


Understanding socialbot behavior on end hosts

*International Journal of Distributed
Sensor Networks*
2017, Vol. 13(2)
© The Author(s) 2017
DOI: 10.1177/1550147717694170
journals.sagepub.com/home/ijdsn


Yukun He^{1,2}, Qiang Li^{1,2}, Jian Cao^{1,2}, Yuede Ji^{1,2} and Dong Guo^{1,2}

Abstract

Server-side socialbot detection approaches can identify malicious accounts and spams in online social networks. However, they cannot detect socialbot processes, residing on user hosts, which control these accounts. Therefore, new approaches are needed to detect socialbots on hosts. The fundamental to design host-side detecting approaches is to gain an insight into the behaviors of socialbots on host. In this article, we analyzed a series of representative socialbots in depth and summarized the typical features of socialbot behaviors. We proposed a new approach to defense against socialbots on end host. The contributions of this article are threefold: (1) our analysis approach can be used for reference during analyzing new socialbots in the future; (2) we provide several behavior features of socialbots on hosts, including network flow through which socialbots communicate with botmasters through the online social network, system calls via which socialbots conduct an activity, and process information of socialbots running on hosts. These features can be used by someone to design approaches to identifying socialbots on a host; (3) our proposed detection approach can effectively distinguish between a socialbot and a benign application on end hosts.

Keywords

Socialbot, behavior analysis, social network security, online social network

Date received: 10 July 2016; accepted: 26 January 2017

Academic Editor: Michele Amoretti

Introduction

Online social networks (OSNs), such as Facebook, Twitter, and Weibo, are widely used in various fields such as government, business, education, and finance. They have more than a billion active users. For example, Facebook has 1.19 billion active users per month, as of 30 September 2013; Twitter has more than 40 million users, and 500 million tweets are sent per day.

Attackers have begun to focus on OSNs due to their popularity. A new breed of bot, called socialbots, has been discovered and has become one of the most serious security threats in recent years. According to reports, 2 million stolen passwords from Facebook, Twitter, Yahoo, and automatic data processing (ADP) were found on the Pony botnet server on 4 December 2013.¹ The Federal Bureau of Investigation (FBI) arrested 10 attackers who had caused more than \$850

million losses in a Facebook botnet scam on 12 December 2012.² Newer socialbots use OSNs as command and control (C&C) channels. In these social botnets, each socialbot is a piece of automated software running on user hosts in the background. The socialbot controls an account on an OSN and communicates with the botmaster through the posting and receiving of messages. The socialbot can perform basic activities,

¹College of Computer Science and Technology, Jilin University, Changchun, China

²Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun, China

Corresponding author:

Dong Guo, College of Computer Science and Technology, Jilin University, Changchun 130012, Jilin, China.
Email: guodong@jlu.edu.cn



Creative Commons CC-BY: This article is distributed under the terms of the Creative Commons Attribution 3.0 License

(<http://www.creativecommons.org/licenses/by/3.0/>) which permits any use, reproduction and distribution of the work without

further permission provided the original work is attributed as specified on the SAGE and Open Access pages (<http://www.uk.sagepub.com/aboutus/openaccess.htm>).

such as posting messages, adding friends, and changing personal information on the OSN. It has great impact not only on normal users but also on the OSN itself. Socialbots differ from earlier bots (such as Internet Relay Chat (IRC) bots, Hypertext Transfer Protocol (HTTP) bots, and peer-to-peer (P2P) bots) in that they use OSNs as their C&C channels and communicate with others on the OSNs as if they were human beings.

Several socialbots have been found on OSNs in recent years. For example, Koobface³ is the most successful botnet that remains active today. It has targeted popular OSNs, such as Facebook and MySpace, since its inception sometime in August 2008. Twebot,⁴ like Nazbot and in the same family as it, also acts on Twitter. The PokerAgent botnet, which has been tracked since 2012, was designed to harvest Facebook login credentials.⁵ It has stolen over 16,000 Facebook credentials. In order to understand and analyze socialbot behaviors both on end hosts and on the network side, researchers have also proposed several socialbot prototypes. EJ Kartaltepe et al.⁶ designed Naz+, which is based on Nazbot. It checks Really Simple Syndication (RSS) feeds on CompactSocial for experimental purposes. CompactSocial is a microblogging service that emulates the constraints of Twitter. S Nagaraja et al.⁷ designed a covert social network botnet called Stegobot. It uses images shared by social network users as its C&C channel. JP Verkamp et al.⁸ designed an undiscoverable botnet called Facebot, which is based on treasure hunting social networks. Facebot uses steganographic techniques to hide information sent by a botmaster. Y Boshmaf et al.⁹ designed a socialbot for Facebook called Yazanbot. A Singh et al.¹⁰ designed a socialbot for Twitter that we call Twitterbot. It uses application programming interface (API) provided by Twitter to communicate with a botmaster and gets commands posted by the botmaster through tweets.

To analyze and detect the social botnet, research works have proposed several constructive works. Botnet detection approaches can be divided into two categories by detection location: (1) the first approach detects abnormal host behavior,^{6,11,12} such as registry modification, file system information, and system calls. However, socialbots perform a few activities on end hosts. (2) The second approach detects abnormal behavior on the OSN^{13–16} based on OSN user information, such as messages posted by users and friend requests sent by users. Using this approach, researchers can find malicious OSN accounts or messages. However, socialbots can mimic how normal users perform activities on OSN. Because of the special ability of socialbots to mimic normal users on OSNs, detection on OSNs is difficulty. Therefore, in order to understand socialbots on end hosts, we present an in-depth analysis of socialbots.

In this article, we analyzed a series of representative socialbots in depth and then summarized the typical

features of socialbot behaviors. First, we collected a representative set of socialbots, which are widely used by other researchers. We also designed two socialbots, one on Facebook and the other on Weibo. Second, we created a socialbot simulated runtime environment equipped with monitoring software in virtual machines. Third, we use the correlation analysis method to analyze the profile data of socialbot at runtime. As part of our analysis, we identified several behavior features of socialbots on hosts. The same socialbots have similar traffic statistics when they execute the same commands. Socialbots have fixed traffic statistics when they execute certain commands. Socialbots connect with fewer independent Internet Protocols (IPs) and perform less GET-POST activity than benign social applications. The system call sequences and graphs of socialbots are quite similar to each other, but much less similar to those of benign social applications. The central processing unit (CPU) usage of certain socialbots is cyclical. Memory usage is low and remains stable. Socialbots access files less often than benign social applications. The relationship trees of socialbots may be different from those of benign apps. For example, Koobface has a circuit in its sub-tree. In addition, we proposed a new approach to defense against socialbots on end host. The results of our analysis¹⁷ can be used to spawn new research, such as socialbot detection approaches based on clustering or machine learning.

The main contributions of this article are threefold. First, our analysis approach can be used for reference during analyzing new socialbots in the future. Second, we provide several behavior features of socialbots on hosts, including network flow through which socialbots communicate with botmasters through the OSN, system calls via which socialbots conduct an activity, and process information of socialbots running on hosts. These features can be used by someone to design approaches to identifying socialbots on a host. Third, our proposed detection approach can effectively distinguish between a socialbot and a benign application on end hosts.

The remainder of this article is structured as follows. In section “Socialbot overview,” we give an overview of the socialbot data set and our analysis environment. In section “Network behavior on hosts,” we describe our in-depth analysis of socialbot network behaviors on end hosts. In section “System call on hosts,” we describe our analysis of socialbot system calls on end hosts. In section “Process information on hosts,” we describe the process information we collected about socialbots. In section “Defense against socialbots on an end host,” we propose an approach to defend against socialbot on end host. In section “Related work,” we present related work. In section “Limitations and ideas for future work,” we discuss the limitations of our work and ideas for future works. In section “Conclusion,” we draw a conclusion.

Table 1. Comparison of existing socialbot.

Bot name	OSNs	Host actions	Social actions	C& C	Commands
Koobface	Facebook, MySpace	Steal data	Post message, get message	–	–
fixNazbot	Twitter	Download, exe	Get status	RSS	set, exe, get
Yazanbot	Facebook	–	Post status, get status	Facebook API	status update
Twitterbot	Twitter	Visit URL, download, DDoS	Get status, get twitter id/name	Twitter API	Run, NICs, attack, browse
fbbot	Facebook	getNetInfo, exeCmd, redirect URL	Post message, get message, add friends	Mimic browser	getNetInfo, post status, add friend
wbbot	Weibo	getNetInfo, exeCmd, redirect URL	Post status, get status, add following	Mimic browser	getNetInfo, pubWeibo, addFollowing

OSNs: online social networks; C& C: command and control; URL: uniform resource locator; RSS: Really Simple Syndication; API: application programming interface; NICs: network interface controllers.

Socialbot overview

Characteristics of socialbots

Botnets can be divided into four categories by the C& C mechanisms that they use: (1) the first category of botnets uses IRC techniques. This is a conventional mechanism used by botnets and has been very popular. Bots connect to an IRC server with an IRC protocol and wait for commands from the botmaster. An example of this type of botnet is sdbot.¹⁸ (2) Some botnets use complicated P2P mechanisms. These botnets do not use a central server. The bots in the botnet act as both bot clients and bot servers. If one botmaster in the botnet is removed, other bots can still act as botmaster, so these types of botnets are hard to completely destroy. An example of this type of botnet is Trojan.Peacomm.B.¹⁹ (3) Other botnets use HTTP for C& C. The botmaster does not send commands directly to bots. It instead leaves commands on a web server, and bots get commands from the server. An example of this type of botnet is Zbot.²⁰ The popularity and decentralization of HTTP service makes detection of this C& C channel very difficult. (4) The last type of botnet uses a new C& C mechanism, social media, to exchange information. In these social botnets, the botmaster and bot may be just accounts on the social network. The bot conducts a few malicious activities on end hosts and fetches commands from popular OSNs. This makes defense mechanisms based on IP or domain blacklists fail.

In traditional botnets (IRC, P2P, and HTTP), bot programs use special IPs or domain names to connect to the botmaster and most of their malicious activities (such as information theft) are on hosts. The new socialbots control social network accounts and mimic real users to build a social botnet.²¹ Social botmasters give commands to socialbots through social media, such as encrypted messages in OSN statuses. Then, socialbot programs perform corresponding operations

on the target OSNs. Their influence is mainly on the OSNs.

Socialbot data sets

In order to study socialbot behavior on end hosts, we collected samples from other research institutions. We got a collection of Koobface samples from Georgia Tech Information Security Center.²² According to ClamAV,²³ an open source, general public license (GPL) antivirus engine, these samples are mainly divided into four groups: Worm.Koobface-14, Worm.Koobface-23, Worm.Koobface-118, and Worm.Koobface-130. However, because we could not obtain the Koobface master, we could not control all of the samples. We could only monitor the Koobface samples running on the host. We also contacted the authors of several socialbots and asked for help analyzing their socialbots. After some discussion, the author of Twitterbot gave us the source code of bot, including the source code of the master. Twitterbot appears in a master's thesis.¹⁰ Because we had the master source code, we could monitor how Twitterbot socialbots reacted to different bot commands. We also received advices for re-implementing some other socialbots. Based on the description provided by Y Boshmaf et al.⁹ with their socialbot, we re-implemented a socialbot called Yazanbot, which focuses on Facebook. Based on analysis of Nazbot in the work by Kartaltepe et al.,⁶ we re-implemented a socialbot called fixNazbot, which focuses on Twitter. We also designed two socialbots with novel techniques, one called wbbot, which focuses on Weibo, and the other is called fbbot, which focuses on Facebook. Instead of using the OSN API, the two socialbots use an Internet Explorer (IE) browser on the Windows system to communicate with the OSN platform. We next provide a brief introduction to the socialbots mentioned above. Table 1 compares the main features of each socialbot.

Koobface. This is the most successful socialbot that remains active today.³ The Koobface botnet started targeting Facebook and MySpace in 2008. The binary of Koobface mainly includes seven components. A loader component determines which OSN the affected user is in; it can also download some special components. The social network propagation component communicates with the OSN. It can get recent messages and various URLs (uniform resource locators) from the OSN. It can also post messages and URLs to the OSN. The web server component may act as a proxy server or lead to the Koobface downloader. Ad pushers and rogue antivirus installers download rogue antivirus software and can push ads. CAPTCHA breakers do not solve CAPTCHA image tests using a computer algorithm. Instead, they allow infected users to solve the challenges with CAPTCHA dialog box. Data stealers steal Windows digital-product IDs, email credentials, and instant messaging (IM) application credentials. Web search hijackers intercept search queries to Google, Yahoo, or Live, and redirect them to malicious websites.

