Practical Attacks Against The I2P Network

Christoph Egger[1], Johannes Schlumberger[2],
Christopher Kruegel[2], and Giovanni Vigna[2]

[1] Friedrich-Alexander University Erlangen-Nuremberg
Christoph.Egger@cs.fau.de
[2] University of California, Santa Barbara
{js,chris,vigna}@cs.ucsb.edu

Abstract. Anonymity networks, such as Tor or I2P, were built to allow users to access network resources without revealing their identity. Newer designs, like I2P, run in a completely decentralized fashion, while older systems, like Tor, are built around central authorities. The decentralized approach has advantages (no trusted central party, better scalability), but there are also security risks associated with the use of distributed hash tables (DHTs) in this environment.
I2P was built with these security problems in mind, and the network is considered to provide anonymity for all practical purposes. Unfortunately, this is not entirely justified. In this paper, we present a group of attacks that can be used to deanonymize I2P users. Specifically, we show that an attacker, with relatively limited resources, is able to deanonymize a I2P user that accesses a resource of interest with high probability.

1 Introduction

In modern societies, freedom of speech is considered an essential right. One should be able to express his/her opinion without fear of repressions from the government or other members of society. To protect against retribution, the laws of democratic countries recognize the importance of being able to publish information without disclosing one's identity in the process. Unfortunately, this essential right to anonymity is not available on today's Internet.

Local observers, such as Internet providers, site administrators, or users on the same wireless network, can typically track a person while she is using the Internet and build a record of her actions. While encryption hides the actual content transmitted, it is still possible to identify which services are used. Therefore, an observer can link the user to the websites that she visits and, based on these observations, take action.

Tor [1,2] was one of the early solutions to provide anonymous communication on the Internet. It works by routing traffic through a number of intermediate nodes, and each node only knows about its direct communication partners. Hence, looking at the first (or last) link, it is not possible to infer the destination (or source) of the traffic. Tor has a centralized design built around trusted authority servers. Each of these servers keeps track of all nodes in the network and their performance. The authority servers regularly publish this list for clients to use. The clients pick nodes from this list to create encrypted tunnels, until they reach exit nodes. These exit nodes then act as proxies, allowing Tor users to access the public Internet (called *clearnet*) without revealing their identity.

As there are only few trusted authority servers, the integrity of these nodes is essential for the entire network, making them a valuable target for attacks. In addition, since all of the authorities need to keep track of the whole network and regularly agree on its state, this design has limited scalability.

To address limitations of Tor's centralized design, researchers have proposed distributed alternatives. Arguably, the most popular instance of decentralized anonymity systems is I2P. I2P stores all metadata in a distributed hash table (DHT), which is called `netDB`. The DHT ensures scalability of the network. Being run on normal I2P nodes, the `netDB` also avoids a small group of authority servers that would need to be trusted. Finally, I2P provide a separate network (called *darknet*) where both, service providers and users, act only within the I2P network. All connections inside the darknet are end-to-end encrypted, and participants are well-aware of the anonymity of each other.

The use of DHTs in peer-to-peer anonymity systems has been successfully attacked in the past [3]. Continued research on this problem finally led to general results [4] that showed that the additional effort to verify the correctness of lookup results directly increases vulnerability to passive information-leak attacks. I2P itself has been attacked successfully by exploiting the decentralized performance analysis of its participants [5].

The developers of I2P have reacted to the publication of attacks, and they have improved their network to resist the DHT-based attacks introduced in [3] and [4], by limiting the database to a subset of well-performing nodes. This reduces the number of nodes involved in each individual lookup to only one for most cases. Moreover, the performance computation techniques were updated to make it more difficult for an attacker to exploit them. As a result, I2P is considered secure in practice. Unfortunately, this is not entirely justified.

In this paper, we describe an attack that can be used to break the anonymity of a victim who is using anonymized resources in I2P – for example, a user browsing `eepsites` (I2P's terminology for anonymous websites) or chatting. We are able, with high probability, to list the services the victim accesses regularly, the time of access, and the amount of time that is spent using the service

We first show how an attacker can tamper with the group of nodes providing the `netDB`, until he controls most of these nodes. This is possible because I2P has a fixed maximum number of database nodes (only a small fraction of nodes in the entire network host the database). The set of nodes can be manipulated by exploiting the normal churn in the set of participating nodes or by carrying out a denial of service (DoS) attack to speed up the change. We show how a Sybil attack [6] can be used as an alternative approach to control the `netDB`.

By leveraging control over the network database, we demonstrate how an Eclipse [7, 8] attack can be launch. This results in services being unavailable or peers getting disconnected from the network.

Finally, our deanonymization attack exploits the protocol used by peers to verify the successful storage of their peer information in the `netDB`. The storage and verification steps are done through two independent connections that can be linked based on timing. Using the information gathered by linking these two interactions, an attacker can determine (with high probability) which tunnel endpoints belong to specific participants (nodes) in the I2P network, and, therefore, deanonymize the participant.

Experimental results were gathered by tests performed both on our test network and on the real I2P network (against our victim nodes running the unmodified I2P software; no service disruption was caused to the actual users of the network).

In summary, the main contributions in this paper are the following:

1. A novel deanonymization attack against I2P, based on storage verification
2. Complete experimental evaluation of this attack in the real I2P network
3. Suggestions on how to improve the I2P to make it more robust

2 I2P Overview

In this section, we will describe key concepts of I2P, as well as how well-known attacks have been taken into account when designing its network infrastructure and protocols. I2P is an application framework (or middleware layer) built around the so-called I2P `router`. The `router` is a software component that runs on a host and provides connectivity for local I2P applications. An application can either accesses darknet services (as client), or it can host a service (as server).

