Journal of Network and Computer Applications 71 (2016) 1–10

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

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

iPersea: Towards improving the Sybil-resilience of social DHT

Mahdi Nasrullah Al-Ameen *, Matthew Wright

Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington, TX, USA

ARTICLE INFO

Article history:
Received 1 May 2015
Received in revised form
13 April 2016
Accepted 18 May 2016
Available online 20 May 2016

Keywords:
Sybil attack
Security
Social DHT

ABSTRACT

P2P systems are highly susceptible to Sybil attacks, in which an attacker creates a large number of identities and uses them to control a substantial fraction of the system. Persea is the most recent approach towards designing a social network based Sybil-resistant DHT. Unlike prior Sybil-resistant P2P systems based on social networks, Persea does not rely on two key assumptions: (i) that the social network is fast mixing, and (ii) that there is a small ratio of *attack edges* to honest peers. Both assumptions have been shown to be unreliable in real social networks. The hierarchical distribution of node IDs in Persea confines a large attacker botnet to a considerably smaller region of the ID space than in a normal P2P system and its replication mechanism lets a peer to retrieve the desired results even if a given region is occupied by attackers. However, Persea system suffers from certain limitations, since it cannot handle the scenario, where the malicious target returns an incorrect result instead of just ignoring the lookup request. In this paper, we address this major limitation of Persea through a Sybil detection mechanism built on top of Persea system, which accommodates inspection lookup, a specially designed lookup scheme to detect the Sybil nodes based on their responses to the lookup query. We design a scheme to filter those detected Sybils to ensure the participation of honest nodes on the lookup path during regular DHT lookup. Since the malicious nodes are opt-out from the lookup path in our system, they cannot return any incorrect result during regular lookup. We evaluate our system in simulations with social network datasets and the results show that catster, the largest network in our simulation with 149,700 nodes and 5,449,275 edges, gains 100% lookup success rate, even when the number of attack edges is equal to the number of benign peers in the network.

1. Introduction

Peer-to-peer (P2P) systems are inherently vulnerable to Sybil attacks, in which an attacker creates a large number of pseudonymous entities and use them to gain a disproportionately large influence over the system (Cholez et al., 2009; Douceur, 2002). The attackers then collude to launch further attacks, such as taking over resources and disrupting connectivity to subvert the system's operation. Such attacks have been shown to be quite problematic in *structured* P2P systems in which nodes are placed into a distributed hash table (DHT) like Chord (Stoica et al., 2001), CAN (Ratnasamy et al., 2001), Pastry (Rowstron and Druschel, 2001), and Kademlia (Maymounkov and Mazieres, 2002). Kademlia was the basis for both the Kad network and Vuze, DHTs used in the popular BitTorrent file-sharing P2P system with millions of users each. Researchers have documented this vulnerability in real-world systems, including the Maze P2P file-sharing system (Lian et al., 2007; Yang et al., 2005) and the Vanish data storage system (Wolchoky et al., 2010).

Recent research has focused on leveraging information from social networks to make the system robust against Sybil attackers, resulting in a number of decentralized approaches (Lesniewski-Laas, 2008; Lesniewski-Laas and Kaashoek, 2010; Yu et al., 2008, 2006; Tran et al., 2011; Wei et al., 2012; Mittal et al., 2012). The key to these approaches is the idea that honest and malicious nodes can be effectively partitioned into two subgraphs in the social network. The link between an honest node and a malicious peer is called an attack edge, which represents an act of social engineering to convince the honest node to add the link.

These mechanisms are based on two key assumptions: (i) that the online social networks are *fast-mixing*, meaning that a random walk in the honest part of the network approaches the uniform distribution in a small number of steps, and (ii) that the number of attack edges are rather limited in online social networks. Recent studies (Mohaisen et al., 2010; Viswanath et al., 2010; Yang et al., 2011; Bilge et al., 2009), however, show that the above assumptions do not hold in real-world social networks. So, it remain an open research problem, until Persea (Al-Ameen and Wright, 2014) was proposed, to design a social

* Corresponding author.
E-mail addresses: mahdi.al-ameen@mavs.uta.edu (M.N. Al-Ameen), mwright@uta.edu (M. Wright).

http://dx.doi.org/10.1016/j.jnca.2016.05.014
1084-8045/

network based Sybil defense mechanism, which does not rely on these assumptions.

1.1. Motivation

Persea (Al-Ameen and Wright, 2014) derives its Sybil resistance by assigning IDs through a bootstrap tree, the graph of how nodes have joined the system through invitations. Persea argues that building a bootstrap tree is more realistic than assuming that the clients have access to lists of social network connections from a system like Facebook. Also, IDs are certified in Persea, making attacks based on ID forging impossible outside of attacker-controlled ID ranges. In Persea, the (key,value) pair is replicated in evenly spaced nodes, so that even if a given region is occupied by the attacker, the desired (key,value) pair can be retrieved from other regions. We give a overview of Persea system in Section 2.

The Persea approach offers a number of important advantages over previous schemes:

- Unlike prior Sybil-resistant P2P systems based on social networks, Persea does not rely on two key assumptions: (i) that the social network is fast mixing, and (ii) that there is a small ratio of attack edges to honest nodes.
- The hierarchical distribution of node IDs limits the attackers to isolated regions in ID space.

However, Persea system suffers from certain limitations that make its use questionable in real world scenario. In Persea DHT, a (key,value) pair is replicated in a number of nodes and the lookup operation is performed for each target node to get the value, associated with the search-key. Persea assumes that when the target node is malicious it does not return incorrect result, rather ignores the lookup request. So, in Persea DHT, if the initiator of a lookup retrieves the correct result from at least one benign target, the lookup is termed to be successful. But in real-world scenario, the malicious target may reply with incorrect result to make it harder for the initiator to retrieve the correct one from the set of different returned results. Persea system cannot handle such obvious attacks.

The simplest solution to this limitation of Persea could be implementing *majority voting* scheme, where the initiator picks the result with higher count. If the counts of two types of results are same, the initiator randomly picks one as the final result. In our simulation, we implement the above strategy of adversaries and evaluate Persea with majority voting scheme. However, our results show that the lookup success rate in Persea sharply decreases with the increase in attack edges (See Section 6), which infers that majority voting is not effective enough to make the system robust, when the malicious target returns incorrect result. So, the efficacy of Persea is left as an open question in real-world scenario.

1.2. Contributions

In this paper, we address this major limitation of Persea and develop a Sybil detection scheme on top of Persea system. We propose *inspection lookup* to detect the Sybils, which is a specially designed lookup mechanism to determine the *status* (honest or malicious) of a node. This lookup seems as a regular DHT lookup to a peer, whose status is being inspected, so that an attacker cannot play a fabricated role during inspection lookup to prove it as an honest node. We introduce the idea of *collaborative friends*, the groups of benign peers, who agree to execute the inspection lookups for detecting the Sybils. We provide a detailed description of our Sybil detection mechanism in Section 3.