Twitterbot. This socialbot uses Twitter as its C&C channel.¹⁰ The botmaster posts tweets with pre-determined keywords. The bots get these tweets from the master account by querying the Twitter search engine. It uses the Open Authorization (OAuth) authentication mechanism to communicate with applications on Twitter. It was developed in Java with the help of the twitter4j API. It has email functionality, which is used to exchange data between bots and the botmaster. It has several generic attacks, such as checkSystem, network interface controllers (NICs), and screenshot. The checkSystem command checks for the user's home directory path and mails the information to the botmaster account. The generic attacks component can also add new functions. The C&C keyword generator generates a new keyword daily. The botmaster puts the keyword before the real command. An example of a tweet with a command is: #metacafe browse http://ccst.jlu.edu.cn. Using a keyword within the tweet raises less suspicion and makes it easier for bots to search for tweets from the botmaster. Twitterbot can update user status on Twitter, fetch the last 20 statuses, fetch follower information, steal the media access control (MAC) address of the system, shut down the system, and so on.

fixNazbot. We re-implement it based on Nazbot.⁶ Nazbot 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. We modified parts of the Nazbot and call it fixNazbot. Instead of using bit.ly URLs, fixNazbot uses a Dropbox file path (Dropbox is a free service that allows you to store various files). The botmaster uploads the payload to the corresponding Dropbox file path. The fixNazbot fetches the payload using the Dropbox API. The other modules of fixNazbot are the same as those of Nazbot.

Yazanbot. This socialbot has two main components: a profile on Facebook and the software.⁹ The socialbot 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 lets the socialbot account connect to other socialbots. rand_connect lets the socialbot accounts connect randomly to non-botmaster-owned accounts. Decluster lets a socialbot account disconnect from all other socialbot accounts. Crawlextneighborhood lets a socialbot find its neighborhood's neighbors and return their profiles as botcargo. Mutualconnect lets the socialbot connect with accounts in botcargo. Harvestdata lets a socialbot return all accessible information. Yazanbot mainly uses two techniques: the OSN API and HTTP request templates. It randomly updates the status with random quotes and blurbs provided by iheartQuotes. It exploits the Facebook Graph API²⁴ to carry out social interaction operations.

wbbot. This socialbot targets the Chinese Weibo platform. Instead of using RSS or the OSN API, it mimics an IE visit Weibo and fetches user personal profile information on Weibo. The wbbot assumes that the infected user has a Weibo account, uses IE to log in to Weibo, and has his or her password remembered automatically. The wbbot first logs in to Weibo and fetches the latest status from the master account. The bot checks whether the status is a command and whether the command has been executed. If it is a new command, the bot decodes the message, parses the command, and carries out the corresponding operations. The botmaster has ten different commands: six of them result in action on the victim's computer and four result in social network activities on Weibo. The commands that act on the host are getNetInfo, which gets local network information; getVersion, which gets the Windows system version; exeCmd, which executes a disk operating system (DOS) command on the victim

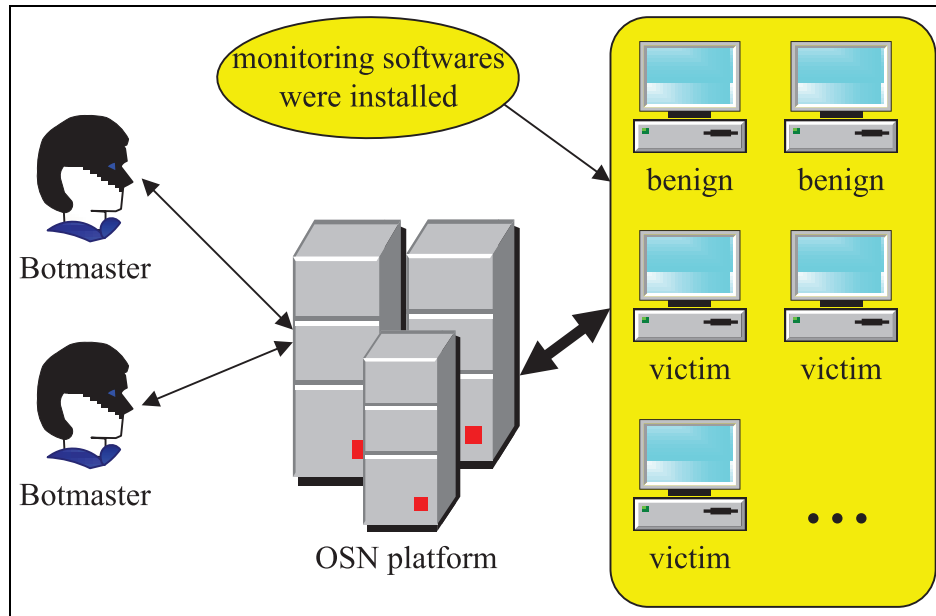


Figure 1. Structure of analysis environment.

host; timeExeCmd, which executes a DOS command on the victim host at a certain time; visit, which forces the IE browser to open a URL; redirect, which rebinds the domain and IP to cause a user to visit a fake website. The OSN commands are pubWeiboText, which causes the infected Weibo account to publish a text message; postComment, which causes the infected account to give a comment to a certain message posted by a special account; addFollowing, which orders an account follow another account.

fbbot. This socialbot targets Facebook. It is almost the same as wbbot in terms of functionality and the technique by which the socialbot communicates with the OSN. It also mimics IE to exchange messages with Facebook. The host commands are identical to those of wbbot. The OSN commands are a little different, due to differences in the OSN platforms. pubWeiboText is replaced by post_status, which public the user's status on Facebook. The addFollowing command orders the infected account to send a request to be added as a friend to a certain account. The control flow of fbbot is the same as wbbot. The fbbot randomly logs in to Facebook, gets the latest status, carries out corresponding actions, and provides feedback information.

Analysis environment

In order to better understand the behavior of socialbots on hosts, we built an analysis environment. We created several accounts on different OSNs, like Facebook, Twitter, and Weibo. We built 10 VMware virtual

machines deployed on systems running Windows XP SP3. We created environments on virtual machines that corresponded to the assumptions made by each socialbot. The virtual machines were equipped with necessary monitoring software, like Process Monitor,²⁵ Wireshark,²⁶ Microsoft Network Monitor,²⁷ and YAPM.²³ For example, fbbot requires that the local IE explorer is used to log in to Facebook with the remember username and password option checked. Then, we created a valid Facebook account to meet the needs of fbbot. We used a master Facebook account to publish several pre-defined commands. Then, fbbot conducted various activities, and we collected the corresponding monitoring data. For different socialbots, we separately ran each socialbot and collected the monitoring data on different virtual machines. To compare socialbot behavior to other behavior, we used four virtual machines as benign computers running normal social applications, such as Facebook Messenger,²⁸ TweetDeck,²⁹ and Weibo Desktop.³⁰ We also used popular browsers on the four machines to visit Facebook, Twitter, and Weibo. The browsers used normal user account.

Figure 1 shows the structure of our analysis environment. The botmasters of different socialbots sent new encrypted messages to the corresponding OSNs. The botmaster was any computer that controlled the profile of the master account. The botmaster could publish commands on the social network through a cellphone. In our environment, the OSNs were Facebook, Twitter, and Weibo. They acted as C & C channels that connected the botmaster and the socialbots. The other virtual machines were used for different socialbots and benign software. Because the environment included both

socialbots and botmasters, bots ran on victim machines and botmasters can send commands where the network was up, we were able to completely control the analysis environment and analyze the socialbots in detail.

Network behavior on hosts

When applications communicate with social network sites, researchers can capture lots of network flows both on user end hosts and on the servers of the social network. Most approaches to detect socialbots analyze network flows captured on the servers. They can only detect malicious accounts on the OSN. Some accounts may be invaded by socialbots, while remaining benign; these accounts are also victims. In our experiments, because we have both socialbots and the corresponding botmasters, we can also monitor network flows of various processes on user end hosts. If malicious network flow is detected, the malicious process and the corresponding infected files can be found. Thus, socialbots can be removed completely. In our analysis of network flow, 3.97GB log files have been recorded, including the benign software and the socialbots. We analyzed three aspects of these network flows: changes in traffic statistics over time, changes in traffic statistic due to commands, and the traffic statistics of benign and malicious software.

Traffic statistics over time

Users use different applications to surf on the OSNs. Even if they carry out the same actions using the same application on different occasions, the application can produce different rates of increase in network flow. In this section, we calculate the total count of network flows over time. We found that the increase in flow rate for some socialbots was the same for the same action at two different running times, while when normal users scan the same OSN with different browsers at different times, the rate of increase is different. And the difference is obvious. For example, Figure 2 shows a 15-min 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 total count of network flow on the y-axis. Line f is the first occasion, and line s is the second occasion. Figure 2(a) shows the rate of increase for Yazanbot on two different occasions. Both times, Yazanbot runs the same botmaster command and produces almost the same rate of increase. Figure 2(b) shows the results of Twitterbot. The rates are different from those of Yazanbot. But again, when Twitterbot is running on two different occasions, it produces almost the same increase line. Figure 2(c) shows the rate of increase in network flow when a person uses Firefox to visit Facebook on two different occasions. Figure 2(d) shows the rate of increase in network flow when a

person uses Chrome to visit Twitter. On different occasions, the two browsers produced more different rates of increase, compared to each other and to the socialbots.

Traffic statistics with various commands

The largest difference between bots and malware is the C&C mechanism. In a social botnet, a socialbot receives commands from a botmaster and then carries out malicious activities on the end host or on the target OSN. In this section, we analyze the network flow produced by different commands. Figure 3(a) shows flow statistics of Yazanbot. In the figure, a, b, c, and d refer to randConnect, crawlExtNeighborhood, mutualConnect, and harvestData, respectively. We ran the four commands with the same parameters several times and calculated the total count of network flows for each command. In our analysis, the average count of network flow is almost the same. It is approximately 500 for each command. Figure 3(b) shows the statistics of Twitterbot. In the figure, bars a–i refer to attack, find, browse, shutdown, upload, execute, run, NICs, and screenshot, respectively. We also used a botmaster to send the nine commands with the same parameters. We found that most commands produced low network flow, such as attack, shutdown, and NIC. These commands led the user host to carry out a host action, so little network flow was produced. However, the browser and screenshot commands produced significant network flow, because they initiate simple network behavior. Because the upload command initiates uploading of a file to the botmaster, it produced the maximum network flow.

Traffic statistics of benign software and socialbots

Both socialbots and benign software communicating with OSN produce a large amount of network flow. Different applications have different flow information, even though they may visit the same OSN. We extracted eight features from the network flows that we captured on user end hosts. The network flow was produced by IE, Chrome, Firefox, Yazanbot, Twitterbot, fixNazbot, Koobface, facebookmessage, TweetDeck, and Weibodesk. The eight features are as follows: FLOWNUM, the total count of flows that the applications generated; IPNUM, the count of independent IP addresses that the applications connected to; IN, the count of flows with the user end host as the destination IP; OUT, the count of flows with the user end host as the source IP; Transmission Control Protocol (TCP), the count of flows with the TCP protocol; HTTP, the count of flows with HTTP protocol; POST, the count of flows which used a POST method; and GET, the count of flows that used a GET method. Table 2 shows

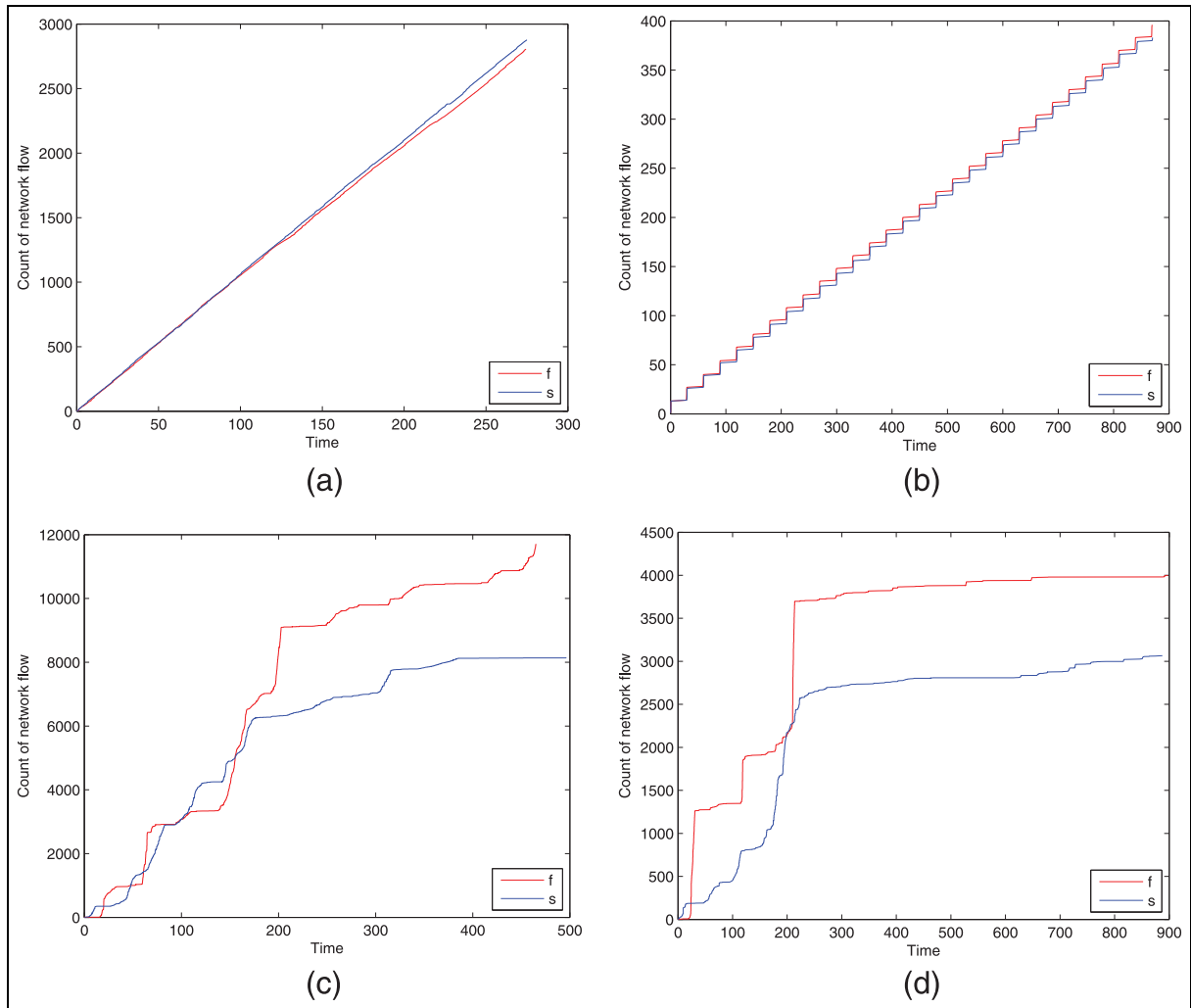


Figure 2. Traffic statistics increase over time. Line f is the first occasion, and line s is the second occasion: (a) Yazanbot, (b) Twitterbot, (c) Firefox on Facebook, and (d) Chrome on Twitter.

