*I. J. Computer Network and Information Security*, 2013, 6, 11-17
Published Online May 2013 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2013.06.02

# Social Networking for Botnet Command and Control

Ashutosh Singh, Annie H. Toderici, Kevin Ross, Mark Stamp
San Jose State University , San Jose, California
itsiashu@gmail.com, anniehii@gmail.com, kevin.ross@sjsu.edu, stamp@cs.sjsu.edu

***Abstract* —** A botnet is a group of compromised computers—often a large group—under the command and control of a malicious botmaster. Botnets can be used for a wide variety of malicious attacks, including spamming, distributed denial of service, and identity theft. Botnets are generally recognized as a serious threat on the Internet. This paper discusses SocialNetworkingBot, a botnet we have developed that uses Twitter for command and control. In SocialNetworkingBot, the botmaster tweets commands that are acted on by the individual bots. We discuss the functionality and implementation of SocialNetworkingBot, as well as a small-scale experiment that we have conducted. The botnet presented here is intended to serve as a proof of concept and a platform to facilitate further research.

***Index Terms* —** Botnet, Twitter, malware

## I. Introduction

A botnet is a collection of compromised computers controlled by a botmaster. The compromised computers, or bots, can be used for attacks such as distributed denial of service (DDoS), click fraud, identity theft, and spamming. Most botnets have a command and control server that the botmaster uses to issue commands to the individual bots [1]. Although it is difficult to determine the size of a botnet, some have been estimated to have millions of active bots [14].

In this paper, we discuss a botnet that we have developed. This botnet, which we refer to as SocialNetworkingBot, uses Twitter for its command and control structure. SocialNetworkingBot is intended to demonstrate the potential for such a botnet, and to serve as a vehicle for further research on defenses against social media-based botnets.

From the attacker's point of view, there are several potential benefits to using Twitter (or other social media) for botnet command and control. Unlike most traditional botnet architectures, in a Twitter-based botnet, there is no need for the botmaster to install or access a server. In addition, a Twitter-based botnet is very simple to create and easy to maintain. But, the most obvious advantage to using Twitter is that it is difficult to distinguish legitimate activity from botnet-related activity. In effect, the botnet command and control messages can "hide in plain sight." Alternatively, we can view Twitter as acting as a type of covert channel for the botnet, that cannot be easily detected or shut down. We have more to say about these issues in the next section.

At least since 2009, there have been reports of botnets using Twitter and other social media for command and control. For example, the botnet discussed in [15] apparently uses Twitter status messages to instruct bots to download executable files. This particular bot is reportedly focused on stealing information.

Another Twitter botnet is discussed in [11]. This particular botnet may have been very short lived—it went down the same day that it was detected. This botnet was supposedly part of a larger group of botnets of Mexican origin that were collectively used for a variety of illicit activities, including spamming, phishing, and DDoS attacks.

Yet another botnet that used Twitter is discussed in [12]. This botnet was designed to attack an electronic currency known as Bitcoin.

In addition to the specific Twitter-based botnets mentioned above, there has recently been some general discussion about the increased activity of social media-based botnets [17]. However, we can find no example of an existing Twitter-based botnet that is well documented or readily available for analysis. Our goal here is to develop a Twitter-based botnet that clearly demonstrates the potential for such a botnet. This work provides researchers with a tangible example that can be used to study possible attacks by such botnets, as well as a tool for testing defensive strategies against this relatively new malware threat.

This paper is organized as follows. In Section II we cover background information on botnets and other relevant topics, including covert channels and selected communication protocols. Section III provides details about our SocialNetworkingBot application, with emphasis on its Twitter-based command and control structure. Then in Section IV, we discuss various attacks that can be performed using our botnet. In Section V we mention some small-scale experimental results. Finally, Section VI contains our conclusions and suggestions for future work.

## II. Background

In this section we provide relevant background information, with the emphasis on botnet command and

 

control structures. We also briefly cover specific examples of botnets.