We develop a mechanism to filter the detected Sybil nodes from the lookup path for ensuring the participation of benign peers during regular DHT lookup. We then incorporate our filtering scheme with the lookup mechanism in Persea. So, in our system the attackers are opt-out of the lookup path and thus, they can neither intercept a lookup (as an intermediate node) nor return an incorrect result (as the target node).

While we incorporate our Sybil detection mechanism with Persea, we name the new system *iPersea* (*i*mproved *Persea*), which inherits the advantages of Persea that it gains over prior social network based Sybil-resistant systems. So, iPersea does not depend on the fast-mixing social network and small ratio of attack edges to honest peers. We justify our claims through evaluations in Section 6. We simulate for networks with different *clustering coefficients* to show that our system does not depend on the fast-mixing nature of a network. The clustering coefficient is a measure of degree to which nodes in a network tend to cluster together (Mislove et al., 2007; Grabowski and Kosinski, 2008), and it is therefore directly related to the fast-mixing property of a network (Grabowski and Kosinski, 2008). To validate the claim that iPersea does not depend on the small ratio of attack edges to honest peers, we evaluate for this ratio, up to 1.5 and find consistent results, where the lookup success rate for any network in our evaluations is no less than 92.9%.

Our experimental results show that 70% of lookups succeed in Persea, while the ratio of attack edges to honest nodes is 0.5 in facebook social network. However, in iPersea, 93.6% lookups succeed in facebook, even when the ratio of attack edges to honest nodes is increased to one. In flickr and catster (largest network in our simulation with 149,700 nodes), 100% lookups get successful in our system when we have equal number of attack edges and honest nodes.

The remainder of the paper is organized as follows: we give a overview of Persea system in Section 2 and explain the design our Sybil detection mechanism in Section 3. We then describe the attack model in Section 4 before presenting our simulation results in Section 6. In Section 7 we discuss the related works in Sybil defense. We give a direction to our future work Section 8 and then conclude in Section 9.

2. Overview of Persea

In this section, we briefly describe the design of Persea (Al-Ameen and Wright, 2014). We begin with the overview of the system and then address the ID space allocation, ID certification and key replication. We also describe the routing table organization and lookup mechanism of Persea.

2.1. Design overview

Persea consists of two layers: a social network layer (the bootstrap graph) and the DHT layer. In Persea, an *edge* refers to a link between two nodes in the DHT layer. The bootstrap network and DHT are simultaneously built starting with a set of bootstrap nodes. The bootstrap nodes are the initiators of the system. They are connected to each other in both the bootstrap network and the DHT. Node IDs in the DHT are assigned to the bootstrap nodes such that they are evenly spaced over the circular ID space. Thus, the ID space of the DHT is divided into one region for each bootstrap node.

A new peer must join the Persea system through an invitation from an existing node in the network. In general, it is expected that a node that is invited is socially known to the inviting peer. When a node is invited, it not only becomes a part of the bootstrap network but also gets a node ID in DHT layer. The new node gets a chunk of node IDs that it can use to invite more nodes for joining the network. ID assignments and chunk allocations are put into

certificates signed by the parent nodes, and certificates are stored in the DHT itself to allow for reliable distributed checking of the chain of certificates all the way up to the bootstrap nodes.

The DHT layer of Persea is based on Kademlia (Maymounkov and Mazieres, 2002), a DHT that is widely adopted for the BitTorrent file-sharing P2P system. The main difference in Persea is that IDs are replicated evenly around the ID space for greater resiliency given the ID distribution scheme.

2.2. Hierarchical ID space

We now briefly describe how node IDs are distributed in Persea. Each bootstrap node has a contiguous range of node IDs called a *chunk*, which includes the bootstrap node's ID. A bootstrap node divides its chunk of node IDs into sub-chunks based on the *chunk-factor*, a system parameter.

When a bootstrap node invites a peer to join the system, it assigns a node ID to the joining node from one of its sub-chunks and also assigns the new node control over the rest of the sub-chunk for further distribution. The newly joined node becomes the authority for distributing node IDs from the given sub-chunk. Thus, once the joining node becomes a part of the system, it can invite more nodes to join the system. Based on the invitation-relationship among peers, a *bootstrap tree* is formed in which an inviter node is the parent of its invited peers. If the number of bootstrap node is more than one, then it would have a forest of trees, where each bootstrap node is the root of each tree. The chunk-factor and size of the ID space define the maximum possible height and width of a tree. This mechanism has the advantage that even if a bot compromises a node and leverages it to add a large number of malicious nodes to the system, they will be still confined in a particular region of ID space.

2.3. Routing table organization

In the DHT layer, each node maintains a routing table of b node lists for a b-bit ID space. Each list has up to k entries and is called a *k-bucket*. Each k-bucket entry contains the IP address, port, node ID, and public key of another node. The list is organized so that the ID of a node in the bth list of a node with ID i should share the first $b-1$ bits of i and have a different bth bit from i.

2.4. Replication

In Persea, a (key,value) pair is stored in evenly spaced nodes, so that even if a given region is occupied by the attackers, the desired (key,value) pair can be retrieved from other regions. In Persea, the ID space is virtually divided into R regions and the (key,value) pair is replicated in R evenly spaced nodes, one in each virtual region. The evaluations in Persea show that $R=7$ gives the optimal results for the networks, considered in their simulation (Al-Ameen and Wright, 2014).

2.5. Lookup mechanism

Node lookup in Persea is initiated by the **lookup**(*key*) request where a node queries the α nodes in its k-buckets that are the closest ones to the desired key. Each of the α nodes sends the initiator β node IDs from its k-bucket closest to the target node. From the set of returned node IDs, the initiator selects α nodes for the next iteration. This process is iterated until the target is found or no nodes are returned that are closer than the previous best results.

The initiator of a lookup performs R such independent parallel lookup operations and when an owner is found from any of R independent lookups, the initiator sends the owner a message for either the store (**put**(*key*, *value*)) or retrieval (**get**(*key*)) operation. We incorporate our Sybil-filtering scheme (see Section 3.3) with this lookup mechanism to opt-out the attackers from the lookup path.

3. Sybil detection mechanism

Our Sybil detection mechanism detects an attacker by exploiting its malicious behavior during a lookup. We propose inspection lookup to detect the Sybils, which determines the status (honest or malicious) of a node based on its response to the lookup request. Inspection lookup accommodates certain strategies to appear as a regular lookup to a peer, whose status is being inspected, so that an attacker cannot fabricate its behavior during inspection lookup to prove it as a benign node. The status (honest or malicious) of a node, determined through inspection lookup, is used to filter Sybil nodes during regular lookup. We incorporate our filtering scheme with the lookup mechanism of Persea to ensure higher lookup success rate by ensuring the participation of honest peers on the lookup path.