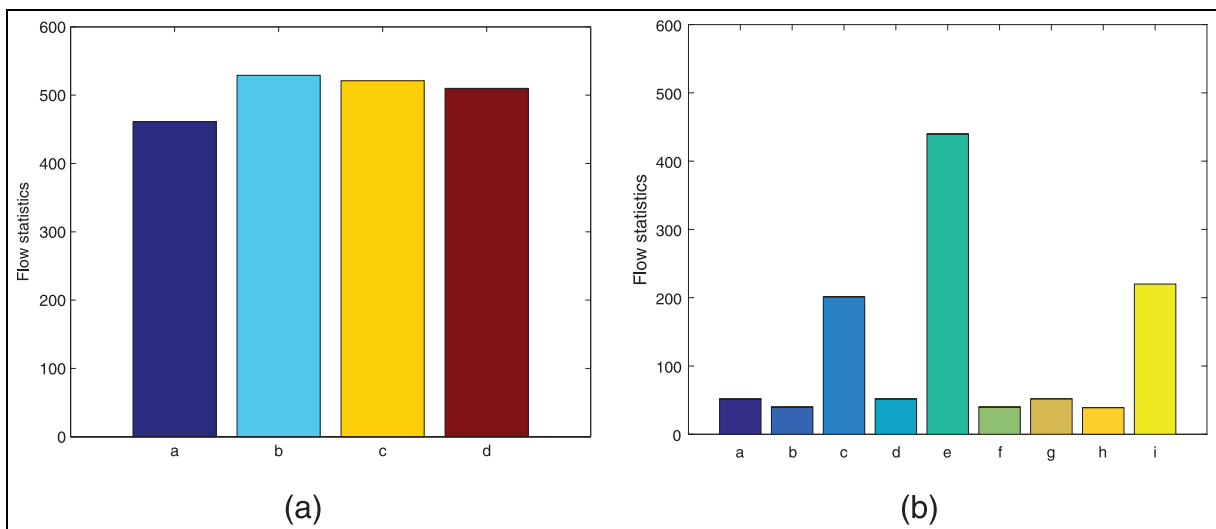


Figure 3. Traffic statistics with different commands: (a) Yazanbot commands and (b) Twitterbot commands.

Table 2. 10-min network log on benign and bot software.

Name	FLOWNUM	IPNUM	IN	OUT	TCP	HTTP	POST	GET
IE	12,085	88	7424	4661	10,800	1235	6	445
Chrome	4023	13	2579	1444	3305	36	1	14
Firefox	11,711	35	7262	4449	9670	187	1	73
Yazanbot	2806	3	1790	1016	1102	92	0	30
Twitterbot	396	2	181	215	306	0	0	0
fixNazbot	45	3	21	24	37	8	0	8
Koobface	50	2	25	25	40	10	0	5
FacebookMessenger	1327	8	777	549	855	0	0	0
TweetDeck	3933	14	2388	1545	2924	0	0	0
Weibo2014	8919	26	5422	3496	8059	760	11	240

IE: Internet Explorer; TCP: Transmission Control Protocol; HTTP: Hypertext Transfer Protocol

logs for 10 min of network flow on these applications. Socialbots clearly had lower IPNUM values: Yazanbot and fixNazbot each connected to only three independent IPs, while Twitterbot and Koobface connected with only two independent IPs. The socialbots had low HTTP, Post, and Get values, but so did the benign FacebookMessenger and TweetDeck software. Based on these features, a good algorithm might be able to differentiate between these socialbots and benign software. For example, machine learning or clusters could be used to deal with the data and discover network flow from socialbots. Neither fbbot nor wbbot is included in this table because they control an IE browser to connect with OSNs and do not directly produce any network flow.

Discussion

We collected a large network flow data set, including data from socialbots, benign social applications, and popular browsers. We acted as the botmasters of all of these socialbots. The socialbots received commands from us and took corresponding actions. The benign applications were used as usual by other students in our college. We invited several students to use a virtual machine as their work environment. The virtual machines were installed with monitoring software. The students ran their social applications on the virtual machine as they would in daily life. For example, when a student uses Weibo2014 to publish a micro-blog or to talk with his or her friend, Weibo2014 produces a large amount of network flow, uses several system calls, and accesses files on the end host. The monitoring software compiled a useful data set. Then, we can analyze Weibo2014 with respect to its network behavior, system calls, and information processing.

In our analysis of traffic data sets, we found that the same socialbots increased traffic at similar rates when the same commands were run at different times. In the traffic of each process on different hosts, if two

Table 3. 10-min logs of system calls.

Socialbot name	Call numbers	Socialbot name	Call numbers
Chrome	66130	fbbot	11109
Firefox	20680	fixNazbot	3174
IE	362513	wbbot	7380
Twitterbot	11561	Yazanbot	5633

IE: Internet Explorer.

processes have similar traffic patterns, they are highly likely to be socialbot processes. When socialbots receive the same command, they may produce fixed amounts of network flows. We can detect which commands the processes are executing by analyzing the amount of network flow. From the traffic statistics, we can easily find that socialbots connect less to remote hosts and produce less network flow in the same period as benign applications. A novel algorithm, such as one that integrates data mining and pattern recognition, might be used to distinguish socialbots from benign applications.

System call on hosts

When programs request services from the Windows system kernel, they need to use system calls. A system call is an essential interface between a process and the operating system (OS). We used Process Monitor on a system running Windows XP to record system calls generated by socialbots. We collected the system calls data set by socialbot command and running time. Then, we analyzed both system call sequences and system call graphs.

Socialbots produce large numbers of system calls when they are not sleeping. Table 3 shows 10-min logs of system calls. IE and fbbot produced more system calls than others, because fbbot executes some command from the botmaster and uses IE to mimic a normal user while acting on Facebook. When dealing with

Table 4. Index of system calls.

Index	Name	Index	Name
1	RegQueryValue	27	FileSystemControl
2	Process Exit	28	Thread Exit
3	QueryAttributeTagFile	29	CreateFileMapping
4	RegCloseKey	30	Load Image
5	QueryNameInformationFile	31	UnlockFileSingle
6	RegSetValue	32	RegDeleteKey
7	RegCreateKey	33	ReadFile
8	SetAllocationInformationFile	34	WriteFile
9	QuerySecurityFile	35	TCP Retransmit
10	UDP Receive	36	QueryFullSizeInformationVolume
11	QueryOpen	37	NotifyChangeDirectory
12	QueryStandardInformationFile	38	QueryAllInformationFile
13	LockFile	39	QueryInformationVolume
14	SetBasicInformationFile	40	QuerySizeInformationVolume
15	QueryStreamInformationFile	41	CloseFile
16	QueryBasicInformationFile	42	UDP Send
17	Thread Create	43	RegEnumValue
18	SetEndOfFileInformationFile	44	RegOpenKey
19	QueryDirectory	45	SetDispositionInformationFile
20	RegEnumKey	46	Process Create
21	TCP Receive	47	CreateFile
22	RegQueryKey	48	TCP Send
23	RegDeleteValue	49	SetRenameInformationFile
24	QueryAttributeInformationVolume	50	Process Start
25	FlushBuffersFile	51	SetSecurityFile
26	TCP Disconnect	52	QueryFileInternalInformationFile

TCP: Transmission Control Protocol; UDP: User Datagram Protocol

system calls, we mapped each system call to a unique number. We gave an index value to each system call, to improve efficiency for a data set with many system calls. Table 4 shows the index of system calls. For example, we mapped the system call RegQueryValue to index number 1.

System call sequence

In our analysis environment, we ran socialbots under the control of botmasters for several times. Each time, the botmaster posted the same commands in the same sequence. We then monitored the system calls for each socialbot and recorded system call times, detail operations, and the pid (process ID) values. With previous system call index, each system call is mapped to a digit. The time when the system call occurs is modified according to start time of the socialbot process. For example, if a socialbot starts running at time t_1 , and the socialbot requests a system call at time t_2 , then we revise occurrence time of the system call at $t = t_2 - t_1$. Figure 4 shows two iterations of four socialbot system call sequences. Figure 4(a) shows information for Twitterbot, Figure 4(b) for fbbot, Figure 4(c) for fixNazbot, and Figure 4(d) for wbbot.

We found that system call sequences for the same socialbot carry stark similarities in different scenarios.

We also see this phenomenon in Figure 4(b)–(d). This feature can be used to classify the socialbot traces and identify which trace a new socialbot belongs to.

We chose different socialbot running in the analysis environment specific to different OSN and C&C techniques. After analyzing the system calls that socialbots produced, we found that when different socialbots ran on the same OSN, their system calls were very similar. When different socialbots running in different OSNs used the same techniques to communicate with the OSN, parts of their system calls were quite similar. However, when different socialbots running in different OSNs used different techniques to connect to the OSN, we did not find similarity between them. Figure 5 compares pairs of socialbots using different techniques in the same OSNs and the same techniques in different OSNs. Figure 5(a) compares Twitterbot and fixNazbot. Twitterbot uses the Twitter API to connect to Twitter, while fixNazbot uses RSS feeds to get status from Twitter. They use different techniques, but both focus on the Twitter OSN. Their system call activity levels were quite similar when they started up. Figure 5(b) shows a similar situation for fbbot and Yazanbot on Facebook. The fbbot mimics a browser to connect to Facebook, while Yazanbot uses the Facebook API. There are two similar parts in the figure. Figure 5(c) compares fbbot on Facebook and wbbot on Weibo:

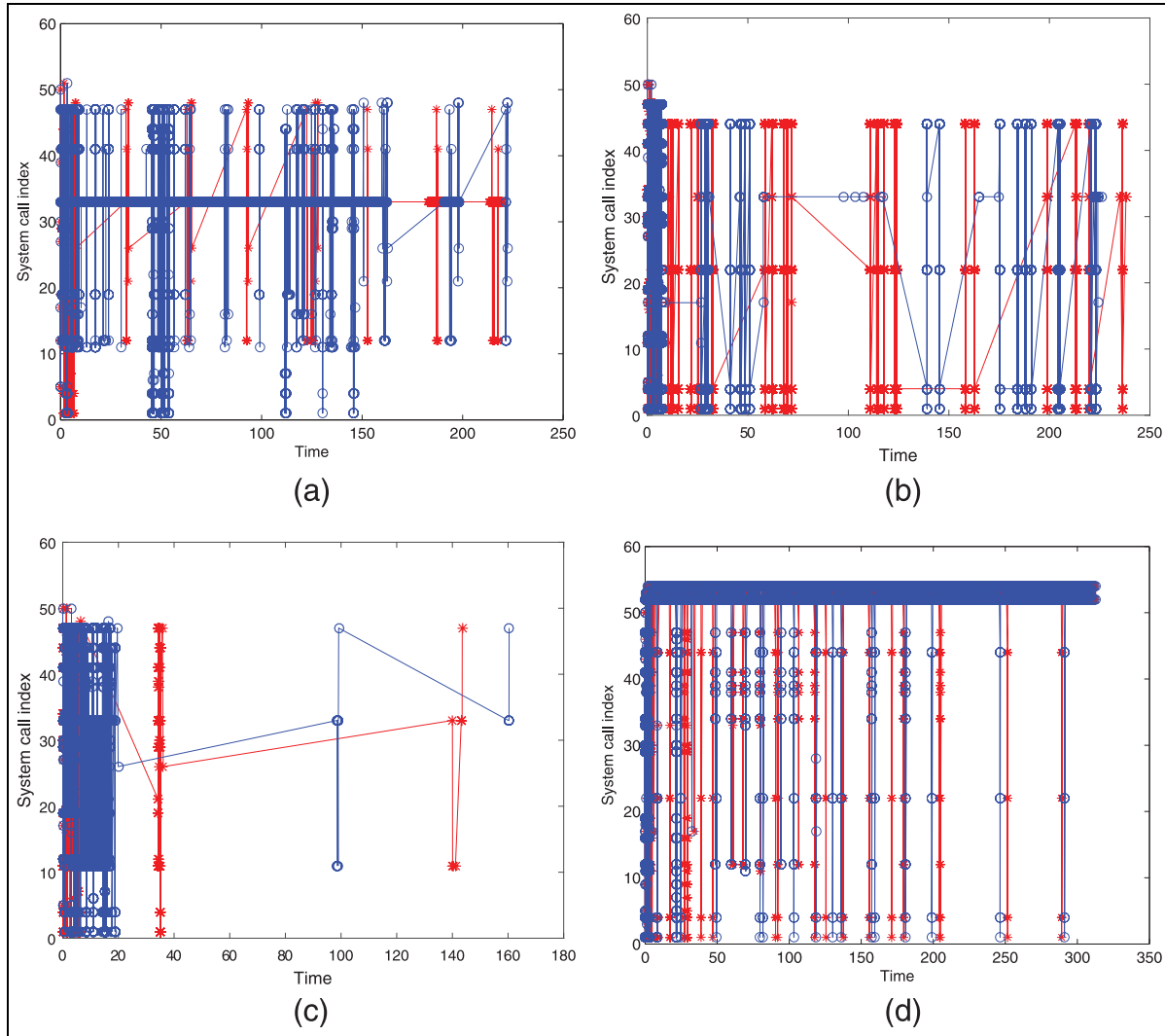


Figure 4. Comparison of system calls from a socialbot for different run times: (a) Twitterbot, (b) fbbot, (c) fixNazbot, and (d) wbbot.

different socialbots accessing different OSNs. Both use the technique of mimicking a browser to connect to the target OSN. In this situation, some parts of the system call sequences are very similar. Figure 5(d) compares fixNazbot and wbbot. They use different techniques to connect to different OSNs. The fixNazbot uses RSS and the target OSN is Twitter, while wbbot mimics a browser to get status from Weibo.

After analyzing the system calls produced by socialbots, we deduced that when different socialbots run on the same OSN, their system calls are very similar. When different socialbots run on different OSNs using the same techniques to communicate with the OSN, parts of their system calls are also quite similar.

Some OSN platforms publish applications that can operate user accounts to carry out various actions. These applications can do almost everything that browsers can do. Some socialbots use the APIs that OSNs

provide to connect to the OSNs. We selected three benign applications and four socialbots to conduct an experiment on this behavior. Figure 6(a) compares Twitterbot and TweetDeck running on Twitter. Figure 6(b) compares Yazanbot and FacebookMessenger running on Facebook. Figure 6(c) compares wbbot and Weibo2014 running on Weibo. Figure 6(d) compares fbbot and FacebookMessenger running on Facebook. We deduced that socialbots and benign software can be distinguished by their system call sequences as represented in Figure 6(c). As a result, this feature can be used to distinguish between a socialbot and a benign application.

System call graphs

Based on system call sequences, we can draw system call graphs for benign software and socialbots. In a

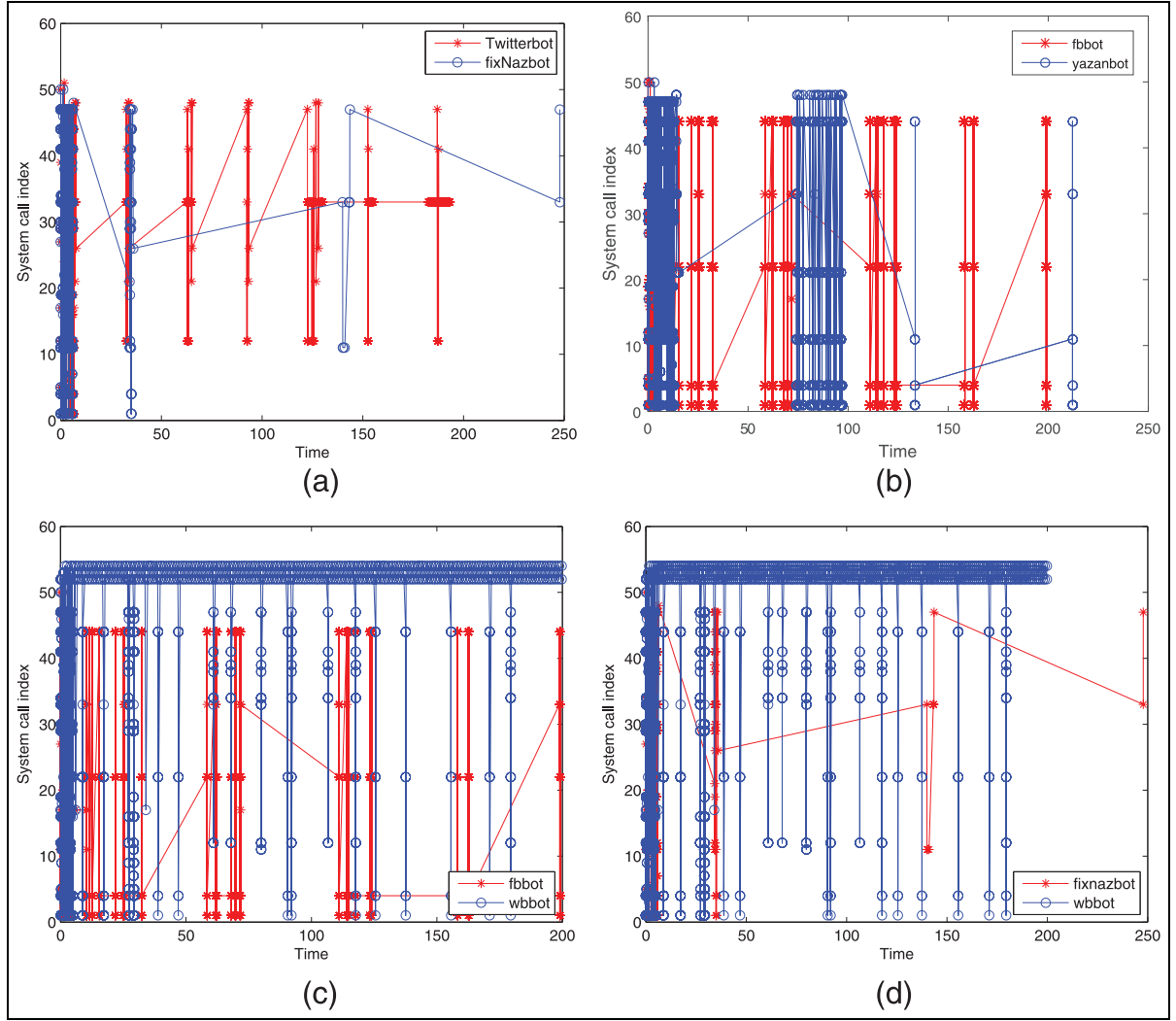


Figure 5. Comparison of different socialbots on different OSNs: (a) Twitterbot and fixNazbot on Twitter, (b) fbbot and Yazanbot on Facebook, (c) fbbot on Facebook and wbbot on Weibo, and (d) fixNazbot on Facebook and wbbot on Weibo.

system call graph, each system call is one point. System calls may be called several times. The time of system call happening determines the directions of relationships in the graph. For example, the Chrome browser produces two system calls continuously: ReadFile and WriteFile. ReadFile always precedes WriteFile. In this situation, we can create a sub-graph. ReadFile and WriteFile are two points in the sub-graph and there is one edge between them, the direction is from ReadFile to WriteFile. In this manner, we can create system call graphs for applications. Then, we identified the strongly connected components of the graphs using Tarjan algorithms. We used the strong connected components to represent each graph. If the system call graph is G , then corresponding strong connected component is $S(G)$.

Figure 7 shows four system call graphs. System calls at the center of a graph are essential to the application.

These are two different socialbots running on two different OSNs. After identifying the strongly connected components of each socialbot system call graph, we get the differences between the two graphs. We just calculate the differences between pairs of strongly connected components. G_a represents the system call graph of socialbot a , and G_b the system call graph of socialbot b . Their strongly connected components are $S(G_a)$ and $S(G_b)$. We used the similarity between their strongly connected components to reflect the similarity between system call graphs. The similarity between sets can be calculated using the Jaccard Index,³¹ which is a famous effective means of measurement. Finally, we calculated the similarity between a and b , $D(a, b)$ as $D(a, b) = |S(G_a) \cap S(G_b)| / |S(G_a) \cup S(G_b)|$.

We also identified some basic system calls that most applications would use. This will mitigate the difference between benign applications and socialbots and expand

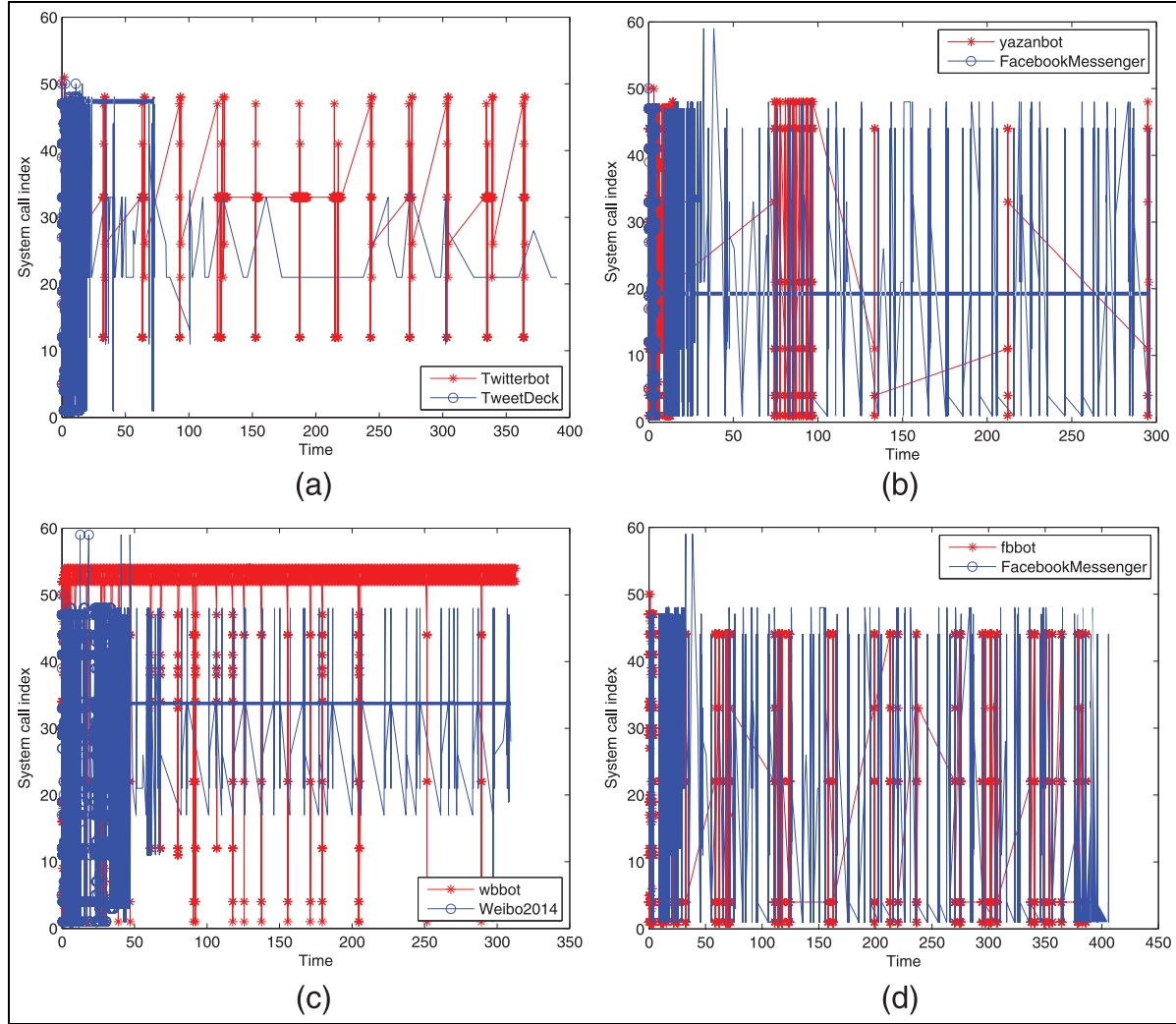


Figure 6. Comparison of benign and socialbot system calls: (a) Twitterbot and TwitterDeck on Twitter, (b) Yazanbot and FacebookMessenger on Facebook, (c) wbbot and Weibo2014 on Weibo, and (d) fbbot and FacebookMessenger on Facebook.

the similarity. To reduce the impact of the basic system call, we filter the basic system calls and recalculate the similarity with the equation: $n = s - m/1 - m$, where m represents the impact factor of a basic system call, s represents the old similarity, and n represents the final similarity. The smaller the m value, the more obvious the difference of the final similarity. As an empirical value in our experiment, we take 40% for m , which is close to the minimal old similarity.