*A. Botnet Structure*

In a generic botnet, several components are necessary, including a command and control structure, a communication protocol, bot related functionality, an infection method, and trigger events [20]. Once infected, a victim's computer typically executes a script (i.e., shell code) that fetches an image of the actual bot binary from a specified location. The bot binary is then installed on the target machine.

Fig. 1 depicts a generic botnet command and control structure. In this case, the botmaster issues commands through Internet Relay Chat (IRC) servers to a set of infected hosts.

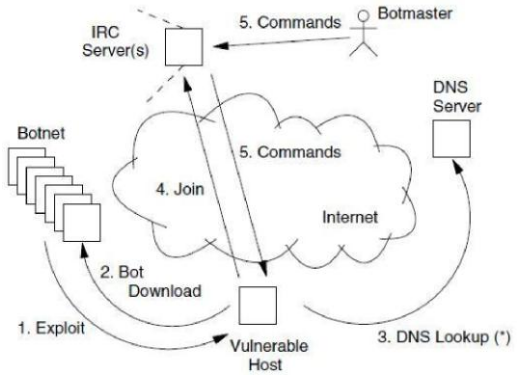


Figure 1: Life Cycle of a Generic Botnet.

Next, we consider the all-important command and control structure of botnets in more detail. Then we discuss other relevant aspects of botnets and related topics.

*B. Command and Control*

For command and control (C&C) of botnets, IRC [9] has proven to be highly successful and has been adopted by the vast majority of botnets [6]. The purpose of an IRC channel is to provide instant messaging and synchronous conferencing. IRC is often used for online chat, audio/video conferencing, and text-based multi-user chat functions. IRC enables a botmaster to easily issue commands to individual bots. Another advantage of using IRC for a botnet is that the command and control traffic is difficult to distinguish from normal IRC usage.

An IRC-based botnet is a centralized approach, since the botmaster uses one server (or a few servers) and is able to communicate directly with all bots. An IRC-based C&C architecture is easy to construct and provides an efficient and effective means for distributing botmaster commands. As illustrated in Fig. 2, a single botmaster can use C&C servers to control a number of bots.

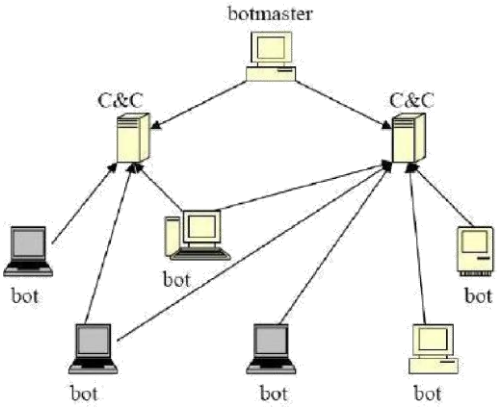


Figure 2: Command and Control Architecture.

Instead of using an IRC channel, a few botnets have employed peer-to-peer (P2P) mechanisms for C&C. In a pure peer-to-peer architecture, any node in the network can act as client or server, or both simultaneously [5]. For a botnet, the advantage of P2P is that there is no single point of failure. Consequently, it is extremely difficult for law enforcement to shut down a P2P botnet—even if the botmaster is taken offline, the botnet may continue to function. However, it is much more difficult to develop a P2P botnet architecture.

In the next section, we consider some examples of recent botnets. First, we discuss a few IRC-based botnets; then we briefly turn our attention to more advanced botnet architectures.

*C. Botnet Examples*

Examples of IRC-based botnets include AgoBot [13], SpyBot [2], GTBot [2], and SDBot [7]. Next, we briefly discuss each of these botnets in turn.

AgoBot is written in C/C++ and, due to its use of standard data structures, it is relatively easy for an attacker to modify or add new functionality. AgoBot is a sophisticated piece of malware that can launch various DoS attacks, harvest sensitive information (traffic sniffing, key logging, searching registry entries, etc.), and can evade detection by patching vulnerabilities, closing back doors, or disabling access to anti-virus sites, among other self-defense techniques. Interestingly, AgoBot is published under a GNU Public License (GPL), which is unusual for malware.

