RFB: Resilient Facebot

Orie Steele
Stevens Institute of Technology
osteele@stevens.edu

Simon Sidhom
Stevens Institute of Technology
ssidhom@stevens.edu

Tom Parisi
Stevens Institute of Technology
tparisi@stevens.edu

Ken Bodzak
Stevens Institute of Technology
kbodzak@stevens.edu

Abstract

Social Media has proved to be becoming a major tool for communication between peers on the net. In addition to this, botnets have been and currently still pose a huge threat to the Internet and its users. Protecting against botnets can be difficult due to the numerous architectures and command and control schemes that can be utilized. Previous work has shown the C&C capabilities that can be provided by utilizing social networks. We intend to show how these social networks can provide more than a C&C infrastructure, and to explore the possibilities of creating a low-key, resilient, social media based botnet.

Here we propose an architecture for a peer to peer botnet leveraging steganographic communication within existing social networks. We rely of python for our web interface, task execution framework, social network interface, media and steganographic services. We discuss the growth of this botnet by the infecting of victims through links on bot profile(s) in social networks. A number of exploits could be leveraged to achieve infection. Finally we outline the inherent threats of such an architecture and propose means for defending against such a botnet.

1 Important Definitions

bot master- A controller of a botnet
zombie- A compromised computer connected to a botnet
botnet- A set of zombies that can be controlled by a bot master
command-and-Control- The method by which a bot master controls zombies in the botnet
steganography - The art of hiding in plain sight. In our case hiding data in media.
captcha- An image related to text that is obscured in hopes that the text is human readable but not machine readable
cracker- A malicious user
obfuscation- Hiding an intended meaning
malware- Malicious software

2 Introduction

Social networks are a new and popular form of expression on the web. Users of these networks enjoy a space to share text, music, videos and pictures. All of these things are useful to the user; we show that they are useful to a botnet as well. Almost any kind of media can be encoded with stegonographicly hidden data. The data hidden can be anything from a command to an entire file. As you may know, a botnet is not very useful if its command and control structure is unprotected. For example a botnet that communicates on Twitter through just commands in plain text would easily be found out. Likewise, a botnet that uses encrypted text would generate messages that dont look like a normal users traffic. To get around this Resilient Facebot (RFB) hides encrypted messages within normal looking user traffic.

Many users on social networks enjoy sharing funny pictures with their friends. What better place to hide commands could there be? A single user could post multiple pictures and links to pictures in one day. Each of these pictures can be encoded with command and control instructions for RFB. Perhaps an even more advanced version of RFB would also post photos that dont have stegonographic content to keep up the appearance of a normal users activities. It could also post text that looks like something a normal user would say. All of these things allow RFB to blend in with the growing number of users on social networks. RFB could be even further advanced by looking at a series of normal users traffic and mimicking it to avoid detection.

The amount of space available for stegonography in images allows RFB to store whole files within pictures without changing the pictures appearance to the naked eye. That is, a bot master can encode a python script into a picture and post the picture to a social network. The zombies could then retrieve the picture, extract the script, and execute it. These python scripts can do anything from distributed denial of service attacks to data mining and click fraud. If undetected RFB could grow large enough to take down any website with a single post of a funny cat picture containing instructions to attack a certain site at a certain time for a given duration.

How the commands are encoded is only half the battle when it comes to a resilient botnet like RFB. The other half is the structure of the botnet. Some botnets are set up in a hierarchical manner. These are susceptible if a high level node of the botnet becomes compromised. The top down structure means that all nodes below the compromised node will stop receiving instructions. RFB uses a peer to peer communication model. This allows zombies to keep receiving messages even if multiple other zombies have been compromised.

The best place to hide is out in the open. We take this old saying and show that it applies to new technology as well. Our botnet implementation, RFB, exploits this free sharing for its command and control structure. To prevent detection Statusnet implements steganography to hide commands in plain sight. The purpose of RFB is to demonstrate how resilient a social network based botnet can be.

3 Related Work

In order to gain a starting point, information was gathered from the following works. Each work is summarized below To give an indication as to why it was used.

Antisocial Networks: Turning a Social Network into a Botnet
E. Athanasopoulos, A. Makridakis, S. Antonatos, D. Antoniades, S. Ioannidis, K. G. Anagnostakis, E. P. Markatos

Since there was only a weak understanding of how to turn a social network into a botnet among the group, the Antisocial Networks: Turning a Social Network into a Botnet was read to gain a deeper understanding of how it can be done. Instead of giving the group a better understanding of how a botnet can be made through a social network, it told the group why a social network is a great place to house a botnet Command and Control server. First there is a lot of traffic already flowing in and out of the website. This is ideal for two reasons. The first is that there must be some type of load balancing mechanisms. This mitigates the chance of the server being taken down. The second is that the botnets malicious traffic can be obscured through the normal traffic with minimal effort. It also showed the group that the users of social networks, mainly Facebook.com, visit those sites in bursts. This means that the traffic from out botnet cannot be continuous otherwise it would look somewhat suspicious. Therefore, the botnet does not continuously connect to a social network.

