Social Network-Based Botnet Command-and-Control: Emerging Threats and Countermeasures

Erhan J. Kartaltepe[1], Jose Andre Morales[1], Shouhuai Xu[2], and Ravi Sandhu[1]

[1] Institute for Cyber Security, University of Texas at San Antonio
{erhan.kartaltepe,jose.morales,ravi.sandhu}@utsa.edu
[2] Department of Computer Science, University of Texas at San Antonio
shxu@cs.utsa.edu

Abstract. Botnets have become a major threat in cyberspace. In order to effectively combat botnets, we need to understand a botnet's Command-and-Control (C&C), which is challenging because C&C strategies and methods evolve rapidly. Very recently, botmasters have begun to exploit social network websites (e.g., `Twitter.com`) as their C&C infrastructures, which turns out to be quite stealthy because it is hard to distinguish the C&C activities from the normal social networking traffic. In this paper, we study the problem of using social networks as botnet C&C infrastructures. Treating as a starting point the current generation of social network-based botnet C&C, we envision the evolution of such C&C methods and explore social networks-based countermeasures.

Keywords: Botnet, command-and-control, social networks, security.

1 Introduction

The critical difference between botnets and other malware is that botmasters use a Command-and-Control (C&C) to coordinate large numbers of individual bots (i.e., compromised computers) to launch potentially much more damaging attacks. Botmasters also evolve their C&C strategies and methods very rapidly to evade defenders' countermeasures. Therefore, from a defender's perspective, it is always important to understand the trends and practices of botnet C&C [8,15,19,23,29,11,26]. Previous studies have mainly focused on two approaches: *host-centric* [31,36] and *network-centric* [16,18,17,3,7,8,14,23,20,6]. The host-centric approach aims to detect suspicious host activities, such as the use of incoming network data as system call arguments. The network-centric approach attempts to detect suspicious network activities by (for example) identifying network traffic patterns. The fact that a social network-based botnet C&C on `Twitter.com` was detected by "digging around" [25] suggests that we need to pursue more detection approaches.

Our contributions. Through an *application-centric* approach, we study the problem of botnets that use social network websites as their C&C infrastructures. First, we characterize the current-generation of social network-based botnet C&C, describing their strengths and weaknesses. Our characterization, while inspired by [25], is broader and deeper. Second, we envision how current social network-based botnet C&C might evolve in the near future, which capitalize on their strengths while diminishing their weaknesses. Third, we explore countermeasures for dealing with both current and future generations of social network-based botnet C&C. Since social network providers as well as client machines are victims of a social network-based botnet C&C, both server-side and client-side countermeasures are demonstrated and tested for both effectiveness and performance. Fourth, we discuss the limitations of the application-centric approach demonstrated in this paper, which suggests the need to integrate it with the aforementioned host-centric and network-centric methods because the three approaches are complementary to each other.

Paper outline. Section 2 discusses related prior work. Section 3 presents a characterization of the current generation of social network-based botnet C&C. Section 4 envisions the next-generation of social network-based botnet C&C. Sections 5 and 6 investigate server-end and client-end solutions to detecting social network-based botnet C&C, respectively. Section 7 discusses how to integrate them and how they can benefit from host-centric and network-centric approaches. Section 8 describes future work and concludes the paper.

2 Related Work

Network-centric approach. This approach aims to detect botnets by correlating the network traffic of a computer population, including destination IP addresses, server names, packet content, event sequences, crowd responses, protocol graphs and spatial-temporal relationships [16,18,17,3,7,8,14,23,20,6]. This approach is especially useful when only network traffic data is available.

Host-centric approach. This approach aims to differentiate malicious from benign processes running on host computers by observing that bot processes often use data received from the network as parameters in system calls [31]. A detection technique based on a high rate of failed connection attempts to remote hosts was recently presented in [36], which does not necessarily apply to the type of botnets we consider in the present paper because the connections are to popular social networking sites and are generally successful. This approach often looks deeply into the software stack [36].

Application-centric approach. This approach looks into the application-specific interactions. Previously, the focus has been put on IRC-based botnets (see, e.g., [14]). Recently, the possibility of exploiting emails as a botnet C&C was investigated [30], and the feasibility of detecting such botnets through their resulting spam traffic was presented in [35,34,37,22]. In this paper we consider a specific class of applications, namely web-based social networking. The concept of social network-based botnet C&Cs can be dated back to 2007 [21,12,28,13], but such botnets became a reality only very recently [2,25]. In particular, [25] served as the starting point of the present study. It should be noted that the focus of the present paper is only remotely related to the abuse of social networks for other purposes [1].

3 Characterizing Current Social Network-Based Botnet

How Does Current Social Network-Based Botnet C&C Work? Jose Narario first reported the actual use of social network as a botnet C&C [25], although the concept had been proposed as early as 2007 [21,28]. We call such a bot Naz, after its discoverer. At a high-level, Naz's botnet used accounts with the name `upd4t3` owned by the botmaster on social network sites `Twitter.com` and `Jaiku.com`. These bots received Base64-encoded commands from these accounts to download malicious payloads onto the victimized bot computers. Since then, other C&Cs were discovered with variations of the botnet's scheme, such as the `Twitter.com` account `Botn3tControl`, which was shutdown days later.