Our Sybil detection mechanism is based on the following assumptions:

- The adversary intercepts as many lookup queries as possible. As a target node, an attacker does not return the correct value, associated with the search-key.
- An honest node adheres to the protocol and acts legitimately.

In our mechanism, the status of a node represents whether it is honest or malicious, represented by '+' (honest) or '−' (malicious). Each parent node determines the status of its direct children with the help of selected peers, called collaborative friends.

3.1. Selecting collaborative friends

A peer (say it node P) requests its parents, grandparents and other ancestor nodes to suggest trusted peers, who agree to be the collaborative friends of node P for detecting the Sybils. An ancestor can suggest any number of collaborative friends, depending upon number of peers it trust and their willingness to collaborate. The more collaborative friends, node P has from different layers of hierarchical ID space, the harder it is for the child (may be, an attacker) of node P to distinguish an inspection lookup from the regular one. Since, node P selects the initiator of an inspection lookup from the set of its collaborative friends, the randomness in the placement of collaborative friends in ID space contributes in rising the hurdles to distinguish between inspection and regular lookup. So, node P requests its ancestors at each upper layer to be its collaborative friends and suggest more trusted nodes.

In ideal case, an ancestor node always returns trusted collaborative friends. However, in real-world scenario, an ancestor, which is not responsible enough in detecting the Sybils, may return randomly-selected collaborative friends. In our experiments, we consider both scenarios and the results for Sybil-detection and lookup success rate show very subtle differences between two approaches. So, random selection of collaborative friends can be effective in detecting the Sybils and consequently gaining high lookup success rate.

3.2. Inspection lookup

The goal of inspection lookup is to detect the Sybils based on their responses to the lookup messages. To design an inspection lookup, we adapt the basic lookup mechanism of Persea (see Section 2) and incorporate following strategies so that an

inspection lookup appears to be a regular lookup to the peer and consequently an attacker cannot distinguish it from a regular lookup.

- The source and target of an inspection lookup and also the node, whose status is to be inspected, are randomly selected.
- Inspection lookups are performed at uniformly distributed random interval.
- During the lookup operation in DHT, the role of a peer on the lookup path may be an intermediate hop or the target. In each inspection lookup, it is randomly selected which role (intermediate hop or target) of a peer would be inspected.

While inspecting the role of a child as an intermediate hop, node P selects a peer (say it node F) from its list of collaborative friends to be the initiator of the inspection lookup. It selects one of its direct children (say it node T) as the target node. These selections are made randomly. From the set of direct children, whose status are not inspected yet, node P randomly selects a child (say it node C) as an intermediate node of the lookup. Based on the success of inspection lookup, the status of node C is determined.

According to the suggestion of node P, node F sends the lookup request to node C. Since the inspection lookup appears as a regular lookup to Node C, it follows the mechanism of regular lookup and returns β nodes from its k-buckets that are closest ones to the search-key. If the set of returned nodes does not include node T, node F then sends lookup request to each of these β nodes in the next iteration. This process is iterated until the target is found or no nodes are returned that are closer than the previous best results.

In the ID space of Persea, where the node IDs are hierarchically distributed, there is an obvious lookup path from node C to node T through their parent node P. So, if the inspection lookup does not reach node T, node C is considered responsible for this failure and gets '−' status. Node C gets '+' status when the lookup succeeds, since an honest node directs a lookup towards the target.

When the role of a peer as the target node is inspected, a randomly selected collaborative friend of node P (say it node F_1) stores a (key,value) pair in node C and makes sure that the key matches with the node ID of node C. After a random interval, the lookup request is sent to node C from a collaborative friend of node P (say it node F_2) to return the value associated with the search-key. Node F_2 is informed by node P about the desired value that should be returned by node C. If it is not returned, node C is marked with '−' status. If node C returns the correct value, it gets '+' status, since an honest peer always returns the appropriate value associated with the search-key.

False positive and false negative: During inspection lookup, an honest intermediate node, whose status is being inspected, may unintentionally return malicious peers that are closest to the target, increasing the probability of a lookup to fail. If the lookup fails, the honest node is marked with '−' status that increases the rate of false positive.

The rate of false positive is zero when the role of a peer as the target node is inspected, as an honest node always returns the correct value associated with the search-key. The rate of false negative is zero in both cases, since our mechanism correctly identifies an attacker exploiting its malicious response to the inspection lookup message.

The above discussions on false-positive and false-negative are applicable to the scenarios, which abide by the assumption of selecting trusted nodes only, as the collaborative friends. However, for random selection of collaborative friends an attacker may get selected. In this case, if the attacker initiates an inspection lookup as a collaborative friend, a peer, whose status is inspected, gets '−' status if it is honest and is marked with '+' status if it is malicious, irrespective of the outcome of inspection lookup. Thus, for random selection of collaborative friends, the rate of both false-positive and false-negative may get increased.

3.3. Sybil-filtering mechanism

In this section, we describe the mechanism to opt-out the Sybils from the lookup path for ensuring the participation of honest nodes during regular lookup. In our evaluations, we incorporate our filtering mechanism with the basic lookup scheme of Persea.

During regular lookup, before selecting a peer (say it node Q) as an intermediate hop, the initiator of the lookup (say it node L) asks the parent of node Q to send its status. We assume, a malicious peer invites other attackers to join the network and promotes its children by giving '+' status. According to these assumptions, a malicious node gets '−' status only from a benign parent. So, if the status of node Q is found '+', node L gets the status of node Q's parent. This process is repeated for the other ancestors of node Q until the bootstrap node is reached or the '−' status is found for an ancestor.

If the bootstrap layer is reached, it suggests that none of node Q's ancestors is marked with '−' status. So, node Q is considered as a benign peer and gets selected for the lookup. If '−' status is found for node Q or any of its ancestor nodes, node Q is termed as a malicious peer and does not get selected as an intermediate node for the lookup. Once node L gets the status of node Q, it stores that status to be used in future lookup.

Node L follows the same procedure, as described above, to get the status of a target node, for deciding whether to accept the value, returned by that node.

4. Attack model

We inherit most of the features of the attack model in Persea (Al-Ameen and Wright, 2014), where attackers use social engineering to create attack edges in the social network. When a malicious peer joins the network, it gets a chunk of node IDs for further distribution and we assume, an attacker invites only malicious peers for joining the network to infiltrate the system with as many attackers as possible. Also, the attacker promotes its children by assigning '+' (honest) status without performing any inspection lookup for them.

As in Persea, we assume that the attackers know the IDs of all other attackers and store only the information of malicious nodes in their k-buckets. The goal of the adversary is to intercept as many lookup queries as possible. During lookup, If an attacker gets the **lookup** message, it returns the node IDs of malicious peers from its k-bucket. When the attacker receives a **get** message, instead of just ignoring the message (as in Persea (Al-Ameen and Wright, 2014)), it returns an incorrect value in our attack model.

