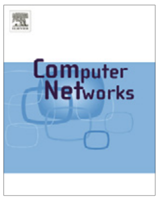
Computer Networks 57 (2013) 540–555

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Peri-Watchdog: Hunting for hidden botnets in the periphery of online social networks

Guanhua Yan [*,1]

Information Sciences (CCS-3), Los Alamos National Laboratory, United States

ARTICLE INFO

Article history:
Received 28 November 2011
Received in revised form 21 June 2012
Accepted 9 July 2012
Available online 30 August 2012

Keywords:
Twitter
Botnet
Anomaly detection
Graph theory
Centrality measure

ABSTRACT

In order to evade detection of ever-improving defense techniques, modern botnet masters are constantly looking for new communication platforms for delivering C&C (Command and Control) information. Attracting their attention is the emergence of online social networks such as Twitter, as the information dissemination mechanism provided by these networks can naturally be exploited for spreading botnet C&C information, and the enormous amount of normal communications co-existing in these networks makes it a daunting task to tease out botnet C&C messages.

Against this backdrop, we explore graph-theoretic techniques that aid effective monitoring of potential botnet activities in large open online social networks. Our work is based on extensive analysis of a Twitter dataset that contains more than 40 million users and 1.4 billion following relationships, and mine patterns from the Twitter network structure that can be leveraged for improving efficiency of botnet monitoring. Our analysis reveals that the static Twitter topology contains a small-sized core subgraph, after removing which, the Twitter network breaks down into small connected components, each of which can be handily monitored for potential botnet activities. Based on this observation, we propose a method called *Peri-Watchdog*, which computes the core of a large online social network and derives the set of nodes that are likely to pass botnet C&C information in the periphery of online social network. We analyze the time complexity of Peri-Watchdog under its normal operations. We further apply Peri-Watchdog on the Twitter graph injected with synthetic botnet structures and investigate the effectiveness of Peri-Watchdog in detecting potential C&C information from these botnets.

To verify whether patterns observed from the static Twitter graph are common to other online social networks, we analyze another online social network dataset, BrightKite, which contains evolution of social graphs formed by its users in half a year. We show not only that there exists a similarly relatively small core in the BrightKite network, but also this core remains stable over the course of BrightKite evolution. We also find that to accommodate the dynamic growth of BrightKite, the core has to be updated about every 18 days under a constrained monitoring capacity.

1. Introduction

Botnets, which are armies of compromised zombie machines sharing the same C&C (Command and Control) channels, have become a severe threat in cyberspace. The current portfolio of botnets that are familiar to cyber-security researchers span from traditional IRC and HTTP botnets to those that use self-organized peer-to-peer (P2P) networks for C&C channels. While a plethora of botnet detection and mitigation techniques have been proposed to defend against these botnets, some bot herders have shift their attention to online social networks, a new

* Tel.: +1 505 6670176; fax: +1 505 6679137.
E-mail address: ghyan@lanl.gov

[1] Los Alamos National Laboratory Publication No. 11-02621.

1389-1286/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comnet.2012.07.016

frontier of communication infrastructures with a rapidly growing population. For instance, currently Facebook has more than 500 million active users, half of which log into the website every day [10], and the number of Twitter users is going to reach 200 million by the end of 2011 [32].

The popularity of online social networks has opened a door for a number of malicious activities, such as spamming [13,37], malware and misinformation propagation [36,25], and link farming [11]. In this work, we consider a new type of attacks, which leverages the communication platforms provided by online social networks to disseminate C&C information for botnet operations. The reasons that online social networks are attractive for botnet operations are manifold. First, the numerous computers that use online social networks can be potentially recruited as bots by spreading malicious links through the messaging infrastructure provided by online social networks. The Koobface botnet, for instance, expands itself by sending obfuscated, malicious URLs to victims' friends when they visit Facebook or Twitter [29]. Second, the communication infrastructure of online social networks can be exploited as botnet C&C channels. Due to the huge number of messages that are delivered in online social networks, it is a daunting task to catch those that are used for botnet C&C. Meanwhile, many enterprise networks allow employees to visit online social networks during their work time but block P2P traffic at the enterprise gateways. Under such circumstance, botnets that use online social networks as their C&C channels can easily penetrate enterprise firewalls.

Botnets that took advantage of the Twitter network for their C&C have already been spotted in the wild [31]. For this first generation of Twitter-based botnets, individual bots visit the public profile page of a Twitter account controlled by the botmaster to obtain C&C information. As of this writing, automatic tools for disseminating C&C on Twitter have already been available [30]. The Achilles' heel of such botnets is, however, that all C&C information is made public and is thus subject to scrutiny by independent cyber-security researchers. Also, such botnets are essentially traditional HTTP-based botnets and the repeated pulling activities by individual bots to obtain C&C information expose them to local bot detectors [20,1,14]. We thus envision that some future botnets would circumvent these weakness by leveraging private messaging capabilities of Twitter-like open online social networks to organize individual bots, as suggested in [30]. In such botnets, C&C information is not exposed on public profile pages any more; rather it is delivered through private messages along following relationships among individual bots. Moreover, individual bots do not need to visit the public profile page repeatedly, as C&C information is pushed to them automatically through the communication infrastructures provided by online social networks.

This work is aimed at discovering this new type of botnets hidden in open online social networks such as Twitter. Given the sheer number of users and voluminous messages in Twitter-like open online social networks, detecting hidden botnets poses significant computational challenges. In this study, we conduct an in-depth graph-theoretic analysis of the Twitter network topology, and look for insights into how to efficiently monitor the Twitter network for potential botnet activities. We analyze a Twitter dataset that contains more than 40 million users and 1.4 billion following relationships [19], and mine patterns from the Twitter network structure that can be leveraged for efficient botnet monitoring. Our analysis reveals that the static Twitter topology contains a relatively small-sized core sugraph, after removing which, the Twitter network breaks down into small connected components, each of which can be handily monitored for potential botnet activities. Based on this observation, we propose a method called *Peri-Watchdog*, which computes the core of a large online social network and derives the set of nodes that are likely to pass botnet C&C information in the periphery of online social networks. We analyze the time complexity of Peri-Watchdog under normal operations. We further apply Peri-Watchdog to the Twitter graph injected with synthetic botnet structures and investigate the effectiveness of Peri-Watchdog in detecting potential C&C information from these botnets.

To verify whether patterns observed from the static Twitter graph can be extended to other open online social networks, we study another dataset, which was collected from a location-based online social network, BrightKite, and contains evolution of social graphs formed by its users almost half a year. Using the evolution data of the BrightKite network, we show that not only does there exist a similarly small-sized core in the BrightKite network, but also this core remains stable over the course of BrightKite evolution. We also find that to accommodate the dynamic growth of BrightKite, the core has to be updated about every 18 days under a constrained monitoring capacity.

The remainder of the paper is organized as follows. In Section 2, we present related work. In Section 3, we provide background information regarding the Twitter dataset we analyze in this work and discuss three different types of botnets that may hide in an online social network. In Sections 4–6, we present graph-theoretic analysis of the Twitter graph to show how to detect three different types of botnets, respectively. In Section 7, we provide the design of Peri-Watchdog, a framework that assists detection of botnets hidden in Twitter-like networks. In Section 8, we study the effectiveness of Peri-Watchdog using the Twitter graph injected with synthetic botnet structures. In Section 9, we further validate the observation made from the graph-theoretic analysis of the Twitter graph with a dataset collected from another online social network, BrightKite, and show how Peri-Watchdog performs on the dynamic graphs during BrightKite's evolution. Finally, we discuss the weakness of Peri-Watchdog in Section 10 and draw concluding remarks in Section 11.

2. Related work

Due to the severe threats posed by botnets to the cyber space, they have gained a lot of attention from researchers recently. A plethora of measurement work has been dedicated to understanding the nature of real-world botnets. These efforts include analysis of a variety of existing botnets, including IRC botnets [8,2], peer-to-peer based botnets [17,27], and HTTP-based botnets [28]. Botnets that exploit online social networks as their C&C channels have already been spotted in the wild [31], but study of such botnets by the research community is still in its infancy. This work considers potential strategies of C&C information dissemination by the herders of such botnets, and accordingly, proposes a method to monitor efficiently C&C messages of these botnets in a large online social network.

In addition to botnet measurement studies, a number of botnet detection methods have been proposed previously [15,16,14,12,39,35]. Gu et al. developed detection techniques that are based on strong correlation among observed activities of bots belonging to the same botnet [15,16,14]. TAMD is another network-level tool that identifies bot traffic by mining communication aggregates in which traffic flows share common characteristics [39]. Liu et al. proposed a host-based bot detection method that relies on a separate virtual machine to expose hidden bot activities [21]. Some other methods detect botnets by looking for suspicious flow-level communications in tier-1 ISP networks [18] or passively monitoring bot-generated DNS traffic [26]. Signature-based botnet detection techniques have also been proposed, including Rishi [12] and AutoRE [35]. Recently, Zeng et al. studied the effectiveness of detecting large botnets at the infrastructure level [43]. Most of these existing botnet detection methods are deployed in an enterprise network environment or at the ISP level. Peri-Watchdog, however, is developed specifically to assist detection of suspicious botnet activities in open online social networks such as Twitter.

Previously, researchers have also worked on detecting anomalous activities based on social networks. Grier et al. analyzed a large number of URLs posted on Twitter and observed that blacklists respond to spamming URLs too slowly [13]. Many other related efforts are dedicated to detecting sybil nodes in these networks [42,41,9,22]. Although similar in nature from a graph-theoretic standpoint, sybils and bots serve different purposes. A sybil node is introduced into a distributed system by the attacker to perform activities that are illegitimate relative to the normal functionality of the system. For instance, in a peer-to-peer system, the attacker can introduce sybil nodes to monitor other users' traffic, or launch collusion attacks. Hence, the goal of sybil attacks is usually related to the functionality provided by the distributed system where those sybil nodes reside. By contrast, botnet herders are only interested in using the communication platform provided by online social networks to deliver C&C information. The existence of hidden botnets thus does not pose a threat to the normal functionality of online social networks.

