The Sybil Attack From the Attacker's Perspective

Andrew Rosen

November 30, 2014

Abstract

This paper explores the feasibility of performing naive Sybil attacks that completely occlude healthy nodes from each other. The vulnerability of Distributed Hash Tables to Sybil attacks and Eclipse attacks has been well known for some time. However, these vulnerabilities has often been explored in a theoretical sense, assuming the attacker is a global adversary from the beginning, nigh-omniscient and omnipotent. This paper examines, from a analytical and practical perspective, how valid that assumption is.

We examine the amount of computational effort required to become a global adversary starting as a non-global adversary. We do this by analyzing the amount of time it takes an attacker with a given IP to choose a port to obtain a desired hashkey, a process we call *potatoing*. We present potatoing to emphasize the ease of this attack, but also demonstrate potential non-security uses of potatoing that are beneficial to DHT load-balancing.

1 Introduction

One of the key properties of structured peer-to-peer (P2P) systems is the lack of a centralized coordinator. P2P systems remove the vulnerability of a single point of failure and and the susceptibility to a denial of service attack (cite original Sybil paper), but in doing so, open themselves up to new attacks. Completely decentralized P2P systems are vulnerable to *Eclipse attacks*, whereby an attacker completely occlude healthy nodes from one another. This prevents them from communicating without being intercepted by the attacker.

One way to accomplish this attack is to perform a *Sybil Attack* [?] [4]. In a Sybil attack, the attacker creates multiple malicious virtual nodes in order to disrupt the network. If enough malicious nodes are injected into the system, the majority of the nodes will be occluded from one another, successfully performing an Eclipse attack.

Security analyses typically assumes an adversary using a Sybil is omniscient and can inject virtual nodes wherever he chooses in a reasonable amount of time. Our goal is to demonstrate how veracity of that assumption by doing analysis and simulations.

Sybil attacks represent a significant threat to the security of any distributed system. Many of the analyses [1] on Tor [3] emphasize the vulnerability of Tor to the Sybil attack. This threatens the anonymity of Tor users, particularly those living in countries with peculiar notions about personal privacy [?].

P2P systems like BitTorrent are essential to a wide variety of users. BitTorrent, for instance, is the *de facto* platform for distributing large files scalably among tens of thousands of clients. An estimated 20 million peers use BitTorrent daily for sharing and retrieving Linux and BSD-based distros, offline copies of Wikipedia, and personal backup Current research demonstrates BitTorrent is vulnerable to the Sybil attack and a persistent attack disabling BitTorrent would e highly detrimental to many users, but especially developers and system administrators [1].

There have been many suggestions on how to defend against Sybil attack, but there is no agreed upon "silver bullet" among researchers that should be implemented for every distributed application [6] Part of this is surely influenced by the only surefire way to defend against a Sybil attack is to introduce a trusted authority to certify and/or bind identities. This solution potentially removes the Sybil attack, but reintroduces vulnerabilities to denial of service attacks, bringing us full circle. Another reason is that there is no single solution for all platforms is that not every solution is compatible with each distributed platform.

Despite the threat represented by the Sybil attack and the research done on the subject, little research has been done from the perspective of an adversary. We sought to rectify this, both to reemphasize the threat of the Sybil attack, but also because this examination introduces some interesting graph theory problems.

Our work presents the following contributions:

- We first discuss the mechanics of performing a Sybil attack and analyze the theoretical effort needed to bring to bear to perform the Sybil attack.
- We present our simulations which show how quickly even a naive and inefficient Sybil attack can compromise a system. We also discuss how a more intelligent attack can be geared to each of the more popular DHT topologies.
- We analyze an interesting graph coloring problem that an attacker needs to solve if the attack is to remain undetected.
- We discuss the implications of our work, specifically how the techniques we developed to perform the attack can be used for automatic load balancing.

2 Formal Analysis

2.1 Assumptions

In our analysis we make a few assumptions. First, we look primarily at fully distributed systems that assign nodes identifiers using a cryptographic hash function These are called distributed hash tables (DHTs). Cryptographic hash functions work by mapping some input value to an m-bit key or identifier. Well-known hash functions for performing this task include MD5 [7] and SHA1 [5]. Keys generated by the hash function are assumed to be evenly distributed and random [2].

[1]Perhaps the only reason that BitTorrent hasn't be attacked in such a way as to render it unusable is that those capable of doing so rely heavily upon it. A sobering thought, since BitTorrent is under an active Sybil attack.