SpyBot, which is an enhanced version of SDBot, is written in C and only has about 3,000 lines of code. In addition to essential C&C structures, SpyBot has a scanning capability (thus, its name), host control functions, and DDoS/flooding attack capabilities. However, SpyBot does not have anywhere near the capability or modularity of Agobot.

The Global Threat Bot (GTBot, also known as Aristotles), can perform DoS attacks, port scanning, and NetBIOS/RPC exploitation. Compared to AgoBot and SpyBot, GTBot only provides limited commands for host control. In addition, a GT bot is only capable of obtaining local system information and can only affect local files.

SDBot's source code is written in C and consists of less than 2,500 lines of code. Its command set and

 

features are similar to those of AgoBot. Although an SDBot has no propagation capabilities and only provides basic functions for host control, attackers seem to like this bot since its commands are easy to extend. SDBot has powerful scanning tools to help it locate potential victims; for example, by using a NetBIOS scanner, SDBot can randomly target systems in any predefined IP range. Since SDBot is able to send ICMP and UDP packets, it can be used for simple flooding attacks.

As mentioned above, P2P botnets are relatively difficult to construct and, consequently, there are fewer examples of such botnets. Recent examples of P2P botnet include Nugache and Storm [18].

Storm is used primarily to propagate spam; at its peak, it was deemed responsible for generating 99% of all spam seen by one large service provider [3, 4]. Storm's effectiveness has been attributed to the following factors [16]:

> Social engineering: It spreads using well-designed email messages.
>
> Use of client-side vulnerabilities: Clicking on a URL in an unsolicited email may be enough to infect a computer.
>
> Obfuscation: The bot uses an effective obfuscated command and control structure overlaid on a P2P network.

Storm also includes a distributed denial of service (DDoS) feature that is triggered based on information gathered from its overlay network [16].

As with any P2P architecture, Nugache has no C&C server to target. For this particular botnet, any bot in the network can become the de-facto botmaster. Nugache also employs encryption and other techniques aimed at hiding its activity.

A hybrid botnet is proposed in the research paper [21], where the following issues are considered.

1. How to generate a robust botnet capable of maintaining control of its remaining bots after a substantial number of its bots have been removed by defenders?
2. How to prevent significant exposure of the network topology when some bots are captured by defenders?
3. How to easily monitor and obtain information on a botnet by its botmaster?
4. How to prevent defenders from detecting bots via their communication patterns?
5. How to take advantage of issues related to a given network?

To the authors' knowledge, no botnet this sophisticated has yet been observed in the wild.

### *D. Infection Methods*

Another important part of a practical botnet design is the means used to infect systems. Details on malware infection methods are beyond the scope of this paper, but we note in passing that the following general approaches may be used by botnets:

1. Exploit client application vulnerabilities (i.e., security bugs) to download and install bot software.
2. Exploit network services such as RPC or MSSQL.
3. Exploit unsecured computers in nearby networks by, for example, finding default passwords, public shares, etc.
4. Spam with malicious code attachments, or with malicious URL links.
5. Trick users into downloading and executing programs.

### *E. Trigger Events*

A trigger event may be used to activate a botnet to perform some malicious activity. For example, a particular date or a certain time of day could serve as a trigger event. Another type of trigger mechanism could be based on a function a user normally performs, such as opening a banking website or executing financial software—a keylogging function, for example, could be tied to such activity.

### *F. Covert Channels*

A covert channel is a communication path not intended as such by a system's designer [19]. Covert channels arise in many situations, particularly within network communication. Covert channels are virtually impossible to eliminate, and in high-security environments, the emphasis is on limiting the capacity of such channels.

In our SocialNetworkingBot, we use Twitter as a covert channel. Our botmaster posts its own tweets, which contain disguised commands that will be correctly interpreted by infected bots. But, these commands appear to be relatively normal tweets and hence they will not generally arouse suspicion. The idea is that since there are a large number of legitimate public tweets, a few C&C tweets will not stand out from the legitimate traffic.

