Persea: A Sybil-Resistant Social DHT

Mahdi N. Al-Ameen
The University of Texas at Arlington
Arlington, TX, USA
mahdi.al-ameen@mavs.uta.edu

Matthew Wright
The University of Texas at Arlington
Arlington, TX, USA
mwright@cse.uta.edu

ABSTRACT

P2P systems are inherently vulnerable to Sybil attacks, in which an attacker can have a large number of identities and use them to control a substantial fraction of the system. We propose Persea, a novel P2P system that is more robust against Sybil attacks than prior approaches. Persea derives its Sybil resistance by assigning IDs through a bootstrap tree, the graph of how nodes have joined the system through invitations. More specifically, a node joins Persea when it gets an invitation from an existing node in the system. The inviting node assigns a node ID to the joining node and gives it a chunk of node IDs for further distribution. For each chunk of ID space, the attacker needs to socially engineer a connection to another node already in the system. This hierarchical distribution of node IDs confines a large attacker botnet to a considerably smaller region of the ID space than in a normal P2P system. Persea uses a replication mechanism in which each (key,value) pair is stored in nodes that are evenly spaced over the network. Thus, even if a given region is occupied by attackers, the desired (key,value) pair can be retrieved from other regions. We compare our results with Kad, Whanau, and X-Vine and show that Persea is a better solution against Sybil attacks.

Categories and Subject Descriptors

C.2.4 [**Communication Networks**]: Distributed Systems-Distributed applications

Keywords

Sybil attack, security, social DHT

1. INTRODUCTION

Peer-to-peer (P2P) systems are highly susceptible 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 [1, 2]. 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 Kademlia [8], which is widely used in file-sharing systems.

Recent research has focused on leveraging information from social networks to make the system robust against Sybil attackers, resulting in several decentralized apporaches [6, 7, 13]. In these defenses, it is often required that the social network be *fast-mixing*, meaning that a random walk in the honest part of the network approaches the unifrom distribution in a small number of steps. However, a recent study shows that the mixing times of real-world social networks may not be as fast as what these approaches assume [10]. Further, Viswanath et al. have shown that a number of Sybil defenses are ineffective for these slower-mixing, highly *modular* social networks [12].

Contributions. In this paper, we propose a new Sybil-resistant DHT called Persea that addresses these problems and provides better Sybil resistance than the state of the art. The Persea approach offers a number of important advantages over existing schemes:

- A Sybil attacker is limited to isolated regions of the ID space.
- Our system does not depend on the assumption that the social networks are fast-mixing, making Persea more dependable in real-world scenarios.
- 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; such lists may also bear little resemblence to social connections inside the P2P system.
- Although we test it with a DHT routing table design similar to Kademlia, which is widely used, it can be adapted to other DHT routing tables.
- IDs are certified, making attacks based on ID forging impossible outside of attacker-controlled ID ranges.

Our simulation results show that Persea performs much better than Kad [3] in terms of lookup success-rates, e.g. with 100% success compared to just 59.3% when the ratio of attack edges to honest nodes is 0.15. In Whānau [6,7] and X-Vine [9], the success rates are less than 100% for similar scenarios.

2. SYSTEM DESIGN

In this section, we describe the design of Persea. We begin with a brief attack model. We then overview the system and address ID space allocation, ID certification, and key replication. Persea uses the routing table organization and lookup mechanism of Kademlia [8].

Attack Model. A link between an honest node and a malicious node is called an *attack edge*, and it represents a successful act of social engineering to cause the user to accept the malicious node as a social connection [7, 13]. In Persea, creating an attack edge means obtaining an invitation to join the network as a child of the inviting node in the

CODASPY'13, February 18–20, 2013, San Antonio, Texas, USA.
ACM 978-1-4503-1890-7/13/02.

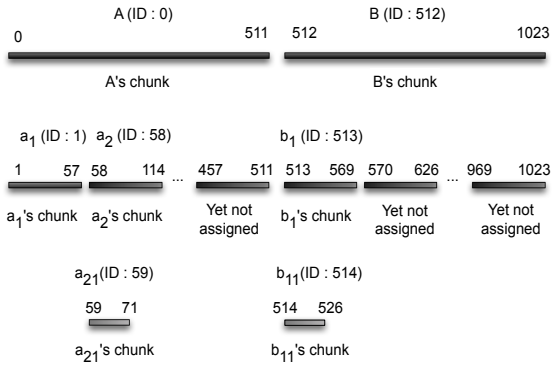

Figure 1: Hierarchical Distribution of Node IDs

bootstrap tree. When an attacker joins the network, it gets a chunk of node IDs for further distribution and may invite more attacker nodes to join the network. During lookup, when the attacker gets a request to return the value associated with the search-key, it simply drops the message and does not reply as part of a denial of service attack.