In distributed hash tables, m is a large value, in order to avoid collisions between mapped inputs, unintentional or otherwise. An $mgeq128$ is typical, with $m = 160$ being the most popular choice.

Our second assumption is that node IDs are generated by hashing their IP address and port. The majority of DHTs present this method as the means of generating node IDs, and although other methods exist[2], they are not nearly as prevalent.

We also assume that nodes store verify that a node's advertised IP address and port generate the advertised node ID. Relaxing the latter assumptions would make the Sybil attack trivial. If the attacker does not have to search for values, he can make up any needed key on the fly and falsely advertise it to the network unchallenged.

Consider an attacker operating under these assumptions who wants to inject a malicious node in between two victims. The attacker must search for a valid IP and port combination under his control that generates a hash key that lies between the two victims's keys, a process we will call *potatoing.* The attackers ability to compromise the network depends on what IP addresses and ports he has available.

2.2 Analysis

Suppose we have a DHT with n members in it, with m-bit node IDs between $[0, 2^m)$. Consider two victim nodes with IDs a and b. The probability P that an attacker can potato a hash key that lands in the range (a, b) is

$$P \approx \frac{|b-a|}{2^m} \cdot num\_ips \cdot num\_ports$$

Where num_ip addresses is the number of IP addresses the attacker has under his control and num_ports is the number of ports the attacker can try for each IP address. If the ports the attacker can try are limited the ephemeral ports, the attacker has 16383 ports to use for each IP address. We can assume that, for a large enough n, node IDs will be close to evenly distributed across the keyspace, meaning there will be $\approx \frac{2^m}{n}$ key between each node ID. This makes the earlier probability equivalent to:

$$P \approx \frac{1}{n} \cdot num\_ips \cdot num\_ports$$

This indicates that the ease of potatoing is independent to m.

The chances of an attacker potatoing all n nodes that partition the entire keyspace is:

$$C \approx 1 - (1 - \frac{1}{n})^{num\_ips \cdot num\_ports}$$

3 Simulations

An essential part of our analysis was demonstrating just how fast an adversary can compromise a system. We performed three experiments to accomplish this

[2]In Mainline DHT, used by BitTorrent, it appears you can choose your own ID at "random."

task. Simulations were performed in Python 2.7 with an AMD Phenom 965 processor.

We used SHA1 for as our cryptographic function, which yields 160-bit hash keys. Again, we use the constraint that victim nodes do an absolute minimum verification which forces an attacker to only present Sybils with hash keys that he can generate with valid IP and port combinations.

Another important aspect of our simulation is that we do not include the memory resources needed in this analysis. This is for a couple of reasons, the primary one being that we assume the attacker is capable and competent. An adversary who wants to use a Sybil attack on a large system would likely roll his own client capable of running on the P2P protocol. Previous research [**?**] shows that an adversary peforming a Sybil attack does not need to maintain the state information of other nodes and can leverage the healthy nodes to perform the routing. Nor does the adversary necessarily create a new running copy for every Sybil created.

If this was impossible, it is still reasonable to assume the attacker's program would be built to have a minimal memory footprint and drop any messages that require an attacker carry any of the network's load.

3.1 Experiment 1: Potatoing 2 random nodes

Our initial experiment was designed to establish the feasibility of injecting in between two random nodes. Each trial, we generated two victims with random IP addresses and ports, and an attacker with a random IP. The experiment was for the attacker to find a key in between the two victims's key, from the lowest to highest. The amount of time to potato two random keys was on average 29.6218323708 microseconds, and was achievable 99.996% of the time.

3.2 Experiment 2: Nearest Neighbor Eclipse via Sybil

The objective of the second experiment is to completely ensnare a network using a Sybil attack, starting with a single malicious node. With simulate this by creating a network of n nodes. Each node is represented by a key generated by taking the SHA1 of a random IP/port combination.

The attacker is given a randomly generated $numIP$ IP addresses, but can use any port between 49152 and 65535?65536. This gives the attacker 16383?16384 $\cdot$ $numIPs$ possible hash keys to use for Sybils. The attacker can easily precompute all of theses hash keys and store them in a sorted list to be used as needed. This requires only $160 \cdot 16384 = 2621440$ bits, or 2.5 megabytes per IP address + IP and port space.

To perform the attack, an adversary choses any random hash key as a starting point to "join" the network. This is his first Sybil and the join process provides information about a number of other nodes. Most importantly, nodes provide information about other nodes that are close in the hash space, which provided to enable fault tolerance between immediate neighbors. The adversary uses this information to inject Sybils in between successive healthy nodes.