## III. SocialNetworkingBot

In this section, we provide more details on our SocialNetworkingBot application. We cover the SocialNetworkingBot authentication mechanism and the use of Twitter for C&C. We also mention some of the features that increase the stealth of the application.

### *A. Application Overview*

We have created our own application that can fetch tweets, update status, and direct the tweets to other applications and accounts. There is one consumer key and consumer secret that remains constant throughout the life of an application. We have a request token and access token URL. By using these URLs, an application can request the token key and token secret string which are used by our application.

Setting details appear in Fig. 3. The Authorize URL field is used to prompt a user (i.e., botmaster) to manually authorize the application. We also obtain an access token and access token secret string. These are used to connect

to our web application within the botmaster application.

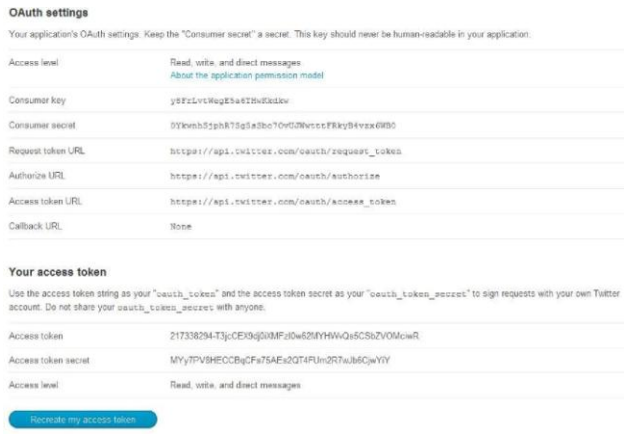


Figure 3: Setting Details.

### *B. Authentication Mechanism*

We use a token keyword and token secret string to authenticate the botmaster to our web application. When we authenticate the application for the first time, the botmaster (manually) obtains an integer PIN. The PIN number is stored and subsequent authentication requests are automated via the OAuth [8] mechanism provided by Twitter. Fig. 4 shows the prompt that the botmaster receives within a browser, while Fig. 5 shows a 7-digit pin number that was generated by the authentication process. Using our SocialNetworkingBot application, tweets are posted to the botmaster's Twitter account. The tweets can be random twitter spam, or they can be used to convey C&C information to bots. In Section IV we discuss various attacks that can result from botmaster tweets.


Figure 4: Authorize the Botmaster.


Figure 5: 7-Digit Pin for Authorization

Note that anyone who has access to the botmaster Twitter account can act as botmaster. Consequently, there could be multiple botmasters acting at various times.

### *C. Keywords*

Each bot has a set of specified keywords that are used to determine what, if any, action should be taken in response to a given botmaster tweet. In our proof of concept application, there are about 300 keywords. To issue a command, the botmaster prepends the "#" symbol to the current daily keyword so that it is in the form of a hashtag. The botmaster appends the actual attack command to the daily hashtag. A small sample of keywords appears in Table 1.

Table 1: Sample Keywords

| Index | Keyword |
|---|---|
| 1 | facebook |
| 2 | hotels |
| 3 | youtube |
| 4 | craigslist |
| 5 | google |
| 6 | yahoo |
| 7 | facebook |
| 8 | myspace |
| 9 | xxx |
| 10 | ??? |
| 11 | walmart |

In our implementation, a new keyword is picked daily based on a predefined set of indexes, such as those in Table 1. For example, suppose that Table 1 is in use and the current daily index is 6. Then, if the botmaster tweets

#yahoo browse http://www.sjsu.edu

each bot obtains the tweet, and based on the hashtag "#yahoo" acts on the command to browse http://www.sjsu.edu. In practice, the actual attack keywords (in this case, browse) and even the arguments (in this example, http://www.sjsu.edu) could easily be obfuscated. Also, in addition to tweets that actually specify attacks, the botmaster can issue any number of inactive tweets, i.e., tweets that do not use the current daily key and, therefore, are not acted on by the bots.