In our Sybil detection mechanism, for random selection of collaborative friends, an attacker (say it node A) may be selected as the collaborative friend. We assume, as the collaborative friend of a node (say it node P), when node A initiates an inspection lookup to inspect the status of a child of node P (say it node C), whatever be the results of lookup; if node C is a malicious peer, node A informs node P that the lookup succeeds, however, if node C is an honest peer, the response of node A to node P says, the lookup fails.

5. Analysis

In this section, we develop analytical models to estimate average hop-count per regular lookup and the rate of false positive and false negative in our Sybil detection mechanism.

5.1. Lookup path length

We develop an analytical model to estimate the average hop-count per lookup. Let e_p represent the average edges per node in a network and a_h be the number of attack edges per honest node. So, $a_h = g/n$, where g represents the number of attack edges and n is number of honest nodes. When a lookup operation starts in our system, the initiator of the lookup selects α nodes from its routing table. So, the number of malicious peers in the set of α nodes is represented by $\alpha \times \frac{a_h}{e_p}$. In the next iteration, each of α nodes returns β closest nodes (to the target) and a malicious peer always returns adversary nodes. Let m_i represent the number of malicious nodes that the initiator gets in the set of $\alpha \times \beta$ nodes in ith$(i \geq 1)$ iterations. So, in the first iteration, the number of malicious peers that the initiator gets in the set of $\alpha \times \beta$ nodes is: $p_{m_1} = \left(\alpha \times \frac{a_h}{e_p} \times \beta\right) + \left((1 - \alpha \times \frac{a_h}{e_p}) \times \beta \times \frac{a_h}{e_p}\right)$.

The attackers try to subvert the lookup query and thus the number of attackers in each iteration contributes to estimate the probability of a lookup to succeed. The number of malicious nodes, selected by the initiator in the set of α nodes for $(i+1)$th iteration is: $\alpha \times \frac{m_i}{\alpha \times \beta} = \frac{m_i}{\beta}$. So, the probability (p_{m_i}) of choosing a malicious nodes from the set of α nodes at iteration i is represented by $\frac{m_i}{\alpha \times \beta}$. Thus, at any iteration $j(j \geq 1)$, we get p_{m_j} by evaluating $\prod_{i=1}^{j} p_{m_i}$. The lookup continues until the target node is found and we estimate the probability of lookup failure at iteration j by evaluating p_{m_j}. In our analysis, to estimate the path length of a lookup, we take a constant l_c representing a very low probability of lookup failure and then we calculate the minimum value of i as the lookup path length, where $p_{m_i} \leq l_c$.

We consider l_c=0.001 to get the lookup path length from our analytical model. To compare our analytical results with the results from our evaluations, we consider a_h=1.0, $\alpha = 5$ and $\beta = 7$ (as used in our simulations). The value of e_p depends upon individual network. Fig. 1(a) shows the differences between our analysis and simulation for average hop-count per regular lookup. We find that our analytical estimations are very close to the experimental results.

5.2. Rate of false positive

To estimate the rate of false positive, we use the above analytical model for measuring p_{m_j}. We consider both trusted and randomly selected collaborative friends in our analysis. For trusted collaborative friends, the rate of false positive is zero when the role of a peer as the target is inspected. However, when the role of a peer as the intermediate node is inspected, we may get a rate of false positive, since an attacker may intercept the lookup (see Section 3 for explanation). So, the number of attackers, selected in an iteration is directly related to the rate of false positive. Thus, we use p_{m_j} to estimate the false positive rate and put j=1 in this case. In our analysis, we assume that j=1 is a reasonable estimation for hop-count per inspection lookup and also, we get the same value for j, when we take the *floor* of the results for average hop-count per inspection lookup in our evaluations (see Table 5).

While comparing our analytical estimations for false positive rate with the experimental results, we consider the ratio of attack edges to honest nodes is one. Fig. 1(b) illustrates the results for this comparison. We find that the difference between analysis and simulation is 0.01 in the network datasets of hamsterster (**ham**), flickr (**flic**), wiki-Vote (**wiki**) and ca-AstroPh (**astro**). In the social network dataset of facebook (**fb**), we get exactly same results from our analytical model and simulations.

Now, we estimate the false positive rate in a scenario, where the collaborative friends are randomly selected. In this case, the false positive rate may get increased not only by the attackers on the lookup path, but also by malicious collaborative friends. An attacker, as a collaborative friend, contributes to increase the false positive rate when it is selected to initiate an inspection lookup and the peer, whose status is being inspected in that lookup, is an honest one. So, to estimate the false positive rate, we measure the probability of selecting a malicious collaborative friend and an honest node (to inspect its status) in the same inspection lookup and then take the union of this probability with $p_{m_j}(j = 1)$.

Let a node N be at level $l_h + 1$ of the hierarchical ID space and c_{nl} be the number of collaborative friends from each of its upper level. For simplicity of analysis, we consider c_{nl} is same for each level. In our analysis, e_p represents the average edges per node in a network and a_h is the number of attack edges per honest node. So, when a node is randomly selected as a collaborative friend, $\frac{a_h}{e_p}$ represents its probability to be malicious. Thus, the expected number of malicious collaborative friends of node N is: $\frac{a_h}{e_p} \times l_h \times c_{nl}$. When node N randomly selects a peer from its set of collaborative friends to initiate an inspection lookup, $\frac{\frac{a_h}{e_p} \times l_h \times c_{nl}}{l_h \times c_{nl}} = \frac{a_h}{e_p}$ represents the probability of this collaborative friend to be an attacker. To inspect the status, the probability of selecting an honest child of node N is: $\frac{e_p - a_h}{e_p}$. Now, we measure the probability of selecting a malicious collaborative friend and an honest node (to inspect its status) in the same inspection lookup as: $\frac{a_h}{e_p} \times \frac{e_p - a_h}{e_p}$. So, for random selection of collaborative friends, we calculate $(\frac{a_h}{e_p} \times \frac{e_p - a_h}{e_p}) + p_{m_j} - (\frac{a_h}{e_p} \times \frac{e_p - a_h}{e_p} \times p_{m_j})$ for the estimation of false positive rate.

We compare our analytical estimations with the results from our simulation, shown in Fig. 1(c), where g/n = 1.0. The difference in false positive rate between analysis and evaluations is 0.01 in **ham** and 0.02 in the network of **fb**, **wiki** and **astro**. We get exactly same results from our analytical model and simulation in the social network datasets of **flic** and **cat** (the largest network in our evaluations).