In our analysis, the similarity of system call graphs for the same socialbot at two different run times is higher than 92%. In Figure 7, a, b, c, and d are four system call graphs of socialbot and benign applications. Figure 7(a) shows the graph of Twitterbot. Figure 7(b) shows the graph of fbbot. Overall, the system call graphs seem to differ greatly. To analyze the similarity between them, we calculate their strong components. The strongly connected component of Twitterbot are

$\langle 24, 41, 30, 33, 5, 44, 39, 27, 26, 47, 11, 19, 17, 12, 21, 4, 29, 7, 1, 48, 16, 18, 43, 51, 6, 22 \rangle$ and for fbbot, they are $\langle 30, 33, 45, 41, 4, 17, 12, 39, 47, 22, 38, 11, 19, 18, 44, 5, 29, 1, 20, 27, 6, 34, 16, 46, 43, 50, 3 \rangle$. Although these are two different socialbots using different techniques to connect to different social websites, their strong components show 35% similarity. Figure 7(c) shows the graph of Yazanbot and Figure 7(d) shows the graph of the benign application FacebookMessenger. They both conduct activities on Facebook. The similarity of their strongly connected components is 20%.

We identified the strongly connected components of all of the socialbots in our analysis. We created the set $\langle 33, 1, 39, 12, 11, 44, 47, 16, 17, 22, 4, 27, 29 \rangle$, which we generated as the intersection of all of the socialbots' strongly connected components. The digits in the set can be obtained from our system call index in Table 4. These are basic system calls that socialbots used

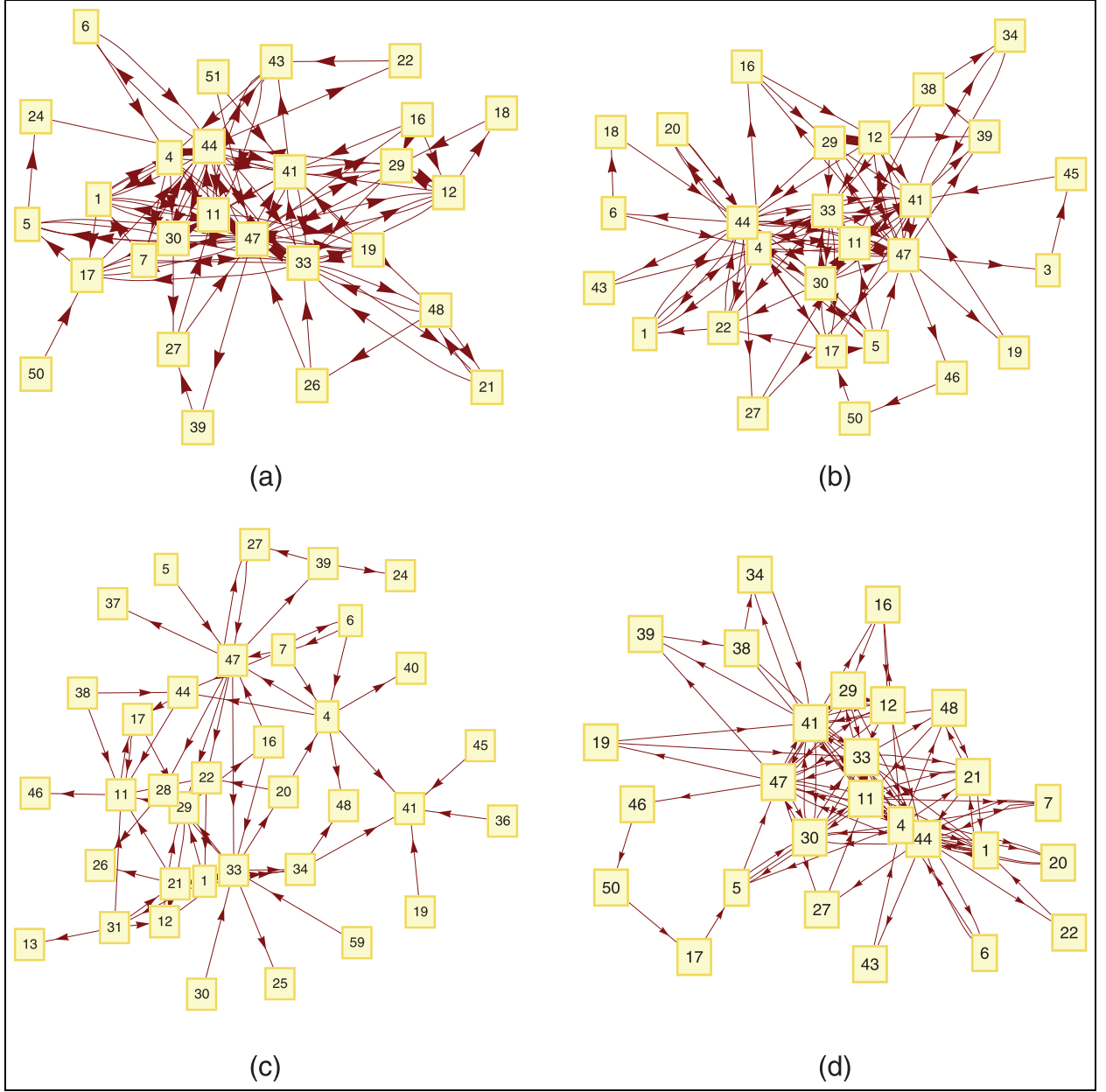


Figure 7. System call graphs of socialbots and benign software: (a) Twitterbot, (b) fbbot, (c) Yazanbot, and (d) FacebookMessenger.

frequently. In our experiment, we also found the similarity between different socialbots ranged from 5% to 35% and the similarity between socialbots and benign social applications ranged from 20% to 67%.

Ultimately, we draw the conclusion: the same socialbots have an obvious high similarity in a system call graph. The different kinds of socialbots have a quite low similarity. The similarity for socialbot and benign applications is uneven. Although we cannot distinguish socialbots and benign applications by the system call graph feature alone, this feature can be used to detect socialbots that belong to the same class. For example, if fbbot (one socialbot sample) has been detected on an

end host, the system call graph of the fbbot can be collected. We can also collect the system call graphs of applications on the other end hosts. If the similarity between the fbbot and one of the applications is higher than 90%, we can conclude that the application belongs to that class of fbbot.

In the system call graphs, fingerprinter of socialbot can also be extracted. Degree of nodes in the system call graphs can distinguish socialbot from each other. For example, system call 50 in Figure 7(a) has no in-degree and has only one out-degree, a few system calls of Yazanbot in Figure 7(c) have only one in-degree and have no out-degree.

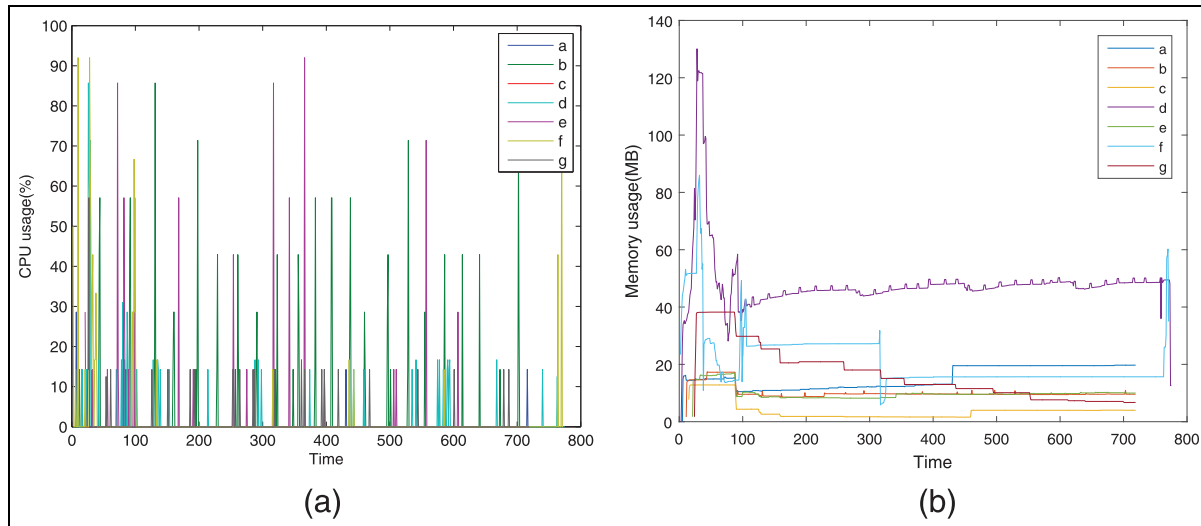


Figure 8. CPU and memory usage of socialbots and benign apps: (a) CPU usage and (b) memory usage.

Discussion

We collected system call data sets from socialbots, benign applications, and popular browsers. We analyzed the system call sequences in three scenarios: (1) when the same socialbot received the same commands at two different times, (2) when different socialbots ran on different OSN websites, and (3) when benign applications and socialbots ran on the same OSN website. We also drew the system call graphs of all applications, identified their strongly connected component, and calculated the similarity between pairs of graphs. In this way, we discovered the set of system calls that socialbots use frequently.

The system call sequence can be used to detect socialbots on different social websites. Different socialbots that use the same technique to connect to their target OSNs may show similar sequences. This feature can then be used to identify as-yet discovered socialbots in the same family as bots that have already been detected. The same socialbots running on different user hosts may have highly similar system call sequences when they receive the same commands from the botmaster. Using appropriate cluster methods, we can discover similar sequences and identify socialbot processes. All system call graphs of one socialbot have very similar strongly connected components. We might use the similarity of the strongly connected components to detect known socialbots and to make inferences about unknown socialbots. Features may be extracted from system call graphs and then used for machine learning to defend against socialbots.

Process information on hosts

In this section, we collect runtime process information. Both socialbots and benign social applications run on

Windows systems running Windows XP SP3. They are all virtual machines with dual-core 2.4-GHz processors and 512 MB of RAM (random access memory). We analyzed three aspects of these data sets: CPU and memory overhead on the system, file access information, and process relationships.

Process overhead

Normal applications running on user hosts consume CPU and memory resources. Traditional bot applications and malware may be injected into other benign processes where they can hide. Thus, we cannot find them in the process list and we cannot accurately obtain their impact on system overhead. In our analysis data sets, the socialbots did not use injection techniques. All socialbot processes were visible in the process list and all consumed more or less CPU and memory resources. The applications ran on different virtual machines for about 10 min. The socialbots received commands and acted accordingly. The benign social applications were controlled by normal users and performed normal social networking behaviors. In Figure 8, from a to g, the applications are Twitterbot, fbbot, fixNazbot, TweetDeck, wbbot, Weibo2014, and Yazanbot. The times in the two figures are modified and relative to the start times of the applications.

Figure 8(a) shows the CPU usage of socialbots and benign applications. Lines b and e are socialbots that mimic browsers to connect to an OSN website. When they act, CPU usage remains between 58% and 92%. Socialbots that use APIs to connect to social network websites (such as lines a and g) consume fewer CPU resources—using a maximum of about 14% of CPU. TweetDeck, a benign application, used CPU as shown

Table 5. 10-min file access statistics.

Bot name	CreateFile	ReadFile	WriteFile	CloseFile
Twitterbot	382	498	61	346
fbbot	305	605	83	197
fixNazbot	209	583	57	154
wbbot	880	293	94	348
Yazanbot	326	1276	130	252
FacebookMessenger	1940	5213	3861	793
TweetDeck	1445	2645	1416	1352
Weibo2014	2833	5733	6742	2603

by line d. It used a maximum of about 17% of CPU, much like Twitterbot and Yazanbot. There were no clear boundaries.

Most socialbot CPU usage is subject to cyclical changes, after special data is discarded. Special CPU usage data may be generated when the socialbot executes a command. The socialbot must query commands from its botmaster at non-fixed times and CPU usage may be very similar at those times. Lines b, e, and g clearly show this phenomenon.

Figure 8(b) shows memory usage by both socialbots and benign applications. When the applications start up, both socialbots and benign applications consume a lot of memory. However, benign applications consume more memory. We can see this from line d and line f, which are benign social applications. The memory usage of d (TweetDeck) reaches 132 MB and the usage of f (Weibo2014) reaches 87 MB. When the application runs for a period of time, memory usage starts to fall and then becomes stable. In our analysis, benign applications still use more memory than socialbots and usage is subject to irregular changes. Socialbot's memory changes a little and they all keep less than 20 MB.

