Peer-to-Peer Botnets: Overview and Case Study

Julian B. Grizzard
julian.grizzard@jhuapl.edu
The Johns Hopkins University
Applied Physics Laboratory

Vikram Sharma, Chris Nunnery,
and Brent ByungHoon Kang
{vsharma, cenunner, bbkang}@uncc.edu
University of North Carolina at Charlotte

David Dagon
dagon@cc.gatech.edu
Georgia Institute of Technology

Abstract

Botnets have recently been identified as one of the most important threats to the security of the Internet. Traditionally, botnets organize themselves in an hierarchical manner with a central command and control location. This location can be statically defined in the bot, or it can be dynamically defined based on a directory server. Presently, the centralized characteristic of botnets is useful to security professionals because it offers a central point of failure for the botnet. In the near future, we believe attackers will move to more resilient architectures. In particular, one class of botnet structure that has entered initial stages of development is peer-to-peer based architectures. In this paper, we present an overview of peer-to-peer botnets. We also present a case study of a Kademlia-based Trojan.Peacomm bot.

1 Introduction

One of the most significant threats to the Internet today is the threat of botnets, which are networks of compromised machines under the control of an attacker. It is difficult to measure the extent of damage caused on the Internet by botnets, but it is widely accepted that the damage done is significant. Further, the potential for orders of magnitude more damage exists in the future.

The beginning of botnets can be traced back to basic forms of benign bots. The *EggDrop* bot is one of the earliest popular bots used for automating basic tasks on Internet relay chat (IRC). Today, there are many botnets that use IRC as a form of centralized command and control (C&C). The basic scripting tasks that a benign bot such as *EggDrop* offers can also be used to coordinate bots.

A number of ad hoc methods exist to detect and stop botnets, and these methods continue to mature. As techniques for botnet detection and mitigation advance, the robustness and resiliency of botnets will also advance. Today, the most easily detected botnets use IRC as a form of communication for command and control (C&C). IRC has many properties that make it attractive for an attacker such as its redundancy, scalability, and versatility. Further, there is a large base of knowledge and source code for developing IRC-based bots. Many botnet authors reuse existing code in order to create their own botnet.

One key property of IRC-based botnets is the use of IRC as a form of central C&C. This property provides the attackers with very efficient communication. However, the property also serves as a major disadvantage to the attacker. The threat of the botnet can be mitigated and possibly eliminated if the central C&C is incapacitated. It is likely that new architectures will emerge as the ability to stop IRC-based botnets matures.

One such architecture that is beginning to appear is a peer-to-peer structure for botnet communication. In a peer-to-peer architecture, there is no centralized point for C&C. Nodes in a peer-to-peer network act as both clients and servers such that there is no centralized coordination point that can be incapacitated. If nodes in the network are taken offline, the gaps in the network are closed and the network continues to operate under the control of the attacker. In this paper, we focus our work on peer-to-peer botnets.

Today, attackers are able gain control of significant portions of the Internet using centralized C&C architectures. However, we are beginning to see peer-to-peer architectures with bots such as the Trojan.Peacomm that we study in this work. The long term goal of our work is to develop methods of detecting, mitigating, and preventing peer-to-peer botnets. In order to reach this goal, this work focuses on increasing the understanding of peer-to-peer botnets by (1) providing an overview and historical perspective and (2) presenting a case study of a Trojan.Peacomm bot.

2 Background and History

In order to better understand botnets, we first define some key terms. Then, we present a timeline of the significant events that relate to bots and peer-to-peer protocols in terms of technological developments. Based on this review of historical trends, we believe that peer-to-peer botnets will be one of the most significant threats on the Internet in the near future.

2.1 Definitions

We define peer-to-peer, bot, and botnet below.

- *peer-to-peer* – A peer-to-peer network is a network in which any node in the network can act as both a client and a server.
- *bot* – A bot is a program that performs user centric tasks automatically without any interaction from a user.
- *botnet* – A botnet is a network of *malicious* bots that illegally control computing resources.

