A Systematic Study on Peer-to-Peer Botnets

Ping Wang, Lei Wu, Baber Aslam and Cliff C. Zou
School of Electrical Engineering & Computer Science
University of Central Florida
Orlando, Florida 32816, USA
Email: {pwang, lwu, ababer, czou}@eecs.ucf.edu

***Abstract*—"Botnet" is a network of computers that are compromised and controlled by an attacker. Botnets are one of the most serious threats to today's Internet. Most current botnets have centralized command and control (C&C) architecture. However, peer-to-peer (P2P) structured botnets have gradually emerged as a new advanced form of botnets. Without central C&C servers, P2P botnets are more resilient to defenses and countermeasures than traditional centralized botnets. In this paper, we systematically study P2P botnets along multiple dimensions: bot candidate selection, network construction, C&C mechanisms and communication protocols, and mitigation approaches. We carefully study two defense approaches: index poisoning and sybil attack. According to the common idea shared by them, we are able to give analytical results to evaluate their performance. We also propose possible counter techniques which might be developed by attackers against index poisoning and sybil attack defenses. In addition, we obtain one interesting finding: compared to traditional centralized botnets, by using index poisoning technique, it is easier to shut down or at least effectively mitigate P2P botnets that adopt existing P2P protocols and rely on file index to disseminate commands.**

I. Introduction

"Botnet" is a network of compromised computers (bots) running malicious software, usually installed via all kinds of attacking techniques such as trojan horses, worms and viruses. These zombie computers are remotely controlled by an attacker (botmaster). Botnets with a large number of computers have enormous cumulative bandwidth and computing capability. They are exploited by botmasters for initiating various malicious activities, such as email spam, distributed denial-of-service attacks, password cracking and key logging. Botnets have become one of the most significant threats to the Internet.

Today, centralized botnets are still widely used. In a centralized botnet, bots are connected to several servers (called C&C servers) to obtain commands. This architecture is easy to construct and efficient in distributing botmaster's commands; however, it has a weak link - the C&C servers. Shutting down those servers would cause all the bots lose contact with their botmaster. In addition, defenders can easily monitor the botnet by creating a decoy to join a specified C&C channel.

Recently, peer-to-peer (P2P) botnets, such as Trojan.Peacomm botnet [1], Storm botnet [2] and its newly improved version Waledac botnet [3], have emerged, as attackers gradually realize the limitation of traditional centralized botnets. Just like P2P networks, which are resilient to dynamic churn (i.e., peers join and leave the system at high rates [4]), P2P botnet communication won't be disrupted when losing a number of bots. In a P2P botnet, there is no central server, and bots are connected to each other and act as both C&C server and client. P2P botnets have shown advantages over traditional centralized botnets. As the next generation of botnets, they are more robust and difficult for security community to defend.

Researchers have started to pay attention to P2P botnets. Grizzard et al. and Holz et al. dissected Trojan.Peacomm botnet [1] and Stormnet [2] in detail respectively. However, in order to effectively fight against this new form of botnets, enumerating every individual P2P botnet we have seen in the wild is not enough. Instead, we need to study P2P botnets in a systematic way. Therefore in this paper we try to explore the nature of various kinds of P2P botnets, analyzing their similarities and differences, and discussing their weaknesses and possible defenses. We hope to shed light on P2P botnets, and help researchers and security professionals be well prepared and develop effective defenses against them.

Our contributions are the following:

- We systematically study P2P botnets along multiple dimensions: infection vectors, bot candidate selection, bootstrap procedure, network structure, and C&C mechanisms and communication protocols.
- We present a number of countermeasures to defend against P2P botnets. Our study on index poisoning and sybil attack shows that both of these defense approaches share a common scheme, which leads us to analytical work on the evaluation of their performance.
- We obtain one counter-intuitive finding: if the index poisoning defense is valid (when the botnet adopts existing P2P protocols and relies on file index to disseminate commands), P2P botnets are equally easy (or hard) to defend compared to traditional centralized botnets.
- From attackers' perspective, we propose a novel and realistic technique that might be deployed by them to counter the index poisoning defense. This method guarantees that command related indices can be generated by and only by botmasters, not by ordinary bots.

The remainder of the paper is organized as follows. In Section II, we study the lifetime of P2P botnets, which is composed of three stages. Upon our understanding of P2P botnets, a number of countermeasures are presented in Section III, and special attentions are given to two defense techniques: index poison (Section III-A) and sybil attack (Section III-B). Finally we conclude this paper in Section IV.

978-1-4244-4581-3/09/$25.00 ©2009 IEEE

II. P2P BOTNETS

The lifetime of botnets is composed of three stages. Stage one - recruiting members, a botmaster needs to compromise many computers in the Internet, so that he/she can control them remotely. Stage two - forming the botnet, bots need to find a way to connect to each other and form a botnet. Stage three - standing by for instructions, after the botnet is built up, all bots are ready to communicate with their botmaster for further instructions, such as launching an attack or performing an update. In this section, we will discuss each stage in detail.

A. Stage one: recruiting bot members

P2P networks are gaining popularity of distributed applications, such as file-sharing, web caching, network storage [5]. In these content trading P2P networks, without a centralized authority it is not easy to guarantee that the file exchanged is not malicious. For this reason, these networks become the ideal venue for malicious software to spread. It is straightforward for attackers to target vulnerable hosts in existing P2P networks as bot candidates and build their zombie army. Many P2P malware have been reported, such as Gnuman [6], VBS.Gnutella [6] and SdDrop [7]. They can be used to compromise a host and make it become a bot.

