Locating Hidden Servers

Lasse Øverlier
Norwegian Defence Research Establishment
and Gjøvik University College
lasse.overlier@{ffi,hig}.no

Paul Syverson
Naval Research Laboratory
syverson@itd.nrl.navy.mil

Abstract

Hidden services were deployed on the Tor anonymous communication network in 2004. Announced properties include server resistance to distributed DoS. Both the EFF and Reporters Without Borders have issued guides that describe using hidden services via Tor to protect the safety of dissidents as well as to resist censorship.
We present fast and cheap attacks that reveal the location of a hidden server. Using a single hostile Tor node we have located deployed hidden servers in a matter of minutes. Although we examine hidden services over Tor, our results apply to any client using a variety of anonymity networks. In fact, these are the first actual intersection attacks on any deployed public network: thus confirming general expectations from prior theory and simulation.
We recommend changes to route selection design and implementation for Tor. These changes require no operational increase in network overhead and are simple to make; but they prevent the attacks we have demonstrated. They have been implemented.

1 Introduction

Tor is a distributed low-latency anonymous communication network developed by the Naval Research Laboratory and the Free Haven Project. It is currently the largest anonymity network in existence, with about 450 server nodes around the world at the time of writing. It is popular and highly recommended: it was rated one of the hundred best products of 2005 by PC World. Since 2004 Tor has also been used to underly services offered from hidden locations. These were introduced [13] as resistant to distributed DoS since they were designed to require a DDoS attack on the entire Tor network in order to attack a hidden server. Hidden servers have also been recommended for preserving the anonymity of the service offerer and to resist censorship. Specifically Undergroundmedia.org has published a guide to "Torcasting" (anonymity-preserving and censorship-resistant podcasting). And both the Electronic Frontier Foundation and Reporters Without Borders have issued guides that describe using hidden services via Tor to protect the safety of dissidents as well as to resist censorship. There have been several recent cases in the news in which anonymous bloggers have or have not been exposed and have or have not lost jobs, etc., as a result, depending on the policy of their ISP, the interpretation of laws by various courts, and numerous other factors. Recommendations for a technology to protect anonymous bloggers and other publishers, regardless of legal protection, would thus seem to be timely and encouraging.

The Tor developers are careful, however, to warn against using Tor in critical situations: upon startup the Tor client announces, "This is experimental software. Do not rely on it for strong anonymity." Nonetheless, with increasing high-profile recommendations to use Tor's hidden services for applications such as those above, it is important to assess

the protection they afford. In this paper we demonstrate attacks (not simulations) on the deployed Tor network that reveal the location of a hidden server. The attacks are cheap and fast: they use only a single hostile Tor node and require from only minutes to a few hours to locate a hidden server.

Although we examined hidden services on Tor, our results are not limited in principle to either hidden services or to Tor. They should apply to the hidden service design even if run on another underlying anonymity network and should apply to other clients using an anonymity network, not just to hidden servers.

We believe that ours are the first attacks that locate hidden servers, whether on hidden services on a deployed network or in simulation. Also, while there have been simulations and analytic results, we believe ours are the first published intersection attacks carried out on a deployed anonymity network.

In Section 2, we review previous related work. In Section 3, we describe the design of Tor's hidden services. In Section 4, we present various attacks and the experimental results from running them. In Section 5, we describe various countermeasures that might be taken to our attacks and the effectiveness of them. We also describe an implementation feature of Tor that our experiments uncovered and how to change it to better resist the attacks. In Section 6, we conclude with recommendations for simple-to-implement design changes to hidden services. These also should not add to the number of hops or otherwise increase overhead to the design, but they should resist our attacks. We have discussed both the implementation feature we uncovered and our recommended design changes with the Tor developers. As a result, the latest version of Tor is resistant to the attacks we present herein. Finally, we discuss open problems and future work.

2 Previous Work on Hiding Services and Anonymity

The earliest reference we can find to a system that hides the location of a service from those using it is Ross Anderson's Eternity Service [2]. Therein it is suggested that servers hold encrypted files, and these files are to be accessed by anonymous communication to prevent uncovering of the location of a server from which the file is being retrieved. Early presentations of onion routing from the same era described the use of onion routing to hide the location of an automated classification downgrader so that users of the service would not be able to attack it. Earlier still, Roger Needham noted the fundamental connection between anonymity and the inability to selectively deny service [19, 20], which was one of the motivating ideas in the Eternity Service. The idea of hiding the location of a document (or encrypted fragment of a document) also underlies many censorship-resistant publishing designs such as Free Haven [11] and Tangler [28].

Anonymous communication networks were introduced by David Chaum [9]. He described a network that distributes trust across multiple nodes that carry the communication. The design is of a public-key-based, high-latency anonymous communication network such as might be appropriate for email. It is not for use in bidirectional, low-latency communication, such as web traffic, chat, remote login, etc. Low-latency communication anonymity was introduced for ISDN [22], but made to anonymize within a group of users exchanging fixed and equal bandwidth with a local telephone switch rather than anonymizing within an Internet-wide group with diverse bandwidth needs such as occur in the just mentioned applications. The oldest anonymous communication system for web traffic is probably the Anonymizer [4]. Unlike the Chaum design, all traffic passes through a single proxy, making it a single point of failure and/or attack in many ways. Also unlike Chaum mixes, it does not actually delay and mix traffic. Traffic is processed FIFO. The Anonymizer is also probably one of the most widely used anonymization systems: they claim to have millions of users.