File access statistics

To compare file access by socialbots and benign applications, we collected information about four types of file access commands: CreateFile, ReadFile, WriteFile, and CloseFile. Table 5 summarizes file access information for five socialbots and three benign social applications. Statistics in the table is about 10 min of application activity. Socialbots received and executed commands from their botmasters. Benign applications were controlled by normal users, running them as usual. Benign social applications initiated more file access actions than socialbots in the same period of time. The difference is quite obvious in WriteFile actions. Why did this phenomenon occur? The reason may be that socialbots remain in a sleep state at most time. Socialbots connect to social websites and just conduct simple actions, such as reading statuses from the botmaster or feeding back stolen information to the botmaster. In contrast, benign applications controlled

by normal users fetch more information from social websites, such as the statuses of the users' friends. The benign applications created more temporary files and accessed files more frequently.

Process relationships

Some bots use multi-process approaches to avoid detection. They have multiple modules. Different processes implement different functions. For example, Nazbot,⁶ also known as Twebot, can download and run executable files from the Internet according to botmaster commands. When the executable files perform malicious actions, antivirus systems may detect and remove them. However, because Nazbot itself does not perform the malicious actions, it is not discovered unless other detection techniques are used.

In our analysis, socialbots do not inject themselves into benign processes. They all appear on the process list. In order to see the relationships between all processes in the operation system, we drew relationship trees for different processes. In a process relationship tree, we defined the operation system as the virtual root node, but the OS process does not actually exist. Nodes in the relationship tree consisted of different processes. If process *a* creates process *b*, then *a* is the parent process of *b* and *b* is the child process of *a*. In a relationship tree, there is a directional connection from parent to child processes. If a process creates a new process of the same name, then there will be only one connection with the same start and end points in the relationship tree. When a user starts a desktop application, the default parent process is usually EXPLORER.

Figure 9(a) shows the process relationship sub-tree of wbbot and fbbot. The two socialbots are manually started. They mimic IE browsers to connect to social websites and both are child processes of Explorer. However, IEXPLORER, running in the background, is the child process of svchost. Normally, the IEXPLORER process is the child process of Explorer. In the sub-process tree, we cannot see any direct relationship between socialbots and EXPLORER, but fbbot and wbbot really use IEXPLORER to connect to the social website.

Figure 9(b) shows the sub tree of Koobface. Koobface is also manually started. In the sub-tree, we can see that Koobface118 is a child process of Explorer. It has two child processes: one create a system process (cmd) to recreate itself and another child process (bolivr30) is used to modify the system registry. There is a circuit between Koobface and its child processes. This phenomenon is rare benign process trees.

For Figure 9(c), we collected process information from all applications and combined the information in one relationship tree. The normal Chrome, IE, and Firefox browsers are the child processes of Explorer. So are the benign social applications FacebookMessenger,

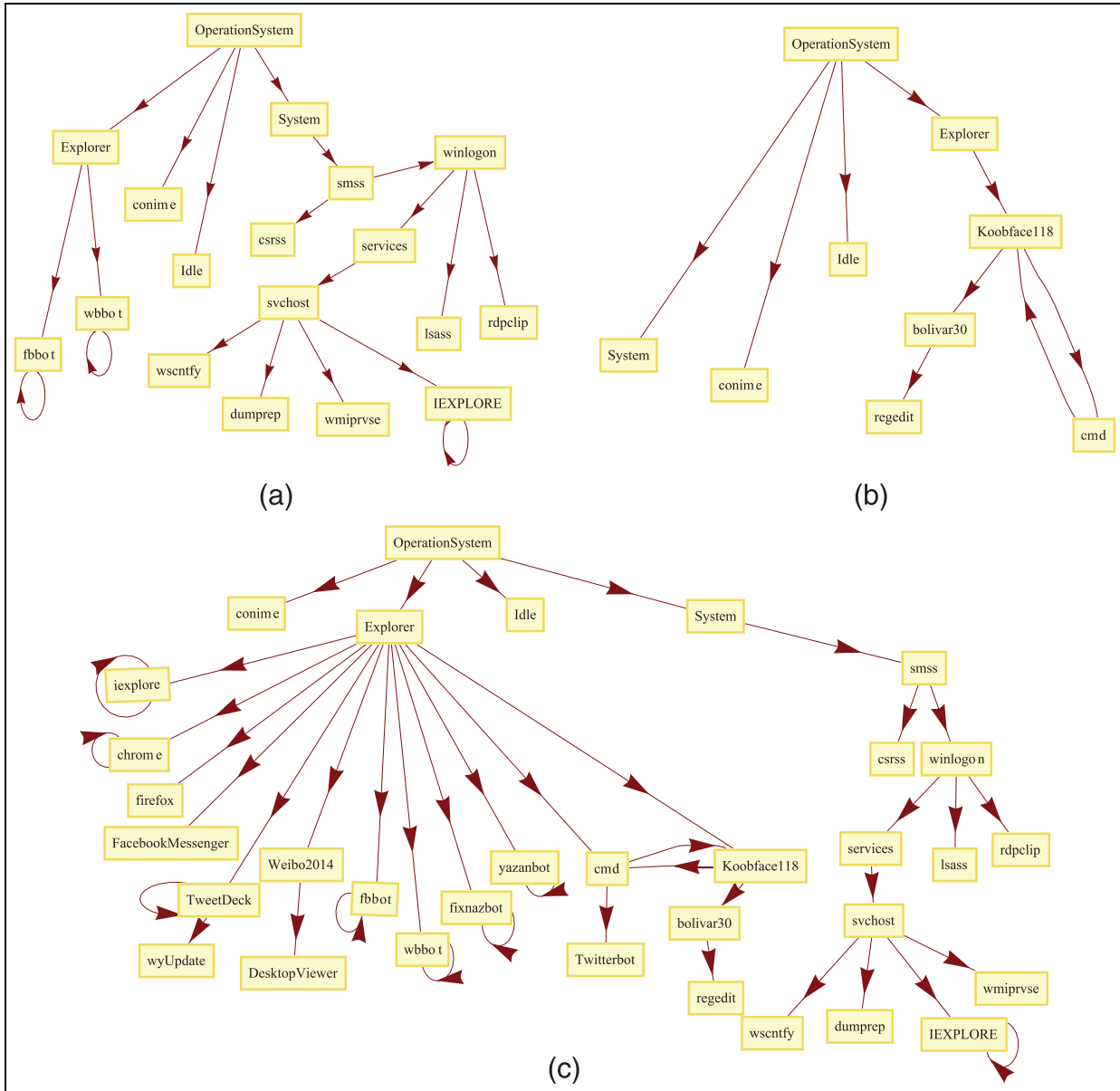


Figure 9. Process relationship trees: (a) wbbot and fbbot, (b) Koobface, and (c) socialbots and benign applications.

TweetDeck, and Weibo2014. Weibo2014 has a child process DesktopViewer. Four socialbots are also child processes of Explorer. They behave like benign applications, judging from the process tree. Twitterbot and Koobface have the same parent process, cmd. From the whole process relationship tree, we can identify how a process is created, the parent process, and the structure of its sub-tree. These features may be useful for socialbot detection, such as the unique features of the Koobface sub-tree.

Discussion

We collected CPU and memory usage data sets for all our samples. In our analysis, socialbots often used little

memory. Their CPU usage changed periodically. Their memory usage remained smooth. We monitored and recorded file access information for all samples. In the same period of time, socialbots usually produce fewer file operations than benign applications. This may be because socialbots are in a sleep state most of time. We also calculated the parent-child relationships of all processes running in the system.

Cycles and other patterns in CPU usage can be used to detect known socialbots. From memory usage analysis, we know that socialbots often use less memory and use a stable amount of memory. In socialbot detection on hosts, we should focus more on processes with low and stable memory usage and cyclical CPU usage. File access statistics can help us distinguish socialbots from

social applications, if we use appropriate methods, such as machine learning and clustering. Some socialbots may be programmed to use multiple processes and have special process trees with unique patterns. These features may help us discover that many malicious processes belong to the same socialbot.

Defense against socialbots on an end host

Approaches for defending OSNs against large-scale infiltration can be divided into prevention on the OSN platform and detection at the end host. To prevent a social botnet from operating on an OSN, the OSN provider must propose various approaches for weakening the socialbot such as the reverse Turing test and restrictions on the OSN API, keyword filtering, and personal information verification. Although these approaches can reduce the influence of socialbots on target OSNs, the socialbot, being the root of the security threat, still exists on the user end host. In this section, we have summarized several features to distinguish socialbots from benign applications on the end host. Our approach can identify suspicious socialbot processes. We discuss data set and features in subsection “Data set and features,” as well as analyze the detection performance in subsection “Detection performance.”

Data set and features

We have examined six classes of representative socialbots. The collection of Koobface samples were provided by the Georgia Tech Information Security Center. These samples are divided into four groups: Worm.Koobface-14, Worm.Koobface-23, Worm.Koobface-118, and Worm.Koobface-130. Unfortunately, we were unable to obtain the master of Koobface. The other socialbots were provided by their authors. The author of Twitterbot provided us with the bot source code, including the master source code. We received advice for re-implementing other socialbots (Yazanbot and fixNazbot) from other cooperative researchers. We also designed two socialbots using novel techniques: wbbot, which targets Weibo, and fbbot, which targets Facebook.

While it is not easy to collect numerous socialbot samples, we tried to collect as many socialbot instances as possible. In contrast, hackers use complicated encryption mechanisms to pack the source code and only spread the bot binaries to infect hosts, and researchers who have source code or prototypes do not share them publicly, citing rules of academic ethics.

To collect different traces for both socialbots and benign applications, we set up VMware virtual machines on a Windows XP SP3 OS. We used Process Monitor to record Registry and File operations, Microsoft Network Monitor to collect network traffic, and a hook program that we solely wrote to record

Table 6. Details of the features.

Index	Feature	Description
F1	HO	Human operation
F2	MBoS	Modifying bootstrap of system
F3	VSI	Visiting sensitive information
F4	UIC	Using Internet cookies
F5	CoIC	Count of IPs communicated
F6	CoDV	Count of domains visited
F7	CoOIV	Count of OSN IPs visited
F8	CoOAV	Count of OSN accounts visited
F9	CoPO	Count of ports opened
F10	CoFI	Count of failed IPs
F11	CoFD	Count of failed domains
F12	CoFOD	Count of failed OSN domains
F13	CoFVOA	Count of failed visited OSN accounts

IPs: Internet Protocols; OSN: online social network.

mouse and keyboard events. When the bots were running, we requested that users operate the virtual machine normally, and both benign and malicious behaviors were then captured. The duration of the original data set is approximately 110 h, and its capacity is approximately 39 GB.

We filter the data set and calculate 13 features. Detailed descriptions of the features are presented in Table 6. The features are divided into three classes: pre-defined host features, OSN connection features, and OSN failed features. Features F1 to F3 are pre-defined host features. F1 is human operation which can be judged by whether a process has a keyboard or a mouse action. F2 represents whether a process has modified bootstrap system. F3 represents whether a process has visited sensitive information, which can be judged by reading and writing registers/files. Features F4–F9 are OSN connection features. F4 means whether a process uses the Internet cookies. F5 represents the count of unique IPs which a process connected with. And the communication protocol includes HTTP, TCP, and User Datagram Protocol (UDP). F6 represents the count of unique domains which a process visited. F7 represents the count of unique IPs of any OSN visited. F8 represents the count of OSN user accounts which a process has visited. This feature can be obtained by URL connection. F9 represents the count of ports which a process has opened. Feature F10 represents the count of IPs which a process failed to connect with. Feature F11 represents the count of failed domains. Feature F12 represents the count of OSN domains which a process failed to visit. Feature F13 represents the count of OSN user accounts which a process connects to.

Detection performance

In this section, we discuss the performance of our detection approach. First, we evaluate the performance

Table 7. Detection results.

Classifier	TPR	FPR	Precision	Recall	F-measure	ROC area
J48	0.961	0.092	0.96	0.961	0.96	0.963
ID3	0.979	0.058	0.979	0.979	0.979	0.967
RandomTree	0.969	0.071	0.969	0.969	0.969	0.98
BayesNet	0.881	0.231	0.882	0.881	0.882	0.927
NaiveBayes	0.881	0.232	0.881	0.881	0.881	0.926
WAODE	0.963	0.089	0.962	0.963	0.962	0.984

TPR: true-positive rate; FPR: false-positive rate; ROC: receiver operating characteristic; WAODE: weightily averaged one-dependence estimator.

by six machine learning classifiers. Then, we analyze the sensitivity and robustness of proposed approach. Finally, we present the detection performance comparison with other existing approaches.

