Covert Botnet Command and Control Using Twitter

Nick Pantic
Cal Poly Pomona
3801 W Temple Ave
Pomona, CA
nmpantic@cpp.edu

Mohammad I Husain
Cal Poly Pomona
3801 W Temple Ave
Pomona, CA
mihusain@cpp.edu

ABSTRACT

Botnets are one of the primary threats in computer security today. They are used for launching denial of service attacks, sending spam and phishing emails, and collecting private information. However, every botnet requires coordination. In order to initiate an attack, a botmaster must communicate to all of the bots in the network. In this paper, we present a steganographic system that demonstrates the feasibility of the social networking website Twitter as a botnet command and control center that an attacker could use to reliably communicate messages to a botnet with low latency and nearly perfect rate of transmission. Our system generates plausible cover messages based on a required tweet length determined by an encoding map that has been constructed based on the structure of the secret messages. The system considers both the input symbol frequencies (e.g. English letter frequencies) as well as the tweet length posting frequencies for constructing the encoding maps. A technique for automatically generating Twitter account names based on Markov chains is also presented so that the bots can connect to new accounts if the existing botmaster account is unavailable. All the experiments were performed using the $7.3M$ actual tweets from $3.7K$ verified accounts collected by the tweet parser developed by us. We have evaluated the efficacy of the system using Emulab and usability of the system through Amazon's Mechanical Turk with promising results. An analysis of the steganographic security of the proposed system has also been provided. By demonstrating how a botmaster might perform such communication using online social networks, our work provides the basis to detect and prevent emerging botnet activities.

Keywords

Twitter, Denial of Service, Social Network, Botnet, Command and Control, Steganography

1. INTRODUCTION

Computing and interconnectivity have spread through modern society as electricity and plumbing have in the past, to become almost entirely ubiquitous. Indeed, it is not uncommon for a single person to possess numerous computing devices of varying power and portability, ranging from handheld smartphones and tablets to notebooks and desktop computers. Although these devices appear different, they are all essentially the same. They act as general purpose computers that connect to the Internet to communicate with other devices across the globe.

Cyber-criminals make use of this vast global Internet by installing or convincing users to install malicious software, or *malware*, on to their devices that allow the criminals to control them remotely. A collection of these "zombie" computers is called a *botnet*. Botnets are one of the most prominent modern computer security threats [17] and are often used for various forms of cyber crime such as sending spam emails or performing *denial-of-service* (DoS) attacks against other computer networks. In fact, the botnet threat spreads beyond what we commonly refer to as computing devices. In the new *internet-of-things* (IoT), many common household appliances that contain embedded computers are being connected to the Internet. A recent news story showed that these smart appliances, such as refrigerators, were being used to distribute spam email[1].

The design and communication patterns of these botnet can vary dramatically, as they are created by cyber-criminals with the intent of hiding their presence. Social networks have exploded in the past few years in the same way that the Internet and the web before them. Today, popular social networks like *Twitter* and *Facebook* have hundreds of millions of users interacting and communicating in real time. Even with the extremely large user bases, these services are rarely ever unavailable and will transmit the communications at an incredible speed. From this information, a clever attacker will recognize that these networks are well suited for crafting cyber attacks such as controlling an existing botnet. They can take advantage of the infrastructure, speed of transmission, and large userbase in which to hide to control the bots. This paper provides a proof of concept of this type of botnet communication that is hidden within the social network *Twitter*.

Understanding how attackers communicate with botnets is vital for botnet defense. If the attacker cannot coordinate the bots, they will be unable to utilize the network. As is common with computer security research, researching both attack and defense can be useful. Before a proper defense can be made, new attacks must be understood. Botnets are capable of attacking the availability of a system using attacks such as DoS, where the bot flood a network with requests to cause it to become unresponsive to real traffic. These attacks are especially dangerous because it is difficult to distinguish between these fake requests and authentic traffic trying to use the network. In order to stop these attacks, it is best to be able to cripple the botnet before it can begin. Therefore, understanding how an attacker might attempt to coordinate these bots is essential.

ACSAC '15, December 07 - 11, 2015, Los Angeles, CA, USA
© 2015 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-3682-6/15/12…$15.00
DOI: http://dx.doi.org/10.1145/2818000.2818047

[1]http://www.proofpoint.com/about-us/press-releases/01162014.php

In this paper, our goal is to develop a method of coordinating bots in a botnet that uses a *stego-system* over a popular social network, Twitter, using only meta data for communication. This *covert channel* can also be used for arbitrary communication of relatively short messages outside of the realm of botnet command and control. By utilizing *steganography*, we hide the existence of the botnet control communication from the outside world while also utilizing the power of the popular social networking website to ensure timely delivery of the messages.

Towards that, we have first developed a stego-system leveraging the Twitter social network platform. This system can be used for secret communication between various parties for many domains. Next, we have implemented a botnet command and control (C&C) communication system that utilizes the stego-system. This C&C system communicates entirely through the stego-system allowing the botmaster to control each of the bots. We have also developed a technique for automatically generating Twitter account names based on *Markov chains* so that the bots can connect to new accounts if the existing botmaster account is unavailable. Finally, we have evaluated the efficacy and performance of both the stego-system and botnet C&C. All the experiments were performed using the $7.3M$ actual tweets from $3.7K$ verified accounts collected by the tweet parser developed by us. We have evaluated the efficacy of the system using Emulab and usability of the system through Amazon's Mechanical Turk with promising results. We have also provided a detailed steganographic security analysis of the proposed system.

This paper is structured as follows: section 2 contains a broad literature review of the various techniques that are used in the development of the research. Section 3 describes the structure and implementation of the stego-system, botnet-cc, and other components of the system. Section 4 discusses the experiments conducted and the evaluation of the results of the experiments including steganographic security analysis. Section 6 contains our concluding remarks and future work related to the paper.

2. BACKGROUND

2.1 Steganography and Steganalysis

Confidentiality has been well established as a security criterion [7]. Essentially, confidentiality is the preservation of authorized access and disclosure to information [5]. In most cases, confidentiality is sufficient for protecting information from disclosure. For example, when using online banking, it is important to conceal the contents of the communications so that no others can impersonate either yourself or the bank or otherwise obtain your private banking information, but it is not usually important to hide the fact that you are performing the online banking. However, there are situations where it is not only important to hide the contents of communication, but also the fact that communication has taken place at all. This is the *undetectability* criterion of security, defined by Pfitzmann and Hansen [16] as the criterion of being able to determine if a message even exists.

Just as cryptography is the science related to confidentiality, steganography is the science related to undetectability [1]. Also, as cryptanalysis is the analysis of cryptographic techniques and how to break them [20], steganalysis is the study of steganographic techniques and finding hidden information [1]. From the definitions of both cryptography and cryptanalysis, we can deduce that a cryptographic system can be considered broken when an attacker can determine the contents of the communication, also called the plaintext. Therefore, we can consider a steganographic system (stego system) broken when an attacker can determine that secret communication has taken place, that is the attacker has *detected* the communication, even if they have not determined the contents of the message [1].

