SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2015; **00**:1–18
DOI: 10.1002/sec

RESEARCH ARTICLE

Understanding a Prospective Approach to Designing Malicious Social Bots

Yukun He[1,2], Guangyan Zhang[3], Jie Wu[4] and QiangLi[1,2,*]

[1] College of Computer Science and Technology,Jilin University, Changchun, 130012, China.
[2] Knowledge Engineer of Ministry of Education, Jilin University, Changchun, 130012, China.
[3] Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, China.
[4] Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA.

ABSTRACT

The security implications of social bots are evident in consideration of the fact that data sharing and propagation functionality are well integrated with social media sites. Existing social bots primarily use RSS (Really Simple Syndication) and OSN (Online Social Network) APIs to communicate with OSN servers. Researchers have profiled their behaviors well, and have proposed various mechanisms to defend against them. We predict that a web test automation rootkit (WTAR) is a prospective approach for designing malicious social bots. In this paper, we first present the principles of designing WTAR-based social bots. Second, we implement three WTAR-based bot prototypes on Facebook, Twitter, and Weibo. Third, we validate this new threat by analyzing behaviors of the prototypes in a lab environment and on the Internet, and analyzing reports from widely-used antivirus software. Our analyses show that WTAR-based social bots have the following features: (1) they do not connect to OSN directly, and therefore, produce few network flows; (2) they can log in to OSNs easily and perform a variety of social activities; (3) they can mimic the behaviors of a human user on an OSN. Finally, we propose several possible mechanisms in order to defend against WTAR-based social bots.

KEYWORDS

social bot; online social network; social network security; web test automation

***Correspondence**
Qiang Li, College of Computer Science and Technology, Jilin University, Changchun, China.
E-mail: li_qiang@jlu.edu.cn

Received . . .

1. INTRODUCTION

Online social networks (OSNs) such as Facebook, Twitter, and Weibo are widely used on the Internet. They are changing the way people communicate and interact with the Internet [1]. Due to the popularity of OSNs, attackers have begun designing malicious software that targets OSNs. These new social botnet programs have become serious security threats to social networks in recent years. A popular new attack toolkit called RIG is affecting accounts on Twitter, Facebook, and other social media websites. According to Cisco threat researchers[1], the *RIG Exploit Kit* spreads dangerous malware by infiltrating advertising networks and embedding malicious links in ads

[1] http://blogs.cisco.com/security/rig-exploit-kit-strikes-oil/, 12.12.2015.

Prepared using secauth.cls [Version: 2010/06/28 v2.00]

that redirect users to ransom ware, which can lead to the locking-up or deletion of victims' files, or other threats.

In an OSN, a user can control one or many accounts to post messages, connect with people, read status updates from friends, and so on. With new social botnets, an OSN is used as a command and control (C&C) channel. A social bot is an automated software running in the background on a user's computer [2]. It controls an account on a particular OSN, and communicates with the botmaster through posting and receiving messages from the OSN. A social botnet is composed of an army of social bots and can have significant impact on the average user and the OSN itself. Several social bots have already been found on popular OSNs in recent years. Koobface[3], which is considered the most successful social bot, began in 2008 and targets Facebook and Myspace; it remains active today. Another social bot, Nazbot[4], uses Really Simple Syndication (RSS) to receive encrypted tweets from the botmaster account on Twitter. Nazbot decrypts these tweets, which are parsed as bot commands.

In order to understand the behavior of social bots and thus detect them, researchers also propose several social bot prototypes. Kartaltepe *et al.* designed Naz+ [4], which is based on Nazbot. Naz+ checks RSS feeds on CompactSocial, which is a microblogging service that emulates the constraints of Twitter for experimental purposes. Nagaraja *et al.* designed a covert social network botnet called Stegobot [5] that uses image steganography to hide the presence of communications when users share images. Verkamp *et al.* designed an undiscoverable botnet called Facebot [6], and Boshmaf *et al.* designed a social bot for Facebook, called Yazanbot [2]. In addition, Singh designed a social bot for Twitter, called Twitterbot [7].

Most early social bots used RSS or an open application program interface (API) to connect to OSNs. Nazbot, Twebot, and Naz+ use RSS to receive encrypted commands from master accounts on Twitter. Yazanbot uses a Facebook open API to act on Facebook. Twitterbot uses a Twitter open API to receive commands from the master account. However, the number of RSS services has decreased in recent years, and number of open APIs employed by various OSNs is limited. Facebook removed RSS support on November 22, 2011. RSS feeds from Twitter are no longer supported as of March 2013. Google quietly shut down the YouTube subscription RSS feed on May 14, 2014. When using an API for OSNs, developers must register basic profiles. Some OSNs, such as the Chinese Weibo platform require real name authentication for full access to the service or else actions will be limited. In addition, the number of APIs that can be called is limited. For example, when using the Twitter REST API, an OAuth-enabled application can only initiate 350 GET-based requests per hour per access token. The API of Facebook and Weibo are also limited to various degrees. Because of the recent limitations on RSS services and open APIs, attackers may need to find new solutions for communication between social bots and OSNs.

In this paper, we draw attention to a technique that attackers can use to design a social bot, *i.e.,* web test automation (WTA). WTA is widely used to automate web application testing. Using the WTA technique, programs can mimic a normal user to conduct various actions on a website, such as visiting a certain URL, clicking a button, and modifying content. Here we assume the perspective of an OSN attacker who is trying to run a social bot on a benign user host while evading detection. By taking an attacking perspective, we can analyze a new rootkit that attackers may use for building social bots. We first discuss the key challenges in designing a successful social bot. Second, we present the principles of designing WTAR-based social bots. Third, we implement three WTAR-based bot prototypes on Facebook, Twitter, and Weibo. Fourth, we validate this new threat from three perspectives: analyzing prototypes in a lab environment, running the prototypes on the Internet, and analyzing reports from widely used antivirus software. Thus, we hope to provide experience for other security researchers to facilitate better understanding of the depth and scope of the threats posed by this new breed of social bot.

The main contributions of our work are as follows.

(1) We evaluate a latent threat, *i.e.*, a new breed of social bots based on WTARs. Earlier social bots used RSS or open APIs to connect tp OSN platforms. We propose the use of the WTA technique to design a social botnet for the first time. The WTAR-based process produces few network flows and does not require OSN authentication . A WTAR mimics a normal user acting in an OSN as if it were a human being.

(2) We discuss the challenges involved in designing a malicious social bot and propose three different C&C structures for social botnets. In addition, we design a novel

DOI: 10.1002/sec
Prepared using secauth.cls

command hiding method for social bots that can confuse normal users and hide the real C&C channel.

(3) We implement three WTAR-based prototypes, *i.e.*,Fbbot on Facebook, Twbot on Twitter, and Wbbot on Weibo. We analyze the prototypes running in a lab environment and in wide-area networks. In addition, we analyze reports from common antivirus software. We also compare the prototypes to early social bots.

(4) We have shared our WTAR-based prototypes and their source files on Dropboxs [2]. for other researchers to better understand social bots. In our experiments, we use existing tools to implement the prototypes for convenience and efficiency. However, the proposed approach is not limited to these tools and can be implemented with other methods, such as the *WinCom.dll* interface.

The remainder of this paper is organized as follows. Section 2 presents related work. Section 3 discusses the key challenges in designing a successful social bot. Section 4 describes the WTAR design principle, including WTAR enabling factors, an overview of WTAR, C&C structure, C&C procedures, command encryption, host activities, and social network activities. Section 5 gives a WTAR case study. Section 6 validates the WTAR theory from three perspectives, i.e., analysis in a lab environment, running on real social networks, and detection by antivirus software. Section 7 compares WTAR-based social bots to early social bots, describes a possible defense mechanism, and discusses limitations of the mechanism. Conclusions are given in Section 8.

2. RELATED WORK

Our work on WTA rootkits (WTARs) is related to four areas of prior research: WTA, malware, traditional botnets, and social bots.

Novel approaches to WTA are continuously proposed by researchers. Dallmeier *et al.* [8] present WebMate, a tool for automatically generating test cases for web applications. WebMate can automatically explore the functionality of a web application across different browsers and operation systems. WebMate can handle full Web 2.0 functionality and explore complex websites like Facebook. Thummalapenta, *et al.* [9] provide insights into problems in test automation and cross-browser testing. They also developed a tool, called ATA, for test automation. ATA can patch scripts automatically for certain types of applications or environment changes. Google has applied a patent for automatically testing a web application running inside a web browser [10].