The first published, as well as the first deployed, distributed system for low-latency anonymous Internet communication was onion routing [16] in 1996, followed by the Freedom Network [8] from 1999 to 2001. The current version of onion routing, Tor [13], was deployed in late 2003, and hidden services using Tor were deployed in early 2004.

All of these low-latency anonymity systems work by proxying communication through multiple hops; at each hop the communication changes its appearance by adding or removing a layer of encryption (depending on whether it is traveling from the circuit originator to responder or vice versa). They all use public key cryptography to distribute session keys to the nodes along a route, thus establishing a circuit. Each session key is shared between the circuit initiator (client) and the one node that was given the key in establishing the circuit. Data that passes along the circuit uses these session keys. Both Freedom and Tor have a default circuit length of three nodes. For more details consult the above cited work. The Java Anon Proxy (JAP)/Web MIXes [6] is another popular system for diffused low-latency anonymity. However, unlike the others mentioned here, it works by mixing and by diffusing only trust and jurisdiction. It does not hide where communication enters and leaves the network. All communication that enters at one location leaves together (now mixed) at another location. As such it is not directly amenable to the hidden service design to be described presently. JAP has been deployed since 2000.

Hidden services in Tor, as described in the next section and in [13], rely on a rendezvous server, which mates anonymous circuits from two principals so that each relies only on himself to build a secure circuit. The first published design for a rendezvous service was for anonymous ISDN telephony [22] rather than Internet communication. As such it had very different assumptions and requirements from the rendezvous servers we describe, some of which we have already noted above. A rendezvous server for IRC chat was mentioned in [16]; however, the first detailed design for a rendezvous server for Internet communication was by Goldberg [15]. It differs in many ways from rendezvous servers as used by Tor's hidden services, but we will not discuss Goldberg's design further here.

There is much literature on attacking anonymous communication [3]. Rather than single out any of it here, we cite the relevant prior literature at appropriate points below. The current paper is the first to focus specifically on attacks for locating hidden services.

3 Location-hidden Services in Tor

One of the major vulnerabilities for a hidden service in Tor is the server's selection of the first and last node in the communication path. To a first approximation, if an adversary can watch the edges of a Tor circuit, then she can confirm who is communicating. This is because the low-latency requirements make it easy to confirm the timing signature of traffic flowing (in both directions) over the circuit. This is true whether the adversary controls the Tor nodes at the edges of the circuit or is just observing the links from those nodes to the initiator and responder. Actually, this vulnerability has always been alleged and assumed but never previously demonstrated. A byproduct of our analysis of hidden services is that we experimentally corroborate this traffic confirmation on Tor circuits. For hidden services, this means that the service is vulnerable in every communication path it sets up with a client if a member of the path can determine it is being used by a hidden service and that it is the first node in the path.

In order to see how our attacks that locate hidden servers are done we need to describe how the hidden service communication works. Fig. 1 shows a normal setup of this communication channel.

In the current implementation of Tor, a connection to a hidden service involves five important nodes in addition to the nodes used for basic anonymous communication over Tor.

- HS, the Hidden Server offering some kind of (hidden) service to the users of the Tor network, e.g. web pages, mail accounts, login service, etc.
- C, the client connecting to the Hidden Server.
- DS, a Directory Server containing information about the Tor network nodes and used as the point of contact for information on where to contact hidden services.
- RP, the Rendezvous Point is the only node in the data tunnel that is known to both sides.

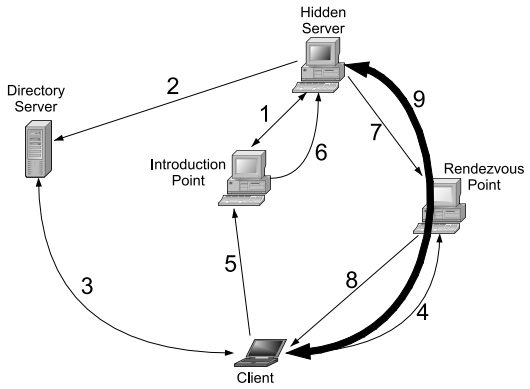

Figure 1. Normal use of hidden services and rendezvous servers

- IP, the Introduction Point where the Hidden Server is listening for connections to the hidden service.

A normal setup of communication between a client and a hidden service is done as shown in Fig. 1. All the displayed message flows are anonymized, i.e., they are routed through several anonymizing nodes on their path towards the other end, as described in Section 2. Every arrow and connection in the figure represents an anonymous channel consisting of at least two or more intermediate nodes. (Hereafter, we use 'node' to refer exclusively to nodes of the underlying anonymization network, sometimes also called 'server nodes'. Although we are considering the Tor network specifically, the setup would apply as well if some other anonymizing network were used to underly the hidden service protocol. The only exceptions are C and HS, which may be anonymization nodes or they may be merely clients external to the anonymization network.)