## IV. SocialNetworkingBot Attacks

In this section, we discuss the various attacks that we have implemented in SocialNetworkingBot. Many more attacks are possible—the attacks discussed here are only intended to illustrate some of the many possible features of a Twitter-based (or other social media-based) botnet.

 

*A. Overview of Attacks*

SocialNetworkingBot includes all of the attacks listed in Table 2. However, some of these "attacks" are used only for communication purposes. For example, Fetch User Information gets specific information relevant to the botmaster's control of a bot.

Table 2: SocialNetworkingBot Attacks.

| Attack | Keyword |
|---|---|
| 1 | browse web |
| 2 | capture screenshot |
| 3 | shutdown system |
| 4 | download/upload files |
| 5 | DoS attack |
| 6 | get last update |
| 7 | update status from botmaster |
| 8 | fetch follower info |
| 9 | get NIC details |
| 10 | get MAC address |
| 11 | change mailing addresses |

Examples of malicious activities given in Table 2 include having a web browser open an advertisement promotion, shutting down the system, taking a screenshot of a user's work, and emailing a file or system information to the botmaster. Several of these attacks are briefly explained in the following section.

*B. Selected Attacks*

**Browse a Webpage**: As illustrated in Section III.C, in this attack, the botmaster instructs bots to browse a particular website. This could be used to increase volume (and therefore page rankings) for the specified URL.

**Capture Screenshot**: This attack involves taking a snapshot of user's work. We can then save the screenshot to a location specified by the botmaster. The saved screenshot (or other harvested information) can then be emailed to the botmaster using the send command.

**Shutdown:** In this attack, the botmaster tweets

#keyword shutdown

where "#keyword" is the current daily hashtag. Once the bots parse the shutdown command, a system call is invoked on the victim's machine, the result of which is shown in Fig. 6.


Figure 6: Shutdown System.

**Restart**: This attack is similar to a system shutdown. The only difference here is that we restart the victim's system.

**Fetch Status**: This causes each bot to fetch its last 20 statuses. Using this command, the botmaster can, in effect, replay recent attacks.

**Fetch Follower Information**: This attack returns the numeric ID that Twitter uses to keep track of the botmaster's followers. With these IDs, we can obtain Twitter screen names and profile pictures of all followers of a given bot. This information could conceivably be used by the botmaster to expand the size of the botnet in a viral manner by infecting followers and, subsequently, followers of followers.

**Find NIC Details**: The botmaster can obtain the bot's network interface information. This information is communicated to the botmaster via email.

**Find MAC Addresses**: In this attack, the botmaster can obtain the MAC address of the victim machine. This information is emailed to the botmaster.

**Change Email Address**: In this attack a botmaster posts a tweet with a change command. As a result, the botmaster can change the email address that bots use to send information to the botmaster.

**Execute Commands**: In this attack the botmaster sends a file consisting of commands. After fetching the tweet and parsing the execute command, the bot malware searches for a file named mycommand to run.txt in the victim's system path. If such a file is present, the command (commands) in the file is (are) executed. This attack is very flexible and could cause significant damage, depending on the level of permission that the bot malware has obtained.

*C. Generic Attack*

In addition to the attacks discussed above, we have implemented a somewhat "generic" attack. The purpose of this attack is to enable the botmaster to initiate attacks that were not built into the application at the time it was installed. That is, we can expand the range of possible attacks as new ideas are developed.

In the current implementation, this generic attack is initiated by the run command, which, as with all of the commands discussed above, could easily be changed to something less obvious. The run command accepts a

 

140-character tweet that can contain multiple commands. The resulting token string is treated as a command or series of commands. In our proof of concept implementation, run only has three possible subcommands, namely, checkSystem, NICs, and Screenshot.

The checkSystem option checks for the user's home directory path and emails this information to the botmaster. The NICs command extracts the bot's network interface card details and emails this information to botmaster, while screenshot captures a current screenshot and emails this image to the botmaster.