Peer-to-Peer Botnets: Overview and Case Study
Julian B. Grizzard, Vikram Sharma, Chris Nunnery, and Brent ByungHoon Kang,David Dagon

From the very start of the botnet creation, it was known that the botnet was to be resilient. One way this can be done is by instituting at least partial peer-to-peer communication. In order to utilize the peer-to-peer topology, some research was needed. As the paper points out, a lot of information about peer-to-peer technology is already known through the use of peer-to-peer downloading clients like Napster and Kazaa. This paper is a case study on the Trojan.Peacomm botnet. Since the paper summarizes the whole botnet, only the communication pieces will be summarized because that is what this paper was used for. Trojan.Peacomm used an initial list of hosts to connect to. This method has two potential weaknesses. The first is that if none of the hosts respond, a viable node of the botnet will not get any of the instructions. Thus, making the botnet even smaller. The second issue is that the list of hosts could be used against the botnet by law enforcement. If law enforcement got a hold of the list, they could monitor those particular hosts and find many of the other infected bots. They could then shutdown the botnet. The paper also makes the point that the peer-to-peer design can help make the botnet more resilient because the bots will have more than one place to access information, the hosts in the initial list. If the bots use a central server, the owner of the DNS could not allow bots to connect to the central server thus crippling the botnet. Based on this work, it was decided that the peer-to-peer topology was necessary for resiliency.

Stegobot: A Covert Social Network Botnet
Shishir Nagaraja, Amir Houmansadr, Pratch Piyawongwisal, Vijit Singh, Pragya Agarwal, and Nikita Borisov

In order to restrict who can see the the instructions that the bots receive from the bot master, steganography is used. This paper discusses the steganography approaches to the Command and Control, C&C, instructions. The authors of the paper describe the botnet that was created for the testing of the steganography. However, since this is not the focus of why the paper

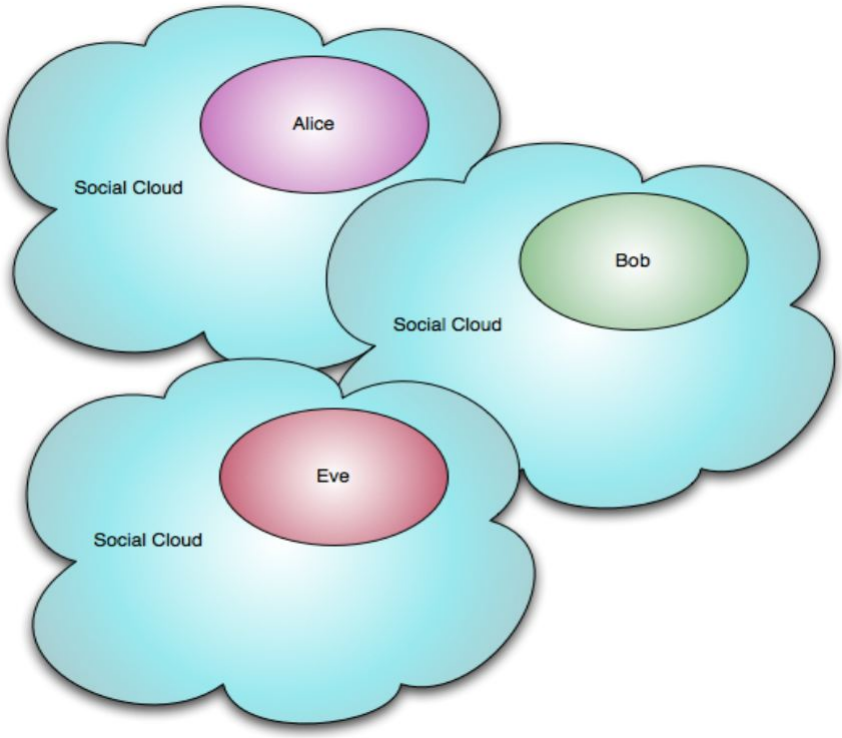

Figure 1: An illustration of peers in multiple social clouds.

was chosen to be read, it will be left out of the summary. The authors of the paper mainly used steganographic technique of placing C&C instructions within pictures mainly JPEGS. The results of the experiments using steganography is pretty is quite interesting. The authors found that not all images are created equally when it comes to harboring hidden information even if they are at a standard size and resolution. This alerted the group to the fact that our stegonagraphic images need to be carefully selected. The authors concluded that using steganography to hide C&C messages or messages between bots makes it harder to distinguish botnet communication from actual traffic. Even though there are techniques that use affect the carrier image minimally, they are hard to detect on the network. The only way to catch this type of communication is to assume that each image being Therefore, in an effort to make the our botnet more secretive, steganography is employed to hide our C&C messages.

Social Network-Based Botnet Command-and-Control: Emerging Threats and Countermeasures
Erhan J. Kartaltepe, Jose Andre Morales, Shouhuai Xu, and Ravi Sandhu

The hardest part of planning how to build the botnet was deciding how to pass the C&C commands from the social networking website to the bots. This is because the social networking websites are all publicly viewable sites. This means that any one can see them. Therefore, putting code on the social networking sites was not an option. As this paper describes, if commands are just posted to a social networking websites, the administrators of social network can detect these bogus profiles and take them off the website. The paper describes using social networking sites to post base-64 encodings of the commands to be performed. This bothered the group because it does not hide itself from the world. This is one one of the reasons that the group decided to use steganography. Steganography will help hide the botnet communications from the world and seem to be a more human profile user.

