

Practical Attacks Against The I2P Network

Christoph Egger

Anonymity networks, such as Tor or I2P, were built to allow users to access network resources (e.g., to publicly express their opinion) without revealing their identity. Newer designs, like I2P, run in a completely decentralized way, 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 thesis, 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 any I2P user with high probability.

Anonymitätsnetzwerke, wie zum Beispiel Tor oder I2P, wurden entwickelt, um Nutzern den anonymen Zugriff auf Informationen im Netzwerk zu ermöglichen, ohne dabei ihre Identität preiszugeben. Neuartige Entwürfe für derartige Netzwerke, wie zum Beispiel I2P, arbeiten dabei vollständig dezentral, während ältere Systeme, wie z. B. Tor, auf eine zentrale Autorität aufbauen. Der dezentrale Aufbau hat Vorteile (keine vertrauenswürdige, zentrale Autorität, bessere Skalierbarkeit), allerdings ermöglicht er durch die Verwendung von verteilten Streutabellen auch neuartige Angriffsszenarien.

I2P wurde unter Berücksichtigung dieser Probleme entworfen und wird als sicher in Bezug auf praktische Angriffe erachtet. Leider ist dieses Vertrauen nicht vollständig gerechtfertigt. In dieser Arbeit zeigen wir eine Sammlung von Angriffen, die verwendet werden können, um I2P Nutzer zu identifizieren. Konkret zeigen wir, dass ein Angreifer mit vergleichsweise geringen Mitteln in der Lage ist, I2P Nutzer mit hoher Sicherheit zu identifizieren.

Contents

1	Introduction	5
2	I2P Overview	8
2.1	Tunnels and Tunnel Pools	8
2.2	Router Info and Lease Set	9
2.3	Network Database	9
2.4	Floodfill Participation	10
2.5	Example Interactions	10
2.6	Threat Model	11
2.7	Sybil Attacks	11
2.8	Eclipse Attacks	12
3	The Attacks	13
3.1	Floodfill Takeover	13
3.2	Sybil Attack	14
3.3	Eclipse Attack	15
3.4	Deanonymisation of Users	15
4	Implementation and Setup	17
4.1	Node Setup	17
4.2	Inter-Node Communication	18
4.3	Distributed Correlation	19
5	Evaluation	20
5.1	Floodfill Takeover	20
5.2	Experimental Setup	21
5.3	Sybil attack	22
5.4	Eclipse attack	22
5.5	Deanonymisation of Users	22
6	Limitations	26
7	Discussion	27
7.1	Potential Improvements	27
7.2	Discovery of the attack	27
7.3	Implemented Improvements	28
7.4	Suggested Improvements	28
8	Related Work	30

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 in 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 [2,3] 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. Specifically, the clients pick nodes from this list to create encrypted tunnels, until they reach so-called 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. Finally, Tor is typically used to anonymously access services on the public Internet. As most communication on the Internet is unencrypted, this exposes the actual content of the interaction to the exit node. It also exposes the services in the clearnet to threats coming from

the anonymity network, where finding the responsible entity (e.g., for legal response) is next to impossible.

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.

I2P uses the public Internet for transporting encrypted data between darknet nodes, but no connections into or out of the darknet are provided. Of course, as I2P provides general-purpose data connections, it is possible for individuals to run proxy servers. This allows I2P users to reach the clearnet anonymously, and for people outside to access anonymized resources inside I2P. However, proxies are not considered critical as users generally stay within the network. Also, as there are only few of these proxies, they are easy to block by service providers who do not want anonymous interactions.

The use of DHTs in peer-to-peer anonymity systems has been successfully attacked in the past [11]. Continued research on this problem finally led to general results [8] 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 [6].

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 [11] and [8], 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 analysis approach was updated to make it more difficult for an attacker to influence it in an exploitable way. As a result, I2P is considered secure in practice. Unfortunately, this is not entirely justified.

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

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 [4] can be used as an alternative approach to control the `NetDB`.

By leveraging control over the network database, we demonstrate how an eclipse [1] [9] 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 thesis 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 against this class of attacks

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 access darknet services (as a client), or it can host a service (as a 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 [10], 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. The administrator of each node, however, can configure the length for each type and each direction of tunnel independently, and even add some randomness to the number. However, there is an upper limit of eight hops.