Design of malware. Researchers have proposed various malware types to help better understand and defend against the threat posed by new malware. King *et al.* [11] evaluate a new type of malicious software called virtual machine based rootkit (VMBR). They implement two proof-of-concept VMBRs on Linux and Windows. The VMBR installs a virtual machine monitor underneath the target operating system. VMBRs are hard to detect and remove because their state cannot be accessed by software running in the target system. Rieback *et al.* [12] explore the concept of malware for Radio Frequency Identification (RFID) systems. They present RFID malware design principles and examples with the aim to encourage RFID designers to adopt safe coding practices. Embleton *et al.* [13] draw attention to the system management mode based rootkit (SMBR). System management mode (SMM) is an obscure mode on Intel processors used for low-level hardware control. Based on the proof-of-concept SMBR, Embleton *et al.* implement a chipset-level keylogger and a network backdoor. The SMBR hides its memory footprint and requires no changes to the target operation system. The authors aim to better understand the threat posed by malware.

Design of traditional bot. To be well prepared for future attacks, it is not enough to study how to detect and defend against the botnets that have appeared. Researchers have to study advanced botnets that may be developed for future attacks. Wang *et al.* [14] present the design of an advanced hybrid peer-to-peer (P2P) botnet. The hybrid P2P botnet requires no bootstrap procedure and communicates via the peer list in each bot. Only bots with a static global IP address which are accessible from the Internet are in the peer lists. Compared with previous P2P botnets, the hybrid botnet is harder to shut down, monitor, and hijack, according to Wang *et al.* Hua *et al.* [15] present two special C&C mechanisms for mobile botnets. They design an SMS-based flooding algorithm to propagate commands. They also utilize Bluetooth to transmit bot commands when hijacked phones encounter

[2] https://www.dropbox.com/sh/w95r3ivn5khu9xa/AABh62K4oWbMBdFW3A_QAbOaa,12.12.2015.

each other. Zeng *et al.* [16] designed a P2P-structured mobile botnet based on SMS messages. To avoid detection of the botmaster, the botnet has no central servers used to command dissemination. Instead, it uses P2P topology, which allows the botmaster to publish and the bots to search commands in a P2P fashion.

Design of social bot. The rapid growth of OSNs has attracted much attention from both attackers and defenders. To defend against threats posed by social bots, researchers have proposed several social bots that may appear in the future. Jin *et al.* [17] present mutual friend-based attacks on OSNs. They find that by using mutual friend queries, an attacker can launch privacy attacks that identify friends and distant neighbors of targeted users. They analyze various attack structures that can be used to build attack strategies. Boshmaf *et al.* [2] design a traditional web-based botnet and build a prototype, dubbed Yazanbot. The social bot can launch two types of operations: social interaction operations, which are used to read and write content on Facebook, and social structure operations, which are used to alter the social graph, such as by connecting or disconnecting two accounts. It mainly uses two techniques: the OSN API and HTTP request templates. Nagaraja *et al.* [5] propose a new-generation social bot called Stegobot. It spreads via social malware and steals information from its victims. It uses image steganography to hide the presence of communications between bots and the botmaster. Boshmaf *et al.* [18] give an overview of key challenges in defending against social bots. The challenges relate to web automation, online-offline identity binding, and usable security. In this paper, we draw more attention to the challenges of web automation, utilizing the WTA technique to solve the problems of web automation in social bots.

3. KEY CHALLENGES

We present the key challenges faced by attackers in designing successful malicious social bots. The problem of detecting malicious social bots is an arms race and will keep both defenders and attackers busy, depending on their available resources [2]. Our objective is not to solve the severity of the problem, as existing security defenses have proposed many approaches [29] [30] [31] to that end . Instead, we aim to determine the enabling factors that make malicious social bots feasible and the difficulties such bots are required to overcome. Next, we discuss popular detection approaches that social bots have to evade.

If a social botnet is eager for success, it must pay attention to evading existing detection approaches. Based on detection location, existing social bot detection schemes are mainly divided into two categories: detection by social network vendors and local antivirus, i.e., server-side and client-side. According to the detection technique, existing social bot detection schemes are mainly divided into three categories: machine-learning based, abnormal based, and graph based. The strengths and weaknesses of these detection approaches are discussed below.

3.1. Detection by social network vendors

Social network vendors deploy an immune system to detect social bots on the OSN server. These approaches usually collect users' actions from the OSN including statuses, friend list, and comments. Then, they select and calculate several appropriate features from the collected dataset. Finally, they classify the user account, links, or comments into benign or malicious. For example, the FIS (Facebook Immune System) [26], an adversarial learning system, performs real-time checks and classification on every read and write action on Facebook's database. FIS works by employing intelligent software to detect suspicious links and patterns of behavior on Facebook. It is thus not surprising that such an adversarial learning system is rather effective at identifying and blocking spam. Cao Q. *et al.* [27] design and implement a malicious account detection system called SynchroTrap. The system clusters user accounts according to the similarity of their actions and uncovers large groups of malicious accounts. Boshmaf Y. *et al.* [28] design Integro, a scalable defense system that helps OSNs detect fake accounts using a meaningful a user ranking scheme. Social bots, in contrast, are much more deceptive than spam because they are designed to appear normal [29]. To evade detection by the immune system, a social bot can perform actions on the target OSN as a normal user and encrypt the context before posting it. The largest weakness of the immune system is that it can only eliminate malicious data (user accounts, contexts, images, *etc.*) on the OSN. The root cause of these malicious data is the social bot malware, and it remains alive on the user end host. Thus, the immune system deployed by social network

Security Comm. Networks 2015; **00**:1–18
DOI: 10.1002/sec
Prepared using secauth.cls

vendors cannot eliminate social bot malware hidden on the user end host.

3.2. Detection by local antivirus

Compared with detection by social network vendors, detection by local antivirus can provide greater insight into social bots, and more importantly, they can completely eliminate social bot malware if they detect it successfully on the end host. Pieter *et al.* proposed an approach for detecting social botnet communication by monitoring user activity [30]. They assumed that that communication with social media is suspicious if it does originate from human activity. Thus, they measured the causal relationship between network traffic and human activity to detect social bots. Zi Chu *et al.* [31] presented a new detection approach that used behavioral biometrics, primarily mouse and keystroke dynamics, to distinguish between humans and bots. The detection system has two main components: a webpage logger and a server-side classifier. The logger records mouse-movement and keystroke data, and the classifier classifies the operations as human or bot. However, these detection approaches have low detection accuracy and heavy overheads, and are unable to face new variants.

3.3. Machine-learning based detection

Various machine learning approaches such as support vector machine, decision tree, random forest, naive Bayes, and clustering have been proposed for detecting malicious activities on OSNs. Gianluca *et al.* [32] created a large and diverse set of honey-profiles on three large social networking sites and logged the kind of contacts and messages they received. Then, they developed six features and used machine learning techniques to classify spammers and legitimate users. Zhi Yang *et al.* [33] attempted to detect, characterize, and understand Sybil account activity in Renren, an OSN. They applied a support vector machine classifier to the ground truth dataset, identified several behavioral attributes unique to Sybils, and leveraged these attributed to build a measurement-based, real-time Sybil detector. When social bots develop into a large-scale network, machine-learning based detection can hinder the expansion of the social bot network. Machine learningCbased approaches rely heavily on training datasets and selected features.

3.4. Abnormal based detection

A number of recently proposed techniques aim to identify bots automatically in OSNs based on their abnormal behavior. For example, V. Natarajan *et al.* [5] presented an image entropyCbased anomaly detection scheme to detect StegoBot. StegoBot discovers social bot communication through social networks. It hides information in images. The goal of StegoBot is to spread social malware and steal information from target machines. They found that entropy plays a central role in significant changes in images owing to the embedding of secret messages. Christian J. *et al.* [34] presented CoCoSpot, a novel approach for detecting botnet C&C activity based on traffic analysis. Their approach can deal with obfuscated and encrypted C&C protocols, and can fingerprint and recognize botnet C&C. It is not surprising that such an adversarial system is effective at identifying and blocking known social bots. These approaches may need to know their design principle or some features of their fingerprints. However, if new variants display less abnormal behavior, they may evade detection by such adversarial systems.

