An Advanced Hybrid Peer-to-Peer Botnet

Ping Wang Sherri Sparks Cliff C. Zou
School of Electrical Engineering and Computer Science
University of Central Florida, Orlando, FL
{pwang, ssparks, czou}@cs.ucf.edu

***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. Possible defenses against this advanced botnet are suggested.**

I. 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") [6], [5]. 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 [3], [6], [1], [4], [7], [5]. 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 his or 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.

A. 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, in order to remove the bootstrap process which is easily exploited by defenders to shut down a botnet, the Slapper worm builds a list of known bots for each infected computer during propagation [8]. Sinit likewise lacks a bootstrap process and 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 in those botnets appeared before are not mature and have many weaknesses. A Sinit bot uses random probing to find other Sinit bots to communicate with. This results in poor connectivity for the constructed botnet and easy detection due to the extensive probing traffic [9]. Phatbot utilizes Gnutella cache servers for its bootstrap process. This also makes the botnet easy to shut down. 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

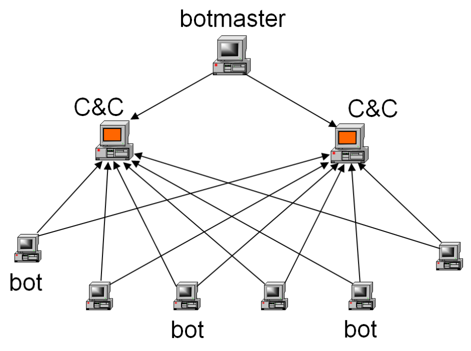

Fig. 1. Command and control architecture of a C&C botnet

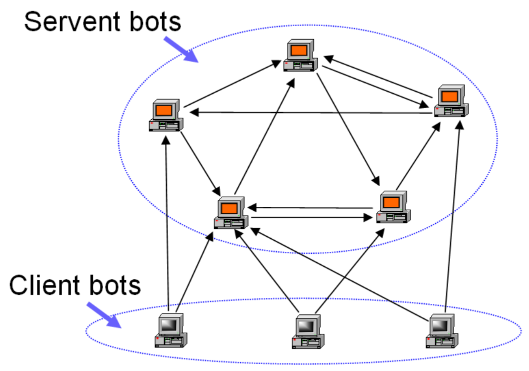

Fig. 2. Command and control architecture of the proposed hybrid P2P botnet

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 lot traffic, rendering it susceptible to monitoring via network flow analysis.

Some other available robust distributed systems include "censorship-resistant" system and "anonymous" P2P system. However, their design goal of robustness is different from a botnet. For example, these robust 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 it best to hide IP addresses of all bots in it.

B. 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) 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 specific 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 reorganizes the botnet 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.

C. Paper Organization

The rest of the paper is organized as follows. Section II introduces related studies. Section III introduces the control communication architecture of the proposed botnet. Section IV discusses the designs to ensure the authentication and security of command communication. In Section V, we present how a botmaster is able to monitor his or her botnet easily. We present how to construct the proposed botnet in Section VI and study its robustness against defense in Section VII. We present possible defenses against the botnet in Section VIII.

We give a few discussions in Section IX and finally conclude the paper in Section X.

II. 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. [6], [3], [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 that 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 or have dynamic IP addresses; (2). The robust botnet control topology cannot be set up through reinfection mechanism, if a botnet does not have substantive reinfections during its built-up, which is the case for most botnets in reality.

III. Proposed Hybrid P2P Botnet Architecture

A. 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[1]. 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.

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

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.

B. 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.

A botmaster injects his or her commands through any bot(s) in the botnet. Both client and servent bots actively and 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.

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.

C. 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 VI and Section VII.

IV. 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.

A. 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.

B. Individualized Encryption Key

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.

C. 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 randomly picked by the bot. Considering this, a peer list needs to contain the service port information as well. For example, the peer list on bot A is:

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

The 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 [19], [18].
- **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.

A randomly-generated service port may not always be good for botnets since network traffic going to a rarely used port is abnormal. To overcome this, a botmaster can specify a set of service ports for each bot to choose, preferably choosing from those standard encrypted ports such as port 22 (SSH), port 443 (HTTPS), or port 993 (IMAPS). Furthermore, a sophisticated botmaster could even program bot code to mimic the protocol format of the service port as what "honeyd" [23] does.

V. Botnet Monitoring by its Botmaster

Another major challenge in botnet design is making sure that a botnet is difficult to be monitored 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, bandwidth, 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.