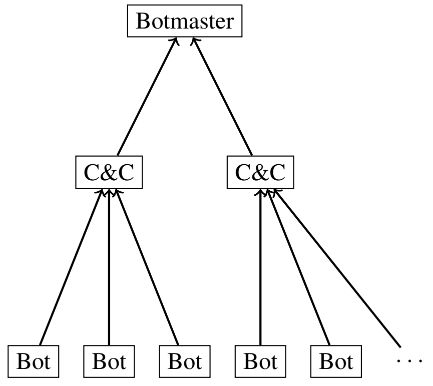


Figure 1: Botnet C&C for a centralized botnet with multiple C&C centers.

2.2 Botnets

Botnet software is a type of malicious software (malware) that is most often placed on a victim's computer silently. Unlike traditional malware, however, the botnet software communicates with a botmaster or bot-herder that coordinates potentially thousands or even millions of other infected machines, called bots or zombies, in other attacks. Once created, a botnet can be used for harvesting personal information on a global scale, or causing significant denial of service attacks to even the largest organizations [11].

Every botnet must have a command and control (C&C) system that directs the bots to perform their attacks. Zeidanloo and Manaf [22] have separated botnet C&C systems in to three groups. Some botnets use centralized C&C centers where the botmaster can control all bots directly. Other botnets use a peer to peer C&C system, where bots communicate with each other. In addition to receiving commands, many botnets must communicate information back to the botmaster, especially if their goal is to obtain the private information of the user whose computer hosts the botnet software. Finally, some botnets use a hybrid approach.

In the centralized model, communication between the bots and botmaster is often done using an IRC channel or over HTTP. This was the original botnet C&C model used. Because the system is centralized, the C&C center acts as a single point of failure for the botnet. When using IRC, the botmaster will create an IRC channel on their server and the bots will then connect to the server to communicate with the botmaster. From this IRC channel, the botmaster could command all bots to initiate a DDoS attack on an enemy. If the botnet communicates over HTTP, it gains the advantage that HTTP traffic is not suspicious in general, because it is the protocol used for web traffic [22]. A centralized approach may also maintain several C&C centers to improve communications and prevent having a single point of failure, as shown in figure 1.

Although most botnets use IRC or HTTP to communicate directly, the botnet designed for this paper will communicate over a social network. This concept has been discussed in some previous work, for example in [19], a method of C&C over Twitter is discussed, however the commands are sent directly as the content of the tweets instead of by using a covert channel. A botnet has been designed to use steganography over a online social network [12], but it uses image steganography to embed messages in the images posted normally by the victim. It requires that other bots in the botnet be on computers socially connected to the victim via the online social network.

3. METHODOLOGY

3.1 Twitter Covert Channel

3.1.1 The Stego System

To perform the botnet command and control communication, a covert channel (stego system) that communicates using the Twitter social network has been developed. This covert channel is similar to Desoky's [4] noiseless steganography and utilizes the cover generation paradigm, however there are some differences. Even in the noiseless steganography systems, the secret messages are usually embedded in to the actual data of the cover objects. For example, in graph steganography, the plotted data contains the secret message. In this system, where the cover objects are tweets, the secret message is not contained in the data of the tweet (the text), instead it is contained in the *metadata* of the tweet (the length). Metadata refers to "data about data." All data has some metadata associated with it, but this metadata is not explitictly stored. It is inferred from the existing data. The tweet's data is the text. The tweet also has metadata such as the time it was posted, the user account, and the length of the text posted. Additional metadata could include the letter frequencies of the posted text or the number of spaces in the text.

Because this system differs from existing steganographic systems, we will define the parts of this system as follows:

1. The set of possible cover messages, $\mathcal{X}^*$, is the set of possible tweets, which is the set of messages of up to 140 UTF-8[2] characters.

2. The set of possible secret messages, $\mathcal{M}$, can be defined as Σ^*, where the Σ notation is taken from the formal languages domain, and refers to an alphabet of symbols, where the symbols can be arbitrarily defined. For example, one implementation may use $\Sigma = \{a, b, c, \ldots, z\}$ (the English alphabet).

3. The set of possible keys, $\mathcal{K}$, is the set of numbers that can be valid pseudo-random keys for the implementation. In our case, the implementation uses the Java programming language's `java.util.Random`[3] class, which uses 48 bit keys.

4. The `Embed` and `Generate` functions are combined. In our implementation we generate reasonable cover messages to have appropriate metadata that contains the secret message.

5. The `Extract` function will also require a `Decode` step, described below.

6. For convenience, we will also use the following notation for the set of natural numbers up to 140: $\mathbb{N}_{140} = \{1, 2, \ldots, 140\}$. Similarly, if we use $\mathbb{N}_n$, it means the natural numbers from 1 to n. Unless otherwise stated, we assume $0 \notin \mathbb{N}$.

The overall system is shown in figure 2, where the numbered components were implemented for the channel.

The secret message is embedded by utilizing the *length* of the posted tweets, by character count. Because a tweet can have a length of up to 140 characters, the length value can store just over

[2]https://dev.twitter.com/docs/counting-characters

[3]http://docs.oracle.com/javase/8/docs/api/java/util/Random.html

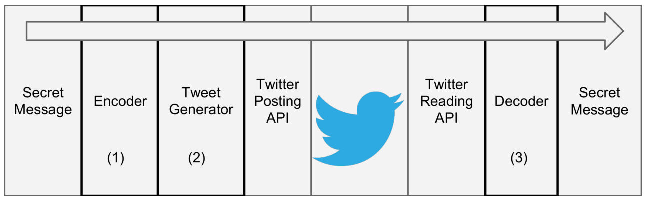


Figure 2: Overview of Twitter covert channel, where the numbered components were implemented for the channel.

$$\begin{tikzcd}
m_i \arrow[r] & \boxed{\texttt{Encode}} \arrow[d, "n \in \mathbb{N}_{140}"'] \arrow[rr, phantom] & & \boxed{\texttt{Decode}} \arrow[r] & m_i \\
& \boxed{\texttt{Generate}} \arrow[rr, "x^{(m_i)}"] & & \boxed{\texttt{Extract}} \arrow[u, "n \in \mathbb{N}_{140}"'] &
\end{tikzcd}$$

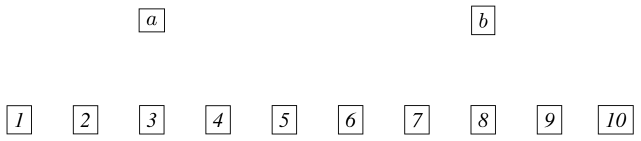
$$k \to \texttt{Encode}, \quad k \to \texttt{Decode}$$

Figure 3: Modified stego system diagram for Twitter covert channel.