3.5. Graph-based detection

Graph theoretic techniques are expected to be less effective and more expensive at identifying social bots. Community detection algorithms [37] [38] are deemed to fail because there will be far more fake relationships than social bots [2]. Guanhua Yan *et al.* [35] explored graphic and theoretic techniques that help to effectively monitor the activities of potential botnets in large OSNs. Their work is based on a Twitter dataset of more than 40 million users and 1.4 billion follower relationships. VoteTrust [36] detects false social network accounts using trust-based polling and a voting graph model. It comprises two graphic models: an invitation diagram (directed graphic) and an acceptance graphic (a directed, weighted graphic). The intuition behind this is that each social bot is expected to gradually, but independently, integrate into the online community it targets, resembling the scenario when a new user joins an OSN.

4. DESIGN PRINCIPLE

A successful WTAR botnet must build a robust C&C channel for when WTARs communicate with the target OSNs. Social bots can use the WTA technique to build the C&C channel. When WTAR connects to the target OSN by automatically controlling the browser, the browser window should be hidden in the background. Thus, the user of the victim computer is not aware that the computer has been infected.

In this section, we discuss the design of WTAR. Section 4.1 presents the enabling factors for WTAR. Section 4.2 give an overview of WTAR botnet and describes that how different components of WTAR work cooperatively. Section 4.3 discusses how to build the C&C structure for social botnet. Section 4.4 describes the C&C procedure for WTAR. Section 4.5 presents the algorithm that WTAR uses to encrypt the commands between bots and the botmaster. Section 4.6 explains the host activities that WTAR may launch. Section 4.7 explains the social network activities that WTAR may launch.

4.1. Enabling factors for WTAR

This section gives an overview of web test automation (WTA), describes a case study of WTA, and introduces three popular WTA tools.

Overview of WTA. In the development of software, software testing is an important step to ensure the quality of the final release of the product. To test web applications, many developers turn to a mechanism for automatic testing so that the software quality assurance engineer does not have to manually test a website. Using the WTA technique, engineers can develop software for automating browsers and testing websites. The WTA-based software can automate the filling in of forms, the reading of data from web pages, the clicking of elements on web pages, and the submitting of data to the web server. A case study of WTA and three popular WTA tools are as follows.

A case study for WTA. We introduce software based on WTA that is used for panic-buying train tickets. It is well known that buying train tickets in China is a great challenge, especially during festivals or holidays. The website *http://www.12306.cn* is the only official online ticket sales outlet for China's *Railway Ministry*. Even if you sit in front of the computer waiting to snatch up train tickets as they become available, during the Spring Festival season, you face a likelihood of tickets quickly selling out. To solve this kind of dilemma, programmers have developed various plug-ins for different browsers based on the WTA technique. Browsers that have installed a plug-in programmed to snatch train tickets can monitor the ticketing web site and automatically buy the train tickets. If the booking is successful, the browser will notify you by playing music or popping up an alert window. Thus, people do not need to manually buy train tickets.

Three popular WTA tools. PAMIE, which is short for Python Automated Module for Internet Explorer *(http://pamie.sourceforge.net/)*, is used to automate website testing within the IE browser by using scripting language. PAMIE manipulates IE's document object model (DOM) via the component object model (COM). Quality assurance engineers and developers can use PAMIE to read data, click elements, and fill in forms in IE. It only supports the Python programming language and IE running on Windows. Selenium *(https://code.google.com/p/selenium/)* is a set of software tools with different approaches that support test automation. The tool suite includes Selenium IDE, Selenium WebDriver, Selenium Remote Control, and Selenium Server. Different parts can be used to solve different test automation problems. The most important feature of Selenium is the support for multiple browsers and different operating systems. Watir (Web Application Tsting in Ruby,*http://watir.com/*) is an open source family of Ruby libraries for automating web browsers. It can help developers to write test cases that are easy to read and maintain. Watir can browse the website the same way as normal users do: read page content, fill in forms, click links, and press buttons on the web page. It can be used in multiple browsers on different operating systems. While Watir can only be used in IE on Windows, the Watir WebDriver module supports most browsers. A brief comparison of the three WTA tools is in Table I.

The WTA technique is widely used in testing web applications. What if someone were to use WTA to design social bots? Instead of using RSS or API from OSNs, social bots could use the WTA technique to automatically connect to the OSN server. A social bot based on WTA, which we call WTAR, can launch various social network activities on target OSNs.

The low-level implementation of WTA may also be socket API. The WTA technique may be a collection of socket APIs, control flows, and encryption methods.

DOI: 10.1002/sec
Prepared using secauth.cls

Table I. Comparison of PAMIE, Selenium, and Watir

Name	Supported browsers	Supported OSs	Key features	Supported languages
PAMIE	IE	Windows	simple, flexible	Python
Selenium	Chrome, IE, Firefox, Safari, Opera, HtmlUnit, phantomjs	Windows, Linux, Mac	multifunctional	Java, JavaScript, PHP, Python, Ruby, Perl, C#
Watir	Chrome, Firefox, IE, Opera	Windows, Linux, Mac	simple, flexible	Ruby

Compared with normal sockets, WTA can help the developer access Web sites automatically and conduct various actions more conveniently and effectively.

4.2. Overview of WTAR botnet

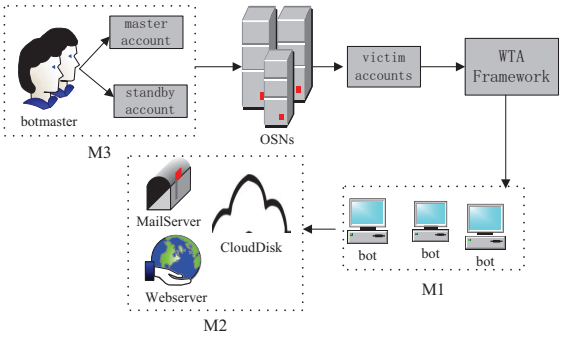


Figure 1. Overview of WTAR botnet

In the process of designing the WTAR botnet, we assume that the WTAR can spread easily. The benign hosts get infected with WTAR in the same way that they get infected with malware. Instead of focusing on the spread of WTAR, we examine the relationships between benign hosts and the WTAR botnet at runtime.

Hosts infected with WTAR become controlled by their botmaster via the WTA technique on OSNs. Figure 1 shows the overview of the WTAR botnet. M1 are the hosts that are infected with WTAR. M2 are servers on the Internet used to store profiles that are stolen by WTAR. Examples of such servers include a normal mail server like Gmail, an http-based web server, or a cloud disk like Dropbox. M3 are the botmasters. The botmaster can use several accounts on the OSNs to send commands to the WTARs. They have a key master account from whence the WTARs get their commands under normal conditions. If the key master account is detected and blocked by the OSN, WTARs can connect to the standby master accounts. Even when the master account has been detected, the WTAR botnet remains. Hence, it is not easy to completely remove the whole WTAR botnet.

A typical procedure of the WTAR botnet at work is as follows. A botmaster from M3 uses the account from OSNs to public commands on the OSNs. The botmaster can connect to OSNs via the OSN website, OSN desktop applications, and even apps on the mobile phone. Then the WTARs in M1 use the WTA technique to connect to the target OSNs via victim accounts. WTARs receive the latest commands from the OSNs and conduct pre-defined activities. The pre-defined commands are mainly divided into two aspects, host activities and social network activities, which will be presented in Section 4.4 and 4.5. If the master commands need a feedback dataset, WTARs can upload the desired data to a server in M2. These servers are set up beforehand by the botmaster, and can be changed at any time.

4.3. C&C structure

To extend the social botnet for large-scale infiltration of OSNs, we introduce three different structures, as shown in Figure 2. The letter M represents the bot master, B represents a compromised social bot, and MB means that the node acts as both a social bot and the bot master.

The first structure, shown in Figure 2(a), is the simplest. It seems similar to an IRC botnet with traditional bot design. In such a structure, all social bots communicate with one bot master. The bot master account on the target OSN is the key factor in social botnet defense. If one social bot has been detected and its communication decrypted, the bot master may be found out and blocked by the OSN vendors. Without the bot master, the other social bots can be destroyed automatically.