Connectivity between applications is implemented via a fully decentralized peer-to-peer network, which runs as an overlay on top of IP. Applications can either use a TCP-like protocol called `NTCP` or a UDP-like protocol called `SSU`. The `router` maps these connections to packet-based *I2P tunnels*. These I2P tunnels provide anonymity using standard onion routing (similar to the well-known approach used by the Tor network). Tunnels are identified by the outermost peer in the chain and a unique `tunnelID` (these elements are roughly analog to the IP-address and port pair used in the clearnet).

Example applications include websites (called `eepsites` in the I2P community) and file sharing services, which together account for at least 30 % of I2P services [9], as well as email and chat systems. In February 2013, there were about 20,000 users in the I2P network at any given point in time; up from around 14,000 at the beginning of 2012.

2.1 Tunnels and Tunnel Pools

I2P uses paired, unidirectional tunnels handling onion-encrypted packets. It uses two different types of tunnels: `Exploratory` tunnels are used for all database lookups. They typically have a length of two hops. `Client` tunnels in contrast are used for all data connections. These client tunnels are bound to a local application but are used to reach any service this application is accessing, or, in the case of a server application, for communication with several clients. They have a typical length of three nodes.

For each application, the I2P `router` keeps a pool of tunnel pairs. `Exploratory` tunnels for interactions with the `netDB` are shared among all users of a `router`. If a tunnel in the pool is about to expire or the tunnel is no longer useable (e.g., because one of the nodes in the tunnel is failing) the `router` creates a new tunnel and adds it to the pool. It is important to recall later that tunnels periodically expire every ten minutes, and hence, need to be refreshed frequently. This is done to prevent long-lived tunnels from becoming a threat to anonymity.

2.2 Router Info and Lease Set

The `netDB` keeps two types of records: Peer and service information. Peer information is stored in so-called `routerInfo` structures containing the information needed to reach a peer – its IP address and port – as well as its public keys. This information is needed also to cooperate in a tunnel with this peer. Peer information has no explicit period of validity, however during normal operation peers refresh their `routerInfo` by uploading it to the `netDB` every ten minutes. Participants invalidate them after a period of time depending on the number of peers they know, in order to make sure a reasonable number of peers are known locally at any point in time.

The `leaseSets` contain service information, more specifically the public keys for communicating with a service as well as the tunnel endpoints that can be contacted to reach the service. Since tunnels expire after ten minutes, old service information is useless after that period of time, and it expires together with the tunnels. Users have to re-fetch them from the `netDB` if they want to continue communicating with the service even if the same application-layer connection is used the whole time.

In order for I2P to provide anonymity, service information has to be unlinkable to the peer information. However, in this paper, we show a way to actually link these two pieces of information and, therefore, deanonymize I2P participants.

2.3 Network Database

Database records are stored in a Kademlia-style DHT [10] with some modifications to harden it against attacks. This modified database is called `floodfill` database and the participating nodes `floodfill` nodes.

To request a resource on vanilla Kademlia implementations, a client requests the desired key from the server node considered closest to the key. If the piece of data is located at the server node, it is returned to the client. Otherwise, the server uses its local knowledge of participating nodes and returns the server it considers nearest to the key. If the returned server is closer to the key than the one currently tried, the client continues the search at this server.

Since a malicious node at the right position relative to the key can prevent a successful lookup in standard Kademlia, I2P adds redundancy by storing each database record onto the eight closest nodes (instead of a single one). Additionally, clients do not give up when they reached the closest node they can find but continue until their query limit (currently eight lookups) is reached.

Both servers and records are mapped into a global keyspace by their cryptographic hash, which is what the notion of closeness is based upon.

The number of `floodfill` nodes is limited to only few well-connected members. This is done because the research by Mittal et al. [4] showed how longer lookup paths compromise anonymity. With only few nodes (around 3 % of total network size) acting as database servers and these being well connected, it is assumed that an I2P client already knows one of the nodes storing the information. This keeps the lookup path length to a minimum.

2.4 Floodfill Participation

`Floodfill` participation is designed to regulate the number of `floodfill` nodes in the network and keep them at a constant count.

There are two kinds of database servers, *manual* `floodfill` participants and *automatic* `floodfill` participants. The *manual* `floodfill` participants are configured by their operator to serve in the database. The *automatic* `floodfill` participants are I2P nodes using the default `floodfill` configuration and are therefore not configured to always or never participate. They consider acting as `floodfill` nodes if the maximum amount of `floodfill` nodes, which was at 300 during our attack and increased in later releases, is currently not reached. As no node has global knowledge about all participants and nodes therefore deciding on their local knowledge only, the actual count is a bit higher. This maximum amount of `floodfill` nodes does not affect *manual* `floodfill` nodes. Based on their performance characteristics, *automatic* nodes can decide to participate. They regularly re-evaluate their performance, and step down if they no longer meet the needed performance characteristics.

To estimate the proportion of *automatic* `floodfill` participants, we monitored the network database from the nodes under our control, and detected peers changing their participation status, which does not happen for *manual* `floodfill` participants but does happen for *automatic* ones. Results show that around 95 % of the database servers are *automatic*.

2.5 Example Interactions

Server applications register themselves on the local I2P `router` with their public key for data encryption. The `router` then allocates a tunnel pool for the server application and publishes the public key and all tunnel endpoints allocated to this application (service information) to the `netDB`. The fingerprint of the application's public key serves as key into the `netDB`. The `router` then keeps the service information up-to-date every time it replaces a tunnel. This key fingerprint remains the primary identifier to reach a service. A list of bookmarks called the address book is supplied with the I2P software and users can amend this list for themselves and share it with others.