2.1 Design Overview

Persea consists of two layers: the bootstrap layer and the DHT layer. In this paper, 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 social 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 boostrap nodes.

The number of nodes that a peer can invite is limited by the number of node IDs in its chunk. Thus, there is an incentive for peers to only invite other peers based on actual social connections and to limit the size of each chunk that it gives out so that it does not run out of node IDs. In this way, Persea offers some resilience against social engineering. We would also leverage the user interface to warn users during the invitation process to not invite strangers.

The DHT layer of Persea is based on Kademlia [8], 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 our ID distribution scheme.

2.2 Hierarchical ID Space

We now describe how node IDs are distributed in Persea. Each bootstrap node has a contiguous range of node IDs

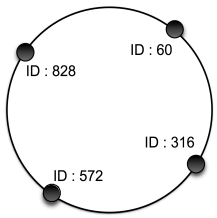

Figure 2: Evenly spaced target nodes

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 we have more than one bootstrap node, then we 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.

We briefly explain the mechanism with an example illustrated in Figure 1. Let A and B be two bootstrap nodes that initiate building the system. If we consider a b-bit ID space, then the total number of IDs in the DHT n_{max} would be 2^b. In this example, we consider a 10-bit ID space, so $n_{max} = 2^{10}$. If Z is the number of bootstrap nodes, $\lfloor \frac{n_{max}}{Z} \rfloor$ represents the number of node IDs that each bootstrap node has in its chunk (with a small difference for the bootstrap node with the highest ID). In this example, both node A and node B have 512 node IDs. The lowest node ID in a chunk is assigned to the bootstrap node itself and the remaining node IDs are for further distribution to the newly joined nodes. In this example, node A's ID is 0 and the interval $[1, 511]$ is its chunk of IDs for further distribution.

Each node divides its chunk into sub-chunks based on the chunk-factor. Let n_c be the number of node IDs in a chunk and n_s represent the number of node IDs in each of its sub-chunks (except the last sub-chunk). In a chunk, the lowest ID is assigned to the owner node and the remaining node IDs are for further distribution. Thus $n_c - 1$ represents the number of node IDs in a chunk available for distribution by the owner node. If the chunk-factor is c_f $(0 \leq c_f \leq 1)$ then n_s would be $\lfloor (n_c - 1)^{c_f} \rfloor$. So, the number of sub-chunks that can be created from a chunk is $\lfloor \frac{n_c - 1}{n_s} \rfloor + 1$. This also represents the maximum number of nodes that can be invited by a node having chunk of size n_c.

Let $c_f = 0.65$ in this example. Node A divides its chunk into nine sub-chunks where each sub-chunk (except the last one) accommodates 57 node IDs and the last sub-chunk has 55 IDs. Node a_1 joins the network after getting an invitation from node A and node A assigns a sub-chunk to node a_1. The lowest node ID in this sub-chunk is 1, which is assigned

as the node ID of node a_1 and the interval $[2, 57]$ represents the remaining node IDs of the sub-chunk that are for further distribution by node a_1. As the chunk space becomes smaller with a deeper tree, a large ID space and appropriate selection of the chunk factor is critical for large systems.

In Persea, we assume that each node knows the value of Z, b and c_f. Thus, if an inviter node intends to assign a node ID to the joining node out of its chunk, the joining node can easily verify it, because any node in Persea can calculate the chunk distribution.

2.3 Replication Mechanism

We describe our replication mechanism in this section, which is the key difference between the Persea DHT layer and Kademlia [8].

When the initiator intends to store or retrieve a (key, value) pair in Persea, it calculates the node ID of the target nodes as follows. Assume a b-bit ID space, such that $n_{max} = 2^b$. We virtually divide the ID space into R regions where each region (except the last one) accommodates at most $D = \lfloor \frac{n_{max}}{R} \rfloor$ IDs, and the last region has $n_{max} - D \times (R-1)$ IDs. The interval $[r, r + D - 1]$ for $0 \leq r < R - 1$ represents the node IDs that are in the rth region; the last region spans $[D \times (R-1), n_{max} - 1]$. A node ID i is replicated to each other region by taking $(i + D \times r)$ mod n for $1 \leq r < R$. In other words, the information is replicated evenly around the ID space to R locations.