Performance evaluation. We evaluate our approach on part of data set as described in section “Data set and features.” We randomly choose 27,216 malicious instances and 100,000 benign instances. We use Weka³² as our machine learning tool. We conducted our evaluation using six different machine learning classifiers: J48, ID3, RandomTree, BayesNet, NaiveBayes, and weightily averaged one-dependence estimators (WAODE). For every machine learning classifier, we used a 10-fold cross validation method to analyze the performance metrics, including true-positive rate (TPR), false-positive rate (FPR), precision, recall, F-measure, and receiver operating characteristic (ROC) area.

Table 7 shows the detection performance of the six classifiers. For the TPR, they are all higher than 88%. Especially, for the TPR, they are all above 96% under J48, ID3, RandomTree, and WAODE. In our experiment, ID3 achieved the best detection result. It has the highest TPR, precision, recall, and F-measure, the lowest FPR and the second highest ROC area. From the above analysis, we see that our approach can achieve a high detection rate with different classifiers.

Performance analysis. To analyze the robustness and sensitivity of our proposed approach, we divided the data set into 40 groups. Because it is easier to obtain traces from benign application than malicious socialbot, we divided benign instances into more groups than malicious instance. As shown in Figure 10, malicious instances are divided into four groups which are labeled as a, b, c, and d. Benign instances are divided into 10 groups which are labeled as A–J. The group a has 5008 malicious instances. The group b has 11,088 malicious instances. The group c has 23,188 instances. The group d has 27,216 malicious instances. The group A has 10,000 benign instances; group B has 20,000 benign instances, and so on.

Figure 10 presents the detection performance changing with data set increased. Figure 10(a) shows the TPR. In group A, the TPR obviously improves when the account of malicious instances increases. When group A upgrade to group E, the TPR goes to stable. The TPR remains stable in groups E–J. Figure 10(b) shows the FPR. Groups c and d have a low FPR and always lower than 0.05. From this figure, we can find that the FPR remains stable when the count of malicious instances is large enough. Figure 10(c) shows the ROC area. We can see that the ROC area remains high and stable when the count of malicious instance is large enough. Figure 10(d) shows precision, Figure 10(e) shows recall, and Figure 10(f) shows F-measure. In these three figures, the detection performance goes to stable when the benign instance is large enough. From the analysis, we can conclude that our new approach can clearly distinguish between a socialbot and a benign application with high accuracy and robustness.

We also have added a comparison of the performance of our approach on Windows 7 and Windows XP. Table 8 presents the detection results on Windows 7 and Windows XP. The TPR can reach 97.9% on Windows XP and 98.8% on Windows 7. The results reveal that our proposed approach is also appropriate for the Windows 7 OS.

Performance comparison. To clearly show the detection results, we compared our results with those of an existing socialbot detection approach. Chu et al.’s approach¹¹ yielded a higher TPR value of 99.4%, whereas the TPR value of our approach is 97.9%. However, in our approach, the advantage of detecting a socialbot on an end host is that we can capture the process information and identify any suspicious socialbot process. When socialbot processes on an end host are identified and eliminated, the problem of malicious socialbot activity on OSNs can be resolved.

To promote the study on socialbot detection in the community, we have shared the complete data set used in our work. The data set includes instances of socialbots collected from various research institutes, analysis environments built using a virtual machine, the original traces captured in our study, the features extracted in

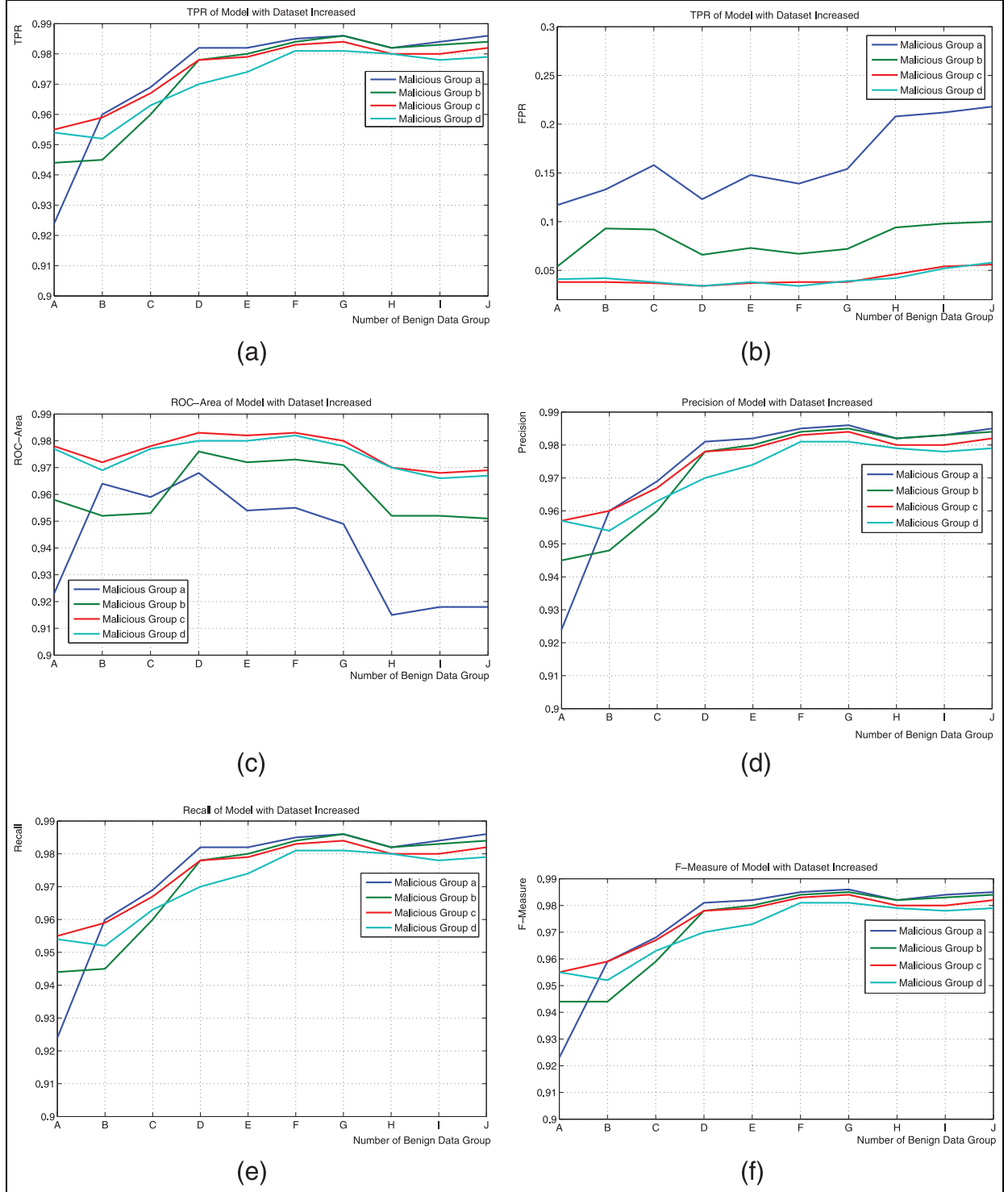


Figure 10. Detection performance with data set increased: (a) TPR, (b) FPR, (c) ROC area, (d) precision, (e) recall, and (f) F-measure.

machine learning, and detailed information on the detection results obtained through our approach. The complete data set can be found on BaiduCloudDisk (<http://pan.baidu.com/s/1qW9QyXe> code: ywy6).

Sniffer system. We have proposed a simple application scenario called socialbot sniffer system according to previous work. This detection system uses the server-client defense mechanism.

Table 8. Detection results on Windows 7 and Windows XP.

OS	TPR	FPR	Precision	Recall	F-measure	ROC area
Windows 7	0.988	0.04	0.988	0.988	0.988	0.987
Windows XP	0.979	0.058	0.979	0.979	0.979	0.967

OS: operating system; TPR: true-positive rate; FPR: false-positive rate; ROC: receiver operating characteristic.

Socialbot sniffer system is mainly divided into S1 and S2. S1 is for the host detection, called sniffer-client which deploys sniffer tools. Sniffer tools collect activities on hosts for every 10 min which will immediately generate detection results and suspicious degrees according to the previous training data set from the sniffer-server. S2 is the sniffer-server. It can collect activities from all sniffer-clients. These activities include the total count of flows that the applications produced; the count of independent IP addresses that each application connected to; the count of flows with the user end host as the destination IP; the count of flows with the user end host as the source IP; the count of flows with the TCP protocol; the count of flows with HTTP protocol; the count of flows which used a POST method; and the count of flows that used a GET method. The sniffer-server can also re-train the data set to improve the accuracy of detection. When sniffer-server updates the detection data set, it will push the data set to all sniffer-clients. The sniffer-clients also support offline detection, when there is no network. We have successfully deployed the simple socialbot detection system on seven hosts (one server, six clients) in the lab environment.

Related work

Malware and botnet phenomena have recently been the subject of study for both researchers and security companies. The community has proposed various methods of malware analysis,^{33–36} botnet detection,^{37–39} and socialbot detection.^{6,11,12,14,15,40}

Malware analysis

K Rieck et al.³³ proposed a framework for automatic analysis of malware behavior using machine learning techniques. The framework can discover novel malware classes and distinguish known classes. C Rossow et al.³⁴ analyzed the network traffic of malicious software. They conducted in-depth analysis of domain name system (DNS) and HTTP traffic used by malware. R Perdisci et al.³⁵ simplified the multi-step clustering process and leveraged incremental clustering algorithms that can run efficiently on large data sets. Their new clustering approach can reduce processing time from several hours to a few minutes and also scales well. M

Graziano et al.³⁶ researched the use of protocol learning techniques for emulation of the external network environments required by malware. Their system can be used for the repeatable analysis of unknown samples that use custom protocols to connect to external hosts.

Botnet detection

One approach for botnet detection is to detect the C&C channel on the user host. For example, in traditional detection of bot C&C channels, JACKSTRAWS³⁷ proposed a method to get the system calls and NetFlow activity graph. Then, they used a machine learning algorithm to detect the C&C channel. BotTee³⁸ gathered system API calls between receiver and sender. They created a large template library and then used a longest common subsequence (LCS) algorithm to detect bot activity. We proposed a BotInfer framework in our previous work. The BotInfer framework correlates host detection results and network analysis results to detect victim hosts in the local network. The framework assumes that part of the local network has been equipped with antivirus software and all network flow can be captured. It then uses an inference algorithm to detect user hosts without installed antivirus software. S Nagaraja et al.³⁹ proposed BotGrep, an algorithm that can isolate efficient P2P communication structures based on communication graphs. Their approach relies on the fast-mixing qualities of structured P2P botnets. The BotGrep algorithm starts with a communication graph $G = (V, E)$. V is the set of hosts in the traffic and E is undirected edges. G_p is a P2P graph and the remaining sub-graph G_n is a non-P2P graph. The goal of the algorithm is to effectively partition the input graph G into two parts G_p and G_n . Shanthi and Seenivasan⁴¹ propose a detection methodology to classify bot host from the normal host by analyzing traffic flow characteristics based on time intervals instead of payload inspection. Their approach is possible to detect botnet activity even when encrypted C&C channels are used.

Socialbot detection

EJ Kartaltepe et al.⁶ studied social botnets that use OSN websites as their C&C channel. They described the strengths and the weaknesses of socialbots. They

proposed two countermeasures: one server-side and one client-side, to deal with current and future generations of social botnets. Pantic and Husain⁴² present a steganographic system that demonstrates the feasibility of the social networking website Twitter as a botnet C&C center. Z Chu et al.¹¹ 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. S Nagaraja et al.¹² presented an image entropy based on anomaly detection scheme to detect Stegobot.

Stegobot discovers socialbot communication through social networks. Stegobot 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 the significant changes in images due to the embedding of secret messages. CJ Dietrich et al.¹⁴ 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. G Yan¹⁵ 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 data set of more than 40 million users and 1.4 billion follower relationships. VoteTrust⁴⁰ detects false social network accounts using trust-based polling and a voting graph model. It has two graphic models: an invitation diagram (directed graphic) and an acceptance graphic (a directed, weighted graphic). Ibrahim and Thanon⁴³ study a life cycle, the attack on the behavior, topologies, and technologies of Zeus. They designed Host Botnet Detection Software (HBD's) to detect Zeus botnet in user's computers. Ferrara et al.⁴⁴ discuss the characteristics of modern, sophisticated socialbots. They also discuss current efforts aimed at detection of socialbots in Twitter. They mainly analyze the socialbots on OSN platform. While, we have collected several classes of socialbot samples and conducted an in-depth analysis on end host.

Arms race

There are lots of antiVirus (AV) companies which have been set up to mitigate the threat of increasingly insidious malware technologies, which encourages an arms race between malicious software and detection systems. In the new social botnet, an OSN is used as the C&C channel. A socialbot is the automated software running on a user's computer in the background. It controls an account on a particular OSN and communicates with the botmaster through posting and receiving messages

from the OSN. The socialbot can use the account on the OSN to post messages, connect with people, read status updates from friends, and so on. Existing socialbots mainly use RSS and OSN APIs to communicate with OSN servers. We have also predicted that a web test automation toolkit in our previous work, called WTAR, is a prospective approach to designing malicious socialbots. Most of the existing detection countermeasures are server side. They could capture activities on certain OSNs from the whole social botnet. They also analyze the big data set to distinguish the infected accounts from normal user accounts. However, socialbots 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 data set to find out the socialbot process of the victim. However, the socialbot may update its behaviors, and the social botnet is still there. To protect the user host and OSN from socialbot, we prefer the integrated server-client defense mechanism. We have designed a simple prototype in section "Defense against socialbots on an end host."