Another structure is shown in Figure 2(b). In such a structure, both social bots and the bot master communicate with the target OSN only. M can be any legitimate account on target OSN. They simply publish encrypted commands on the OSN as a normal user. Then, social bots search bot

commands from the target OSN. The social bot account does not need to add the bot master account as friend or visit the home page of the bot master account. The bot master account cannot be found just by one social bot acting alone. If several social bots are detected, the social botnet can still work well. Even in the event that the bot master account is detected and blocked by the OSN vendor, other benign user accounts may upgrade to become the new master account.

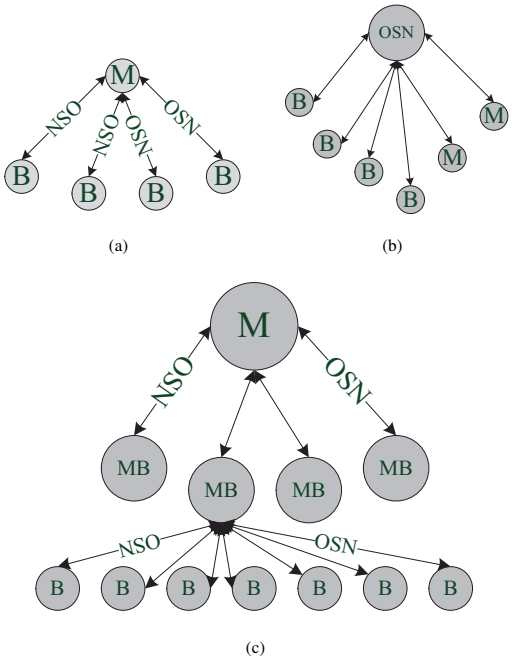


Figure 2. Structure of social botnet

The third structure is shown in Figure 2(c). In this kind of structure, the bot master does not control the social bot directly. It is a hybrid model and has multiple layers. When a bot master publishes a command, the command is first received by several intermediate compromised social bots, which are represented here by MB. Then, the MB nodes resend the command to other social bots in their community. Each MB node may have a different communication model structure and different encryption method or encryption key. Even when one social bot is detected and its communications are decrypted, vendors can find only the corresponding MB node. The other MB nodes remain alive. Thus, the original bot master is hard to detect.

Among the three types of structures, the first one is simple and easy to destroy. However, it is also easy to rebuild, and the communication between a bot and the master is fast and convenient. The third type of structure is the most stable and robust. It is hard to destroy, but it is complex and hard to maintain.

4.4. C&C procedure

To simulate a user browsing an OSN, WTAR must solve the problems of logging into the target OSN, entering the CAPTCHAs, and issuing predefined commands in the C&C procedure.

WTARs can use the WTA technique to control the browser, commanding it to open the login page of the OSN website. Nowadays, a lot of people allow their browser to remember the user name and password and log in automatically. If the password is saved and the user set the browser to automatically log in, a WTAR can easily enter the target OSN in the same way that the user would: by opening the browser and logging in. If the user did not set the browser to store the login information, the WTAR can use a valid username/password combination from the botmaster to log in. The botmaster collects username/password combinations on various OSNs. The WTAR can also use a keylogger to get the username/password combination when the user logs on to the target OSN.

To differentiate automated software from human users, the OSN communities require entering a CAPTCHA before some activities. CAPTCHA, which stands for "Completely Automated Public Turing test to tell Computers and Humans Apart", typically requires the user to decipher and type in a combination of numbers and letters from a distorted image [19]. The CAPTCHA can impede the development of social bots to a degree. However, the reverse CAPTCHA technique has also been explored by lots of researchers [20, 21, 22, 23]. There are also commercial organizations that provide fast-response CAPTCHAs, such as *FastCaptchas*. *FastCaptchas* offers a quick solving service for CAPTCHAs using simple APIs. Social bots could also trick the infected user into solving the CAPTCHA by disguising it as a normal request. For example, Koobface fools the infected user into solving the CAPTCHA by popping up an alert window that warns if you do not solve the CAPTCHA, the system will shut down. However, WTARs using the WTA technique rarely need to solve CAPTCHAs, if the users automatically remember the username/password combinations.

Security Comm. Networks 2015; **00**:1–18
DOI: 10.1002/sec
Prepared using secauth.cls

To ensure the effectiveness of commands from WTARs, we use a command queue and validate the time-effectiveness of commands. The command queue is first in first out (FIFO). A subprocess of WTAR fetches the latest commands from the botmaster at random time intervals. The commands are encrypted content and may be statuses, comments, messages, or profiles of botmaster accounts. The subprocess can filter the executed commands and put the latest and any unexecuted commands at the front of the command queue. The executed commands are put into the command history list. The details of the fetching commands algorithm is shown in Algorithm 1. The algorithm updates the command queue and makes it possible for WTARs to get the latest and unexecuted commands from the botmaster. Another subprocess gets the commands from the tail of the command queue. After parsing the encrypted commands, WTARs launch corresponding attacks.

Algorithm 1 Fetch commands algorithm

Input:

a list used to store the commands that have been received, $CMDHistory$

a FIFO queue used to store unexecuted commands, $CMDQueue$

a random time interval, $RTime$

Output: $CMDQueue$ that has been updated

```
while True do
    CMDList= fetch the latest commands from the botmaster via WTA
    for C in CMDList do
        if C in CMDHistory then
            continue
        else
            Add C into the front of CMDQueue
            Add C into the CMDHistory
        end if
    end for
    Wait for RTime seconds
end while
```

4.5. Command encryption

In order for the C&C of WTAR to avoid detection, the command encryption procedure should be efficient and have good steadiness. We design the commands in three aspects: unique command, encrypted command, and confused command.

To execute the same command twice within a short time without arousing suspicion, each command should be unique. To ensure that each bot command is unique, we add timestamps to the original bot command. The timestamp is just used to distinguish the same commands published at different time.

To make it even more difficult to detect the communications between WTARs and the botmaster, the bot commands should be encrypted. We use Data Encryption Standard (DES) to encrypt the bot command with a predefined key. The DES was developed by an IBM team around 1974, and has been adopted as a national standard of the United States in 1977 [24].

To hide the bot commands on the OSNs and not attract attention, bot commands should look like benign content. We mix URLs from top-visited domains with encrypted commands to make a long URL. The domain of the long URL is in the top-visited domains, and thus, looks benign. The encrypted commands are the parameters of the long URL. A list of top-visited domains can be easily obtained from Alexa *(http://www.alexa.com/topsites)*. The long URL can be shortened using a free URL-shortening service, such as *Google URL Shortener*. If users click the short URL, they will be redirected to the top-visited domain without noticing the commands embedded in the short URL.

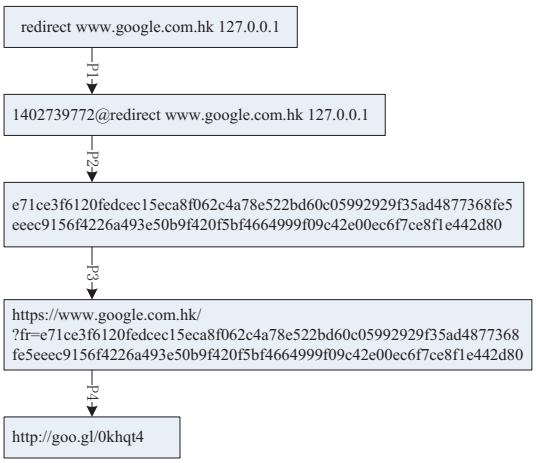


Figure 3. An example of commands encryption

Figure 3 shows an example for commands encryption. It has four procedures (P1, P2, P3, and P4). The original bot command is redirect *www.google.com.hk* 127.0.0.1, which means when the user visits the *www.google.com.hk* website, they will be redirected to 127.0.0.1. P1 mixes the original command with the local time: 1402739772 is the timestamp for 2014-06-14 17:56:12 in China Standard

Time. P2 uses DES to encrypt unique bot commands. P3 mixes the encrypted command with a top-visited domain, which is *www.google.com.hk* in the example. P4 uses the Google URL Shortener to shorten the mixed long URL to get the benign short URL *http://goo.gl/0khqt4*. We describe the generic algorithm for WTAR command encryption in Algorithm 2. The commands-encryption algorithm is a symmetric encryption algorithm. The WTARs on the victim's computer can easily parse the short URLs from the botmaster account, and get the final command.