6. Simulation and results

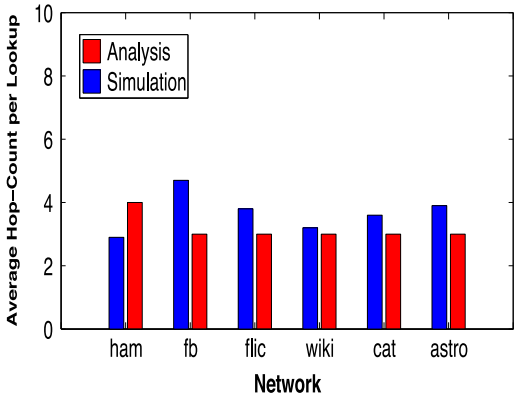
In this section, we describe the design of our simulation and present the results of our experiments. We build our Sybil detection mechanism on top of Persea system and inherit hierarchical node ID distribution and certification, routing table organization and replication mechanisms of Persea (Al-Ameen and Wright, 2014). We incorporate our Sybil-filtering scheme with the lookup mechanism of Persea to achieve higher lookup success rate.

We evaluate for a 31-bit ID space. For different system parameters, we use the same values as used in Persea, such as: chunk-factor c_f=0.65, redundancy R=7, and Kad parameters $\alpha = 5$, $\beta = 7$, and bucket size k=7. In our evaluations, we assume that each node has one collaborative friend at each of its upper layers.

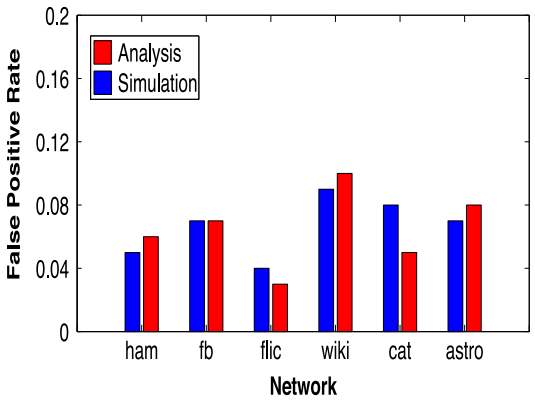
We simulate for networks with different clustering coefficients and the experimental results show that the effectiveness of our Sybil detection scheme and the lookup success rate do not depend on the fast-mixing nature of a network. We represent the number of attack edges by g and the number of benign peers by n. To validate our claim that iPersea gains high lookup success rate, even for a high value of g/n, we evaluate for this ratio, up to 1.5.

6.1. Building the network and joining of attackers

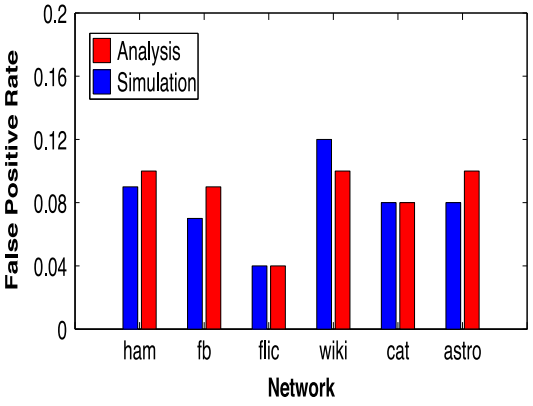
We follow the exact same approach, as in Persea (Al-Ameen and Wright, 2014), for building the network and joining of attackers. So, we build the bootstrap tree by emulating the process



(a) Average hop-count per lookup



(b) Rate of false positive (trusted collaborative friends)



(c) Rate of false positive (randomly selected collaborative friends)

Fig. 1. Comparison between results from analysis and simulation.

of nodes joining via existing connections in a social network graph. Although our system does not rely on the structure of the social graph for its security properties, we use real social network graphs to provide a realistic basis for the choices that nodes make in building the tree.

As we construct the initial bootstrap graph, the nodes in these datasets are considered to be honest. We build our system starting with seven bootstrap nodes. In deployment, bootstrap nodes would be the users who take initiative to build the system. In our experiments, we choose seven highly connected nodes from the social network to start building the network. We then use breadth-first-search over the social graph to add other nodes. A link between node P and node Q in the dataset is interpreted as an invitation from node P to node Q. Thus, P becomes Q's parent in the bootstrap graph. Also, P and Q add each other to their k-buckets.

After adding all of the honest nodes, we add Sybil nodes by creating attack edges to randomly selected honest peers. An attack edge represents an invitation from the honest node, providing the attacker with a certified ID and chunk of ID space through which it could invite more Sybil nodes. An attack edge can be created with a benign peer from any level of the hierarchical ID space.

One may think that the attacker is at a disadvantage by being added after honest nodes build a bootstrap tree. This is not the case. The attackers have an equal chance to get attack edges at all levels of the tree, and there are always chunks to be given out at the highest levels of the tree. To demonstrate this in Persea (Al-Ameen and Wright, 2014), the authors examined the ratio of attack edges to honest nodes (g/n) in their simulations for all levels of the ID space, where the ratios are roughly equal across the levels.

We evaluate our mechanism in simulations for one collaboration network: ca-AstroPh (**astro**) and five social network datasets: facebook (**fb**), flickr (**flic**), catster (**cat**), wiki-Vote (**wiki**) and hamsterster (**ham**). Here, **wiki** uses directed edges to indicate "who trusts whom", which we believe is a good proxy for the notion that parent node would accept another node as a child in the bootstrap graph. In our evaluations, **ham**, **fb**, **flic** and **cat** are social networks drawn from the users on http://Hamsterster.com, http://Facebook.com, http://Flickr.com and http://Catster.com Websites, respectively.

In our evaluations, **cat** is the largest network with 149700 nodes and 5449275 edges. While considering clustering coefficients, **ham** and **astro** are the networks with the smallest (0.08) and largest (0.63) clustering coefficient, respectively. Table 1 shows the sizes and clustering coefficients of the network datasets.[1]

[1] http://snap.stanford.edu/data, http://konect.uni-koblenz.de, http://socialcomputing.asu.edu/pages/datasets.

6.2. Rate of false positive and false negative

In this section, we show the results for the rate of false positive and false negative in our Sybil detection mechanism, when the collaborative friends are either trusted or randomly selected.

When trusted nodes are selected as collaborative friends, the results in Table 2 show that for $g/n = 1.0$, the rate of false positive is 0.05 in **ham** (network smallest clustering coefficient) and 0.067 in **astro** (network largest clustering coefficient). When the number of attack edges is equal to the number of honest peers, the maximum rate of false positive for any network is found 0.09 in **wiki**.

For the random selection of collaborative friends, the rate of false positive slightly increases as compared to the trusted collaborative friends; for example, in **cat**, the largest network in our experiments, when $g/n = 1.0$, the rates of false positive are 0.082 and 0.087 for trusted and randomly selected collaborative friends, respectively. Table 3 shows the rates of false positive for randomly selected collaborative friends.

The rate of false negative is zero for trusted collaborative friends. However, for randomly selected collaborative friends, the rates of false negative vary in the range between 0.001 (**fb**) and 0.04 (**ham**), when the ratio of attack edges to the number of honest peers is one (shown in Table 4). The rate of false negative gets much lower for decreasing g/n; for example, in **flic**, the rate of false negative is zero when $g/n = 0.80$.