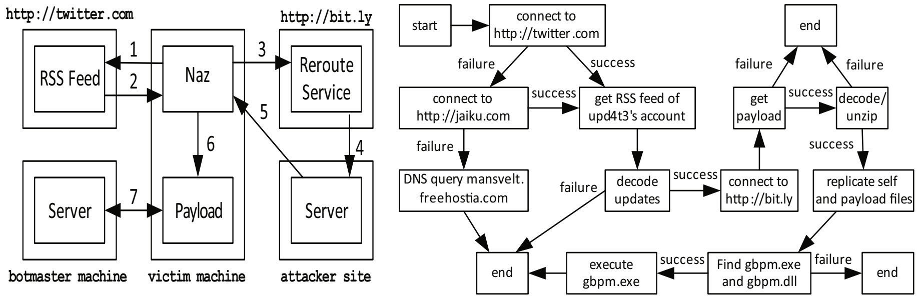

Fig. 1. Naz's C&C attack flow (left) and control flow (right)

To understand the behavior of Naz's botnet, we conducted two experiments. The first confirmed and extended Naz's C&C flow reported in [25]. Specifically, we obtained and ran a Naz sample on a machine by replacing its references to `Twitter.com` with a server under our control. This was necessary because the `Twitter.com` account was shutdown by `Twitter.com`'s administrators shortly after its detection. We used our own Base64-encoded messages with a `bit.ly` URL we set up that redirected to the payload stored on our server. The payload was a Base64-encoded, compressed archive containing two files: `gbpm.exe` and `gbpm.dll`. These files were the originally identified payload package of Naz. In our analysis, we reconstructed the original C&C flows described below and depicted in the left-hand side of Figure 1.

1. The bot makes a `HTTP GET` request to `upd4t3`'s RSS (Really Simple Syndication) feed at `Twitter.com`.
2. `Twitter.com` returns the RSS feed, containing Base64-encoded text in the description nodes.

3. The bot decodes the Base64-encoded text, revealing one or more `bit.ly` URLs, and makes a `HTTP GET` request to each. The `bit.ly` website provides short aliases for long URLs.
4. Each `bit.ly` URL redirects to a malicious zip file hosted on an independent attack server.
5. The bot downloads the malicious zip file as a payload.
6. The bot decodes and unzips the payload, replicates itself and the payload's uncompressed files, and then executes the payload's contents.
7. The payload attempts to gather user information from the victim computer and send it to a server selected by the botmaster.

To have a deeper understanding of the internal control flow of Naz, we conducted further black-box testing using data provided by Network Monitor [9] and CWSandbox [10], from which we draw the control flow details on the right-hand side of Figure 1. We observed that the bot made a copy of itself and the two files mentioned above in a temporary directory, and that when executing `gbpm.exe`, the bot dynamically injected code into `gbpm.exe`'s process. Moreover, we observed that Naz handled unexpected inputs as follows:

1. When we provided a URL to a bogus RSS feed, the bot failed to connect and instead attempted to access an RSS feed from a `Jaiku.com` account. This account name was hardwired in the bot program and had been deactivated by `Jaiku.com`'s administrator. This second connection failure led the bot to issue a DNS query on the domain name `mansvelt.freehostia.com`, after which the bot stopped producing network traffic. The site `mansvelt.freehostia.com` is currently unregistered and has no IP address.
2. When we placed plaintext sentences and URLs in our in-house RSS feed, the bot read the RSS feeds but did not show evidence of decoding and using the text in connection attempts.
3. When we modified the payload file to Base64-encoded only, compressed only, and replaced it with an executable file, the bot did not attempt to unzip or execute the file's contents. When we renamed the two payload files `gbpm.exe` and `gbpm.dll`, the bot did not attempt to execute the renamed `gbpm.exe` file, implying that the bot was program name sensitive.

Finally, it is interesting to note that a dynamic analysis of `gbpm.dll` revealed that the payload attempted to connect to a bank in Brazil. Moreover, both the analysis in [25] and our independent experiments demonstrate that Naz's botnet C&C serves primarily as a Trojan downloader [32].

Strengths of Naz's Botnet C&C. Naz's botnet C&C has the following advantages when compared with other botnet C&C infrastructures and methods:

1. Abusing trusted popular websites as a C&C server. Social networks and Web 2.0 sites such as `Twitter.com`, `FaceBook.com`, `LinkedIn.com`, and `YouTube.com` are not only legitimate, with verifiable SSL or EV-SSL certificates, but also heavily used by millions of users. Due to this heavy usage, light occasional traffic to one or more accounts is unlikely to be noticed

compared to a user's actual traffic pattern. This avoids any unnecessary and sometimes suspicious DNS queries (e.g., for non-popular DNS names).
2. Exploiting popular port(s) for C&C communication. Port 80 is the de facto standard for web traffic, and most network traffic will flow through it. This helps bots blend in with benign traffic.
3. Abusing application features for automatic C&C. The botmaster uses application features, such as RSS feeds, to automatically update bots. Moreover, the commands are so light-weight that they cannot be easily discerned from normal social network traffic.

The above discussion demonstrates that botmasters have begun to exploit "*hiding in plain sight*" to conduct stealthy botnet C&C. By piggybacking on the reputation and legitimacy of social network websites, botnet C&C activities may remain hidden, while defeating the "many eyes" defense [30].

Weaknesses of Naz's Botnet C&C. We need to understand the weaknesses of current generation of social network-based botnet C&Cs because these weaknesses will likely be absent in future generations. This is not meant to help the attackers, rather it is meant to help the defenders look ahead. Our examination shows that Naz's botnet C&C has weaknesses, which are omitted due to space limitation.

4 Envisioning Future Social Network-Based Botnet

In order to defeat future social network-based botnets, we must think ahead of the attackers. For this purpose, we can show how the aforementioned weaknesses of the current generation of social networks-based botnet C&Cs can be avoided. Due to space limitation, details are omitted.

5 Server-Side Countermeasures

5.1 The Detection Mechanism

A key observation behind our detection mechanism is that, regardless of the channel, provider, or account, social network messages are in text. Thus, if botmasters want to use social networks for their C&C, they would encode their commands textually. Moreover, just like legitimate messages may include web links, so might C&C messages (e.g., links for downloading payload). These observations inspired us to distinguish between encoded and plain texts and to follow unencoded links to their destination. Our detection mechanism can be adopted by the webserver as shown in Figure 2, and the resulting system would operate as follows (with steps 2 and 3 relevant to our countermeasure):

1. Alice logs into her social network and updates her status display using a content form.
2. The social network's content updater sends the text content to our server-end system.