However, in this way, the scale of a botnet will be limited by the size of the existing P2P network, and the network will be the only propagation media. On the contrary, P2P botnets we have witnessed recently [1], [2], [8] do not confine themselves to existing P2P networks. They have shown that it is more flexible and practical if bot members are recruited from the entire Internet through all possible spread mediums like emails, instant messages and file exchanging.

B. Stage two: forming the botnet

Upon infection, the next important thing is to let newly compromised computers join the botnet network and connect to other bots. Otherwise, they are just isolated individual computers without much use for botmasters.

Now for the convenience of further discussion, we first introduce three terms: "*parasite*", "*leeching*" and "*bot-only*" P2P botnets. Each of them represents a class of P2P botnets. In a parasite P2P botnet, all the bots are selected from vulnerable hosts within an existing P2P network, and it uses this available P2P network for command and control. A leeching P2P botnet refers to one whose members join an existing P2P network and depend on this P2P network for C&C communication, but the bots could be vulnerable hosts that were either inside or outside of the existing P2P network. For example, the early version of Storm botnet [2] belongs to this class of botnet. A bot-only P2P botnet builds its own network, in which all the members are bots, such as Stormnet [2] and Nugache [8].

If all the bots are selected from an existing P2P network, it is not necessary to perform any further action to form the botnet, because bots can find and communicate with each other by simply using current P2P protocol. In other words, for a parasite P2P botnet, up to this point, the botnet construction is done and the botnet is ready to be operated by its botmaster.

However, if a random host on the Internet is compromised, it has to know how to find and join the botnet. As we know current P2P file-sharing networks provide the following two general ways for new peers to join a network:

1) An initial peer list is hard-coded in each P2P client. When a new peer is up, it will try to contact peers in that initial list to update its neighboring peer information.
2) There is a shared web cache, such as Gnutella web cache [9], stored at some place on the Internet, and the location of the cache is put in the client code. Thus new peers can refresh its neighboring peer list by going to the web cache and fetching the latest updates.

This initial procedure of finding and joining a P2P network is usually called "bootstrap" procedure. It can be directly adopted for P2P botnet construction. Either a predetermined list of peers or the locations of predetermined web caches need to be hard-coded in the bot code. Then a newly infected host knows which peer to contact or at least where to find candidates of neighboring peers it will contact later.

For instance, Trojan.Peacomm [1] is a piece of malware to create a P2P botnet which uses the P2P protocol implemented on Overnet for C&C communication. A list of Overnet nodes that are likely to be online is hard-coded into the bot's installation binary. When a victim is compromised and runs a Trojan.Peacomm, it will try to contact peers in this list to bootstrap onto the Overnet network. Another P2P botnet, Stormnet [2], uses a similar bootstrap mechanism: the information about other peers with which a new bot member communicates after the installation phase, is encoded in a configuration file that is also stored on the victim machine by Storm worm binary.

C. Stage three: standing by for instructions

Once a botnet is built up, all the bots are standing by for instructions from their botmaster to perform illicit activities or updates. Therefore C&C mechanism is very important and is the major part of a botnet design. It directly determines the communication topology of a botnet, and hence affects the robustness of a botnet against network/computer failures, or security monitoring and defenses.

The C&C mechanisms can be categorized as either *pull* or *push* mechanism. Pull mechanism, i.e., "command publishing/subscribing", refers to the manner that bots retrieve commands actively from a place where botmasters publish commands. On the contrary, push mechanism, i.e., "command forwarding", means bots passively wait for commands to come and will forward received commands to others.

For centralized botnets, pull mechanism is commonly used. Take HTTP-based botnets as an example, a botmaster publishes commands on a web page, and bots periodically visit this web page via HTTP to check for any command updates. In the following, we will discuss how pull and push C&C mechanisms can be applied in P2P botnets.

1) Leveraging Existing P2P Protocols: As we discussed above, both parasite and leeching P2P botnets depend on existing P2P networks. Thus it is natural to leverage the

TABLE I: COMPARISON AMONG THREE TYPES OF P2P BOTNETS

Features	Parasite	Leeching	Bot-only
Infection vectors	P2P malware	Any kind of malware	Any kind of malware
Bot candidates	Vulnerable hosts in P2P networks	Vulnerable hosts in the Internet	Vulnerable hosts in the Internet
Bootstrap procedure	None	Required	Optional
Members in the network	legitimate peers & bots	legitimate peers & bots	Only bots
Communication protocols	Existing P2P protocols	Existing P2P protocols	Self-designed or existing P2P protocols
C&C styles	Pull or push	Pull or push	Pull or push

existing P2P protocols used by the host P2P networks for C&C communication. Besides, these protocols have been tested in P2P file-sharing applications for a long time, so they tend to be less error-prone than newly designed ones, and have nice properties to improve performance of P2P systems and mitigate network problems, such as link failure or churn. The following discussion is based on parasite and leeching P2P botnets, but bot-only botnet can adopt these protocols as well.

