ContradictionC2

A TAKEDOWN-RESISTANT BOTNET BASED ON DEAD DROPS

Michael George, Tyler Rosonke, and Jonathan von Kampen

IASC-4580 SPRING 2014 IA CAPSTONE PROJECT

Table of Contents

Table of Figures

Executive Summary

Botnets are networks of malware-infected computers that are coordinated to accomplish typically malicious tasks. The compromised hosts run programs called *bots* to carry out the commands of *bot masters* who either develop the malware or purchase access to botnets (Criddle, n.d.). Botnets are managed with a variety *of command and control* (C2 or C&C) tactics, ranging from direct connections to individual bots to message-passing within a decentralized peer to peer network. Communication strategies have evolved to inhibit detection or compromise of any critical component of a botnet.

C2 centralization and the ability to reverse-engineer C2 communications have proven to be core botnet weaknesses. Even peer to peer networks are vulnerable to takedown by exploiting trust relationships between peers. Takedowns not only threaten the malware business but allow bot masters to be traced and apprehended. Bot masters have rapidly adopted new tactics and sophisticated mitigations to outrace malware researchers and defenders.

In the interest of proactive defense, we propose a novel botnet C2 architecture, "ContradictionC2," that is more resilient to detection and bot master unmasking. What's more, even if detected, our tactic is impractical to take down or render inaccessible. Our architecture uses *dead drop* file uploads to deliver commands to bots. In espionage, dead drops are messages concealed in public places for later retrieval by other parties (the opposite of *live drops* in which two parties meet to exchange information).

We suggest bot masters can uploading steganographically concealed messages to popular third party public websites for bots to retrieve. Our message format and choice of host sites minimize the possibility that uploads or downloads will be detected amidst legitimate traffic. Additionally, if some messages are detected, it will be impractical to detect and block all future messages, impossible to take down the host sites, and unlikely that the popular hosts will be blacklisted by end users. The location of control instructions is encrypted and maintained in a forward chaining structure which prevents disruption by an active attacker.

We created a small botnet to demonstrate that our hypothesis is practical and reveal potential limitations. The bot client is simple because we focused on securing the C2 (network communication) layer. However, we invested time in creating a web-based management interface to simplify the creation and distribution of instruction payloads. It is worth noting that the botnet does not depend on the management interface at all; it is solely a convenience mechanism.

Our botnet in theory and practice leads to a few defensive measures, albeit expensive ones, for both unwitting drop sites ("surrogates") and end users. Steganography cannot be reliably detected, so one defense would be to re-encode or subtly alter every user-uploaded media file to disrupt any steganographic payloads. This is a resource intensive process with little apparent gain for surrogates' administrators. For end user defense, malware researchers can always potentially reverse engineer the malware to obtain payloads and next locations. Next locations could be proactively, perhaps temporarily blacklisted. This requires an efficient payload recovery technique that can be automatically applied to each payload. Furthermore, researchers will never be able to get more than one step ahead of the latest payload because it only points to the next single instruction. Although the implications of our project seem dire, it is our hope that this project will spark further research and more efficient mitigations.

1. Introduction

Botnets are networks of malware-infected computers that are coordinated to send spam, perform denial of service attacks, and accomplish other typically malicious tasks. The compromised hosts, sometimes called *zombies*, run programs called *bots* to carry out the commands of *bot masters* who either develop the malware or purchase access to botnets (Criddle, n.d.). Botnets are managed with a variety *of command and control* (C2 or C&C) tactics, ranging from direct connections to individual bots to message-passing within a decentralized peer to peer network. Communication strategies have evolved to inhibit detection or compromise of any critical component of a botnet.

Nevertheless, botnets are often taken down by attacking the C2 infrastructure, which may lead to unmasking and prosecuting the bot masters. For example, in 2010, Dutch police cooperated with web host Leaseweb to take over the Bredolab botnet's C2 servers and trace their owner, leading to his arrest (Espiner, 2010). C2 centralization and the ability to reverse-engineer C2 communications have proven to be core botnet weaknesses. Even peer to peer networks are vulnerable to takedown by exploiting trust relationships between peers.

Bot masters have rapidly adopted new tactics and sophisticated mitigations to outrace malware researchers and defenders. It is becoming prudent to predict the next state of the art as part of malware defense. The history of computer security has shown that defenders have no choice but to forge double-edged swords; it is the only way to design effective shields because malware developers do not let ethics restrict them from pursuing dangerous new ideas.

In the interest of proactive defense, we propose a novel botnet C2 architecture that is more resilient to detection and bot master unmasking. What's more, even if detected, our tactic is impractical to take down or render inaccessible. Our architecture uses *dead drop* file uploads to deliver commands to bots. In espionage, dead drops are messages concealed in public places for later retrieval by other parties (the opposite of *live drops* in which two parties meet to exchange information). We suggest uploading steganographically concealed messages to popular third party public websites for bots to retrieve. Our message format and choice of host sites minimize the possibility that uploads or downloads will be detected amidst legitimate traffic. Additionally, if some messages are detected, it will be impractical to detect and block all future messages, impossible to take down the host sites, and unlikely that the hosts will be blacklisted by end users.

We created a small botnet to demonstrate that our hypothesis is practical and reveal potential limitations. This paper concludes by suggesting defensive measures, albeit expensive ones, for both unwitting drop sites and end users. It is our hope that this project will spark further research and more efficient mitigations.

2. Evolution of Command and Control Tactics

The first major malware to spread over the Internet was the Morris worm of 1988 (Reynolds, 1989), which was autonomous and relatively harmless. Since then, malware has become a revenue-generating underground industry. Infected computers harvest their users' credit card data for resale (Krebs, 2012) and mine Bitcoins (Rossow, et al., 2013), among other tasks. Rather than develop a new virus, worm, or Trojan horse for each task, malware developers often create general purpose programs called bots that can accept a variety of instructions. The distribution of instructions to the bots and the collection of any responses are collectively termed command and control (C2). A botnet is a collection of bots under a single C2 infrastructure, controlled by a bot master, which may be a person or a group.

An effective C2 infrastructure may enable one bot master to control millions of hosts. Accordingly, major foci of malware research and defense include understanding C2 tactics, dismantling C2 components, and leveraging the components to locate the bot masters. As with the methods of malware infection and obfuscation, C2 has become an arms race between malware developers and defenders. Although a diversity of tactics remain in use, over the past couple of decades, the state of the art has shifted from direct control of bots, to the use of central servers, to peer to peer message-passing. Each tactic has been enhanced (or perhaps patched back together) with a variety of resiliency and anonymity techniques. This section describes tactics and techniques, their advantages, real-world examples, and the environmental pressures (i.e., continuing malware research and defense) that have forced them to evolve or fall by the wayside.

2.1. Direct Control

The simplest command and control tactic is for the bot master to connect directly to each bot simultaneously or in turn. This is often called an "agent/handler" model (Specht & Lee, 2004), where the master is the handler and the bot is the agent. The master may make connections from his/her own computer or a compromised remote computer. Direct control allows masters to transmit instructions quickly and does not require masters to maintain a C2 infrastructure. A distributed denial of service program called Stacheldraht[1] was discovered to use this tactic in 1999 (Dittrich, The "stacheldraht" distributed denial of service attack tool, 1999). However, we were unable to discover any botnets that use this tactic currently or have within the past several years. Figure 1 diagrams an example directly controlled botnet.

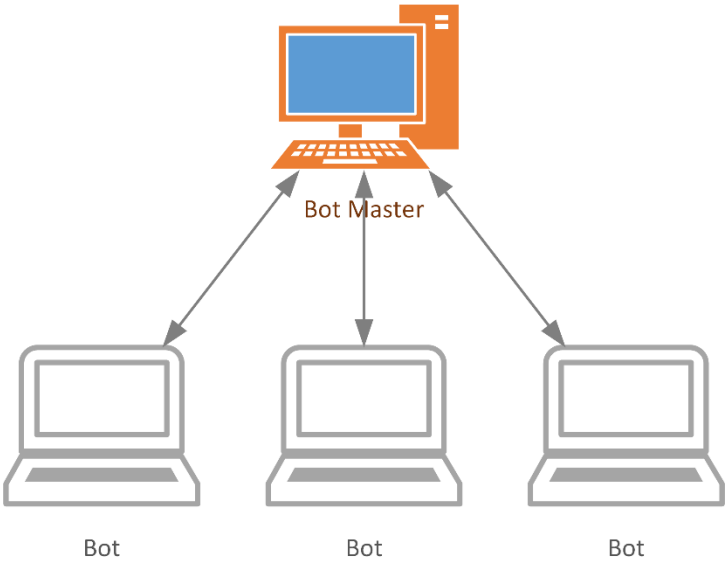

Figure 1. Topology of a directly controlled botnet

The disadvantages of such an architecture are almost as obvious as the advantages. The master's computer is a single point of failure and if bots are to

[1] In German, "barbed wire."

call back automatically it must maintain network connectivity -- with adequate bandwidth and open network sockets -- at a constant IP or domain name. If a bot is discovered by a defender, bot configuration files, network logs, or open connections will reveal the master's address. The master's weak defensive options include obfuscating and encrypting traffic and hiding behind proxy servers. Nonetheless, Dittrich and Dietrich claim that as of 2000, direct C2 structures could be dismantled "in several hours, in the best case" (2007). If unmasked, a master's computer would probably offer juicy forensic evidence as to the scale of the botnet's activities.