Understanding the intricacies of botnet Command-and-Control
Gunter Ollmann

Another goal of our botnet was to make it as resilient as possible. By making a botnet as resilient as possible, it is logical to assume that it will remain active for a number of years. Of course there will need to be updates to the botnet software but as long as the botnet can stay communicating, the updates can reach the bots. The paper outlines some of the different ways this can be done. The paper details the importance of having the C&C server readily available for obvious reasons; if the

bots cannot obtain new instructions, the botnet is dead. It details two ways of making the C&C servers available to the bots. The first is called IP Flux. This is where the botmaster constantly changes the IP addresses of the C&C servers that are associated with a particular domain name. There are also two ways of implementing the IP Flux. The first is called single flux where only the IP addresses of the domain name are registered and deregistered. The second way to implement IP Flux is to use something called double flux. This is where the IP addresses are registered and deregistered as before but different DNS servers are used to find the domain names. This helps to further obfuscate the botnet traffic. In addition to IP Flux, Domain Flux can also be used. This is the situation where bot masters have a single IP address but use many different domain names. One way of doing this is through the wild card character. This allows many domain names for a specific IP address. Yet another way to hide C&C messages and botnet traffic is to use proxies that look like the actual C&C servers but are actually just compromised proxies. This allows the proxies to be investigated instead of the C&C servers. Since this botnet uses social networks, many of these techniques are not necessary for this implementation. However, the paper made the group about what would happen if the C&C profile was to be shutdown. For this reason, there are multiple profiles within each social networking site that can be a C&C profile as well as multiple social networking sites that house many C&C profiles. The social networking infrastructures are perfect for this.

An Advanced Hybrid Peer-to-Peer Botnet
Ping Wang Sherri Sparks Cliff C. Zou

This papers outlines the weaknesses of both the CC server and peer-to-peer topologies of botnets. Of course, the server is the single point of failure in the single server topology. Even if there are multiple servers in the botmasters possession, eventually, they will run out of servers and thus the botnet will become unusable. In terms of detection, there will be an increased amount of traffic to a server alerting server administrators. Once the C&C server is discovered, the all of the bots are at risk of being discovered. Thus CC servers are a bit outdated. As for peer-to-peer botnets, the main weakness is in gathering a list of peers in which to communicate. Instead the paper proposed a combination of a centralized servers and peer-to-peer botnet. It uses a peer-to-peer protocol for the many servers being used. The servers that are used are actually bots themselves. The peers are found in a list that is downloaded in the original malware. This list of peers is not a complete list of the botnet and can be updated from one of the servant bots. The paper outlines a hierarchical structure for the bots. The bots are placed into two categories client bots and servant bots. The servant bots are the ones that will accept peer-to-peer connections where as the client bots will not. The servant bots are the ones that will appear in the peer list because they are the ones that will accept connections. The servant bots are more like a centralized server. This paper made the group think that perhaps it would be best to have a form of a hybrid structure. Of course, the central server will be the profile on the social network but instead of having every bot grab the C&C messages from the profile, only a few would grab at while others would connect to the peers to get the C&C messages.

4 System Design

Our botnet architecture is designed to achieve effectiveness and resilience while maintaining low visibility to third parties. The system involves bot masters, bots, puppets, and the use of social networks for command-and-control infrastructure. Our bot masters are the user issuing command-and-control directions to the network that should result in the completion of the request. Puppets refer to social network members / accounts that do not entirely represent a true user of the social network. Puppets are the building blocks that allow us to utilize social network infrastructure as a command-and-control server. Bots or Zombies are infected computer systems that can interact with puppets or other bots to receive command-and-control traffic, and act out any instructions received. Finally, social networks like Facebook, Twitter, etc. provide the infrastructure needed to serve as our command-and-control server. Our infrastructure will be designed to be social network agnostic, as to allow the incorporation of multiple social networks into the design in addition to make it extremely versatile.

To accomplish our goals of resilience and low visibility, we plan to use the benefits employed by the social networks themselves , steganography, and the high volumes of social media traffic that is becoming more and more present on the Internet today. Utilizing social networks as our command-and-control servers, we are able to leech off their own hard work at delivering a reliable service to their customers. If our social network puppets are sufficiently able to become a part of the social networks user base then our command-and-control server will be afforded the reliability and resilience of the social network being used. To remain low-key, our botnet will attempt to lose itself amongst regular social network traffic. Utilizing available steganographic techniques, our bot net will hide command-and-control messages within what appears to be ordinary social media usage. This includes, but is not limited to posted images, status updates, user messages, and profile updates.

To provide an even greater level of resilience and obscurity, the network is able to communicate using multiple social networking infrastructures. This allows the botnet to create a distributed web of connections amongst infected machines and puppets on various social networks. See Figure 1 for a simplified diagram of peers within connected social clouds.

4.1 Communication

The flow of communication within our proposed botnet involves both command-and-control traffic from a botmaster and communication between infected peers. This communication can be used to further relay command-and-control messages between peers that are in contact with social network puppets on disjoint networks. This type of communication is vital to our botnets resilience and diminished visibility. The flow messages through these social network based command and control servers will be designed to be low-key ontheir own, but the fact that infected hosts can also communicate via other p2p means or even communicate using an alternate social network command-and-control server will greatly increase the obscurity. Furthermore, by increasing the amount of possible means of connections between hosts in our network we also increase the amount of redundant pathways to send communication through. This increase in redundancy leads directly to an increase in the resilience of botnet should a single means of message transport be severed.

Our botnet embraces the vast amount of popularity that social networks have received recently, and uses it to hide in plain sight. Users post hyperlinks, textual updates, audio, video, and images to these social media websites extremely frequently, and even more users access these uploads and sometimes post their own replies to them. With so much information sharing going on, an infected host will be able to easily access shared data through these social networks while appearing as an ordinary user. In addition to simple accesses, information and files will be able to uploaded as well. This type of behavior is almost necessary because the constant accessing and trading of data over these social networks is exactly what makes up the usual everyday behavior for a typical user. It is in these common routines of logging into a social network, trading and accessing data, and repeating the process that we will hide our botnets command-and-control traffic.