3. The detection mechanism, which will be implemented as a classifier in our prototype system, determines if the text is suspicious and returns a result.
4. The content form updates the database with the message and whether it was marked suspicious.
5. Bob check's Alice content display, either through a feed like RSS or by visiting the site.
6. The content display requests the content from the database.
7. The database returns non-suspicious content; if a threshold level of suspicious messages (determined by policy) has been reached, the database returns a "suspicious account" message.
8. The content display shows those retrieved messages to the user, or a "suspicious account" message if the suspicious message threshold has been reached.

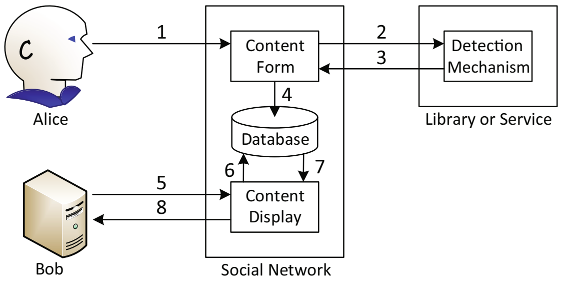

Fig. 2. Example scenario using our server-side detection mechanism

The server-side detection mechanism has the following advantages. First, it is *account agnostic* because it looks for text attributes that are shared with encoded text rather than individual behavioral patterns. Second, it is *language agnostic* because it looks at text for attributes that are shared with encoded text rather than individual words. As a result, the detection mechanism is effective for any language using Roman characters (English, Spanish, French, etc.). Third, it is *easy to deploy* because it can take advantage of light-weight machine learning algorithms and thus make decisions in real-time. Moreover, the code is easy to deploy as a library or software-as-a-service. Fourth, it can *follow unencoded links* to determine if the destination is a trusted source, say by using SSL authentication as a trust infrastructure. In the next two subsections we analyze the effectiveness and performance of our approach.

5.2 Prototype Implementation and Its Effectiveness and Limitations

Prototype implementation. To demonstrate the effectiveness of our system, we instantiated the detection mechanism as `Weka`'s [33] J48 decision tree algorithm (because it is quick and readily usable, but other tailored algorithms can be used instead in a plug-and-play fashion) to classify input messages so as to distinguish between Base64- or Hexadecimal-encoded text and natural language text. For links in the clear, by following links to their destination, we can mark the content as "suspicious" if it is an atypical file (e.g., an executable, library, encoded, or compressed file, or a combination of these). To build a pool of "non-suspicious" text, we screenscraped 200 `Twitter.com` accounts to build a list of 4000 messages. Our pool of bot commands were 400 short random commands of fifteen to thirty characters that were then encrypted using RC4 stream cipher and then encoded, giving a 10:1 set of normal to suspicious text. We then split the messages into a training set with 70% of both types and a test set with the remaining 30%. Recognizing that altering the natural Base64 or Hexadecimal alphabet with alternate characters such as spaces or punctuation could be used to obfuscate the text, we also ran our classifier against such alternate encoding schemes.

Effectiveness. For standard Base64 and Hexadecimal encoding schemes, our classifier was able to quickly distinguish between our "normal" and "suspicious" text samples in an account-agnostic way, no false positives and no false negatives, for both Base64 and Hexadecimal encoding (see Table 1). Moreover, our classifier maintained this accuracy even when the commands were obfuscated with other words—the distinctiveness of the encoded commands was readily apparent. The results were so perfect because the attributes we used—number of spaces in the text, longest word, and shortest word—cleanly divided the "normal" and "suspicious" text. To produce non-standard Base64 and Hexadecimal encoding schemes, we randomly swapped ten of the standard alphabet with alternate ones from a pool of space and punctuation characters. Our profiler was able to distinguish between them in an account-agnostic way, with a false positive rate of 0.0% and false negative rate of 1.25% for Alternate Base64 encodings, and a false positive rate of 3.25% and false negative rate of 12.5% for Alternate Hexadecimal encodings (see Table 1).

Table 1. Results with respect to various Base64 and Hexadecimal encodings

	Base64		**Hexadecimal**		**Alt. Base64**		**Alt. Hexadecimal**	
	Actual Positive	Actual Negative	Actual Positive	Actual Negative	Actual Positive	Actual Negative	Actual Positive	Actual Negative
Tested Positive	100%	0%	100%	0%	100%	1.25%	96.75%	12.5%
Tested Negative	0%	100%	0%	100%	0%	98.75%	3.25%	87.5%

The classifier's accuracy dropped significantly when the commands were obfuscated with other words, especially with Hexadecimal encoding schemes and with spaces as alternate characters. We note that with such an encoding mechanism, *a priori* knowledge of words or characters to excise from the message would be necessary to extract the non-command content from the meaningful botnet commands. This form of steganography is essentially indistinguishable from typical steganography, where a botmaster would hide the bot commands in such a way as to not attract attention to themselves, i.e., using natural language words as code for commands or URLs.

Limitations. Hiding commands in a social network-based C&C using steganography makes it difficult for programs or even humans to identify the presence of a command within a message. Since social network messages left by users are unstructured content, a crafty adversary can hide a bot command within a message in such a way that a human reading the message could not identify the message as a command. Combined with encryption, reverse analysis—even with a captured bot—may not yield the interpreted commands. Thus, running a steganographic reversal algorithm on a C&C message would not return data that was a clear bot command. [30]. However, our server-side solution coupled with a client-side counterpart that detects when a process is acting on input from a social network-based C&C would provide a complete solution to this emerging threat. In Section 7 we will discuss how these limitations may be overcome in a bigger solution framework.