If an application wants to access an I2P service, it first needs to locate the service. It asks the `router` for the service information. The `router` may have this service information stored locally (e.g., if it runs a `floodfill` node or the same information was already requested recently) and be able to return it to the application immediately. If the information is not available locally, the `router` sends a `lookupMessage` through one of the `exploratory` tunnels and returns the service information to the application, if it could be found on the `netDB`, or an error otherwise. The service lookup is thereby anonymized by the use of an exploratory tunnel. Otherwise, `floodfill` nodes would be able to link users to services, and avoiding such links is the main goal of anonymity networks. The application can then hand packets to the `router` and request them to be sent to the service through one of the `client` tunnels allocated to the application. If the `router` receives any packets through one of the client tunnels allocated to an application, it forwards them appropriately.

2.6 Threat Model

The I2P project has no explicit threat model specified but rather talks about common attacks and existing defenses against them[3]. Overall, the design of I2P is motivated by threats similar to those addressed by Tor: The attacker can observe traffic locally but not all traffic flowing through the network and integrity of all cryptographic primitives is assumed. Furthermore, an attacker is only allowed to control a limited amount of peers in the network (the website talks about not more than 20 % of nodes participating in the `netDB` and a similar fraction of total amount of nodes controlled by the malicious entity). In this paper, we present an attack that requires fewer malicious nodes while still deanonymization users. This threat model is also used by Hermann et al. [5], putting our result in some context.

2.7 Sybil Attacks

One well-known attack on anonymity systems is the so-called Sybil attack [6], where a malicious user creates multiple identities to increase control over the system. However, I2P has some defense mechanisms aimed at minimizing the risk of Sybil attacks.

It is possible to control more identities in the network by running multiple I2P instances on the same hardware. However, participants evaluate the performance of peers they know of and weight them when selecting peers to interact with instead of using a random sample. As running multiple identities on the same host decreases the performance of each of those instances, the number of additional identities running in parallel is effectively limited by the need to provide each of them with enough resources for being considered as peers.

[3] `http://i2p2.de/how_threatmodel.html`

Additionally, the mapping from `leaseSets` and `routerInfos` to `netDB` keys, which determines the `floodfill` nodes responsible for storing the data, includes the current date so the keyspace changes every day at midnight UTC. Nodes clustered at a certain point in the keyspace on one day will, therefore, be distributed randomly on any other day. However, this change does not include any random inputs, and is thus completely predictable.

2.8 Eclipse Attacks

With a vanilla Kademlia DHT, all requests would be answered by the node nearest to the searched key. If this node is malicious and claims not to know the key and not to know any other database server nearer to the key, the lookup will fail [8]. To circumvent this attack, I2P stores the key on the eight nodes closest to the key and a requesting node will continue asking nodes further away from the key if they no longer know any candidate nearer to the searched key.

3 The Attacks

The final goal of our attacks is to identify peers using a particular service on I2P and their individual usage patterns, including when and for how long they use this service. We describe different ways to gain the necessary control on the `netDB` and include a brief discussion of how to perform a classical Eclipse attack where access to a service inside the I2P network is blocked by the attacker. Our attack uses a group of 20 conspiring nodes (fully controlled by us) that are actively participating in the network and that act as `floodfill` peers. The description of our attacks is structured as follows:

a) We take control over the floodfill database. We either forcible remove all other nodes and take full control (Section 3.1), or use a Sybil attack (Section 3.2) to take control over a region of the database
b) Leveraging this control of the database, we implement an Eclipse attack (Section 3.3)
c) Alternatively, we exploit our control to link store and verification connections that done by peers who update their `routerInfos`, hence deanonymizing these peers (Section 3.4)

3.1 Floodfill Takeover

In this section, we describe an attack that can be used to control the majority of database nodes in the I2P network. By taking control of the `netDB`, one can log database actions for the full keyspace. The attack is possible with relatively few resources (only 2 % of total nodes in the network are needed). Note that the threat model limits an attacker to 20 % of `floodfill` nodes. This assumption is violated by this attack. Nonetheless, the I2P developers still consider this a serious and valid attack.

The attacker can configure his nodes as *manual* `floodfill` nodes to make sure his nodes participate in the database. In the remaining part of this section, we discuss how the number of legitimate `floodfill` nodes can be decreased, facilitating takeover of the network database.

Around 95 % of the `floodfill` nodes are *automatic*, that is, they participate due to the need for more database nodes and the availability of resources on their side. While there will not be the need for more participants once the attacker has set up his nodes, all current participants continue to serve as `floodfill` nodes as long as they do not get restarted and continue to have enough resources.

Available resources are both measured in terms of available data rate, which is statically configured for each node by the admin, and job lag, which is measured during operation taking the average delay between the scheduled time where each task (e.g., tunnel building, database lookups) is supposed to run and the actual point in time when it is started. As this delay largely depends on the number of open tasks, and an attacker can cause additional tasks to be scheduled, this job lag is a good target for attack.

As load varies and `routers` tend to be rebooted from time to time, the least noisy and easy-to-deploy possibility is waiting for the number of legitimate `floodfill` participants to decrease while the attacker adds malicious nodes to the network. This is especially effective every time an update to the I2P software is distributed, as updating I2P includes a restart of the `router`.

However, to speed up churn in the `floodfill` set, an attacker can influence the job lag using a denial-of-service (DoS) attack against a legitimate `floodfill` participant. The attacker creates many new tunnels through the attacked node adding a tunnel build job for each. When specifying a non-existing identity for the node after the victim in the tunnel, it also adds a total of eight search jobs looking for the peer information to the victim's job queue. If the attacker is able to create more open jobs than the node can handle, these jobs get started late, building up a job lag. The attacker needs to be careful to not actually send large amounts of data through the attacked node as this would trigger the data rate limiting functionality and make the victim drop tunnel requests instead of adding them to the job queue. As soon as the attacked node drops its `floodfill` flag, the attacker continues with the next active `floodfill` node. It is important to note that an attacker only needs capacity to launch a DoS attack on a single legitimate `floodfill` node at a time. Nodes will only regain `floodfill` status if there are too few active `floodfill` nodes in the network. In the attack scenario, however, the attacker inserted his own nodes in the network, replacing the failing, legitimate ones.

