Privacy-Implications of Performance-Based Peer Selection by Onion-Routers: A Real-World Case Study Using I2P

Michael Herrmann and Christian Grothoff

Technische Universität München, Munich, Germany
{herrmann,grothoff}@net.in.tum.de

Abstract. I2P is one of the most widely used anonymizing Peer-to-Peer networks on the Internet today. Like Tor, it uses onion routing to build tunnels between peers as the basis for providing anonymous communication channels. Unlike Tor, I2P integrates a range of anonymously hosted services directly with the platform. This paper presents a new attack on the I2P Peer-to-Peer network, with the goal of determining the identity of peers that are anonymously hosting HTTP services (Eepsite) in the network.

Key design choices made by I2P developers, in particular performance-based peer selection, enable a sophisticated adversary with modest resources to break key security assumptions. Our attack first obtains an estimate of the victim's view of the network. Then, the adversary selectively targets a small number of peers used by the victim with a denial-of-service attack while giving the victim the opportunity to replace those peers with other peers that are controlled by the adversary. Finally, the adversary performs some simple measurements to determine the identity of the peer hosting the service.

This paper provides the necessary background on I2P, gives details on the attack — including experimental data from measurements against the actual I2P network — and discusses possible solutions.

1 Introduction

Onion routing [13] is an established technique to provide sender- or receiver-anonymity for low-latency network applications. Both Tor [2] and I2P [15] provide anonymity to their users via an open network of onion routers run by volunteers. However, there are significant differences in the details of how these networks implement the basic technique. For many of the differences, the existing related work does not provide a clear answer as to which approach is better.

In this paper, we report on our exploitations of some of the *design choices* in I2P to deanonymize I2P services, specifically I2P Eepsites.[1] An Eepsite is a website hosted anonymously within the I2P network and accessed via HTTP tunneled through the I2P network, which also acts as an anonymizing SOCKS

[1] Our basic technique could be applied to other kinds of I2P services as well.

S. Fischer-Hübner and N. Hopper (Eds.): PETS 2011, LNCS 6794, pp. 155–174, 2011.

proxy. Our attack requires a modest amount of resources; the only special requirement, to run I2P peers in several different /16 peers, can also be met by any Internet user, for example by using cloud based services. While this requirement may put us outside of the I2P attacker model, our other requirements — participation in the I2P network and a modest amount of bandwidth — are easily within common attacker models for anonymizing P2P networks, including I2P and Tor. We have implemented and tested the attack on the extant I2P network in early 2011, making our attacker a credible real-world adversary.

Our attack is primarily based on exploiting I2P's performance-based selection of peers for tunnel construction, I2P's usage of unidirectional tunnels and the fact that Eepsites are located at a static location in the network. Using a combination of peers that participate as monitors in the network and other peers that selectively reduce the performance of certain other peers, our attack deduces with high degree of certainty the identity of the peer hosting the targeted Eepsite. In contrast to previous deanonymization attacks (such as [9,3]), our attack does not rely on congestion-induced changes to latency. In fact, the denial-of-service component of the attack focuses on peers that are not known to participate in the Eepsite's active tunnels at the time.

We have evaluated our technique not merely in simulation or a testbed but against the real I2P network. This paper presents experimental results obtained in early 2011 using I2P version 0.8.3, modified for our attack.

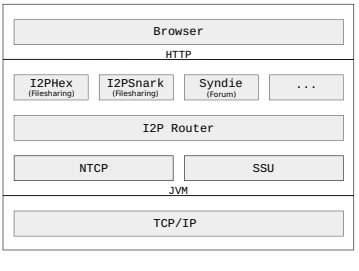
The main contributions of this paper are as follows:

- An independent characterization of the I2P protocol
- A novel attack on anonymity based on the heuristic performance-based peer selection for uni-directional tunnels
- Experimental evaluation of the attack
- Recommendations for improving the I2P design to thwart the attack

The rest of the paper is structured as follows. Section 2 provides a detailed overview of the I2P network. Section 3 describes our attack and Section 4 presents the experimental results. Finally, Section 5 discusses possible solutions and relates our attack to previous work on deanonymization for similar systems.

2 Background: I2P

I2P is a multi-application framework for anonymous P2P networking written in Java. On top of the native Internet protocol, I2P specifies the use of two different peer-to-peer transport protocols. The first is called *NIO-based TCP* (NTCP), where NIO refers to the Java New I/O library. The second is called *Secure Semireliable UDP* (SSU), providing UDP-based message transfer.



I2P Architecture

The core of the I2P framework is the I2P router, which implements key components of the I2P protocol. Tasks of the I2P router include: maintaining peer statistics, performing encryption/decryption and building tunnels. I2P applications rely on the anonymizing tunnels provided by the I2P router for privacy protection; consequently, the I2P router is central to the security of all I2P applications and the analysis presented in this paper.

Many Internet applications can be implemented on top of the I2P router. An application provided by a particular I2P peer is referred to as a service. For example, I2P includes services to host HTTP servers, to provide IRC-based communication and to perform POP/SMTP-based email transfer. Most I2P services are controlled and used via a web browser interface.

2.1 Peer and Service Discovery

Like most other P2P networks, I2P has to deal with the problem of finding peers and subsequently the services offered by those peers. Every peer in the I2P network is uniquely identified by a data structure called *routerInfo*. This data structure holds all the key information about the peer, including public keys of the peer, a 256 bit hash-identifier and information about how the peer can be contacted. I2P addresses the bootstrapping problem, the problem of initially discovering some other peer in the network, by using a non-anonymous HTTP download of a list of routerInfos for available I2P peers from a fixed location.

I2P's DHT: the netDB. After bootstrapping, I2P uses a super-peer DHT to build a network database, called the *netDB*, with information about all the peers and services available in the network. The super-peers that maintain this database are called *floodfill* peers; each floodfill peer is responsible for the information closest to its ID. Proximity is determined using Kademlia's XOR distance metric [8]. If a peer has sufficient bandwidth and its configuration allows it, a peer can promote itself to floodfill status and will do so as soon as the number of active floodfill peers in the network drops below a certain threshold.