Algorithm 2 Commands-encryption algorithm

Input:

the plain command, $PCommand$

the encryption key, key

the top 500 domains on the web, $TopURL$

Output: the encrypted command, $ENCommand$

1: $time$ = Get the local system time as a ten-digit number
2: make the command unique $UniqueCommand$ = $time$+$PCommand$
3: encrypt $UniqueCommand$ with DES $DESCommand$ = DES($UniqueCommand$,key)
4: $RURL$= random a URL from $TopURL$
5: $URLCommand$= $RURL$ + $DESCommand$
6: $ShortCommand$= get the short URL from $URLCommand$
7: $ENCommand$= $ShortCommand$
8: **return** $ENCommand$

Command encryption is very important for preventing information leakage. The command message can be disguised as a short encrypted URL. When one or several bots are caught and fail in a social botnet, the encrypted commands can be decrypted only if the corresponding encryption key is used. However, the key is controlled by the bot master and cannot be easily obtained by others.

4.6. Host activities

Just like traditional bots (IRC, HTTP, P2P), social bots can also carry out various malicious activities on the victim hosts. The host activities are basic functions on various OSNs. When designing WTAR, we reduce the number of host activities to avoid antivirus software on the local host from detecting it. We design only basic host activities for WTAR. These activities are the same, even though the OSNs are different. The botmaster of WTAR can send host commands to various OSNs to order WTAR to carry out corresponding actions.

Table II. Host commands

Name	Description
getNetInfo	get the local network information
getVersion	get the system version of the victim
visit	force the browser to open a URL
redirect	rebind the domain and IP
download	download data from the Internet
upload	upload data to the botmaster
exe	execute a program on the host
timeExe	execute at the predefined time

Table II shows a possible host command list. The *getNetInfo* command helps the botmaster obtain the basic network information of the victim, including the MAC address, IP address, USERNAME, etc. The *getVersion* command helps the botmaster obtain the system version of the victim. The *visit* command orders the browser on the victim host to open a predefined URL. This function can lead to the spread of spam. Advertising service providers can use this function to widely promote their products. The *redirect* command rebinds the domain and IP on the victim. When a user of the victim host browses a normal website, they may actually be on a fake website because the real IP of the original domain has been polluted. This function can be easily used to carry out commercial fraud. The *download* command orders WTAR to download data from the Internet. The data can be WTAR modules that need to be updated, or other malware. The *upload* command orders WTAR to upload data from the victim to servers controlled by the botmaster. The data can be personal information or configuration files. The *exe* command orders WTAR to run a program on the victim host. The program can be a DOS command or executable software. The *timeExe* command orders WTAR to run a program on the victim at a certain time, which can lead to timing attacks.

The host activities are not limited to the eight shown in Table II. The number of host activities is extensible by updating the host activities module of WTAR.

4.7. Social network activities

The special social network activity for social bots makes them different from traditional bots. Social network activity represents a social action (*e.g.*, +1, comment, vote, share, favorite) that references content on OSNs. WTAR uses the WTA technique to compel an OSN account on the victim host to carry out social network activities. These activities can change according to the

Security Comm. Networks 2015; **00**:1–18
DOI: 10.1002/sec
Prepared using secauth.cls

OSNs. The frameworks of different OSNs are different, and the operations on the OSNs are different. For example, the messages published on OSNs are called "statuses" on Facebook, "tweets" on Twitter, and "weibos" on Weibo. Hence, we need to design modules for different OSNs and make social activities exclusive to particular OSNs.

Table III. Social commands

Name	Description
postStatus	Publish a status, a message, or a tweet on the target OSN
postComment	Post a pre-defined comment to a certain status on the target OSN
addFriend	Send a friend request to a certain OSN account or focus on an account
addLike	Like a status or favorite a message on the target OSN
autoAdd	Automatically add friends on the target OSN

Table III shows a basic social commands list for WTAR. The *postStatus* command orders WTAR to use the compromised account to publish a certain message on the target OSN. The message may be a status on Facebook, a tweet on Twitter, or a weibo on Weibo. The function can be used to disperse fake news and spread malicious software. The *postComment* command orders WTAR to publish a comment on a certain message. Thus, the number of comments on the message increases, enhancing the influence of the message. This function can be used to guide the direction of public opinion. The *addFriend* command orders WTAR to send a friend request to a certain OSN account. This function can enlarge the community of the whole social botnet and lead to more "zombie fans", which are artificial followers. The *addLike* command orders WTAR to like (+1, favorite, vote) a message. This function can have a major impact on the direction of public opinion. The *autoAdd* command orders WTAR to automatically add friends who are friends' friends of the compromised account. This function can expand the WTAR botnet.

The social commands are not limited to the five shown in Table III. As with host activities, the number of social activities is extensible by updating the social activities module of WTAR.

5. CASE STUDY

To explore the threat, we implement three proofs-of-concept prototypes for WTAR. Our prototypes are Wbbot, Fbbot, and Twbot on the OSNs Weibo, Facebook, and Twitter, respectively. They all implement the structure described in Section 4. The only difference between them is the social network activity.

Our proof-of-concept WTARs can run on both Windows XP and Windows 7. The WTA technique of our WTARs is implemented with PAMIE, which is known to be simple and flexible. We can easily control the DOM of IE through an API from PAMIE. With simple scripts, WTAR can order IE to carry out various social network activities, such as updating a status on Facebook. First, Fbbot finds the form element for posting statuses. Second, Fbbot fills in the form by setting the value attribute of the form with predefined content. Third, Fbbot finds the button element for posting statuses. Finally, Fbbot clicks the *"post"* button element on the Facebook page. The whole process runs in the background without attracting the user's attention. Activity on other OSNs can be implemented in the same way.

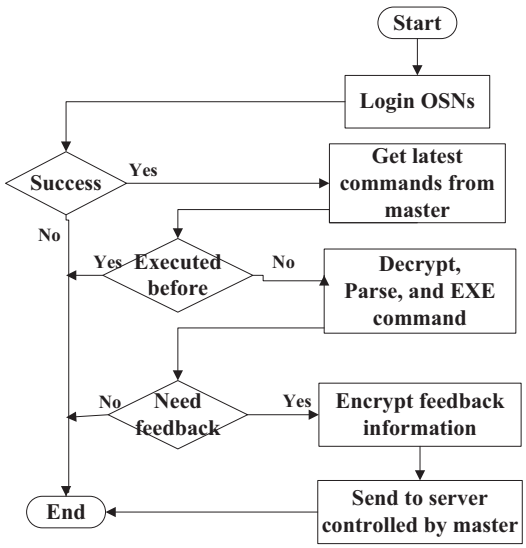


Figure 4. Control flow of the prototypes

We illustrate the basic control flow of the prototypes in Figure 4. The control flow is the same for all three. We can easily extend the prototypes by modifying the social network module to adapt to other OSNs. WTAR first tries to use IE to log into the OSN with a cookie from the

local user account. If the login fails, WTAR will suspend for a random time. If the login succeeds, WTAR will try to get the latest messages posted by the botmaster. If WTAR gets a message, it will check whether the message has been received before. If WTAR has not npreviously received the message, it will try to decrypt the message and then parse the command hiding in the message. If the command needs to feed information back to the botmaster, it will first execute the command and then encrypt the information that the botmaster needs. Then it will send the stolen information to the server, being controlled by the botmaster via email, open API, or simple http request. If the command does not need feedback, WTAR will finish the current job and wait for another command.

In the login process of the WTAR in Figure 4, we assume that the user of the victim host uses IE to browse the OSN and has set the browser to automatically remember the username/password combinations. We make this assumption based on two facts: IE is one of the most popular browsers, and automatic login is widely used in many OSN websites, such as Facebook, Twitter, MySpace, LinkedIn, and Weibo. If the target OSN requires to solve a CAPTCHA, DeCaptcher (http://de-captcher.com) libraries were used to enhance our prototypes. The server used to collect stolen data is built on Dropbox, which is a cloud disk. In the feedback process, our prototypes use open API to upload and download files from Dropbox.

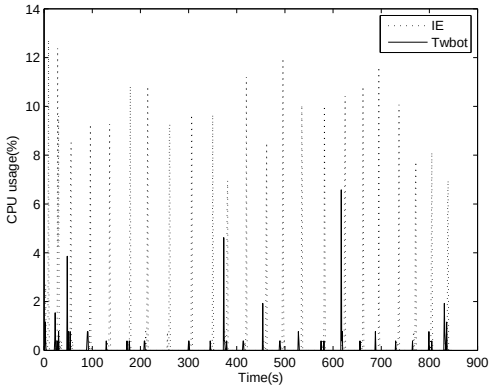
The WTA technique is not limited to the framework described above. PAMIE, the framework used in our designed prototypes, is not complex. The packaged executable file of our prototypes is small and can run on any Windows XP or Windows 7 system without any other WTA environment. It is a malicious file only on the end host. Moreover, the prototypes can be injected into other benign applications. We believe other more robust social bots can be implemented with our proposed designing principle. While our subject is not to design a perfect social bot, we hope to help researchers understand social bots from the perspective of attackers and propose more effective approaches for social bot detection. We have shared the prototypes via Dropbox.