Some definitions of peer-to-peer networks require no form of centralized coordination. Our definition is more relaxed because the attacker may be interested in hybrid architectures. Our definition of a bot is not inherently malicious. However, the malicious nature of a bot is implicit under some contexts. Finally, we do define a botnet to be malicious in nature.

2.2 History

Table 1 provides an overview of some important bots and peer-to-peer protocols. The timeline ranges from the one of the earliest bots, *EggDrop*, through the Trojan.Peacomm peer-to-peer bot recently released. More recent years have seen significant developments of malicious bots. In particular, the first peer-to-peer bots are beginning to emerge, such as the Trojan.Peacomm bot.

Not shown in Table 1 is a timeline of worms. Worms can serve as one form of a delivery mechanism for bots. Although worms are relevant to the development of botnets, they are more relevant to the spread of bots than to the botnet communication after infection. Our work focuses on the communication mechanism in place after the botnet has spread to its victims. Kienzle et al. provide a survey of worms [1].

During the early stages of the Internet, a non-malicious bot was developed called *EggDrop*. There were likely many other bots developed prior to *EggDrop*. However, *EggDrop* is recognized as one of the first popular Internet relay chat (IRC) bots. Example non-malicious uses of *EggDrop* include playing games (i.e., Turing test), coordinating file transfer (legally transferred files), automating channel admin commands, etc. Thus, the early bot developments seem to have been motivated by simply improving automation on the Internet.

The *GTBot* variants are one of the earliest wide-known malicious bots. There are likely many prior malicious bots. *GTBot* variants included an IRC client, *mIRC.exe*, as part of the bot [2]. This bot represents some of the early trends to use IRC as a form of coordinating botnets.

Independent of botnet activity, we believe that peer-to-peer protocols came into prominence with the release of *Napster*. The Napster client was built as an application that allowed peers to find and share music files with other peers in the network. File indexing was done on a centralized server, so Napster is not entirely a peer-to-peer service. Users would connect to the centralized server in order to upload an index of their files and search for files on other user's computers. If a particular file was found, the user would directly connect to another peer in order to retrieve the file. Because many of the music files shared between users were illegally traded, a court found *Napster's* service illegal and the service was shutdown. Later variants of peer-to-peer file sharing focused on evading authorities by avoiding centralized control.

Although not entirely motivated by the shutdown of *Napster*, completely decentralized peer-to-peer services began to emerge after *Napster* was shutdown. The *Gnutella* protocol marks the beginning of completely decentralized peer-to-peer services. There are numerous peer-to-peer protocols developed since the release of Gnutella, as seen in Table 1, which were designed to be as resilient, efficient, and reliable as possible. Recent peer-to-peer protocols such as Chord [3] and Kademlia [4] have introduced distributed hash table as efficient methods for finding information in peer-to-peer networks. Peer-to-peer networks offer design characteristics that are attractive to attackers.

Malicious bots have seen much development in the recent years. *Agobot* variants are possibly one of the most widespread bots due to its well designed and modular code base [2]. In our opinion, *Agobot* marks a turning point in which botnets have become a more significant threat.

Finally, as Table 1 shows, peer-to-peer bots are now under widespread development. Some peer-to-peer bots have used existing peer-to-peer protocols while others have developed custom protocols. We predict that peer-to-peer botnets will mature to a level in which they might become more widespread than traditional decentralized C&C architectures.