Storing Data in the netDB. Information about how to contact a service provided by an I2P peer is kept in a so-called *leaseSet*. LeaseSets are stored in the same netDB that also contains routerInfos; nevertheless, leaseSets and routerInfos are independent entities that only share the same storage facility. A leaseSet primarily specifies a set of entry points (called *leases*) to the service. An entry point is the identification of an inbound tunnel at a peer currently serving as an inbound gateway to the service.

The lookup and storage of leaseSets and routerInfos is achieved by sending the respective requests to a floodfill server. Figure 1 illustrates the storage process for a leaseSet. After a floodfill peer receives a request, it replicates the information at seven additional closest floodfill peers and sends a confirmation to the initiator.

Retrieving Data from the netDB. Retrieving routerInfos and leaseSets is also performed via tunnels. The request is transmitted to the — with respect to the destination address — two closest floodfill peers known to the requester.

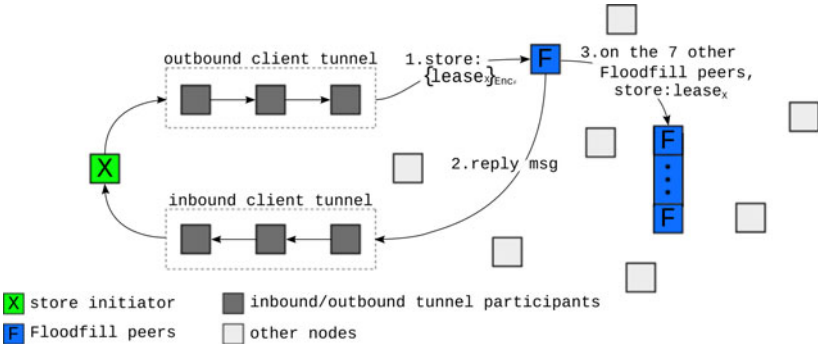


Fig. 1. I2P uses tunnels to store a lease in the floodfill database to hide the identity of the (HTTP) server

If a floodfill peer does not have the requested information, a list of other close floodfill peers is sent back. The replies are transmitted to the initiator using an inbound tunnel. If both floodfill peers do not have the requested information, the requesting peer queries two other floodfill peers until all known floodfill peers have been contacted.

2.2 I2P Tunnels

I2P uses *tunnels* to hide the IP address of a participant in an online interaction. I2P tunnels closely resemble onion routing as implemented in Tor with circuits [2]: the initiator selects the route through the network, no artificial delays are introduced when forwarding, and link- and layered-encryption are used to protect the data against observers.

I2P Tunnels are Unidirectional. Tunnels in I2P only transfer payload data in one direction. In order to achieve bi-directional communication, I2P uses *inbound* and *outbound* tunnels. Inbound tunnels are used to transmit data to the peer that constructed the tunnel and outbound tunnels are used to transfer data from the peer that constructed the tunnel. Note that only the peer that constructed the tunnel knows all of the peers in the tunnel.

For outbound tunnels, multiple layers of encryption are added by the creator of a message; each one is then removed by the corresponding peer as the message traverses the outbound tunnel.

For inbound tunnels, adding all layers of encryption at the first peer is not possible; this would require the first inbound node to know the secret tunnel keys for all of the participants of the tunnel. Instead, every node in an inbound tunnel *adds* an additional layer of encryption. Finally, the creator of the tunnel, who knows the tunnel keys used by each peer from the tunnel construction phase, removes all layers of encryption to obtain the original message.

Tunnel Diversity. Every I2P peer creates multiple tunnels; the specific number of tunnels and the tunnel length depend on the peer configuration. The length of the tunnel is considered to be a trade-off between speed and anonymity and I2P gives the end-user control over this setting. The user specifies two non-negative numbers, x and y. For each tunnel, I2P selects a random number $r \in [-y, y]$ and constructs a tunnel of length $\max(x + r, 0)$.

In addition to the distinction between inbound and outbound tunnels based on the tunnel's transfer direction, I2P further distinguishes between *exploratory* and *client* tunnels. Exploratory tunnels are for routerInfo queries to the netDB and for tunnel management. They are not used for privacy-sensitive operations. Client tunnels are used for all typical application level network messages, for example to provide tunnels for Eepsites and for leaseSet operations on the netDB.

Tunnel Construction. In order to select peers for tunnel construction, I2P first categorizes all known peers into tiers. Depending on the type of tunnel that is being created, the peer selection algorithm then attempts to select peers exclusively from a particular tier. In addition to selecting peers from particular tiers, I2P also avoids the selection of multiple peers from the same /16 (IPv4) network for the same tunnel.

After selecting peers for the tunnel, the initiator sends tunnel construction requests (via that partially built tunnel) to the selected peers. A peer receiving a tunnel construction request is free to either accept to participate in the tunnel or reject the request, indicating a reason for the refusal. Naturally, tunnels can still fail if peers that accepted a tunnel construction request are later unable to sustain the tunnel. The behavior of a peer faced with tunnel construction requests (including the reason given for rejection) as well as tunnel failures are important for the performance evaluation of peers, which is used for assigning peers to tiers.

Tier-based Peer Selection. An I2P peer chooses other peers randomly from a particular tier depending on the type of the tunnel. A tier consists of peers that share certain performance characteristics. I2P places certain well-performing peers into two special tiers:

Fast tier. Peers with high throughput

High-capacity tier. Peers that will accept a tunnel request with high probability.

The fast tier is considered the most valuable tier and is used for constructing client tunnels. In the theoretical case where the fast tier does not have a sufficient number of peers, I2P falls back to using peers from the high-capacity tier for peer selection in the construction of client tunnels. In practice, we were unable to observe this behavior since the fast tier was always sufficiently populated during our evaluation.

The high-capacity tier is the default choice for exploratory tunnels. Peers must also be in the high-capacity tier to be eligible for the fast tier. All other peers are only used as fallback options if the fast and high-capacity tiers lack available peers. In practice, this is unlikely to happen.