2.2. Central Servers

The most common botnet command and control tactic today relies on one or many central servers to receive commands from bot masters, distribute them to bots, and collect responses. The master-bot relationship is more abstract than under direct control but it is thus easier to scale communications as botnets grow and to add obscuring layers between masters and bots. Bot masters may either use their own servers or hijack others' servers. Bots may constantly connect to the servers or check in periodically to receive commands and exfiltrate data from their hosts. Although custom application protocols are possible, commands and data are often exchanged over legitimate protocols like Internet Relay Chat (IRC) or Hypertext Transfer Protocol (HTTP). Figure 2 diagrams an example central server architecture.

Among the first malicious bots were 1999's PrettyPark (Krogoth (pseudonym), 2008) and Sub7 (Ferguson, 2010), which each used IRC servers. IRC is well suited to "interactive" bot control. Users (masters or bots) connect to a central server which facilitates both group channels and one-to-one conversations (Oikarinen & Reed, 1993). Multiple servers may be networked together to increase capacity or redundancy. Direct client-to-client (DCC) connections allow the exfiltration of large files (Rollo, n.d.) although the master must then take care to anonymize his/her side of the connection. One disadvantage particular to IRC is that it has fallen out of common use for Internet conversations, so IRC traffic may be blocked wholesale by firewalls or flagged as an anomaly by intrusion detection systems (IDS). Persistent bot connections pose another detection risk. These factors may explain why HTTP C2 is now several times more popular than IRC (Leyden, 2010).

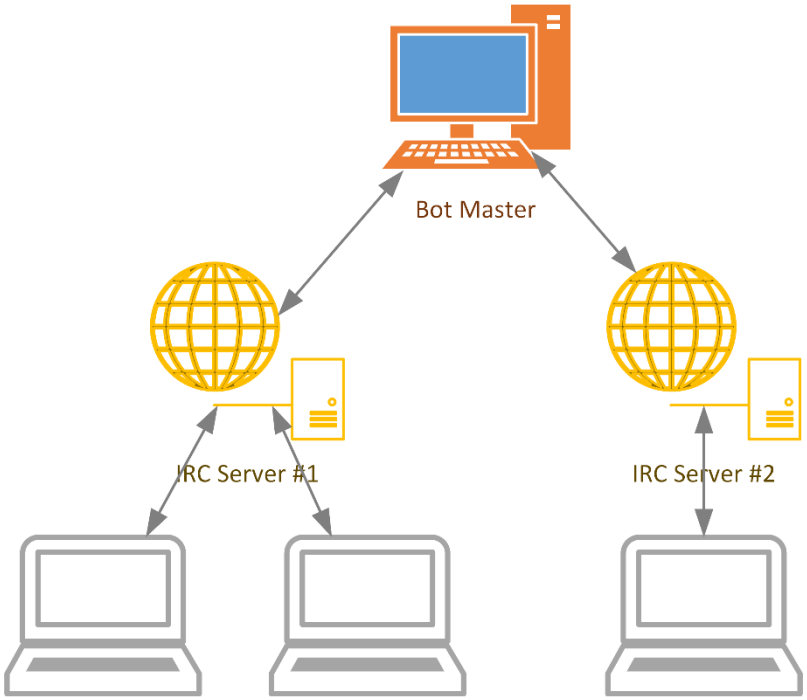

Figure 2. Topology of a central server botnet

HTTP is currently the most popular method of centralized C2. Communication in either direction can be made to look like inconspicuous web traffic. In fact, setting up an HTTP C2 server can be as easy as hosting any other website, as demonstrated by the Zeus breed of botnets. Zeus is a professionally marketed botnet package that bot

masters can purchase and customize (Macdonald, n.d.). Bot masters administer their Zeus botnets through a web control panel, sending commands and receiving bot data as HTTP POST messages. Zeus' ease of use is probably one reason the whole Zeus family made up an estimated 58% of botnet infections in 2013 (Nichols, 2013). Zeus also has variants with a peer to peer C2 architecture (Stone-Gross, 2012). This type of architecture is discussed in the next section.

Like direct control, a central server tactic imposes a small number of choke points that can be attacked to dismantle a botnet. Bots' paths to the C2 servers must remain relatively static. If discovered, the network paths can simply be rendered untraversable. This practice is known as *black holing* because C2 traffic associated with certain Internet Protocol (IP) addresses or domain names is silently dropped as though it has been sucked into a black hole. Taking down the servers themselves through legal or administrative action is obviously also possible in these cases. McColo was a US Internet host that was implicated in hosting and protecting several prominent botnets (Stewart, 2008). Global spam traffic briefly fell by as much as 75 percent when McColo was cut off by its upstream providers Global Crossing and Hurricane Electric (Krebs, 2008). A Zeus-based botnet, Bredolab, not only had many of its servers taken down by law enforcement; its alleged bot master was arrested thanks to cooperation between Dutch and Armenian authorities (Espiner, 2010).

Central C2 bot masters have come to rely on elaborate techniques of layered communications networks, constantly shifting targets, and misdirection to remain anonymous and avoid takedown. Masters and bots alike can use layers of proxies to access C2 servers. IP flux and domain flux refer to several techniques to vary the servers' IP addresses and domain names (Ollmann, 2009). One example of domain flux is a domain generation algorithm scheme where bots attempt to contact a large number of pseudorandomly-generated domain names, only some of which have been registered for a brief time by the bot master (abuse.ch, 2011). This practice is akin to taking a haystack and randomly turning its straws into needles and back again. Extreme examples of domain generation include the Conficker worm. A working group of several major Internet companies attempted to hobble the botnet by blocking as many as 50,000 generated domains a day (Rendon Group, 2011).

2.3. Peer to Peer

Botnet developers are beginning to eschew central control and hence single points of failure by using peer to peer (P2P) networks for C2. A P2P network is fundamentally non-hierarchical and is sometimes termed a *random* network (Krogoth (pseudonym), 2008). Each bot is made aware of several peer bots and is responsible for gradual message propagation by retransmitting received messages to its other peers. A bot master injects commands by connecting to an individual bot or simply posing as a bot. P2P botnets were proposed as early as 2000 (Dittrich & Dietrich, Command and control structures in malware, 2007). Figure 3 diagrams an example P2P architecture.

Dittrich and Dietrich contend that P2P botnets first achieved major success in 2006. Prominent P2P botnets around that time included Nugache and Storm, which were each used for distributed denial of service attacks and theft of user credentials. However, the success of P2P botnets has been difficult to estimate because researchers have to "crawl" the networks to determine their sizes. Bot programs may lack unique identifiers; hosts may have dynamic IP addresses or reside within networks with only a few public IPs; or bots may communicate very infrequently, slowing both legitimate message propagation and researchers' queries (Rossow, et al., 2013). For example,

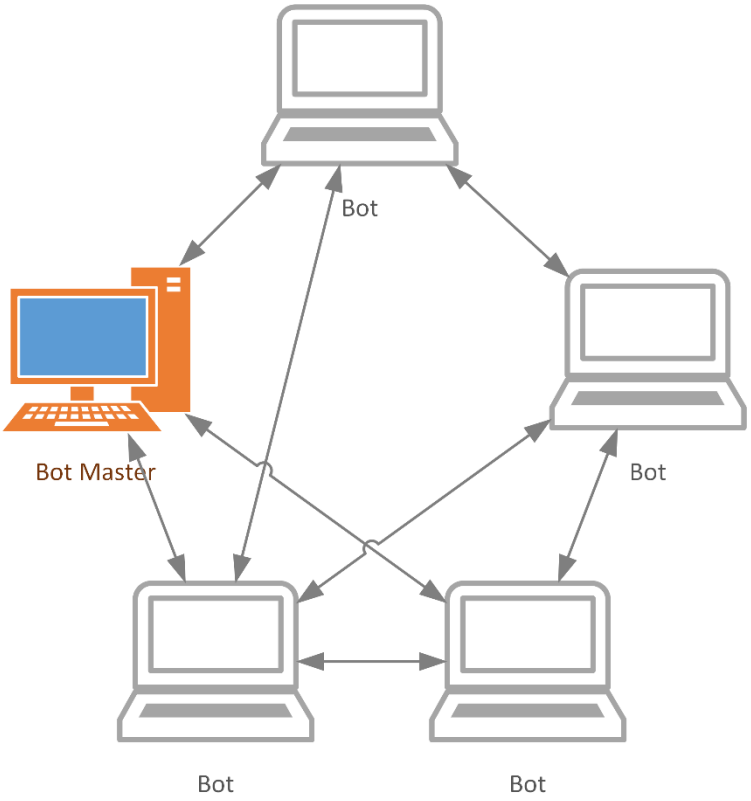

Figure 3. Topology of a peer to peer botnet

estimates of the peak size of Storm ranged from 1.5 million to 50 million bots (McMillan, 2007). This hints at the obscuration that makes P2P botnets difficult to take down.