3.2 Sybil Attack

Under certain conditions, the `floodfill` takeover described in the previous section is not optimal. The Eclipse attack described in the next section requires several `floodfill` nodes closest to a keyspace location, while there are still legitimate `floodfill` nodes at random places in the keyspace after a successful `floodfill` takeover. Additionally, the takeover attack requires over 300 active malicious nodes in the network.

A Sybil attack will allow the attacker to get close control over a limited part of the keyspace, and it requires fewer resources than the complete takeover. While an attacker cannot run (too many) I2P nodes in parallel due to the peer profiling that is in place, it is possible to compute huge quantities of identities offline and then use the best placed ones (the ones closest to the victim in the keyspace). To exhaust the query limit with negative responses, a total of eight nodes near the target key are necessary (near means closer than any legitimate participant in this region of the `netDB`). To log lookups, a single attacker would suffice. As there are currently only 320 `floodfill` nodes active, a set of 10,000 identities, which can be computed in few minutes, already gives the attacker many possible identities to completely control any position in the keyspace.

Introducing a new node into the network requires a setup time of about an hour, during which the node gets known by more and more of its peers and actively used by them for lookup. Hence, it takes some time until the Sybil attack reaches the maximal impact. In addition, as mentioned previously, the storage location of the keys that the attacker is interested in (e.g., the key at which the service information, that should be eclipsed, is stored) changes every day at midnight. This requires attacking nodes to change their location in the keyspace, opening a window during which legitimate nodes control the position in question. However, as the rotation is known in advance, a second set of attack nodes can be placed at the right spot before midnight, so they are already integrated once the keyspace shifts. As a result, this keyspace rotation does not prevent our attack but only requires few additional resources.

3.3 Eclipse Attack

Our Eclipse attack allows an attacker to make any database record unavailable to network participants. It is an example of how Sybil attacks can be used against the network, independent from the deanonymization described in the next section. As clients use up to eight `floodfill` nodes to locate a key in the network database, the attacker needs to control at least the eight nodes closest to the key. The list of other close servers piggybacked on a negative lookup answer is used to increase the probability of the client knowing all `floodfill` participants controlled by the attacker.

Once control over a region in the keyspace is established, the attacker can block access to items in this region by sending a reply claiming to not know the resource. If the blocked resource contains service information, this effectively prevents anyone from accessing the service. Similarly, if peer information is blocked, network participants are unable to interact with it.

3.4 Deanonymization of Users

Finally, we show an attack allowing an attacker to link any user with his IP address to the services he uses. For this attack, we use the Sybil attack described earlier to place malicious nodes in the netDB so they can observe events in the network related to each other. We later use information from these events to deanonymize users.

Nodes store their database records on the closest floodfill node that they are aware of. To verify proper storage of a database record, a node subsequently sends a lookup to another floodfill node nearby. This is done after waiting for 20 seconds. If both nodes, the one stored on and the one handling the verifying lookup, are controlled by the same (malicious) entity, the attacker can observe both interactions and connect them (with some probability).

Storage of peer information is done without a tunnel. That is, it is done in the clear, as the client is exposed by the content of the database entry anyways. Storage verification, on the other hand, is done through an exploratory tunnels to make it more difficult to distinguish storage verification from normal lookup (if floodfill nodes could distinguish verifications from normal lookups, they could allow verification and still hide the stored information from normal lookups). As a result, the first part of this interaction exposes the client node, while the second part exposes an exploratory tunnel endpoint. This combination allows us to create a probabilistic mapping between exploratory tunnel endpoints and the peers owning the tunnel.

If an attacker can later link actions to an exploratory tunnel endpoint, she can use this probabilistic mapping to identify the client initiating this action, effectively deanonymizing this client. Exploratory tunnels are used for all regular database lookups, including those for service information. A floodfill node controlled by the attacker will therefore see the exploratory tunnel endpoints for all lookups for services that this node handles. Thus, if the attacker places malicious floodfill nodes at the right positions to observe the lookups for interesting services, he can combine the probabilistic mapping with the service lookups.

The attack process is shown in Figure 1: The client (victim) stores its peer information on Node 7 in the netDB. This node then pushes the peer information to other floodfill nodes that are close in the netDB. In this case, these close nodes are Node 6, Node 8 and Node 9. After 20 seconds, the client starts the verification process and requests its own peer information from Node 6, using one of its exploratory tunnel pairs. Later, it requests the service information for an eepsite from Node 4, using the same exploratory tunnel! If the attacker controls Nodes 4, 6 and 7, he can (i) leverage the store and verification operation (on Node 6 and 7) to map the victim's tunnel identifier to the actual victim node, and (ii) see the victim requesting the service (on Node 4).

As service information expires after ten minutes, each client needs to fetch it before starting an interaction with a service and update it regularly during the interaction. This allows the attacker to identify which of the observed clients interact with each of the monitored resources and when she does so. The regular update of service information additionally reveals how long the service has been used. As a result, the attacker is able to deanonymize users with respect to their usage of certain services.

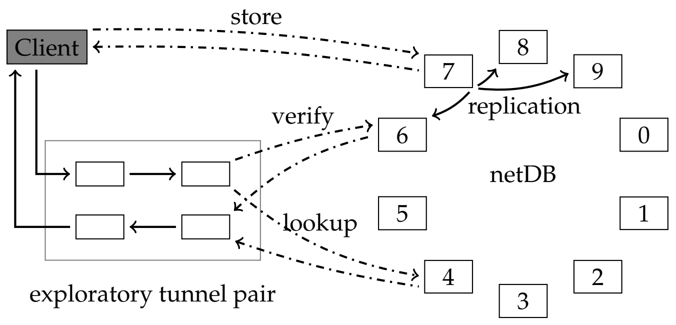


Fig. 1. Deanonymizing attack

4 Evaluation

In this section, we describe our experiments confirming the attacks described in the previous section. We have made sure to not disrupt any participant in the I2P network apart from our own nodes and no identifying information has been collected about other participants in the network. For testing the DoS attack, which we describe first, a special, separated test network was created to prevent any harm on the real network. All other attacks were tested in the real I2P network.