From a detection point of view, sybils and bots can both be treated as anomalies introduced into the normal network by attackers. Recent analysis [33] shows existing Sybil defense schemes largely fall into the detection of *local communities*, where nodes inside the community are more tightly knit together than they with those outside the community. Such an assumption, however, does not necessarily hold for botnet operation. On one hand, as long as there exists a bot-only path from the master node to each individual bot, the botmaster is able to broadcast his C&C information. On the other hand, for open online social networks such as Twitter, bots can follow a large number of normal users. Hence, the subgraph containing all the bots does not necessarily form a local community. Based on this observation, Peri-Watchdog does not make the assumption that bots form a local community as most existing Sybil defense techniques do.

Graph-theoretic analysis has been applied to identify bots in large networks. Zhao et al. developed BotGraph [44] to detect large-scale spamming botnets. They analyzed 2 months of Hotmail log and constructed a user-user graph in which a node represents a Hotmail account and two nodes are connected by an edge when they shared the same IP address. Recently, Nagaraja et al. developed a framework called BotGrep [24] to find P2P botnets at the ISP level. Their method leverages fast mixing times of P2P network structures to separate P2P flows from other types of network flows. Although under the same umbrella of graph-theoretic analysis as these previous methods, our approach concerns relative ranking of nodes in a graph and addresses botnet detection in a different network environment.

3. Threat model and Twitter dataset

3.1. Threat model

In this work we are interested in monitoring hidden botnet activities in open online social networks, which have the following two distinguishing features: (1) *The freedom to open a new account in the network*. Although many online social networks perform CAPTCHA test when a new user registers, numerous instances in the past have shown that it is not a bullet-proof technology that prevents attackers from circumventing its protection [3]. In this work, we assume that the attacker has the capability of creating a large number of fake accounts in an open online social network. (2) *The freedom to establish a uni-directional friendship with any other user*. Hence, LinkedIn cannot be treated as an open online social network because a relationship between two users is bidirectional and thus requires confirmation from both sides. In an open online social network, a directional relationship dictates how information flows. For example, when user *A* follows user *B* in Twitter, information flows from *B* to *A*. Depending on who initiates such a directional relationship, there are two types of online open social networks: *pull-mode* and *push-mode*. In a pull-mode open online social network, a user decides whose information she is willing to receive (i.e., who she follows). Obviously, Twitter is a pull-mode online social network. By contrast, in a push-mode open online social network, a user decides who can receive her information (i.e., who can follow her). An example of this type of online social network is BrightKite [5,6], which we will use later for cross validation.

For clarity, we assume a bot to be a bot-controlled account, rather than a compromised machine, in an open online social network. When a botherder wants to hide a botnet inside an open online social network, he can create the following relationships among bots at will. However, as the botmaster needs to spread C&C information through these following relationships created among bots, we assume that there exists a master bot (i.e., the "snakehead" of the botnet) from which C&C information can flow to every individual bots in the botnet. In the simplest form, the botnet can be a star topology where the master bot is located at the center. Moreover, if the online social network is a pull-mode one, the botmaster can let a bot follow any normal user; otherwise, if it is a push-mode online social network, the botmaster can let a bot be followed by any normal user. In the following discussion, we focus on pull-based online social networks unless explicitly mentioned otherwise.

In this work, we will focus on three types of botnets that attempt to hide themselves in open online social networks. The first type is *standalone botnets*, which are isolated from the normal online social network. The second type is *appendix botnets*. If it is a pull-mode online social network, appendix botnets have nodes following users in the normal online social network but not vice versa; if it is a push-mode one, appendix botnets have nodes followed by users in the normal online social network but not vice versa. The third type of botnets, which we call *crossover botnets*, have following relationships in both directions between bots and normal users. Supposing that the normal online social network is the original Twitter network, the three types of botnets are illustrated in Fig. 1.

When a pull-mode online social network is considered, the key challenge for the attacker is to find normal users who are willing to follow bots. This usually can be done through phishing attacks, in which a bot impersonates as a legitimate user (sometimes, a celebrity) to attract interested normal users to follow it. In this context, it is more difficult to launch spearphishing attacks, which target specific normal users and allure them to follow a bot.

3.2. Twitter dataset

We start this work by analyzing the structure of the graph formed by following relationships of Twitter users. Our goal is to use graph-theoretic analysis to identify important characteristics of this graph, which are further utilized to assist detection of malicious botnet activities. The Twitter dataset we use in this study is collected by Kwak et al. [19], and contains 41,652,230 user profiles and 1,468,365,182 following relationships. The dataset was obtained by crawling the Twitter network between June 31 and September 24 in 2009. The giant component was crawled by seeding from Perez Hilton who has more than one million followers, and the remaining user profiles were discovered through the trending topics their tweets referred to. In total, there were 1,838,934,111 tweets sent by the users. We refer interested readers to the article [19] for more details on the dataset. For clarity, Table 1 summarizes the notations used throughout this paper. In this table, a directed edge (u,v) in a graph means that user v follows user u. Here, the direction of an edge indicates the direction of information flow between two socially connected online users. In graph theory parlance, the giant weakly (strongly) connected component of a graph means the largest weakly (strongly) connected component of this graph.

In the following sections, we will analyze the Twitter graph and look for features that can be used for efficient monitoring of botnet activities in open online social networks. These features will form the foundation of a generic botnet monitoring framework, Peri-Watchdog, which computes a list of user accounts that can potentially pass C&C information in a hidden botnet.

4. Twitter analysis: standalone botnets

A standalone botnet does not have any bot that has following relationships with a normal user. Assuming that bots in a standalone botnet are weakly connected so that the master node can deliver C&C information to all individual bots, the botnet introduces a new weakly connected component (WCC) into the Twitter graph. We thus investigate the distribution of the sizes of WCCs in G_t. The results are depicted in Fig. 2. Clearly, the sizes of WCCs in G_t are highly skewed: the giant weakly connected component of G_t, i.e., $\mathcal{W}(G_t)$, has 41,652,156 users, which leaves out only 74 nodes and 112 edges in the entire Twitter graph. The second largest WCC has only three nodes. Moreover, the users outside the giant WCC seem to be dormant during the data collection period, as they did not produce any tweets. Hence, if a large standalone botnet, say, with thousands of bots, is formed in the Twitter network, it would stand out as a second largest WCC and can thus easily be monitored for malicious activities.

We, however, should cautiously interpret this observation in practice. This is because user profiles outside the giant WCC of the Twitter graph were discovered through those that mentioned trending topics in Twitter. As not all Twitter users refer to trending topics in their tweets, the dataset we obtained may not have complete information about those users outside the giant component. Even

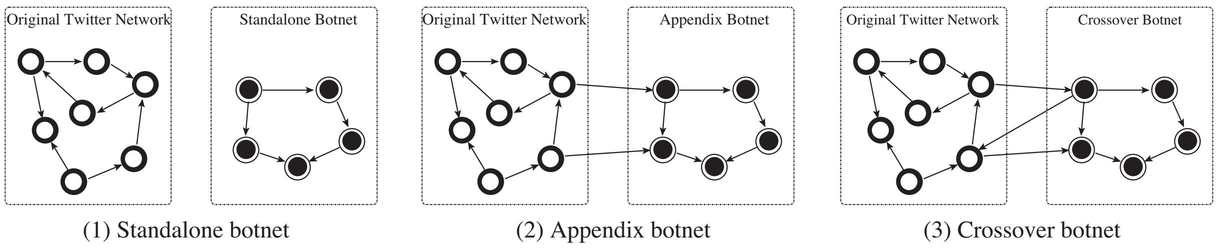


Fig. 1. Illustration of three types of botnets.

Table 1
Notations.

Notation	Meaning
$G_t(V_t, E_t)$	Original Twitter graph
$\mathcal{W}(G)$	Giant weakly connected component of graph G
$\mathcal{S}(G)$	Giant strongly connected component of graph G
$\mathcal{V}(G)$	Vertex set of graph G
$\mathcal{T}(G)$	Transpose graph of G
$\mathcal{R}(G, V)$	Set of nodes in graph G reachable from V
$\mathcal{H}(G, V)$	Subgraph of graph G induced on nodes in set V

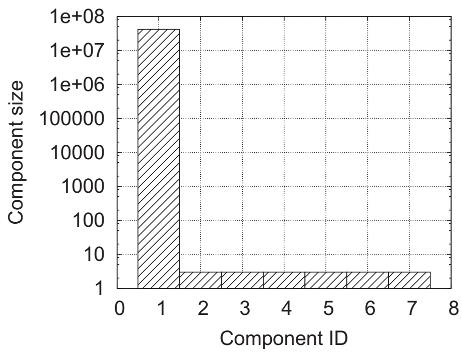


Fig. 2. Weakly connected components of G_t.

so, the Twitter server is still at an advantageous position to discover standalone botnets because it knows the full Twitter graph, rather than relying on crawling by a third party. Computing the set of nodes outside the giant WCC takes $O(|V_t| + |E_t|)$ time.

5. Twitter analysis: appendix botnets

An easy extension to standalone botnets is having some of the bots follow users in the giant WCC of $G_t, \mathcal{W}(G_t)$. As Twitter does not require the followee to agree with any following relationship initiated from another user, the attacker can easily turn a standalone botnet into an appendix botnet. We, however, note that as normal users do not follow bot nodes in an appendix botnet, normal nodes and bots cannot coexist in the same strongly connected component (SCC) of the Twitter graph. The sizes of the SCCs in $\mathcal{W}(G_t)$ are depicted in Fig. 3.