A. 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 IX).
- 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 its hard-coded public key K^+ to ensure the confidentiality of its report data. In addition, a botmaster could use several compromised machines as stepping stones in retrieving the data on sensors. These are standard practices so we will not explain more.

B. *Additional Monitoring Information*

From our understanding, there are three additional measurements directly affecting the efficiency of botnet attacks: attack bandwidth, IP address type, and diurnal dynamics.

First, to conduct an effective denial-of-service attack, a botmaster may want to measure the actual bandwidth from each bot to a target machine. It could be done by letting each bot to have a couple of normal connections with the target machine, based on any available bandwidth measurement techniques.

Second, each bot could have its randomly-generated unique ID[2] and report its ID with other information to its botmaster's sensor. In this way, a botmaster could obtain an accurate report, and also know the properties of bots with DHCP or NAT addresses. With this information, a botmaster could conduct fine-tuned attacks, e.g., only letting bots that frequently change their IP addresses to send out email spam in order to avoid being blocked by DNSBL-based spam filter [5].

Third, as pointed out by [4], the online population of a botnet exhibits a clear "diurnal" dynamics due to many users shutting down their computers at night. In a time zone, the peak online population of a botnet could be as much as four times of the bottom level online population. To maximize botnet attack strength, a botmaster may 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 an optimal release time to maximize its propagation speed [4]. Since bots could function as spyware, it's not hard for a bot to obtain its host machine's diurnal dynamics.

[2]To make sure each bot has a unique ID, a simple way is to let every bot randomly generate a very large number for its ID—one or two collisions do not matter much.

VI. Botnet Construction

A. *Basic construction procedures*

Botnet connectivity is solely determined by the peer list in each bot. A natural way to build peer lists is to construct them during propagation. 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 on 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 procedures:

- **New infection:** Bot A passes its peer list to a vulnerable host B when compromising it. If B is a servent bot, A adds B into its peer list (by randomly replacing one bot if its peer list is full). Similarly, if A is a servent bot, B adds A 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.

The reinfection procedure makes it harder for defenders to infer the infection time order ("traceback") among bots based on captured peer lists. In this process, 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 changes 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 [25], [26], 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.

From our simulation experiments, we find that a botnet constructed only with the above two procedures is not robust enough. Because a botnet stops growing after reaching the size of 20,000, the reinfection events rarely happen (around 600). Due to this phenomenon, 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. 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.

Vogt et al. [21] constructed a super-botnet only with the algorithms that are similar to the "new infection" and "reinfection" procedures 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. In this simulation, 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.

B. 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 V, a botmaster could monitor his botnet easily whenever he 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. Entries in the updated peer list in each bot are randomly chosen from those peer-list updating servent bots.

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.

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. 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 reconnect a broken botnet.

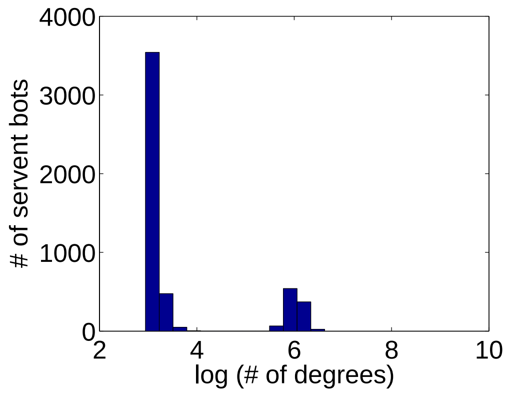

Fig. 3. Servent bot degree distribution

Fig. 3 shows the degree distribution for servent bots (client bots always have a degree of M) when a botnet uses all three construction procedures. 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 although those 4000 servent bots infected after the peer-list updating procedure still have small degrees, the first 1000 servent bots used in peer-list updating have large and balanced connection degrees, ranging from 300 to 500. They form the robust backbone, connecting the hybrid P2P botnet tightly together.

VII. 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.

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).

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:

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

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

The metric $C(p)$ shows how well a botnet survives a defense action; the metric $D(p)$ exhibits how densely the remaining botnet is connected together.