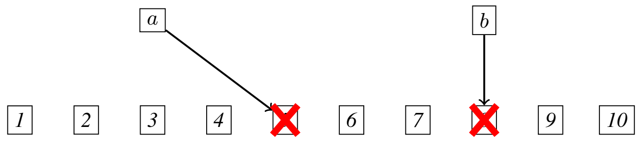
7 bits of information per tweet. However, embedding 7 bits of information per tweet is not reasonable in practice. Certain length tweets rarely appear on Twitter so seeing, for example, many tweets of length one or two on a single account would be suspicious. To solve these problems, we can use a one-to-many encoding technique to hide information in the tweet lengths. We will modify the normal stego system definition to include the following functions: `Encode`: $\mathcal{M} \times \mathcal{K} \to \mathbb{N}_{140}$ and `Decode`: $\mathbb{N}_{140} \times \mathcal{K} \to \mathcal{M}$.

The modified stego system definition is shown in figure 3. A message $m \in \mathcal{M}$ is broken up in to symbols of Σ: $m_1 m_2 \ldots m_a$. Each symbol m_i is mapped to one of several possible values using `Encode` along with the appropriate key k to generate $n \in \mathbb{N}_{140}$, the appropriate tweet length value to use for this piece of the message. This value is passed to `Generate`, which generates a plausible cover message $x^{(m_i)} \in \mathcal{X}^*$ to be posted to Twitter. The `Extract` function reads the posted tweets and calculates the length n of each tweet. This value is passed to `Decode` along with the original key k to reconstruct the original message m piece by piece. This design assumes that $|\Sigma| \leq 140$, and in fact, smaller alphabets should improve the security of the channel. A smaller alphabet allows mapping each symbol to more possible length values, so repetitions of each length value are less likely. We will now present a simplified example to show the process of the stego system.

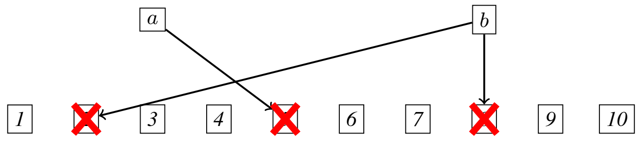
EXAMPLE 3.1 (ENCODING TABLE GENERATION). *In this example, we will show the process for generating the encoding table. Instead of using $\mathbb{N}_{140}$ (all possible length values), we will use a reduced output alphabet of $\mathbb{N}_{10}$ (1, 2, ..., 10) with an equal distribution. For the input alphabet, we will use $\Sigma = \{a, b\}$ with an equal distribution.*



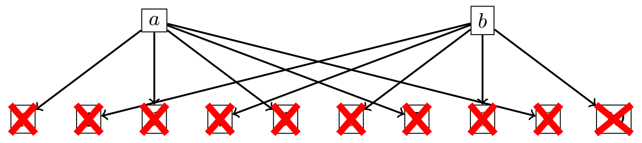
First, choose one element of the output alphabet for each element of Σ. This guarantees that at least one output symbol will *be mapped to each input symbol. Remove the chosen values of the output alphabet as options for future choices.*



Now, choose one element from both sets probabilistically based on the weights.



Continue this process until all elements of the output alphabet have been used.



After all elements from the output alphabet have been used, the encoding table is composed of all of the choices made:

Symbol	*Possible Length Values*
a	*1, 3, 5, 7, 9*
b	*2, 4, 6, 8, 10*

EXAMPLE 3.2 (SIMPLE MESSAGE ENCODING EXAMPLE). *In this example, we will use tweet lengths up to 10, i.e. we will use* $\mathbb{N}_{10} = \{1, 2, \ldots, 10\}$ *instead of* $\mathbb{N}_{140}$. *We will use* $\Sigma = \{a, b\}$ *and* $\mathcal{X}^* = \{x\}^+$, *i.e. secret messages will be composed of combinatiosn of* a *and* b *and cover messages will be strings of* x.

Suppose we want to send the secret message m = *abba using the simple* `Encode` *map from example 3.1. First, the message is broken in to the sequence of symbols* a, b, b, a. *The* `Encode` *function will then map each symbol to a possible length value, e.g. 3, 6, 2, 3. Note that because each input symbol from* Σ *can map to more than one length value from* $\mathbb{N}_{10}$, *the same symbol may or may not be mapped to the same length value in any given message. The* `Generate` *function will then create cover messages that match these length values from the set of possible cover messages* $\mathcal{X}^*$: $xxx, xxxxxx, xx, xxx$. *Each cover message would then be posted to a Twitter account in the order of the original secret message. The* `Extract` *function on the recipient's side would then take the tweets in the posted (chronological) order, returning the length values 3, 6, 2, 3. The* `Decode` *function can then apply the same map as the* `Encode` *function and reconstruct the original message abba.*

This system is generic in that it can be used with many possible input alphabets, e.g. the English alphabet or arbitrary half-byte values (0x0, 0x1, ..., 0xF). The English alphabet allows sending simple messages. The half-byte alphabet allows sending arbitrary binary data by splitting each byte of the input data in half and sending each half as one symbol of the message. It is impossible to send an entire byte in one message using this system because the maximum tweet length is only 140 characters. We chose half-bytes because it is easy to deconstruct and reconstruct the original bytes and because it is a relatively small alphabet with only 16 symbols, so it is possible to map each input symbol to almost 10 different tweet length values. To obtain symbol frequencies for the half-byte values, it would be best to empirically sample the types of data being sent across the channel because, in general, each value would likely have an equal weight. If the specific type of data being sent is biased toward certain byte values, that should be considered when weighting the alphabet.

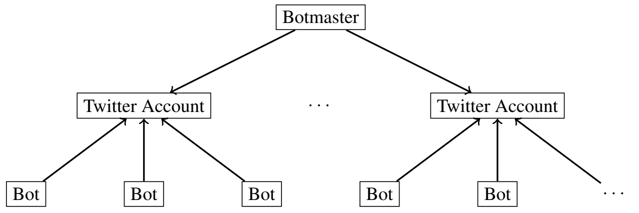


Figure 4: The botnet C&C diagram for the system.

The botnet command and control diagram for this system resembles the diagram for a centralized botnet, as shown in figure 4. A botmaster controls one or more Twitter accounts that have tweets containing the commands and the bots read from these accounts.

3.1.2 The Tweet Generator

The `Generate` function is one of the most challenging aspects of this type of stego system. As discussed in [1], generating appropriate and plausible cover messages for a stego system is a nontrivial problem. In this system, the generator must be capable of generating messages that can convince a reader of the Twitter account page that they are viewing regular tweets. This component has the largest impact on the detectability of the channel. In essence, the generator must pass a simplified Turing test. Twitter bots are not a new phenomenon, and in fact several bots were created that successfully convinced other users that they were real people [2]. Additionally, chat bots exist, such as Cleverbot[4] which are reasonably successful [3]. However, aside from competent English skills, the generator must utilize the "language of Twitter" that consists of many retweets[5] and hashtags[6]. We consider a strong generator out of the scope of this work, but we leverage the collected Twitter data to create a Twitter language model based on tweet contents that can be used to generate new tweets.