Peers are placed into tiers based on certain performance metrics. A peer is put in a particular tier if its corresponding performance value exceeds a threshold calculated by I2P for that tier.[2] The size of the fast and high-capacity tiers is bounded. For the fast tier the number of peers is between 8 and 30 and for the high-capacity tier between 10 and 75. If the number of peers in those tiers drops below the threshold, the best-performing peers from lower tiers are promoted. If the number of peers in a tier exceeds the upper limit, the lowest rated peers are demoted.

The I2P router keeps track of various performance statistics in order to sort peers into the correct tiers. Performance metrics are gathered more often for peers in the fast and high-capacity tiers, since performance metrics are always gathered if a peer is used for a tunnel. Furthermore, performance scores are cumulative; this generally results in higher performance values for peers in the fast and high-capacity tiers and reduces fluctuation.

Metrics for Tier Assignment. I2P is careful about only including performance metrics that are hard to manipulate, relying only on measurements entirely controlled by the peer for throughput and tunnel maintenance properties. In particular, information about tunnels created by other peers is not taken into consideration.

The *capacity value* of a peer is based on the number of times the peer accepts a tunnel request, the number of tunnel rejections and the number of tunnel failures that happen after a peer accepted to participate in a tunnel.

The goal of the capacity calculation is to estimate how a peer is likely to behave in the future in terms of its participation in tunnels. The calculation is primarily based on the accept, reject and failure actions of that peer. Furthermore, if the peer rejected events in the last 5 minutes, the reason given for the rejection is also considered. A detailed description of the capacity calculation algorithm can be found in [4]; the main point for this paper is that peers accepting tunnel requests score high, peers rejecting tunnel requests score low and peers participating in tunnels that then failed score very low in terms of their capacity value.

A peer's *speed value* is the mean of its three highest, one second throughput measurements in any tunnel established by the measuring peer over the course of the last day. Throughput is measured whenever data is sent through a peer via a tunnel created by the measuring peer. Naturally, throughput is bounded by the throughput capacity of the measuring peer as well as, for each individual measurement, the slowest peer in the tunnel. While it would be nice to be able to influence speed values of other peers, the fact that I2P uses the observed maximum over an entire day makes this unattractive: attacking a peer to reduce its speed for a whole day is expensive.

[2] The complex threshold calculation is described in detail in [4].

2.3 Eepsites

The I2P software comes with the *Jetty* web server[3]. Using Jetty, every I2P user can offer HTTP web pages to the I2P network using a domain under the `.i2p` TLD. Given such a domain name, I2P creates inbound and outbound client tunnels for the service and (periodically) publishes a leaseSet in the netDB.

Accessing an Eepsite involves several steps (illustrated in Fig. 2):

1. Eepsite host (server) creates inbound and outbound tunnels for sender-anonymity and publishes gateway information as a leaseSet in the netDB (as described in Section 2.1). Fresh tunnels and corresponding leaseSet updates are created at least every 10 minutes.
2. The peer running the HTTP client (client) uses a tunnel to access the netDB and retrieves the leaseSet information.
3. The client uses inbound and outbound tunnels (for receiver-anonymity) to contact the gateways from the leaseSet.
4. A handshake is performed via the tunnels for end-to-end encryption between server and client, using the public key in the leaseSet.
5. The HTTP request is transmitted through the outbound tunnel of the client and the inbound tunnel of the server.
6. The HTTP response is transmitted through the outbound tunnel of the server and the inbound tunnel of the client.

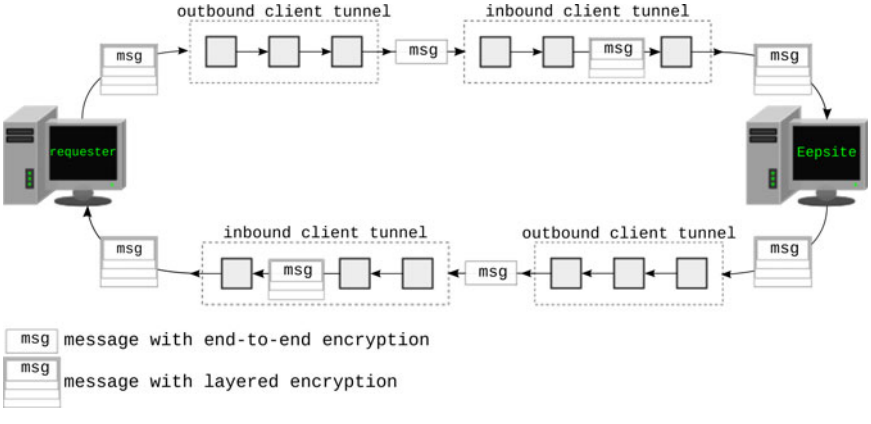


Fig. 2. Accessing an I2P Eepsite

Steps 5 and 6 can then be repeated; I2P reuses the resulting channel for subsequent HTTP requests to improve performance. This is somewhat relevant to the attack presented in this paper since it allows an attacker to repeatedly query the HTTP server without the need to perform the costly tunnel setup operations each time.

[3] `http://jetty.codehaus.org/jetty/`

Table 1. Key technical differences between Tor and I2P

Tor	I2P
3-hop tunnels	user-configurable, randomized number of hops
bi-directional tunnels	uni-directional tunnels
guards, bandwidth-based peer selection	performance-based peer selection
7 directory servers with complete data	super-peer DHT (floodfill peers)
link- and layered-encryption, but not (necessarily) end-to-end-encryption	end-to-end-, link- and layered-encryption
many exit nodes, few hidden services	one exit node, many integrated services
hidden services are external TCP servers	build-in servers for many services
implemented in C	implemented in Java
transport over TCP only	transport over TCP or UDP

2.4 Threat Model

The I2P project does not specify a formal threat model, it instead provides a list of possible well-known attack vectors (such as intersection / partitioning, tagging, DoS, harvesting, sybil and analysis attacks) and the authors discuss how the design relates to these attack vectors.[4]

Based on the scenarios described, I2P's attacker model closely resembles that of Tor: malicious peers are allowed to participate in the network, collect data and actively perform requests. However, the attacker is assumed to be unable to monitor the entire network traffic, should not control a vast number of peers (80% is used as an example) and should not be able to break cryptographic primitives.