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. Tunnels are built using `tunnelBuildMessages`. They contain, for each node in the tunnel, a session key, the `tunnelID` on which the node should listen for packets, and both name and `tunnelID` for the next node in the chain. The `tunnelBuildMessage` is iteratively encrypted using the node's public key to only reveal the necessary information at each node and sent to the first node. Each node then decrypts the packet and removes its information passing the `tunnelBuildMessage` to the next node.

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 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 thesis, 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 [7] 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 on 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, which currently is eight lookups, is reached.

Both servers and records are mapped into the keyspace by their cryptographic hash on which the notion of closeness is based.

The number of `floodfill` nodes is limited to only few well-connected members. This is done because the research by Mittal et al. [8] 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 these *automatic* nodes can decide to participate. These *automatic* `floodfill` participants 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% database servers are of the *automatic* kind.

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.

Figure 2.1 shows a typical client interaction: 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

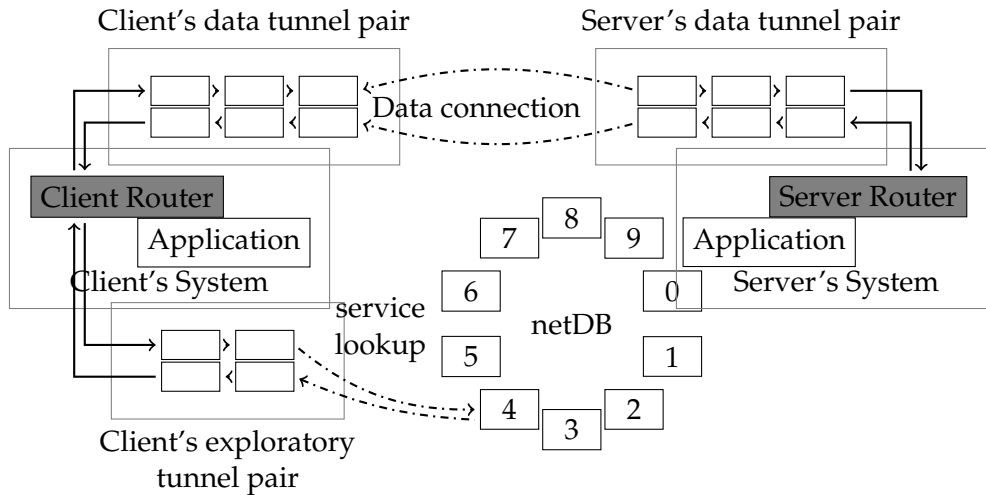


Figure 2.1: User accessing an eepsite

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 them¹. 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 thesis, we present an attack that requires fewer malicious nodes while still deanonymizing users. This threat model is also used by Hermann et al. [6], putting our result in some context.

2.7 Sybil Attacks

One well-known attack on anonymity systems is the so-called Sybil attack [4], 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 hosts, the

¹http://i2p2.de/how_threatmodel.html

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.

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 key space changes every day at midnight UTC. Nodes clustered at a certain point in the key space on one day will, therefore, be distributed randomly on any other day. However, this change does not include any randomness and is thus completely predictable, which can be used in attacks.

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. 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. Only after reaching a limit of eight lookups without success, the search is considered to have failed.

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) actively participating in the network and acting as `floodfill` peers. The description of our attacks is structured as follows:

- a) We take control over the floodfill database. We either forcibly 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, which 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 key space location, while there are still several legitimate `floodfill` nodes at random places in the key space 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 key space, 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 key space). 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 time, already gives the attacker many possible identities to completely control any position in the keyspace.

Introducing a new node in the network has 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 so it takes some time until the Sybil attack reaches the maximal impact. In addition, as mentioned previously, the storage location of the keys, 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 due to the keyspace rotation. This requires attacking nodes to change their location in the keyspace and opening a window where 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 deanonymisation 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 a peer, isolating it in the network.

3.4 Deanonymisation 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 they know of. To verify storage of a database record the node sends a lookup using one of its tunnels to another server nearby after waiting for 20 seconds. If both servers, the one stored to and the one handling the verifying lookup, are controlled by the same entity, she can observe both interactions and determine the probability of both interactions originating from the same node. As many database records are stored on the same `floodfill` node such a pair of malicious `floodfill` nodes can easily create such a probabilistic mapping for all of these records.