6.3. Lookup success rate

We evaluate Persea with majority voting scheme and then compare the lookup success rate with iPersea for networks with different clustering coefficients, shown in Fig. 2. The results show that iPersea performs much better than Persea. For increasing g/n, lookup success rate in Persea decreases sharply. We evaluate iPersea for g/n up to 1.5 and find consistent results for increasing g/n; in iPersea, the lookup success rate for any network in our simulation is no less than 92.9%. We get 100% lookup success rate in **cat** and **flic** for both trusted and randomly selected collaborative friends, even when the number of attack edges is equal to the number of honest peers in our system.

When $g/n = 1.0$ in **ham** (network with lowest clustering coefficient), the lookup success rate in Persea is 59%, whereas the percentage of successful lookup in iPersea is 100% for trusted collaborative friends and 99% for randomly selected collaborative friends. In **astro** (network with largest clustering coefficient), 54% lookups succeed in Persea when $g/n = 1.0$. For the same ratio, the lookup success rates in iPersea are 97.5% and 96.6% for trusted and randomly selected collaborative friends, respectively.

Our experimental results show that the lookup success rates for trusted and randomly selected collaborative friends remain same in **cat**, **flic** and **wiki**, when the number of attack edges is equal to the number of honest peers. For the same ratio of attack edges to honest peers, while comparing these two approaches of selecting collaborative friends, the differences in the percentage of successful lookup in other networks are as follows: 0.04% in **fb**, 0.09% in **astro** and 1% in **ham**. Hence, the lookup success rates for random selection of collaborative friends are very close to that in the ideal scenario, which assumes, the collaborative friends are trustworthy.

6.4. Overhead

We evaluate to figure out the overhead of our system in terms of average hop-count per lookup. Our experimental results show the overhead for both inspection and regular lookups, when the collaborative friends are either trusted or randomly selected. We also compare our overheads for regular lookup with Persea.

Inspection lookup: Let us assume, the role of node C as the intermediate hop gets inspected. In ideal case, node C has the target of the lookup in its friend-list and in this scenario, the number of intermediate peer to reach the target is one. However, hop-count increases if the target node is not a direct friend of node C. Table 5 shows the results for average hop-count per inspection lookup.

The results illustrate that when the trusted nodes are selected as collaborative friends, average hop-count for different networks vary in the range between 1.10 (**astro**) and 1.69 (**cat**). For random selection of collaborative friends, the minimum and maximum average hop-counts per inspection lookup are 1.11 (**astro**) and 1.71 (**cat**), respectively. Hence, the differences between these two approaches of selecting collaborative friends are quite subtle, in terms of average hop-count per inspection lookup.

Regular lookup: Our evaluations figure out the overheads associated with regular lookups, shown in Table 6, which also represent the overhead for a peer to get the status of a node from its parent. The increase in hop-count is found very small, when the collaborative friends are randomly selected instead of selecting the trusted peers only.

The results show that iPersea achieves some improvements over Persea while considering the average hop-count per regular

Table 1
Topologies.

Network		Nodes	Edges	Avg. Clustering Coeff.
hamsterster	(**ham**)	2426	16,631	0.08
facebook	(**fb**)	63,731	1,545,686	0.15
flickr	(**flic**)	80,513	5,899,882	0.17
wiki-Vote	(**wiki**)	7115	103,689	0.21
catster	(**cat**)	149,700	5,449,275	0.43
ca-AstroPh	(**astro**)	18,772	396,160	0.63

Table 2
Rate of false-positive for varying g/n [Trusted collaborative friends].

g/n	0.10	0.50	0.80	1.0	1.25	1.50
ham (0.08)	0.046	0.047	0.049	0.05	0.09	0.095
fb (0.15)	0.063	0.064	0.066	0.067	0.069	0.07
flic (0.17)	0.037	0.039	0.04	0.042	0.048	0.055
wiki (0.21)	0.07	0.078	0.08	0.09	0.092	0.093
cat (0.43)	0.077	0.079	0.08	0.082	0.09	0.12
astro (0.63)	0.06	0.063	0.064	0.067	0.09	0.11

Table 3
Rate of false-positive for varying g/n [Randomly selected collaborative friends].

g/n	0.10	0.50	0.80	1.0	1.25	1.50
ham (0.08)	0.046	0.047	0.08	0.09	0.14	0.19
fb (0.15)	0.063	0.064	0.067	0.068	0.071	0.073
flic (0.17)	0.037	0.039	0.04	0.045	0.052	0.06
wiki (0.21)	0.07	0.078	0.09	0.121	0.124	0.126
cat (0.43)	0.077	0.08	0.084	0.087	0.097	0.13
astro (0.63)	0.06	0.07	0.079	0.083	0.108	0.13

Table 4
Rate of false-negative for varying g/n [Randomly selected collaborative friends].

g/n	0.10	0.50	0.80	1.0	1.25	1.50
ham (0.08)	0.0	0.0	0.031	0.04	0.05	0.095
fb (0.15)	0.0	0.0	0.001	0.001	0.002	0.003
flic (0.17)	0.0	0.0	0.0	0.003	0.004	0.005
wiki (0.21)	0.0	0.0	0.01	0.031	0.032	0.033
cat (0.43)	0.0	0.001	0.004	0.005	0.007	0.01
astro (0.63)	0.0	0.007	0.015	0.016	0.018	0.02