6. VALIDATION

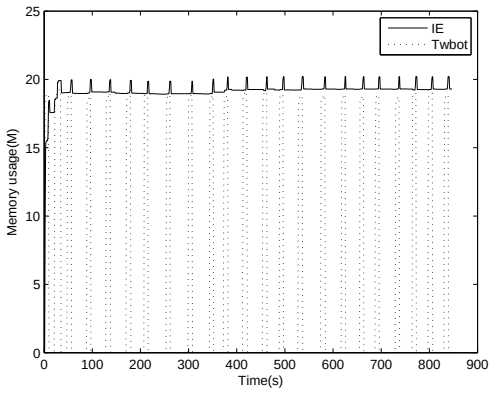
In this section, we validate the threat from WTARs in three aspects. Section 5.1 discusses the experiment in the lab. Section 5.2 discusses the experiment conducted on a real network. Section 5.3 shows the antivirus reports regarding the three prototypes.

6.1. Experiments in the lab

To analyze the three prototypes on a host, we capture the system API and monitor the overhead in the lab. In the experiment, a controllable local network is built. Windows 7 systems are installed on user hosts. They have been equipped with quad core 2.40GHz CPU and 2G RAM. We run Wbbot, Fbbot, and Twbot separately.



(a) CPU usage of Twbot and IE



(b) Memory usage of Twbot and IE

Figure 5. Overhead of Twbot

Overhead. We run the three prototypes on the host and use the master account to order WTARs to execute all the

Security Comm. Networks 2015; **00**:1–18
DOI: 10.1002/sec
Prepared using secauth.cls

commands. We find that WTARs have a little impact on the performance of the victim host. For example, Twbot is as shown in Figure 5. Twbot runs on our victim host for about 15 minutes and executes all the bot commands. Because Twbot compels IE to visit the OSN of Twitter, both Twbot and IE have an impact on the victim host, even though IE is a normal benign application and runs in the background.

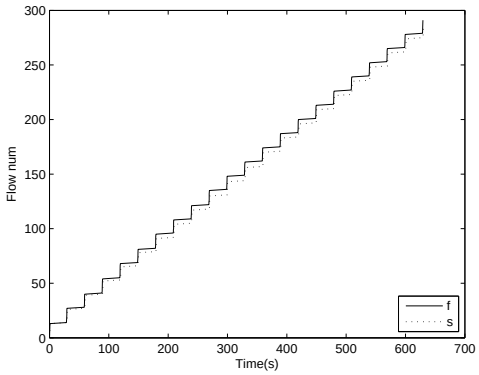
Figure 5(a) shows the CPU usage of Twbot and IE. CPU usage of both Twbot and IE changes cyclically. In about the first 100 seconds, CPU usage changes frequently and is relatively high, because Twbot is receiving commands from the botmaster and carrying out corresponding actions. However, when there is no command to execute, Twbot has a low CPU usage. Only when Twbot tries to update the latest commands does the CPU usage of Twbot increase. When IE is carrying out social network activities, its CPU usage increases.

Figure 5(b) describes the memory usage of Twbot and IE. Throughout the whole monitoring process, memory usage of both Twbot and IE changes cyclically, because Twbot needs to refresh the latest commands from the botmaster every few minutes. When Twbot starts to run, its memory usage is relatively low. However, after a while, its memory usage is smooth and steady. Its memory usage on the host is between 18(Mbytes) and 10(Mbytes). When there is a social command to execute for Twbot, IE is controlled and runs for a short time to finish the command. After the command finishes, IE exits and the memory usage for IE drops to zero.

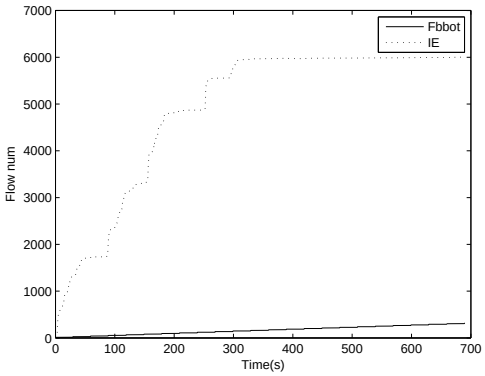
Table IV. Statistical data for system call

API	Wbbot	Fbbot	Twbot
CreateFile	1153	1235	805
RegOpenKey	656	584	548
RegQueryValue	363	284	366
CloseFile	355	349	399
RegQueryKey	275	194	221
RegCloseKey	263	282	195
ReadFile	221	258	242
QueryDirectory	166	146	165
QueryStandard-InformationFile	116	110	102

System call. We use *Process Monitor* to monitor the system calls that these prototypes have called when they are running. Table IV lists the statistical data for system calls over over a period of about ten minutes. We know that *CreateFile* is the most frequent. We analyze the system call sequence of the prototypes according to execution time. For example, if α and β are two system calls, Fbbot calls α at $t1$ and then calls β at $t2$; $t1$ is earlier than $t2$, so the system call sequence should be $\alpha \rightarrow \beta$. We found that the following API sequence occurs frequently: *QueryBasicInformationFile* $\rightarrow$ *CloseFile* $\rightarrow$ *CreateFile* $\rightarrow$ *CreateFileMapping* $\rightarrow$ *QueryStandardInformationFile* $\rightarrow$ *CreateFileMapping* $\rightarrow$ *CloseFile* $\rightarrow$ *CreateFile*. This discovery can be used to detect this breed of WTARs.



(a) Flow statics on different occasions



(b) Flow statics for Fbbot and IE

Figure 6. Flow statistics for Fbbot

Network flow. We use the *Network Monitor* to capture the network flows which these prototypes have produced. The total number of network flows is calculated over time. Through analyzing the network flow dataset, we found that the processes of WTAR-based bot produce a few network flows. When the same prototype runs the same commands in different occasions, the increases in flow rate for them are very similar. Figure 6 shows a ten-minute log, and the rate at which flow increased over time. In the figure, the start time is corrected. Time is plotted on the x-axis, and the total flow number is on the y-axis. Figure 6(a) shows the

rate of increase for Fbbot on two different occasions. f is the first occasion, and s is the second occasion. Both times, Fbbot runs the same botmaster command and produces at almost the same rate of increase. Figure 6(b) is the increase in flow rate for Fbbot and IE. IE connects to the OSN and preforms various social network activities. IE produces far more network flows than Fbbot does .

6.2. Real world experiments

To validate the real threat of WTARs, we operated ten prototypes of Fbbot on Facebook for two weeks. Prior researchers also use this mechanism to design malicious applications. Wang *et al.* [25] presented a novel attack method that allows attackers to reliably hide malicious behavior in an app on iOS devices. Once the app was published and installed on an end-user's device, it can be required to carry out the intended attacks. Yazan *et al.* [2] operated their social botnet on Facebook and collected data about user behavior.

We create ten prototypes of Fbbot and a single bot master, all of which are separately run on different hosts in our laboratory. We use the master account to send *autoAdd* command. According to previous survey, we decided to send 25 friendship requests per Fbbot per day in order to avoid detecting by FIS (Facebook Immune System). We order each Fbbot send 350 friendship requests. Each Fbbot sends 1 to 3 friendship requests every 20 to 40 minutes. And the actual number of friendship request and time interval is randomly generated. The friendship requests are all from their extended neighborhoods which are their friends' friends. When a Fbbot has sent 25 friendship requests, it would stop sending friendship request and keep silent. For most of time, the Fbbot was idle on the end host.

We kept the Fbbot botnet running for 14 days and monitoring the status of the requests every day. During that time, the ten Fbbots sent 3500 friendship requests. Finally, 1017 requests were accepted with a maximal acceptance rate of 29.1%, as shown in figure 7. In particular, acceptance rate tends to stable in the seventh day after the requests being sent. After a large-scale infiltration, the bot master account can harvest large amounts of publicly inaccessible privacy information of users, such as phone number, email address and so on. Our investigation suggests that WTAR-based social bot can be a threat to the OSN of Facebook, and other OSNs as well.