Storage of peer information is done without a tunnel, that is, it is done in the clear, as the client is exposed in the content of the database entry anyway. Storage verification on the other hand is done through one of its `exploratory` tunnels to make it more difficult distinguishing storage verification from normal lookup (if `floodfill` nodes could distinguish verifications from normal lookup, they could allow verifications and still hide the stored information from

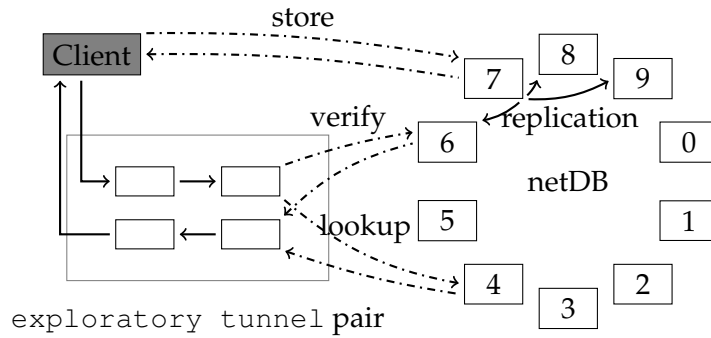


Figure 3.1: Deanonymizing attack

normal lookups). As a result, the first part of this interaction exposes the client node, while the second part exposes one of his `exploratory tunnel` endpoints. This combination allows us to create a probabilistic mapping between `exploratory tunnel` endpoints and the peers owning the tunnel.

Therefore, if the attacker can link actions to an `exploratory tunnel` endpoint, she can use this probabilistic mapping to identify the client initiating this action, effectively deanonymizing the 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 as he interacts directly with the endpoints. If the attacker now places malicious `floodfill` nodes at the right positions to observe the lookups for services interesting to her, she can combine the probabilistic mapping with the service lookups.

The whole process is shown in Figure 3.1: The client stores its peer information on node 7 in the `NetDB`. This node then pushes the peer information to other `floodfill` nodes close in the `NetDB`, in this case nodes 6, 8 and 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` using the same `exploratory tunnel` pair from node 4. If the attacker controls nodes 4, 6 and 7 she can deanonymize the tunnel due to the `verify` interaction and knows which client is requesting the service information 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 interacts 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.

4

IMPLEMENTATION AND SETUP

In this chapter, we give in-depth information on the test setup – inside both the “real” I2P network as well as our private test installation – and the implementation used to obtain the presented results.

As bootstrapping a separated test network was never intended to work with I2P, several modifications were necessary to complete this process. Care was taken to deploy I2P nodes on different continents to measure the effect of geographical distance. Finally, in terms of implementation, we discuss the communication infrastructure used to coordinate the attack as well as the design built to allow online analysis of the identifying data obtained during the attacks.

4.1 Node Setup

All nodes were set up on their own virtual machines. Every node was running slightly patched I2P software on top of OpenJDK 6. The attack code was plugged in using ABCL. The use of ABCL together with a Common Lisp development environment allowed for interactive development inside the running I2P process and allowed to inspect the running process for evaluation purposes without adding additional tailored interfaces.

For the local test network, 150 of these nodes were launched. These were running on a single VM host and connected to a virtual network without connectivity to the outside world.

For the tests in the “normal” I2P network, 30 nodes were used, 20 attack nodes in Santa Barbara as well as five victim nodes in both locations. They were all using one public IPv4 address each and connected to the university network in Erlangen and Santa Barbara. They were configured to use a moderate data rate just below the XX kbps ordinary I2P `floodfill` nodes use.

4.1.1 Test Network

In order to set up a test I2P network, several challenges must be met. All of these required modifications of the I2P source. First, one needs to lower several magic numbers in the I2P source code, as the test network will contain significantly less nodes (around 150 for our experiments). Therefore, the number of nodes expected to participate in the network must be lowered as I2P nodes employ several health checks by counting the nodes they know or are connected to.

Second, the plain I2P router rejects to participate in any connection until it is up and running for at least 20 minutes. During this timeframe, it verifies whether it is able to create connections through nodes in the network. However, if all nodes were just started up, no node will accept connections. Therefore, the individual routers assume connectivity problems and abort the startup. To handle this situation, it is necessary to have peers accept tunnel participation as soon as they are properly initialized and before they start connectivity checks.