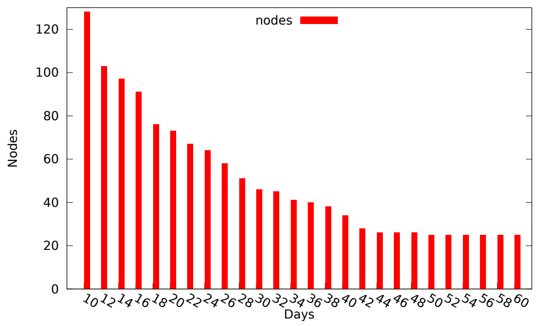
4.1 Floodfill Takeover

We discuss the impact of a takeover attack and the time needed for a passive takeover where the attacker only waits for automatic `floodfill` nodes to resign due to normal fluctuations in the network.

The fraction of automatic `floodfill` nodes in the network was determined by monitoring the local peer storage on the `routers` under our control. These `routers` participated as `floodfill` nodes in the real I2P network, and logged whenever a node removed or added the `floodfill` flag to its peer information. Automatic `floodfill` nodes add the `floodfill` status only after being online for at least two hours and can lose and regain `floodfill` status depending on network load. Manual `floodfill` nodes, instead, will always have the floodfill flag set. Over a period of ten days, we saw a total of 597 `floodfill` nodes and an average of 413 `floodfill` nodes each day. During these days, only 128 of them did not change their floodfill status. Therefore, a passive `floodfill` takeover attempt lasting for ten days would leave 128 legitimate nodes in place while adding 258 malicious nodes.

As seen in Figure 2, the amount of `floodfill` nodes never losing `floodfill` status decreases almost linearly by five nodes every day, until it reaches 26 nodes after 44 days. From there on, the count remains stable, and after 60 days, still 25 nodes are left. These are likely to be manual `floodfill` nodes, which would also not have resigned in a DoS attack.

Fig. 2. Legitimate floodfill nodes after *n* days



As the active `floodfill` takeover uses a DoS attack on target nodes, we decided to test this attack on a closed local network. The test network consisted of 100 nodes split into five groups: 30 slower users with default data rate configuration (96kB/s down- and 40kB/s upload), 30 faster users configured to use up to 200kB of data rate in both directions, 20 automatic `floodfill` nodes, and 5 manual `floodfill` nodes, as well as 15 attackers. To simulate a large-enough number of `floodfill` nodes, a larger fraction of peers were configured as `floodfill` nodes, and the maximum number of active `floodfill` nodes was lowered from 300 to 20. In this setup, a group of five attacking nodes was able to slow down the attacked nodes enough for them to give up `floodfill` status.

4.2 Experimental Setup

In this section, we describe the setup used for all the following attacks. All of these attacks have been successfully tested on the real I2P network. All nodes being attacked were controlled by us.

We ran 20 attacking nodes connected to the normal I2P network. These nodes acted as `floodfill` peers. Six additional nodes served as legitimate peers, and were used to verify the attacks. All attackers were set up on a single VM host in the US and configured to use 128kB/s of download and 64kB/s of upload data rate. The legitimate nodes were split evenly between the VM host in the US and a second VM host in Europe (to make sure the results do not rely on proximity between attackers and victims). Attackers were configured to act as manual `floodfill` nodes and had additional code added, which logged network events and allowed for the blacklisting of specific information, as required by the Eclipse attack.

During our experiments, the I2P statistics[4] reported between 18,000 and 28,000 nodes and 320 to 350 `floodfill` nodes, fluctuations during the day.

[4] `http://stats.i2p.in`

Therefore, we were controlling less than 7 % of `floodfill` nodes and a negligible part of total nodes.

4.3 Sybil Attack

To test our Sybil attacks, we created a set of 50,000 precomputed `router` identities. Each identity consists of one signing and one encryption key (as well as a certificate, which is unused). Computing this set of identities took less than 30 minutes on a twelve-core Xeon server. We then made this set of identities available to all our I2P nodes for the following experiments.

Additionally, we modified the `router` software to enable our attacking nodes to change their identity to any of the precomputed ones on demand, as well as to enable a group of attackers to use a set of identities, one per node, close to a target.

4.4 Eclipse Attack

To evaluate the Eclipse attack, we configured our victims to download a test `eepsite` every minute, and log the results. Ten attack nodes were moved to the storage location of the service information for the test `eepsite`. The attackers were configured to give negative response to all lookups for the test `eepsite` and only refer to each other in these negative responses such that the victims would learn about all malicious `floodfill` nodes as fast as possible. A second group of ten attack nodes was moved to the test `eepsite`'s storage location for the following day, and was configured to keep the service information unavailable across the keyspace shift.

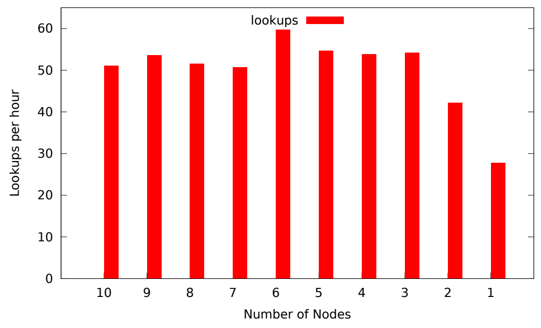
We ran the Eclipse attack over a period of 42 hours. During this time, victims were on average able to reach the blocked `eepsite` for a total of five minutes. Three out of six nodes were not able to reach the `eepsite` at any point in time, and the most successful victim was able to interact with the destination for a total of only 16 minutes during that period. When the second set of attackers was not used, all victims could successfully reach the `eepsite` during a 15-minute window around midnight (when the keyspace rotation happens).