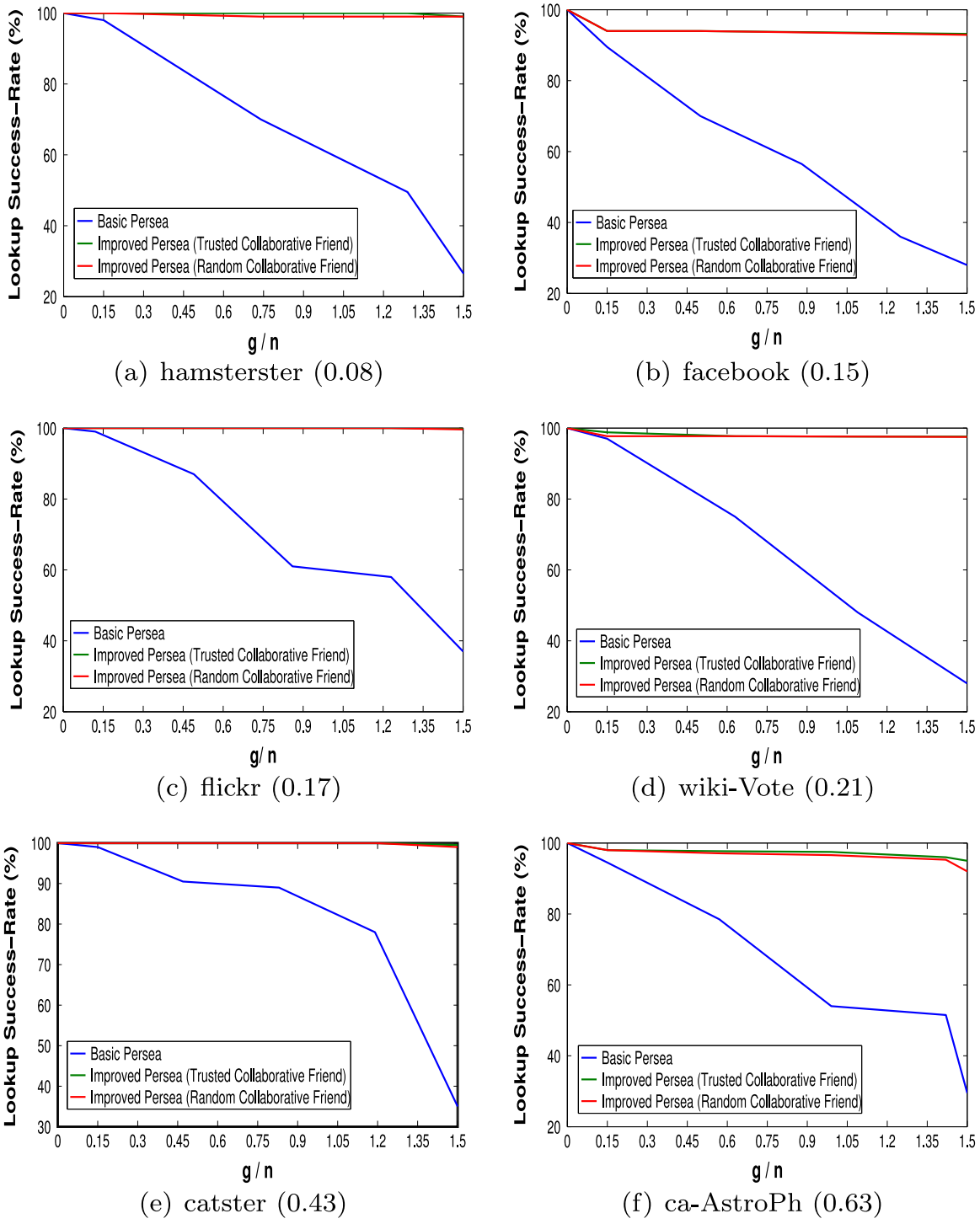


Fig. 2. Lookup success rates in networks with different clustering-coefficients.

Table 5
Average hop-count in inspection lookup [g/n = 1.5].

Network	Trusted collaborative friend	Random collaborative friend
ham (0.08)	1.27	1.35
fb (0.15)	1.29	1.29
flic (0.17)	1.41	1.42
wiki (0.21)	1.24	1.26
cat (0.43)	1.69	1.71
astro (0.63)	1.10	1.11

Table 6
Average hop-count in regular lookup [g/n = 1.5].

Network	Persea	iPersea (Trusted collaborative friend)	iPersea (Random collaborative friend)
ham (0.08)	2.85	2.80	2.84
fb (0.15)	4.79	4.70	4.70
flic (0.17)	3.84	3.44	3.50
wiki (0.21)	3.24	3.18	3.20
cat (0.43)	3.69	3.59	3.62
astro (0.63)	3.93	3.87	3.88

lookup. In both Persea and iPersea, the maximum hop-count per regular lookup is found in **fb**, which is 4.79 in Persea and 4.70 in iPersea for both trusted and random collaborative friends. In **cat**, the largest network of our simulation, average hop-count per regular lookup is 3.69 in Persea, which 3.59 for trusted collaborative friends and 3.62 for randomly selected collaborative friends in iPersea.

7. Related work

Sybil attacks can be leveraged to greatly undermine the operations of a variety of systems, including P2P systems (as examined in this paper), online social networks (OSNs), wireless sensor networks, online ratings systems, and more. Because of the power and generality of the Sybil attack, a large number of defenses have been proposed (Levine et al., 2006). Since we have already discussed about Persea, in this section we examine other major approaches proposed in literature that use social network in their Sybil defense mechanisms.

A number of works have proposed Sybil detection techniques or Sybil resistance based on random walks over a social network (Lesniewski-Laas, 2008; Lesniewski-Laas and Kaashoek, 2010; Mittal et al., 2012; Yu et al., 2008, 2006; Danezis and Mittal, 2009; Wei et al., 2012; Xinhui et al., 2015). The basic idea is that we can divide the social network into a Sybil region and an honest region connected via a small number of attack edges (a *small cut*). Random walks starting from the honest region have a low probability of ending in the Sybil region. This can be leveraged in a variety of ways, leading to detection mechanisms (Yu et al., 2006, 2008; Danezis and Mittal, 2009; Wei et al., 2012), and Sybil-resistant P2P designs (Lesniewski-Laas, 2008; Lesniewski-Laas and Kaashoek, 2010; Mittal et al., 2012). In this case, Sybilshield (Shi et al., 2013; Gunturu, 2015) assumes that there exist multiple number of large, medium and small communities in real-time social networks. The authors conducted a real-time experiment on Myspace dataset, which reveals that it can be divided into 19 communities (Shi et al., 2013).

All of these mechanisms require the absence of small cuts within the honest region in the underlying social network (i.e., the honest region should be fast-mixing). Experimental results of Mohaisen et al., however, show that the mixing time of many real social networks is slower than the mixing time assumed by these works (Mohaisen et al., 2010). Mohaisen et al. also point out that some of these works have made questionable assumptions in their evaluations, which may have helped lead to the good results that have been published for these schemes (Mohaisen et al., 2010). Additionally, many real-world social networks fail to satisfy the other requirements of the systems, either because a significant fraction of nodes are sparsely connected or the users are organized in small tightly-knit communities, which are sparsely interconnected (Viswanath et al., 2012).

Lesniewski-Laas proposes a protocol (Lesniewski-Laas, 2008) in which a node constructs its routing table through independent random walks and recording the final node in each walk as the finger in its routing table. The protocol (Lesniewski-Laas, 2008) is extended in Lesniewski-Laas and Kaashoek (2010) where the idea of layered identifiers is introduced to counter clustering attacks.

As mentioned before, Whānau relies on the assumptions of a fast-mixing social network and small number of attack edges, which may not hold in real social networks (Mohaisen et al., 2010; Viswanath et al., 2010; Yang et al., 2011; Bilge et al., 2009). Lesniewski-Laas et al.'s own results on mixing in real social networks (Lesniewski-Laas and Kaashoek, 2010) still show a noticeable gap from the expected result for a fast-mixing network. Also, Whānau requires significant routing table state on the order of $O(\sqrt{n}\log n)$, where n is the number of objects stored in the DHT. Mittal et al. point out that the network overhead for maintaining this state can be substantial (e.g. 800 Kbps per node) (Mittal et al., 2012).