In P2P file-sharing systems, file index which is used by peers to locate the desired content, may be centralized (e.g., Napster), distributed over a fraction of the file-sharing nodes (e.g., Gnutella), or distributed over all or a large fraction of the nodes (e.g., Overnet). A peer can send out query message for the file it is searching for, and the message will be passed around according to the routing algorithm implemented in the system. The search will terminate when query hits are returned or the query message expires.

Botmasters can easily adopt the above procedure to disseminate commands in pull style. They can insert records associated with some predefined file titles or hash values into the index, but rather than putting the content location information, botnet commands are attached. In order to get commands issued by botmasters, bots periodically initiate queries for those files or hashes, and nodes who preserve the corresponding records will return query hits with commands encoded. In other words, bots subscribe the content, i.e., commands, published by botmaster.

Take the early version of Storm botnet [2] for example, it utilizes the Overnet, and implements pull C&C mechanism. In this botnet, every day there are 32 hash keys queried by bots to retrieve commands. These 32 keys are calculated by a built-in algorithm, which takes the current date and a random number from [0-31] as input. Therefore, when issuing a command, the botmaster needs to publish it under 32 different keys. Trojan.Peacomm botnet [1] employs the similar C&C design.

Compared to pull mechanism, implementation of push mechanism on existing P2P protocols is more complex. There are two major design issues:

- Which peers should a bot forward a command to?
- How to forward commands: using in-band (normal P2P traffic) or out-of-band messages (non-P2P traffic)?

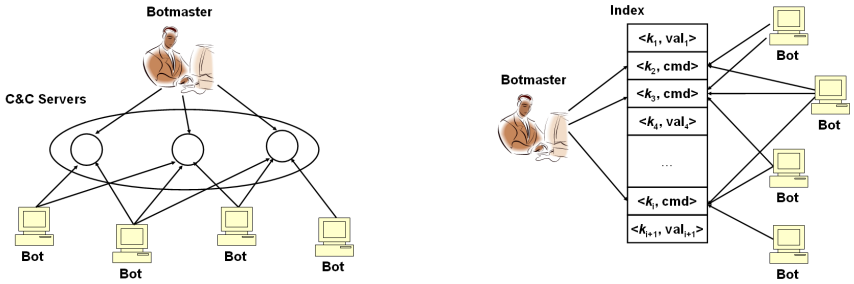
To address the first issue, the simplest way is to let a bot use its current neighboring peers as targets. But the problem of this approach is that command distribution may be slow or sometimes disrupted, because 1) some bots have a small number of neighbors, or 2) some peers in a bot's neighbor list are not bot members in the case of parasite or leeching P2P botnets. One solution to this problem is that letting bots claim they have certain popular files available which are predefined, and forwarding commands to peers appearing in the search results for those files. Thus the chance of commands hitting an actual bot is increased. These predefined popular files behave as the watchwords for the botnet, but could give defenders a clue to identify bots.

For the second issue, whether using in-band or out-of-band message to forward a command depends on what the peers in the target list are. If a bot targets its neighboring peers, in-band message is a good choice. A bot could encode a command in a query message, which can only be interpreted by bots, send it to all its neighbors, and rely on them to continue passing on the command in the botnet. This scheme is easy to implement and hard for defenders to detect, because there is no difference between command forwarding traffic and normal P2P traffic. On the other hand, if the target list is generated in other ways, like using peers in returned search results discussed above, bots have to contact those peers using out-of-band message. Obviously out-of-band traffic are easier to detect, and hence, can disclose the identities of bots who initiate such traffic.

The above discussion mainly focused on unstructured P2P networks, where query messages are flooded to the network. In structured P2P networks (e.g., Overnet), a query message is routed to the nodes whose node IDs are closer to the queried hash key, which means queries for the same hash key are always forwarded by the same set of nodes. Therefore, to let more bots receive a command, the command should be associated with different hash keys, such that it can be sent to different parts of the network.

2) Design A New P2P Communication Protocol: It is convenient to adopt existing P2P protocols for P2P botnet C&C communication, however, the inherited drawbacks may limit botnet design and performance. A botnet can be more flexible if it uses a new protocol designed by its botmaster.

The advanced hybrid P2P botnet [10] and the super botnet [11] are two newly designed P2P botnets, whose C&C communication are not dependent on existing P2P protocols. Both of them implements push and pull C&C mechanisms. In a hybrid P2P botnet, when a bot receives a command, it forwards the command to all the peers in the list (push), and those who cannot accept connection from others periodically contacts other bots in the list and try to retrieve new commands (pull). A super botnet is composed of a number of small centralized botnets. Commands are pushed from one small botnet to



(**a**) Centralized Botnet (**b**) Index-based P2P Botnet

Fig. 1: Similarity of logical C&C structures between traditional centralized botnets and index-based P2P botnets

another, and within a small centralized botnet, bots pull the command from their C&C servers. Furthermore, the hybrid P2P botnet is able to effectively avoid bootstrap procedure, which is required by most of the existing P2P protocols, by 1) passing a peer list from one bot to a host that is infected by this bot, and 2) exchanging peer lists when two bots communicate.

The drawback of designing a new protocol for P2P botnet communication is that the new protocol has never been tested before. When a botnet using this protocol is deployed, the network may not be as stable and robust as expected due to complex network conditions and defenses.