In the current implementation, the generic run attack simply duplicates attacks available using predefined commands. However, this generic attack could be used to update the botnet with additional attacks after it has been released into the wild. That is, we could use the file download capability to install new scripts and executable files, then use the generic command to control execution of these new attacks.

## V. Experiments

In this section, we briefly discuss some small-scale tests performed with our SocialNetworkingBot application. These experiments were designed to provide a proof of concept while testing the various features of the botnet.

All of our tests were conducted using a network consisting of six bots and a botmaster. These seven systems resided on two physical machines with an additional ve virtual machines installed. The two physical systems had Windows XP and Ubuntu 12.04 (64 bit) Debian Kernel 3.2.0+ installed. The virtual machines included three Windows XP, one Windows 7, and one Ubuntu 11.10 (64 bit) system. For virtualization, both VMWare Player and Oracle Virtual Box were used. This experimental setup is summarized in Table 3.

Table 3: Experimental Setup.

| Role | Operating System |
|---|---|
| Botmaster | Windows XP |
| Bot 1 | Ubuntu 12.04 |
| Bot 2 | Windows 7 |
| Bot 3 | Windows XP |
| Bot 4 | Windows XP |
| Bot 5 | Windows XP |
| Bot 6 | Ubuntu 12.04 |

Fig. 7 shows the botmaster posting commands in the form of tweets to its Twitter account. In this case, the command instructs the bots to browse a specific webpage. Fig. 8 shows a victim's system running the bot code.

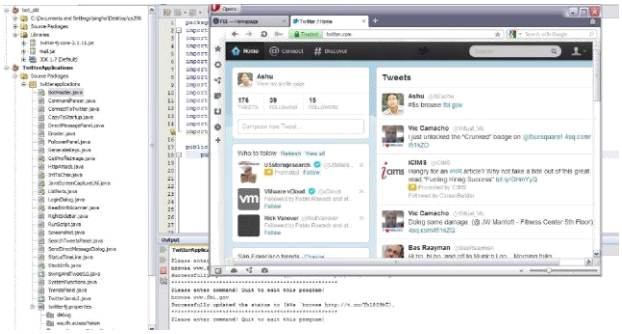

Figure 7: Botmaster in Action.

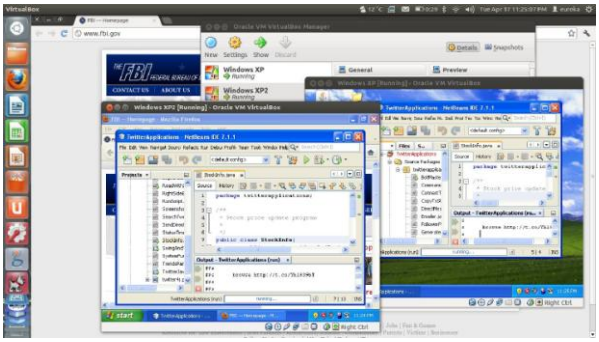

Figure 8: Bots in Action.

For our experiments, the botmaster posted thousands of tweets, of which a few hundred required action by the bots. All tweets were handled correctly by the bots, and the resulting activity was not detected by anti-virus software on the hosts. In addition, Twitter did not detect or prevent any of these activities.

## VI. Conclusions and Future Work

In this paper we discussed SocialNetworkingBot, a proof of concept botnet that uses Twitter for command and control of individual bots. A variety of attacks were implemented, including fetching information from a user's system and denial of service attacks.

The work presented here illustrates the potential for social media-based botnets and highlights the difficulties associated with detecting such activity. Our work is intended to serve as a tool for further research into this challenging, and relatively new, malware problem. For example, our implementation could be used to test various detection strategies, such as those discussed in [10].

## References

[1] B. Lokesh, Covert Botnet implementation and defense against covert botnets, Utah State University, 2009.