The main difference between bots and malware is that bots use C&C channels. The main difference between bots and socialbots is that socialbots focus on OSNs as their main attacking target. Researchers have proposed a lot of methods of malware analysis and some analysis systems are very mature. Socialbots must receive commands from their botmasters via C&C channels. When a socialbot is in a malware analysis environment, it may not be able to fetch commands from its botmaster as usual. Analysis results are highly influenced by the behavior of the botmaster, because most socialbots that we could use to conduct an analysis have been published and their botmasters may have been shut down. So, when a socialbot is running in a malware analysis system, the socialbot may do nothing except connect to a social website. Many researchers have proposed various approaches for botnet detection. Some focus on detection of the bot on the end host. They analyze local network flows and system calls to detect the bot host or bot process. Some focus on network flows captured from network providers. They analyze these network flows with novel methods to find botnets. Because their C&C channels are based on OSN, socialbots are more difficult to detect with normal approaches. New approaches have been proposed for socialbot detection. Most research analyzes network flow from social websites to discover suspicious accounts and message. But to find socialbot processes hiding on user hosts, we need to detect the socialbots on the hosts. So, we conducted analysis of socialbot behavior on end hosts.

Limitations and ideas for future work

Our analysis was subject to several limitations, some specific to the socialbot data samples, some related to

the analysis environment, and some limited by the data set processing algorithm. For details, we can group the current limitations into three aspects as follows.

Socialbot samples

Ideally, we should collect as many socialbots as possible from their authors. Then, we can control both the bot clients and botmaster. To analyze socialbots in detail, it is better to fetch their source code. In this article, we were lucky enough to collect two kinds of socialbots. Regarding other socialbot, we are incredibly grateful for their authors' advice. Some socialbots in this article we re-implemented ourselves based on their descriptions. These socialbots may not be identical to the originals, but we guarantee that they have the same structures and functions. At the same time, we still do not have enough socialbot samples and continue to work on this situation.

Limited factors

In our analysis, we did not consider the infection stage, which is an important part of socialbot lifecycle. So, the analysis data set does not contain bootstrap information. There are also many other factors that we should analyze. For example, in the analysis of system calls in this article, we did not consider the delta time between system calls. We could also conduct analysis on the page view sequences of socialbots and benign applications.

Our analysis data set was shared on Dropbox.¹⁷ In the future, we will continue to collect socialbots and share them with other researchers. We will try to build an analysis system specific to socialbots and generate useful reports. We plan to use effective algorithms to process the socialbot features and propose novel approach to detection of socialbots on user hosts.

Conclusion

Socialbots are one of the most serious security threats to OSNs today. Although much research has been conducted on the many aspects of socialbot detection, little has targeted the host-based activity of socialbots. Due to a lack of real socialbots and corresponding botmasters, researchers rarely analyze socialbot behavior on user hosts. In our previous work,¹⁷ we proposed BotInfer for detecting bot hosts in the local network, and part of it is to analyze bot behavior on the user host. In this article, we aimed to shed light on socialbot behavior.

We collected various socialbots and benign social applications. The socialbots had client sides and master side. One socialbot was given to us directly by its author. Another class without source code is from Georgia Institute of Technology.²² We re-implemented

the other socialbots according to their descriptions and their author's advice. We also designed a class of socialbots focused on Facebook and Weibo. We also collected different benign applications that focus on different OSNs.

We have built a socialbot analysis system. The system includes virtual machines with the Windows XP OS. Socialbots and benign applications can run on the system. The socialbots can receive commands from their botmasters as usual. We used some monitoring software to log the behavior both of the socialbots and benign applications on user hosts. We also shared the socialbot behavior data set on the Internet.¹⁷

We have analyzed three aspects of the socialbot behavior data set on end hosts. For the network behavior of socialbots, we analyzed traffic statistics over time, traffic statistics in response to various commands, and traffic statistics of benign software and socialbots. For the system calls of socialbots, we analyzed the system call sequence and graph. For the process information, we analyzed the overhead of CPU usage and memory usage, the file access data set, and the process relationship tree.

We have proposed several metrics that can be used to process socialbot data set. The effectiveness of these measures and data sets is verified with clustering method. Although our analysis cannot detect socialbots on end hosts, understanding socialbot behavior is also important to the development of socialbot detection.

Acknowledgements

Socialbot samples were, to a great degree, provided by other researchers. The authors thank researchers for their generosity in sharing their samples (and their advice) with us.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the National Natural Science Foundation of China under Grant numbers 61472162 and 61170265.

References

1. Pony botnet server. <http://www.zdnet.com/two-million-stolen-facebook-twitter-yahoo-adp-passwords-found-on-pony-botnet-server-7000023915/> (2013, accessed 16 December 2015).
2. Mlot S. Facebook botnet scam, <http://www.pcmag.com/article2/0,2817,2413156,00.asp> (2012, accessed 16 December 2015).

3. Baltazar J, Costoya J and Flores R. The real face of Koobface: the largest web 2.0 botnet explained. *Trend Micro Res* 2009; 5(9): 10.
4. Jose N. Twitter-based botnet command channel, <http://www.arbornetworks.com/asert/2009/08/twitter-based-botnet-command-channel/> (2009, accessed 16 December 2015).
5. Lipovsky R. Pokeragent botnet, <http://www.welivesecurity.com/2013/01/29/pokeragent-botnet-stealing-over-16000-facebook-credentials/> (2013, accessed 16 December 2015).
6. Kartaltepe EJ, Morales JA, Xu S, et al. Social network-based botnet command-and-control: emerging threats and countermeasures. In: *Proceedings of the 8th international conference on applied cryptography and network security (ACNS)*, Beijing, China, 22–25 June 2010, pp.511–528. Berlin: Springer.
7. Nagaraja S, Houmansadr A, Piyawongwisal P, et al. Stegobot: a covert social network botnet. In: *Proceedings of the 13th international conference of information hiding (IH) (Lecture notes in computer science, vol. 6958)*, Prague, 18–20 May 2011, pp.299–313. Berlin: Springer.
8. Verkamp JP, Malshe P, Gupta M, et al. Facebot: an undiscoverable botnet based on treasure hunting social networks, <http://blog.jverkamp.com/2011/08/17/facebot-an-undiscoverable-botnet-based-on-treasure-hunting-social-networks/> (2011, accessed 18 December 2015).
9. Boshmaf Y, Muslukhov I, Beznosov K, et al. Design and analysis of a social botnet. *Comput Netw* 2013; 57(2): 556–578.
10. Singh A, Toderici AH, Ross K, et al. Social networking for botnet command and control. *Int J Comput Netw Inf Secur* 2013; 5(6): 11–17.
11. Chu Z, Gianvecchio S, Koehl A, et al. Blog or block: detecting blog bots through behavioral biometrics. *Comput Netw* 2013; 57(3): 634–646.
12. Nagaraja S, Houmansadr A, Piyawongwisal P, et al. Stegobot: a covert social network botnet. In: *Proceedings of the 13th international conference of information hiding (IH)*, Prague, 18–20 May 2011, pp.299–313. Berlin: Springer.
13. Wagner C, Mitter S, Körner C, et al. When socialbots attack: modeling susceptibility of users in online social networks. In: *Proceedings of the WWW'12 workshop on "making sense of microposts" (CEUR workshop proceedings, vol. 838)*, Lyon, 16 April 2012, pp.41–48. Europe: CEUR-WS.org, ISSN 1613-0073
14. Dietrich CJ, Rossow C and Pohlmann N. CoCoSpot: clustering and recognizing botnet command and control channels using traffic analysis. *Comput Netw* 2013; 57(2): 475–486.
15. Yan G. Peri-watchdog: hunting for hidden botnets in the periphery of online social networks. *Comput Netw* 2013; 57(2): 540–555.
16. James TL, Khansa L, Cook DF, et al. Using network-based text analysis to analyze trends in Microsoft's security innovations. *Comput Secur* 2013; 36: 49–67.
17. Socialbot datasets, <https://www.dropbox.com/sh/dlxzjm32zhufsr5/L6fP54pB2q> (accessed 16 December 2015).
18. Sdbot report by Symantec, http://www.symantec.com/security_response/writeup.jsp?docid=2002-051312-3628-99 (accessed 16 December 2015).
19. Peacomm.B report by Symantec, http://www.symantec.com/security_response/writeup.jsp?docid=2007-041314-1900-99 (accessed 16 December 2015).
20. Zbot report by Symantec, http://www.symantec.com/security_response/writeup.jsp?docid=2007-041314-1900-99 (accessed 16 December 2015).
21. Silva SS, Silva RM, Pinto RC, et al. Botnets: a survey. *Comput Netw* 2013; 57(2): 378–403.
22. Malware samples, <http://oc.gtisc.gatech.edu/> (accessed 16 December 2015).
23. ClamAV. <http://www.clamav.net/> (accessed 16 December 2015).
24. Facebook Graph API. <https://developers.facebook.com/docs/graph-api> (accessed 16 December 2015).
25. Process Monitor. <http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx> (accessed 16 December 2015).
26. Wireshark. <http://www.wireshark.org/> (accessed 16 December 2015).
27. Microsoft Network Monitor. <http://www.microsoft.com/en-us/download/details.aspx?id=4865> (accessed 16 December 2015).
28. Yet Another Process Monitor (YAPM). <http://yaprocm.sourceforge.net/> (accessed 16 December 2015).
29. TweetDeck. <https://about.twitter.com/products/tweetdeck> (accessed 16 December 2015).
30. Weibo desktop 2014, <http://desktop.weibo.com/> (accessed 16 December 2015).
31. Jaccard P. The distribution of the flora in the alpine zone. *New Phytol* 1912; 11(2): 37–50.
32. Weka: data mining software in java, <http://www.cs.waikato.ac.nz/ml/weka/> (accessed 18 July 2015).
33. Rieck K, Trinius P, Willems C, et al. Automatic analysis of malware behavior using machine learning. *J Comput Secur* 2011; 19(4): 639–668.
34. Rossow C, Dietrich CJ, Bos H, et al. Sandnet: network traffic analysis of malicious software. In: *Proceedings of the first workshop on building analysis datasets and gathering experience returns for security*, Salzburg, 10–13 April 2011, pp.78–88. New York: ACM.
35. Perdisci R, Ariu D and Giacinto G. Scalable fine-grained behavioral clustering of http-based malware. *Comput Netw* 2013; 57(2): 487–500.
36. Graziano M, Leita C and Balzarotti D. Towards network containment in malware analysis systems. In: *Proceedings of the 28th annual computer security applications conference (ACSAC)*, Orlando, FL, 3–7 December 2012, pp.339–348. New York: ACM.
37. Jacob G, Hund R, Kruegel C, et al. JACKSTRAWs: picking command and control connections from bot traffic. In: *Proceedings of the 20th USENIX security symposium*, San Francisco, CA, 8–12 August 2011. Berkeley, CA: USENIX Association.
38. Park YH and Reeves DS. Identification of bot commands by run-time execution monitoring. In: *Proceedings of the twenty-fifth annual computer security applications conference (ACSAC)*, Honolulu, HI, 7–11 December 2009, pp.321–330. New York: IEEE Computer Society.
39. Nagaraja S, Mittal P, Hong C, et al. BotGrep: finding P2P bots with structured graph analysis. In: *Proceedings of the 19th USENIX security symposium*, Washington,

- DC, 11–13 August 2010, pp.95–110. Berkeley, CA: USE-NIX Association.
40. Xue J, Yang Z, Yang X, et al. VoteTrust: leveraging friend invitation graph to defend against social network sybils. In: *Proceedings of the IEEE INFOCOM 2013*, Turin, 14–19 April 2013, pp.2400–2408. New York: IEEE.
41. Shanthi K and Seenivasan D. Detection of botnet by analyzing network traffic flow characteristics using open source tools. In: *Proceedings of the IEEE 9th international conference on intelligent systems and control (ISCO)*, Coimbatore, India, 9–10 January 2016, pp.1–5. New York: IEEE.
42. Pantic N and Husain MI. Covert botnet command and control using Twitter. In: *Proceedings of the 31st annual computer security applications conference*, Los Angeles, CA, 7–11 December 2015, pp.171–180. New York: ACM.
43. Ibrahim LM and Thanon KH. Analysis and detection of the zeus botnet crimeware. *Int J Comput Sci Inf Secur* 2015; 13(9): 121.
44. Ferrara E, Varol O, Davis C, et al. The rise of social bots. *Commun ACM* 2016; 59: 96–104.