$\mathcal{S}(\mathcal{W}(G_t))$, the giant SCC of $\mathcal{W}(G_t)$, has 33,479,727 nodes in it, comprising 80.4% of the nodes in $\mathcal{W}(G_t)$. As there are still a significant portion of nodes that are in $\mathcal{W}(G_t)$ but not in $\mathcal{S}(\mathcal{W}(G_t))$, we further sort out nodes that cannot be in an appendix botnet. Note that no normal users follow bots in an appendix botnet. If we assume that the majority of the nodes in $\mathcal{W}(G_t)$ *are not* bots, then no nodes in $\mathcal{S}(\mathcal{W}(G_t))$ should belong to an appendix botnet. Also, nodes that are followed by any of those in $\mathcal{S}(\mathcal{W}(G_t))$, either directly or indirectly, should also not belong to an appendix botnet.

Hence, we compute the entire set of nodes, denoted by R, that are reachable in the *transpose graph* of $\mathcal{W}(G_t)$ from the nodes in $\mathcal{S}(\mathcal{W}(G_t))$, i.e., $\mathcal{R}(\mathcal{T}(\mathcal{W}(G_t)), \mathcal{V}(\mathcal{S}(\mathcal{W}(G_t))))$. Note that in a directed graph G, the set of nodes *reachable* from a vertex v contains all the nodes that have a directed path from v in G. Note that $\mathcal{V}(\mathcal{S}(\mathcal{W}(G_t))) \subseteq R \subseteq \mathcal{V}(\mathcal{W}(G_t))$. Clearly, bot nodes belonging to an appendix botnet must *not* be contained in set R, so a bot node in an appendix botnet must belong to set $\mathcal{V}(\mathcal{W}(G_t)) \setminus R$. We thus analyze the structure of the subgraph induced on the nodes in this set, which is $\mathcal{H}(\mathcal{W}(G_t), \mathcal{V}(\mathcal{W}(G_t)) \setminus R)$. For clarity, let H_{app} denote this subgraph. Fig. 4 depicts the sizes of WCCs with at least 10 nodes in H_{app} in non-increasing order. The giant component has only 159 nodes.

In practice, in order to discover large appendix botnets in the Twitter network, we can analyze the Twitter graph as described above to obtain H_{app} and then get its weakly connected components. If there are such components whose sizes are sufficient to contain a large appendix botnet, we monitor the tweets generated by the nodes in it for potential malicious activities. The computational complexity of the aforementioned graph analysis is $O(|V_t| + |E_t|)$.

6. Twitter analysis: crossover botnets

In a crossover botnet, there exist following relationships in both directions between bot accounts and normal users. Recent measurement studies revealed that finding normal users willing to follow sybil accounts in an open online social network such as Twitter is not difficult due to existence of so-called *social capitalists* [11] or *social butterflies* [37], who follow back anyone that follows him or her. Since a crossover botnet in Twitter can be treated as a collection of sybil nodes that are connected to the main network in both directions, one would wonder whether a plethora of techniques that have already been proposed to detect sybil nodes [40,42,41,9,34,7] can be leveraged to detect crossover botnets. Common to these sybil detection techniques is a fundamental assumption that sybil nodes form a community in which nodes are tightly connected among each other but are only loosely connected to outside nodes [33]. Hence, if the density of connections inside a crossover botnet is higher, statistically speaking, than that of edges that connect the botnet to the main network, it is possible for us to identify the botnet structurally. This assumption, however, severely limits the application of existing sybil detection methods to finding crossover botnets, as for a botnet to function properly, bots do not have to connect closely to each other. Rather, as long as each bot can receive C&C from a master node along a bot-controlled path, bots can perform tasks designated by the botmaster. Moreover, in an open online social network like Twitter, it is not difficult for the botmaster to manipulate the topology of the botnet and recruit normal users to follow bot accounts. Consider, for instance, a botnet that has 100 bots, which form a line graph. Each bot is assumed to be seamlessly integrated into a separate community formed by normal users. Any community structure detection algorithm will likely reveal these communities, each containing a single bot, but the knowledge of these communities does not help us identify the botnet. This simple example suggests that *the assumption of bots forming a closely knitted community is too strong for detecting crossover botnets*. Interestingly, a recent work by Yang et al. reveals that the assumption of sybil communities does not necessarily hold in real-world online social networks [38].

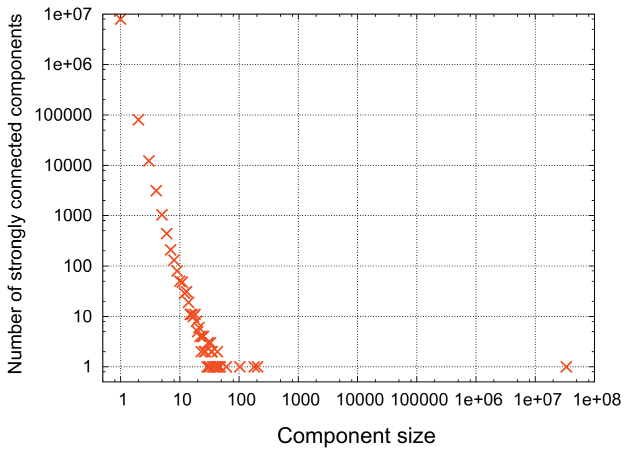


Fig. 3. Strongly connected components of $\mathcal{W}(G_t)$.

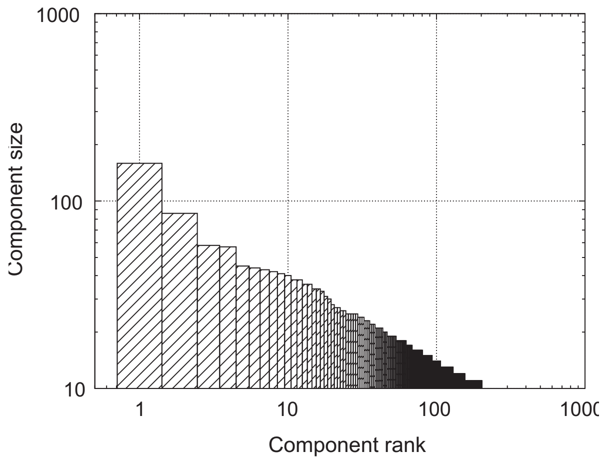


Fig. 4. Sizes of weakly connected components in H_{app}.

On another note, even though bots in a crossover botnet form a local community that is detectable by existing community detection algorithms, we still need to tease it out as a “bad” community by inspecting traffic circulating in it. The problem of finding communities suspicious of operating a botnet still poses another significant computational challenge in addition to community detection. Moreover, given the dynamics of open online social networks like Twitter and the fact that users commonly belong to multiple communities (for example, a user may be both a music and sport fan), structures of communities may change over time, making it difficult to reach stable knowledge whether a community is indeed “bad”.

Given the above observations, we take a different direction in this work to detect crossover botnets. Our goal is to detect crossover botnets of decent sizes, and thus we will not consider those botnets that consist of only a small number of nodes. The key idea is to *identify a set of suspicious nodes which are able to pass C&C information to a large number of individual bots, either directly or indirectly*. Based on this intuition, we first introduce another metric that is more indicative of its potential attack capability as a botnet than the component size. This is the *capacity* of a component, which is defined as follows. Let $H(V, E)$ be a directed graph. The number of nodes that are reachable from node $v \in V$ in H is denoted $\alpha(v)$. The capacity of graph H, which is denoted $\beta(H)$, is given by: $\beta(H) = \max_{v \in V} \alpha(v)$. Supposing that a WCC of Twitter is used as a botnet, its capacity provides an upper bound on the number of nodes that can receive C&C messages sent from any single source in the graph. Therefore, it can be reasonably assumed that a large crossover botnet of interest has a large capacity; that is to say, there exists a single source in it from which C&C information can be delivered to a large number of individual bots.

6.1. Ranking of Twitter users

To find suspicious nodes that are able to pass C&C information to a large number of bots, we experiment with the following heuristic: we aim to find a small-sized core of the large online social network that comprises only nodes trusted not to be bots. Ideally, the core should be *stable* over the evolution course of the online social network so that we can focus on monitoring nodes in the periphery, i.e., those outside the core, for potential botnet activities, or those nodes that are newly admitted into the core. From the perspective of computational complexity, a good core should also render it much easier to monitor potential bot nodes in the periphery.

In the following, we shall show how to obtain such a core of a large online social network based on the “importance” of individual nodes. In the parlance of graph theory, the “importance” of a node in a graph is its centrality measure. In the following, we rank nodes in $\mathcal{H}(\mathcal{W}(G_t), \mathcal{R}(\mathcal{T}(\mathcal{W}(G_t)), \mathcal{V}(\mathcal{S}(\mathcal{W}(G_t)))))$, i.e., the subgraph induced by the nodes reachable from the giant SCC of the giant WCC of the Twitter graph on its transpose graph.

6.1.1. Pagerank centrality

The pagerank of a node v, denoted $C_p(v)$, in a connected graph $G(V, E)$ is defined as follows:

$$C_p(v) = 1 - d + d \times \sum_{\forall u \in Out(v)} C_p(u)/i(u), \tag{1}$$

where $Out(v)$ is the set of nodes that have edges pointing from node v, $i(u)$ is the outdegree of node u, and d is the damping factor. Usually d is set to be 0.85.

6.1.2. Outdegree centrality

The outdegree centrality of node v, denoted by $C_o(v)$, is the same as its outdegree.

6.1.3. Betweenness centrality

Let σ_{st} denote the number of shortest paths from s to t where $s \neq t$. Also let $\sigma_{st}(v)$ be the number of shortest paths from s to t that have node v on them, where $s \neq v \neq t$. Then, the betweenness centrality of node v in graph $G(V, E)$ is defined as: $C_b(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$. To calculate the betweenness centrality of each node in the Twitter graph efficiently, we implemented Brandes' algorithm [4], whose computational complexity is $O(nm)$, where n and m are the number of vertices and edges in an unweighted graph, respectively. This algorithm iteratively chooses each node as a source, and from this source, computes the shortest paths to all other nodes and accordingly updates their betweenness centrality measures.