D. Discussion

Several features can be extracted to represent a P2P botnet during its lifetime: infection vectors, bot candidates, bootstrap procedure, members in the network, communication protocols and C&C styles. Parasite, leeching and bot-only P2P botnets, they share common features but differ in others, which have been summarized in Table I. It is shown that parasite P2P botnets are less flexible except that no bootstrap is required. This is an advantage of the parasite over the other two classes. During bootstrap, botnets are vulnerable and the propagation could be stopped if bootstrap information is compromised by defenders. Leeching and bot-only P2P botnets are more like each other, but the former ones are more stealthy, because they resides in existing P2P networks, and bot members are mixed with legitimate nodes and not easy to be detected.

III. COUNTERMEASURES

In this section, we exploit possible directions for P2P botnet detection and mitigation. It has been mentioned in literature [1], [2], [12] that index poisoning and sybil attack can be used to fight against two recent P2P botnets - Trojan.Peacomm and Stormnet. Therefore, we put more effort on these two defense techniques to explain how and why they work, how attackers can evade them, and give analytical studies to evaluate their performance. This research could provide guidance for security professionals and researchers to effectively use these techniques, and at the same time be aware of their limitations. Notations used in this section are explained in Table II.

A. Index Poisoning Technique

1) Defense Idea: Originally, index poisoning attack was introduced to prevent illegal distribution of copyrighted content in P2P networks. The main idea is to insert massive number of bogus records into the index. If a peer receives bogus record, it could end up not being able to locate the file (nonexistent location), or downloading the wrong file [13].

As we discussed in Section II-C, P2P botnets which implement C&C mechanism of command publishing/subscribing, make use of the indices in P2P networks to distribute commands. We refer such botnets as "*index-based*" P2P botnets. Trojan.Peacomm botnet [1] and Stormnet [2] are two typical examples. If defenders are able to figure out the index keys of the botnet command related index records, they can try to "poison" the index by announcing false information under the same keys. If the false information is overwhelmed over the real command content, when bots query to retrieve the commands, they may get false commands with a high probability. In this way, the C&C channels of the botnet are disrupted.

We believe there are three reasons that index-based P2P botnets are vulnerable to index poisoning defense.

First, a security defect of P2P protocol itself is the root cause. In most of the P2P networks, there is no central authority to manage the file index, such that any node no matter benign or malicious is able to insert records into the index, and there is no way to authenticate the identity of the node and content of the records.

Second, with the help of honeypot and reverse engineering techniques, defenders are able to analyze bot behaviors and bot code, and figure out the bot command related index keys.

Third, in some sense, the C&C architecture of this type of P2P botnets is similar to that of the traditional centralized botnets because of the limited number of index keys for command distribution. As shown in Fig. 1, in centralized botnets, commands are published at central sites, where bots are going to fetch the commands; on the other hand, in index-based P2P botnets, commands cmd are inserted in the index by botmasters under special index keys, such as k_2, k_3 and k_i which are known by bots and queried for retrieving commands.

From the aspect of C&C architecture, index-based P2P botnets logically rely on central points (predefined index keys), while traditional botnets physically rely on central points (predefined C&C servers) for communication. From the aspect of defense, for a traditional C&C botnet, defenders

shut down C&C channels by physically removing the C&C servers or blocking access to the servers; while for a P2P botnet, defenders overwhelm real command related records by many bogus records under the same keys (the basic idea of index poisoning technique) to disrupt C&C communication. Therefore, we can draw a conclusion that P2P botnets are not absolutely harder to defend than traditional centralized botnets. If index poisoning defense is valid for a P2P botnet, it is equally easy (or hard) to defend that P2P botnet compared to a centralized botnet.

2) Attackers' Counter-defense Idea: Although index poisoning is effective, it is still possible for attackers to evade it, if they can eliminate the causes discussed in Section III-A1.

Overbot [14], a new botnet protocol designed by Starnberger et al., addressed the second and the third issues. In Overbot, each bot generate its own index key for retrieving command and that key dynamically changes at a certain rate, in addition the communication between bots and sensors (nodes used by botmaster to publish commands) is encrypted. Thus, it is very difficult for defenders to crack or predict the index key, even though defenders are able to do it for one single bot, it is still not helpful, because each bot has different index key. However, for the same reason, sensors have to publish a <hash key, command> pair for each bot they know periodically, which increases the sensors' workloads and makes them more suspicious. In other words, the advantages of Overbot come with the lack of scalability and increase of detectability.

Now we present a novel and realistic method to deal with index poisoning defense—an authentication enforcement for command generation and index manipulation. It addresses the first cause of index-based P2P botnets being vulnerable to index poisoning (Section III-A1). In this approach, only botmasters can insert records to the command index preserved on bots. Bots can only query to fetch commands.

To realize the authentication, a botmaster generates a pair of public/private keys $<K^+, K^->$, and hard-codes the public key K^+ into the bot code. Later, when the botmaster wants to issue a command m under key k, he/she can insert a record $<k, m, K^-(H(m))>$ instead of $<k, m>$ into the index on a bot, saying bot A, where $H(m)$ is the hash value of m (i.e., the command is signed by the botmaster). Bot A can decide if the record is created by its botmaster by using the public key K^+ to verify the signature. If the signature is authentic, bot A stores this record in the index and waits for others to query, otherwise it discards the fake one. In this way, the index on a bot will not be polluted.

