harder to be monitored, and much harder to be shut down. It provides robust network connectivity, individualized encryption and control traffic dispersion, limited botnet exposure by each captured bot, and easy monitoring and recovery by its botmaster. To defend against such an advanced botnet, we point out that honeypots may play an important role. We should, therefore, invest more research into determining how to deploy honeypots efficiently and avoid their exposure to botnets and botmasters.

Acknowledgement

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

References

[1] S. Kandula, D. Katabi, M. Jacob, and A. Berger, "Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds," in *2nd Symposium on Networked Systems Design and Implementation (NSDI)*, May 2005.

[2] C. T. News, "Expert: Botnets No. 1 emerging Internet threat," 2006, http://www.cnn.com/2006/TECH/internet/01/31/furst/.

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

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

[5] A. Ramachandran, N. Feamster, and D. Dagon, "Revealing botnet membership using dnsbl counter-intelligence," in *USENIX 2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI 06)*, June 2006.

[6] 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.

[7] J. R. Binkley and S. Singh, "An algorithm for anomaly-based botnet detection," in *USENIX 2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI 06)*, June 2006.

[8] I. Arce and E. Levy, "An analysis of the slapper worm," *IEEE Security & Privacy Magazine*, Jan.-Feb. 2003.

[9] Sinit P2P trojan analysis. Http://www.lurhq.com/sinit.html.

[10] Phatbot trojan analysis. Http://www.lurhq.com/phatbot.html.

[11] R. Lemos. (2006, May) Bot software looks to improve peerage. Http://www.securityfocus.com/news/11390.

[12] "Servent," http://en.wikipedia.org/wiki/Servent.

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

[14] B. McCarty, "Botnets: Big and bigger," *IEEE Security & Privacy Magazine*, vol. 1, no. 4, July 2003.

[15] P. Barford and V. Yegneswaran, *An Inside Look at Botnets, To appear in Series: Advances in Information Security*. Springer, 2006.

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

[17] F. Monrose. (2006) Longitudinal analysis of botnet dynamics. ARO/DARPA/DHS Special Workshop on Botnet.

[18] T. Strayer. (2006) Detecting botnets with tight command and control. ARO/DARPA/DHS Special Workshop on Botnet.

[19] Y. Chen. (2006) IRC-based botnet detection on high-speed routers. ARO/DARPA/DHS Special Workshop on Botnet.

[20] C. Zou and R. Cunningham, "Honeypot-aware advanced botnet construction and maintenance," in *Proceedings of International Conference on Dependable Systems and Networks (DSN)*, June 2006.

[21] R. Vogt, J. Aycock, and M. Jacobson, "Army of botnets," in *Proceedings of 14th Annual Network and Distributed System Security Symposium (NDSS)*, Feburary 2007.

[22] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys and Tutorials*, vol. 7, no. 2, 2005.

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

[24] R. Bhagwan, S. Savage, and G. M. Voelker, "Understanding availability," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, Feburary 2003.

[25] A. Ramachandran and N. Feamster, "Understanding the network-level behavior of spammers," in *Proceedings of ACM SIGCOMM*, September 2006.

[26] C. Gkantsidis, T. Karagiannis, P. Rodriguez, and M. Vojnovic, "Planet scale software updates," in *Proceedings of ACM SIGCOMM*, September 2006.

[27] (2005, November) CNET news: Bots slim down to get tough. Http://news.com.com/2104-7355_3-5956143.html.

[28] (2006, February) Washington Post: The botnet trackers. http://www.washingtonpost.com/wp-dyn/content/article/2006/02/16/AR2006021601388.html.

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

[30] M. Liljenstam and D. Nicol, "Comparing passive and active worm defenses," in *Proceedings of the 1st International Conference on Quantitative Evaluation of Systems (QEST)*, September 2004.

[31] V. A. Skormin, J. G. Delgado-Frias, D. L. McGee, J. Giordano, L. J. Popyack, V. I. Gorodetski, and A. O. Tarakanov, "BASIS: A biological approach to system information security," in *MMM-ACNS '01: Proceedings of the International Workshop on Information Assurance in Computer Networks*. London, UK: Springer-Verlag, 2001, pp. 127–142.

[32] M. Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in *Internet Measurement Conference*, October 2006.

[33] 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.

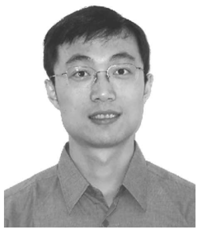
[34] Y. Tang and S. Chen, "Defending against internet worms: A signature-based approach," in *Proceedings of the IEEE INFOCOM*, May 2005.

[35] Y. Chen, R. Venkatesan, M. Cary, R. Pang, S. Sinha, and M. H. Jakubowski, "Oblivious hashing: A stealthy software integrity verification primitive," in *IH '02: Revised Papers from the 5th International Workshop on Information Hiding*. London, UK: Springer-Verlag, 2003, pp. 400–414.

[36] J. R. Douceur, “The sybil attack,” in *IPTPS ’01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. London, UK: Springer-Verlag, 2002, pp. 251–260.
[37] 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)*, October 2003, pp. 190–199.
[38] S. Staniford, V. Paxson, and N.Weaver, “How to own the Internet in your spare time,” in *Proceedings of USENIX Security Symposium*, August 2002, pp. 149–167.
[39] D. Moore, C. Shannon, G. Voelker, and S. Savage, “Network telescopes: Technical report,” CAIDA, Tech. Rep. TR-2004-04, 2004.
[40] “Honeyd security advisory 2004-001: Remote detection via simple probe packet,” 2004, http://www.honeyd.org/adv.2004-01.asc.
[41] “University of Michigan Internet Motion Sensor,” http://ims.eecs.umich.edu/.
[42] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, “BotHunter: Detecting malware infection through ids-driven dialog correlation,” in *Proceedings of the 16th USENIX Security Symposium (Security’07)*, August 2007.
[43] Wikipedia: Binormial distribution. http://en.wikipedia.org/wiki/Binomial_distribution.
[44] K. Seifried, “Honeypotting with VMware basics,” 2002, http://www.seifried.org/security/index.php/ Honeypotting_With_VMWare_Basics.
[45] J. Corey, “Advanced honey pot identification and exploitation,” 2004, http://www.phrack.org/fakes/p63/p63-0x09.txt.

Ping Wang received the BS and MS degrees in computer science from Beijing University of Aeronautics and Astronauts, China, in 2001 and 2004, respectively. Currently she is working toward the PhD degree in School of Electrical Engineering and Computer Science from University of Central Florida. Her research interests include computer and network security.

Sherri Sparks is a PhD student at the University of Central Florida. Currently, her research interests include offensive/defensive stealth code technologies and digital forensics. She has been involved in both academia and industry for the past several years with articles and papers published in Usenix Login, ACSAC, SecureComm, Security Focus, and Phrack magazine.



Cliff C. Zou (M’05) received the Ph.D degree in Department of Electrical and Computer Engineering from University of Massachusetts, Amherst, MA, in 2005.

He is an Assistant Professor in School of Electrical Engineering and Computer Science, University of Central Florida. His research interests include computer and network security, network modeling and performance evaluation.