Previous work like the proposed system Stegobot [8], outline solutions to creating unobservable communication channels over social networks. A key idea in these systems is to exploit the possibilty of stenagraphically hiding data inside of images. Images are an extremely popular in social networks, which will provide a sufficient crowd for which our botnet can hide its message within. Images have proven to have a large capacity for embedded information, in terms to steganography. We use this capacity to embed our botnets communication and provide a passageway for command-and-control traffic to flow through a social network from bot to bot. Since repetive images are extremely common sights on social networks, popular images can simply be downloaded, manipulated, and re-uploaded to a social network without straying from common social network activity.

Further obscurity and protection for the botnet can be achieved by encrypting the data that is embedded into these images via steganography. This stops botnet command-and-control traffic from being analyzed if it is detected, and furthermore allows us to use encryption at a level that wont be visible to third parties. Our messages will pass through the social network in the form of commonly found images, accessible by many users but containing instructions for zombies that know where to look.

4.2 Communication between social networks

Social networks pose as central nodes that zombies can access to retrieve any command-and-control messages that have been posted. This centralized method is powerful in its own right, as it allows the peers to achieve mass communication without directly interacting with each other. In terms of resilience, this centralized method can have pose a threat due to the widespread reliance on the social network for relaying messages.

Our botnet is extensible to multiple social networks, and will encompass the possibility of an individual social network becoming compromised or unusable. To circumvent this scenario we take advantage of zombie hosts that have the ability to communicate on multiple social networks. These hosts are capable of identifying that one centralized node has become unusable and now leverage its connection to a still-working social network to perform triage to the botnet as whole. The incorporation of multiple social networks into our botnet provide an extremely resilient means of transporting command-and-control information to zombie hosts.

In Figure 2 we see that both Alice and Bob share communication channels between multiple social networks. These passageways allow information to be conveyed from say Twitters network to Facebooks Network, and vice-versa. This structure provides resilience in the respect that if Bob lost his connection to Facebook, Alice could convey the same information over to Twitter, where Bob is still connected. In addition, the ability for command-and-control traffic to be both shared within a social network, but also between social networks allows

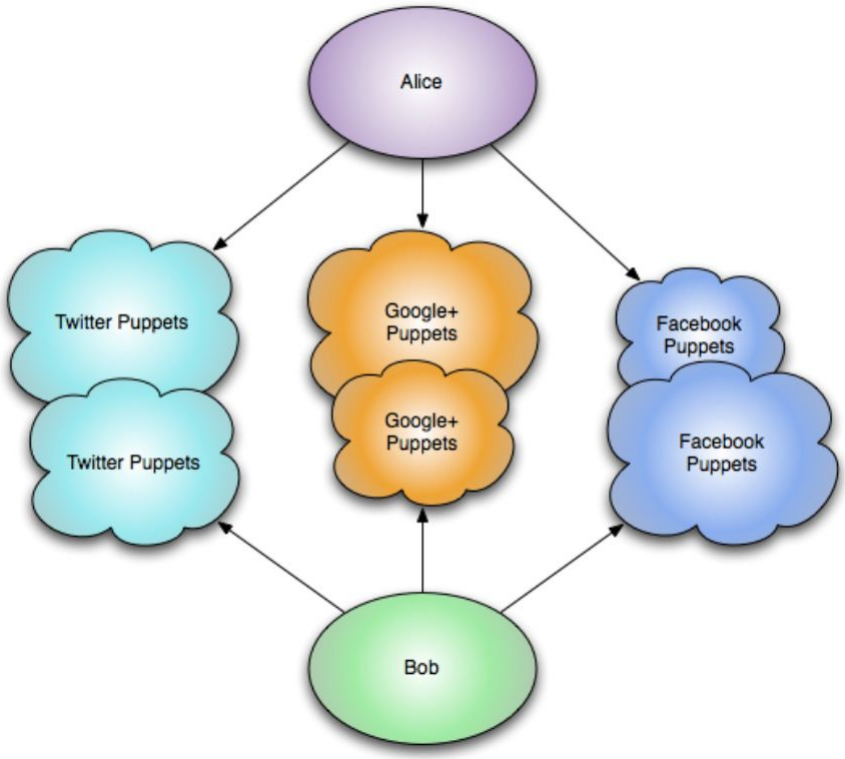

Figure 2: Peers connected via multiple social networks.

for the botnets zombies to be widespread and share very few connections in common with one another; affording resilience and obfuscation.

4.3 C&C Messages

As mentioned previously, our command-and-control messages will exist within images posted to a social network utilized by the botnet. These messages are steganographically hidden within the images, allowing the host image to appear innocent to the social network while still serving as a command-and-control payload for infected hosts looking for instructions. We go about creating these command-and-control packed images by first selecting an image that will be our host. We are looking for a common image, that we can easily upload and post to a social network for our zombies to have access to. This can be accomplished by using a hosting service built into the social network or an external image hosting service like imgur.com. What defines a good candidate for this step, is that the service is popular, common, and does not modify the uploaded image in a way that could damage any data that is covertly embedded within it. With these criteria met, zombies will be able to retrieve the image and interpret the hidden payload effectively while also hiding their tracks amongst the large amount of regular traffic that accesses images from these services every day.