In addition, this authentication mechanism can prevent the spread of false commands. Even if defenders manage to store entries with forged commands in the index on controlled peers (e.g., honeypots infected by a captured bot binary), bots can verify the authenticity of received commands using the public key and disregard the false ones.

Most of existing P2P protocols have not implemented this kind of authentication mechanism. Thus in order to do it, attackers need to modify the existing protocols, which implies this counter index poisoning technique can only be applied to bot-only P2P botnets.

TABLE II: PARAMETERS USED IN ANALYSIS

Symbol	Meaning
m	The number of bits used to represent a node ID or a hash key.
b	The number of bits improved per step for a lookup.
n	The number of nodes in the network.
n_{tz}	The number of nodes in the target zone.
n_{index}	The number of nodes poisoned in the target zone.
n_{sybil}	The number of sybil nodes added to the target zone.
l_{tz}	The length of a search path in the target zone.
c	The number of the first bits in common with a hash key.
$P(success)$	The probability of a bot getting a real command.

3) Analytical Study: In this section, we give an analytical study on performance of index poisoning technique fighting against index-based P2P botnets. Our target is a P2P botnet like Trajon.Peacomm botnet and Stormnet, which implements Kad [15], a Kademlia-based distributed hash table (DHT) protocol [16], for C&C communication. Similar study can be conducted on P2P botnets utilizing other protocols.

In Kad network, a node is identified by a node ID consisting of m bits. In the DHT, each entry is a <hash key, value> pair, where the hash key contains m bits as well. A pair is stored on a node whose node ID is closest to the hash key in the network, and the distance is calculated by using XOR operator. In real applications, a <hash key, value> pair will be saved on k closest nodes for redundancy. Because of page limit, we cannot provide detailed introduction. For more information about Kademlia and Kad, please refer to [16], [17], [15].

If defenders want to pollute index records under hash key Key, they need to contact nodes (poisoned nodes) whose IDs are around Key, and store pairs like $<Key$, false value$>$ on them. So when a bot queries for Key to retrieve commands, hopefully those poisoned nodes will appear on the search path and return false value, without letting bots reach the k nodes who possess the real commands. As illustrated in Fig. 2, a bot node A initiates a lookup for index key Key, the search path are supposed to go through node B_1, B_2, B_3, B_4 and B_5, until it reaches node D who has the $<Key$, command$>$. If a pair $<Key$, false command$>$ has been added in the index on node B_4, when the lookup message gets node B_4, it would return the false command, and the search terminates.

We assume that node IDs are uniformly distributed over the entire Kad ID space. Actually this has been shown to be true in [15]. Suppose defenders choose n_{index} nodes whose IDs agree at least the first c bits in common with Key. We call the zone around Key the "*target zone*". Only when a lookup path enters the target zone, is it possible that a poisoned node will be chosen, return a search result and terminate the search. Let x be the probability of choosing a poisoned node in one step, then $1 - x$ is the probability of not choosing one. Therefore the probability of a bot obtaining the real command is

$$P(success) = (1 - x)^{l_{tz}} \tag{1}$$

where l_{tz} is the length of a search path within the target zone.

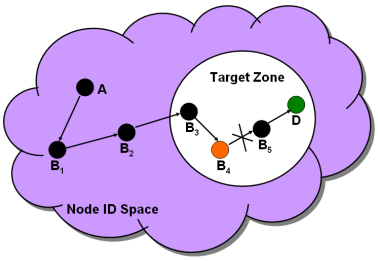


Fig. 2: A search path for a hash key, where node A is the initiator and node D is the destination. On this path node B_4 could be a node targeted by defenders to interrupt the search.

When a peer initiates a lookup for a hash key, in general, the expected number of steps required to perform a lookup is given as follows [17]:

$$l = \frac{\log_2 n}{b} \tag{2}$$

where n is the size of the network, and b is the number of bits improved per step, which depends on the structure of the routing table. Thus within the target zone, $l_{tz} = log_2 n_{tz}/b$. And since node IDs are uniformly distributed, the number of nodes in this zone $n_{tz} = n/2^c$, and $x = n_{index}/n_{tz}$. The complete formula to calculate $P(success)$ is

$$P(success) = (1 - \frac{2^c \cdot n_{index}}{n})^{(\log_2 n - c)/b} \tag{3}$$

According to Equation (3), the performance of index poisoning technique depends on four parameters. We have provided numerical results in Fig. 3 to show their impacts on $P(success)$ by changing the parameters.

Fig. 3(a) illustrates that a botnet would be more robust to index poisoning, if for each lookup more bits can be improved, i.e., the average length of search path is short, this is because poisoned nodes have less chance to be chosen along the path[1].

It is shown in Fig. 3(b) that in order to achieve better performance, defender could choose a larger c, i.e. choosing nodes which are closer to the command related hash key to poison. However it is not always a good idea to choose a large c, because we want there are at least one step along the lookup path falls in the target zone, otherwise bots can directly get commands without going through a poisoned node. In other words, $l_{tz} \geq 1$, i.e., $c \leq \log_2 n - b$. In our case, $n = 980,000$, $b = 3.25$, so $c \leq 16.7$, and for the setup $c = 9$ and $c = 10$, l_{tz} is 3.35 and 3.05 respectively. It is noticed that the x axis in Fig. 3(b) is the number of poisoned nodes in target zone, rather than the percentage which is used in Fig. 3(a). This is because with a low cost of poisoning a node, defenders can try as many nodes as possible, in order to increase the probability of a poisoned node being chosen. Moreover, as we can see obviously, given a fixed percentage of poisoned nodes, larger c would decrease the performance, since there are less steps of a search left within the target zone.