Date	Name	Type	Distinguishing Description
12/1993	EggDrop	Non-Malicious Bot	Recognized as early popular non-malicious IRC bot
04/1998	GTbot Variants	Malicious Bot	IRC bot based on mIRC executables and scripts
05/1999	Napster	Peer-to-Peer	First widely used hybrid central and peer-to-peer service
11/1999	Direct Connect	Peer-to-Peer	Variation of Napster hybrid model
03/2000	Gnutella	Peer-to-Peer	First decentralized peer-to-peer protocol
09/2000	eDonkey	Peer-to-Peer	Used checksum directory lookup for file resources
03/2001	Fast Track	Peer-to-Peer	Use of supernodes within the peer-to-peer architecture
05/2001	WinMX	Peer-to-Peer	Proprietary protocol similar to FastTrack
06/2001	Ares	Peer-to-Peer	Has ability to penetrate NATs with UDP punching
07/2001	BitTorrent	Peer-to-Peer	Uses bandwidth currency to foster quick downloads
04/2002	SDbot Variants	Malicious Bot	Provided own IRC client for better efficiency
10/2002	Agobot Variants	Malicious Bot	Incredibly robust, flexible, and modular design
04/2003	Spybot Variants	Malicious Bot	Extensive feature set based on Agobot
05/2003	WASTE	Peer-to-Peer	Small VPN-style network with RSA public keys
09/2003	Sinit	Malicious Bot	Peer-to-peer bot using random scanning to find peers
11/2003	Kademlia	Peer-to-Peer	Uses distributed hash tables for decentralized architecture
03/2004	Phatbot	Malicious Bot	Peer-to-peer bot based on WASTE
03/2006	SpamThru	Malicious Bot	Peer-to-peer bot using custom protocol for backup
04/2006	Nugache	Malicious Bot	Peer-to-peer bot connecting to predefined peers
01/2007	Peacomm	Malicious Bot	Peer-to-peer bot based on Kademlia

Table 1: Timeline of Peer-to-Peer Protocols and Bots

3 Goals and Metrics

Botnets have a set of common goals and metrics. Peer-to-peer botnets are distinctive from centralized C&C botnets in that they focus on resiliency through the uses of a peer-to-peer network. However, peer-to-peer botnets are similar to centralized botnets in most other aspects. Below is an overview of the goals and metrics of botnets with distinctive highlights for peer-to-peer botnets.

The primary goals of botnets fall under one of three categories: *information dispersion*, *information harvesting*, and *information processing*. An attacker may not be motivated by these goals and perhaps creates the botnet for fun or fame; however, we focus on goals that clearly indicate economic incentive as we believe these goals are the most dangerous. The goal of *information dispersion* includes sending out spam, creating denial of service attacks, providing false information from illegally controlled sources, etc. The goal of *information harvesting* includes obtaining identity data, financial data, password data, relationship data (i.e., email addresses of friends), and any other type of data available on the host. The goal of *information processing* is to process data such as cracking a password stored as a MD5 hash.

Information dispersion has economic benefit because a buyer may wish to pay a botnet controller to disperse spam in some cases or to halt a denial of service attack in other cases. *Information harvesting* has direct economic benefits because a buyer may wish to pay the botnet controller for the information or the botnet controller may be able to get money directly (i.e., a harvested credit card number). An attacker could sell *information processing* as a service or could use the processing capability to crack passwords for access to additional hosts.

A botnet needs basic computing resources to accomplish its goals including *CPU cycles*, *network*, *memory*, and *other* resources. Table 2 summarizes these resources. The table also lists metrics for each resource that can be used to characterize botnets. The distinguishing characteristics of peer-to-peer botnets are the network characteristics. In particular, peer-to-peer botnets communicate with other peer bots rather than a central server, so the communication graph will be distinctive. Also, we would expect the command latency to be higher for peer-to-peer botnets.

Table 3 shows methods of infection. The table summarizes the method of primary infection upon which many different methods of secondary infection can be executed. In our case study, the Trojan.Peacomm bot uses a Trojan horse as a method of primary infection and a peer-to-peer network for secondary infection.

4 Case Study: Trojan.Peacomm

The Trojan.Peacomm bot is the most recently known peer-to-peer bot to date. The Trojan.Peacomm botnet uses the Overnet peer-to-peer protocol for controlling the bots. The Overnet protocol implements a distributed

Resource	Metrics
CPU cycles	MIPS
	Command list
network	Mbps
	IP list
	Port list
	Communication graph
	Command latency
memory	MB storage
	MB information
	Value/bit
other	Time unit, size unit, etc.