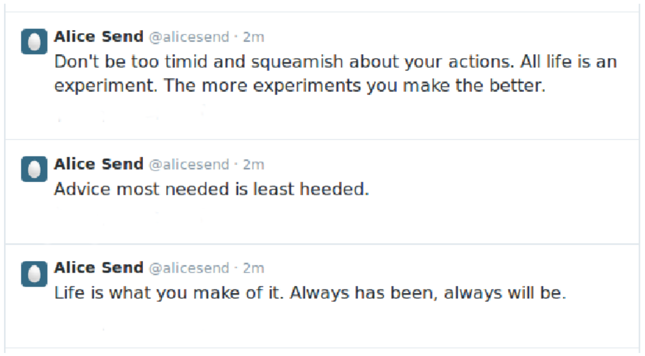
3.1.3 Posting to Twitter

Along with the `Encode`, `Generate`, `Extract`, and `Decode` functions, we need a system that can post to Twitter. This is easy to do for testing purposes thanks to Twitter's official API[7] and a third party Java library, Twitter4J[8]. For a real botnet scenario, the implementer would likely write their own system that uses raw HTTP requests because the Twitter API requires authentication of every call, detects the posting method, and limits the number of posts allowed for each account. However, because this would violate Twitter's terms of use, we will only post tweets using the official API and abide by all limitations for testing. Now that the rest of the

[4]http://www.cleverbot.com/
[5]https://support.twitter.com/articles/77606-faqs-about-retweets-rt
[6]https://support.twitter.com/articles/49309-using-hashtags-on-twitter
[7]https://dev.twitter.com/docs/api
[8]http://twitter4j.org

Symbol	Weight	Encoding
A	14810	32, 16, 19, 131, 84, 37, 106, 140, 76, 111
B	2715	105, 138, 67
C	4943	75, 36, 125, 46, 62
F	4200	17, 122, 61, 87
...	...	...
O	14003	35, 121, 43, 107, 92, 12
...	...	...

Table 1: Sample encoding map example for a few English alphabets.


Alice Send @alicesend · 2m
Don't be too timid and squeamish about your actions. All life is an experiment. The more experiments you make the better.

Alice Send @alicesend · 2m
Advice most needed is least heeded.

Alice Send @alicesend · 2m
Life is what you make of it. Always has been, always will be.

Figure 5: Example showing posted tweets for secret message FOO.

components have been explained, a more complete example will be presented.

EXAMPLE 3.3 (COMPLETE EXAMPLE). *In this example, we will use the full range of tweet lengths* $\mathbb{N}_{140}$. *We will use* $\Sigma = \{A, B, \ldots, Z\}$ *(the English alphabet), and* $\mathcal{X}^*$ *as a pre-constructed list of various proverbs and phrases of lengths ranging from one to 140. The* `Generate` *function will lookup an appropriate phrase for each length message provided by the* `Encode` *function. Table 1 shows a portion of a generated encoding map from English letters to tweet lengths. The weights shown in the second column are taken as letter frequencies[9]. Those weights were used to decide the number of entries for each letter in the third column.*

Suppose we want to send the message FOO. *First, the message is separated in to the sequence of symbols* F, O, O. *Each is passed to the* `Encode` *function, which chooses appropriate lengths, e.g. 61, 35, 121. The* `Generate` *function then generates tweets and they are posted to Twitter, as shown in figure 5. The figure should be read from bottom up, because the newer messages are posted on top of the older messages. The account shown is a test account created for this work. The recipient then reads these tweets, obtains the lengths, then uses* `Decode` *with the same table as was used for the* `Encode` *process to get the original message.*

3.1.4 The Botnet Command and Control Language

The stego system described in section 3.1.1 can be used with an arbitrary input alphabet as long as its size is not larger than the tweet length range (up to 140 characters), so for botnet command and control we have developed a language that can be mapped to tweet lengths and interpreted to execute botnet commands. We have included some common botnet commands as described in [19]. The weights were decided somewhat arbitrarily, because in a real scenario the botmaster would tailor the weights based on which commands they believe that they are likely to send most often. In this case, We are assuming the byte values (indices 0 to 15) are more likely, because for some commands arguments must be sent using these. We don't assume any single command is more likely than another.

Index	Weight	Description
0	25	Literal hex value 0
1	25	Literal hex value 1
2	25	Literal hex value 2
...	...	...
16	5	Take screenshot
17	5	Shutdown computer
18	5	Reboot computer
19	5	Perform DoS attack to IPv4 address in next 4 bytes sent
20	5	Stop DoS attack
21	5	Download and execute file from address in next k bytes (until delimiter)
22	1	Message delimiter

Table 2: Botnet command and control language for use with the stego system.

3.1.5 Username Generation using Markov Chains

It is necessary to have a system for generating user names from an initial seed so that if the original botmaster account is blocked, they can start a new account and the bots can also generate the new account name and begin reading from it. To do this, we employ Markov chains [14]. The Markov chain being used can generate strings of letters, numbers, and underscores and is trained using an existing corpus of such text (in our case, a collection of verified Twitter usernames).

To use this Markov chain for generating a sequence of usernames, both the bot and botmaster must have the same initial seed. Using this seed and the same type of pseudorandom number generator, the Markov chains will generate the same sequences as long as both bot and botmaster follow the same procedure. First, they need to use the random number generator to choose a user name length. Second, they use the Markov chain with this seed to choose a starting character. Finally, they generate enough symbols to fit the length chosen. If one performs an action out of order, it will affect the sequences generated after that action. For example, if the botmaster chooses the length first while the bot chooses the starting character first, the sequence generated from the random number generator will cause each to generate a potentially different length and starting character.

4. EVALUATION OF RESULTS

4.1 Tweet Collection

Several portions of this paper required collecting data from Twitter. To determine the posting rate of tweets of each length, it was necessary to collect tweets from real accounts. Therefore, data from verified[10] Twitter accounts was collected. A verified account

[9]http://www.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html

[10]https://support.twitter.com/articles/119135-faqs-about-verified-accounts

is an account that Twitter has manually verified to be a specific person or brand. By using verified accounts, this prevents obtaining data from other bots or fake accounts. However, it has been noted that verified accounts may not be a perfect representation of the average Twitter user, who is not generally a brand or celebrity. User information stored includes the username and user ID, a unique integer that Twitter stores for each user. A total of 54,114 users were collected. Additionally, for creating the Markov chain, tweet content was parsed using a regular expression from the collected tweets to find username references in the tweets. In a tweet, usernames are preceded by an @ symbol.