[2] P. Barford and V. Yegneswaran, An inside look at botnets, Special Workshop on Malware Detection, Advances in Information Security, Springer 2006 http://pages.cs.wisc.edu/~pb/botnets_final.pdf

 

[3] D. Dittrich and S. Dittrich, P2P as botnet command and control: A deeper insight, International Conference on Malicious and Unwanted Software, 2008 http://staff.washington.edu/dittrich/misc/malware08-dd-final.pdf
[4] S. Gaudin, Storm worm erupts into worst virus attack in 2 years, Information Week, July 24, 2007 http://www.informationweek.com/news/201200849
[5] J. Grizzard, et al, Peer-to-peer botnets: Overview and case study, In Proceedings of Hot Topics in Understanding Botnets (HotBots'07), 2007 http://static.usenix.org/event/hotbots07/tech/full_papers/grizzard/grizzard.pdf
[6] G. Gu, J. Zhang, and W. Lee, BotSniffer: Detecting botnet command and control channels in network traffic, In Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08), San Diego, California
[7] T. Holz, S. Marechal, and F. Raynal, New threats and attacks on the world wide web, IEEE Security & Privacy, 4 (2), pp. 72-75, March/April 2006
[8] Java API http://www.oracle.com/technetwork/java/javamail/javamail143-243221.html
[9] C. Kalt, Internet Relay Chat: Client Protocol, RFC 2812, 2000
[10] E. Kartaltepe, et al, Social-network based botnet command-and-control: Emerging threats and countermeasures, Applied Cryptography and Network Security 8th International Conference (ACNS 2010), J. Zhou and M. Yung (editors), LNCS 6123, pp. 511-528
[11] J. Leyden, Mexican Twitter-controlled botnet unpicked, The Register, September 15, 2010 http://www.theregister.co.uk/2010/09/15/mexican_twitter_botnet/
[12] J. Leyden, Twitter-control botnet mines Bitcoins, The Register, August 3, 2011 http://www.theregister.co.uk/2011/08/03/twitter_controlled_bitcoin_botnet/
[13] L. Liu, et al, Botnet: classi cation, attacks, detection, tracing, and preventive measures, EURASIP Journal on Wireless Communications and Networking, Volume 2009, Article ID 692654
[14] E. Messmer, America's 10 most wanted botnets, Network World, July 22, 2009 http://www.networkworld.com/news/2009/072209-botnets.html
[15] J. Nazario, Twitter-based botnet command channel, The Arbor Networks Security Blog, August 13, 2009 http://ddos.arbornetworks.com/2009/08/twitter-based-botnet-command-channel/
[16] P. Porras, H. Saidi, and V. Yegneswaran, A multi-perspective analysis of the Storm (Peacomm) worm, CSL Technical Note, Computer Science Laboratory, SRI International, October 2007
[17] P. Roberts, Sophisticated attackers now using social net for command and control, ThreatPost, January 27, 2011
[18] B. Schneier, Nugache and Storm http://www.schneier.com/blog/archives/2007/12/the_nugache_wor.html
[19] M. Stamp, Information Security: Principles and Practice, 2nd edition, Wiley, May 2011
[20] Twitter Fan Wiki, Bots http://twitter.pbworks.com/w/page/1779741/Bots/
[21] P. Wang, S. Sparks, and C. Zou, An advanced hybrid peer-to-peer botnet, IEEE Transactions on Dependable and Secure Computing, 7(2), 113-127, April-June 2010

**Ashutosh Singh** received his MS degree in Computer Science from San Jose State University in May 2012. Currently, he works for Oracle in the NAS Protocol Development team. Ashutosh's interests include file systems, storage virtualization, cryptography, and information security.

**Annie H. Toderici** recently received her Master's degree in Computer Science from San Jose State University. She is currently working as a consultant at TCS. Annie's interests include information security, web development, and mobile phone development.

**Kevin Ross** is currently a Master's student in Computer Science at San Jose State University. Kevin also works as a System Administrator for the university.

**Mark Stamp** is Professor of Computer Science at San Jose State University. His research interests include malware, cryptography, other aspects of information security, and applications of machine learning. Professor Stamp has authored (or co-authored) more than 80 research papers and 2 textbooks.

 