In parallel, one needs to take care of bootstrapping the `NetDB` in order to allow nodes to find each other in the network. For the purpose of the test network, we collected router identities via `ssh` and distributed them using a standard webserver running on the master node. This is a time-critical process as well, as nodes only fetch router identities the very first time they start up. As a result, only collecting the identities once all routers are running is not enough. Rather, it is necessary to start a few nodes, collect their identities, and then start the other nodes.

This way, the first few routers will not know of any peers at the very beginning. Peers starting later retrieve the peer information for the first few from the webserver and use these first few peers for tunnel building and `NetDB` interactions. As a result, the first few learn about later nodes and due to normal network activity, the knowledge on the network structure quickly converges to normal state.

4.1.2 Real Network

Peers running in the real I2P network required significantly more resources than ones running in the test network did. While we were able to run 150 nodes in the test network on a single host, only 25 of them could run in parallel while participating in the real network. This is in part due to the fact that 20 of these nodes were participating as `floodfill` servers. While the whole network database consisted of 150 `routerInfos`, the real network counts 20,000 nodes in total and the 20 `floodfill` nodes accounted for more than 5% of all `NetDB` participants. As a result, each of these nodes had to handle significantly more `NetDB` requests.

In addition, as the I2P network is run across the whole globe, it is not enough to measure the impact of our attacks against nodes running on the same VM Host on the same physical network. Additional nodes in Erlangen were used to get a more geographically diverse view on the impact. However, all nodes were running on fast, well-connected university networks. While this limits the generality of the results, no difference, as far as the deanonymization attack is concerned, was measurable between the geographical diverse nodes, which can be seen as an indication that the attacks are mostly independent from external influences.

4.2 Inter-Node Communication

All I2P routers implemented an IRC command and control channel. IRC was chosen, because it provides both, one-to-one communication between two attacking nodes and between one node and the user operating the attack as well as communication between groups of participants. Several channels were used to separately instruct each group of attackers. These groups were

given distinct roles during the attacks. For example for the deanonymization described in 3.4, one group is controlling the `NetDB` address space occupied by the monitored peer information, while the other group monitors the address space used for the relevant service information.

The channel with all nodes was used for general commands such as reloading the attack code. One-To-One queries were, for example, useful to move logged information to the node responsible for correlation without burdening other nodes with parsing and handling of these messages. In addition, results were forwarded to the controller in this way.

4.3 Distributed Correlation

The load for calculating correlations in the deanonymization attack was distributed across all malicious I2P nodes in the network. The implementation made use of the already-present kademlia distance calculation. We were, however utilizing a different modification (the string "EVILEVIL" instead of the current date) to distribute the load evenly across all participating nodes while the attack concentrated their IDs – and therefore also their responsibility in the `NetDB` – on a single spot in the keyspace. All results were forwarded as soon as they were available from the individual logging nodes to the human controlling the attack. The resulting infrastructure proved scalable and fast enough to perform the whole analysis online.

5

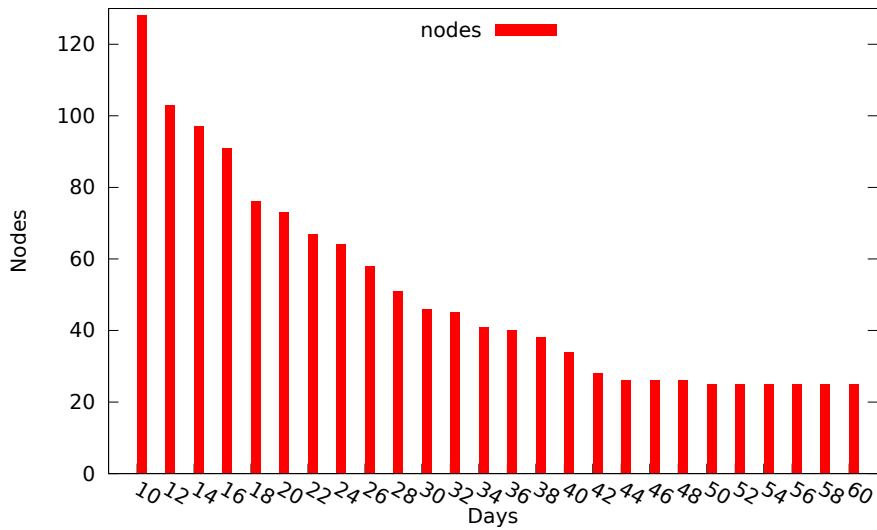
EVALUATION

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. Changing the identity of attacking nodes has resulted in some broken tunnels for independent peers but this happened at most once a day and the same loss of tunnels would have been caused by shutting down the nodes during the night. 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.