5.3 Performance

Evaluation methodology and environment setup. In order to demonstrate the efficiency of our server-side detection mechanism, we measured the performance of our prototype. We implemented it into CompactSocial, a microblogging service that emulates the constraints of `Twitter.com`. CompactSocial provides a simple interface to both update a status message and view any account's messages using a web browser. Moreover, an auto-updated RSS feed contains the text of the last ten account updates. CompactSocial was written in Java 6, update 11 and ran as a web application deployed to Apache Tomcat 6.0.20. When used as a library, our server-end solution was deployed as a `.jar` file; when deployed as a service, the classifier ran as a stand-alone web application deployed to Apache Tomcat 6.0.20. The classifier and CompactSocial used shared cryptographic keys for authentication. For processing incoming and outgoing messages, Javas crypto library was used to compute any hash value or HMAC it needed, in both cases using the SHA1 algorithm.

The system environment is depicted in Figure 3. Both servers reside within a university campus network, and the CompactSocial clients are both within and outside the campus network. The CompactSocial and text-classifier servers are called `mercury` and `apollo`, respectively. There are two CompactSocial client machines: `minerva` acted as an external computer within the LAN with authorized access to `mercury` through a simple CompactSocial client, and `mars` was an adversary client machine within the campus network, employing Naz+ to read

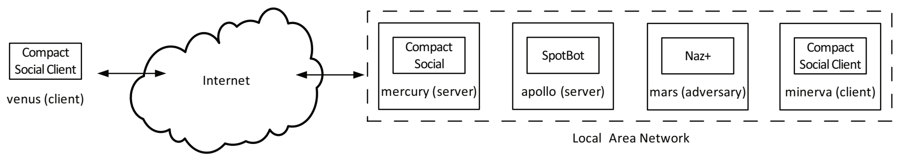

Fig. 3. Integrating the server-side solution into real life systems

updates made by `minerva`. A fifth machine, `venus` mimicked the `minerva`'s functionality and tested the performance of the classifier on a non-dedicated internet connection. The three servers, hermes, jupiter, and euclid recognized each other by sharing some pair-wise keys. Table 2 reviews the concrete configurations of the machines and networks.

Table 2. System settings (all machines use Intel Core 2 Duo, 2.93 GHz processor)

Machine	Internet Connection	Relevant Software
mercury	Gigabit LAN	Apache Tomcat 6.0.20, Sun JVM, CompactSocial
apollo	Gigabit LAN	Apache Tomcat 6.0.20, Sun JVM, classifier service
minerva	Gigabit LAN	Firefox 3.5, CompactSocial client
mars	Gigabit LAN	.NET 3.5 Framework, Naz+
venus	100 Megabit Cable	Firefox 3.5, CompactSocial client

A CompactSocial client was developed in JavaScript to simulate a user updating their status in the CompactSocial web application. The CompactSocial client was developed as an addon and installed into Mozilla Firefox. Additionally, Naz+ was developed as a Windows service and written in C#, targeting the Microsoft .NET Framework, version 3.5. Naz+ periodically checked the RSS feed for CompactSocial test account, parsed the XML for the description node which contained the bot command as encrypted text (using a shared key with `mars` who updated the status message), and executed the command.

Performance benchmarking. To examine the delay incurred by the classifier when utilized by CompactSocial, time was marked before and after each transaction over 30000 requests at varying rates. We repeated this test ten times and took the average over the runs. Figure 4 shows the results over varying requests per second when the classifier was used as a library and a service.

When used as a library, our classifier performed roughly twice as fast than its service counterpart, since no network traffic was required. At even 500 requests per second, the classifier handled all requests without incident. In practice, a

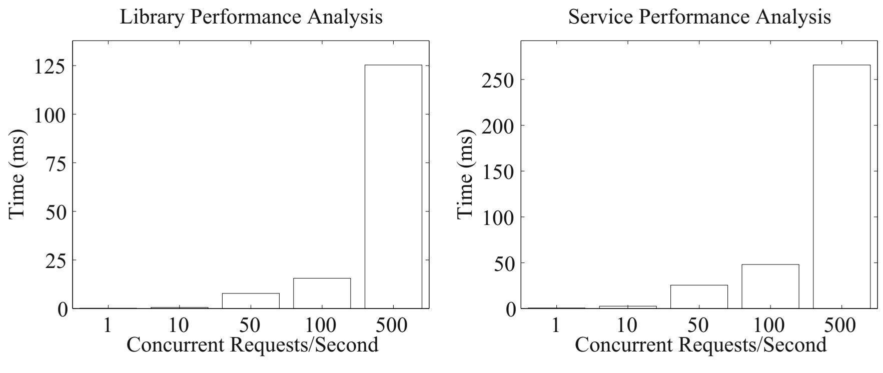

Fig. 4. Classifier library (left) and service (right) performance analysis

large-scale enterprise would use our server-side classifier as a service, whether as an in-house server to make requests or a pay-as-you-go service to a third party. In our preliminary testing, the classifier service handles 500 requests per second on a non-dedicated machine with cycles to spare. In the advent that a larger throughput was necessary, a load balancer can reduce the requests for a particular machine to a manageable level.

Meeting the needs of real-life systems. For a social network provider like `Twitter.com`, that has fifteen million users, and is increasing at roughly one million users a month, we wondered how our classifier would stack up. A large percentage (85.3%) of `Twitter.com` accounts post less than once a day, whether due to lost interest or not having much to post. We classify these accounts as "passive users". The vast majority (99.5%) of accounts post less than sixteen messages daily, which would be a class of "active" users, who post regularly about events. The remaining 0.5% post more than sixteen updates, although `Twitter.com` restricts accounts to a 1000 updates per day limit. These are particularly engaged users or are shared accounts by multiple people in an organization posting under a common account. We classify these users as "explosive" users [5].

If all accounts hit their ceilings on `Twitter.com`, we'd have the posting rates in Table 3. With fifteen million users, `Twitter.com` would handle a ceiling of 1410.7 messages per second (in actuality, most users do not hit these ceilings, so the actual threshold is far less). In this worst case, assuming the one million account per month growth rate and the same distribution of account usage, `Twitter.com` will accommodate an uptick per month of nearly 100 messages per second.