2.5 Summary: I2P vs. Tor

The key philosophical difference between the well-known Tor network and I2P is that I2P tries to move existing Internet services into the I2P network and provide service implementations within the framework whereas Tor enables anonymous access to external Internet services implemented and operated separately. While Tor has hidden services and I2P has exit nodes, the canonical usage of Tor is accessing external services and the canonical usage of I2P is accessing integrated services.

I2P and Tor also differ in a number of technical details, some of which are key to the attack presented in the following section. Table 1 summarizes the main technical differences between the two projects.

3 Our Attack

Our attack assumes an adversary that actively participates in the network. Malicious nodes are distributed over different /16 subnets. The adversary should be

[4] `http://www.i2p2.de/how_threatmodel.html`

distributed in order to work around I2P's restriction of one node per subnet per tunnel and to provide reasonably well-performing malicious peers as neighbors regardless of the location of the victim on the Internet. Each of the participating peers is expected to have resources comparable to typical normal peers in the I2P network. The peers participate in the I2P network according to the network protocol. Our adversary does not have the capability to monitor the traffic of any other node. Our attack influences the performance of I2P peers likely to be chosen by the host of an Eepsite — the victim — for creating its client tunnels.

The goal of the attacker is to identify the peer "anonymously" hosting a given Eepsite with high probability. Furthermore, it is assumed that the Eepsite is available to the entire I2P network for the duration of the attack and hence resists intersection and partitioning attacks.

For our attack, the adversary uses three types of peers (illustrated in Fig. 3). The first type, a *monitor* peer, simply participates in the I2P network as "normal" peer, but reports certain statistics about tunnel operations back to the adversary. The most expensive operation (in terms of time and/or bandwidth) is getting the victim to select these monitor peers as its direct neighbors during tunnel construction. While there is always a (small) chance that the victim will select the adversary's monitor peers, the adversary uses a second type of peer, an *attack* peer (which performs a limited type of DoS attack) to influence the victim's tiers to the adversary's benefit. Note that, in contrast to [11], the goal of the attack is to change the fast tier, not to impact the availability or reachability of the Eepsite. Finally, the adversary also uses one peer to act as a "normal" visitor to the Eepsite, querying the I2P NetDB for leaseSets and issuing HTTP requests to the Eepsite. The leaseSets are used to determine which peers should be attacked (by the attack peers), and the HTTP requests are used to create a pattern which is detected by the monitor peers.

3.1 Distributed Monitoring

The main goal for the adversary is to control the nodes closest to the victim in the inbound and outbound tunnels of the Eepsite. I2P never picks two nodes from the same /16 network for the same tunnel twice. This makes it highly beneficial to use a distributed attacker that deploys monitor nodes across many /16 networks.

The attacker needs to only control the guard node and not multiple nodes per tunnel. Still, it is necessary to distribute the attacker across many /16 networks because for inbound tunnels, the guard node is the last node being chosen. So if the attacker's monitor nodes were all from the same /16 network, none of the attacker's monitor nodes must have been picked previously to participate in the tunnel before the guard node is selected in order to allow I2P to pick an attacker's monitor node as the guard node. If tunnels are of length n and the adversary controls a out of s (where $s = 30$ for the current version of I2P) monitor peers from the same /16 network in the victim's fast tier, the probability of being chosen as guard node would be only $\left(\frac{a}{s}\right)^n$ if all monitor peers are from the same /16 network. Even for small values of n, the attacker's /16 network

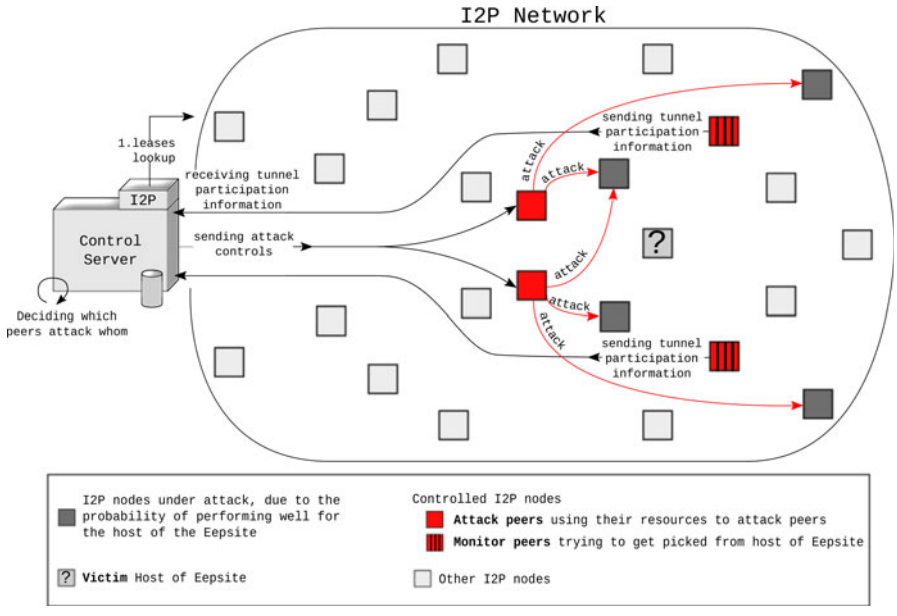


Fig. 3. Our attack on I2P uses several participating peers in different roles. Monitor peers gather statistical evidence, attack peers accelerate getting the monitor peers into the right position and the control server orchestrates activities.

would be blocked from being selected as the guard node most of the time. Since our attacker distributes his monitor nodes over many /16 networks, the chance of successfully becoming a guard node for the incoming tunnel is $\frac{a}{s}$, independent of the path length n.