5.1 Floodfill Takeover

After presenting evidence on how many `floodfill` participants are manual, 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, our 26 nodes 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. If we limit the experiment to a single host instead of all hosts combined, the same fraction of about 30 %

Figure 5.1: Legitimate floodfill nodes after n days

did not change `floodfill` status. Therefore, the nodes not changing their `floodfill` status were not known to a larger subset of our hosts. As seen in Figure 5.1, 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.

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.

5.2 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 and the modification of their I2P software was limited to additional logging, which allowed us to later confirm our results.

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 and configured to use 128kB/s of download and 64kB/s of upload data rate. The legitimate nodes were split evenly between this 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 statistics¹ reported between 18,000 and 28,000 nodes and 320 to 350 `floodfill` nodes, fluctuations during the day. Therefore, we were controlling less than 7% of `floodfill` nodes and a negligible part of total nodes.

5.3 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. Building up the database took less than 30 minutes on a twelve-core Xeon server. This set of identities was made 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 without any two accidentally taking the same identity. The same setup was used for the Sybil attacks in all further experiments.

5.4 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.

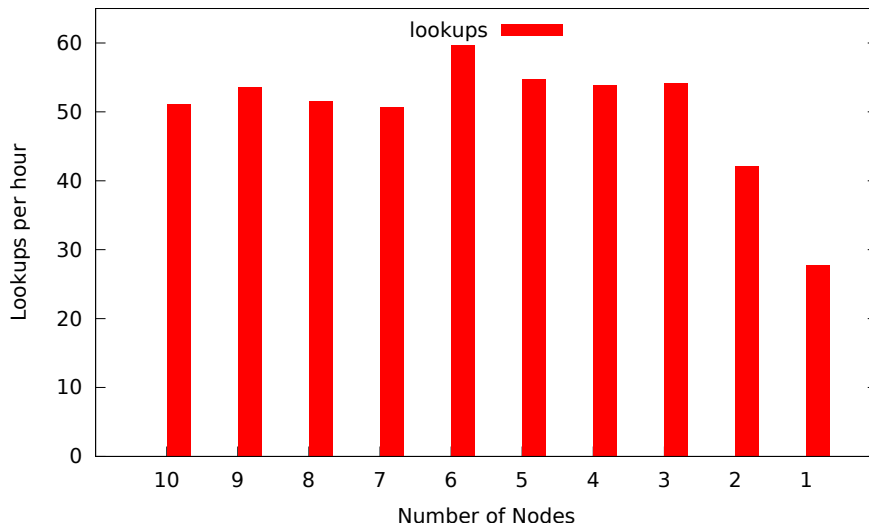
We ran the eclipse attack over a period of 42 hours. During this time, victims were on average able to reach the blocked `eepsite` for a total of five minutes. Three out of six nodes were not able to reach the `eepsite` at any point in time, and the most successful victim was able to interact with it 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).

5.5 Deanonymisation of Users

To simulate deanonymisation of users, we changed our victims' identity so they all mapped to the same keyspace region – they need to be closer to the malicious `floodfill` nodes than to legitimate ones in the network. For our experiments, we choose the identities in such a way, that the first fourteen bits matched. For legitimate `floodfill` nodes, on average the first eight to nine bits match. As a result, a single attack group could monitor all our victims. This allowed us to verify the success of the attack for all our nodes without requiring additional resources. The actual attack was then carried out by ten malicious nodes distributed to act as `floodfill` nodes in the keyspace region occupied by the victims. Using this setup and considering all lookups in a time window of 60 seconds before and after the storage as matches, we were able to see the correct verification step for over 90% of all storage interactions. However, for every correctly detected verification step we additionally recorded an average of nine unrelated lookups as

¹<http://stats.i2p.in>

Figure 5.2: Logged service lookups per hour



likely matches. Limiting the results to all lookups happening 20 to 25 seconds after the storage, we were still able to see over 60 % of all store-verify pairs, and 52 % of all potential correlations matched the victim’s real tunnels. The five seconds window was chosen after analyzing the recorded pairs as large enough to account for latency in the network while keeping the rate of misclassification low.