Table 3. `Twitter.com` usage analysis

15000000 users	**Percentage**	**Update Rate**	**Messages Per Day**	**Messages Per Second**
Passive users	85.3%	1/day	12795000	148.1
Active users	14.2%	16/day	34080000	394.5
Explosive users	0.5%	1000/day	75000000	868.1
Total users	100%	—	121875000	1410.7

Because our server-side approach is account agnostic, it does not need to build an account history for each user and as a result, would only need to check a few messages to determine if the account is being used in a suspicious way. Given the above scenario and a policy that checks periodically verifies one message daily and the first three messages for a new user's account, then our classifier would only need to check fifteen million messages per day, or 173.7 messages per second, with a monthly increase of 12.1 messages per second. If only active and explosive accounts were targeted (which would be more likely behavior for a botnet C&C), this would decrease to 25.6 messages per second with an increase of 2.1 messages per second. Thus, even with these simple non-discriminating policies and worst-case `Twitter.com` usage scenario, our classifier as a service can handle an enterprise-level throughput of requests, and different policy strategies may be employed to throttle down the throughput further.

6 Client-Side Countermeasures

6.1 The Detection Mechanism

Detection attributes. We propose detecting social network-based botnet C&Cs using three attributes: self-concealing, dubious network traffic, and unreliable provenance.

- Self-Concealing: A self-concealing process is one which attempts to avoid detection with the use of stealth mechanisms. We consider two specific instances of this type:

  Graphical User Interface. Many applications that read RSS feeds interact with a user via a graphical user interface. Bots and other malware will attempt to avoid detection, and as a result may run in the background as a service or hidden process without an explicit interface. A process without a graphical user interface can be identified as possibly self-concealing.

  Human Computer Interaction. Most benign software works by reacting to user input via a keyboard or mouse. Malware processes tend to run hidden and independent of user input and don't require explicit keyboard or mouse events provided by a user to perform a nefarious act. A process without human/computer interaction can be identified as possibly self-concealing.
- Dubious Network Traffic: A process with dubious network traffic is one which engages in network communication with another machine in a covert or devious way. We consider three specific instances of this type:

  Social Network Request. Exclusively visiting social networking sites is not suspicious; however, social network-based botnet C&C craftily abuse the popularity and good name of social networking sites; thus, exclusive requests to social networking or web 2.0 sites is considered a possible trigger event for dubious network traffic.

  Encoded Text Processing. Since social network-based bots read commands as encoded text, processes making connections to sources that provide encoded or encrypted text is anomalous. Accepting connections with encoded text and processing it by decoding or decrypting it can be considered as possibly dubious network traffic.

  Suspicious File Downloading. In general, applications do not download suspicious files such as executable, library, compressed, or encoded/encrypted files without permission (though they may download image or text files). Social network-based C&C bot processes, on the other hand, act as Trojan downloaders and almost exclusively save executables or DLLs to the filesystem as malicious payload. Downloading such suspicious files can be considered as dubious network traffic.
- Unreliable Provenance: A process with unreliable provenance is one which lacks a reliable origin. We consider three specific instances:

  Self-Reference Replication. This is a feature malware uses to survive disinfection on a host machine, occurring when a process copies itself into a newly created file or an existent file (by modifying it) on the file system [24]. An installed file with its installer not having a verified signature can be identified as possibly having an unreliable provenance.

Dynamic Code Injection. This is used by malware to insert malicious code into the memory space of an active process. Its end goal is to modify the process to perform nefarious deeds, possibly by piggybacking on that application's authorization settings. A process whose injector lacks a digital signature can be identified as possibly having an unreliable provenance since the injector's origin cannot be established.

Verifiable Digital Signature. Digital signatures may be considered a hallmark of trust between users and well established software. Most organizations that publish software provide a signature for their program and related files. Malware authors typically do not employ digital signatures; as a consequence, a process running without a verifiable digital signature can be identified as possibly having an unreliable provenance.

Detection model. We say a process P has the self-concealing attribute (P_{sc}) if it lacks a graphical user interface ($P_{gui} = \mathsf{false}$) and does not accept human computer interaction ($P_{hci} = \mathsf{false}$). More formally,

$$(\neg P_{gui}) \wedge (\neg P_{hci}) \rightarrow P_{sc}.$$

We say a process P has the dubious network traffic attribute (P_{dnt}) if it performs social network requests ($P_{snr} = \mathsf{true}$) and encoded text processing ($P_{etp} = \mathsf{true}$) or does suspicious file downloading ($P_{sfd} = \mathsf{true}$) (or both). More formally,

$$P_{snr} \wedge (P_{etp} \vee P_{sfd}) \rightarrow P_{dnt}.$$

We say a process P has the unreliable provenance attribute (P_{up}) if it performs self-reference replication ($P_{srr} = \mathsf{true}$) or does dynamic code injection ($P_{dci} = \mathsf{true}$), and also lacks a verified digital signature ($P_{vds} = \mathsf{false}$). More formally,

$$(P_{srr} \vee P_{dci}) \wedge (\neg P_{vds}) \rightarrow P_{up}.$$

Putting the above altogether, we classify a process P as being suspicious of being a social network-based bot C&C process (P_{snbb}) if it is either self-concealing ($P_{sc} = \mathsf{true}$) or has an unreliable provenance ($P_{up} = \mathsf{true}$) (or both), and engages in dubious network traffic ($P_{dnt} = \mathsf{true}$). More formally,

$$(P_{sc} \vee P_{up}) \wedge P_{dnt} \rightarrow P_{snbb}.$$

6.2 Effectiveness and Limitations

Evaluation methodology and environment setup. To examine the effectiveness of the detection model described above, we collected data with respect to our detection attributes for both benign and malicious processes in order to distinguish them. For this purpose, we considered eighteen benign applications,