3.2 Taking over the Victim's Fast Tier

The main challenge for the adversary is to force the victim to use the adversary's monitoring peers in its fast tier. Naturally, this requires the adversary to run several well-behaved and fast (monitor) peers. Clearly, depending on the size of the I2P network, just having a few monitor peers participate in the network would make it unlikely that the victim chooses these peers. Our attack takes advantage of the peer selection algorithm of I2P, which tries to select only well-performing peers for the tunnels. Thus, the adversary can increase its chances of entering the victim's fast tier by actively hampering the performance of the peers that are currently in the fast tier. While our goal is to enter the victim's fast tier, I2P's use of the highest observed speed over the last 24h makes it impractical to remove peers from the fast tier directly. Furthermore, the adversary may not be able to simply perform faster than the fastest s peers in the network — not to mention the victim may normally take a long time to even evaluate nodes controlled by the adversary. Thus, our attack makes use of the fact that I2P only allows high-capacity peers to remain in the fast tier; as a result, our attack

influences the peer selection algorithm by causing peers to reject tunnels, which in turn makes it likely that they will be removed from the high-capacity tier (and thereby also the fast tier). This increases the chance that the victim will then select the adversary's monitoring peers as replacements.

Before the adversary can get peers from the victim's fast tier to reject tunnel requests, the current nodes in the victim's fast tier must be identified. Our attack uses nodes that were recently specified in the leaseSet of the Eepsite as good targets. After all, nodes that are in the leaseSet must be in the fast tier of the victim at that time, and are thus likely to remain in the fast tier for a while. We found that this method worked better than trying to predict the fast tier from performance measurements done by adversarial nodes.

Given a (small) set of peers that are likely in the fast tier, the adversary performs a denial-of-service (DoS) attack against these peers. Possible venues we considered were attacks against the CPU (by forcing the victims to perform many public key operations) and bandwidth exhaustion. In the end, overloading the peers with a large number of idle tunnels turned out to be the most cost-effective strategy for the current I2P release. This attack either exhausts the amount of bandwidth the peer is configured to use, or, if that limit is rather high, creates more than the 2500 tunnels that an I2P peer can participate in at any time. It should be noted that the specifics of the DoS attack are not terribly relevant to the big picture of the attack, and alternative strategies would likely work as well.

3.3 Confirmation via Traffic Analysis

The final step of the attack is to observe the victim's participation in a pair of tunnels carrying the adversary's signal with monitor peers adjacent to the victim in both directions.

There are many established traffic analysis techniques to confirm that two endpoints are participating in the same low-latency tunnel [6,7]. Existing theoretical models typically assume that a single message moves through the tunnel largely unmodified with only small chances of message loss. For I2P, the situation is a bit different; HTTP requests are explicitly converted into an HTTP responses, and, moreover, individual HTTP requests result in two distinct peaks in the packet frequency plots (see Fig. 5 (a)). Thus, we deployed a simple, application-specific method for detecting this particular traffic pattern instead of using more complex, generic methods that do not incorporate this domain knowledge.

A periodic HTTP request at a fixed frequency t is issued by the adversary's control server to create a statistical pattern that is then used to identify the correct tunnels at the monitor peers (Fig. 4). For our experiments we use $t = 15s$. For each tunnel, each monitoring peer counts the number of packets received in buckets representing time intervals of packet arrival times modulo t. If the total number of packets is smaller than those transmitted by the adversary to the Eepsite, the circuit is ignored. If the number of packets is close to or exceeds the expected number, the monitoring peers compute how many standard deviations the largest bucket size is from the average bucket size. If the resulting

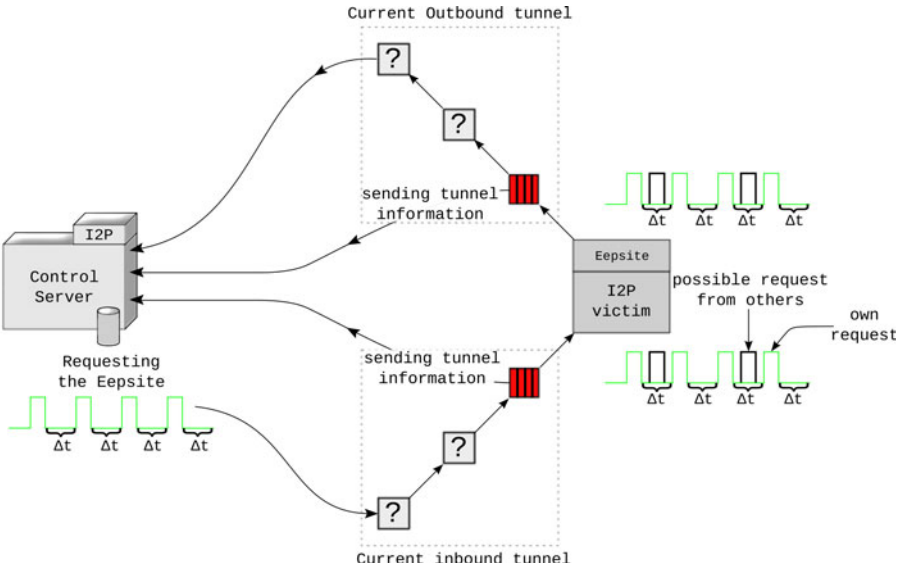


Fig. 4. A periodic signal is induced by the control server and detected by the monitor nodes. They report likely Eepsite hosts to the control server which aggregates the information.

factor is large, the packets were not equally distributed. Then, to exclude false-positives from short, non-periodic bursts, the monitoring peers perform the same calculation, this time for a time interval modulo q where $gcd(t, q) = 1$ and $|t - q|$ is small (we use $q = 16s$). If the signal had a frequency of t, the resulting factor should be very small; however, if a burst caused a false-positive, the resulting factor should be about as big as for the calculation modulo t. If the distribution is normalized modulo q, the tunnel is reported to the adversary as detected. If two monitoring peers report a peer between them at the same time, that peer is flagged as likely to be the Eepsite host. The sensitivity used for the standard deviation factor threshold determines how often the same peer needs to be flagged before the adversary can be certain.

4 Experimental Results

In this section, we present results from our experiments based on extending the extant I2P network with 70 "malicious" nodes (corresponding to less than 3.6% of the nodes in the network) on PlanetLab [12]. Monitor and attacker peers were configured to use at most 64 kb/s upstream and downstream bandwidth. We set up the control peer on a machine we controlled to minimize jitter. Furthermore, one of our peers was set up to host an Eepsite to serve as a victim for testing. This host was configured to use the standard I2P bandwidth settings (96 kb/s downstream and 40 kB/s upstream).