Even with the faster implementation of the betweenness calculation algorithm, a full computation of it on a graph as large as the Twitter network is still computationally prohibitive. To speed up the computation, we randomly draw k nodes from the graph as the source nodes, rather than considering all possible n sources, and use Brandes' algorithm to compute the betweenness centrality measure of each node based on only these k source nodes. Hence, the time complexity is now reduced to only $O(km)$.

Fig. 5(1) depicts the convergence of the computation when k grows. The overlapping ratio is defined as the fraction of overlapping nodes among the top n ones between cases when $k = i$ and $i + 100$. We observe that for the top n nodes when n is sufficiently large (say, above 1 million), the computation converges well after k reaches 1000.

Table 2
Overlapping ratio among the top 15,000,000 nodes.

	Pagerank	Outdegree	Betweenness	Closeness
Pagerank	X	65.2%	67.3%	66.6%
Outdegree	65.2%	X	72.1%	70.0%
Betweenness	67.3%	72.1%	X	64.6%
Closeness	66.6%	70.0%	64.6%	X

6.1.4. Closeness centrality

Let $d(s, t)$ denote the length of the shortest paths from node s to t. The closeness centrality measure of node v in graph $G(V, E)$ is given by: $C_c(v) = \frac{1}{\sum_{t \in V} d(v,t)}$. We have a similar scalability problem, and thus randomly draw k nodes as the destination node set V'_k. The closeness centrality measure of node v in graph $G(V, E)$ is simplified as $C_c(v) = \frac{1}{\sum_{t \in V'_k} d(v,t)}$. The convergence of the computation is depicted in Fig. 5(2). For the top n nodes when n is sufficiently large, the computation converges well when k hits 900.

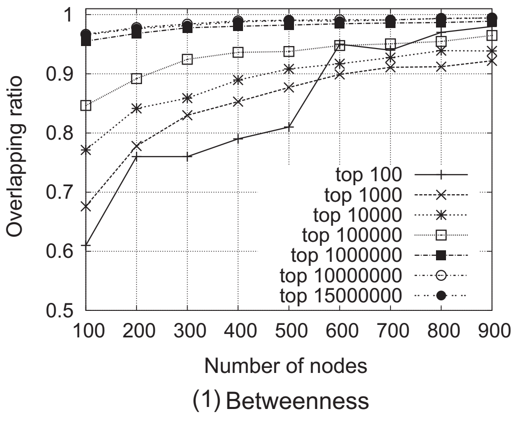
6.1.5. Differences among centrality measures

Table 2 presents the overlapping ratios for the top 15,000,000 nodes among different centrality measures. We note that the distinctions among different centrality measures are significant, suggesting that the four centrality measures evaluate the importance of nodes in the Twitter graph differently.

6.2. Core of Twitter Graph

We assume that there is a core of the Twitter graph, after removing which the graph breaks down into small pieces. Recall that in the giant WCC of the Twitter graph, i.e., $\mathcal{W}(G_t)$, about 80% of the nodes in it are strongly connected. We now remove from $\mathcal{W}(G_t)$ the top k nodes in the ranking based on each centrality measure, and see how the size of the giant SCC varies with k.

The results are presented in Fig. 6. From it, we observe that nodes with the highest betweenness or outdegree centrality measures are more likely to comprise the core



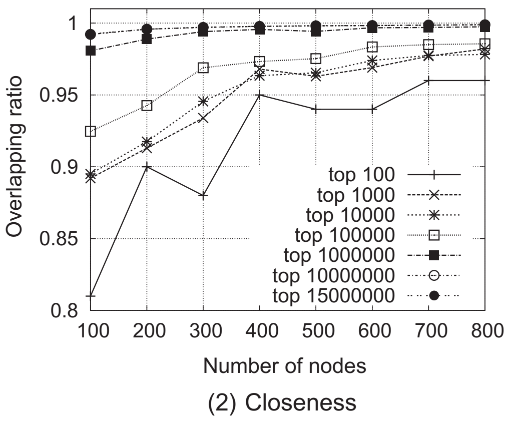


Fig. 5. Convergence behavior of centrality measure computation.

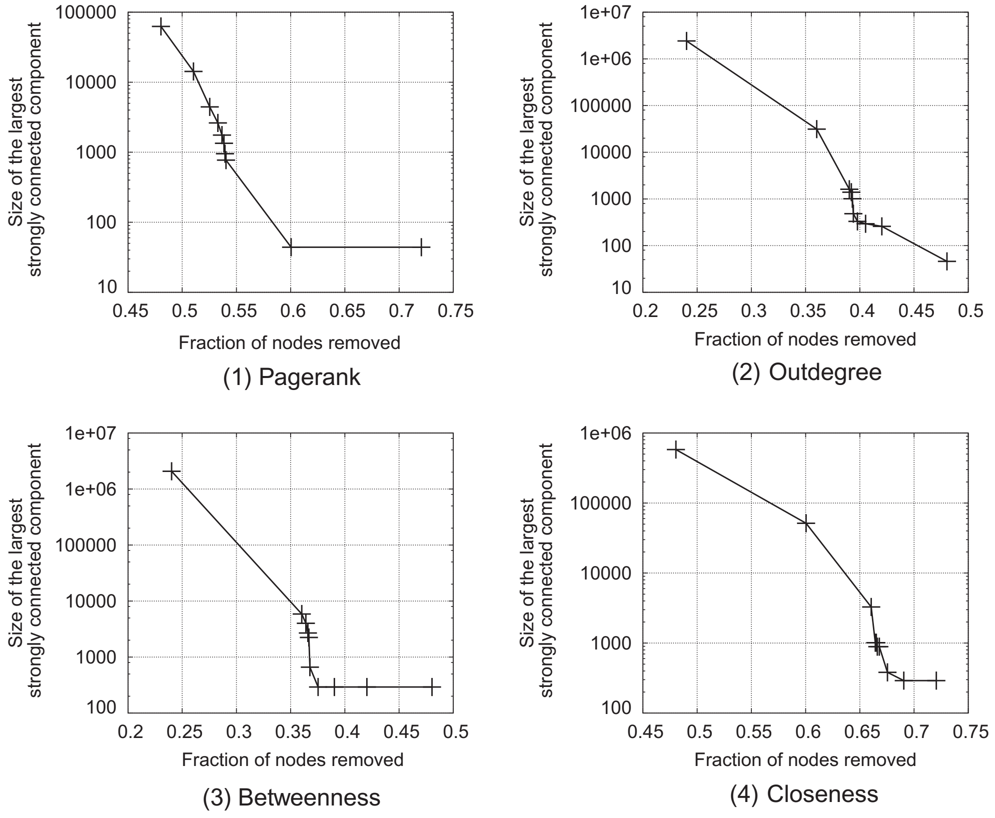


Fig. 6. Size of the largest strongly connected component after removing the top k nodes in each ranking.

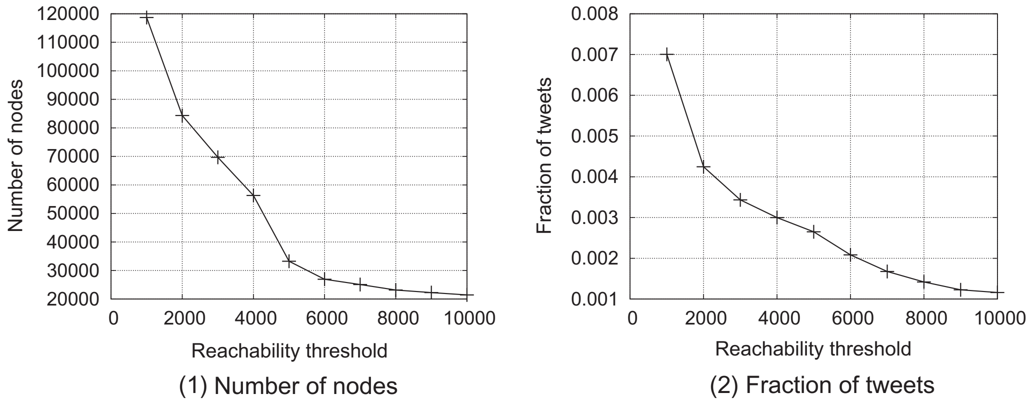


Fig. 7. Number of nodes and fraction of tweets under different reachability thresholds.

of the Twitter graph than the other two. This is particularly true for the betweenness centrality measure because it measures how frequently nodes appear on the shortest paths in the graph. For instance, in order to break the Twitter graph into components where the size of the giant SCC is smaller than 1000, for betweenness and outdegree centrality measures, removing the top 36.8% and 39.4% of the nodes in $\mathcal{W}(G_t)$, respectively, would be sufficient; but for pagerank and closeness centrality measures, at least the top 54.0% and 64.6% of the nodes have to be removed, respectively.

Given this observation, we use the betweenness and outdegree centrality measures to define the core of the Twitter graph in this work. Among the top 36.8% of the nodes in $\mathcal{W}(G_t)$ based on betweenness centrality, we rank them based on their outdegrees. We find that by removing

14,121,948 nodes with the highest outdegrees (34.3% of nodes in $\mathcal{W}(G_t)$) we are still able to break the Twitter graph into components among which no SCCs have more than 714 nodes. Considering the subgraph induced on these nodes, we obtain its giant SCC, which has 14,100,109 nodes in it. After only removing these nodes from $\mathcal{W}(G_t)$, the giant SCC in the remaining graph still has only 718 nodes. We let this giant SCC be the core of the Twitter graph, denoted by $Q(G_t)$.

6.3. Reachability of nodes outside the core

We now calculate the reachability of the nodes outside the core, i.e., the graph $\mathcal{W}(G_t) \setminus Q(G_t)$. Let $\widehat{Q}(G_t)$ be the residual graph after removing $Q(G_t)$ from $\mathcal{W}(G_t)$. Our goal is to find the set of nodes in $\widehat{Q}(G_t)$ from which the number of reachable nodes in the directed graph are above a certain threshold $\hbar$. Here, note that it is not sufficient to compute the capacity of each WCC in $\widehat{Q}(G_t)$. This is because in a WCC, any node in it can be the node controlled by the botmaster to send C&C messages.