First the Hidden Server connects (1) to a node in the Tor network and asks if it is OK for the node to act as an Introduction Point for his service. If the node accepts, we keep the circuit open and continue; otherwise HS tries another node until successful. These connections are kept open forever, i.e., until one of the nodes restarts or decides to pull it down.[1] Next, the Hidden Server contacts (2) the Directory Server and asks it to publish the contact information of its hidden service. The hidden service is now ready to receive connection requests from clients.

In order to retrieve data from the service the Client connects (3) to DS and asks for the contact information of the identified service and retrieves it if it exists (including the addresses of Introduction Points). There can be multiple Introduction Points per service. The Client then selects a node in the network to act as a Rendezvous Point, connects (4) to it and asks it to listen for connections from a hidden service on C's behalf. The Client repeats this until a Rendezvous Point has accepted, and then contacts (5) the Introduction Point and asks it to forward the information about the selected RP.[2] The Introduction Point forwards (6) this message to the Hidden Server who determines whether to connect to the Rendezvous Point or not.[3] If OK, the Hidden Server connects (7) to RP and asks to be connected to the waiting rendezvous circuit, and RP then forwards (8) this connection request to the Client.

Now RP can start passing data between the two connections and the result is an anonymous data tunnel (9) from C to HS through RP.

From this we observe the following facts about the nodes in the network:

- C does not know the location (IP address) of HS, but knows the location of RP;
- HS does not know the location of C, but knows the location of RP;
- RP does not know the location of either C or HS, and he knows neither the service he is serving nor the content of the messages relayed through him;
- there are multiple (currently three) nodes between HS and RP and two nodes between C

[1] In Tor, any node in a circuit can initiate a circuit teardown.

[2] Optionally, this could include authentication information for the service to determine from whom to accept connections.

[3] This flow is over the same anonymous circuit as (1), similarly for (4) and (8).

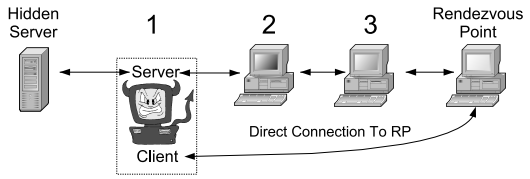

Figure 2. Vulnerable location of Attacker in communication channel to the Hidden Server

and RP, to hide traffic and create a degree of anonymity on both ends; and

- *any* member of the network which claims to offer stability can be used by HS to form an anonymous tunnel to RP, including C if it is a node in the anonymization network. This is the basis of our attacks.

4 Attacks and Experimental Results

We have done experiments using multiple attack methods in order to determine the IP address of the Hidden Server. We will here first describe the setup of the experiment and then four attack methods. The attacks can be carried out by an adversary that controls merely a single node in the network. Since anyone can run a Tor node simply by volunteering, this is trivial. (In fact the adversary need only run a "middleman" node, which never lets circuits exit the anonymization network. The burden of running a middleman node is typically less than that of running an exit node. Of the roughly 250 nodes in the Tor network at the time the experiments were done only about 100 allowed exit to port 80.) At the end we describe an accelerated attack using two compromised nodes.

Fig. 2 shows the scenarios that an attacker, hereafter Alice, wants to achieve in connections to the Hidden Server. Alice controls the Client and one node. Her goal is to control Node 1 of the circuit. Certain circuits will yield a match of traffic pattern with what is expected given when C sends to and receives from HS. Alice will look for such pattern matches among all active circuits through the node she owns. If she finds a match, then her node has been made part of the circuit between the Hidden Server and the Rendezvous Point as Node 1, 2 or 3. From this she will be able to determine a few facts. First she will know when she has the node closest to RP (Node 3) since she knows RP's IP address, and she can easily abandon the circuit and attack again. Second, if her node has an unknown IP address on both sides of the matching circuit, she knows she is either Node 1 connected directly to HS or Node 2 in the circuit. This enables her to use timing or statistical methods to determine her position as will be described later.

We will continue sampling data until we have enough to determine when Alice is connecting to the hidden service as Node 1 in the circuit towards RP, at which point we will know the Hidden Server's IP address.

Our attack description is obviously based on hidden services as deployed on Tor; however, the basic approach will identify a client of a low-latency, free-route anonymity network, not just hidden severs using Tor. These attacks should work on networks such as Freedom [8] or Crowds [23], despite their many differences from Tor. For systems such as Web MIXes [6], it is difficult to briefly say anything about either what a hidden service design over such a system would look like or about the relation to our attacks. On the one hand, becoming a node in the network is tightly controlled, and all circuits are through cascades (shared uniform fixed routes). Thus, our attacks would simply not be possible. On the other hand, much of the point of the attacks is to determine the point where connections enter and leave the anonymity network. In Web MIXes, this information is given, so there is no need to attack to obtain it.

4.1 Experimental Setup

Our experiments were conducted using two different hidden services running at client nodes connecting to the Tor network, one in Europe and one in the US. The services offered a couple of web pages and images, which were pulled down in dif-

ferent ways and with different timing patterns. The documents and images varied in size from 2KB to 120KB. The connection from the Client to the Hidden Server was done through a random Rendezvous Point (cf. Section 4.6), and the connection from the Client to the Rendezvous Point was shortened down to a path length of one. (This will be described more fully presently).