P2P botnets are generally attacked by posing as peers. False peers can distribute instructions to do nothing or to shut down. They can also distribute lists of peers that are in fact unreachable or controlled by the attackers, crowding out legitimate message pathways. These capabilities depend on individual botnet implementations (Rossow, et al., 2013) and are not guaranteed to be successful. For example, in 2011 Kaspersky and Microsoft collaborated to redirect the traffic of 50,000 Kelihos.A peers to Kaspersky servers (Werner, 2011). A few months later, Kaspersky and additional partners redirected at least 116,000 peers within the new Kelihos.B variant (Ortloff, 2012). Kelihos's bot masters immediately launched a new variant, which was attacked dramatically at an RSA Conference 2013 presentation (Mimoso, Latest Kelihos botnet shut down live at RSA Conference 2013, 2013), but months later the botnet was still active and evolving (Mimoso, Kelihos relying on CBL blacklists to evaluate new bots, 2013). In this way, the business of malware is often a race between bot masters and researchers.

Bot masters often implement public key cryptography to ensure only authentic, signed commands are executed by bots. To also limit the injection of false peer lists, botnets may use reputation systems. For example, bots in the Sality network prioritize peers that are known to always carry out orders. Researchers may face the awkward legal and ethical paradox of having to contribute to botnet activities before attempting to dismantle a network. Finally, several P2P botnets maintain central C2 servers as a last resort, even employing flux techniques like domain generation algorithms (Rossow, et al., 2013). Nonetheless, P2P botnets are not invulnerable. Reputation systems recentralize power in individual high-reputation peers which may be feasibly traced and taken down. Resilient peer networks can attract unwanted attention at the network level because each peer must communicate with a large number of IP addresses or domain names (Mannon, 2013). Obviously, reintroducing central C2 tactics also increases the risk of tracing a botnet back to its masters.

2.4. Surrogate C2 Infrastructure

The previous three sections describe the only C2 "topologies" the security community is aware of. Bot masters continue to experiment within these topologies to enhance redundancy, better authenticate commands, and improve scalability. One trend worth mentioning is the number of

botnets that ride on top of websites and services rather than hijacking their servers outright. This is possible within both central server and P2P tactics.

For example, the "upd4t3" botnet used encoded Twitter tweets to distribute links to its actual commands (Nazario, 2009). Twitter is an appealing enough C2 medium that builder applications exist to construct Twitter-based botnets at the click of a button (Zeltser, 2011). Symantec also discovered a prototype botnet using Google Groups newsgroups for both C2 and data exfiltration from hosts (Gorman, 2009). These botnets take advantage of Twitter and Google's high-availability infrastructure and stellar reputations; social networks may be blacklisted by work and school networks but are unlikely to be blocked by consumers or consumer-oriented security software.

P2P botnets can likewise be implemented on top of existing reputable services. Our research uncovered no real-world implementations. However, researchers have demonstrated that a P2P botnet could utilize ordinary user communications on social networking sites like Facebook (Nagaraja, et al., 2011). All of these developments provided inspiration for the C2 structure we introduce in the next section.

3. Dead Drop Command and Control

Our analysis of the evolution of command and control tactics led us to conclude that any successful botnet architecture must overcome a few key challenges. First, the botnet should avoid detection as long as possible. Second, the botnet should resist takedown and black holing. Attendant with these, links between the botnet and the bot master should be obscured as much as possible to avoid arrest. To satisfy the C2-layer aspects of these challenges, we apply an espionage technique called the *dead drop* to the distribution of instructions.

3.1. Dead Drops in Spycraft

Dead drops have a long and storied history in espionage as a secure message-passing technique. Consider two spies who would like to transfer a message. Spy #1 walks into a public park and leaves the message in a hole in a tree, then exits the park. Spy #2 enters the park at a later time, locates the tree, and retrieves the message. (The location of the message and the execution of the drop are signaled by another channel.) The technique is called a dead drop because the two spies need never meet "live." If one spy is captured, he or she may be truthfully unable to identify the other.

Adversaries face two key problems, first of which is detecting the drop. The spies avoid suspicion by avoiding an in-person meeting and choosing an inconspicuous and often public drop location. The second problem is preventing retrieval of the drop, especially if a drop is suspected but cannot be located. Assume that an intelligence agency is aware only that its rivals are using the park for dead drops. The agency could close down the entire park. It could alternately search every visitor on entry and exit. Unfortunately, these techniques are so disruptive (at least to the mayor's re-election campaign) as to be impractical in all but the most extreme cases.

Dead drops are not foolproof. The process of arranging the drop must be secured from adversaries. Spies are also vulnerable if detected making or retrieving the drop. John Walker, one of the most notorious Soviet spies, was definitively incriminated when one of his dead drops was located, laden with military secrets (Prados, 2010). Russia uncovered a British spy ring because the British agents repeatedly visited the same dead drop, a rock with an embedded digital storage device that frequently malfunctioned (Paton Walsh, 2006). It remains true that dead drops significantly reduce risk and impact compared to in-person exchanges.

3.2. Dead Drops for Botnets

Dead drops translate well into the realm of malware and botnets. In this context we can use the park metaphor using the command and control as spy #1, the bot as spy #2, and the dead drop location as a public facing website. The command and control software uploads a file to a public facing website then the bot will go to the website and retrieve the information. In the realm of technology many of the disadvantages of dead drops can be overcome with encryption, steganography and automation. Dead drops overcome some of the flaws in peer to peer by not sharing information about each other making sure each party is secure from each other including the command and control application.

3.3. Frustrating Network-based Detection

A key feature of using http connections for command and control is that the traffic doesn't initially look suspicious. This will make figuring out that there is a bot on your network more difficult. Other botnets have used this style including APT 1 as described in a report by Mandiant (Mandiant, 2013). APT 1 was found to be using embedded messages in HTML on predefined servers that would allow

them to covertly control their bots. The bots can even further camouflage themselves by making the traffic go to legitimate websites and by going at times when the traffic to those sites make the most sense such as during work hours possibly even near the beginning or end of the work day. The defense to a communication like this would include the whitelisting certain websites or blacklisting others that may contain malicious content. This type of defense may frustrate the end user.

3.4. Frustrating Payload Takedown and Black-holing

Take-down and black-holing resistance is possible due to use of popular public facing site that uploads. Assuming that the host was compromised and the malware was fully reversed a researcher still might not be able to stop the botnet from receiving commands. If you watch a datagram get processed and the next location of the dead drop is revealed you won't know which image contains the datagram until after it gets processed. If you decide to block the entire website you may affect the end users experience. Because of this the design is more resistant to black-holing techniques. Assuming that you figure out the website and which image on the webpage is the dead drop a researcher would have to make the web administrator aware and have the link taken down. This is possible but it might not be fast enough to keep bots from accessing it. Another aspect that may stop the takedown is that each datagram contains multiple locations for a dead drop so, blocking one website is not enough to stop the transmission of data.

3.5. Limitations

With any implementation there are some limitations. The limitations with a dead-drop based schema are that it is difficult to have many connect to one. This is because managing this type of system comes with a lot of overhead from organizing drop locations to and from. Another limitation is where you can hide information. Some types of information can only be hidden a finite number of methods. Along with the difficulty of hiding the message it can also be difficult to not get caught hiding the method. One vulnerability in the dead drop schema is that if your dead drop is discovered it could be possibly replaced and the person who recovers the message wouldn't be the wiser.

4. Proof of Concept Implementation

We implemented our own botnet to assess the practicality of a command and control architecture based on dead drops. Our C2 tactic is a hybrid of central distribution using surrogate servers and peer to peer-style authentication of bot master instructions. As in physical dead drops, instruction payloads are made to look mundane and are placed in inconspicuous public locations. To focus the project scope, implementation of bots was restricted to retrieving and executing simple instructions. However, we created a web-based interface to manage creating and uploading instructions. As section 4.4 will explain, C2 is not dependent on a persistent interface; it is a convenience mechanism and could be substituted or foregone entirely.

4.1. Assumptions

During the design and construction of our implementation we had to limit the scope of our project to just the command and control aspect. Several assumptions were made about the host machines of the C2 application and the bots host machine. We assume the host machine of the C2 application is secure from attacks and is not likely to be compromised. We also assume that the bots code would not be detected by host based applications such as antivirus. We are also not going to cover how the machine was originally exploited. The last assumption is that the bot master has a method of anonymizing file uploads to these websites perhaps through a third party proxy service like TOR or through previously exploited computers using a different method of communications.

4.2. Architecture

In this case the C2 architecture is a more abstract concept than usual because we host instruction payloads on surrogate servers that we do not directly control. Additionally, C2 is not tied to a static set of servers either for a single instruction or over time. The implementation of the payload thus becomes the cornerstone of our architecture. It is responsible for not only carrying instructions but maintaining the links from one instruction to the next that are necessary for a botnet to persist. To minimize detection, the payload is concealed analogous to a physical dead drop item. Because of the payload's importance to our scheme, we apply an authentication strategy inspired by modern peer-to-peer botnets.

4.2.1. Communications Network

In our implementation we wanted to make sure that we applied the concept of a dead drop as securely as possible. The messages are uploaded to websites that would be less likely to be blocked by company firewall settings. The message that is uploaded is in the form of a JSON object that we call a datagram (see section 4.2.2 below). The datagram contains all the essential information that the bot should know about executing the current set of instructions and the information it needs to know about finding the next set of instructions. In the datagram you may notice that there are multiple locations that the datagram can be uploaded to. This is because redundancy is a key factor in our ability to control the bots. If a bot is unable to retrieve the next set of instructions our ability to control it will be lost. This is why we need redundancy built-in in case we lose communication with the bot.