In Fig. 7, we show the number of nodes whose reachability is above a certain threshold $\hbar$under different settings of $\hbar$. Clearly, the number of nodes that we need to monitor for large cross-over botnet activities is very small among the entire Twitter graph. For instance, to discover cross-over botnets of size no smaller than 1000, we need to monitor only 118,696 nodes, 0.29% of the entire Twitter graph producing 0.7% of all the tweets; to detect cross-over botnets of size no smaller than 10,000, we need to monitor only 21,435 nodes, 0.05% of the entire Twitter graph producing 0.12% of all the tweets. Hence, we can reduce the computational cost of monitoring cross-over botnets significantly as opposed to monitoring the entire network.

7. Design of Peri-Watchdog

Our key observations from Sections 4–6 are summarized as follows: (1) Outside the giant WCC of the Twitter graph, most nodes form only small WCCs. (2) Outside the SCC of the Twitter graph and the nodes that those in the SCC of the Twitter graph follow, either directly or indirectly, most nodes form only small WCCs; (3) Given the giant WCC G, we can find a core such that removing it breaks down G, that is to say, most nodes in the residual graph have low reachability. Given these observations, we design *Peri-Watchdog* to assist detection of botnets based on Twitter-like online social networks. The key idea behind Peri-Watchdog is to narrow down the list of nodes that can possibly pass C&C information of a hidden botnet. Machine learning or watermark detection techniques can be used to discover whether communication messages contain C&C information, which may be hidden in steganography [23]. Computationally, these techniques are usually not cheap, and it is thus impractical to apply them on all traffic in a large online social network. By examining only the traffic generated from suspicious nodes identified by Peri-Watchdog, we are relieved from inspecting every communication message in the network when searching for potential botnet activities.

Algorithm 1. Online detection of hidden botnets given graph $G^{(k)}$, and X, the set of nodes in the core that need to be vetted

```
1: $newcore \leftarrow false$
2: **RESTART:**
3: $L \leftarrow X$
4: {**Get the list of nodes that can be the master of standalone botnets**}
5: Compute $\mathcal{W}(G^{(k)})$ and $A = G^{(k)} \setminus \mathcal{W}(G^{(k)})$
6: **for** each node $v$ in $A$ **do**
7:    if the reachability from $v$ is not smaller than $\alpha$, add $v$ to $L$
8: **end for**
9: {**Get the list of nodes that can be the master of appendix botnets**}
10: $R \leftarrow \mathcal{R}(\mathcal{T}(\mathcal{W}(G^{(k)}), \mathcal{V}(\mathcal{S}(G^{(k)}))))$
11: $B \leftarrow \mathcal{H}(\mathcal{W}(G^{(k)}), \mathcal{V}(\mathcal{W}(G^{(k)})) \setminus R)$
12: **for** each node $v$ in $B$ **do**
13:    if the reachability from $v$ is not smaller than $\alpha$, add $v$ to $L$
14: **end for**
15: {**Get the list of nodes that can be the master of crossover botnets**}
16: **if** Core $Q$ exists **then**
17:    $C \leftarrow \mathcal{W}(G^{(k)}) \setminus Q$
18:    $L' \leftarrow \emptyset$
19:    **for** each node $v$ in $C$ **do**
20:       if the reachability from $v$ in $C$ is not smaller than $\alpha$, add $v$ to $L'$
21:    **end for**
22:    **if** $|L| + |L'| \leqslant \beta$ or $newcore$ is true **then**
23:       Add all nodes in $L'$ to $L$
24:       Goto **MONITOR**
25:    **end if**
26: **end if**
27: {**We need to generate the core** $Q$}
28: $X \leftarrow \emptyset$
29: Calculate the core of $\mathcal{W}(G^{(k)})$ as $Q'$ using Algorithm 2
30: **for** each node $v$ in $Q'$ but not in $Q$ **do**
31:    Add $v$ to $X$
32: **end for**
33: $Q \leftarrow Q'$, $newcore \leftarrow true$
34: goto **RESTART**
35: **MONITOR:**
36: Monitor traffic of nodes in set $L$ for potential botnet activities
```

We now discuss how Peri-Watchdog operates in an online fashion in a dynamic environment where new nodes constantly join the network and following relationships change over time in the network. Suppose that the graphs observed in the network are represented as $\{G^{(k)}\}$, where $k = 1, 2, \ldots$. Our goal is to find the list of nodes that can potentially pass C&C information to at least α individual bots. During the process, we maintain the core of the graph as Q. Also, we assume that Peri-Watchdog is only capable of monitoring β nodes for potential botnet activities.

Algorithm 1 presents how Peri-Watchdog operates at the kth timestep. This algorithm has three steps. *First*, the algorithm checks all the nodes that do not belong to the giant WCC, and for each one of them, if it has reachability no smaller than α, its traffic is inspected for potential botnet activities. *Second*, the algorithm obtains the SCC of the graph, and the list of nodes that nodes in this component follow either directly or indirectly. Removing all these nodes from the graph yields a residual graph. For each node in this residual graph, if it has reachability no smaller than α, its traffic is inspected for potential botnet activities. *Third*, the algorithm removes all nodes in the core from the giant WCC of the graph; for each node in the residual graph, if it has reachability no smaller than α, its traffic is inspected for potential botnet activities. Here, as it is possible that the core is outdated, the algorithm checks whether there are too many nodes to monitor. If so, the core is regenerated. After a new core is generated, Peri-Watchdog also monitors nodes that are newly admitted to the core. Note that it is possible that the eventual list of nodes that need to be monitored from Algorithm 1, i.e., L, has more than β nodes. This occurs even when a new core is computed, the size of L is still larger than β. Under such a circumstance, Peri-Watchdog may not be able to monitor all the nodes in set L in its capacity.

The method for generating the core of a graph is presented in Algorithm 2. The algorithm first ranks all the nodes in the graph based on their betweenness centrality measures. Note that Peri-Watchdog maintains a blacklist of suspicious nodes that have been discovered, which is denoted by S in the algorithm. We assign -1 to the betweenness centrality measure of each node in set S to ensure that they do not appear in the core. Next, it tries to find the top x nodes, after removing which, the size of the giant SCC is below a certain threshold γ. To do so, the algorithm uses a binary search method and stops the search when the range [*low*, *high*] is sufficiently small (i.e., $high - low$ is no greater than a predefined fraction δ of the network size). Based on the results from the betweenness centrality, the algorithm further ranks the top x nodes according to their outdegree centrality. Similarly, the algorithm uses a binary search method to find the top x' ones of those x nodes, removing which leads to a giant SCC smaller than γ. The core of the graph is the giant SCC of the subgraph induced on these x' nodes.

Complementary to the online detection counterpart shown in Algorithm 1 is a component of Peri-Watchdog that checks the sanity of nodes in the core in an offline fashion. The purpose of this offline component is to prevent bot nodes from slipping into the core without being caught by the online detection component, as well as compromised core nodes from being exploited as master bot nodes for disseminating C&C information. To deal with the large number of messages generated by core nodes, the offline component can adopt efficient algorithms such as sampling, or use anomaly detection methods based on statistical profiles of core nodes' behaviors. The details of finding bot activities in an offline fashion are out of the scope of this work, and we plan to explore along this line in the future based on real-world message datasets.

7.1. Complexity analysis

We first consider the time complexity of core generation in Algorithm 2. Suppose that the input graph H has n nodes and m edges. Given a single source, $O(m)$ time is needed for calculating the betweenness centrality measures. As mentioned in Section 6.1, we can randomly sample hundreds of sources to achieve good convergence for graphs as large as the Twitter topology. Given the number of sources r sampled, the overall computational cost on calculating betweenness centrality measures is thus $O(rm)$ using Brandes' algorithm. Given graph H, its SCC can be obtained in time $O(n + m)$. As we use a binary search method, the overall computational cost on obtaining set F and F' is $O(n\log n + m\log n)$. Hence, the overall computational cost of Algorithm 2 is $O(rm + m\log n + n\log n)$.

Algorithm 2. Calculate the core of directed, weakly connected graph H with suspicious node set S

```
1: $w \leftarrow$ number of nodes in $H$
2: Rank the nodes in $H$ based on their betweenness
   centrality as $v_1, v_2, \ldots, v_w$ (if a node belongs to the
   suspicious node set $S$, we change its betweenness
   centrality measure to $-1$)
3: $low \leftarrow 1$, $high \leftarrow w$, $D \leftarrow \{(w, 0)\}$
4: while $high - low > \delta w$ do
5:    $mid \leftarrow \lfloor (low + high)/2 \rfloor$
6:    $H' \leftarrow H \backslash \{v_i\}_{i=1,\ldots,mid}$
7:    $s \leftarrow$ the size of the giant SCC in $H'$
8:    Add $(mid, s)$ to set $D$
9:    If $s \geqslant \gamma$ then $low \leftarrow mid$ else $high \leftarrow mid$
10: end while
11: Find $(x, y) \in D$ where $y < \gamma$ and $x$ is minimized
12: $F \leftarrow \{v_i\}_{i=1,2,\ldots,x}$
13: Rank nodes in $F$ based on their outdegree
    centrality as $v'_1, v'_2, \ldots, v'_x$
14: $low \leftarrow 1$, $high \leftarrow x$, $D' \leftarrow \{(x, y)\}$
15: while $high - low > \delta w$ do
16:    $mid \leftarrow \lfloor (low + high)/2 \rfloor$
17:    $H' \leftarrow H \backslash \{v_i\}_{i=1,\ldots,mid}$
18:    $s \leftarrow$ the size of the giant SCC in $H'$
19:    Add $(mid, s)$ to set $D'$
20:    If $s \geqslant \gamma$ then $low \leftarrow mid$ else $high \leftarrow mid$
21: end while
22: Find $(x', y') \in D'$ where $y' < \gamma$ and $x'$ is minimized
23: $F' \leftarrow \{v'_i\}_{i=1,2,\ldots,x'}$
24: Return the set of nodes in $F'$ as the core of $H$
```

Next, we consider the time complexity of Algorithm 1. We abuse notations n and m slightly by letting them denote the number of nodes and edges in $G^{(k)}$, respectively. In addition to the cost that may be needed to generate the core, the algorithm needs $O(m + n\alpha)$ time to check the existence of standalone and appendix botnets, and $O(m + n\alpha)$ time to obtain the list of nodes to be monitored under a given core Q. Note that when computing the reachability from a node, we do not need to explore all the nodes that are reachable from this node; instead, we count it as a potential node for monitoring once the number of reachable nodes from it goes beyond α. Hence, the computational cost of Algorithm 1 in obtaining the list of nodes to be monitored is $O(n\alpha + rm + m\log n + n\log n)$.