Our process of creating these images can be seen in Figure 3. We start with our raw command-and-control message. This will most likely be some kind of input that can be interpreted by our malware running on an infected host. In the illustration and our later discussed implementation we will use a serialized data type to contain our command-and-control message. This object will then be encrypted and signed to both protect the contained instructions and also ensure that they have remained intact. At this point we can use steganographic techniques to hide this data within an image of our choice. With our new loaded image, we can finally upload it to a popular hosting service and use a social network to make it available to our zombies.

4.4 Interpretation

With command-and-control messages available in social networks, we must now discuss a means for our bots to retrieve the messages. There are two ways in which we will go about retrieving relevant information from the websites being utilized by the botnet. The first way is though direct access to a website and parsing the html for structured data. This approach is very fast, but also easily detected as automated traffic. A second approach is through a web-driver. The web-driver approach will

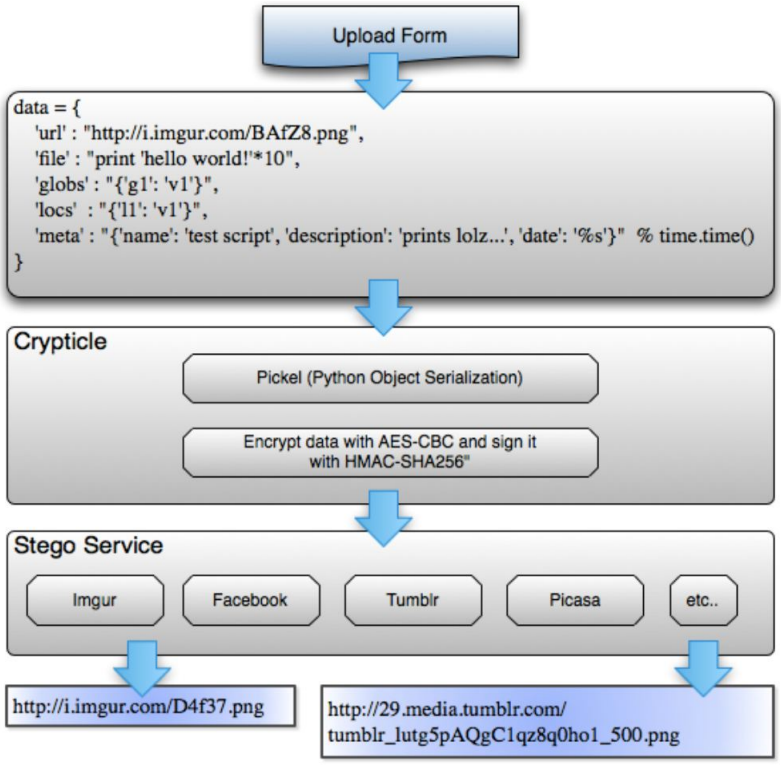

Figure 3: Embedding C&C into an image and uploading to the botnet.

be able to control a web-browser from the infected host. While this approach gives the bot an enormous amount of functionality, as it will be able to do just about anything that could be achieved via a web browser it has many downsides. This approach will be both slower than direct access, and extremely obvious on the host system. Regardless of the method used, the bot will search the social network for instructions and be able to download and interpret media containing embedded command-and-control instructions that will then be carried out.

5 Propagation

This section is about how the compromise the systems so that the botnet software can be loaded onto the target system and become part of the botnet. The best case scenario would be for an exploit to work on any system regardless of operating system so that the botnet will have diversity within it. This is a possibility because the botnet is written in python which is portable between operating systems. Currently, it runs on Windows, Linux, Unix, and Mac OSX. This is means that if there is an attack vector that all of the above operating systems share, all of the previous systems can become part of the botnet. Obviously, Windows is vastly different from the other operating systems but each operating system is different enough that there is not a single attack vector between them within the normal operating system. Instead attention must be used on the way the operating systems connect to the internet because it is the biggest commonality between the operating systems. Each operating system needs a browser to connect to the internet. This browser can be used against the system to compromise it for the botnet. Since the botnet has been developed for the use with the Linux operating system, more specifically Ubuntu, Linux will the main focus but the same basic steps can be used to compromise all systems. After the compromise has taken place, privilege of the user needs to be escalated to root to install necessary programming libraries as well as a backdoor to the system. Of course, the botnet could be installed locally to only the compromised user but that could make it much easier to detect because only the specific user would have access to his or her files. Thus if the libraries are installed globally, there is more of a chance that the user who found them will believe that they belong to someone else. Unfortunately, the browser does not have to be run as the super user of the system. This makes gaining root access slightly harder but still feasible. Gaining root access will allow continued access to the system incase manual changes need to be made to the botnet.

5.1 Compromising the Host

As stated before, a huge commonality between all of the operating systems, Windows, Linux, Unix, and Mac OSX is the browser. Compromising the browser is a way to introduce diversity into the botnet.

Importance of Diversity

Diversity, in the technological sense, is quite important if an entity wants to remain running for a long period of time. Diversity means that key aspects of an entity are different. In this case, the key aspects of the bots are the operating systems. If the creator of the operating system, Ubuntu for example, issues a patch that cripples the botnet, all of the Ubuntu bots will be lost until a patch for can be created for the botnet. If only Ubuntu systems are created, then the botnet would cease to exist. However, if multiple operating systems are used, the botnet would not be lost; only the Ubuntu bots would be.