The bot utilizes a decentralized design and implementation. With this we do not need to host payloads on a single web server or set of web servers (although our current proof of concept infrastructure is hosted on a single server for ease of use). The design allows our C2 management interface to never have to talk directly to any of the bots in the network. This is a good thing and a

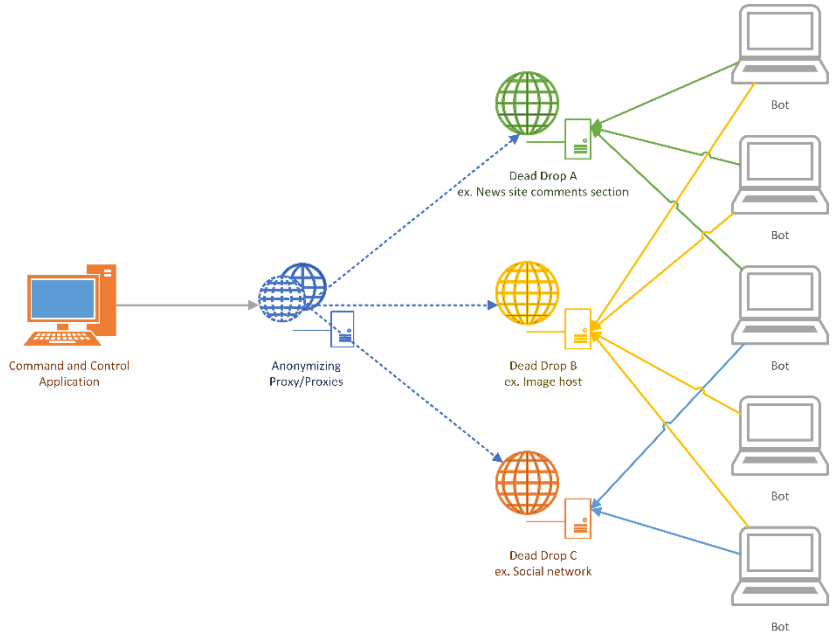

Figure 4. Communication from the bot master's point of view

bad thing as it is difficult to accurately count the number of infected hosts in the network but allows the system to not give away which machines are infected. Another key aspect of the decentralized nature is that we do not need to have a static machine anywhere as the dead drop can be uploaded from anywhere (see Figure 4). There is a concern that if the C2 server is the only place uploading to these sites it may be suspicious but with proper anonymizing channels the bot master can protect themselves from suspicion.

4.2.2. Payload Data Structure

The datagram is the core of the botnet structure (see Figure 5). It contains instructions, keys to the next datagram, time posted, and the locations of the next dead drops. Each part of the datagram contains an important aspect to maintaining the state of the botnet. The timestamp contains a UNIX epoch timestamp of when the dead drop was posted this information is used to find out if the datagram contains stale instructions. The instruction section contains maintenance and primary instructions. The maintenance instructions are used to update the code base and add features to the communication structure. The primary instructions are instructions that would be issued to the bot for malicious reasons, this is the area where a bot master would issue direct commands to the bot such as extract information and denial of service attacks. The decryption key is the private key for decrypting the next dead drop.

Datagram post timestamp		1398430800
Maintenance instructions		cHJpbnQgIkknbSBhI*[…]*
Primary instructions		cHJpbnQgIkhlbGxvI*[…]*
Decryption key of next datagram		MIGJAoGBANxn+vS*[…]*
List of next datagram locations:		
1.	**Location**	http://example.org/
	Time to start checking	1398430800
2., …	*[…]*	
Bot pool ID		1

Figure 5. Datagram contents

Inside the datagram there is a section that includes the next locations for the next dead drop (see Figure 6).

The last section of the datagram is the bot pool. The bot pool is an important part to the administration of a botnet as it allows us to separate commands to a more specific branch of computers. The bot pool allows the bot

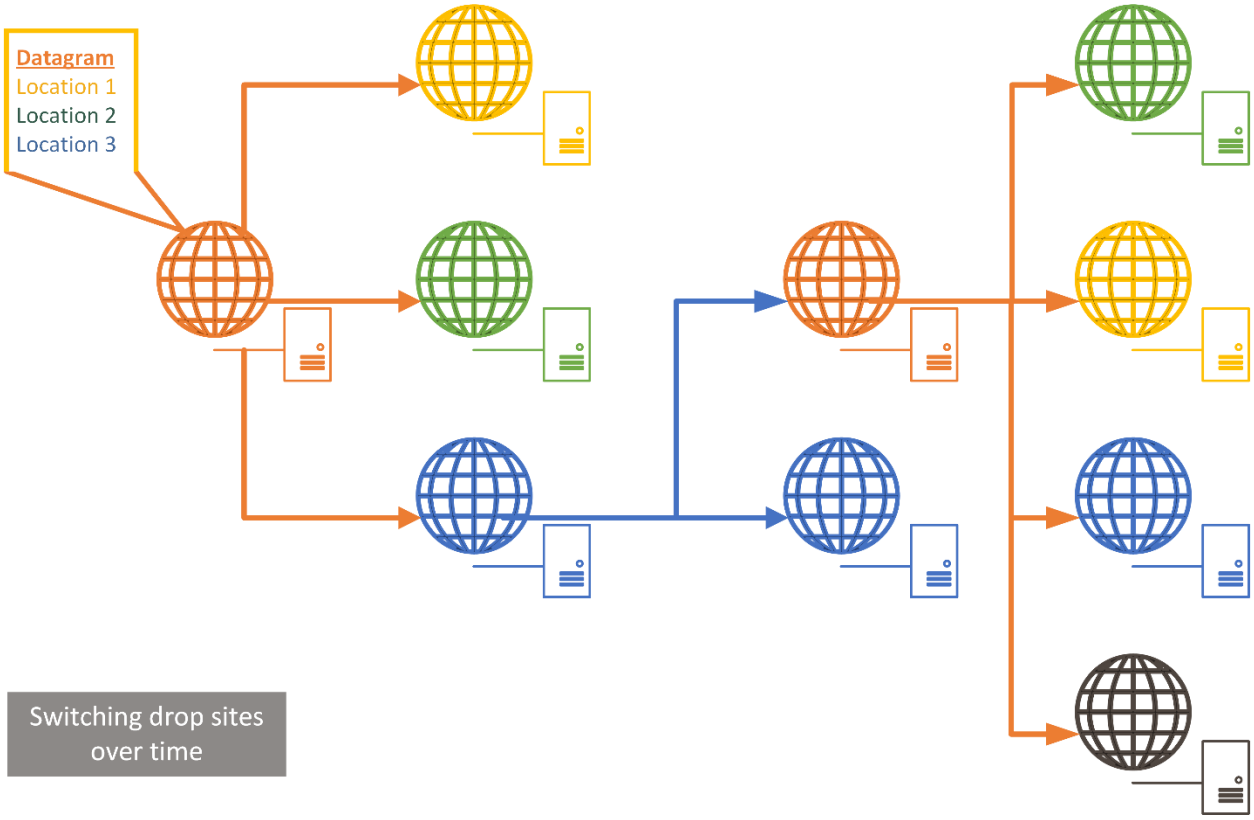

Figure 6. Specifying the next locations in each datagram allows a branched linked list of surrogate servers

master to separate part of the bot net and possibly sell it to another bot master. Each bot has a bot pool number assigned to it and that depicts what chain of commands it is running. Each dead drop has only one bot pool number attached to it and share no information with its pool to another pool.

4.2.2.1. Hiding the Payload

Simply dropping this datagram as text would result in alarm for both administrators and users alike for obvious reasons. There needs to be a way to hide the data, not only to stop being from reading it, but so that they don't even know it's there. This is where a technique called steganography comes into play. Steganography is the science of hiding information in plain sight (Kessler, 2001). The idea behind steganography is to hide a small amount of data within a large set of cover data. Data that makes good cover data is files there are very large, media files like pictures, music, and video typically make great cover data. This method works due to the concept of a least-significant-bit. The least-significant-bit is the bit inside of string of bits that has the least amount of precedence, or "affect", on the data overall. If every pixel of picture, for example, is defined by a string of bits and the least-significant-bit is replaced with a single bit of the data being hidden, the overall color and quality of the picture is left nearly unchanged due the principles of the least-significant-bit (Kessler, 2001).

Note the close similarities between the shades of gray on the right side of Figure 7. Even so, they have different digital representations (indicated by the blue labels). A slight change to any of the rightmost digits in each row would be undetectable by the human eye. Using this technique for every pixel of a picture would allow for a large amount of data to be stored with a picture. Using the steganography technique, a datagram could be dropped on a vulnerable website, later to be picked up decoded by a bot.

Figure 7. Steganography modifies data in a way that is undetectable by human senses

4.2.2.2. Protecting the Payload

Even if the datagram is embedded into an image, that does not guarantee its confidentiality or integrity. An adversary could potentially detect and decode the data embedded in an image. Not only would this allow the advisory to disclose all of the datagram data, but the advisory could modify the datagram, re-embed it in the image, and bot would pick it up and follow the datagram's instructions all the same. In order to guarantee confidentiality and integrity, this design uses both asymmetrical and symmetrical keys. A datagram's data is encrypted using a dynamic symmetric key, which changes for every datagram. The symmetrically encrypted data is stored in what we call the *data* group within the payload (see Figure 8). The symmetrical key is encrypted with an asymmetrical encryption key. The asymmetrically encrypted data is stored in what we call the *key group* within the payload. In order for a bot to decrypt a datagram, they must first decrypt the key group by using the asymmetrical decryption key given to them in the previous datagram. This will reveal that dynamic symmetrical key to decrypt the data group, thus allowing them to see the datagram in plain text.