From the collected user IDs, tweet content, length, unique identifier, and posting time were collected. Because of the number of collected users and the number of tweets posted by each user, during the data collection we only obtained tweets from 3,709 users. However, this totaled 7,345,681 tweets. This data collection was done automatically from a list of verified users that was obtained from Twitter. Twitter has a special account with username *verified* that will follow all verified accounts. Therefore, we were able to search for all accounts followed by that special account using the Twitter API and then begin obtaining tweets posted by each of those accounts. Not all accounts post in English, some collected data is in other languages including Portuguese, Spanish, Japanse, and Arabic among others. In order to remove these languages, we used the third-party Java library NGramJ[11], which performs language recognition using n-grams. An n-gram is a sequence of symbols of length n. For example, in English a common bigram (2-gram) is *th*. This library ranked each tweet according to which language it most resenbled. We kept only the tweets where the highest ranking language is English. This left us with 5,461,009 tweets out of the original 7,345,681. However, this method is not perfect. Tweets contain some non-typical English characters in hashtags, names, misspellings, URLs, etc. Because of this, the n-gram analysis likely had some false positives and false negatives. For example, one tweet had the text "Vancouver, 9/25", but the n-gram model marked it as French. After this n-gram analysis, tweets were further restricted by checking the character values in each tweet. If a tweet contains too many non-ASCII characters, then it is not likely to be English. In this case, we removed all tweets that were 10% or more non-ASCII characters. This allows up to approximately 14 non-ASCII characters in a full length tweet. The final tweet count is 5,011,973.

4.2 Stego System Evaluation

4.2.1 Emulab Performance and Reliability Experiment

Emulab [15] is a network testbed and software system designed for testing networked systems. It allows experimenters to request a set of physical machines, or nodes, that are configured in a specific network configuration as defined by a script file supplied to the Emulab website.

The experiment for this paper was performed by generating symbols from a test alphabet (the English alphabet) and posting generated tweets using a random text generator and a constructed encoding map. The input symbols were chosen probabilistically with the weight associated with their letter frequencies. The "botmaster" acted from a desktop computer outside of the Emulab environment posting the tweets while the "bots" acted from inside an Emulab experimental setup. There were five bots in the experiment running Emulab's `UBUNTU12-64-STD` operating system image. The botmaster generated a new symbol and posted the corresponding tweet

[11] http://ngramj.sourceforge.net/index.html

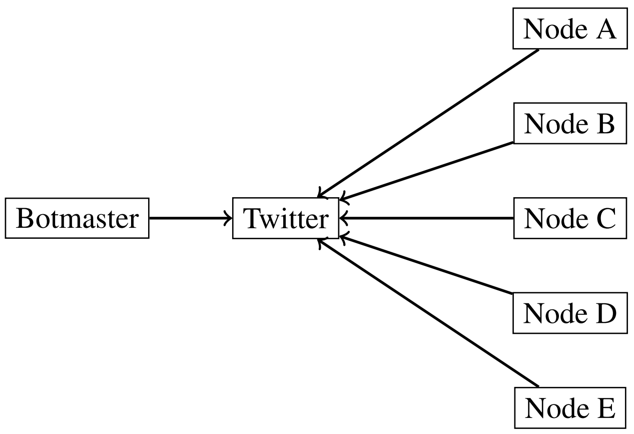


Figure 6: Diagram for the Emulab experiment network layout.

Node A	Node B	Node C	Node D	Node E
5.725	5.644	5.600	5.912	5.888

Table 3: Average read time in seconds for each node.

every 30 seconds while the bots performed an HTTP request to the correct Twitter account every 10 seconds checking for new tweets. Each time the botmaster posted, it is recorded to a log file. Each time the bots read a tweet, they also recorded to a log file. Afterwards, the logs were collected to compare the post time with the retrieval time and also to match each original input symbol with the decoded symbols from the bots. Due to a time zone difference between the botmaster machine and the bots in the Emulab setup, the original time stamps from the bots appeared one hour later, so one hour was subtracted from their times when comparing the difference in posting and reading time between the botmaster and bots. Figure 6 shows the network configuration for the experiment.

In the experiment, 100% of input symbols were correctly decoded by the bots except for a small set that were off by one. After examining the data it was determined that in these cases, the tweets being posted were generated with a trailing space that was then trimmed by Twitter while posting. If the tweets had been generated without spaces, this would not have occurred and so these cases were dropped from the results. The average read time in seconds for each node are shown in table 3. Each node averaged just over five seconds from the botmaster's post to the bot's read. This is likely due to the synchronization issues of the botmaster's posting and the bot's sleep time between reads. A total of 305 tweets were posted for this test and 10 of them were dropped for having trailing whitespace. With an average transmission time of less than six seconds, the overall transmission rate is up to 10,800 bytes per day.

4.2.2 Capacity

There are three major criteria for stego system evaluation: capacity, steganographic security, and robustness [1].

At the most basic level, the devised stego system can be used to transmit at most seven bits of information per tweet because a tweet can have a length of at most 140. If eight bits were to be transmitted, there would be 256 different values. With seven bits, there are 128 different values, so each value can be sent with a different length tweet. Therefore, we can state the maximum capacity as seven bits per tweet. Tweets are posted to Twitter using UTF-8, which is a variable length character encoding scheme that is a superset of the ASCII characters. UTF-8 characters range from one

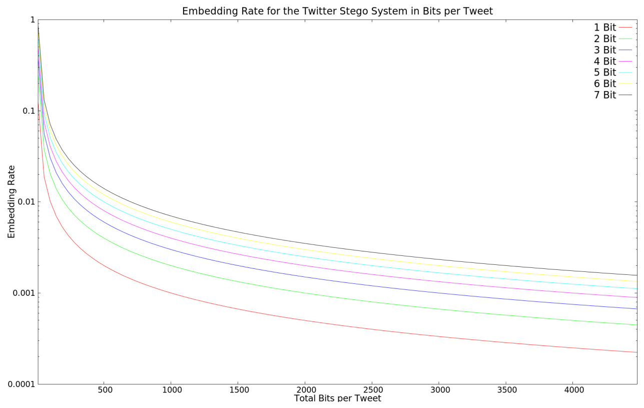


Figure 7: Possible embedding rates for the stego system in bits on a logarithmic scale.

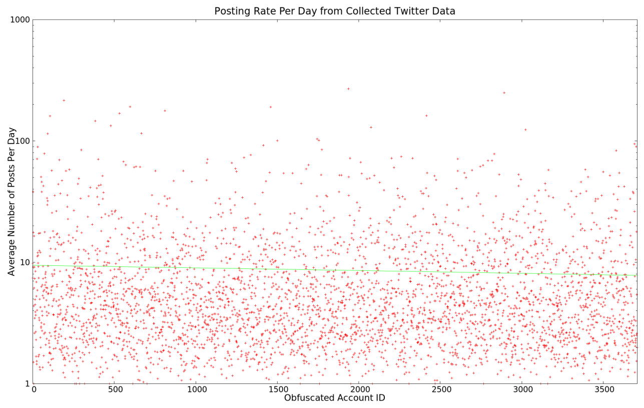


Figure 8: Average posting rate in tweets per day on a logarithmic scale. The green line indicates the mean.

to four bytes [21]. Therefore, the embedding rate will vary depending not only on the length of the tweet in characters, but also on how many characters per byte are being used. The total number of bytes in a tweet is 560, so the total number of bits is then 4480. The possible embedding rates are shown in figure 7.