The Hidden Service was not offered, or known to, any other node in the network except the directory service. Only the Client knew about how to contact the service, so that all contact to and from the Hidden Server was either caused by the Client, or by the Hidden Server preparing to operate (making circuits and downloading new updates from the Directory Servers). This is not a limitation on the implications of the experimental results for publicly known and accessed hidden services: the timings of data are done with high enough precision so the possibility of two identical patterns from the same service routing through the same node at the exact same time is negligible.[4] No hidden server is likely to get two specially designed requests (like ours) from distinct clients and respond to them at the exact same time. Thus a false positive in our timing analysis is highly unlikely (Section 4.2).

The Client computer was also announced as a middleman node, i.e. not having connections out of the anonymity network, and this node is where all Alice's sampling of data takes place. By using the node both as a server inside the network and as the Client asking for the web pages from the Hidden Server, the attacker is able to get precise timing without having to externally synchronize the time with another node. This server node in the Tor network had to use a logging mechanism when sampling the active circuits during the attacks. In order to avoid reference to the correct IP address during the timing analysis we converted the IP addresses by use of a simple prefix preserving scheme. If we were to use permanent logging of data, we would use a better and more secure pseudonomizing IP logging scheme [21].

The attacker must also make some minor changes to the application code running at the Client node in order to enable and strengthen the attacks:

- Alice's Client will connect directly, i.e. in one hop, to the Rendezvous Point to shorten the path and latency of traffic between the Client and the Hidden Server, thereby making it easier to set up and correlate the traffic patterns.
- Alice's Client will tear down the circuit to a Hidden Server after each pattern is successfully communicated. This will disable reuse of circuits and force the construction of a new circuit on the next connection request.
- In addition to being the Client, Alice is also running as a server middleman node participating in the network and carrying traffic for the other nodes. She will maintain a list of active circuits (routed connections) and try to correlate the generated circuit data with all the other circuits to find out if she is carrying the same traffic data as both Client and as a server node.
- Alice's server node will report a false higher uptime and the maximum network bandwidth to the directory server in order for other nodes to trust it for their circuits. This is still possible as there is (yet) no method for keeping reliable track of uptime at the different servers.

Once this is implemented, Alice is ready to use the methods of attack described below.

4.2 Timing Analysis

The attacker uses the logged timing data and direction information from the generated data set and the sampled data set (from each circuit active in that period of time) to accomplish two different things:

1. Positively identify that Alice's node is made a part of a circuit; and

[4] One reason for not doing the experiment on a publicly known server in the Tor network, is of course the possible legal implications. In addition, not wanting to cause harm to the project and its participants, we avoided announcements until there were countermeasures available and deployed.

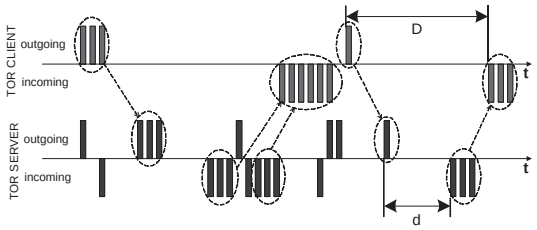

Figure 3. Example of data sets, matching and response times

2. If (1) is true; determine at which position in the circuit she is located.

To identify if the generated data is found within one of the sampled data sets, Alice is faced with a comparison of one sampled data set, the generated set done by the Client, to all of the sampled data sets done by the server. For all connections to the Hidden Server there were a few hundred circuits active at Alice's Tor node during each of the sample periods.

Our match confirmation is an extended version of the packet counting attack described by Serjantov and Sewell [24]. In addition to basic counting of cells, we also make use of precise timing information of when cells were received and transmitted, and the direction of each individual cell passing in order to determine a circuit match. An example is depicted in Fig. 3. Alice uses the direction of traffic in addition to the timing of the cells' arrival to match *all* outgoing and incoming traffic in the generated data set. Notice that there is also noise occurring in the sampled data set. We compare our known data to *one* other specific set at a time, and our algorithm only checks if the generated data *may* be a part of the sampled data. Therefore, it makes no estimate of how probable the match is.

Alice is also able to separate when there is a single match and when there are multiple matches in a data set. There is a potential for multiple matches in a set, for example, if a circuit is carrying lots of traffic in both directions, we will probably have a timing "match" due to the data load. In the attack Alice knows that only one attack circuit is set up at a time, and each attack circuit is set up for only Alice's requests at that time. So, she can use the amount of traffic relayed through the attack circuit as a parameter. The small overhead and extra information from setting up the tunnelled connections etc., should not be more than a few cells, something our experiments confirm. Therefore the attacker may discard the samples that are more than a few percent[5] larger than the generated set.

Multiple matches could also be a possible result in a future scenario where the circuits may be used to carry data for different clients. In this case the attacker must try to retrieve a list of all possible matches of the generated data within the sampled set and should then be able to use correlation techniques on the timing data to calculate a probability of the best match. We have not tested this part as it would require major changes in functionality for the deployed Tor network.

4.3 Service Location Attack

First we look at two different situations, the *Server Scenario* and the *Client Scenario*, based on whether the hidden service is located on a node within the anonymity network, or on a client using the network but not participating as a network node.