Compromising the host

After weeks of developing a way to get into a victims browser to no avail, Metasploit became a viable option. Metasploit is an exploitation framework that can be used for penetrating a system. In this case, it will be used to set up a server that hosts a malicious java applet. At first glance, this may seem to be a very inefficient way to infect many systems. However, if the domain name is carefully crafted, the server can have great results. This is true because the domain name can be something that is very close to a popular domain name. For example, the domain names www.faecbook.com or www.epsn.com would be desirable because both facebook.com and espn.com have a lot of traffic to them. If the URL is typed wrong, which happens often, the java applet would corrupt the browser and the system is compromised. These websites will have to have a homepage that is very similar to the normal webpages for it to be convincing. For instance, the www.faecbook.com malicious page can be the page of facebook.com that asks the user to log in. However, before the user logs in the applet must first be run. Many users will click run because they will think that they are on facebook.com so it is safe. Once the user tries to login, the information can be sen to the real facebook.com and the user can be logged in. This will allow the attack to be transparent to the user. In general, having a similar malicious page to the homepage that is being spoofed is ideal because it will distract the user from detecting the attack. Another necessary level is to have all the links on the malicious homepage link directly to the real website. This way, everything seems functional. First, the Metasploit framework was used to confirm that the corrupted java applet would indeed work on systems. The signed java applet is a different kind of exploit then the normal buffer overflow exploits usually used. Instead it takes advantage of the java virtual machine that runs jar files. Generally, jar files are run in the java virtual machine making it sandboxed. This means that it cannot access the physical machine and thus, it cannot do harm. However, there is a flaw in the logic of the Java creators. The sandboxing can be circumventing if the applet that is run is signed by another system. If the applet is signed, the applet does not have to be run in the sandbox and thus can access the physical system. Not only does it access the physical system but it runs with the same privileges that the user who invoked it has. Therefore, if a person accidentally runs a signed Java applet as root, the applet has root capabilities. Even though there is a pop-up box with the information about that a Java applet wants to run, most users will not read it and just click run. This also mitigates the risk of the shell closing when the browser closes. Since the shell is running on the actual machine instead of within the browser, the shell will continue to run. To get just a shell the commands there are specific commands to run within the Metasploit console. [6]

```
use /exploit/multi/browser/java_signed_applet
set TARGET 2
set PAYLOAD linux/x86/shell_bind_TCP
set SRVHOST x.x.x.x
set LHOST x.x.x.x
```

The above commands will set a webserver that will server the signed applet. Once a user connects to it, the user is prompted with a message from the browser asking if they would like to run the applet or not. If they click the run button, a shell is generated for the attacker. In this shell privilege escalation can be started. However there is a complication if the with this type of shell; it does not support text editors. The remedy for this predicament is to relaunch the above applet using the Meterpreter. This allows a user to upload files to the compromised host.

```
use /exploit/multi/browser/java_signed_applet
set TARGET 2
set PAYLOAD linux/x86/meterpreter/reverse_tcp
set SRVHOST x.x.x.x
set LHOST x.x.x.x
```

With this configuration, the privilege escalation can be done. The Meterpreter allows the attacker to start a shell on the target system. Privilege Escalation After gaining access, the attacker needs to gather the information of the system. For example, if it is a Linux system the attacker can execute the command

```
uname a
```

This will return the vital information on the system like the version of the Linux kernel being run. To find exploits for this version of the kernel, the websites exploit-db.com and milworm.com have a plethora of exploits for privilege escalation that in the worst case need to be compiled and run. In stead of reinventing the wheel this code should be used.

6 Implementation

The main software component for our botnet will be a python based application designed to allow infected hosts to communicate and act based upon the command-and-control traffic received through various social networks. Python was chosen due to its versatility and the large amount of libraries available to aid in development and get our bots do what is desired of them. A general outline of the bot can be seen in Figure 4. Bots will follow a simple lifecycle that involves:

1. Host infection
2. New bots go through a stage of peer discovery, where theya ccumulate a local database of social network profiles to follow and post to.
3. Once a sufficient number of peers has been established, the bot can begin carrying out missions. Missions are python scripts which can be executed in the context of the bot (with access to bot resources) or a separate attack scripts (flood scripts for example).
4. Report results of mission through posts to social networks.

As a proof of concept, we have implemented some of the functionality that such an RFB bot would use in the form of a locally run web application. This application allows users to upload attack scripts and arguments via web forms. The result of uploading such a script is a link to an image containing all the relevant info for that mission. This is accomplished by first parsing the form input and constructing a dictionary such as the one below:

```
data = {
   'url' : "http://i.imgur.com/BAfZ8.png",
   'file' : "print 'hello world!'*10",
   'globs' : "{'g1': 'v1'}",
   'locs'  : "{'l1': 'v1'}",
   'meta' : "{'name': 'test script',
   'description': 'prints lolz...',
   'date': '%s'}"  % time.time()
}
```

Here the url is the location of the image we would like to use to embed this dictionary in. This dictionary is then serialized with pickle (pythons object serialization protocol), encrypted with AES-CBC, signed with HMAC-SHA256 and encoded to base64. This base64 string is then passed to the stego service for embedding and uploading. The stego service is designed to support a number of different wrappers for media sharing services, but currently only supports imgur. The imgur service first downloads the image specified by the the url in the data dict above. The image is converted to BMP format, and then the base64 string is embeded in the image using the least significant bit. The embedded image is then encoded to base64 and uploaded to imgur using their PyCurl API, which returns a link to the image that has been uploaded. This link is then stored in the bots local db along with the data object described above.