Table 4. Client-side test set ("SN-Based Bot" stands for "Social Network-Based Bot")

Application	Type	Application	Type	Application	Type
AOL Explorer	Web Browser	Internet Explorer	Web Browser	RSS Bandit	RSS Aggregator
Avant	Web Browser	K-Meleon	Web Browser	RSS Owl	RSS Aggregator
Bobax	Traditional Bot	Maxthon	Web Browser	SeaMonkey	Web Browser
BlogBridge	RSS Aggregator	Mercury	RSS Aggregator	Snarfer	RSS Aggregator
FeedDemon	RSS Aggregator	Naz	SN-Based Bot	Tweetdeck	Twitter Client
FireFox	Web Browser	Naz+	SN-Based Bot	Twhirl	Twitter Client
Flock	Web Browser	Opera	Web Browser	Virut	Traditional Bot
Google Chrome	Web Browser	Ozdok	Traditional Bot	Waledac	Traditional Bot

four traditional bots, and the malicious Naz and prototype Naz+ bots (listed in Table 4). To provide a wide breadth, the benign applications are a broad selection of the most popular web browsers, RSS aggregators, Twitter clients, and RSS aggregators which read subscription feeds. Testing was performed using VMWare Workstation running Microsoft Windows SP3 using `NAT` for Internet access. Each application was executed separately for a period of four hours, followed by post-analysis. During testing of the eighteen benign applications, we interacted with each application by subscribing to and viewing different RSS feeds; attempting to subscribe to bogus RSS feeds, updating all RSS feeds every hour, reading individual feed articles. These tests were done to provide a wide range of expected and unexpected scenarios for each application to deal with while recording their behavior. In addition, we used a number of sensors to gather information about each process. Tracing of the three detection attributes described in Section 6.1 occurred from the moment a process starts executing.

Network traffic was collected using Windows Network Monitor [9]. Keyboard and mouse input was collected with a modified version of `GlobalHook`. Digital signatures were verified using SigCheck. Self-reference replication and dynamic code injection were accomplished with kernel hooks implementing known techniques [24]. The presence of a user interface was recorded by observing the creation of any window upon executing each application using `EasyHook`. User input, network traffic, graphical user interface interaction, self-reference replication and dynamic code injection were all recorded in real-time. For Virut and Waledac, static analysis and previous executions of these bots yielded their dubious network traffic results. Digital signatures were verified after the four hour testing of each process.

Effectiveness. The results are summarized in Table 5 and highlight some observations below. First, we observe that all benign applications lacked the self-concealing attribute as they all utilized a graphical user interface and accepted inputs from the user, such as reading an article, following a link, or updating an RSS feed. All bots but Virut demonstrated the self-concealing attribute since they did not have a graphical user interface or accept user input, although it appears that the command prompt window Virut displays may be accidental. Second, all applications but Naz and Naz+ possessed the dubious network traffic attribute; RSS applications did not download suspicious files, but all of the bots did. All bots but Virut read inordinate amounts of encoded text; legitimate RSS

Table 5. Client-Side detection results ("IE" stands for "Internet Explorer")

Application	Self-Concealing		Unreliable Provenance			Dubious Network Traffic			Result
	Graphical User Interface	Human Computer Interaction	Self-Reference Replication	Dynamic Code Injection	Verifiable Digital Signature	Social Network Request	Encoded Text Processing	Suspicious File Download	Social Network-Based Bot?
AOL Explorer	Y	Y	N	N	Y	N	N	N	N
Avant	Y	Y	N	N	Y	N	N	N	N
BlogBridge	Y	Y	N	N	Y	N	N	N	N
FeedReader	Y	Y	N	N	Y	N	N	N	N
Firefox	Y	Y	N	N	Y	N	N	N	N
Flock	Y	Y	N	N	Y	N	N	N	N
IE	Y	Y	N	N	Y	N	N	N	N
Chrome	Y	Y	N	N	Y	N	N	N	N
K-Meleon	Y	Y	N	N	Y	N	N	N	N
Maxthon	Y	Y	N	N	Y	N	N	N	N
Mercury	Y	Y	N	N	Y	N	N	N	N
Opera	Y	Y	N	N	Y	N	N	N	N
RSS Bandit	Y	Y	N	N	Y	N	N	N	N
RSS Owl	Y	Y	N	N	Y	N	N	N	N
SeaMonkey	Y	Y	N	N	Y	N	N	N	N
Snarfer	Y	Y	N	N	Y	N	N	N	N
Tweetdeck	Y	Y	N	N	N	Y	N	N	N
Twhirl	Y	Y	N	N	N	Y	N	N	N
Bobax	N	N	Y	Y	N	N	Y	Y	N
Ozdok	N	N	Y	Y	N	N	Y	Y	N
Virut	Y	N	Y	Y	N	N	N	Y	N
Waledac	N	N	Y	Y	N	N	Y	Y	N
Naz	N	N	Y	Y	N	Y	Y	Y	Y
Naz+	N	N	N	N	N	Y	Y	Y	Y

reader applications generally did not. Additionally, only Naz and Naz+ communicated nearly exclusively with social networking sites (benign processes read from them only a fraction of the time, and traditional bots made no such communication requests). Third, all benign applications did not exhibit the unreliable provenance attribute; none attempted to replicate itself or inject code into another process, and they all possessed a verifiable digital signature. On the other hand, all bots tested displayed this attribute, as they all copied themselves or injected code into other processes (except Naz+), and lacked a verifiable digital signature.

Of note is that no social network-based bot was misclassified as a benign process and no benign application or traditional bot was misclassified as a social network-based bot. We reiterate that our goal is to detect social network-based botnet C&C, which explains why the other bots Bobax, Ozdok, Virut and Waledac were not classified as social network-based bots. These bots may be detected by using complementary host-centric or network-centric approaches, or by applying the relaxed dubious network traffic attribute described above. This also justifies why the countermeasures presented in the paper need be integrated into a comprehensive defense framework.