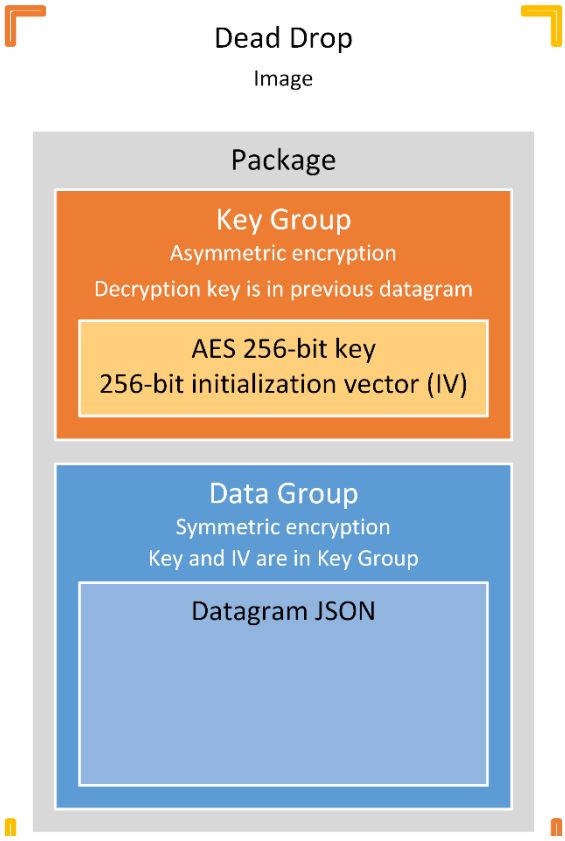

Figure 8. Payload layered data structure

This schema guarantees confidentiality due to the fact that no one will be able to read the datagram besides the bots who have an asymmetrical encryption key. This schema guarantees integrity because the bot master is the only individual to hold the asymmetrical encryption key. Even if an adversary was able to decrypt and modify a datagram, they would never be able to re-encrypt it with the same asymmetrical key held by the bot master, thus the bots would know the datagram is not legitimate.

4.3. Zombie Application

The client side application is what the bot uses to communicate with the bot master. The bot is written in python and follows a precise algorithm for processing datagrams. On the initial infection the

application has a predefined datagram inside that issues the first set of instructions. After processing the datagram by de-steganogrifying the package and separating out the key group and the data group the program will use the previous datagrams decryption key to decrypt the key group and extract a symmetrical key to decrypt the data group. Once decrypted the datagram is extracted and its instructions will be executed. There are two sets of instructions the maintenance instructions and the primary instructions. The maintenance instructions are always executed after the datagram is processed. After maintenance the primary instructions will be executed if they are found to not be stale. A set of primary instructions are thought to be stale when they are older than 3 days old. After execution of all the instructions the program will check to see if is time to go to the next location and retrieve the next dead drop. If it is not time to check it will wait until it is time then go retrieve the instructions.

When it is time to fetch the next dead drop the program will visit the URL and take all image tag URLs and download them from that page. This is useful since it doesn't point to a specific image and that means that the dead drop can be placed after the previous datagram has been made and posted. Each image is then processed to see if it contains a package. If no package is found then it will change to the next the location in the list of locations in the datagram. If no packages are found in any of the images in any of the locations then the ghost protocol will be initiated. The ghost protocol is where it tries to reinfect itself hoping that if it gets infected then the new malware will contain an updated datagram so that it can further progress through the chain of commands. If all options fail the program will attempt to destroy itself to remove evidence of its presence.

4.4. Portable Management Interface

We wrote an application we call the Portable Management Interface (PMI) to simplify the task of creating payloads and distributing them to surrogate servers (see Figure 9). This section is a walkthrough of the PMI's features.

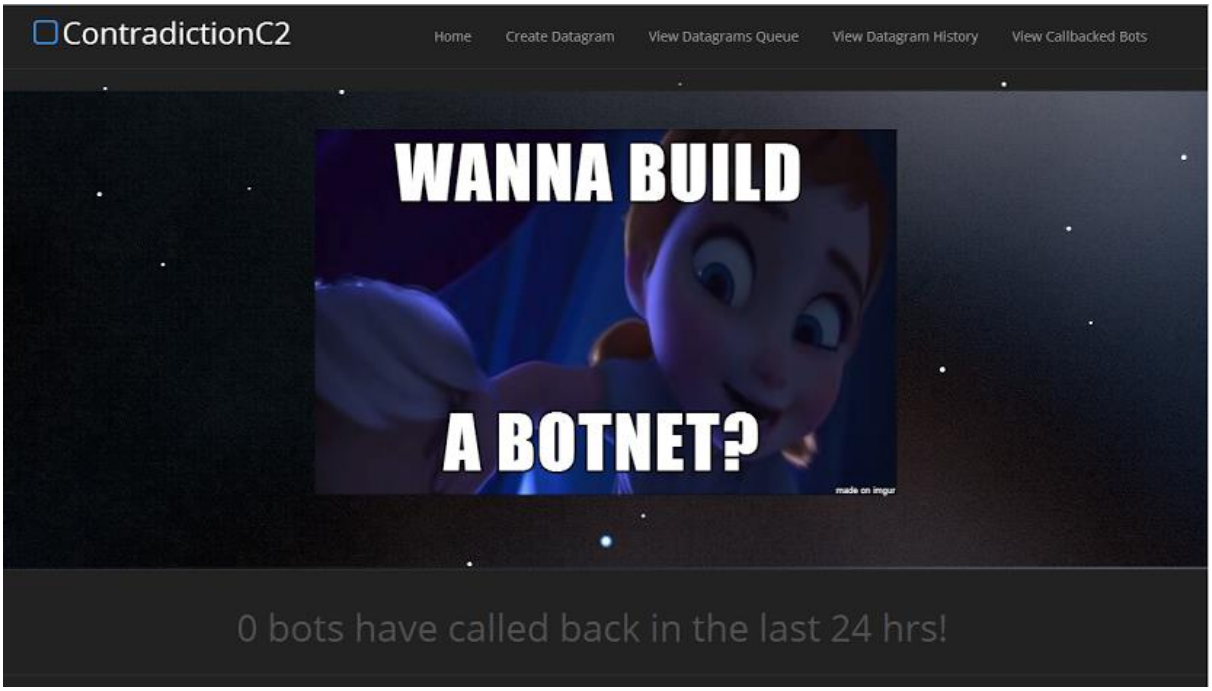

Figure 9. Portable Management Interface home page

4.4.1. Design and Purpose

Between managing all the encryption keys, keeping track of what datagrams are pointing to what, and trying to remember what datagrams did, managing an asynchronous decentralized botnet of this magnitude can become very confusing and tedious. This was why a portable management interface (PMI) was designed. The idea behind the PMI was to make an easy to use frontend where the user is only asked for data they must provide, while encryption keys and alike are handled behind the scenes automatically. Our implementation uses the LAMP stack (Linux, Apache, MySQL, and PHP) to provide the PMI services. The PMI can be packaged and ran from any machine, thus making it portable. The major functions of the PMI are to create, edit, view, post, and view past datagrams.

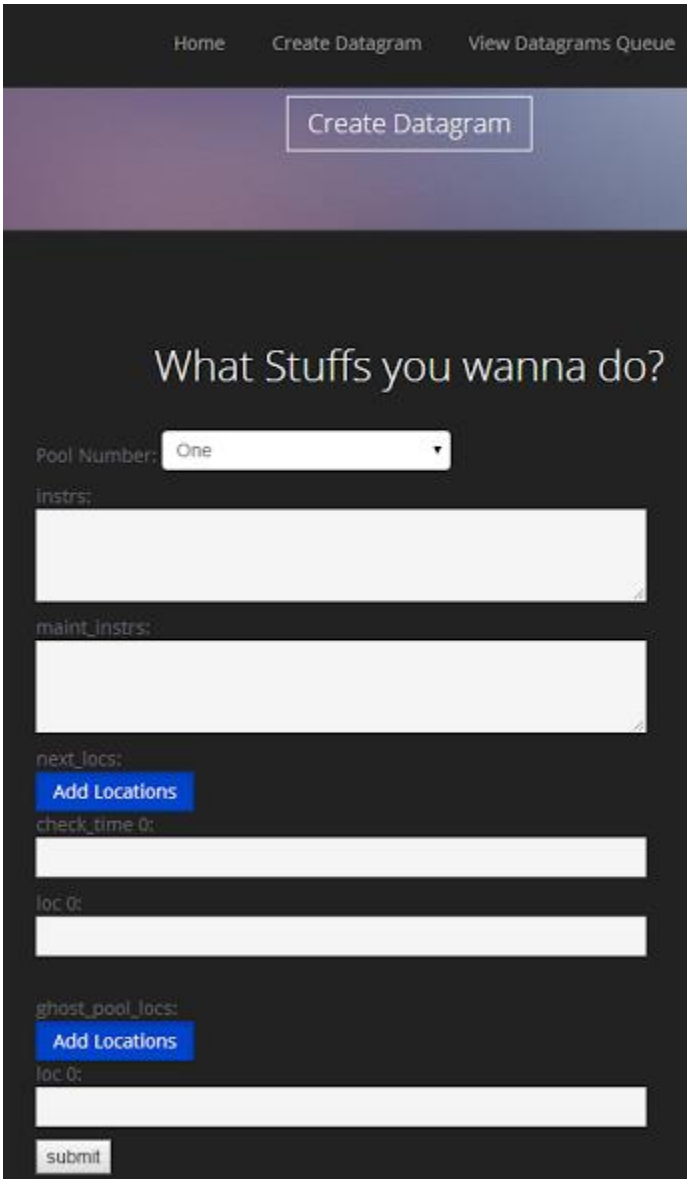

Figure 10. Datagram creation form

4.4.2. Create Datagrams

The first major function of the PMI is the ability to create datagrams (see Figure 10). The create datagram dialogue only asks for fields the user must provide, all other fields are handle automatically behind the scenes. The "next location" and "ghost pool location" fields are dynamic, meaning you can add as much redundancy as you wish on a per datagram basis. When the form is filled out and submitted, the data is added the datagram queue within the MySQL database. The reason that there is a queue is that this allows an attacker to create a multitude of instructions before actually posted any data. This allows for the attacker to build a series of instructions at once and review them for accuracy before the datagram is posted for the bots to retrieve.