Through the use of another form, a user can then construct a public post which will include a link to the image containing a chose script, and then choose a number of puppet social network profiles to post to. We also provide forms for adding puppet profiles, and a tool for scanning a url for embedded messages. Puppets can be controlled in two different ways.

The first is through the use of a web driver. This allows for scripted access to a web browser of our choice. The advantage of this approach is that the bots traffic will look like normal web traffic, because the bot is using a normal web browser to access resources inside the social networks. This approach also allows for complex missions such as online poll manipulation, comment spamming or any activity a user can carry out with a browser (aside from flash and silverlight activities). The disadvantage is that this technique is slow, and very obvious to the victim as the will be able to watch everything the bot is doing online in real time.

The second way of controlling puppets relies on leveraging url libraries and html parsers. This approach is much faster, but the traffic will be clearly automated. This approach is good for retrieving structured data from urls which do not require any sort of authentication.

7 Threats and Countermeasures

It is obvious that a single bot master leading an army of zombies is a dangerous thing. Why? Because there are a multitude of attacks that the bot master can employ. One very simple example could be a denial of service attack. That is, the bot master commands all of his zombies to access a certain site hundreds of times a minute until the servers bandwidth is saturated and and the website goes down. Along the same lines, a bot master may have a website with advertisements on it that generate revenue based on the number of clicks by different users. The bot master can simply command the zombies to go to his site and click the advertisements. This is known as click fraud.

Another, less obvious, attack is to use distributed

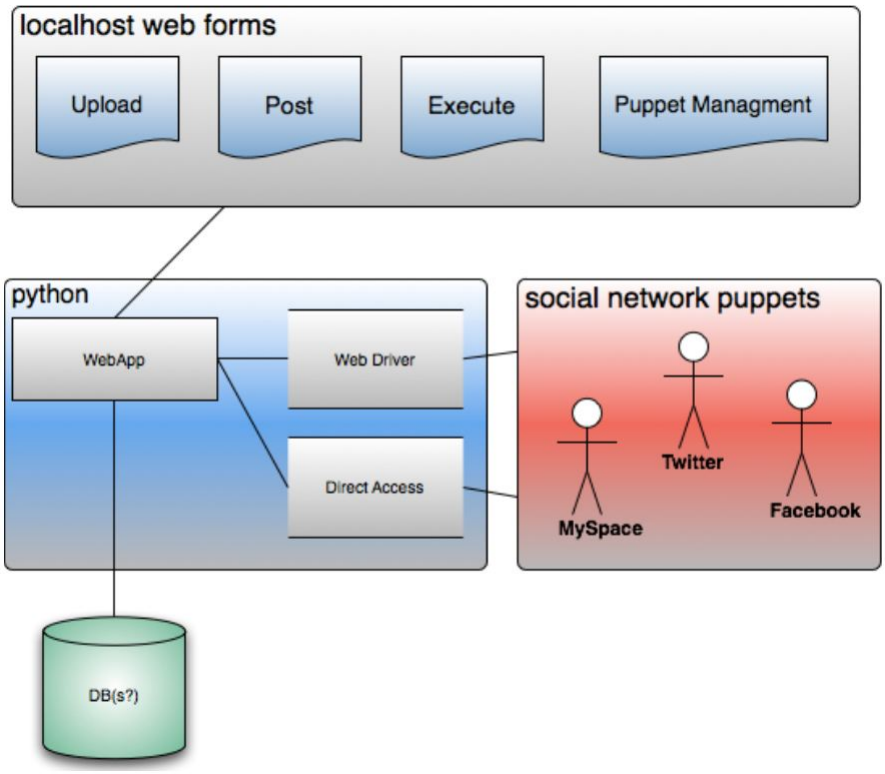

Figure 4: Diagram of our bot design.

computing techniques and a botnet to do brute force attacks on anything from password hashes to SSL (Secure Sockets Layer) keys. With a big enough botnet a bot master can crack encryptions in orders of magnitude less time than it would take for him to crack them on a single machine. Some sites dont allow more than a few log-in attempts from a single IP address in a given amount of time. The bot master has multiple IP addresses to work with so he can get around this block.

There are other forms of attacks like adware and spyware. A botnet using adware would present the user of a compromised machine with advertisements that would not otherwise be present. Spyware collects information on the user of a compromised machine and returns that data to the bot master. A bot master can also use zombies to send email spam that looks like it is coming from legitimate users. This type of phishing sometimes leads to the installation of scareware. Scareware is malware advertised as anti-virus software. It infects the computer and forces a user to pay for protection from it.

Without the initial formation of a botnet the bot master will not be able to preform the attacks that he would like to. There are two ways to keep the bot master from attacking. The first way is to prevent computers from being compromised in the first place. The second way is to cut off command and control communications between the bot master and the botnet.

A computer can be compromised in any number of ways. Any exploit that allows running code on a remote machine can be used by a bot master to add to his army of zombies. If we could prevent the bot master from ever being able to run code remotely we would be done. Unfortunately, the reality is that most system administrators dont really know about the risks of leaving a computer unprotected. This is because they arent real system administrators they are regular people with no training in system administration or cybersecurity. The fact of the matter is that unless all computers patch up all security flaws automatically, there will be users left unprotected that a bot master can take advantage of.

We need to accept that it is possible for a bot master to compromise a set of zombies and start thinking about what to do to stop it after it is in place. There is an old saying that holds true for botnets, You cant fight if you cant see. In the case of botnets seeing means communicating. Once a bot master can no longer give commands to his zombies he can no longer launch attacks.