Table 2. Accuracy of the prediction for peers in the fast and high-capacity tiers using the n most recently observed peers from the lease set. The given percentage refers to the fraction of the peers from the n most recent leases that are actually in the respective tier. The fast tier typically consists of $s = 30$ peers, the high-capacity tier typically has 75 peers. At the time of the measurement, the I2P network contained at least 1921 peers in total.

# leases (most recent)	% nodes from lease set in fast tier	% nodes from lease set in high-capacity tier
5	60%	60%
10	40%	50%
15	40%	47%
20	45%	55%
25	36%	52%
30	30%	50%

All tests were performed by having all of our peers join the live I2P network and participate normally (except, of course, for attack-specific behavior). For our tests, we used 40 attack peers and 30 monitor peers. The 40 attack peers consistently utilized their 64 kb/s bandwidth; utilization of the 30 monitor peers differed widely depending on how they were used by normal I2P traffic. The I2P network contained *at least* 1921 peers at the time of our experiments.

We should note that the main impact of our experiments on the public I2P network was that a small fraction (about 1–2%) of the network was slowed down for a few hours. No personally identifiable information was collected. Despite our expectation that the impact of the experiment on the network would be small, an I2P developer did notice "strange" behavior (a significant increase in tunnels and traffic) when his node was (by chance) chosen as one of the targets for the attack. Those members of the I2P community we interacted with generally approved of us doing these kinds of (limited) experiments on I2P. Naturally, given an open community of anonymous participants, asking for everyone's approval is not possible.

4.1 Tier Evolution

First, we wanted to see how well the adversary would be able to predict the victim's fast tier from the public leaseSets for the Eepsite. This determines how much of the attack actually has a chance to have an effect on the victim's peer selection algorithm. Table 2 shows what fraction of the last n peers observed in the leaseSet were actually in the fast tier of the victim at the time. We configured the victim to use only one inbound and one outbound tunnel (I2P's default is two tunnels for each direction). This configuration captures the worst case scenario from the point of view of the adversary; with more tunnels, more leases could be learned and the adversary would get a better picture of the victim's fast tier.

Table 3. Direct impact of the tunnel acceptance rate of a peer under attack from various number of attackers with a configured bandwith limit of 64 kb/s. Note that an increasing number of attackers not only causes the peer under attack to reject tunnels, but additionally causes requests for tunnels to be lost and hence not be answered at all.

		under attack, number of attackers				
	normal	2	3	5	7	10
Tunnels accepted	82%	63%	52%	16%	9%	1%
Tunnels rejected	18%	36%	41%	40%	36%	28%
Tunnels lost	0%	1%	7%	44%	55%	71%

Table 4. Impact of the DoS attack on the network using 40 peers with a configured bandwidth limit of 64 kb/s. This table shows the increase in the churn for the high-capacity and fast tiers of the victim that the attacker tries to deanonymize. Each value represents the churn of nodes per 45 seconds tier evaluation cycle of the victim. Note that the attack uses our (limited-precision) leaseSet-based prediction heuristic (Section 3.2) to determine which peers to attack. If the attacker could be certain about which peers are in the respective tiers, the increase in churn would be significantly higher. Monitor peers provided by the attacker are not subjected to the attack.

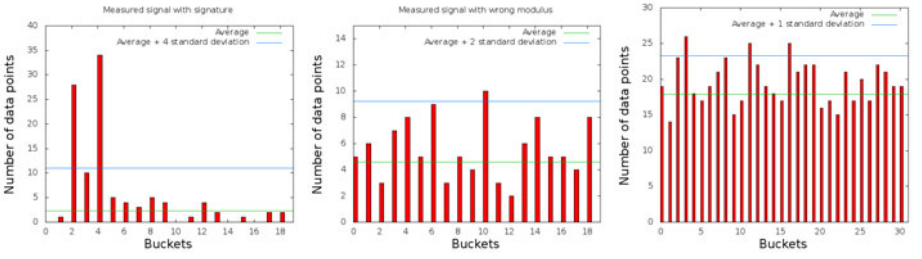
	normal	under attack
High-capacity tier churn	0.89 peers/cycle	3.41 peers/cycle
Fast tier churn	0.76 peers/cycle	1.71 peers/cycle

4.2 Attack Effectiveness

Next, we determined the impact of the DoS attack, first on the attacked peers (to confirm that the attack works as expected), and then on peer fluctuation in the fast and high capacity tier. Table 3 shows the impact of our attack on a single peer. It compares the tunnel request acceptance rate of an ordinary peer with the acceptance rate when that peer is attacked by several attackers. Table 4 shows the typical churn rate for peers in the high-capacity and fast tiers of the victim in two states: under normal operation, and under attack. The data corresponds to the adversary attacking the last 30 peers observed as leases (with the expected inaccuracies as listed in Table 2). The data shows that the DoS attack is effective at obstructing tunnel operations and that the victim reacts to these obstructions by replacing peers in its high-capacity and fast tiers more often.

4.3 Deanonymization

Finally, we measured how effective our statistical analysis is at determining the victim once the monitor peers are in place. First, we will provide some examples for what the statistical patterns observed by the monitor peers (Section 3.3) look like. Figure 5a shows a representative pattern for the case where the adversary observes the correct circuit with the signal and performs the statistical analysis



(a) (b) (c)

Fig. 5. Subfigure (a) shows a packet frequency plot for a circuit containing the signal (with timestamps calculated modulo the correct modulus t). Subfigure (b) shows the same packet frequency plot, but with timestamps calculated modulo a different modulus q. Finally, Subfigure (c) shows a packet frequency plot for a typical circuit not created by the adversary. In all plots, average and standard deviation are calculated over the distribution excluding the two largest values (since we expect two peaks).