Table 2: Botnet Resource Requirements and Metrics

Type	Description
server	Actively exploit remote service
client	Passively exploit client process
Trojan horse	Exploit trust of privileged program
physical	Tamper with physical computer
other	Other methods to control execution

Table 3: Infection Vectors

hash table based on the Kademlia algorithm as described in [4]. After infection, secondary injections are automatically downloaded from the peer-to-peer network, which provides a basic communication primitive from the attacker to the infected hosts. This peer-to-peer communication primitive enables the attacker to arbitrarily upgrade, control, or otherwise command infected hosts without relying on a central server.

In January 2007, we observed a production machine that was infected with a *Trojan.Peacomm* bot. We analyzed the malicious Trojan horse binary, the secondary injections, and the network traces of the infection. Further, we ran the malicious binary in a controlled honeypot environment at the UNCC Honeynet Laboratory. We believe this malicious bot represents a significant step toward more sophisticated peer-to-peer botnets. Below is our analysis and discussion of this bot.

4.1 Experimental Setup

In order to examine the Peacomm specimen, it was executed within a honeypot environment [5]. The honeypot consisted of a VMWare GSX 3.2 virtual machine running Windows XP. The connection to the Internet was filtered with a honeywall in order to prevent the honeypot from attacking machines on the Internet. The PerilEyez malware analysis tool was used to detect changes in the system [6]. Further, a pcap log of the entire session was kept for network analysis. The specimen was run for a period of two weeks under a carefully controlled environment.

4.2 Initial Bot

The Trojan.Peacomm binary is an executable that installs the initial bot on a victim. The initial bot has enough functionality to maintain persistence and connect to the peer-to-peer network in order to download secondary injections containing the payload functionality. The attacker can change the secondary injections in order to change functionality of bots on infected hosts.

Typically, the binary is distributed in the form of a Trojan horse email in order to infect victims, but any infection vector described in Table 3 is possible. In most observed cases, a victim receives an email with an attachment that is named “FullVideo.exe,” or some variant, along with some enticing text that urges the user to open the seemingly innocent attachment. The attachment appears to be a video, but in fact it installs the initial bot on the user’s computer. The Trojan targets Windows operating systems including Windows 95/98/ME/2000/NT/XP [7].

We analyzed the Trojan.Peacomm binary using the PerilEyez tool [6]. Instances of the file system, open ports, and running services on the system are captured prior to and following malware infection. Comparing these two images reveals changes made to the system environment as a result of the malware’s execution.

The Trojan.Peacomm binary sets up the initial bot by adding the system driver “wincom32.sys” to the host. This driver is injected into the Windows process “services.exe”. This service then acts as the peer-to-peer client that downloads the secondary payload injections. Additionally, Trojan.Peacomm disables the Windows firewall. The setting for the ICF/ICS service (Internet Connection Firewall / Internet Connection Sharing) is changed from “manual” to “disabled.” Presumably, this step is taken to ensure proper communication with peers. The following ports are opened:

```
TCP: 139, 12474
UDP: 123, 137, 138, 1034, 1035, 7871, 8705, 19013, 40519
```

The first packets sent by this piece of malware are for the bootstrap process to become part of the Overnet network. In order to bootstrap onto the Overnet network, the bot includes a list of nodes that are presumably Overnet nodes likely to be online. The initial peer list is created by the installation process into the file %windir%\system32\wincom32.ini. This peer list is hard-coded into the bot’s installation binary. It is not clear how the attacker chose these nodes. Conceivably, the list could be updated with each successful propagation cycle. The inclusion of initial Overnet bootstrap

```
[peers]
1: <128 bit md4 hash>=<IP address><Port><2 byte flag>
2: <128 bit md4 hash>=<IP address><Port><2 byte flag>
   ...
N: <128 bit md4 hash>=<IP address><Port><2 byte flag>
```

Figure 1: Format of wincom32.ini file

nodes could prove to be a centralized point of failure if the attacker does not have a method to change the bootstrap nodes for different infections.