The Client Scenario is most often used when it is desired that HS not be listed in the directory service as a participating server node of the network. Another reason for this scenario is that the user may be unable to set up a node directly reachable from the Internet (e.g., it must be located behind a firewall) but still wants to offer his service.

The Server Scenario is most often used to hide the service traffic within all the other traffic running through the server node. This is often regarded as a reasonable effort to improve the cover of traffic originating at the node.

A problem with the Server Scenario is that it is possible to correlate information about availability of a service and the availability of the nodes listed in the Directory Service. E.g. we poll each listed

[5] Actually the overhead is normally less than 10 cells, but the extra margin has not given any false positives yet.

Table 1. Experimental results of our attacks.

	Sample time	Time to first match	Circuits completed	Matched circuits	Largest single IP	Second largest
Server 1	7.8h	15 min	676	37	46%	5%
Server 1	6.8h	3 min	432	26	54%	7%
Server 2	4.9h	28 min	447	31	71%	3%
Server 2	10.6h	3 min	990	56	54%	7%

server every five minutes and correlate the lists of active servers when we are able and unable to contact the hidden service.

Our attack is based on the availability of a directory service containing a public list of all server nodes in the anonymity network. From this list the attacker will immediately be able to tell the difference between connections from mere clients and connections from network nodes. If a service is located at a client outside the anonymizing network, Alice will know both this and the client's IP address, as soon as she has a positive match in the timing analysis (Section 4.2) on a connection originating from outside of the currently listed server nodes. There is no other way the hidden service can communicate with the attacker's node from outside the Tor network unless this is the actual location of the hidden service, or its point of contact, e.g., a firewall hiding internal addresses, an IPSec tunnel endpoint, etc.

Experimental Results: Our results confirmed this by a simple matching of the IP addresses in the sampled circuits against the list of known servers. Both of our Hidden Servers were run on Client nodes and were easily confirmed as the source of the service. The time for the attack until successful identification of the IP address in the four tests of the experiment are shown in Table 1 under the column "Time to first match".

So, if the hidden service is located at a client of the anonymity network an attacker will find it *in a matter of minutes* using only one node in the network, but if the service is located at a network node we will have to use another method.

4.4 The Predecessor Attack

The current implementation of the network is vulnerable to the predecessor attack [31]. This is a form of intersection attack. Since intersection attacks treat the intervening anonymity network as a black box, they are a threat to any anonymity network. Like other intersection attacks, the predecessor attack has been shown to be devastating in theory and simulation against various anonymity networks but has never before been demonstrated on a live network. Roughly, the predecessor attack looks at repeated connections suspected to be to (from) the same correspondent and looks at intersections of predecessor nodes to see which occurs most often. Our use of this attack is based on the assumption that the attacker is able to positively identify the actual streams of data to and from the client in other circuits, e.g. by using the Timing Analysis described in Section 4.2.

In the case of Hidden Servers and using our scenario of attack, the Predecessor Attack becomes trivial. Alice can now make statistics of the IP addresses that contacted the server in the cases where a positive traffic-pattern match was found. By selecting only circuits where there has been a match, and using an m node path towards RP, one single IP address will occur in around $\frac{1}{m}$ of these connections when HS is selecting its first node. The attacker will then easily identify the IP address of the Hidden Server as long as m is significantly smaller than the number of nodes in the network.

Experimental Results: Our results confirmed the theory of the predecessor paper [31]. Sorting out the possible circuits based on timing informa-

tion and then statistically examining the connecting IP addresses we immediately found the expected results from the predecessor attack. In every test[6] we found that around 50%, or more, of all connections identified as a part of the stream were made from a single IP address, as shown under "Largest single IP" in Table 1.[7]

From the experimental results we can also conclude that we need far less data to pinpoint the location of the Hidden Server than we gathered. A rough estimate is that within the order of an hour or two we should have a positive match of the location of the hidden service using the predecessor attack.

4.5 Distance Attack

If there is no information in IP address statistics (e.g., due to mixing of traffic or using other countermeasures), an attacker must use other techniques to locate the hidden service.

When Alice has a dataset that matches the generated set of data, she can look at the response times in the communication with the service. The attacker times the periods where the sampled data switches from outgoing to incoming traffic, *round trip time*, enabling the calculation of a rough estimate measuring the distance to the Hidden Server. These periods are marked in the example in Fig. 3, with D for the Client's response times and d for the round trip time at the participating node. By grouping nodes based on measured round trip times, the attacker is able to find some groups of nodes closer to the Hidden Server than others.

Experimental Results: Our results confirmed the assumptions of the distance attack. Using the data from the experiment we could see a clear correlation between the response times and the distance from the Hidden Server. When the Hidden Server was local it was of course easy to find a match showing the attacker's node next to the service (order of 100-1000 compared to the other connections). But even when our service was located on computers on the other side of the globe we could still statistically observe when Alice was connecting directly to the Hidden Server. The round trip times were an order of two or more larger for the nodes not adjacent to the Hidden Server, as shown in Fig. 4. The lower line represents the average response times for 52 samples of the nodes closest to the Hidden Server, and the upper line is for the other 35 samples in our set where Alice is located at Node 2 in Fig. 2. Due to the previously mentioned implementation feature of Tor we were unable to find data when Alice is located as Node 3, cf. Section 5.1.