4.4.3. View and Edit Datagrams

After a datagram is created, it can be viewed on the viewed on the queue on the View Datagram Queue Queue page (see Figure 11). This page lists all the all the datagrams currently queued and some brief some brief information about each datagram. If an datagram. If an attacker wants to view the full details full details of a datagram, they can click the specific specific datagram from the list. This will bring up a bring up a page that lists the datagram in full detail full detail (see

).

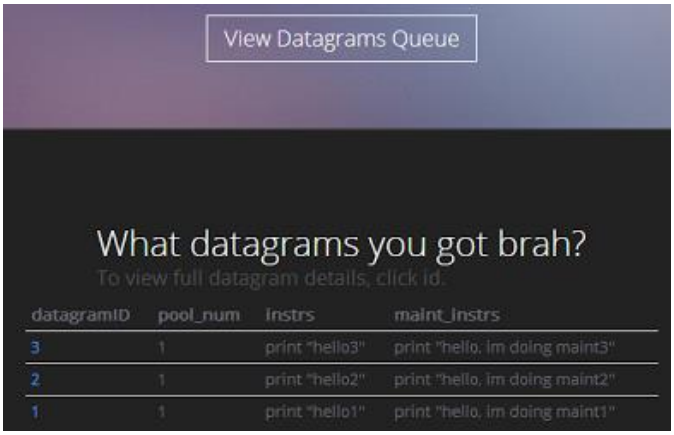

Figure 11. Datagram queue

From this page an attacker can delete, edit, and initiate and initiate the post functionality. The delete button will button will delete the datagram from the queue queue permanently, while the edit button will allow you will allow you to tweak any details of the datagram via datagram via a popup modal (see

).

Using these functions an attacker can build a series of instructions and check their validity and accuracy before posting any data to a vulnerable website.

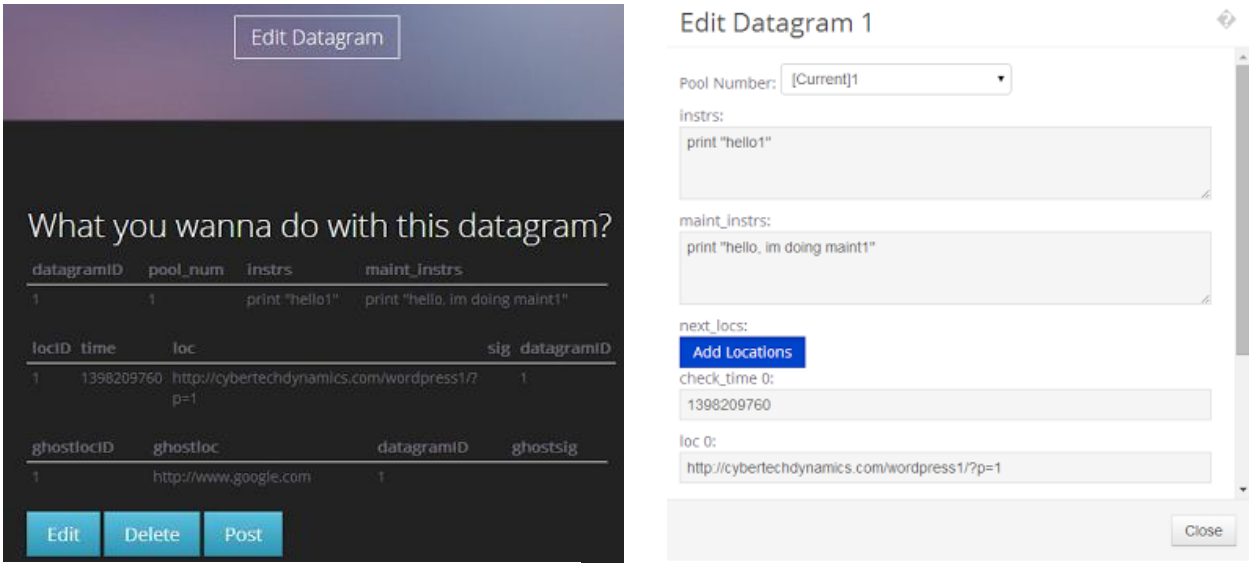

Figure 12. Datagram editing

4.4.4. Post Datagrams

Once a series of datagrams or a single datagram is ready to be posted, an attacker can click the post datagram button from the datagram details page. This button will bring up modal where the attacker can select what site the attacker would like to post to (see Figure 13).

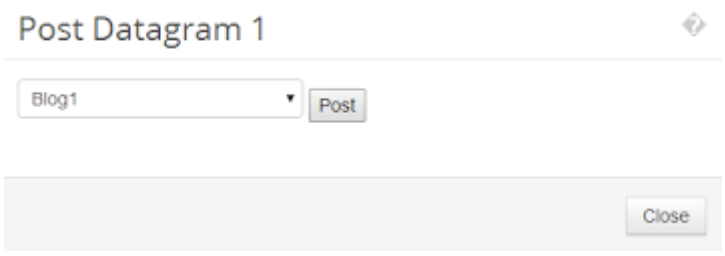

Figure 13. Datagram posting

Currently, the only website we have written posting procedures for a WordPress blog site. The implementation has been built modularly, so that more posting procedures can be written for a variety of vulnerable websites. Once a specific website has been selected to be posted to and the user clicks the post button, the PHP page will call a Python script with the datagram data and posting specifics as parameters. The

Python script will first generate encryption keys to be used in encrypting/decrypting of the next datagram. It will store the encrypting key in the MySQL database and add the decryption key to the current datagram. Next, the Python script will encrypt the datagram using the encryption key generated when the previous datagram was posted, which is stored in the MySQL database, so that the bots will be able to use the decryption key given in the previous datagram. The encrypted datagram is then inserted into an image using steganography. The last task the Python script performs is to upload the image with the datagram onto the vulnerable website specified by the PMI. See the example result in Figure 14.

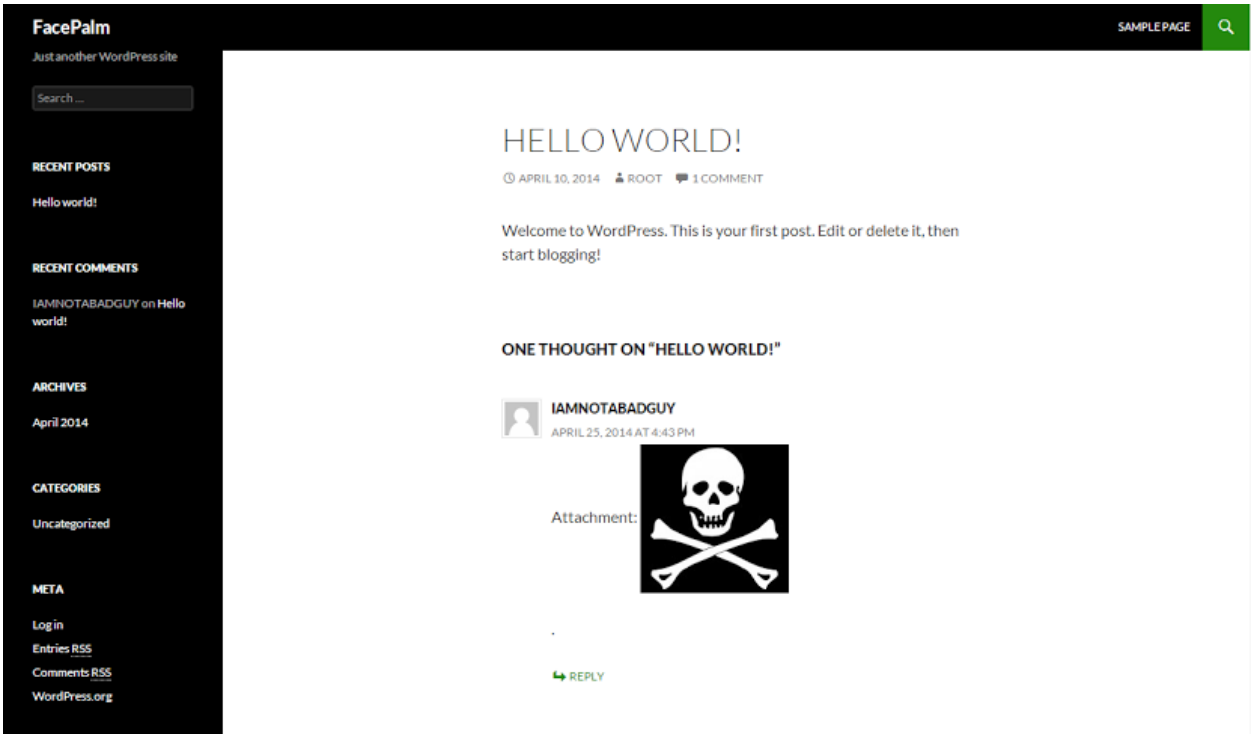

Figure 14. Payload uploaded to a WordPress blog

Figure 15. A previously posted datagram

4.4.5. View Past Datagrams

Once a datagram has been posted, that datagrams information will be removed from the queue and added to the Past Datagrams section of the MySQL database (see Figure 16). The datagrams that have already been posted can be viewed on the View Past Datagrams page. Similar to the View Datagram Queue page, the View Past Datagrams page displays a list of all the past datagrams and a brief description of each. If a specific datagram is

clicked on, a page with the datagrams full details is loaded (see Figure 15).