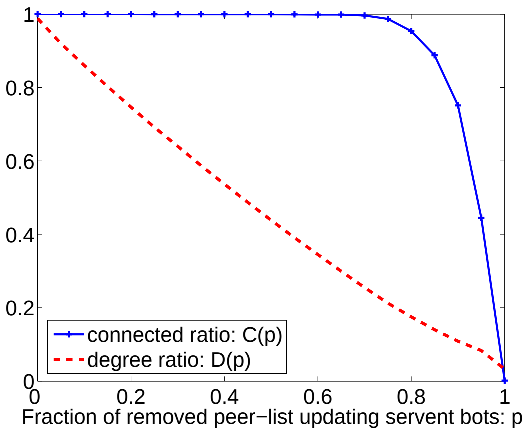

Fig. 4. Botnet robustness study

Fig. 4 shows the robustness study. The botnet is the one shown in Fig. 3 that has a vulnerable population of 500,000 and runs the peer-list updating procedure 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 with further removal (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.

A. 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):

$$C(p) = 1 - p^M \quad (5)$$

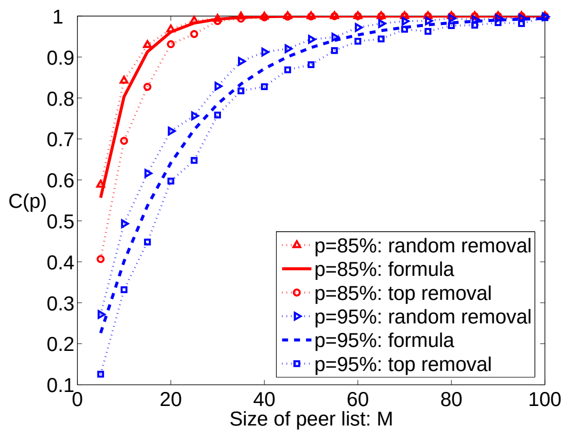

Fig. 5. Comparison of the analytical formula (5) and simulation results

Fig. 5 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.

The robustness study presented here is a static study and analysis. We have not considered how a botnet behave if bots are removed by defenders gradually, or when bots are removed as the botnet spreads. For this reason, the botnet infection rate and spreading speed does not matter to our robustness study, and we will study these issues in the future.

VIII. Defense Against the Proposed Hybrid P2P Botnet

A. Annihilating

We introduce possible annihilating defense in three ways. 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 III-C), 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 VI, 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.

Third, defenders could try to *poison* the communication channel of a P2P botnet based on honeypot techniques. If they 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 on their claimed global IP addresses), they will be treated as servent bots. As a result, they will occupy positions in peer lists in many bots, decreasing the number of valid communication channels in the hybrid P2P botnet.

As discussed in Section VII, 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, perhaps, 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 to 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.

B. Monitoring

In this area, defenders hold a better position with the help from honeypots. If they utilize a honeypot on a large IP space, they may be able to trap a large number of botnet infection attempts. If the bot program cannot detect the honeypot and passes its peer list in each infection attempt, the defenders could get many copies of peer lists, obtaining the important information (IP addresses, encryption key, service port) of many servent bots in a botnet.

Second, 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. 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.

For the simulated botnet shown in Fig. 3, we conduct another set of simulations where one of its servent bot is a honeypot. If the honeypot is one of the initial servent bot, the honeypot knows the identity of on average 20% of bots in the botnet after the botnet propagation stops (the botnet has 20,000 bots). If the honeypot joins in the botnet before the peer-list updating procedure (when the botnet infects 500 servent bots), it knows on average 2.3% of bots in the botnet after the botnet propagation stops. If the honeypot joins in the botnet right after the one and only one peer-list updating procedure, the honeypot could only know around 30 bots in the botnet.

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 destroying the sensor (after completing botmaster's monitoring task), they still could use the captured traffic log to figure out the IP addresses of potential bots who contacted the sensor in the past. In this way, defenders could get a relatively complete picture of a botnet.

Not like the other monitoring methods, the above traffic logging and analysis approach does not rely on honeypot systems. This makes it important to conduct further research on this approach since we must be prepared in case a future smart botnet can detect and disable honeypot.

IX. 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 [27], [28], [29], or exploiting the legal and ethical constraints held by honeypot owners [20]. Most of current botnets do not attempt to avoid honeypots—it is simply because attackers have not feel 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], [30]. 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.

X. 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 much harder to be shut down or monitored. 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 honeypot 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. The authors would like to thank the anonymous reviewers for their helpful comments for improving this paper.

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)", month = "Feburary", year="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] C. News. (2005, November) Bots slim down to get tough. Http://news.com.com/2104-7355_3-5956143.html.

[26] (2006, February) Washington post: The botnet trackers. http://www.washingtonpost.com/wp-dyn/content/article/2006/02/16/AR2006021601388.html.

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

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

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

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