4.5 Deanonymization of Users

In the next step, we ran an experiment that simulates our ability to deanonymize a particular victim user Alice, who is accessing a specific resource R of interest. This resource could be a dissident's web page or a sensitive file. The idea is that the attacker knows that resource and tries to determine whether a user under suspicion actually accesses R.

For the simulated attack, we first configured ten malicious nodes and set them up as `floodfill` nodes in the keyspace region occupied by our six victim nodes. We then configured these six victim nodes to repeatedly query our test `eepsite`. In a first step, we wanted to understand how many service lookups

Fig. 3. Logged service lookups per hour



could be observed by the malicious `floodfill` nodes. In particular, we checked for an increasing number of malicious nodes (from 1 to all 10), the number of lookups from the victim machines that we could observe. We ran this experiment for a total of eight hours for each number of nodes, during different parts of the day. This was done to avoid that the different number of `routers` at different times during the day would influence the results.

The experiments (Figure 3) show a roughly constant amount of around 50 lookups logged every hour, until fewer than three malicious nodes are left. More precisely, there was a lookup observed from all our victim nodes approximately every nine to ten minutes, which was caused by the lifespan of service information. Under optimal conditions, one would expect 36 to 40 lookups per hour, which is the total for six hosts updating their local information every nine to ten minutes. However, shortly after the service information expired, there were more than six lookups due to nodes retrying their lookup after losing the response, adding up to the total of around 50 lookups. This means that the attacker needs only three malicious nodes in the vicinity of the victim nodes to observe all their relevant lookups.

In the next step, we tried to understand how many lookups observed at the malicious nodes could be properly attributed to the queries made by the victims. Observing lookups, of course, is not enough. It is also necessary to attribute different lookups (and tunnel endpoints) to the victim machines. Otherwise, we cannot determine whether a victim has requested a particular service. Since the network is not only used by the victims, the malicious nodes receive unrelated lookups by other (random) nodes in the I2P network.

The results were similar for the sites both in Europe and the US: 52% of the tunnel endpoints that we attributed to a victim user were indeed originating from this user (call her Alice), while in 48% of the cases, a specific lookup (and thus, tunnel endpoint) that we attributed to Alice actually belonged to a different, random user. That is, in this step, we only correctly identify about half the tunnel endpoints. However, this does **not** imply that we can detect Alice only half the time, or that the results are only slightly better than a coin toss.

Instead, it means that we can detect a single access that Alice performs for resource R half the time. Monitoring Alice's accesses over a longer period of time then allows us to mount a much stronger attack, as discussed below.

Assume that we monitor Alice and a resource R for a certain time period T. Let's partition this period into N time slots of duration d, where $d = 10$ minutes. This is the time interval after which I2P refreshes the tunnel identifiers, and hence, a new lookup is performed. During each of the $i : 0 <= i < N$ time slots, we see a list L_i of all tunnel identifiers that access resource R. Moreover, we learn one tunnel identifier t_i that we *believe* belongs to Alice (but we could be wrong, since we are right only half the time). We call this probability u, and, as discussed above, we empirically found $u = 0.52$. We then check whether $t_i \in L_i$. If this is true, we have a hit. If not, we have a miss for time slot i. If we could always attribute each lookup (and tunnel endpoint) correctly to the corresponding user, a single hit would be enough. Unfortunately, $u < 1.0$, and hence, we require to monitor for multiple time slots.

Assume further that we observe k hits over the time period T, we want to determine the probability that Alice has indeed accessed R. We need to assume certain parameters to compute this probability (and ultimately, to determine a suitable threshold for k for deanonymization). In particular, we need to assume the fraction of time slots in N where Alice accesses R (we call this fraction p). Intuitively, if Alice accesses R often, our task will be easier. Moreover, we need to know the probability q that any other, random node accesses R. When q p, then Alice behaves similar to any random node, and we cannot meaningfully distinguish her accesses from other nodes. Hence, we require that $p > q$; intuitively, as p grows larger than q, our task becomes easier.

The probability that we have k hits over N time slots can be computed with the binomial distribution. Recall that a hit occurs when we attribute a certain lookup (tunnel id) with Alice, and we see this tunnel identifier accessing R.

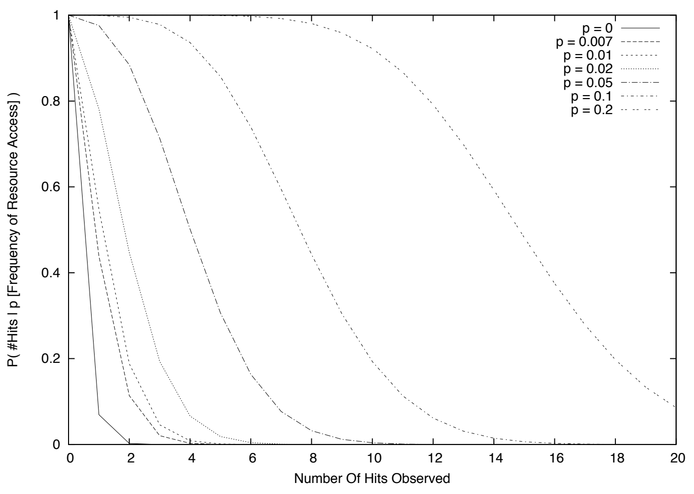
The probability that $t_i \in L_i = x = u * p + (1 - u) * q = 0.5p + 0.5q$. This is the chance of Alice accessing resource, in case we guessed correctly, plus the chance of a random hit when we misidentified the tunnel. Thus:

$$P(k\ hits) = \binom{N}{k} x^k * (1 - x)^{N-k} \qquad (1)$$

Since we care about the probability of *at least k hits*, we require the cumulative distribution function. In Figure 4, one can see the probability (shown y-axis) that one observe at least k hits (shown on the x-axis) for different values of p (the probability that Alice accesses R during an arbitrary time slot). For this graph, we assume the length of the observation period to be one day ($N = 144$), and we set $q = 0.001$.