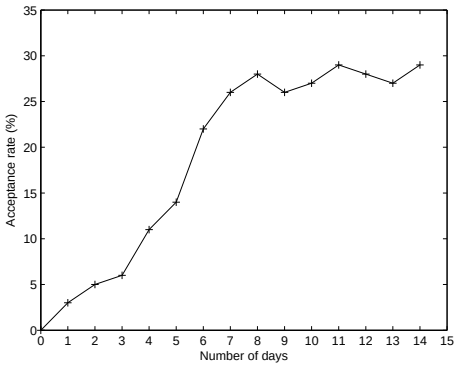


Figure 7. Average acceptance rate of the infiltrated users

6.3. Reports from antivirus

Table V. File information for the three WTARs (%)

Name	Wbbot	Fbbot	Twbot
UPX compressed Win32 Executable	42.3	30.6	42.3
Win32 EXE Yoda's Crypter	36.7	26.5	36.7
Win32 Dynamic Link Library	9.1	6.5	9.1
Win32 Executable	6.2	32.1	6.2
Generic Win/DOS Executable	2.7	-	2.7

To validate the robustness of our prototypes, we analyze them by using the well-known online file scan services *VirusTotal* and *Anubis*. We share the analysis result on *Dropbox*.

The results from virustotal show that only a few antivirus programs report that our WTARs are malware. In 54 scans, only two give the detection result that the three prototypes are all *Trojans*. *Antiy AVL* reports that the prototypes are *Trojan/Win32. Mulo* and *NANO Antivirus* reports them as *Trojan.Win32.Rozena.cwyxkn*. However, other top popular antivirus programs, such as *McAfee*, *ESET NOD32*, and *AVG*, report that our prototypes are benign applications. We also analyze a simple executable file compiled by Python. It is only a "Hello world!" program, but *Antiy AVL* and *NANO Antivirus* detect it as *Trojan*, too. So we believe our WTARs have a very good robustness. Table V shows the file information for our prototypes. *UPX compressed Win32 Executable* and *Win32 EXE Yoda's Crypter* are the main components. This

Security Comm. Networks 2015; **00**:1–18 © 2015 John Wiley & Sons, Ltd.
DOI: 10.1002/sec
Prepared using secauth.cls

feature can be used as evidence to detect WTAR-based social bots.

7. PROFILING

In this section, we compare our WTARs with prior social bots, discuss the defense mechanism for WTAR, and describe the limitations.

7.1. Comparison

Comparison with Nazbot. Nazbot is a successful social bot on Twitter. It uses an account named upd4t3 owned by the botmaster on Twitter to receive commands. Nazbot first makes an HTTP GET request to *upd4t3's* RSS. Twitter then returns an RSS feed containing base64-encoded text. Then Nazbot decodes the encoded text and gets the real URLs from bit.ly URLs. The short bit.ly URL redirects to a malicious Zip file on an independent server. Then Nazbot downloads the malicious Zip file as a payload, unzips the payload, and executes it. Finally, the payload steals the user's information and sends it back to a server controlled by the botmaster.

The key features of Nazbot are RSS and tweets encoded by base-64. However, Twitter does not provide RSS services anymore, and tweets encoded with base-64 can be easily decoded. What's more, if the master accounts publish a base-64 encoded tweet directly, the tweet can easily draw other users' attention because the tweets seem like spam. Thus, the botmaster account may be flagged as a malicious account. Our WTARs use the popular WTA technique to carry out actions on OSNs. We use the DES algorithm to encrypt the message on OSN. The final messages hiding the command just look like a normal short URL. We believe our WTARs are more robust than Nazbot.

Comparison with Yazanbot. Yazanbot is a proof-of-concept Facebook social bot designed by Boshmaf *et al.* [2]. It has two main components: a profile on Facebook and the software. It can conduct two types of operations: social interaction operations that are used to read and write content on Facebook, and social structure operations that are used to alter the social graph, such as by connecting or disconnecting two accounts. The botmaster can send six commands. The cluster command orders the social bot account to connect to other social bots. The *rand_connect* command orders the social bot accounts to connect randomly to non-botmaster-owned accounts. The Decluster command orders a social bot account to disconnect from all other social bot accounts. The *Crawlextneighborhood* command orders a social bot to find its neighborhood's neighbors and return their profiles as botcargo. The *Mutualconnect* command orders the social bot to connect with the accounts in botcargo. The *Harvestdata* command orders a social bot to return all accessible information. Yazanbot mainly uses two techniques: the Facebook API and HTTP request templates. It randomly updates its status with random quotes and blurbs provided by *iheartQuotes*.

Yazanbot draws more attention to social network activities. However, our WTARs can carry out both host and social network activities. Yazanbot uses the Facebook API to communicate with Facebook, but the Facebook API has a lot of limitations, such as app-level and user-level limiting. Our WTARs use the WTA technique to connect with OSNs in the same way that normal users act. Hence, there are no such limitations. Besides, our WTARs can be easily adapted to other OSNs by modifying only the social network module.

7.2. Countermeasures

In this section, we discuss possible countermeasures against WTAR-based social bot. Server-side defense countermeasures can capture activities on certain OSNs from the whole social botnet. They could analyze the big dataset to distinguish the infected accounts from normal user accounts. However, social bots which control these infected accounts remain on the user host. Client-side defense countermeasures can monitor the behaviors on the user host. They can analyze the behavior dataset to find out the social bot process on the victim. However, the social bot may update its behaviors, and the social botnet is still there. To protect the user host and OSN from WTAR-based social bot, we prefer the integrated server-client defense mechanism.

On the server-side, we can use the page-view graph. All WTAR-based social bots need to refresh the latest commands from the master account in a random time interval. The social bot may have a very large page-view number on an OSN page from a botmaster account, for example, the home page of master. Thus, the page-view graph can be drawn on the server-side, and used to detect the social bot account, even the master account. On the

client-side, three factors should be considered. (1)OSN login. The first step for WTARs to perform activities on OSNs is to log in to the OSNs. Thus, new countermeasures should prevent the abuse of local cookies and sessions. The automated login of OSN should also be checked to prevent WTARs. (2)Process correlation. WTARs may control the popular browsers on the user host to perform activities on OSNs. The WTARs themselves do not directly connect to the target OSN. They produce a few network flows. Considering this fact, we should consider the correlation of different processes. (3)Self-Concealing. WTARs use stealth mechanisms to hide themselves. Generally, this includes the processes of WTAR's lack of Graphical User Interface (GUI) and Human Computer Interaction (HCI). This can be considered as cofactors for detecting the processes of WTAR.

7.3. Limitations

Our proof-of-concept social bot has a number of implementational limitations. We focus on two of them: bootstrap and large data test in the real Internet.

We do not pay much attention to the bootstrap phrase of WTAR. We focus on the C&C, command encryption, host activities, and social network activities. Thus, our prototypes should be run manually, or the threat of WTAR will be killed automatically if the host has been shut down. However, we believe WTARs can be easily extended if good bootstrap mechanisms are used.

We have not carried out a large data test in the real network. Our prototypes only carry out a few activities in the Internet. We only collect a 14-day data set from Fbbot on Facebook and only *autoAdd* command is verified. The finally acceptance rate of friendship requests is not high. But we believe that if the WTARs have a better promotion, and the base number of victim hosts is large enough, WTARs can be used to build up a large social botnet.

8. CONCLUSION

Social bots have become a security threat to both users and service providers of OSNs. Prior social bots mainly used RSS or OSN APIs to communicate with OSN. Both defenders and attackers have developed various social bots for different purposes. The details of these social bots can be different, but the ways that these social bots communicate with OSNs are similar. To defend against social bots, the OSN service providers also take several defense mechanisms. Some OSN services do not provide RSS services anymore. OSN service providers also limit the use of open OSN API.

In this paper, we assume the perspective of the social network attacker, who is trying to design effective social bots that target OSNs. By assuming this perspective, we hope to help defenders understand and defend against the threat posed by a new class of social bots. We present a novel breed of social bot called WTAR. It uses the WTA technique to communicate with an OSN, and can launch various attacks on OSNs.