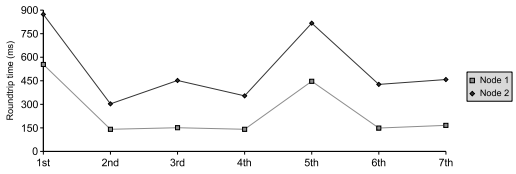

Figure 4. Average round trip times at seven locations in a sample of circuits

4.6 Owning the Rendezvous Point

By extending adversary resources and using two nodes in the network, it is possible for Alice to run attacks where she owns the Rendezvous Point. This will significantly enhance the attack.

Only knowing RP's IP address will give the attacker knowledge of when she is the last node (Node 3 in Fig. 2) in the circuit out from the Hidden Server. Selection of the Rendezvous Point is done by the Client and enables Alice to choose one of her nodes as RP, while still leaving her other node free to be chosen by HS for the circuit to RP. This allows Alice to tell when she is the second to last node in the circuit as well (since both C and RP are connected to the same node). This implies that if the path length is three before connecting to HS (as currently implemented) the attacker is able to deter-

[6] This result is for every test running without the use of "helper" guard nodes. Cf., Section 5.4.

[7] Given three nodes between HS and RP, we would expect to find a common predecessor IP address in only about 33% of matching connections. The discrepancy is due to an implementation feature of Tor uncovered by our experiments. Cf., Section 5.1.

mine the instance where she is Node 1, thus directly revealing the IP address of the Hidden Server. The speed and accuracy of the attack is then greatly improved, and the result will be as fast as in the Service Location Attack—except that this own-the-rendezvous attack will identify services located at network servers as well as those located at clients.

5 Countermeasures: More Hidden Services

5.1 Allowing Middleman nodes to connect to Rendezvous Points

This first point is really about an implementation feature of Tor's hidden services that facilitates our attacks rather than a limitation of the hidden services system design. But since changing the feature does slow down the attacks, we list it here.

To save time, all Tor clients (including hidden servers) establish circuits offline, i.e., while awaiting service requests. Upon receiving a rendezvous request and an RP location, HS extends such circuits to RP. Tor clients *not* operating as hidden services typically will need circuits that terminate at nodes that allow exit from the Tor network on common ports, such as those for http, ssh, and https. Almost all of the new "stand-by" circuits established thus go to a node that allows such exit, which seems quite reasonable considering normal client use. A hidden server should similarly always have at least one circuit available at a random node of the network ready for connection to the Rendezvous Point.

This creates an advantage for our attacker. By running in middleman mode (never allowing circuits to exit the Tor network at that node) she both reduces the overhead of running a node and guarantees that whenever her network node is used between HS and RP, it will almost[8] always be in the first or second position, which increases the efficiency of her attack. Our experiments uncovered this undocumented feature of the Tor implementation. It is a trivial change to allow the third node from HS to RP to be any node not just an exit node. This has now been implemented by the Tor developers and is available in the latest versions.

[8]We had no occurrence of being Node 3 in the sample sets described in this paper.

5.2 Dummy traffic

In anonymous communication, dummy traffic is a countermeasure to traffic analysis that is often initially suggested. However, dummy traffic is expensive, and, despite research, it has yet to be shown that dummy traffic defeats any active attacks on low-latency systems unless the system will also bring most or all of the network to a stop in response to one non-sending client (as in Pipenet [10]). Since this makes it trivial for any user to bring down the network, it is generally seen as a price few would pay for anonymity, which means that even those who would pay it would be hiding in a very small anonymity set [5, 1]. While some dummy traffic schemes have been proposed [7, 17], that attempt to address some active attacks, no fielded low-latency systems currently use dummy traffic. In light of our attacks, we describe why dummy traffic would be an especially ineffective countermeasure for hidden services.

In our attack scenario, Alice can develop a list of candidate circuits by labeling any circuits through her network node that show a response from the server shortly after she sends a request with an RP address to HS. This would potentially include many false positives. She can then induce a timing signature in her network node on all responses from a server on candidate circuits. This can be done exactly in the manner of the timing signature used by Murdoch and Danezis [18], except that our attacks do not require collusion by the external server. Alice's client then simply looks for the same timing signature. The important thing to note is that no dummy traffic scheme could prevent this. If dummy traffic is sent all the way to the client, Alice can of course detect it since she controls the client. If dummy traffic is sent by HS to some node between Alice's node and Alice's client, this will result in some differences between the induced signature and the one seen by the client. Nonetheless, for low-latency traffic this strong signature would

clearly remain easily identifiable. In the experiments of [18], the corrupt server would send for between 10 and 25 seconds then stop sending for between 30 and 75 seconds. This was detected indirectly in those experiments by interference with external probes of Tor nodes. In our case, Alice would have direct access to the circuits. This also means that the attack would scale to current network sizes and beyond.[9] Note also that no existing dummy scheme would even affect the signature of traffic sent from Alice's client to the Hidden Server through Alice's node. While this traffic is typically much lower volume it can still be used to identify the circuit.