While all verification should fall into the smaller five seconds window the larger time frame also allows us to capture store/lookups where the first verification failed and storage was retried by the victim. In this case, we could correlate the correct tunnel endpoint even when we could only see the first store and the second verification. Better detection should be possible taking into account that peers use the same peers over an extended period of time.

To determine the number of malicious `floodfill` nodes needed to reliably capture the client’s lookup, we had all our six victims configured to query the service information of our test `eebsite` and monitor how many lookups could be observed for each number of malicious `floodfill` nodes. For all numbers of malicious nodes, we ran the experiment for a total of eight hours each, during different parts of the day. This was done to avoid that the different number of `routers` at different times in the day would influence the results. The experiments (Figure 5.2) show a constant amount of around 50 lookups logged every hour until fewer than three malicious nodes are left in the network. More precisely, there was a lookup from all our legitimate 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 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.

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 $i : 0 \leq i < N$ 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 $q \ll p$, 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 identifier) with Alice, and we see this tunnel identifier accessing R .

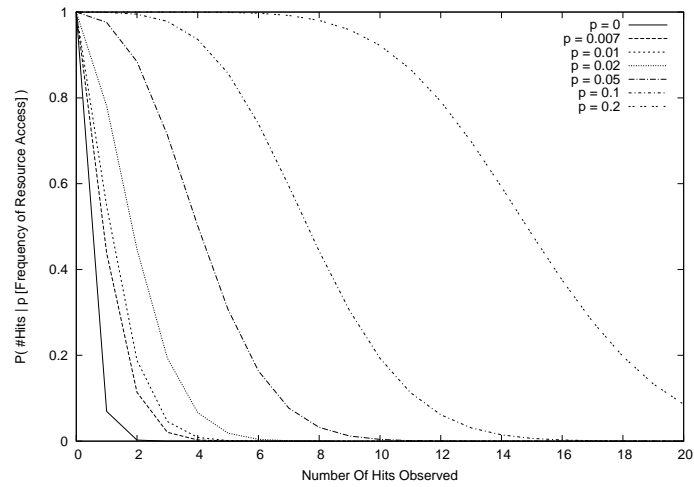
The probability that $t_i \in L_i = x = u * p + (1 - u) * q = 0.5p + 0.5q$. This is the chance of Alice accessing resource, in case we guessed correctly, plus the chance of a random hit when we misidentified the tunnel. Thus:

$$P(k \text{ hits}) = \binom{N}{k} x^k * (1 - x)^{N-k} \quad (5.1)$$

Since we care about the probability of *at least* k hits, we require the cumulative distribution function. In Figure 5.3, 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 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 5.3 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.

Figure 5.3: Probability of k or more hits, depending on p 

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.

6

LIMITATIONS

In this section, we discuss limitations to our attacks imposed by the nature of the design, as well as practical limitations when carrying out the attacks in the real I2P network.

A given node has, under normal operation, three pairs of exploratory tunnels in use at any given time and these tunnels expire after ten minutes. The correlation attack presented in Section 3.4, however, can at most observe one tunnel every ten minutes, covering one third of the total number of tunnels. Furthermore, as leases expire after ten minutes, the chance of correct association increases with longer or repeated use of a darknet resource.

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 `floodfill` nodes required for each monitored service and the number of darknet services interesting to the attacker is likely to be small, tracking relevant actions of a 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 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 in the keyspace.

DISCUSSION

Our results confirm the well-known shortcomings of DHTs in anonymity systems [8]. Moreover, we show that ad-hoc countermeasures implemented in the real-world distributed anonymity system I2P open up new vulnerabilities.

7.1 Potential 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 for the eclipse attack, one could 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 verification step, and only signal successful lookup for the verification while replying with a negative response to all regular lookups. The association of storage verification for service information is more reliable than that for peer information, as verification is done using one of the tunnels already mentioned as part of the service information.