8. Detection

In this section, we apply the Peri-Watchdog to detect synthetically generated botnets in the Twitter network. We consider three different topologies that the botmaster can use to organize individual bots: *star*, *tree*, and *graph*.

- **Star:** the botnet has 10,000 bots, each of which follows a central master node.
- **Tree:** the botnet is organized as a tree, whose root node is the master node. The height of the tree is 5, and each non-leaf node has 10 children. Hence, in total there are 11,111 bot nodes including the master one. A child node follows its parent node, but not vice versa.
- **Graph:** the graph botnet topology has 10,000 bots, and the master node is randomly chosen from it. We number all these bots from 0 to 9,999. Bot i follows every other bot j, where $j = (k + i) \bmod 10{,}000$ with $k = \pm 1, \pm 2, \ldots, \pm 10$.

For each of these topologies, we *civilize* a bot (including the master node) as follows. For a civilized bot, we let it follow 1000 uniformly chosen normal Twitter user accounts. If it is *unconstrained civilization*, the bot is also followed by 10 uniformly chosen normal Twitter user accounts. If it is *constrained civilization*, the bot is followed by 10 normal Twitter accounts which are randomly chosen among the 1000 normal users accounts the bot follows, each with a probability proportional to its indegree. Constrained civilization is inspired by a recent measurement study which shows that link farming is more successful from a small number of popular and highly active Twitter users than lay Twitter users with few followers [11]. Intuitively speaking, when bots are civilized in a constrained manner, it limits the mixture of bot nodes and normal user accounts, as a bot can be followed by only those normal accounts that if follows.

In summary, we have the following scenarios:

Scenario	Topology	Bots civilized	Civilization
Star I	Star	A random non-master bot	Unconstrained
Star II	Star	Master bot node	Unconstrained
Star III	Star	Every bot node	Unconstrained
Star IV	Star	A random non-master bot	Constrained
Star V	Star	Master bot node	Constrained
Star VI	Star	Every bot node	Constrained
Tree I	Tree	A randomly chosen leaf bot	Unconstrained
Tree II	Tree	Master bot node	Unconstrained
Tree III	Tree	Every bot node	Unconstrained
Tree IV	Tree	A randomly chosen leaf bot	Constrained
Tree V	Tree	Master bot node	Constrained
Tree VI	Tree	Every bot node	Constrained
Graph I	Graph	Every bot node	Unconstrained
Graph II	Graph	Every bot node	Constrained

We apply Peri-Watchdog on these topologies. In Algorithms 1 and 2, we set the parameters as follows: $\gamma = 1000$, $\alpha = 1000$, and $\delta = 0.001$. β is not important here because we apply Peri-Watchdog only on the static Twitter graph injected with synthetic botnet structures. According to Algorithm 1, there are two cases for bot detection with Peri-Watchdog. In the first case (*pre-new-core detection*), Peri-Watchdog sticks to the original core, and in the second case (*post-new-core detection*) Peri-Watchdog recalculates the new core and then perform bot detection.

Table 3
Results of pre-new-core detection under different topologies.

	Number of peripheral nodes monitored under original core	Fraction of peripheral nodes monitored under original core (%)	Number of bots (outside the core) monitored under original core
Original	118,696	0.29	0
Star I	118,704	0.28	1
Star II	125,023	0.30	1
Star III	13,081,477	31.40	9764
Star IV	118,708	0.28	1
Star V	133,677	0.30	1
Star VI	9,149,977	21.96	4830
Tree I	123,267	0.30	14
Tree II	123,760	0.30	11
Tree III	20,823,822	50	11,111
Tree IV	118,966	0.29	11
Tree V	125,965	0.30	11
Tree VI	10,591,333	25.42	6400
Graph I	13,216,572	31.72	10,000
Graph II	13,199,345	31.68	10,000

8.1. Pre-new-core detection

The detection results are illustrated in Table 3. (1) We observe that under the original core, the numbers of nodes outside the core monitored by Peri-Watchdog are comparable when only one node is civilized (Star-I, II, IV, V; Tree-I, II, IV, V). But when the number of civilized bot nodes increases significantly (Star-III, VI; Tree-III, VI; Graph-I, II), they create substantial changes to the structure of the original Twitter graph, thus leading to a large number of nodes that have to be monitored by Peri-Watchdog. Hence, for these cases, computation of a new core is inevitable. (2) Under each of topologies Star-I, II, IV and V where only one master node is followed by all other individual bots inside the botnet, only this master node will be scrutinized by Peri-Watchdog when one bot node is civilized. Under the tree topology, other top-level bot nodes than the root master node can also have high reachability, so we observe more bot nodes are monitored by Peri-Watchdog when only one node is civilized. (3) Moreover, we observe that under Tree III, a substantially larger number of nodes have to be monitored than in the other two cases where a large number of nodes are civilized (Star III and Graph I). This is plausible because the tree topology has more bot nodes (11,111) than the other two topologies (10,001 for the star topology and 10,000 for the graph topology). (4) Whether civilization is performed in a constrained or unconstrained fashion does not seem to affect the cases when there is only one node civilized. However, when all bots are civilized, we find that if bots are civilized in a constrained fashion under the star and tree topologies, a fewer number of bots are monitored under the original core. This is because under constrained civilization, the set of normal user accounts is limited to only those that follow bot nodes, which limits the mixture of bot nodes and normal user accounts. This does not affect the cases under the graph topology because nodes inside the botnet themselves are already highly mixed. Hence, civilizing any bot node is likely to mix the botnet well into the original Twitter network.

8.2. Post-new-core detection

The results of post-new-core detection are shown in Table 4. For the moment, we assume that during the new core computation, none of nodes admitted to the new core are suspicious of bot activities. (1) One observation is that under each scenario, the new core is very similar to the core of the original Twitter graph. Particularly, when only one node is civilized, the core almost remains the same except only a few nodes. (2) We, however, also note that bots can get into the core. This is because when a botnet is added to the graph, nodes in the original graph have to pass some bots in the botnet to reach other parts of the botnet, thus boosting the betweenness centrality of these bots. For instance, when only one bot node is civilized, this bot node is likely to be added to the core (e.g., Star-I,II and Tree-I,II). When a large number of bot nodes are civilized, the bot nodes can serve as shortcut nodes that bridge nodes in the original graph within shorter distances, which also helps increase their betweenness centrality measures. Hence, we see that when a large number of bot nodes are civilized, the majority of them get admitted into the core. Under Graph I, we actually see all the bot nodes become part of the core if bots are civilized in an unconstrained manner. (3) When bots can be civilized only in a constrained manner, it is more difficult for bots to slip into the core, which is plausible because constrained civilization limits the mixture of normal user accounts and bot accounts.

The above analysis hinges on the assumption that no bot nodes are found suspicious during the new core computation. It suggests the importance of scrutinizing newly added core nodes to ensure their sanity. The process of vetting newly admitted core nodes can be treated as part of hunting for bot activities: Peri-Watchdog keeps monitoring these nodes for an extended period of time to make sure that they are not suspicious of bot activities themselves. As shown in Table 4, the majority of the nodes in the original core appear in the new core, implying that monitoring these newly admitted core nodes does not introduce substantial computational burden. Close examination of

Table 4
Results of post-new-core detection under different topologies. For Tree V, we had a hard disk failure and could not finish the experiment.

	Fraction of core nodes	Core overlapping	Number of bots in the core	Number of bots monitored outside core
Original	33.9%	–	–	–
Star I	33.8%	100.0%	1	1
Star II	33.8%	100.0%	1	0
Star III	34.5%	97.7%	9888	5
Star IV	33.8%	100.0%	0	1
Star V	33.8%	100.0%	1	0
Star VI	34.4%	98.0%	8600	4
Tree I	33.8%	100.0%	1	11
Tree II	33.8%	100.0%	1	10
Tree III	34.7%	97.1%	11,011	3
Tree IV	33.8%	100.0%	0	11
Tree V	X	X	X	X
Tree VI	32.6%	98.7%	9748	40
Graph I	33.1%	98.4%	10,000	0
Graph II	32.3%	98.8%	9969	1

the results in Table 4 also reveals that even if some bot nodes get admitted into the core, some bot nodes in the periphery (outside the core) will still be monitored except three cases (Star-II,V and Graph-I). Hence, even if some bot nodes slip into the new core, it is still possible to discover bot activities from those nodes monitored in the periphery.

9. Cross validation

The design of Peri-Watchdog proposed in Section 7 is based on our extensive analysis of the Twitter graph. Its performance under a practical setting, however, still rests on the answers to the following questions: (1) How often does Peri-Watchdog update its core? (2) How stable is the core of the graph over time? (3) Are observations made from analysis of the Twitter graph common to other online social networks? Unfortunately, we are not able to obtain the dynamic changes of the Twitter graph to address the first two questions. To gain insights into how Peri-Watchdog performs, we conduct cross validation using data collected from another online social network, BrightKite [5]. BrightKite is a location-based online social network that was launched in 2007. Due to its open nature, we were able to collect the full social graphs formed by its users from April 14, 2009 to October 4, 2009, almost every day. It is noted that the friendship in BrightKite follows a different model from that in Twitter [6]: when user A adds B as her friend in BrightKite, all updates by A are visible to B. In our directed graph representation, edge (A, B) means that B is on A's friend list. We note that there may exist some missing data for a few days in the dataset, but we use Peri-Watchdog on the raw dataset because in practice noise due to measurement mistakes may be inevitable in observed data.