Limitations. A limitation of our effectiveness analysis is the lack of real-time analysis of other social network-based botnet C&C, due to other botnets undiscovered in the wild. Moreover, our analysis is conducted in post-analysis. We plan to develop an implementation of this technique to provide real-time data gathering and evaluation. Another limitation is that any one sensor can be defeated if the malware author knows the concrete details of its implementation; knowledge of the high-level detection model is not enough. For example, if we only seek `.exe` and compressed files, a malicious file can purposely be renamed to `.jpg` or `.html` which would bypass our file download sensor. In this case, the bot's internal logic will have extra overhead, possibly checking every file in the network traffic with these file extension to identify the malicious one, thus making this approach infeasible. Malware in general are known to have attributes that trigger many of these sensors [32] and thus it is unlikely that a bot process will effectively work and bypass all our sensors.

6.3 Performance Analysis

In order to measure the client-side countermeasure's performance, we used PassMark's PerformanceTest 7.0 benchmarking to gather information on its CPU usage [27]. We benchmarked the baseline case with no data collection running, and then ran the data collector tracking zero to five processes after an hour of data collection had passed. Running the data collector added a 4.8% overhead to the overall system, and running it to track one to five processes had between a 13.3% and 28.9% overhead (See also Figure 5). With optimization, these numbers could be lowered further.

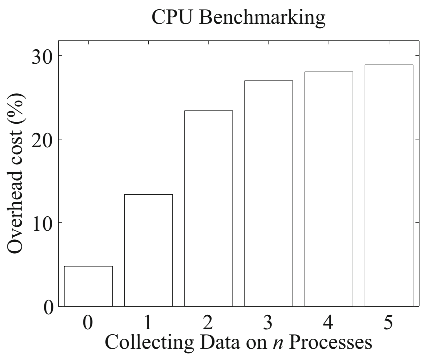

Fig. 5. CPU utilization for our client-side countermeasure process tracker

7 Integrating Server- and Client-Side Countermeasures

Integrated solution. As discussed above, the server-side and client-side countermeasures are integral parts of our client/server solution to detecting social network-based botnet C&C. When each type of countermeasures deployed alone, they can be defeated in certain ways. Integrating our server-side classifier into a social network webserver is straightforward whether it operates as a library or service, and integrating the client-side detection algorithms into existing malware detection schemes or operating its sensors as its own detection framework is equally as uncomplicated. Because both systems are stand-alone, there is no need to have the systems interoperate. Indeed, a user that has the client-side solution installed while using a social network that employs the server-side solution gains the benefits of both.

Limitations. Even when our classifier is utilized by a social network provider and a machine has our client solution installed, using both still has some limitations. Specifically, if the botmaster employs steganography into their social network-based C&C, the server-side solution in its current form will not detect that message being passed. Employing steganography in such a way will diffuse the content in the message, essentially expanding the text [4]. In this case, if using popular social networks like `Twitter.com` or `FaceBook.com` with character length limitations, spreading a command over multiple messages would likely be required. More specifically, our approach can benefit from host-centric and network-centric approaches as follows.

– **A bot that reads steganographic commands and can evade our client-side sensors.** One way for a bot to evade the client-side sensors is to exist at the kernel level. Since some of the client-side solution sensors exist at the user-level, the bot can effectively bypass enough of these sensors to mask its presence on the machine. Additionally, a bot that with intimate knowledge of the *implementation details* of the client-side sensors can maneuver around our countermeasures, such as writing code that falsifies sensor data. A host-centric approach to capture additional anomalous information at the kernel level would help mitigate this attack.
– **A bot that reads steganographic commands and masquerades as a benign process.** A bot that behaves as a benign process would have to lack the self-concealing or unreliable provenance attribute. By masquerading as a benign application, say by presenting itself as a graphical application that masks its true purpose, a bot could exist with such an interface. This bot would additionally have to trick the user into starting it and keeping it running, which might prove difficult. To avoid possessing an unreliable provenance, this bot would have to have a digital signature, which is difficult to forge. Additionally, it must not dynamically inject code into another source or replicate itself, which are hallmark signs of bots, since they wish to inculcate themselves into the host machine. A network-centric solution is necessary to analyze network layer data for similar events occurring from many machines in a network during a small timeframe.
– **A bot that reads steganographic commands and runs scripts.** A bot that behaves as a social network-based bot but downloads text files instead of executables will not be classified as a social network-based bot, although it would be marked as a possibly suspicious bot. If the bot contains or is aware of a scripting engine such as a Python interpreter, the bot can run the script instead of an executable. A host-centric approach to contain general purpose malware or prevent or alert the user of script/program execution would help stop this attack. Additionally, a network-centric strategy to detect script file downloads would help prevent the scripts from being downloaded to the client machine.

8 Conclusion and Future Work

We systematically studied a social network-based botnet and its C&C and discussed their future evolutions. We investigated, prototyped, and analyzed both server-side and client-side countermeasures, which are integral parts of a solution to the emerging threat of social network-based botnets. We also discussed how our solution can benefit from host-centric and network-centric botnet detection solutions so as to formulate a comprehensive defense against botnets.

Our future work includes: (1) implementing the client-side countermeasures as real-time detection systems, (2) improving the server-side classifier to detect steganography, (3) handling multiple stepping stones in payload redirection, and (4) and porting the client-side countermeasures to other platforms.

Acknowledgments. This work is partially supported by grants from AFOSR, ONR, AFOSR MURI, and the State of Texas Emerging Technology Fund.