First, we discuss the key challenges involved in designing a malicious social bot. Second, we present how to design the WTAR. We describe the enabling factors for WTAR, C&C structure of social botnet, C&C procedure of WTAR, command encryption, host activities, and social network activities. Third, we implement three prototypes to target Facebook, Twitter, and Weibo. Fourth, we validate our concepts of WTAR in three aspects: (1) we analyze the system call and overhead of our prototypes in a lab environment; (2) we validate our prototypes in the real network and monitor acceptance rate of friendshp requests sent by WTAR; (3) we also analyse the reports from popular antivirus software. Finally, we compare it with prior social bots, discuss the possible defense mechanism for WTAR, and describe the limitations of our WTAR.

We believe that social bots on OSNs represent only one of the many emerging cyber threats. The first step to defend against such a threat is to understand how social bots are designed and how they work. With the experience of designing WTAR, we hope to help defenders propose novel approaches to detect social bots in the future.

9. ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under Grant No. 61170265, 61472162.

DOI: 10.1002/sec
Prepared using secauth.cls

REFERENCES

1. J. Jiang, C. Wilson, X. Wang, W. Sha, P. Huang, Y. Dai, B. Y. Zhao, Understanding latent interactions in online social networks, ACM Transactions on the Web (TWEB) 7 (4) (2013) 18.
2. Y. Boshmaf, I. Muslukhov, K. Beznosov, M. Ripeanu, Design and analysis of a social botnet, Computer Networks 57 (2) (2013) 556C578.
3. Thomas K, Nicol D M. The Koobface botnet and the rise of social malware[C]//Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on. IEEE, 2010: 63-70.
4. E. J. Kartaltepe, J. A. Morales, S. Xu, R. Sandhu, Social network-based botnet command-and-control: emerging threats and countermeasures, in: Applied Cryptography and Network Security, Springer, 2010, pp. 511C528.
5. S. Nagaraja, A. Houmansadr, P. Piyawongwisal, V. Singh, P. Agarwal, N. Borisov, Stegobot: a covert social network botnet, in: Information Hiding, Springer, 2011, pp. 299C313.
6. J.-P. Verkamp, P. Malshe, M. Gupta, C. W. Dunn, Facebot: An undiscoverable botnet based on treasure hunting social networks.
7. A. Singh, A. H. Toderici, K. Ross, M. Stamp, Social networking for botnet command and control, International Journal of Computer Network & Information Security 5 (6).
8. V. Dallmeier, B. Pohl, M. Burger, M. Mirold, A. Zeller, Webmate: Web application test generation in the real world, in: Software Testing, Verification and Validation Workshops (ICSTW), 2014 IEEE Seventh International Conference on, IEEE, 2014, pp. 413C418.
9. S. Thummalapenta, P. Devaki, S. Sinha, S. Chandra, S. Gnanasundaram, D. D. Nagaraj, S. Sathishkumar, Efficient and change-resilient test automation: an industrial case study, in: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, 2013, pp. 1002C1011.
10. Lee T H, Look G, Zhang H, et al. Automatically testing a web application that has independent display trees: U.S. Patent 8,572,505[P]. 2013-10-29.
11. S. T. King, P. M. Chen, Subvirt: Implementing malware with virtual machines, in: Security and Privacy, 2006 IEEE Symposium on, IEEE, 2006, pp. 14Cpp.
12. M. R. Rieback, P. N. Simpson, B. Crispo, A. S. Tanenbaum, Rfid malware: Design principles and examples, Pervasive and mobile computing 2 (4) (2006) 405C426.
13. S. Embleton, S. Sparks, C. C. Zou, Smm rootkit: a new breed of os independent malware, Security and Communication Networks 6 (12) (2013) 1590C1605.
14. P. Wang, S. Sparks, C. C. Zou, An advanced hybrid peer-to-peer botnet, Dependable and Secure Computing, IEEE Transactions on 7 (2) (2010) 113C127.
15. J. Hua, K. Sakurai, Botnet command and control based on short message service and human mobility, Computer Networks 57 (2) (2013) 579C597.
16. Y. Zeng, K. G. Shin, X. Hu, Design of sms commanded-and-controlled and p2p-structured mobile botnets, in: Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks, ACM, 2012, pp. 137C148.
17. L. Jin, J. B. Joshi, M. Anwar, Mutual-friend based attacks in social network systems, Computers & security 37 (2013) 15C30.
18. Y. Boshmaf, I. Muslukhov, K. Beznosov, M. Ripeanu, Key challenges in defending against malicious socialbots, in: Proceedings of the 5th USENIX Conference on Large-scale Exploits and Emergent Threats, LEET, Vol. 12, 2012.
19. B. Zhu, J. Yan, G. Bao, M. Mao, N. Xu, Captcha as graphical passwords–a new security primitive based on hard ai problems, Information Forensics and Security 9 (6).
20. M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, S. Savage, Re: Captchasunderstanding captcha-solving services in an economic context., in: USENIX Security Symposium, Vol. 10, 2010, pp. 4C1.
21. E. Bursztein, S. Bethard, Decaptcha: breaking 75% of ebay audio captchas, in: Proceedings of the 3rd USENIX conference on Offensive technologies, USENIX Association, 2009, p. 8.
22. E. Bursztein, R. Beauxis, H. Paskov, D. Perito, C. Fabry, J. Mitchell, The failure of noise-based noncontinuous audio captchas, in: Security and Privacy (SP), 2011 IEEE Symposium on, IEEE, 2011,

pp. 19C31.

23. P. Golle, Machine learning attacks against the asirra captcha, in: Proceedings of the 15th ACM conference on Computer and communications security, ACM, 2008, pp. 535C542.
24. D. Coppersmith, The Data Encryption Standard (DES) and its strength against attacks, IBM journal of research and development 38 (3) (1994) 243C250.
25. T. Wang, K. Lu, L. Lu, S. Chung, W. Lee, Jekyll on ios: When benign apps become evil., in: Usenix Security, Vol. 13, 2013.
26. Stein T, Chen E, Mangla K. Facebook immune system[C]//Proceedings of the 4th Workshop on Social Network Systems. ACM, 2011: 8.
27. Cao Q, Yang X, Yu J, et al. Uncovering Large Groups of Active Malicious Accounts in Online Social Networks[C]//Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security(CCS). ACM, 2014: 477-488.
28. Boshmaf Y, Logothetis D, Siganos G, et al. Integro: Leveraging victim prediction for robust fake account detection in OSNs[C]//Proc. of Network and Distributed System Security Symposium (NDSS15), San Diego, CA, February 2015.
29. Wagner C, Mitter S, Korner C, et al. When social bots attack: Modeling susceptibility of users in online social networks[J]. Making Sense of Microposts (MSM2012), 2012: 2.
30. Burghouwt P, Spruit M, Sips H. Towards detection of botnet communication through social media by monitoring user activity[M]//Information Systems Security. Springer Berlin Heidelberg, 2011: 131-143.
31. Chu Z, Gianvecchio S, Koehl A, et al. Blog or block: Detecting blog bots through behavioral biometrics[J]. Computer Networks, 2013, 57(3): 634-646.
32. Stringhini G, Kruegel C, Vigna G. Detecting spammers on social networks[C]//Proceedings of the 26th Annual Computer Security Applications Conference. ACM, 2010: 1-9.
33. Yang Z, Wilson C, Wang X, et al. Uncovering social network sybils in the wild[J]. ACM Transactions on Knowledge Discovery from Data (TKDD), 2014, 8(1): 2.
34. Dietrich C J, Rossow C, Pohlmann N. CoCoSpot: Clustering and recognizing botnet command and control channels using traffic analysis[J]. Computer Networks, 2013, 57(2): 475-486.
35. Yan G. Peri-Watchdog: Hunting for hidden botnets in the periphery of online social networks[J]. Computer Networks, 2013, 57(2): 540-555.
36. Xue J, Yang Z, Yang X, et al. VoteTrust: leveraging friend invitation graph to defend against social network Sybils[C]//INFOCOM, 2013 Proceedings IEEE. IEEE, 2013: 2400-2408.
37. Viswanath B, Post A, Gummadi K P, et al. An analysis of social network-based sybil defenses[J]. ACM SIGCOMM Computer Communication Review, 2011, 41(4): 363-374.
38. Fortunato S. Community detection in graphs[J]. Physics Reports, 2010, 486(3): 75-174.

DOI: 10.1002/sec
Prepared using secauth.cls