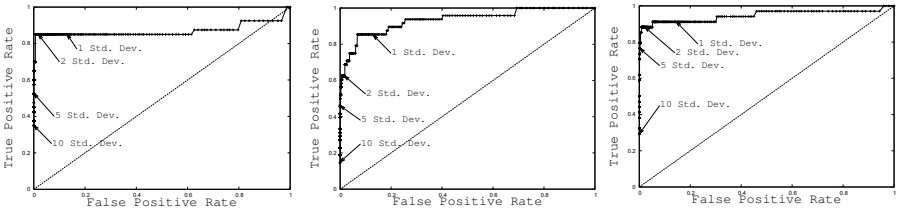
using the correct modulus (here $t = 15$). Internals of the I2P implementation typically create two distinct (and close) peaks if the signal is present. Since we expect to see these two peaks, we remove them from the distribution when calculating the average and standard deviations.

Figure 5b shows the same data using a different modulus (here $q = 16$), resulting in the peaks being destroyed. This would not be the case if the signal was not due to requests at the adversaries frequency of t. Sometimes, a circuit may experience spikes in load at a single point in time. Such spikes would show up as false-positive signals $\mod t$, but also as spikes $\mod q$. Our analysis eliminates these false-positives by only considering signals valid that show up $\mod t$ but are extinguished $\mod q$.

Finally, Figure 5c shows a typical pattern for a circuit that does not contain the signal. It should be noted that during our experiments, most circuits never reached the required minimum number of messages (approximately the number of messages transmitted by the adversary via the tunnel) and were hence filtered long before this statistical analysis is even performed. As a consequence, the adversary also does not have to worry about small sample sizes for calculating averages and standard deviations.

Figure 6 shows the ROC curves with the ratios for true-positives and false-positives for different standard deviation thresholds (in the range of 0 to 10 standard deviations). Tunnels with too few packets to carry the adversary's signal are not considered; for instance, for the 1-hop experiment, 47,503 out of 62,366 tunnels (76%) did not carry a sufficient number of packets. If such tunnels were included, the false-positive rate of the analysis would be lower.

The data from Figure 6 was obtained over the course of four hours with the victim manipulated to give the attacker control over the entire fast tier (to control this variable in the experiment). In practice, during a long-term measurement the adversary would not know at what times his monitor peers



(a) 1-hop tunnel (b) 2-hop tunnel (c) 3-hop tunnel

Fig. 6. Final result of the statistical analysis. For each tunnel length, the monitor peers collected 4h worth of data. During this time, the victim created 40, 48 and 34 tunnels for 1-hop (Fig. 6a), 2-hop (Fig. 6b) and 3-hop (Fig .6c) tunnels respectively. The true-positive rates represent the fraction of those tunnels flagged by the statistical analysis for the given threshold. The monitor peers also observed a varying number of other tunnels (with a sufficient number of packets) unrelated to the victim (14,823 for 1-hop (Fig. 6a), 11,861 for 2-hop (Fig. 6b) and 5,898 for 3-hop (Fig. 6c)). The false-positive rates represent the fraction of those tunnels flagged by the statistical analysis for the given threshold. We marked the 1, 2, 5 and 10 standard deviation thresholds in the charts.

are in the correct position, making false-positive measurements more frequent. If the adversary is weak, he might rarely be in the correct position and hence would need to apply an aggressively high threshold to avoid false-positives. For example, if the adversary is only able to observe the signal 10% of the time, the ratio between the target and the top FP peer must be significantly larger than 10:1 to avoid identifying the wrong peer as the host.

Using a threshold of just one standard deviation would be a bit low, given that the adversary must expect many more non-victim tunnels over the duration of an experiment. During our experiment, the ratio was 40:14,823 for the 4h measurement with 1-hop tunnels. In reality, the adversary should expect even higher ratios because the adversary is likely to control a smaller fraction of the fast tier of the victim. Figure 6 shows that the signal is strong enough to be detected even when using a wide range of thresholds that are so high that there are virtually no false-positives. There are also no significant differences in the quality of the results between 1, 2 and 3-hop tunnels. Additional experimental results are included in [4].

5 Discussion

This work confirms the well-known result [1] that attacks on availability or reliability of an anonymizing service can be used to compromise anonymity. What we have shown specifically is that anonymizing networks that have a strong bias towards well-performing peers for tunnel construction are particularly vulnerable to this type of attack. Once the tunnel is compromised, other researchers

have shown that latency measurements could be used to determine the likely identity of the victim [5].

5.1 Simplifying the Attack

The presented attack uses both monitor peers and attack peers. In theory, the attack could work without the attack peers, after all, the attack peers only speed-up the churn rate in the fast and high-capacity tiers of the victim. However, without attack peers, it is quite possible that the victim may rarely, if ever, choose the adversary's monitor peers: they might be too slow or not even ever measured by the victim.

In our attack, the attack peers more than doubles the churn in the fast tier, so we can expect that they cut down the time for the attack by about a factor of two. Using twice as many monitor peers would have been about as expensive; thus using a small number attackers to double the effect of the monitor peers represents a reasonable trade-off (as long as doubling the monitor peer effect is about as expensive as doubling the number of monitors). Using even more attackers would allow us to attack more peers; however, given that our knowledge about the fast tier of the victim is limited, the ratio between attack bandwidth and attack effect quickly gets worse. Using significantly fewer attacker peers also does not work — at 64 kb/s, a handfull of attackers might not cause a significant increase in the number of rejected tunnel requests.

5.2 Uni-directional vs. Bi-directional Tunnels

Because of the uni-directional nature of the I2P tunnels the attacker has to wait a longer time to observe the victim in the correct position for deanonymization; monitoring peers have to be in the correct position for both the inbound and the outbound tunnel. Thus, with a being the number of monitor peers in the fast tier of the victim, the probability for deanonymization in a fast tier of size s is $\left(\frac{a}{s}\right)^2$. For bi-directional tunnels the attacker would only need one peer in the correct position, resulting in a probability of $\frac{a}{s}$. This shows that the attacker has to wait t times longer to be in a position to confirm the victim in the uni-directional case when compared to the bi-directional case.