In Figure 2, we show an example of key replication for a 10-bit ID space and $R = 4$, where the key of the (key, value) pair to be stored is 60. The target nodes are evenly spaced around the circular ID space.

3. SIMULATION AND RESULTS

We evaluate Persea in simulations for the social network dataset of wiki-Vote (7,115 nodes, 103,689 edges) [4,5] and soc-Epinions1 (75,879 nodes, 508,837 edges) [11].In our experiments, the nodes in these datasets are considered to be honest and attacker nodes are dynamically added to the network. We use n to refer to the number of honest nodes and g to denote the number of attack edges.

We compare our results with other DHTs: Kad [3], Whanau [6, 7], and X-Vine [9]. We simulate Kad to get the lookup success-rate for varying attack edges. The results show that Persea performs much better than Kad in terms of lookup success-rate (see Figure 3). For Whanau and X-Vine, we compare our results with the results reported in [6, 7, 9].

Whanau: The network size used in [6] is comparable to our wiki-Vote dataset. Our simulation results show that for $g/n = 0.15$ the lookup success rate in Persea is 100%, which is higher than that reported in [6]. For higher values of g/n up to one, the lookup success rate in [6] is no better than Persea. Moreover, Whanau is built upon one-hop DHT routing mechanism and has high maintenance overheads [9].

X-Vine: For different network sizes, the maximum value of g/n considered in [9] is no greater than 0.1. For this value of g/n the maximum probability of success reported in [9] is less than one. But in Persea for higher value of g/n, which is 0.15 we find that the percentage of successful lookup is 100% for wiki-Vote and 99% for soc-Epinoins1 dataset. In topologies with 100,000 nodes, X-Vine requires 10-15 hops for routing. In Persea, the average hop-count per lookup is only 3.58 for a topology of 75,879 nodes (soc-Epinions1)

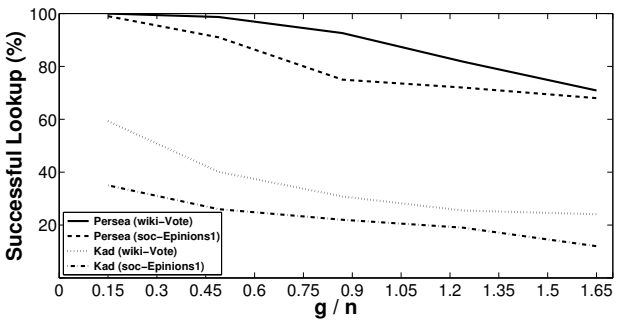

Figure 3: Comparison between Persea and Kad

and we extrapolate that it would remain less than four hops for 100,000 nodes.

4. FUTURE WORK

We would implement Persea in larger networks and our future experiments would include detail performance and overhead analysis for varying system parameters. Also, we would perform theoretical analysis of Persea to find the probability of lookup failure for varying attack edges.

5. ACKNOWLEDGEMENT

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

6. REFERENCES

[1] T. Cholez, I. Chrisment, and O. Festor. Evaluation of Sybil attacks protection schemes in KAD. In *AIMS: Scalability of Networks and Services*, 2009.

[2] J. R. Douceur. The Sybil attack. In *IPTPS*, 2002.

[3] H. J. Kang, E. Chan-Tin, N. J. Hopper, and Y. Kim. Why Kad lookup fails. In *P2P*, 2009.

[4] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Predicting positive and negative links in online social networks. In *WWW*, 2010.

[5] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Signed networks in social media. In *CHI*, 2010.

[6] C. Lesniewski-Laas. A Sybil-proof one-hop DHT. In *Workshop on Social Network Systems*, 2008.

[7] C. Lesniewski-Laas and M. F. Kaashoek. Whānau: A Sybil-proof distributed hash table. In *NSDI*, 2010.

[8] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information sytem based on the XOR metric. In *IPTPS*, 2002.

[9] P. Mittal, M. Caesar, and N. Borisov. X-Vine: Secure and pseudonymous routing in DHTs using social networks. In *NDSS*, 2012.

[10] A. Mohaisen, A. Yun, and Y. Kim. Measuring the mixing time of social graphs. In *IMC*, 2010.

[11] M. Richardson, R. Agrawal, and P. Domingos. Trust management for the semantic web. In *ISWC*, 2003.

[12] B. Viswanath, A. Post, K. P. Gummadi, and A. Mislove. An analysis of social network-based Sybil defenses. In *ACM SIGCOMM*, 2010.

[13] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. SybilLimit: A near-optimal social network defense against Sybil attacks. In *IEEE S&P*, 2008.