Figure 1 shows the format of the peer list file. The peer list in our specimen contains 146 lines, each composed of two segments: a 128 bit MD4 peer hash represented in hexadecimal format and a node ID consisting of an IP address, port number, and an unknown flag. An equals sign acts as a delimiter between the two. These peers are used to bootstrap onto the Overnet network. Although the nodes as a collection act as a central point of failure, the file contains 146 nodes, so it may prove difficult to ensure all 146 nodes fail. However, monitoring traffic to these nodes could provide a measurement for the size of the Trojan.Peacomm botnet.

4.3 Communication Protocol

A botnet needs a basic communication protocol between the attacker and the bots. In centralized architectures, the protocol is fairly simple. The clients connect to the central server and wait for commands. Peer-to-peer botnets have more flexibility. The Trojan.Peacomm bot provides one such method for the attacker to issue commands to bots in a peer-to-peer architecture. Essentially, the bot downloads a secondary injection that can be arbitrary, which allows flexibility in the payload of the bot.

In order to download the secondary injection, the bot uses the Overnet network. Overnet is a Kademlia-based protocol, which provides an efficient method to locate values that correspond to given search keys [4]. For a more detailed discussion of Kademlia, see [4]. The important concepts of the Kademlia-based Overnet protocol are summarized below.

– A common 128-bit numeric space is used.
– Node IDs are within the numeric space.
– Values are mapped into numeric space with keys.
– Key/value pairs are stored on the “closest” nodes.
– “Close” is calculated by an XOR function.
– List of nodes kept for each bucket in numeric space.

Based on our analysis of the network trace data, the communication protocol for the Trojan.Peacomm bot can be divided into five important steps as described below:

1. *Connect to Overnet* – The bot publishes itself on the Overnet network and connects to peers. The initial list of peers is hard coded in the bot.
2. *Download Secondary Injection URL* – The bot uses hard coded keys to search for and download a value on the Overnet network. The value is an encrypted URL that points to the location of a secondary injection executable.
3. *Decrypt Secondary Injection URL* – The bot uses a hard coded key to decrypt the downloaded value, which is a URL.
4. *Download Secondary Injection* – The bot downloads the secondary injection from a web server using the decrypted URL.
5. *Execute Secondary Injection* – The bot executes the secondary injection, possibly scheduling future upgrades on the peer-to-peer network or scheduling bot stat tracking at some other resource.

There are a few interesting properties with the communication protocol. First, the initial list of peers is a weakness. If these peers stop responding to requests to join the Overnet network, then the Trojan.Peacomm binary will fail to bootstrap and download secondary injections. Also, these nodes could be monitored in order to detect possible infected hosts.

Another interesting observation is that the peer-to-peer protocol is essentially being used as a name resolution server for upgrading the bot. In previous bots that used DNS or dynamic DNS, the botnet can be incapacitated if the owner of the DNS registry cooperates with authorities. In the case of the equivalent peer-to-peer DNS, there is not a clear authority that can control the peer-to-peer content, especially since the data is encrypted. If the data is encrypted/decrypted with a public/private key pair, then it would also be challenging to fake the URL.

4.4 Secondary Injections

At the time of writing, Peacomm is designed to progress through a variety of secondary injections, including: (1) downloader and rootkit component, (2) SMTP email spamming component, (3) email address harvester for the previous spamming stage, (4) email propagation component, and (5) distributed denial of service tool [8]. These secondary injections can all be rooted from one secondary injection retrieved from the peer-to-peer network. Also, the secondary injections can be changed if the value is changed for the given key. Further, the bot can be programmed so that it periodically updates itself by searching through the peer-to-peer network. These basic primitives provide the attacker with botnet command and control.

The peers transfer files that contain URLs for the actual payload. To successfully exchange secondary injection URLs, Peacomm requires only the search response containing the meta tag and result hash as described in [8], and we also see these results. Following the delivery of a secondary injection URL, the secondary payloads are downloaded via HTTP on the compromised machine. In our analysis, secondary injections were downloaded from the URL *http://XXX.XXX.XXX.XXX/aff/dir/* where *xxx.xxx.xxx.xxx* is an IP address. There were different payloads for the production machine infection than our later tests showed, which seems to indicate that the attacker upgraded the secondary injections.