However, the chance of being correct about the deanonymization is different for both cases as well. To really compare the two styles, we need to consider the probability of deanonymizing the wrong peer (false-positive). For the uni-directional case, it is possible to accuse the wrong peer if the same ordinary tunnel participant happens to be adjacent to the victim for both the inbound and the outbound tunnel. For this to happen, the peer running the Eepsite has to first choose an ordinary, non-malicious peer for the first hop of the inbound tunnel. This happens with probability $\frac{a-s}{s}$. The same peer then also needs to be in the outbound tunnel, which happens with probability $\frac{1}{s}$. Additionally, the victim has to choose a monitor peer for the second hop of the inbound and the outbound tunnel, which happens with probability $\left(\frac{a}{s-1}\right)^2$. Combining all these

probabilities, the probability for false-positives with uni-directional tunnels is:

$$\frac{s-a}{s}\frac{1}{s}\left(\frac{a}{s-1}\right)^2 = \frac{s-a}{s^2}\left(\frac{a}{s-1}\right)^2 \approx a^2\frac{s-a}{s^4} \tag{1}$$

With bi-directional tunnels, the probability for a false-positive is higher, because any other peer between a monitor peer and the victim can be falsely accused. The overall probability for a bi-directional 2-hop tunnel is:

$$\frac{s-a}{s}\frac{a}{s-1} \approx a\frac{s-a}{s^2} \tag{2}$$

We now relate the probabilities for getting a false-positive for uni-directional and bi-directional tunnels. Dividing (1) and (2) we get $\frac{a}{s^2}$, which shows that the accuracy for the uni-directional case is up to s^2 times higher when compared to the bi-directional case.

This result indicates that uni-directional tunnels help an attacker due to the much higher certainty an attacker gets once the monitor peers are in the correct position. Considering this, using uni-directional seems to be a bad design decision; it makes the statistical evaluation for the adversary easier for the attack presented in this paper. However, it should be said that the false-positive rate of bi-directional paths is not tremendously high and might still be manageable for an attacker.

5.3 Suggestions for Improvements to I2P

While making the I2P network more robust towards DoS attacks is always a good goal, we do not believe that this would address the main problem: the ability of the adversary to influence peer selection. While I2P's heuristics seem to make it hard for an adversary to directly influence the metrics used for peer selection, influencing performance itself is likely always possible. Hence, a better solution would be to limit churn in the fast and high-capacity tiers. Furthermore, when the Tor network was subjected to a similar attack [10], guard nodes were introduced into the design of Tor; this would also help in the case of I2P.

Another problem is the fact that Eepsites allow repeated measurements, giving the attacker the opportunity to possibly collect data for many months. This problem is not unique to I2P, but also applies in exactly the same way to Tor's hidden services. The I2P developers are currently working on integrating a version of the secure distributed Tahoe filesystem [14], which may address this issue.

I2P could try to detect the specific attack, for example by watching for periodic requests. However, such a defense would likely not be effective because an adversary could use signals that are much harder to detect, for example using [6].

Most importantly, I2P should avoid leaking information about its fast tier by selecting random peers for the leases. This would make it harder for an adversary to determine which peers should be attacked with the DoS attack while maintaining performance advantages for the rest of the tunnel.

6 Conclusion

Biasing peer selection towards well-performing peers has previously been seen as a mostly theoretical issue. This work shows that combined with a limited, selective DoS attack on a few peers it enables an adversary to compromise the anonymity of long-running services. This work also shows that peers reacting quickly to changes in observed network performance can be detrimental to anonymity.

Acknowledgments

This work was funded by the Deutsche Forschungsgemeinschaft (DFG) under ENP GR 3688/1-1. We thank Katie Haus for her help with the figures and Nathan Evans, Matthew Wright and the anonymous reviewers for their feedback on the paper.

References

1. Borisov, N., Danezis, G., Mittal, P., Tabriz, P.: Denial of service or denial of security? How attacks on reliability can compromise anonymity. In: CCS 2007: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 92–102. ACM Press, New York (2007)
2. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: Proceedings of the 13th USENIX Security Symposium (August 2004)
3. Evans, N.S., Dingledine, R., Grothoff, C.: A practical congestion attack on tor using long paths. In: 18th USENIX Security Symposium, pp. 33–50. USENIX (2009)
4. Herrmann, M.: Privacy-Implications of Performance-Based Peer Selection by Onion-Routers: A Real-World Case Study using I2P. Master's thesis, Technische Universität München (2011)
5. Hopper, N., Vasserman, E.Y., Chan-Tin, E.: How much anonymity does network latency leak? ACM Transactions on Information and System Security 13(2) (2010)
6. Houmansadr, A., Borisov, N.: Swirl: A scalable watermark to detect correlated network flows. In: NDSS 2011 (2011)
7. Levine, B., Reiter, M., Wang, C., Wright, M.: Timing attacks in low-latency mix systems. In: Juels, A. (ed.) FC 2004. LNCS, vol. 3110, pp. 251–265. Springer, Heidelberg (2004)
8. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the xor metric, pp. 53–65 (2002)
9. Murdoch, S.J., Danezis, G.: Low-cost traffic analysis of Tor. In: SP 2005: Proceedings of the 2005 IEEE Symposium on Security and Privacy, May 2005, pp. 183–195. IEEE Computer Society Press, Washington, DC, USA (2005)
10. Øverlier, L., Syverson, P.: Locating hidden servers. In: SP 2006: Proceedings of the 2006 IEEE Symposium on Security and Privacy, pp. 100–114. IEEE Computer Society Press, Washington (2006)
11. Øverlier, L., Syverson, P.: Valet services: Improving hidden servers with a personal touch. In: Danezis, G., Golle, P. (eds.) PET 2006. LNCS, vol. 4258, pp. 223–244. Springer, Heidelberg (2006)

12. Peterson, L.: PlanetLab: Version 3.0. Tech. Rep. PDN–04–023, PlanetLab Consortium (October 2004)
13. Syverson, P., Goldschlag, D., Reed, M.: Anonymous Connections and Onion Routing. In: IEEE Symposium on Security and Privacy, Oakland, California, vol. 7, pp. 44–54 (1997)
14. Wilcox-O'Hearn, Z., Warner, B.: Tahao – the least-authority filesystem. In: Proceedings of the 4th ACM international workshop on Storage security and survivability. ACM Press, New York (2008)
15. zzz, Schimmer, L.: Peer profiling and selection in the i2p anonymous network. In: PET-CON 2009.1., TU Dresden, Germany (March 2009)