[1]The value of b was estimated in [17]. 3.25 is the worst case, while 6.98 is the best case.

In addition, the size of the network would also affect the performance of index poisoning technique. However, it does not matter that much, since given a fixed percentage of poisoned nodes in target zone, it can barely change l_{tz} because of the log_2 operator ($log_2 980000 = 19.90$ and $log_2(12000 \times 2^8) = 22.29$)[2].

B. Sybil Attack Technique

1) Defense Idea: A Sybil attack is referred as the forging of multiple identities to subvert the reputation system in a P2P network [18]. The reason of P2P networks being vulnerable to sybil attack is that peers can join the network without authentication or validation of their identities. It is troublesome for existing P2P networks [19], [20].

However, the same idea can be used to defend against index-based P2P botnets. With the knowledge of index keys which are used for command distribution, defenders can add sybil nodes into the botnet to re-route or monitor the command related traffic. How to set up sybil nodes depends on the P2P system implementation. In unstructured P2P network, in order to capture more traffic, it would be good for sybils to be peers with more important roles, such as ultrapeers in Gnutella, because only ultrapeers are allowed to forward messages. In structured P2P network, such as Kad, the node IDs of sybil nodes should not be chosen randomly, but be close to a known command related index hash key, as discussed in [2], [12], such that query traffic for the key will go through the sybil nodes with high probability according to the Kad's routing algorithm. We call such defense "*targeted*" sybil attack.

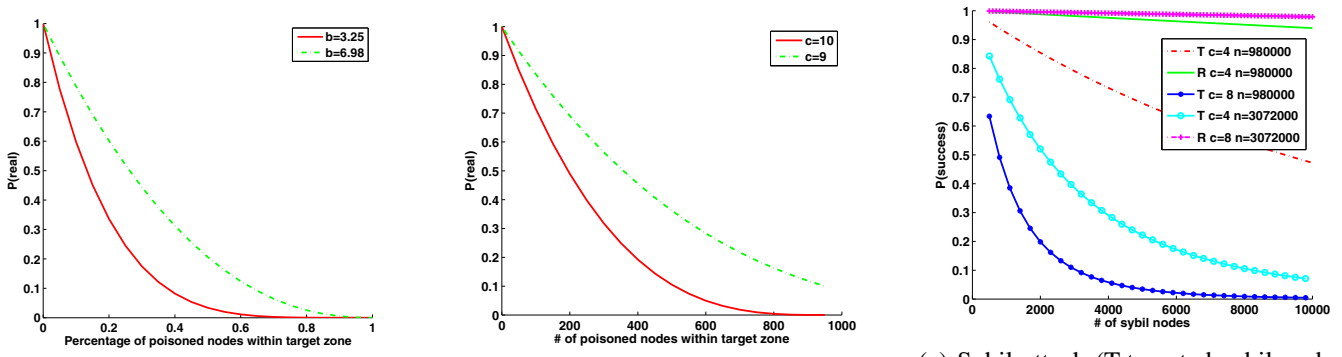
The cost for sybil attack defense is higher than index poisoning defense. This is because either a physical or a virtual machine is needed to set up a sybil node, in other words, more sybil nodes require more computer resources, while publishing different records to poison index can be done by a single node.

2) Attackers' Counter Defense Idea: Similarly, approaches used for protect today's P2P networks from Sybil attack may also work for attackers to prevent defenders from infiltrating their botnets using sybil nodes.

In Kademlia-based P2P networks [16], a node ID can be constructed by hashing the node's IP address as in Chord [21], rather than being randomly generated like what Kad does [15], such that sybil nodes cannot choose any IDs they want. When applying this scheme in such a P2P botnet, defenders cannot target a specific hash key to set up their sybil nodes. And if the sybil nodes are just randomly added in the botnet, which is referred as "*random*" sybil attack, it would be less effective. This will be shown in the next section.

Furthermore, caching technique [16], which was meant to solve "hot spots" problem, can also reduce the effectiveness of sybil attack defense technique. Because the command related index records will be stored not only on bots that were chosen at the beginning by their botmaster (e.g., unstructured network) or according to the protocol (e.g., structured network), but

[2]An estimate was given in [17] that the Kad network has around 980,000 concurrent peers. In [15], they claimed the population of peers in Kad network is between 12,000$\times 2^8$ and 20,000$\times 2^8$.



(**a**) Index poisoning with different b

(**b**) Index poisoning with different c

(**c**) Sybil attack (T-targeted sybil nodes, R-random sybil nodes)

Fig. 3: Defense performance of index poisoning and sybil attack techniques

also on bots that may not be easily identified. Thus even targeted sybil attack cannot cover all the range that the bots who possess the command information fall in.

3) Analytical Study: Now we try to analyze sybil attack on the same cluster of P2P botnets as in III-A3, Kad-based P2P botnets. The notations have the same meaning unless mentioned.