The first step to stopping communication is finding it. In the case of RFB data is hidden inside images. In order to extract that data, without knowing the steganographic method used to encode it, one must run expensive algorithms that analyse the image and find patterns.

As in RFB there can be a layer of encryption under the steganography that makes it even harder to find and subsequently read command and control data hidden in the image. To find all the zombies one must run this algorithm on a huge set of images. Without incredible processing power this method may not be feasible in the near future. The next thing to try would be to create an algorithm that crawls the social network in search of users that make suspicious posts. Such an algorithm may be possible but, once a bot master figures out how the algorithm classifies profiles as real he can design bots that pass the test.

If the above method of finding communications doesnt work then we may need to consider trading convenience for security in some cases. If we dont want bots to be able to post on social networks then we may need to include captchas that verify that someone is a person and not a bot. As with any security measure, a dedicated cracker can get around this. The field of computer vision is a hot topic for research right now and is making great advancements. Using computer vision techniques a cracker can theoretically decipher any captcha that a human can. Of course more captchas means more time spent by the cracker trying to decipher them. It also means more time spent by every single legitimate user to decipher the captchas.

More of an after the fact method for getting rid of a botnet is detecting where attacks are coming from. This would allow identification of infected computers. A security professional can then un-compromise the machines and create a signature for the virus that infected them. This virus signature can be used to detect if a given machine is a zombie. A patch can then be employed that will cure the infected computer. A dedicated cracker can create a botnet that uses obfuscation of code to change its signature on a regular basis. This would nullify the possibility of detection based on signature. Code obfuscation is accomplished by adding and moving around code to make the program appear different even though it performs the same job. This method of adding noise relates back to the way RFB could one day adapt postings on a social network based on the trend of postings by real users. The more real users there are to copy the harder it is for a bot to be found out.

When it comes to botnets it seems that where there is a will there is a way. Short of securing every computer in the world, it seems there is no way to guarantee that botnets will not form. With enough time and skill a cracker can design a botnet like RFB that avoids detection. Like RFB, the botnet could be peer to peer to avoid loosing large branches of zombies. It can also be resilient to anti-virus signatures by using obfuscation of code. Botnets are a real threat and it seems that the only way to stop them is to take system administration out of the hands of the untrained user. Maybe cloud based computing systems with trained administrators are the answer. Or maybe automated security patches are a better short term solution. Either way the point is to stop a cracker from ever being able to run code your machine in the first place.

8 Conclusion

By hiding in plain site and blending in with normal users, RFB can do major damage. It is a framework that can be used to do denial of service attacks as well as brute force cracks. RFB is resilient to detection. Zombies act like regular users and post inconspicuous pictures. Therefore, RFB zombies are hard to identify. If any one zombie is identified the rest of the botnet will still function. This is true because of the peer to peer approach that RFB takes to command and control.

As social networks grow more and more popular they become bigger targets for bot masters. The wealth of user data on these sites allows RFB to blend into the noise and operate undetected. Using encryption and steganography RFB is able to pass around command and control messages in a peer to peer way. Further improvements include obfuscation of code to hide from anti-virus signatures. This would allow the botnet to still function even if anti-virus programs create signatures for it. Generation of data that looks like it comes from a legitimate user is another future improvement. It would make it even harder for zombies to be detected. RFB would accomplish this by using data generated based on the posts of actual users on the social network.

There are defenses against RFB but with more time and effort from a dedicated bot master almost all of the solutions can fail. If a bot master can execute code remotely on your machine then it is possible for your machine to become a zombie. The only way to stop botnets completely is to lock down security on every machine. That means distributed patches for security holes fast. This paper aims to awaken users to the real threat of botnets on social networks. They can be sleek and fly under the radar and they can be very hard to stop. If you dont lock down your computer you may be the next unknowing owner of a zombie machine controlled by a bot master you have probably never met.

References

[1] J. B. Grizzard, P. S. Lomdahl, et al.
Peer-to-Peer Botnets: Overview and Case Study
USENIX Association. (2007)

[2] Jose Nazario
Twitter-based Botnet Command Channel
`http://ddos.arbornetworks.com/2009/08/`

twitter-based-botnet-command-channel/
(2009)

[3] Rope
Python refactoring framework
http://rope.sourceforge.net/library.html#quick-start

[4] US-CERT
Microsoft Internet Explorer buffer overflow in PNG image rendering component
http://www.kb.cert.org/vuls/id/189754
(1995)

[5] Erhan J. Kartaltepe, Jose Andre Morales, Shouhuai Xu and Ravi Sandhu
Social Network-Based Botnet Command-and-Control: Emerging Threats and Countermeasures
Lecture notes in Computer Science (2010)

[6] Rapid7 Community
Recent Developments in Java Signed Applets
https://community.rapid7.com/community/metasploit/blog/2011/05/26/recent-developments-in-java-signed-applets
(2011)

[7] Athanasopoulos, Elias, Makridakis, et. al.
Antisocial Networks: Turning a Social Network into a Botnet
Lecture Notes in Computer Science (2008)

[8] Shishir Nagaraja, Amir Houmansadr, et. al.
Stegobot: A Covert Social Network Botnet
Lecture Notes in Computer Science (2011)

[9] Gunter Ollmann
Botnet Communication Topologies Understanding the intricacies of botnet Command-and-Control
Damballa, Inc. (2009)

[10] Ping Wang, S. Sparks, C.C. Zou
An Advanced Hybrid Peer-to-Peer Botnet
IEEE Transactions on Dependable and Secure Computing (2010)