Notice how this detailed view does not have any buttons to edit or delete. This is because this page is designed to be an exact record of what datagrams have been posted. Any errors found on this page will reflect real errors, which could potentially result in attacker losing control of bots. The queue schema was created to help avoid these types of errors.

4.4.6. View Bots

To aid development and debugging, the management interface also has functionality to track bots who call home (see Figure 17). If an attacker chooses to, they can command bots to call home. This would allow the attacker to get an accurate count of all the bots within their botnet. The bots who have called home can be seen on the View Callbacked Bots page. Here an IP, pool number, and timestamp can be viewed for each bot that called home.

Figure 17. A list of bots that have called back to the management interface

4.4.7. Database Schema

Figure 18 is the entity relationship diagram of the database that was used to implement the PMI. Notice how there are separate tables for the queue and history. The history tables are used for the posted datagrams.

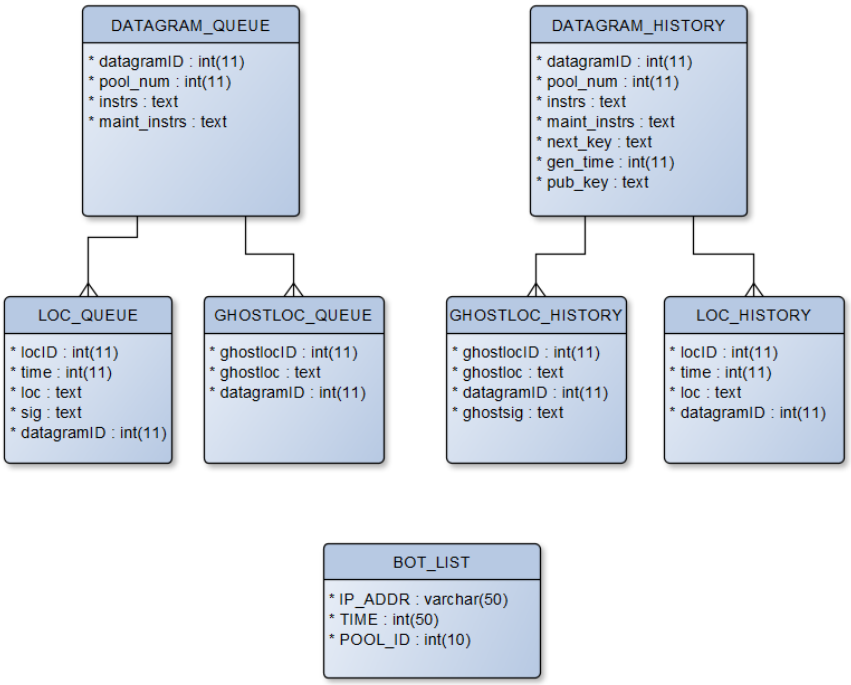

Figure 18. Management interface database entity relationship diagram

5. Lessons Learned

Throughout the process of designing and implementing the botnet, we had to overcome and achieve goals. Our first goal was that the botnet should be able to avoid detection for as long as possible. We achieve this goal by using dead drops. Dead drops have been time tested and the vulnerabilities for this design are well documented and can be mitigated. The second goal was to be resistant to black holing and takedowns. By placing our dead drops on legitimate popular websites we avoid administrators from black holing websites because it could possibly impact the legitimate users on their network. Takedown of the requested content if detected would require timely cooperation between researchers and administrators of the website which may be too late for the bots. Lastly the bot master has a clear neutral area that separates them from the bots which can keep them from being traced back to.

5.1. Practical Limitations

Some of the problems associated with this style of botnet are size, complexity, network traffic spikes, and maintenance of drop sites. As the size of the botnet grows the more complex the management of the botnet becomes with keys to each dead drop. With each bot added to the botnet we run into network congestion issues where administrators may notice the spike in network traffic to a particular page or resource. These issues can be resolved through pools where each pool follows a separate chain of commands. This method also increases the complexity but with automation this task can be significantly easier to maintain. Ghost pools or reinfection sites will also have to be maintained by the bot master as they are the last resort for keeping the bot moving forward.

5.2. Improvements to our Implementation

Improvements to our proof of concept could include the ability to upload to other drop sites besides WordPress. This style could allow us to utilize a more modular approach to upload to site function. Since this C2 management is decentralized we don't need a central static server to control anything it is theoretically possible to create a C2 mobile application for a phone that could execute and post instructions from the bot master. This could also feature a mobile malware application where the mobile device is also infected.

5.3. Defenses for Victim Drop Sites (Surrogate Servers)

The first type of defense is to prevent website from being vulnerable to hosting datagrams on. There are a few mitigation techniques in order to prevent this type of botnet from functioning on a website. The root cause for this botnets capable existence is the lack of user input sanitization. Input sanitization describes the cleansing of user input to prevent a system from interpreting it as an instructions and exploiting possible security holes (Weiss, 2012). The specific type of input sanitization the vulnerable websites are lacking is user media upload sanitization. They allow a user, sometimes even an unauthenticated user, to upload arbitrary media files and then have them visible to the internet without any sort of approval.

The implementation proposed above is heavily dependent on data being able to be hidden within an image using steganography. If a website was able to sanitize their user uploads, the steganography would be ruined and the datagram hidden would be lost. A website can sanitize an image by doing any form of re-encoding, such resolution change, cropping, or adding any filters to an image. If any of the listed actions was performed on an image, the hidden datagram would be lost, thus successfully sanitizing the image. The potential problem with re-encoding all the user uploaded

images is that this process would be expensive. It would require a large amount computer resources to perform such a task on every image uploaded. Due to this costly nature, website administrators would not likely spare the expensive to catch the off chance of this type of botnet exists on their site, thus making this defense mechanism impractical. One may consider using some sort of steganography detection application in order to decide what images to re-encode. This method would work, but would be costly within itself and inaccurate due to the wide variety of customization among steganography, thus making this defense mechanism impractical as well.

5.4. Defenses for Victim Networks and Hosts

The second type of defense is to prevent a computer from being part of the botnet. This type of defense is typical among any form of malware or viruses. If a law enforcement agent or security researcher was able to reverse the botnet client, they would be able to generate a signature that can be used by signature based detection systems, whether that be on a network intrusion detection system or on an endpoint, such as an antivirus or antimalware on a workstation. This sort of defense is inherent with all forms of malicious code.

Something more interesting a law enforcement agent or security researcher could do is attempt to build a list of bots currently operating on the botnet. This would require the adversary to reverse the bot client and being able to view the datagram in clear text, which is a challenging task. If an adversary could see the datagram in clear text, then they would be able to see the next drop location and when the bots would be browsing to it. Knowing this the adversary could attempt to monitor and log all computers connecting to that particular webpage at that particular moment. Not only would this give the adversary a list of all the bots operating on the botnet, but also all the legitimate users using that web page at that time. The adversary could refine this list by going to the next drop site and doing the same procedure, and then cross reference the connection lists. This would help eliminate false positives, which are the legitimate users using that web page. If the adversary used this method on a series of drops, there list of bots would become more refined with each drop page monitored. After accumulating the bot list, the adversary could then investigate the bots and potentially removing the computer from the botnet. This however, does not affect the bot command and control infrastructure.

Glossary

Anonymizer: Traffic obfuscation through proxies, redirects, etc. (e.g., TOR)

Bot: An infected machine that can be issued commands remotely

Bot Master: An entity capable of issues commands to botnet

C2 Server: Any computer hosting the c2 software

Datagram: The JSON object that is processed by the bot to execute instructions

Data Group: Contains the base64 output of the AES256 encryption of the datagram

Dead Drop: The Item transporting the datagram

Drop Point: Site Hosting Dead drop

Ghost Pool: A pool of bots that have lost sync with their group and are being redirected into a new pool using the ghost pools instructions

Instructions: A section inside the datagram that contains what the bot should execute

Key Group: Contains the base54 output of the Asymmetrical encryption of the AES key that is used to encrypt the Data Group of the Package

Maintenance Instructions: Located in the datagram these instructions are always executed even if the datagram is stale

Package: The Package is the grouping of the data group and the key group that is steganography hidden inside the image

PMI: The Portable Management Interface is the frontend, control center, and toolbox for a bot master.