If node IDs can be chosen randomly, defenders can create special n_{sybil} sybil nodes, whose node IDs share at least the first c bits with index key Key, and add them into the botnet. Once a sybil is chosen on the path of a command lookup, it can re-route the message or return a false command and terminate the search. No matter what, the bot will not be able to obtain the real command.

As we can see, sybil attack shares the same idea with index poisoning technique. They both try to manipulate the command lookup path, as shown in Fig. 2. However, the former tries to do it by adding new special nodes (controlled by defenders) to the network, i.e., node B_4 in Fig. 2 is a sybil node added by defenders, while the latter by working on the nodes (bots probably) already in the network.

Following the same analysis logic as what we did III-A3, the probability of bot successfully getting the real command $P(success)$ can be calculated using Equation (1), except x becomes the probability of choosing a sybil at one step along the search path, which is $\frac{n_{sybil}}{n_{sybil}+n_{tz}}$. So

$$P(success) = (1 - \frac{n_{sybil}}{n_{sybil} + n_{tz}})^{l_{tz}} \quad (4)$$

where $l_{tz} = log_2(n_{sybil} + n_{tz})/b$, and $n_{tz} = n/2^c$.

However, if a verification mechanism for node ID is applied in the botnet (Section III-B2), which means defenders cannot choose node IDs for sybil nodes as they wish, then the whole network becomes a target zone, i.e., $n_{tz} = n$. Simply substituting n_{tz} in Equation (4) by n, we can get the following formula to compute $P(success)$ in this "random" sybil attack.

$$P(success) = (1 - \frac{n_{sybil}}{n_{sybil} + n})^{log_2(n_{sybil}+n)/b} \quad (5)$$

It is shown in Fig. 3(c) that under the same circumstance targeted sybil attack outperforms the random one, because in the former case, sybil nodes with specially chosen IDs have more chances to appear along a search path than those in the latter case. With limited resources that defenders may use to launch sybil attack, if the sybil nodes are closer to the hash key Key (i.e., larger c), the performance is better. Furthermore, sybil attack is more effective if the network is smaller.

C. Others

In the following, we present several general ideas to counter P2P botnets.

1) Detection: Being able to detect bot infection can stop a new-born botnet in its infant stage. Signature-based malware detection is effective and still widely used. But anti-signature techniques, such as polymorphic technique [22], make it possible for malware to evade such detection systems. Therefore, instead of doing static analysis, defenders start considering dynamic information for detection. The system proposed by Gu et al. [23] is one based on dynamic pattern matching.

Anomaly detection is another direction, since bots usually exhibit different behaviors from legitimate P2P users, such as sending queries periodically, always querying for the same content, or repeatedly querying but never downloading.

Besides, distributed detection is another approach, such as a self-defense infrastructure presented in [24], and two approaches against ultra-fast topological worms in [25].

2) Monitoring: Monitoring botnets help people better understand their motivations, working patterns, evolution of designs, etc. There are two effective ways to conduct P2P botnet monitoring.

For parasite and leeching P2P botnets, we can choose legitimate nodes in the host P2P networks as sensors for botnet monitoring. Usually sensors are peers that play important roles in the network communication, such as ultrapeers in Gnutella networks, thus more information can be collected. In DHT-based P2P networks, the search path of a specific hash key is relatively fixed, even if the search starts at different nodes. So the sensor selection depends on the monitoring targets and routing algorithm implemented in the system.

Honeypot techniques [26] are widely used for botnet monitoring. The way to set up honeypots in a P2P botnet is similar to choosing sensors. The difference is that honeypots are hosts added to the network on purpose by defenders, while sensors are chosen from the nodes who are already in the network.

3) Shutdown: The ultimate purpose of studying botnets is to shut them down. We can either 1) remove discovered bots, or 2) shut down C&C channels of botnets.

Botnet construction relying on bootstrapping is vulnerable during its early stage. Isolating or shutting down bootstrap servers or the bots in the initial list that are hard-coded in bot code can effectively prevent a new-born botnet from growing into a real threat.

P2P botnets can also be shut down or partially disabled by removing bot members. There are two modes of bot removal: random and targeted. The former means disinfecting the host whenever it is identified as a bot. The latter means removing critical bots, such as the ones that are important for C&C communication, when we have the knowledge of the topology or C&C architecture of a P2P botnet. Two metrics to evaluate the effectiveness of targeted removal were proposed in [10].

Shutting down detected bots is slow in disabling a botnet and sometimes impossible to do (e.g., you have no control of infected machines abroad). So a more effective and feasible way is to interrupt botnet C&C communication such that bots cannot receive orders from their botmaster. This approach has been carried out well for centralized botnets through shutting down the central C&C sites, but is believed to be more difficult to do for P2P botnets.

However, we find that this general understanding of "*P2P botnet is much more robust against defense*" is misleading. In fact, index-based P2P botnets are as vulnerable as centralized botnets, if the counter defense methods we presented in Section III-A2 and III-B2 are not implemented. Index poisoning technique (Section III-A) and sybil attack technique (Section III-B) can be quite effective to fight against such botnets.