According to [8], the search key for secondary injection is generated using a built-in algorithm that uses the current date and a random number from [0..31] as input to the algorithm. This means that the botmaster needs to publish a new URL under 32 different keys for a particular day. One interesting problem with this algorithm is that some machines do not keep accurate clocks. We have not studied exactly how the algorithm uses the date as input, but presumably this could prevent bots from locating the secondary injections if their clock is not accurate.

An interesting defense strategy for this Overnet architecture is index poisoning. Liang et. al describe index poisoning in Overnet and FastTrack in [9]. In the case they describe, the motivations for index poisoning are different as they are analyzing techniques related to file sharing of copyrighted materials. However, index poisoning could also be applied to bots such as the Peacomm bot. For example, index poisoning could be used in order to slow the infection rate of the bot or possibly to measure the number of bots infected. We plan to study these methods in our future work.

4.5 Network Trace Analysis

We have analyzed a trace of an infection of the Trojan.Peacomm on a production machine. The network trace contains normal traffic as well as the infection traffic of both the host of interest and approximately 10 additional hosts. All additional local hosts in the trace have the same local IP address as the infected host because the machines were located behind a NAT. Figure 2 shows a trace of the number of unique IP addresses contacted over time. The trace starts at time zero, which is a short period before the point of infection.

The slope of the curve in Figure 2 changes rapidly at 800s, which indicates the time of infection. At this point, there is a significant increase in the number of unique IPv4 addresses contacted over time. At some short period of time preceding the infection, the user has opened the Trojan horse “FullVideo.exe” from an attachment in their email. The first operation of the executable is to join the peer-to-peer network in order to retrieve secondary injections. Therefore, the spike in traffic at 800s represents the initial peer-to-peer traffic as the host begins contacting peers.

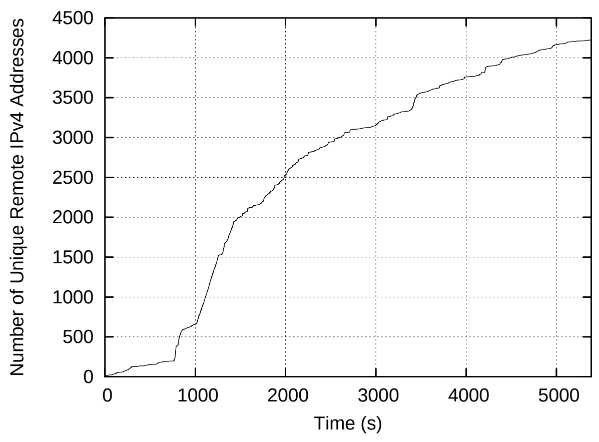

Figure 2: Number of Remote IPv4 Addresses Contacted Over Time for Duration of Infection

In the figure, the slope of the curve decreases around 2000s and continues to decline. The reason for decreasing slope is that the peer-to-peer botnet is saturating its list of known peers. As time progresses, the bot begins to maintain a more steady list of peers. Many of the peers in the original spike never respond because they are either no longer part of the Overnet network or they are unreachable. If the trace of infection had a longer duration, we would expect to continue to see new unique IPs as the nature of peer-to-peer networks is fairly dynamic.

In order to provide a deeper understanding of the trace, we wrote a tool to parse the Overnet packets in the network trace. Using the tool to analyze the trace, we found that the bot searches for five unique keys during its activity. Table 4 lists the five hashes that the bot searches for denoted h_1 through h_5. The h_1 hash is special because this is the node’s own ID hash. Part of the Overnet algorithm specifies that nodes should periodically search for their own ID in order to make sure they know the closest nodes to themselves. As for the other keys, two of them are never found and two of them are found. Of the two that are found, there are five total responses and four unique responders. All of the responses are equivalent and direct the bot to a secondary injection URL.