The value of q is relevant for false positives, and has been chosen conservatively here. Our concrete values assumes that about 7% of all nodes access R once a day. The false positives (incorrect attributions) are represented by the solid line for $p = 0$; that is, Alice does not at all visit R. It can be seen that this line quickly drops close to zero. When we require at least two hits per day, the

Fig. 4. Probability of k or more hits, depending on p



chance for a false positive is about 2.4%. For less frequently-accessed resources, this value drops quickly (0.003% for two or more hits, 0.7% for a single hit for $q = 0.0001$).

When we require three hits per day, Figure 4 shows that we would detect Alice with more than 80% probability when she accesses the site with $p = 0.05$. This translates to about 7 visits per day. In case Alice visits the site only one time ($p = 0.007$), we would need to lower the threshold k to 1. In this worst case, we would have 52% chance of detection (exactly the probability to get the correct tunnel), and we would risk about 7% false positives.

Overall, when Alice visits a certain resources a few times per day, and this resource is not very popular, our approach has a very high probability to correctly deanonymize Alice. As expected, when a resource is popular in the network and Alice's visits become more infrequent, our system becomes less accurate and more prone to false positives.

5 Discussion

5.1 Limitations

For a successful deanonymization of a client's lookups, the attacker needs to have his `floodfill` nodes both next to the client's peer info storage position and the service information's storage position in the `netDB`. Therefore, a Sybil attack requires the attacker to limit himself to a small number of services and peers. However, as there are just three malicious `floodfill` nodes required for each monitored service, and the number of darknet services interesting to the attacker is likely to be small, tracking specific user is not a problem. As many clients map to the same region in the keyspace and, therefore, store their peer information to the same set of `floodfill` nodes, it is also possible to track all these users without additional resources. However, as the mapping to the keyspace is essentially random, the attacker cannot select an arbitrary group of clients, but only clients close together in the keyspace.

5.2 Potential Attack Improvements

The experiments have all been run with relatively few nodes configured with limited data rates. It should be easy to set a higher limit on data rates, which will make the nodes better known throughout the network, and, therefore, improve the results of the attacks. In order to deal with the increased number of interactions, one needs to either improve performance of the attack code or assign more processing power to the attack nodes.

Instead of blocking lookups, an Eclipse attack could also block the store operation. An approach similar to the one used for the deanonymization attack can be used to make the storing node believe that the storage was successful, while it was actually blocked: More precisely, the attacking `floodfill` nodes can identify the victim's verification step, and only signal successful lookup for this verification, while replying with a negative response to all regular lookups.

5.3 Experiments in the I2P Network

After running our nodes for three weeks in the I2P network, developers noticed our group of 20 `floodfill` nodes that were connecting with consecutive IP addresses and had cloned configuration. These were changing their identity together at midnight each day, and were suspiciously close to each other in the keyspace. Using the notes already prepared for discussing our results with the I2P development community, we used this opportunity to start the interaction following a responsible disclosure strategy. This discussion resulted in some improvements made to I2P, which we will discuss in Section 5.4 and 5.5 below.

5.4 Implemented Improvements

After sharing our results with the I2P developers, first improvements were implemented to make our attacks more difficult. The limit of `floodfill` nodes was raised from 300 to 500, requiring an attacker to run almost twice as many malicious nodes to take control over the full network database and reducing the fraction of the keyspace controlled by a single node. Additionally, the number of tunnels built with the same previous node in the chain was limited, so that the attacker has to route tunnel build requests through an additional hop. Therefore, the attacker has to add an additional encryption layer to the tunnel initiation packets, requiring expensive public key cryptography. However, as an attacker already needs 500 malicious nodes to replace legitimate `floodfill` nodes, and our experiments showed that we were able to run the DoS attack with only five malicious nodes, it is save to assume, that the attacker has the necessary resources for this additional encryption.

Finally, only one `floodfill` node per /16 subnet is considered now for database lookups, requiring an attacker to spread nodes over several networks in order to successfully execute an Eclipse attack. However, several legitimate `floodfill` nodes in the same /16 subnetwork are unlikely to also serve the same part of the network database, so only malicious nodes are affected by this change. As our attacks require at most ten `floodfill` nodes in the same region, the attacker can work around this limitation by using several cloud services.

I2P developers also started to discuss replacing the Kademlia implementation of the network database with R^5N [11] used by gnunet, which is designed to deal with malicious peers. This will allow I2P to profit from current research in this area.

5.5 Suggested Improvements

While the desire to have slow nodes not participate in the `floodfill` database is understandable, this is giving an attacker the possibility to permanently remove legitimate nodes from the database using a DoS attack. If nodes that once had `floodfill` status will return independent of the current number of active `floodfill` nodes, an attacker needs to constantly DoS the legitimate participants to keep them out of the database. Additionally, this should not increase the number of `floodfill` nodes beyond a constant number, as once a certain number of `floodfill` nodes is reached there will always be a large enough fraction of them online to reach the limit of `floodfill` nodes, and no new volunteers will join even under high load or attack.

Alternatively, the hard-coded number of active `floodfill` nodes could be removed completely, and the count of `floodfill` nodes could be solely regulated by the suitability metric, which would also prevent an attacker from permanently removing legitimate nodes. After discussing the issues with I2P developers, they confirmed that this is the direction I2P is taking.