5.3 Extending the Path from Hidden Server to Rendezvous Point

As described in Section 4.6 and illustrated in Fig. 2, if Alice owns at least two nodes she can have her client name one of them as the Rendezvous Point. If her other node is chosen by HS as Node 2, then she will be able to confirm this immediately with high confidence because both her nodes will be connected to Node 3. And as before, if Alice were connected as Node 3, then she would also know this. This means that Alice can easily know when she has a circuit match being Node 1, which is especially significant if HS is configured as in the server scenario of Section 4.3. However, this also means that Alice can more quickly abandon circuits when she does not have Node 1 position, speeding up the attack. It also allows rapid identification of guard nodes (cf., Section 5.4).

A simple countermeasure to this is to allow HS to extend the path length, l, to RP by one hop. The attacker owning the Rendezvous Point will now be able to determine when she is located as Node 3 or Node 4, but unable to differentiate between the positions 1 and 2, forcing Alice to use the predecessor or service location attack. Extending the path will also slow down the predecessor attack and the timing analysis by a factor of $1/l$ since that is the frequency with which Alice's node will be chosen as Node 1 within the matching circuits. So this countermeasure only causes a minor effect on the speed of the predecessor attack, and has no effect on the location attack.

As an alternative, we could allow HS to choose RP. This would be a minor code change to the Tor hidden services protocol. Whether adding a node before the Rendezvous Point or allowing HS to choose RP, this would also seem to imply a longer path between client and HS than the current default Tor HS protocol, i.e., seven nodes vs. six. This would also create an easier attack if our techniques were used to locate the client, which would then require its own countermeasure.

[9] The attacks of [18] required probing the entire network and were done on a network an order of magnitude smaller than the current Tor network. It is an open question whether they would scale to the current one.

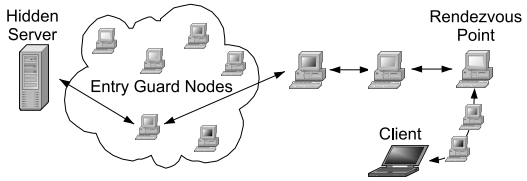

Figure 5. Use of Entry Guard Nodes

5.4 Using Entry Guard Nodes

All of our attacks rely on Alice being able to force the Hidden Server to create new circuits until she can cause it to create a circuit that first connects directly to her node. What if this could never happen, or if the rotation of first nodes in the circuit were slowed down? This would prevent or substantially slow our attacks. This is the motivation behind *entry guard nodes* (or simply *entry guards*) a concept introduced by Wright et al. [30].[10] That work looked at attacks on various anonymous communication systems, but it did not consider at all the specific concerns of hidden services. The basic idea of a helper node in [30] was to always choose a single node as the first node in a communication. If this is compromised, then that end of your circuit is

[10] Wright et al. named these nodes *helper nodes*, which we have found to be a too general expression.

Table 2. Experimental results when Hidden Server is using Entry Guard Nodes.

	Total circuits completed	Matched circuits	Largest single IP	Second largest	Third largest
Test 1	292	8	7	1	0
Test 2	106	6	5	1	0
Test 3	296	13	12	1	0
Test 4	292	10	4	3	3

always compromised. However, if it is not compromised, then the attacks we have described cannot work because Alice will never own the node adjacent to HS on the rendezvous circuit. (For a circuit initiator to better hide the responder, Wright et al. also considered helper nodes as the last node in the circuit as well as the first.)

Tor design has long allowed a specified list of entry nodes (and exit nodes) which, when specified, will require all circuits to enter (resp. exit) the network through nodes in that set, as illustrated in Fig. 5. This can be set as either a preference request or as a strict requirement [26]. The effectiveness of using these as entry guard nodes to counter predecessor attacks is noted by the Tor designers as an open research problem [12].[11] We now explore the idea of using entry guard nodes specifically to improve the protection of hidden servers.

There are several parameters and options possible in choosing entry guard nodes. The first parameter is the *entry guard set size*, i.e, the number of nodes that HS will use as entry guards. The smaller the set, the less risk that Alice owns a node in it; however the greater the chance that all the nodes in the set will be subjected to monitoring or unavailable from either failure or attack. As already noted, entry guard nodes may be *preferred* or *strictly required*. As a refinement, there may be a successively preferred set of entry guards. There may be a *single layer* of entry guard nodes, i.e., nodes chosen to be immediately adjacent to HS, or they may be *layered*, i.e., some guard nodes are chosen to be used for the first hop, some for the second, and possibly further. Finally, they may be *chosen at random* or *chosen based on trust or performance*.

Each of these choices is orthogonal, so each of the combinations of choice will lead to systems with different properties. For space, we will limit discussion to some of the more salient combinations.

Choosing a small set of entry guard nodes that are both permanent and strictly required could lead to a higher percentage of service failures, either by accident or by design (assuming a very powerful adversary, with the capability to DoS all entry guard nodes). If a permanent set is simply a preference, then DoS of all entry guard nodes could lead back to our attacks if the guard nodes can be kept down long enough. Of course this assumes that entry guards can be identified by the attacker. We ran our attacks on a hidden server that had chosen three entry guard nodes.