Fig. 8(1) depicts the total number of BrightKite users, the number of nodes in the giant WCC, the number of nodes not in the giant WCC, and the number of users in the second largest WCC in the BrightKite network. Generally speaking, both the size of the entire BrightKite graph and that of its giant WCC grow over the measurement period, but we do observe a few drops in both metrics, which we conjecture occurred due to some measurement errors. Interestingly, the number of users not in the giant WCC remains relatively stable, which suggests that most newly added users joined the giant WCC. Also, nodes that do not belong to the giant WCC do not form large components, as evidenced by the constant size of one for the second largest WCC. These results confirm that large standalone botnets, if hidden in the BrightKite network, will stand out easily.

We further show in Fig. 8(2) that the size of the giant SCC, and the number of nodes that are reachable from the giant SCC in the BrightKite graph, respectively, over time. Still, generally speaking, both numbers increase over time, but there are a few drops for a few days. Suppose that the attacker wants to hide an appendix botnet in the BrightKite network. As he has the freedom to add people onto his friend list but normal users do not add bots onto their friend lists, there are edges from the botnet to the normal BrightKite graph but not vice versa. Hence, all nodes that are reachable from the giant SCC of the BrightKite cannot belong to an appendix botnet. After removing these nodes from the BrightKite graph on each collection day, we find the size of the giant WCC in the residual graph is always 1. Hence, if the attacker wants to hide an appendix botnet in the BrightKite network, it would be easy to monitor its activities through graph-theoretic analysis by Peri-Watchdog.

Peri-Watchdog uses Algorithms 1 and 2 to discover and monitor crossover botnets. To apply it on the BrightKite network, we need to change its operation slightly. Due to the difference in how friendships are initiated in BrightKite, we use the indegree centrality measure instead on Line 15 in Algorithm 2. The parameters in Peri-Watchdog are set as follows: $\alpha = 100$ (i.e., we monitor botnets of at least 100 nodes), $\beta = 2000$ (i.e., we monitor at most 2000 nodes), $\gamma = 50$ (i.e., the threshold for small connected components), and $\delta = 0.001$ (i.e., the resolution to decide when to stop searching top nodes for breaking the graph into small connected components). Due to the small size of the BrightKite network, we consider all possible source nodes when calculating betweenness centrality measures. In the experiments, we assume that no nodes are deemed as suspicious, as we are concerned with the dynamics introduced by Peri-Watchdog operations.

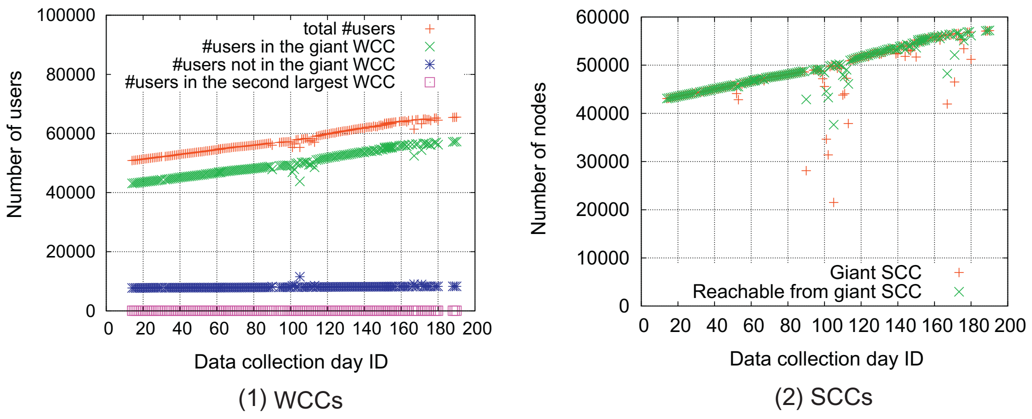


Fig. 8. WCCs and SCCs in BrightKite graph (April 1, 2009 is day 1).

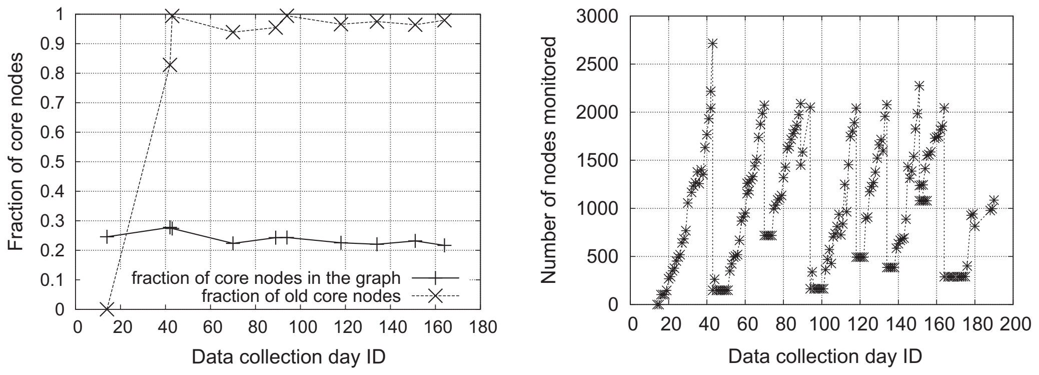


Fig. 9. Core stability and fraction of nodes to be monitored.

Fig. 9(1) presents when the core is computed and also the fraction of nodes that belong to the core. In total, the core has been computed 10 times and thus a new core is needed every 18 days. On average, each core contains 24.0% of the nodes in the entire BrightKite graph at the point of computation. Recall that the core of the Twitter graph contains 33.9% of the nodes in its giant WCC. As it is likely to miss some nodes that do not belong to the giant WCC for the Twitter graph due to the limitation of data collection, it seems that the fractions of core nodes are comparable for both networks. Fig. 9(1) also shows the fraction of previously observed nodes that are included in a new core. Clearly, the majority of the nodes in a new core have actually been seen in previous cores. The implication of this observation is: Peri-Watchdog does not need to verify the sanity of each node in a newly calculated core; rather, only a small fraction of nodes need to be checked.

We further show in Fig. 9(2) the number of nodes to be monitored by Peri-Watchdog for potential botnet activities. We note that this number increases as new nodes and friendships are added into the network. In most cases, once the number exceeds 2000 (i.e., β), a new core is generated, bringing the number of nodes for monitoring close to 0 again. One interesting data point occurs on May 11, 2009 (Day 42 in Fig. 9). We observe the following sequence of events: (1) the number of monitored nodes (i.e., 2040) exceeds 2000 on May 11; so a new core is calculated, which contains 2217 new nodes. (2) On May 12, 498 nodes in the periphery need to be monitored, which triggers another core computation; the new core has 148 new nodes, compared to the previous one. This suggests that if a new core contains too many nodes that did not appear in the previous one, the number of nodes that Peri-Watchdog has to monitor may exceed its capacity. Close examination reveals that the large number of new nodes admitted into the core calculated on May 11, 2009 results from a long period of time after the previous core computation, which occurred on April 14, 2009. April 14, 2009 was the first time a core was generated, when we assume that no new nodes were added to the core compared to the previous one. This allowed extra time for the number of monitored peripheral nodes to accumulate before it exceeded the threshold, thus postponing the next core computation. In practice, to prevent this from happening, we can force a new core computation – rather than wait for the number of monitored nodes to exceed the threshold – after a certain period of time has elapsed since the last core computation.

10. Discussion

In previous sections, we have presented the design of Peri-Watchdog and studied how it performs on synthetically generated botnet structures. We also have done cross validation with the BrightKite dataset. However, we do not pretend that Peri-Watchdog is a panacea for botnet detection in open online social networks. In this section, we play devil's advocate and discuss potential weaknesses of Peri-Watchdog.

Peri-Watchdog is not suitable for monitoring current botnets that rely on bot-controlled Twitter accounts to spread C&C information to individual bots that visit their public profile pages. Fortunately, such botnets are essentially traditional HTTP-based botnets, and many detection techniques have already been proposed to monitor repeated pulling activities by individual bots [14,20,1]. Also, the exposure of C&C information to the general public puts such botnets under scrutiny by independent cyber-security researchers.

Some online social networks such as Facebook allow a user to accept private messages from non-friend users if she opts to do so. As this kind of message circumvents the social graph formed by online users, botnets operating based on such communications can evade detection by Peri-Watchdog. However, as these messages stand out from those normal ones sent between socially connected users, we can simply monitor them for potential botnet activities, as we do on the set of suspicious nodes identified by Peri-Watchdog.

Peri-Watchdog requires a clean core without malicious bots to function effectively. For instance, the results in Table 4 show that when the botnet is organized as a star topology, the master node may slip into the core without being caught. The challenges for the attacker to evade the detection of Peri-Watchdog are threefold. First, after the botnet comes into existence, there is some time before a new core is generated. During this period of time, the master node could expose itself as it still stays in the periphery of the online social network, as seen from Table 3. To evade detection by Peri-Watchdog, the master node has to behave as a benign node in order to get a chance to be selected into the core in the next round of core computation. Second, even if the master node gets admitted into the new core, Peri-Watchdog will still monitor those nodes that are new comers to the core before the next round of core computation. Hence, the master node still has to behave normally to evade detection during this period. Third, Peri-Watchdog checks the sanity of core nodes in an offline fashion. Although this postpones the detection of the master node, the master bot node that stealthily slips into the core eventually will be caught by the offline component if it has performed suspicious bot activities.