IV. Conclusion

P2P botnets as a new advanced form of botnets have attracted people's attention. In this paper, we presented a systematical study on P2P botnets. We detailedly discussed each stage in the lifetime of P2P botnets, and classified P2P botnets into three categories: parasite, leeching and bot-only P2P botnets. Among possible directions for P2P botnet detection and mitigation that were pointed out, we focused on two effective defenses against P2P botnets: index poisoning and sybil attack, and were able to give analytical studies to evaluate their defense performance. In addition, we discussed how attackers might react to avoid or reduce the effectiveness of these two defenses. Furthermore, we have shown that logically the C&C structure of P2P botnets that rely on index for command dissemination is similar to that of traditional centralized botnets, therefore, they may be as easy (or as hard) to be shut down as the centralized botnets.

References

[1] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon, "Peer-to-peer botnets: Overview and case study," in *Proc. of the 1st USENIX Workshop on Hot Topics in Understanding Botnets (HotBots '07)*, April 2007.

[2] T. Holz, M. Steiner, F. Dahl, E. W. Biersack, and F. Freiling, "Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm," in *Proc. of the 1st Usenix Workshop on Large-scale Exploits and Emergent Threats (LEET '08)*, 2008.

[3] G. Hulme, "New and improved storm botnet morphing malware," Febrary 2009, http://www.informationweek.com/blog/main/archives/2009/02/new_and_improve.html.

[4] F. Kuhn, S. Schmid, and R. Wattenhofer, "A Self-Repairing Peer-to-Peer System Resilient to Dynamic Adversarial Churn," in *Proc. of the 4th Int. Workshop on Peer-to-Peer Systems (IPTPS '05)*, February 2005.

[5] "Peer-to-peer data mining, privacy issues, and games," White paper, University of Maryland, April 2007.

[6] Http://www.symantec.com/security_response/index.jsp.

[7] Http://www.viruslist.com/en/viruses/encyclopedia?virusid=24282.

[8] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich, "Analysis of the storm and nugache trojans: P2p is here," *USENIX ;login:*, vol. 32, no. 6, pp. 18–27, December 2007.

[9] Http://www.gnucleus.com/gwebcache/.

[10] P. Wang, S. Sparks, and C. C. Zou, "An advanced hybrid peer-to-peer botnet," in *Proc. of the 1st USENIX Workshop on Hot Topics in Understanding Botnets (HotBots '07)*, April 2007.

[11] R. Vogt, J. Aycock, and M. Jacobson, "Army of botnets," in *Proc. of the 14th Network and Distributed System Security Symp. (NDSS '07)*, Febuary 2007.

[12] C. R. Davis, J. M. Fernandez, S. Neville, and J. McHugh, "Sybil attacks as a mitigation strategy against the storm botnet," in *Proc. of the 3rd Int. Conf. on Malicious and Unwanted Software (Malware '08)*, 2008.

[13] J. Liang, N. Naoumov, and K. W. Ross, "The index poisoning attack in p2p file sharing systems," in *Proc. of the IEEE INFOCOM*, April 2006.

[14] G. Starnberger, C. Kruegel, and E. Kirda, "Overbot - a botnet protocol based on kademlia," in *Proc. of the 4th Int. Conf. on Security and Privacy in Communication Networks (SecureComm '08)*, September 2008.

[15] M. Steiner, T. En-Najjary, and E. W. Biersack, "A global view of kad," in *Proc. of the ACM Internet Measurement Conf. (IMC '07)*, 2007.

[16] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Proc. of the 1st Int. Workshop on Peer-to-Peer Systems (IPTPS '02)*, 2002.

[17] D. Stutzbach and R. Rejaie, "Improving lookup performance over a widely-deployed dht," in *Proc. of the IEEE INFOCOM*, April 2006.

[18] J. R. Douceur, "The sybil attack," in *Proc. of the 1st Int. Workshop on Peer-to-Peer Systems (IPTPS '02)*, March 2002.

[19] M. Steiner, T. En-Najjary, and E. W. Biersack, "Exploiting kad: possible uses and misuses," *Computer Communication Review*, vol. 37, no. 5, pp. 65–70, October 2007.

[20] P. Wang, J. Tyra, E. Chan-Tin, T. Malchow, D. F. Kune, N. Hopper, and Y. Kim, "Attacking the kad network," in *Proc. of the 4th Int. Conf. on Security and Privacy in Communication Netowrks (SecureComm '08)*, August 2008.

[21] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications, ac," in *Proc. of the ACM SIGCOMM*, August 2001.

[22] O. Kolesnikov, D. Dagon, and W. Lee, "Advanced polymorphic worms: Evading ids by blending in with normal traffic," Georgia Tech, Tech. Rep., 2004-2005.

[23] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "BotHunter: Detecting malware infection through ids-driven dialog correlation," in *Proc. of the 16th USENIX Security Symp. (Security '07)*, August 2007.

[24] L. Zhou, L. Zhang, F. McSherry, N. Immorlica, M. Costa, and S. Chien, "A first look at peer-to-peer worms: Threats and defenses," in *Proc. of the 4th Int. Workshop on Peer-To-Peer Systems (IPTPS '05)*, 2005.

[25] L. Xie and S. Zhu, "A feasibility study on defending against ultra-fast topological worms," in *Proc. of The 7th IEEE Int. Conf. on Peer-to-Peer Computing (P2P '07)*, September 2007.

[26] L. Spitzner, "Honeypots," 2003, http://www.tracking-hackers.com/papers/honeypots.html.