Additionally, we must consider how frequently tweets can be posted. From the collected Twitter data (see section 4.1), the time stamp of the tweets was also collected. For each unique user, the average number of posts per day was then calculated from this data. This data is shown in figure 8. In total, the averege daily posting rate is 8.621 tweets per day. Therefore, if the user of the stego system is trying to match real Twitter user posting rates, they cannot send more than approximately 60 bits of data per day. If using the system for botnet command and control, this will allow the botmaster to post a small number of commands per day. The botmaster does also have the choice to exceed this value, but then risks a higher detection rate. Because the data shows the average number of tweets posted per day *per account*, that means there are many accounts that do post more tweets per day. In this data, there are several accounts that post on average more than one hundred tweets per day.

4.2.3 Steganographic Security

In this stego system we are assuming a *passive warden* model. In a passive warden model, an adversary can view each message but cannot modify them. The warden must solve the decision problem:

Length	Phrase Generator	Database Generator
10	Lunch time	no ragrets
11	Hello World	I'm so.....
12	Good Morning	@J_Baxt16 14

Table 4: Sample tweets from the database based tweet generators.

does this tweet contain a secret message? Because our implementation does not embed any data in to the tweets, most techniques that would be used on a normal stego system are not sufficient. The posted tweets appear identical to any other tweet from a textual perspective. However, the tweet generation method is a large determiner of detectability. It is possible to create the tweets manually, but if the user wants to send many messages using the stego system this will be cumbersome.

In order to automate the process, a Twitter bot program can be used to create the tweets. An ideal generator would be a sophisticated Twitter bot that can convince other users that it is human. This is similar to passing a Turing test with the Twitter bot. If an account is suspected of being a Twitter bot, it does not mean that the communication has been detected, however it will cause suspicion. The adversary would have to recognize that the account is being used to pass secret messages and that the secret messages are done using the lengths of the tweets. The adversary would likely assume that the text somehow contains the secret messages. If we follow Kerckhoffs' principle [10], then we must assume the adversary knows that that the stego system passes messages by tweet lengths. The two factors that must then be determined are then (i) the account being used for transmission, and (ii) which tweets being posted contain the secret message.

If the adversary has no knowledge of which account is being used, it will be exceptionally difficult to find. The Twitter website states that there are now 271 million active monthly users and over 500 million daily tweets posted[12] as of August, 2014. Because the tweets have no distinguishing factors in general, an adversary cannot easily search for the account by tweet content. If the adversary understands the tweet generator being used, they may be able to search for the account by the content. So far, the system has been discussed in a way that implies that all tweets posted on the account are part of the secret messages, however it is possible to extend an input alphabet to leave space for ignored tweet lengths. The Twitter bot could then post these between tweets that contain actual parts of the secret messages.

The generator that we used is a database generator, which looks up tweets of the appropriate length from an existing database. The database may be populated by collecting real tweets from other accounts or from collecting text from other sources. Two of these database based generators were created. The first uses a small set of common phrases and for longer tweets some proverbs were collected from the Internet. The second uses the Twitter data previous collected (see section 4.1). Samples from each of these generators are shown in table 4.

4.2.4 Network Packet Analysis

In addition to analyzing the stego system content on Twitter, a small experiment was performed using Wireshark[13] to monitor packet contents while accessing Twitter. Twitter allows connecting through HTTPS to access user pages, so when using this system it is best to always access with HTTPS. In this experiment, the `wget` command was run twice. First, it was run to access another known

[12]https://about.twitter.com/company

[13]https://www.wireshark.org/

Twitter account, @*BarackObama*. Then, it was run to access the test account used for this work, @*alicesend*. In both cases, the HTML content of the user's page was downloaded and Wireshark monitored all traffic between the two hosts. After searching the wireshark packet contents, there is no noticable difference in the network traffic. Searches were conducted for identifying strings such as "alice" and "Obama" but all application data was encrypted using TLS 1.2 according to Wireshark. Therefore, this traffic information is insufficient for determining which accounts are being viewed by the source host. To a network observer, it simply appears as regular Twitter traffic, which is generally common due to Twitter's popularity.

4.2.5 Robustness

Robustness is based on extracting the secret messages from the cover objects [1]. As shown in the Emulab experiment in section 4.2.2, aside from some anomalous entries, every bot decoded the appropriate input symbols perfectly. This assumes a passive warden model where no one has tampered with the data in transit. In an active warden scenario, there are two possibilities: (i) Twitter is modifying the tweets as they are posted or (ii) an adversary has taken control of the botmaster's Twitter account.

The first scenario is extremely unlikely. Twitter does perform some modification as described in section 4.2.2 where trailing whitespace was removed before the tweets were posted. However, this modification is well defined and is not intended to modify the contents of the secret or cover messages. It can be handled by properly implementing the tweet generator. The second condition would be devastating for the system. In most of the paper, a passive warden model was assumed because it was assumed that the botmaster could maintain control of their Twitter account. It is possible that the account is taken down if Twitter discovers that it is a bot or that some other party obtains control of the account.

Aside from the steganographic robustness, the robustness of the system in general is largely dependent on Twitter's infrastructure, which is one of the advantages of using such a service as the communication medium. In previous years, Twitter has suffered with outages, however it has recently improved significantly. Because of Twitter's business model, downtime is very costly for many corporations, organizations, and individuals that rely on Twitter for marketing[14], giving them great incentive to ensure that service is maintained.

4.3 Username Generation Analysis

4.3.1 Scoring Names Based on the Generated Markov Chain

One of the components in the botnet command and control system is a method of generating plausible Twitter user names. As described in section 3.1.5, Markov chains were used to generate such user names. In order to analyze the usernames generated from these Markov chains, two experiments were performed. First, a probability measure was calculated on names based on the Markov chain. We calculate the probability that a given string would have been generated by the Markov chain. Let N be a name consisting of the sequence of characters $n_1 n_2 \dots n_k$. The probability, $P(N)$, of choosing N from the Markov chain is then

$$P(N) = P(n_1) \times P(n_2 \mid n_1) \times \cdots \times P(n_k \mid n_{k-1}). \quad (1)$$

Because Markov chains are "memoryless" in that the next state is entirely dependent on the current state, it is not necessary to factor

[14]http://www.cnet.com/news/the-cost-of-twitter-downtime/

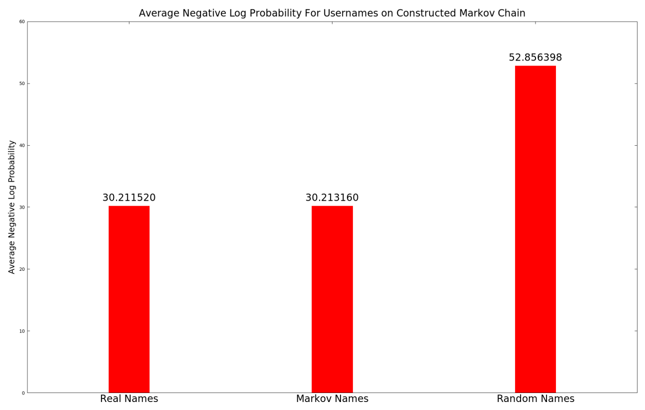


Figure 9: Average negative log probability score for usernames based on constructed Markov chain.