Another way to compromise core nodes is by hacking some accounts in the core. Due to weak password protection, some celebrities' Twitter accounts have been compromised before. As these celebrities usually have a large number of followers, they tend to belong to the core more likely than normal users. In this case, if the abused account uses public messages for C&C communications, it can be spotted easily by its followers, and therefore an open online social network that allows its users to report such suspicious activities helps Peri-Watchdog remove compromised nodes from the core quickly. If the abused account uses private messages to deliver C&C information, Peri-Watchdog has to rely on offline inspection to discover it. From Section 9, we have observed that the core of an open online social network tends to be stable (i.e., the overlapping ratio of nodes in consecutive cores is high), and the fraction of core nodes in the entire graph does not vary significantly over time. This suggests that offline inspection is limited to a relatively stable set of nodes.

Peri-Watchdog is designed to aid detection of large botnets hidden in open online social networks. Thus, the botherder can divide his botnet into small isolated pieces to evade detection by Peri-Watchdog. By doing so, however, the botherder has to maintain a number of small-sized botnets, which complicates botnet operation and management.

11. Conclusions

The increasing popularity of online social networks has made them attractive to miscreants to hide botnet activities. In this work, we perform graph-theoretic analysis of the Twitter graph to look for patterns useful for identifying suspicious bot activities. Accordingly, we propose Peri-Watchdog to narrow down the list of nodes that are likely to pass botnet C&C information in an open online social network. We ran Peri-Watchdog on a dataset containing the evolution of social graphs formed by BrightKite users and make some interesting observations such as the stability of the core in the online social network. In our future work, we plan to explore other methods (e.g., more efficient streaming algorithms) to calculate the core of an evolving online social network.

References

[1] D. Ashley, An algorithm for http bot detection. <http://security.utexas.edu/consensus/HTTP%20Bot%20Detection.pdf>.
[2] P. Barford, V. Yegneswaran, An inside look at botnets, in: M. Christodorescu, S. Jha, D. Maughan, D. Song, C. Wang (Eds.), Malware Detection, Advanced in Information Security, vol. 27, Springer, 2006.
[3] L. Bilge, T. Strufe, D. Balzarotti, E. Kirda, All your contacts are belong to us: automated identity theft attacks on social networks, in: Proceedings of ACM WWW'09, 2009.
[4] U. Brandes, A faster algorithm for betweenness centrality, Journal of Mathematical Sociology 25 (2) (2001) 163–177.
[5] http://brightkite.com.
[6] http://techcrunch.com/2010/02/26/brightkite-2-million-users/.
[7] Z. Cai, C. Jermaine, The latent community model for detecting Sybils in social networks, in: Proceedings of the 19th Annual Network & Distributed System Security Symposium, February 2012.
[8] E. Cooke, F. Jahanian, D. McPherson, The zombie roundup: understanding, detecting, and disrupting botnets, in: Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop, USENIX Association, 2005, pp. 6.
[9] G. Danezis, P. Mittal, Sybilinfer: detecting sybil nodes using social networks, in: Proceedings of NDSS'09, 2009.
[10] http://www.facebook.com/press/info.php?statistics.
[11] S. Ghosh, B. Viswanath, F. Kooti, N.K. Sharma, G. Korlam, F. Benevenuto, N. Ganguly, K.P. Gummadi, Understanding and combating link farming in the twitter social network, in: Proceedings of the 21st International Conference on World Wide Web, WWW'12, 2012, pp. 61–70.
[12] J. Goebel, T. Holz, Rishi: identify bot contaminated hosts by irc nickname evaluation, in: Proceedings of UNENIX HotBots'07, 2007.
[13] C. Grier, K. Thomas, V. Paxson, M. Zhang, @spam: the underground on 140 characters or less, in: Proceedings of the 17th ACM Conference on Computer and Communications Security, 2010.
[14] G. Gu, R. Perdisci, J. Zhang, W. Lee, Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection, in: Proceedings of the 17th Conference on Security Symposium, 2008.
[15] G. Gu, P. Porras, V. Yegneswaran, M. Fong, W. Lee, Bothunter: detecting malware infection through ids-driven dialog correlation, in: Proceedings of 16th USENIX Security Symposium, 2007.
[16] G. Gu, J. Zhang, W. Lee, BotSniffer: detecting botnet command and control channels in network traffic, in: Proceedings of NDSS'08, The Internet Society, February 2008.
[17] T. Holz, M. Steiner, F. Dahl, E. Biersack, F. Freiling, Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm, in: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats, USENIX Association, 2008, pp. 9:1–9:9.
[18] A. Karasaridis, B. Rexroad, D. Hoeflin, Wide-scale botnet detection and characterization, in: Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets, USENIX Association, pp. 7–7, 2007.
[19] H. Kwak, C. Lee, H. Park, S. Moon, What is twitter, a social network or a news media? in: Proceedings of ACM WWW'10, 2010.
[20] J.-S. Lee, H. Jeong, J.-H. Park, M. Kim, B.-N. Noh, The activity analysis of malicious http-based botnets using degree of periodic repeatability, in: Proceedings of the 2008 International Conference on Security Technology, 2008.
[21] L. Liu, S. Chen, G. Yan, Z. Zhang, Bottracer: execution-based bot-like malware detection, in: Proceedings of the 11th International Conference on Information Security, Springer-Verlag, Taipei, Taiwan, 2008, pp. 97–113.
[22] A. Mohaisen, N. Hopper, Y. Kim, Keep your friends close: incorporating trust into social network-based sybil defenses, in: Proceedings of IEEE Infocom'11, 2011.
[23] S. Nagaraja, A. Houmansadr, P. Piyawongwisal, V. Singh, P. Agarwal, N. Borisov, Stegobot: a covert social network botnet, in: Proceedings of the 13th International Conference on Information Hiding, IH'11, 2011.
[24] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, N. Borisov, Botgrep: finding p2p bots with structured graph analysis, in: Proceedings of USENIX Security'10, Washington, DC, 2010.
[25] N.P. Nguyen, G. Yan, M.T. Thai, S. Eidenbenz, Containment of misinformation spread in online social networks, in: Proceedings of the 4th ACM Web Science (WebSci'12), 2012.
[26] A. Ramachandran, N. Feamster, D. Dagon, Revealing botnet membership using dnsbl counter-intelligence, in: Proceedings of the 2nd Conference on Steps to Reducing Unwanted Traffic on the Internet, San Jose, CA, USENIX, 2006.

[27] B. Stock, J. Göbel, M. Engelberth, F.C. Freiling, T. Holz, Walowdac – analysis of a peer-to-peer botnet, in: Computer Network Defense (EC2ND), 2009 European Conference, November 2009, pp. 13–20.
[28] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, G. Vigna, Your botnet is my botnet: analysis of a botnet takeover, in: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS '09, ACM, 2009, pp. 635–647.
[29] K. Thomas, D.M. Nicol, The Koobface botnet and the rise of social malware, in: Proceedings of IEEE MALWARE'10.
[30] http://www.itpro.co.uk/623323/twitter-botnet-creation-made-simple.
[31] http://www.wired.com/threatlevel/2009/08/botnet-tweets/.
[32] http://www.tgdaily.com/software-brief/52284-twitter-on-pace-to-reach200-million-users-by-2011.
[33] B. Viswanath, A. Post, K.P. Gummadi, A. Mislove, An analysis of social network-based sybil defenses, in: Proceedings of the ACM SIGCOMM'10, 2010.
[34] W. Wei, F. Xu, C.C. Tan, Qun Li, Sybildefender: defend against sybil attacks in large social networks, in: Proceedings of IEEE Infocom'12, 2012.
[35] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, I. Osipkov, Spamming botnets: signatures and characteristics, in: Proceedings of SIGCOMM'08, 2008.
[36] G. Yan, G. Chen, S. Eidenbenz, N. Li, Malware propagation in online social networks: nature, dynamics, and defense implications, in: Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ASIACCS '11, 2011, pp. 196–206.
[37] C. Yang, R. Harkreader, J. Zhang, S. Shin, G. Gu, Analyzing spammers' social networks for fun and profit: a case study of cyber criminal ecosystem on twitter, in: Proceedings of the 21st International Conference on World Wide Web, WWW '12, 2012, pp. 71–80.
[38] Z. Yang, C. Wilson, X. Wang, T. Gao, B.Y. Zhao, Y. Dai, Uncovering social network sybils in the wild, in: Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference, IMC '11, 2011, pp. 259–268.
[39] T.-F. Yen, M.K. Reiter, Traffic aggregation for malware detection, in: Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, 2008.
[40] H. Yu, Sybil defenses via social networks: a tutorial and survey, SIGACT News 42 (3) (2011) 80–101.
[41] H. Yu, P.B. Gibbons, M. Kaminsky, F. Xiao, Sybillimit: a near-optimal social network defense against sybil attacks, in: Proceedings of the 2008 IEEE Symposium on Security and Privacy, 2008.
[42] H. Yu, M. Kaminsky, P.B. Gibbons, A. Flaxman, Sybilguard: defending against sybil attacks via social networks, SIGCOMM Computing Communication Review 36 (4) (2006) 267–278.
[43] Y. Zeng, G. Yan, S. Eidenbenz, K.G. Shin, Measuring the effectiveness of infrastructure-level detection of large-scale botnets, in: Proceedings of the Nineteenth International Workshop on Quality of Service, IWQoS '11, IEEE Press, 2011, pp. 1:1–1:9.
[44] Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, E. Gillum, Botgraph: large scale spamming botnet detection, in: Proceedings of USENIX NSDI'09, 2009.

Guanhua Yan obtained his Ph.D. degree in Computer Science from Dartmouth College, USA, in 2005. From 2003 to 2005, he was a visiting graduate student at the Coordinated Science Laboratory in the University of Illinois at Urbana-Champaign. He is now working as a Research Scientist in the Information Sciences Group (CCS-3) at the Los Alamos National Laboratory. His research interests are cyber-security, networking, and large-scale modeling and simulation techniques. He has contributed about 40 articles in these fields.