To counter Sybil attacks, a client node could only start to trust a `floodfill` node after seeing it participate for n days in the network. This would increase the cost for multi-day attacks, as the attacker needs to have $n + 1$ attack groups active at the same time. This adds a multi-day setup time during which his intentions could be discovered, and potential victims could be warned using the newsfeed of the I2P client software. Since we have observed 600 distinct `floodfill` nodes over the period of ten days, it should be safe to assume that enough `floodfill` candidates exist in the network, even after adding this additional restriction. However keeping track of clients active in the past creates problems on the client, if he is just bootstrapping and does not have any knowledge of the past. This is also problematic for a client that has been offline for several days. In addition, keeping track of known identities for a larger time-frame requires storing and accessing the information effectively.

An alteration of this idea is currently being discussed by the I2P developers: If the modification used for keyspace rotation is not predictable, requiring identities to be known in the network for one day is enough. Since it will be hard to build consensus on such an unpredictable modification in a fully distributed manner, one could observe daily external events that are hard to predict, such as the least significant digits of stock exchange indices at the end of each day. The problem with this approach will be finding a way to automatically collect this information in a censorship-resilient and reliable way.

Storage verification does not work against a group of malicious nodes. The randomization of the delay between storage and verification introduced in I2P as a reaction to our research will make correlation less certain but still allows an attacker to reduce anonymity. One way around this would be to use direct connections also for the verifying lookup. By doing this, problems on legitimate nodes and attacks carried out by a single malicious `floodfill` node could still be detected, while no information about exploratory tunnels would be leaked. Also, if the redundant storing is done by the client, no verification is needed.

6 Related Work

Distributed anonymity systems, as well as I2P specifically, have been discussed in previous work. Tran et al. [3] described common failures of DHT-based anonymity schemes and Mittal et al. [4] later provided a proof on the trade-off between passive information-leak attacks and verifiability of the data. I2P was built with this limitation in mind. In particular, I2P limits the number of database nodes to a small fraction of the network and selects peers for tunnel building from a local pool rather than random walks in the `netDB`, discussed in detail and attacked by Herrmann et al. [5], to counter these problems. With only few nodes participating in the DHT, it is a reasonable assumption that all nodes in the I2P network know the right node for every DHT lookup already, and, therefore, no attacks on lookup capture due to increased path lengths are possible. We have shown that I2P is still vulnerable to database-based attacks, and focused on store events, as opposed to blocking certain lookups. Wolchok et al. [12] used Sybil nodes with changing identities, which enabled them to crawl DHTs faster. Similar identity changing was utilized by our work to counter the daily keyspace rotation and may also be used to cover larger parts of the `NetDB` for deanonymization.

Herrmann et al. [5] showed a way to identify peers hosting I2P services exploiting the peer-profiling algorithm to influence the set of nodes the victim interacts with. In contrast, our identification shows the actions that a specific user (victim) performs in the network. Also, while they showed the individual steps needed to deanonymize users, the complete attack was evaluated only with victim nodes patched to only consider their attackers as tunnel participants.

7 Conclusions

In this paper, we presented attacks that can be combined to deanonymize I2P users. This confirms that critical attacks (such as Sybil and Eclipse attacks) against DHTs used for anonymity systems are still valid, even when these systems are designed to resist these threats for practical purpose.

Acknowledgements

This work was supported in part by the ARO under grant W911NF-09-1-0553 and Secure Business Austria.

References

1. Dingledine, R., Mathewson, N., Syverson, P.: Tor: the second-generation onion router. In: Proceedings of the 13th conference on USENIX Security Symposium - Volume 13. SSYM'04, Berkeley, CA, USA, USENIX Association (2004) 21–21
2. Dingledine, R., Mathewson, N., Murdoch, S., Syverson, P.: Tor: the second-generation onion router 2012 draft. (2012)
3. Tran, A., Hopper, N., Kim, Y.: Hashing it out in public: common failure modes of DHT-based anonymity schemes. In: Proceedings of the 8th ACM workshop on Privacy in the electronic society. WPES '09, New York, NY, USA, ACM (2009) 71–80
4. Mittal, P., Borisov, N.: Information leaks in structured peer-to-peer anonymous communication systems. ACM Trans. Inf. Syst. Secur. **15**(1) (March 2012) 5:1–5:28
5. Herrmann, M., Grothoff, C.: Privacy-implications of performance-based peer selection by onion-routers: a real-world case study using I2P. In: Proceedings of the 11th international conference on Privacy enhancing technologies. PETS'11, Berlin, Heidelberg, Springer-Verlag (2011) 155–174
6. Douceur, J.: The sybil attack. In Druschel, P., Kaashoek, F., Rowstron, A., eds.: Peer-to-Peer Systems. Volume 2429 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2002) 251–260
7. Castro, M., Druschel, P., Ganesh, A., Rowstron, A., Wallach, D.S.: Secure routing for structured peer-to-peer overlay networks. SIGOPS Oper. Syst. Rev. **36**(SI) (December 2002) 299–314
8. Singh, A., wan Ngan, T., Druschel, P., Wallach, D.S.: Eclipse attacks on overlay networks: Threats and defenses. In: In IEEE INFOCOM. (2006)
9. Timpanaro, J.P., Chrisment, I., Festor, O.: Monitoring the I2P network
10. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the xor metric. In Druschel, P., Kaashoek, F., Rowstron, A., eds.: Peer-to-Peer Systems. Volume 2429 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2002) 53–65
11. Evans, N., Grothoff, C.: R5n: Randomized recursive routing for restricted-route networks. In: Network and System Security (NSS), 2011 5th International Conference on. (sept. 2011) 316 –321
12. Wolchok, S., Hofmann, O.S., Heninger, N., Felten, E.W., Halderman, J.A., Rossbach, C.J., Waters, B., Witchel, E.: Defeating Vanish with low-cost Sybil attacks against large DHTs. In: Proc. of NDSS. (2010)