One of the most interesting observations from the data is that for h_3, it only takes 6 seconds for the value to be found after the first related search packet is sent. Similarly, the h_5 hash is also quickly located. It only takes 3 seconds to find. These observations indicate that the command latency metric for peer-to-peer bots can be

Hash	Found	Search	Search Reply	Get Search Results	No Result	Result
h_1 (self ID)	N/A	N/A	N/A	N/A	N/A	N/A
h_2	no	5	2	2	2	0
h_3	yes	39	13	13	11	2
h_4	no	30	7	6	7	0
h_5	yes	39	13	13	7	3

Table 4: Hash Search Results

quite low although perhaps not as low as centralized command and control.

In our analysis, the Overnet packets included 10,105 unique IPs in the Overnet network. Not all of these hosts are directly contacted since our trace only shows packets sent or received from approximately 4200 unique hosts. The number of unique Overnet IPs includes all peers described in fields of the Overnet protocol packets. It is not certain what percentage of these peers are part of the botnet. In fact, it is difficult to get information about many other peers in the botnet from just the network trace data. We know that of the five Result values returned, there were four unique hosts. We do not know if those hosts are infected with the bot. By the end of the trace, there is a new machine that sends our bot a search request looking for the same hash value that we previously requested. We think it is safe to conclude that host is infected with the Trojan.Peacomm bot. Thus, since we can only confirm one additional bot with reasonable certainty, we conclude that it is difficult to detect other infected hosts. We plan to develop detection methods in future work.

5 Related Work

Rajab et al. presented a measurement methodology that can be used to study botnets [10]. One of primary results of this study shows IRC as the prevalent C&C. We believe that peer-to-peer will likely be prevalent in the future, so our work is focused on understanding of peer-to-peer C&C.

Vogt et al. describe a botnet architecture called the *super-botnet* [11]. The basic idea of their architecture is that rather than having one large botnet, the botnet consists of many smaller botnets for some size parameter. The smaller botnets route commands to each other and can collectively achieve the same results as a larger botnet but with more resiliency. The communication architecture they describe could be classified as a hybrid peer-to-peer and centralized command and control architecture.

Dagon, et al. offered an analytical model for diurnal botnet propagation and population growth rate in the Internet [12]. This work is especially relevant to our study given the regional bias and usage trends of peer sharing application are skewed towards regions with higher computer penetration and better network bandwidths. We suspect that the diurnal botnet propagation and growth rate may impact peer-to-peer botnet growth.

In a seminal paper, Cooke et al. pointed out the potential threat posed by bots using peer-to-peer protocols for their C&C [13]. This work identifies some of the foundational analysis techniques for handling botnets including incapacitation of the botnet itself, monitoring the C&C channels, and tracking the propagation and attack mechanisms. This work highlights the underlying difficulties in monitoring the channel that may lead back to the bot controller [13]. Monitoring centralized C&C topologies is easier relatively but still difficult. In our work, the challenges in detecting the bot controller in a peer-to-peer network is more difficult due to the dynamic and distributed design of the architecture.

John Canavan describes attacks that use user deception techniques such as spreading bots by placing them in shared directories to be replicated and copied across the peer network [14]. This method describes the use of peer-to-peer applications for bot propagation. Our work focuses on the communication after infection, which can also be established via peer-to-peer networks.

Distributed denial-of-service (DDoS) attacks are a well known research problem. Much of the research concludes that simple checks such as IP header, packet content, or packet arrival rates can distinguish between legitimate and malicious traffic [15]. However, attackers continue to defeat these defenses. Our current work has been in studying peer-to-peer botnets, which enable DDoS attacks. Our goal is to develop methods of detecting, preventing, or mitigating peer-to-peer botnets. Derived techniques that accomplish these goals can likely be coupled with techniques for DDoS attack detection.

Constantinou et al. presented a novel approach for peer-to-peer traffic identification that relies on the fundamental characteristics of peer-to-peer protocols as opposed to application-specific details. These characteristics include large network diameters and large numbers of entities acting as both as clients and servers [16]. Future work will examine application of their techniques for the detection of peer-to-peer botnets.

A recent work by Barford et al. presented the source code analysis for effective understanding of mechanisms used by malware [2]. This work shows the increased sophistication in bots such as their modular design and encapsulated functionality. For example, the Agobot family has shown polymorphic obfuscations and a highly modular design. We believe that peer-to-peer botnets will likely become a more serious threat once a highly modular design becomes available.