Pool: A subset or grouping of the botnet that are following the same instructions.

Primary Instructions: Located in the datagram these instructions are only executed when the stale time has not be achieved

Signatures: A SHA512 hash of the datagram to verify the integrity of the datagram.

Stale: A datagram that has been placed but not executed by the bot before the stale date. Used to keep the bot from performing malicious actions as it catches up the newest dead drop.

Works Cited

abuse.ch. (2011, July 28). *How criminals defend their rogue networks*. Retrieved from abuse.ch: https://www.abuse.ch/?p=3387

Criddle, L. (n.d.). *What are bots, botnets, and zombies?* Retrieved from Webroot: http://www.webroot.com/us/en/home/resources/tips/pc-security/security-what-are-bots-botnets-and-zombies

Dittrich, D. (1999, December 31). *The "stacheldraht" distributed denial of service attack tool.* Retrieved from Dave Dittrich (personal homepage): http://staff.washington.edu/dittrich/misc/stacheldraht.analysis

Dittrich, D., & Dietrich, S. (2007, December). Command and control structures in malware. *;login:, 32*(6). Retrieved from https://www.usenix.org/legacy/publications/login/2007-12/openpdfs/dittrich.pdf

Espiner, T. (2010, October 26). *Dutch police take down Bredolab botnet*. Retrieved from ZDNet: http://www.zdnet.com/dutch-police-take-down-bredolab-botnet-3040090649/

Espiner, T. (2010, October 26). *Dutch police take down Bredolab botnet*. Retrieved from ZDnet: http://www.zdnet.com/dutch-police-take-down-bredolab-botnet-3040090649/

Ferguson, R. (2010, September 24). *History of the botnet, the - part 1*. Retrieved from Trend Micro CounterMeasures: http://countermeasures.trendmicro.eu/the-history-of-the-botnet-part-i/

Gorman, G. O. (2009, September 11). *Google Groups trojan*. Retrieved from Symantec Connect: http://www.symantec.com/connect/blogs/google-groups-trojan

Kessler, G. C. (2001, September). *Steganography: Hiding data within data*. Retrieved from Gary Kessler Associates: http://www.garykessler.net/library/steganography.html

Krebs, B. (2008, November 12). *Host of Internet spam groups is cut off*. Retrieved from Washington Post: http://www.washingtonpost.com/wp-dyn/content/article/2008/11/12/AR2008111200658.html

Krebs, B. (2012, June 12). *Feds arest 'Kurupt' carding kingpin?* Retrieved from Krebs on Security: http://krebsonsecurity.com/2012/06/feds-arrest-kurupt-carding-kingpin/

Krogoth (pseudonym). (2008). *Botnet construction, control and concealment: Looking into the current technology and analysing tendencies and future trends.* MSc Thesis. Retrieved from http://www.shadowserver.org/wiki/uploads/Information/thesis_botnet_krogoth_2008_final.pdf

Leyden, J. (2010, November 16). *IRC botnets dying off*. Retrieved from The Register: http://www.theregister.co.uk/2010/11/16/irc_botnets_dying_off/

Macdonald, D. (n.d.). *Zeus: God of DIY botnets*. (D. Manky, Editor) Retrieved from FortiGuard Center: http://www.fortiguard.com/legacy/analysis/zeusanalysis.html

Mandiant. (2013, February 18). *APT1: Exposing one of China's cyber espionage units.* Retrieved from Mandiant: http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf

Mannon, C. (2013, August 27). *Kelihos botnet: What victims can expect*. Retrieved from Zscaler: http://research.zscaler.com/2013/08/kelihos-botnet-what-victims-can-expect.html

McMillan, R. (2007, October 21). *Storm worm now just a squall*. Retrieved from PCWorld: https://www.pcworld.com/article/138721/article.html

Microsoft Corporation. (n.d.). *What is a botnet?* Retrieved from Microsoft Security Intelligence Report: http://www.microsoft.com/security/sir/story/default.aspx#!botnetsection_history

Mimoso, M. (2013, August 28). *Kelihos relying on CBL blacklists to evaluate new bots*. Retrieved from Threatpost: http://threatpost.com/kelihos-relying-on-cbl-blacklists-to-evalute-new-bots/102127

Mimoso, M. (2013, February 27). *Latest Kelihos botnet shut down live at RSA Conference 2013*. Retrieved from Threatpost: http://threatpost.com/latest-kelihos-botnet-shut-down-live-rsa-conference-2013-022613

Nagaraja, S., Houmansadr, A., Piyawongwisal, P., Singh, V., Agarwal, P., & Borisov, N. (2011). Stegobot: a cover social network botnet. *Proceedings of the 13th International Conference on Information Hiding* (pp. 299-313). Prague: Springer.

Nazario, J. (2009, August 13). *Twitter-based botnet command channel*. Retrieved from Arbor Networks DDoS & SEcurity Reports: http://www.arbornetworks.com/asert/2009/08/twitter-based-botnet-command-channel/

Nichols, S. (2013, March 29). *Zeus retains botnet crown, according to McAfee*. Retrieved from V3: http://www.v3.co.uk/v3-uk/news/2258398/zeus-still-king-of-the-botnets-say-researchers

Oikarinen, J., & Reed, D. (1993, May). *Internet Relay Chat Protocol (RFC1459).* Retrieved from Internet Engineering Task Force - Network Working Group: https://www.rfc-editor.org/rfc/rfc1459.txt

Ollmann, G. (2009). *Botnet communication topologies: Understanding the intricacies of botnet command-and-control.* Retrieved from Damballa: https://www.damballa.com/downloads/r_pubs/WP_Botnet_Communications_Primer.pdf

Ortloff, S. (2012, March 28). *Botnet shutdown success story - again: Disabling the new Hlux/Kelihos botnet*. Retrieved from Kaspersky SecureList: https://www.securelist.com/en/blog/208193431/Botnet_Shutdown_Success_Story_again_Disabling_the_new_Hlux_Kelihos_Botnet

Paton Walsh, N. (2006, January 22). *Moscow names British "spies" in NGO row*. Retrieved from The Guardian: http://www.theguardian.com/world/2006/jan/23/russia.politics

Prados, J. (2010, June). The Navy's biggest betrayal. *Naval History Magazine, 24*(3). Retrieved from U.S. Naval Institute: http://www.usni.org/magazines/navalhistory/2010-06/navys-biggest-betrayal

Rendon Group. (2011, January). *Conficker Working Group: Lessons learned.* Retrieved from Conficker Working Group: http://www.confickerworkinggroup.org/wiki/uploads/Conficker_Working_Group_Lessons_Learned_17_June_2010_final.pdf

Reynolds, J. (1989, December). *The helminthiasis of the Internet (RFC1135)*. Retrieved from Internet Engineering Task Force - Network Working Group: https://tools.ietf.org/html/rfc1135

Rollo, T. (n.d.). *Description of the DCC protocol, a*. Retrieved from Internet Relay Chat Help: http://www.irchelp.org/irchelp/rfc/dccspec.html

Rossow, C., Andriesse, D., Werner, T., Stone-Gross, B., Plohmann, D., Dietrich, C. J., & Bos, H. (2013). SoK: P2PWNED - Modeling and evaluating the resilience of peer-to-peer botnets. *Security and Privacy, IEEE Symposium on* (pp. 97-111). Berkeley, CA: Institute of Electrical and Electronics Engineers. Retrieved from http://www.christian-rossow.de/publications/p2pwned-ieee2013.pdf

Specht, S. M., & Lee, R. B. (2004). Distributed denial of service: Taxonomies of attacks, tools, and countermeasures. *Proceedings of the International Workshop on Security in Parallel and Distributed Systems, 2004*, (pp. 543-550). Retrieved from http://palms.ee.princeton.edu/PALMSopen/DDoS%20Final%20PDCS%20Paper.pdf

Stewart, J. (2008, October 15). *The return of Warezov*. Retrieved from Dell SecureWorks Counter Threat Unit Threat Intelligence: http://www.secureworks.com/cyber-threat-intelligence/threats/warezov/

Stone-Gross, B. (2012, July 23). *The lifecycle of peer-to-peer (Gameover) ZeuS*. Retrieved from Dell SecureWorks Counter Threat Unit Threat Intelligence: http://www.secureworks.com/cyber-threat-intelligence/threats/The_Lifecycle_of_Peer_to_Peer_Gameover_ZeuS/

Weiss, A. (2012, October 26). *PRevent web attacks using input sanitization*. Retrieved from eSecurity Planet: http://www.esecurityplanet.com/browser-security/prevent-web-attacks-using-input-sanitization.html

Werner, T. (2011, September 28). *Botnet shutdown success story: How Kaspersky Lab disabled the Hlux/Kelihos botnet*. Retrieved from Kaspersky SecureList: https://www.securelist.com/en/blog/208193137/Botnet_Shutdown_Success_Story_How_Kaspersky_Lab_Disabled_the_Hlux_Kelihos_Botnet

Zeltser, L. (2011, June 28). *When bots use social media for command and control*. Retrieved from Lenny Zeltser on Information Security: http://blog.zeltser.com/post/7010401548/bots-command-and-control-via-social-media

Image Credits

The kitten image incorporated into Figure 7 was created by Larisa Koshkina and is in the public domain. It was obtained from PublicDomainPictures.net: http://www.publicdomainpictures.net/view-image.php?image=29803&picture=kitten