For clarity we present this example. Consider the small segment of the network made up of adjacent nodes a, b, c, and d. The Sybil joins between nodes a and b, and the joining process informs the adversary about node c, possibly node d, and a handful of other nodes in the network. The adversary will always learn about node c because a normal node between a and b would need to know about node c for fault tolerance.

The adversary's next move would be to inject a node between nodes b and c. This is done by selecting a precomputed hash key k, such that $b \leq k \leq c$. The adversary injects a Sybil node with key k, which joins in between b and c, and the joining process informs the adversary about node d and several other nodes, including many close nodes. The adversary then aims to inject a node in between c and d, and continues ad nauseum.

Depending on the network size and the number of keys available to the adversary, it is entirely possible the adversary will not have a key to inject between a pair of successive nodes. In this case, the node moves on to the successive pair that the node has learned about.

We simulated this attack on networks of size 50, 100, 200, 300, 400, 500, 1000, 5000, 10000, 15000, 20000, 25000, 50000, and 100000. We gave the attack access to 1, 3, 5, and 10 IP addresses. We presents our results in table **??**.

RESULT TABLE NEED TO TEST TO SEE WHAT NEEDED TO ATTACK 50,000,000 size network, twice the size of the BitTorrent network. CAN WE?

Each experiment took less than a second to perform. Our results show that an adversary, given only modest resources, can inject a Sybil in between the vast majority of successive nodes in moderately sized networks. In a large network, modest resources still can be used to compromise more that a third of the network, an important goal if the adversary wishes to launch a Sybil attack.

However, this only covers the short links of the network, but not the long distance link.

3.3 Experiment 3: Fully Complete Eclipse via Sybil

We extended the previous experiment by considering the long-distance hops of each node in addition to the short-range links of the DHT. We choose to model an attack on a P2P system built using Chord [**?**]. We did this for a number of reasons. Chord is an extremely well understood DHT and is very simple to evaluate using simulations. Nodes in Chord generate long distance links independent of information provided by other nodes, rather than directly querying neighbors. This prevents adversaries from poisoning the node's routing table via false advertisements, which can be on other DHTs such as Pastry [**?**]. A Sybil attack the most straightforward means of effecting an Eclipse attack on a Chord network.

Nodes in Chord have m long-range links, one for each of the bits in the keys, which is 160 in our experiments. Each of node a's long-range hops points to the node with the lowest key $\geq a + 2^i \mod 2^m, 0 \leq i \leq 160$.

The attack is very similar to the Nearest Neighbor attack demonstrated above. Beside injecting a node in between successive nodes, the attacker also attempts to place a Sybil in between each of the long-range links. We simulated this attack under the same parameters as above and are presented in Table **??**.

3.3.1 Chord

Most of the above attacks

Chances the finger is already covered:

3.3.2 Kademlia

In MLDHT you can pick your own ID, so this process is moot. Research has been done on Mainline's vulnerability to Sybil attacks.

3.3.3 Plaxton Based networks

More efficient to lie.

4 Masking the Attack

Now that we have established that a Sybil attack can be performed with great ease, our focus now turns to avoiding detection. The only surefire way to achieve this is by preventing a node from seeing We need a different IP for each point surrounding our victim. In the Nearest-Neighbor attack, we need a

We can reduce this into an interesting graph coloring problem.

The hard maximum, in general, is m seperate IP addresses, one for each bit in a $\log n$ routing/routing table DHT. Recall that the vast

5 Simple Load Balancing Injection Framework

Costs: Need to hold 15000ish hashkeys, effectively 160-bit numbers

References

[1] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource routing attacks against tor. In *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, pages 11–20. ACM, 2007.

[2] Mihir Bellare and Tadayoshi Kohno. Hash function balance and its impact on birthday attacks. In *Advances in Cryptology-Eurocrypt 2004*, pages 401–418. Springer, 2004.

[3] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[4] John R Douceur. The sybil attack. In *Peer-to-peer Systems*, pages 251–260. Springer, 2002.

[5] Donald Eastlake and Paul Jones. Us secure hash algorithm 1 (sha1), 2001.

[6] Brian Neil Levine, Clay Shields, and N Boris Margolin. A survey of solutions to the sybil attack. *University of Massachusetts Amherst, Amherst, MA*, 2006.

[7] Ronald Rivest. The md5 message-digest algorithm. 1992.