Real Usernames		Markov Chain Usernames		Random Text Usernames	
Name	Score	Name	Score	Name	Score
davepeck	20.189543	Coccpthe	24.123132	wBc3HLqy	44.472027
nytimes	19.178421	JorayLa	17.191901	gzQbhCT	33.078578
focuspolitik	31.618525	beteckucovao	30.477696	KvJhlRwF4G45	67.152348
MarsHill	20.768233	Diajan_m	20.492829	PthGonXE	30.381789
Scobleizer	26.922820	Boumezzost	25.873167	vLdHuXVqDO	56.543208
warrenellis	24.335242	shltirreaha	30.877799	4ok1MwHzWD1	60.301202
redjumpsuit	32.200608	SEMarannesi	25.831416	QzFS7n4StQt	58.562229
joshspear	23.335631	McolitePa	23.613131	ZhZ1B28vX	46.774850
FUELTV	21.209548	tudwpi	20.180779	hFIn6O	23.278177
fredwilson	27.280397	MarassttyM	21.652989	__hKc4vhHi	50.487769

Table 5: Sample names from each category and their scores.

in previous choices in the probability calculation. The result of equation 1 becomes small very quickly because of the number of possibilities, so the answer is stored in log-probability space. That is, the actual calculation is as follows:

$$\log P(N) = \log P(n_1) + \log P(n_2 \mid n_1) + \cdots + \log P(n_k \mid n_{k-1}). \quad (2)$$

The score of a name is then calculated as the negative log-probability of the name. Therefore, a lower score is considered a better name according to the Markov chain. The Markov chain was constructed based on real name statistics, so if a name has a higher probability of occurring according to the Markov chain, it should appear to be a plausible name.

An experiment was run that computed these name scores for all 1.5 million names collected from Twitter along with the same number of names generated by the Markov chain, each name having the same length as one of the names from the original set and also a set of the same number of equally-lengthed random text names. The average scores calculated from this experiment are presented in figure 9. Some sample names from each category are presented in table 5. Keep in mind that a lower score correlates to a *higher* probability of being generated by the Markov chain.

These results show that names generated by the Markov chain are statistically identical (within 0.002) to the real usernames used to create the Markov chain. It is unlikely that an automated system could distinguish between them. Random text, however, will not work in this case. Randomly generated names appear significantly different than real user names.

4.3.2 Scoring Names using Human Analysis with Mechanical Turk

In addition to the statistical analysis, an experiment was performed using Amazon's Mechanical Turk[15] system. The Mechanical Turk system is a way to connect experimenters that need a human to evaluate something with willing participants that can perform the evaluation for a small fee per task.

For our experiment, we chose the sentiment analysis template. This asks users to choose from one of five choices and provides some more specific instructions about each choice. Each worker was shown one user name that is either a real user name from Twitter, a name generated by our Markov chain, or a name that is just random text. The worker did not know from which group the name appeared.

Essentially, we asked them to score their confidence in recognizing whether or not the name was real. If they were confident that it was fake, they should have answered in the negative. If they were confident that it was real, they should have answered in the positive. If they were not sure, they should have answered neutral. The negative responses are scored with either -1 or -2 depending on their degree of confidence and the positive responses are scored with either 1 or 2 depending on their confidence. A neutral response scores 0. Each worker was compensated \$0.10 for each name they scored. It should be noted that this experiment is exempt from IRB approval because no identifying information about any participants is collected. In fact, Mechanical Turk does not provide a way for requesters to identify workers. Instead, each worker has a pseudonymous user ID number. Participants were only asked to respond to a survey of their opinions about the validity of the user names, not even the user ID was collected after the experiment was performed.

In total, 50 names were chosen from each category and each name was shown to five different workers. The aggregated results are shown in figures 10, 11, and 12.

These results show some differences compared with the automated statistical analysis above. Random names, however, appear obviously fake to human examiners. Almost all responses for the random names were negative. It does appear, though, that humans are not confident in recognizing real names either, with a plurality of results being zero and some negative. They seem to have some inclination toward recognizing fake names generated by the Markov chain, with more negative answers than the real user names, but very few considered themselves very confident (a score of -2).

The results from both the statistical analysis and the Mechanical Turk survey indicate that the method being used for username generation is sufficient to create usernames that would not appear abnormal to either an automated name analysis tool or to human users viewing the page on Twitter. Therefore, this method can be used as the username generation method for a botnet that must create a new account and switch communications to it for some reason, e.g. if Twitter blocks the previous account. By starting with a common seed, each bot can generate the appropriate new name and connect to the new account.

5. RELATED WORK

Stegobot [12] is a botnet designed to communicate using social networks (specifically Facebook) and image stegangraphy. The authors design two separated types of message: bot commands and bot cargo. The bot commands are messages from the botmaster to the bots instructing them. The bot cargo are messages from the bots back to the botmaster containing stolen information.

Stegobot [12] uses a distributed, peer to peer communication channel and does not generate its own cover messages. Instead, it uses the image files that the victims are already uploading to the social network as the cover messages, embedding the secret messages within. The botnet software intercepts the images being posted to embed the message before it is sent to Facebook. Stegobot uses the existing network of relationships for each victim as the communication channel. In their experiments, the authors used a set of 116 images. One of the difficulties of this technique is the automatic image manipulation performed by Facebook as the images are uploaded. This can tamper with the content of the embedded message, requiring a highly robust stego system.

Natarajan et al. [13] designed a detection scheme for Stegobot that uses the information entropy of the image files that are acting as the cover objects. Their detection technique achieved average detection rates exceeding 70% in their experiments for several different image steganography methods.

A similar work by Singh et al. [19] uses Twitter for botnet C&C, but does not apply steganography to hide the communications. Instead, the commands are posted directly to the Twitter account. This allows the botmaster to leverage the benefits of social networks for botnet C&C, but they used communication methods that will likely appear highly suspicious to any viewers.

SocialClymene [6] is a detection scheme designed specifically for detecting stego-based botnet command and control methods using social networks, however it is designed for image steganography such as stegobot. CatchSync [9] is another detection technique that looks at connectivity of nodes in a directed graph to find suspicious nodes. In the case of Twitter, connectivity is determined by which accounts follow other accounts.