References

1. Athanasopoulos, E., Makridakis, A., Antonatos, S., Antoniades, D., Ioannidis, S., Anagnostakis, K., Markatos, E.: Antisocial networks: Turning a social network into a botnet. In: Wu, T.-C., Lei, C.-L., Rijmen, V., Lee, D.-T. (eds.) ISC 2008. LNCS, vol. 5222, pp. 146–160. Springer, Heidelberg (2008)
2. Balatzar, J., Costoya, J., Flores, R.: The real face of koobface: The largest web 2.0 botnet explained. Technical report, Trend Micro (2009)
3. Binkley, J.R., Singh, S.: An algorithm for anomaly-based botnet detection. In: Proc. Reducing Unwanted Traffic on the Internet, SRUTI '06 (2006)
4. Chapman, M., Davida, G.I.: Plausible deniability using automated linguistic stegonagraphy. In: Conference on Infrastructure Security (October 2002)
5. Cheng, A., Evans, M.: Inside twitter: An in-depth look inside the twitter world, `http://www.sysomos.com/insidetwitter`
6. Collins, M., Reiter, M.: Hit-list worm detection and bot identification in large networks using protocol graphs. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 276–295. Springer, Heidelberg (2007)
7. Collins, M., Shimeall, T., Faber, S., Janies, J., Weaver, R., De Shon, M., Kadane, J.: Using uncleanliness to predict future botnet addresses. In: Proc. IMC '07 (2007)
8. Cooke, E., Jahanian, F., McPherson, D.: The zombie roundup: understanding, detecting, and disrupting botnets. In: Proc. SRUTI '05 (2005)
9. Microsoft Corporation. Network monitor 3.3, `http://go.microsoft.com/fwlink/?LinkID=103158&clcid=0x409`
10. CWSandbox.org. Cwsandbox—behavior-based malware analysis, `http://www.cwsandbox.org`
11. Dagon, D., Gu, G., Lee, C., Lee, W.: A taxonomy of botnet structures. In: Choi, L., Paek, Y., Cho, S. (eds.) ACSAC 2007. LNCS, vol. 4697, Springer, Heidelberg (2007)
12. DigiNinja. Kreiosc2: Poc using twitter as its command and control channel, `http://www.digininja.org`
13. Easton, T., Johnson, K.: Social zombies. In: DEFCON '09 (2009)

14. Goebel, J., Holz, T.: Rishi: identify bot contaminated hosts by irc nickname evaluation. In: Proc. HotBots '07 (2007)
15. Grizzard, J.B., Sharma, V., Nunnery, C., Kang, B.B., Dagon, D.: Peer-to-peer botnets: overview and case study. In: Proc. HotBots '07 (2007)
16. Gu, G., Perdisci, R., Zhang, J., Lee, W.: BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In: Security '08 (2008)
17. Gu, G., Porras, P., Yegneswaran, V., Fong, M., Lee, W.: BotHunter: Detecting malware infection through ids-driven dialog correlation. In: USENIX Security '07 (2007)
18. Gu, G., Zhang, J., Lee, W.: BotSniffer: Detecting botnet command and control channels in network traffic. In: Proc. NDSS '08 (2008)
19. Holz, T., Steiner, M., Dahl, F., Biersack, E., Freiling, F.: Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In: LEET '08 (2008)
20. Hu, X., Knysz, M., Shin, K.G.: Rb-seeker: Auto-detection of redirection botnets. In: Proc. NDSS '09 (2009)
21. Finjan Software Inc. Web security trends report q4 2007. Technical report, Finjan Software Inc. (2007), `http://www.finjan.com/Content.aspx?id=827`
22. John, J., Moshchuk, A., Gribble, S., Krishnamurthy, A.: Studying spamming botnets using botlab. In: Proc. NSDI '09 (2009)
23. Karasaridis, A., Rexroad, B., Hoeflin, D.: Wide-scale botnet detection and characterization. In: Proc. HotBots '07 (2007)
24. Morales, J.A., Clarke, P.J., Deng, Y., Kibria, B.G.: Identification of file infecting viruses through detection of self-reference replication. Journal in Computer Virology (2008)
25. Nazario, J.: Twitter based botnet command and control (2009), `http://asert.arbornetworks.com/2009/08/twitter-based-botnet-command-channel`
26. Nazario, J., Holz, T.: As the net churns: Fast-flux botnet observations. In: Proc. MALWARE '08 (2008)
27. PassMark.com. Passmark performancetest 7.0, `http://www.passmark.com/products/pt.htm`
28. Poland, S.: How to create a twitter bot (2007), `http://blog.stevepoland.com/how-to-create-a-twitter-bot/`
29. Rajab, M.A., Zarfoss, J., Monrose, F., Terzis, A.: A multifaceted approach to understanding the botnet phenomenon. In: Proc. IMC '06 (2006)
30. Singh, K., Srivastava, A., Giffin, J., Lee, W.: Evaluating email's feasibility for botnet command and control. In: Proc. DSN
31. Stinson, E., Mitchell, J.C.: Characterizing bots' remote control behavior. In: Hämmerli, B.M., Sommer, R. (eds.) DIMVA 2007. LNCS, vol. 4579, pp. 89–108. Springer, Heidelberg (2007)
32. Szor, P.: The Art of Computer Virus Research and Defense. Symantec Press (2005)
33. Weka 3 data mining software, `http://www.cs.waikato.ac.nz/ml/weka/`
34. Xie, Y., Yu, F., Achan, K., Panigrahy, R., Hulten, G., Osipkov, I.: Spamming botnets: signatures and characteristics. In: Proc. SIGCOMM '08, pp. 171–182 (2008)
35. Zhao, Y., Xie, Y., Yu, F., Ke, Q., Yu, Y., Chen, Y., Gillum, E.: Botgraph: large scale spamming botnet detection. In: Proc. NSDI '09 (2009)
36. Zhu, Z., Yegneswaran, V., Chen, Y.: Using failure information analysis to detect enterprise zombies. In: Proc. Securecomm '09 (2009)
37. Zhuang, L., Dunagan, J., Simon, D., Wang, H., Osipkov, I., Hulten, G., Tygar, J.: Characterizing botnets from email spam records. In: Proc. LEET '08 (2008)