6 Conclusions and Future Work

We have presented an overview of peer-to-peer botnets. Peer-to-peer botnets have the same basic goals of centralized C&C botnets, which include information dispersion, information harvesting, and information processing. Peer-to-peer botnets are distinctive from centralized C&C botnets in that there is no central point of failure for a peer-to-peer botnet; however, peer-to-peer botnets must communicate with many different peers. There has been a recent trend in increased development of peer-to-peer botnets, and we expect the level of sophistication to increase. Agobot is one of the most successful IRC-based botnets, which created a wealth of IRC botnets. We imagine that the peer-to-peer equivalent of Agobot may be released in the near future and will show a similar trend.

Our case study of the Trojan.Peacomm bot demonstrates one implementation of peer-to-peer functionality used by a botnet. The bot uses a peer-to-peer network to download secondary injection payloads. These secondary injections provide the basic primitive needed for command and control. Follow on work will include methods of detecting peer-to-peer botnets and simulation results to better study the resiliency of peer-to-peer botnets.

7 Acknowledgments

We would like to acknowledge the anonymous reviewers for their helpful comments. Additionally, we would like to thank Vernon Stark and Kevin Wenchel for their useful conversations. Finally, we would like to thank UNCC's Department of Software and Information System for their support through the Honeynet Lab and NSF award (DUE- 0415571) which partly funded this work at UNCC.

References

[1] D. M. Kienzle and M. C. Elder, "Recent worms: a survey and trends," in *WORM'03: Proceedings of the 2003 ACM workshop on Rapid Malcode*, pp. 1–10, ACM Press, 2003.

[2] P. Barford and V. Yegneswaran, "An inside look at botnets," in *Special Workshop on Malware Detection, Advances in Information Security*, 2006.

[3] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," in *ACM SIGCOMM 2001*, pp. 149–160, August 2001.

[4] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric," in *1st International Workshop on Peer-to-Peer Systems*, pp. 53–62, March 2002.

[5] "The honeynet project." `http://www.honeynet.org`, February 2007.

[6] "Perileyez." `http://www.digitalninjitsu.com/downloads.html`, February 2007.

[7] M. Suenaga and M. Ciubotariu, "Symantec: Trojan.peacomm." `http://www.symantec.com/security_response/writeup.jsp?docid=2007-011917-1403-99`, February 2007.

[8] J. Stewart, "Storm worm DDoS attack." `http://www.secureworks.com/research/threats/view.html?threat=storm-worm`, February 2007.

[9] J. Liang, N. Naoumov, and K. W. Ross, "The index poisoning attack in P2P file-sharing systems," in *Infocom 2006*, 2006.

[10] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in *Proceedings of ACM SIGCOMM/USENIX Internet Measurement Conference (IMC)*, pp. 41–52, 2006.

[11] R. Vogt, J. Aycock, and M. J. Jacobson, Jr., "Army of botnets," in *Proceedings of the 2007 Network and Distributed System Security Symposium (NDSS 2007)*, pp. 111–123, february 2007.

[12] D. Dagon, C. Zou, and W. Lee, "Modeling botnet propagation using time zones," in *Proc. of the 13th Annual Network and Distributed System Security Symposium (NDSS'06)*, 2006.

[13] E. Cooke, F. Jahanian, and D. McPherson, "The zombie roundup: Understanding, detecting, and disrupting botnets," in *Proceedings of USENIX WOrkshop on Steps to Reducinng Unwanted Traffic on the Internet*, pp. 39–44, USENIX, July 2005.

[14] J. Canavan, "The evolution of malicious IRC bots," in *Proceedings of Virus Bulletin Conference 2005*, pp. 104–114, October 2005.

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

[16] F. Constantinou and P. Mavrommatis, "Identifying known and unknown peer-to-peer traffic," in *Proc. of Fifth IEEE International Symposium on Network Computing and Applications*, pp. 93–102, 2006.