Sebastian et al. [18] have created a similar mechanism for botnet command and control using encrypted tweets. This method mixes irrelevant sentences among tweets that contain botnet commands. Their command tweets follow the formula of `#keyword command`, where the value of `command` would be encrypted. While this method can hide the commands being issued, it does not conceal the existence of the commands. Each command follows the same formula and can be differentiated from other tweets posted on the botmaster's account. Additionally, no mechanism is described for recovering if the botmaster's account has been closed due to detection of malicious activity.

6. CONCLUSION

In this paper, we have demonstrated a general-purpose stego system that allows secret communication through the Twitter social network using only metadata for communication. We have showed that this system can be used for botnet command and control through the development of a Twitter stego system and a specialized botnet C&C language. We have discussed the performance evaluation of the proposed system using Emulab, and usability study using Amazon MTurk. We have also discussed how the vast userbase and large scale of Twitter facilitates ample steganograpic security. By demonstrating how a botmaster might perform such communication using online social networks, our work provides the basis to detect and prevent emerging botnet activities.

There are other possible steganographic techniques that can be applied using Twitter. For example, Twitter allows posting images along with tweets[16] and there are many existing image steganography techniques [8] that are often used for watermarking, but can also be used for communication. The existing system can also be used on other social network websites such as Facebook, but it may be necessary to collect data from these websites when deciding on

[15]https://www.mturk.com

[16]https://support.twitter.com/articles/20156423-posting-photos-on-twitter

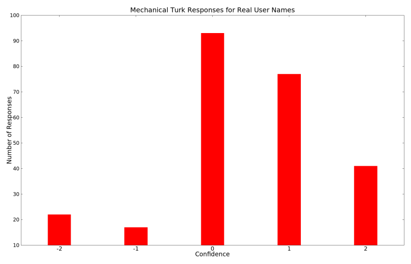


Figure 10: Aggregate scores for real usernames from the Mechanical Turk experiment.

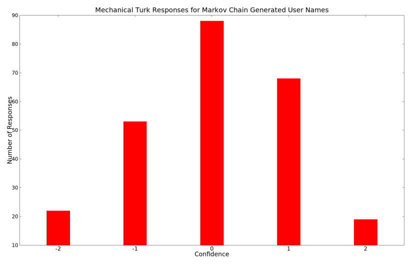


Figure 11: Aggregate scores for Markov chain generated usernames from the Mechanical Turk experiment.

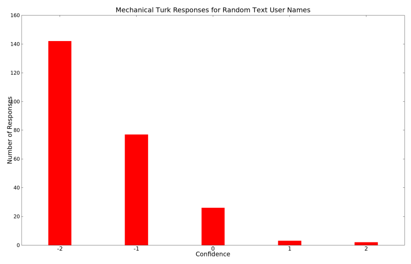


Figure 12: Aggregate scores for random text usernames from the Mechanical Turk experiment.

the message length distribution. Unlike Twitter, these other websites generally allow much longer posts, so the system could take advantage of the increased variation.

It is possible to use this system for key exchange for other existing stego systems. It is necessary in steganographic key exchange to have a stego system that can be used to transmit the key, otherwise an adversary can detect the communication of the keys. Because this system has a relatively low information bandwidth, it may be well suited for key exchange that does not require a significant amount of information. This concept is applied in cryptography where a public key algorithm such as RSA is used to send a key for a symmetric algorithm such as AES because AES can achieve better encryption and decryption performance than RSA, although it is technically possible to send all of the communications using RSA.

7. REFERENCES

[1] Rainer Böhme. *Advanced Statistical Steganalysis*. Springer, August 2010.

[2] Zack Coburn and Greg Marra. Realboy: believable twitter bots. http://ca.olin.edu/2008/realboy/index.html.

[3] Franck Dernoncourt. Designing an intelligent dialogue system for serious games, 2012.

[4] Abdelrahman Desoky. *Noiseless Steganography: The Key to Covert Communications*. CRC Press, February 2012.

[5] Donald L. Evans, Phillip J. Bond, and Arden L. Bement. Federal information processing standards publication: Standards for security categorization of federal information and information systems, February 2004.

[6] M. Ghanadi and M. Abadi. Socialclymene: A negative reputation system for covert botnet detection in social networks. In *Telecommunications (IST), 2014 7th International Symposium on*, pages 954–960, Sept 2014.

[7] D. Gollmann. *Computer Security*. Wiley, 3 edition, 2012.

[8] F. Hartung and M. Kutter. Multimedia watermarking techniques. *Proceedings of the IEEE*, 87(7):1079–1107, Jul 1999.

[9] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. Catchsync: Catching synchronized behavior in large directed graphs. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 941–950, New York, NY, USA, 2014. ACM.

[10] Auguste Kerckhoffs. La cryptographie militaire. *Journal des Sciences Militaires*, 5(38):161–191, 1883.

[11] Jan Kok and Bernhard Kurz. Analysis of the botnet ecosystem. In *Telecommunication, Media and Internet Techno-Economics (CTTE), 10th Conference of*, pages 1–10, May 2011.

[12] Shishir Nagaraja, Amir Houmansadr, Pratch Piyawongwisal, Vijit Singh, Pragya Agarwal, and Nikita Borisov. Stegobot: a covert social network botnet. In *Information Hiding*, pages 299–313. Springer, 2011.

[13] V. Natarajan, Shina Sheen, and R. Anitha. Detection of stegobot: A covert social network botnet. In *Proceedings of the First International Conference on Security of Internet of Things*, SecurIT '12, pages 36–41, New York, NY, USA, 2012. ACM.

[14] J. R. Norris. *Markov Chains*. Cambridge University Press, 1 edition, 1998.

[15] A Perez-Garcia, C. Siaterlis, and M. Masera. Studying characteristics of an emulab testbed for scientifically rigorous experiments. In *Distributed Simulation and Real Time Applications (DS-RT), 2010 IEEE/ACM 14th International Symposium on*, pages 215–218, Oct 2010.

[16] Andreas Pfitzmann and Marit Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf, August 2010. v0.34.

[17] Rafael A. Rodríguez-Gómez, Gabriel Maciá-Fernández, and Pedro García-Teodoro. Survey and taxonomy of botnet research through life-cycle. *ACM Comput. Surv.*, 45(4):45:1–45:33, August 2013.

[18] S. Sebastian, S. Ayyappan, and P. Vinod. Framework for design of graybot in social network. In *Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on*, pages 2331–2336, Sept 2014.

[19] Ashutosh Singh, Annie H. Toderici, Kevin Ross, and Mark Stamp. Social networking for botnet command and control. *International Journal of Computer Network and Information Security*, 6:11–17, May 2013.

[20] William Stallings and Lawrie Brown. *Computer Security: Principles and Practice*. Pearson, 2 edition, 2012.

[21] F. Yergeau. UTF-8, a transformation format of ISO 10646. RFC 3629 (INTERNET STANDARD), November 2003.

[22] H.R. Zeidanloo and A.A. Manaf. Botnet command and control mechanisms. In *Computer and Electrical Engineering, 2009. ICCEE '09. Second International Conference on*, volume 1, pages 564–568, Dec 2009.