X-Vine (Mittal et al., 2012) works by communicating over social network edges. It builds a DHT on the top of a social network, where each node in the system selects a random numeric identifier. In the identifier space, each node maintains paths to its neighbors. In X-Vine, honest peers rate-limit the number of paths that are allowed to be built over their adjacent edges, which helps to limit the number of Sybil nodes that can join the system. X-Vine, however, relies on the fast-mixing assumption and was only evaluated with a small number of attack edges (one for every ten honest nodes).

8. Future work

In our future work, we would extend our attacker detection scheme to handle the *oscillation attack*, in which case, an attacker performs as both honest and malicious peer at regular interval. So, if an inspection lookup is performed during its honest-role, the attacker may get '+' status. To prevent such attacks, the parent node continues to perform inspection lookup for the child with '+' status at random interval and once a node gets '−' status, it is sealed permanently, since we assume that an honest node always performs legitimately.

The above mechanism is also effective in getting the current status of a node, whose role is changed from honest to malicious because of being compromised by an attacker. In our future work, we would implement the above strategies to make our system robust against such attacks. Moreover, we would evaluate our system for larger datasets in future work.

9. Conclusion

In this paper, we propose a Sybil detection mechanism, which accommodates a specially designed lookup mechanism to detect the Sybils and a filtering mechanism to opt-out those detected attackers during regular DHT lookup. We incorporate our mechanisms with Persea to develop the system: iPersea, which inherits the advantages of Persea that it gains over prior systems, and ensures higher lookup success rate even when the malicious targets respond with incorrect results. Our Sybil detection mechanism can be amended for incorporating with any DHT, where the children of a node are connected through their parents, required to implement the inspection lookup.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1117866 and CAREER Grant No. CNS-0954133.

References

Al-Ameen, M.N., Wright, M., 2014. Design and evaluation of Persea, a Sybil-resistant DHT. In: AsiaCCS.

Bilge, L., Strufe, T., Balzarotti, D., Kirda, E., 2009. All your contacts are belong to us: automated identity theft attacks on social networks. In: WWW.

Cholez, T., Chrisment, I., Festor, O., 2009. Evaluation of Sybil attacks protection schemes in KAD. In: International Conference on Autonomous Infrastructure, Management and Security: Scalability of Networks and Services.

Danezis, G., Mittal, P., 2009. SybilInfer: detecting Sybil nodes using social networks. In: NDSS.

Douceur, J.R., 2002. The Sybil attack. In: International workshop on Peer-to-Peer

Systems (IPTPS).
Grabowski, A., Kosinski, R., 2008. Mixing patterns in a large social network. Acta Phys. Pol. B 39, 1291.
Gunturu, R., 2015. Survey of Sybil Attacks in Social Networks, Technical Report.
Lesniewski-Laas, C., Kaashoek, M.F., 2010. Whānau: a Sybil-proof distributed hash table. In: USENIX NSDI.
Lesniewski-Laas, C., 2008. A Sybil-proof one-hop DHT. In: Workshop on Social Network Systems.
Levine, B., Shields, C., Margolin, N., 2006. A Survey of Solutions to the Sybil Attack, Technical Report, 2006-052, University of Massachusetts Amherst.
Lian, Q., Zhang, Z., Yang, M., Zhao, B.Y., Dai, Y., Li, X., 2007. An empirical study of collusion behavior in the Maze P2P file-sharing system. In: IEEE ICDCS.
Maymounkov, P., Mazieres, D., 2002. Kademlia: a peer-to-peer information system based on the XOR metric. In: International Workshop on Peer-to-Peer Systems (IPTPS).
Mislove, A., Marcon, M., Gummadi, K.P., Druschel, P., Bhattacharjee, B., 2007. Measurement and analysis of online social networks. In: Proceedings of ACM SIGCOMM.
Mittal, P., Caesar, M., Borisov, N., 2012. X-Vine: secure and pseudonymous routing in DHTs using social networks. In: NDSS.
Mohaisen, A., Yun, A., Kim, Y., 2010. Measuring the mixing time of social graphs. In: ACM Internet Measurement Conference (IMC).
Ratnasamy, S., Francis, P., Handley, M., Karp, R.M., Shenker, S., 2001. A scalable content-addressable network. In: Proceedings of ACM SIGCOMM.
Rowstron, A., Druschel, P., 2001. Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Lect. Notes Comput. Sci. 2218, 329, URL citeseer.ist.psu.edu/rowstron01pastry.html.
Shi, L., Yu, S., Lou, W., Hou, Y.T., 2013. Sybilshield: an agent-aided social network-based Sybil defense among multiple communities. In: INFOCOM, 2013 Proceedings IEEE, IEEE, Turin, Italy, pp. 1034–1042.
Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H., 2001. Chord: a scalable peer-to-peer lookup service for internet applications. In: Proceedings of ACM SIGCOMM.
Tran, N., Li, J., Subramanian, L., Chow, S.S., 2011. Optimal Sybil-resilient node admission control. In: IEEE INFOCOM.
Viswanath, B., Post, A., Gummadi, K.P., Mislove, A., 2010. An analysis of social network-based Sybil defenses. In: ACM SIGCOMM.
Viswanath, B., Mondal, M., Clement, A., Druschel, P., Gummadi, K.P., Mislove, A., Post, A., 2012. Exploring the design space of social network-based Sybil defenses. In: COMSNETS.
Wei, W., Xu, F., Tan, C.C., Li, Q., 2012. SybilDefender: defend against Sybil attacks in large social networks. In: INFOCOM.
Wolchoky, S., Hofmanny, O.S., Heninger, N., Felten, E.W., Halderman, J.A., Rossbach, C.J., Waters, B., Witchel, E., 2010. Defeating vanish with low-cost Sybil attacks against large DHTs. In: NDSS.
Xinhui, H., Xianquan, X., Jianyu Zhang, B.L., et al., 2015. Sybil defenses in DHT networks based on social relationships. J. Tsinghua Univ. (Sci. Technol.) 54 (1), 1–7.
Yang, M., Zhang, Z., Li, X., Dai, Y., 2005. An empirical study of free-riding behavior in the Maze P2P file-sharing system. In: International workshop on Peer-to-Peer Systems (IPTPS).
Yang, Z., Wilson, C., Wang, X., Gao, T., Zhao, B.Y., Dai, Y., 2011. Uncovering social network Sybils in the wild. In: IMC.
Yu, H., Kaminsky, M., Gibbons, P.B., Flaxman, A., 2006. SybilGuard: defending against Sybil attacks via social networks. In: ACM SIGCOMM.
Yu, H., Gibbons, P.B., Kaminsky, M., Xiao, F., 2008. SybilLimit: a near-optimal social network defense against Sybil attacks. In: IEEE Symposium on Security and Privacy.