7.2 Discovery of the attack

After running our nodes for three weeks in the I2P network developers noticed our group of 20 `floodfill` nodes connecting with consecutive IP addresses and cloned configuration. They were changing their identity together at midnight each day, and were suspiciously close to each

other in the keyspace. Additionally, implementation details in our attack code resulted in repeated error messages – related to loosing tunnels – written to the logs of the peers with which our attackers were interacting. Apart from the closeness in the keyspace, which is a core property of the attack, most of these observed indications could have been hidden by an attacker if desired by using cloud services to get nodes in different IP networks and using several hours to actually move the attack nodes to the target location.

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 7.3 and 7.4.

7.3 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 safe 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 should be able to work around this limitation by using several cloud services.

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

7.4 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 timeframe 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 keypace 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.

RELATED WORK

Distributed anonymity systems, as well as I2P specifically, have been discussed in previous work.

Tran et al. [11] described common failures of DHT-based anonymity schemes and Mittal et al. [8] 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. [6], 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. [6] 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 known user takes 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.

Bibliography

- [1] CASTRO, M., DRUSCHEL, P., GANESH, A., ROWSTRON, A., AND WALLACH, D. S. Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.* 36, SI (Dec. 2002), 299–314.
- [2] DINGLEDINE, R., MATHEWSON, N., MURDOCH, S., AND SYVERSON, P. Tor: the second-generation onion router 2012 draft. 2012.
- [3] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: the second-generation onion router. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13* (Berkeley, CA, USA, 2004), SSYM'04, USENIX Association, pp. 21–21.
- [4] DOUCEUR, J. The sybil attack. In *Peer-to-Peer Systems*, P. Druschel, F. Kaashoek, and A. Rowstron, Eds., vol. 2429 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2002, pp. 251–260.
- [5] EVANS, N., AND GROTHOFF, C. R5n: Randomized recursive routing for restricted-route networks. In *Network and System Security (NSS), 2011 5th International Conference on* (sept. 2011), pp. 316–321.
- [6] HERRMANN, M., AND 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* (Berlin, Heidelberg, 2011), PETS'11, Springer-Verlag, pp. 155–174.
- [7] MAYMOUNKOV, P., AND MAZIÈRES, D. Kademia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems*, P. Druschel, F. Kaashoek, and A. Rowstron, Eds., vol. 2429 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2002, pp. 53–65.
- [8] MITTAL, P., AND BORISOV, N. Information leaks in structured peer-to-peer anonymous communication systems. *ACM Trans. Inf. Syst. Secur.* 15, 1 (Mar. 2012), 5:1–5:28.
- [9] SINGH, A., WAN “JOHNNY NGAN, T., DRUSCHEL, P., AND WALLACH, D. S. Eclipse attacks on overlay networks: Threats and defenses. In *In IEEE INFOCOM* (2006).
- [10] TIMPANARO, J. P., CHRISMENT, I., AND FESTOR, O. Monitoring the I2P network.
- [11] TRAN, A., HOPPER, N., AND 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* (New York, NY, USA, 2009), WPES '09, ACM, pp. 71–80.
- [12] WOLCHOK, S., HOFMANN, O. S., HENINGER, N., FELTEN, E. W., HALDERMAN, J. A., ROSSBACH, C. J., WATERS, B., AND WITCHEL, E. Defeating Vanish with low-cost Sybil attacks against large DHTs. In *Proc. of NDSS* (2010).

Glossary

`client` individual applications running on top of the I2P framework and either providing a service to the network or allowing the user to interact with services.. 8, 11

`eebsite` are anonymously hosted websites inside the I2P network. 6, 8, 11, 16, 22, 23

`exploratory tunnel` used by the router for all NetDB interaction. Exploratory tunnels are shared for all clients. 10, 11, 15, 16, 29

`floodfill` nodes are Database supernodes. They are the subset of all nodes in the network with good connectivity used to serve metadata. 9–18, 20–23, 26–29

`leaseSet` piece of data describing a service inside the I2P Network. leaseSets are required in order to communicate with the service. 9, 12

`NetDB` Distributed Hashtable used by I2P to locate peers and information. 6, 9–16, 18, 19, 26, 30

`router` core component of each I2P node. Routers are responsible for maintaining the network connections, netDB lookup and all packet dispatching. 8–11, 14, 20, 22, 23

`routerInfo` piece of data describing a node in the I2P network. routerInfos contain all the information needed to communicate with the node over the Internet. 9, 12, 13, 18

`tunnelID` unique identifier used by I2P peers to associate incoming packets with the tunnel they belong to. tunnelIDs are local to the individual node. 8, 9