Experiment - Attacking Entry Guard Nodes: Letting the Hidden Service use three permanent, preferred entry guards we found that these nodes combined represented **all** identified connections through Alice's node, as shown in Table 2. A quite unexpected result, but caused by the implementation feature in Tor described earlier: we were never Node 3, only Node 2 (Node 1 being the entry guard).

As in our previous experiments, identifying the entry guard nodes through our attacks never took more than a few hours.

Backup guard nodes: Suppose there is a short list of entry guard nodes that is preferred (e.g., three nodes) and a longer list of guard nodes to be used as backup (e.g., nine) if those are not available. If

[11] It is also possible to set an exit node preference in the URL for specific HTTP requests [27].

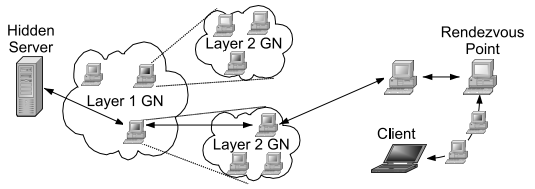

Figure 6. Use of Layered Entry Guard Nodes

the adversary has the capability of keeping say four nodes at a time offline, then it can cause HS to use other nodes in the Node 1 position than those on the short list. But, all that will accomplish is causing HS to rotate to three new nodes from the longer list as primary guards. Alice can cause rotation of circuits but only through a still relatively small set of entry guard nodes, and this only through sustained attacking. We can however make it more difficult for Alice to find the entry guard nodes at all via our attacks.

Layering guard nodes: Suppose, e.g., that HS has a set of three entry guard nodes from which to choose Node 1, and for each of these has a set of three guard nodes from which to choose Node 2, as illustrated in Fig. 6. As before, Alice can only successfully find HS if she owns one of these three layer 1 entry guard nodes. But if she does not, in order to even identify one of those layer 1 nodes to attack she must own one of the three layer 2 guard nodes associated with that node.

Layering guard nodes would require much more substantial changes to the Tor code than the experiments we have already run, albeit probably fairly straightforward changes. We have thus not had a chance to conduct experiments on either layering or backup guard node configurations. However, a version of backup guard nodes has recently been implemented in Tor in response to our results. At the time of writing, by default each client running the latest Tor code chooses three nodes as initial preferred entry guards. When the available entry guard set shrinks below two nodes, two more nodes are added to the set. However, the software keeps track of when a node enters the set and prefers to choose entry nodes for circuits that were in the set sooner. Nodes are only deleted from the set when they have been unreachable for an extended period (currently one month).

Nonrandom choice of entry guard node sets: To avoid circuit rotation simply from failed entry guard nodes it might seem that it is best to choose as guard nodes those that have the best uptime, and perhaps bandwidth. This is, however, subject to abuse since adversaries may run highly reliable, highly performing nodes in order to increase their chances of being chosen as entry guard nodes. And, this is especially easy to abuse in the current Tor directory statistics in which nodes report their own performance. This is a specific instance of a more general problem in trying to build reliable anonymous communication. One possible solution is to order node performance and reliability but then to choose from a large enough set in this order that the adversary is unlikely to be able to substantially alter the chances of being chosen as an entry guard node. Dingledine and Syverson described this strategy to form a reliable anonymous communication network of mix cascades [14].

Another possibility is to choose entry guard nodes based on trust in the node administrator. It is difficult to attach probabilities to Alice's being trusted by the Hidden Server administrator, or perhaps more likely, to compromise a node run by someone trusted by the Hidden Server administrator. (Trust in honesty should not be confused with trust in competence.) Perhaps a greater concern is that common properties of the administrators of chosen entry guard nodes (e.g., they are all family relatives) may lead an adversary to form a hypothesis of who is running HS, which may then lead to attacks unrelated to use of the network per se. Here the layering approach described above may prove useful. If the layer 1 nodes are personally trusted, and the layer 2 nodes are chosen as random sets, then it becomes more difficult for an adversary to discover the set of entry guard nodes and thus to correlate external properties.

6 Conclusion

Our results show that Tor's location-hidden servers are not really hidden—or rather they were not really hidden prior to the recent introduction of guard nodes as countermeasures to our attacks. Using our attacks, all an attacker needed was one compromised node in the network and the "Hidden Server" was identified.

We have demonstrated that an attack with one compromised node in the anonymity network takes only minutes if the service is located at a client, or a couple of hours when located on a server node. By using two nodes in the network it only takes minutes to find the Hidden Server regardless of where it is located.

We have also argued that neither dummy traffic nor extending the path length from the Hidden Server to the Rendezvous Point will protect against all of our attacks. However, requiring hidden services to always use entry guard nodes, which are currently available as a general option in the Tor code, greatly reduces the probability of successful attacks against a hidden service.

Using random entry guard nodes may still leave the Hidden Server vulnerable to our attacks if the attacker is powerful enough to completely deny service to a small sets of nodes or to compromise them by physical or other means. But, using *backup guard nodes* and/or *layering guard nodes* will significantly slow down even such an attacker.

Using random selection of backup and layering entry guard nodes will be an improvement, but as in all Tor circuits, someone connecting through random nodes will always be compromised if an attacker owns just two nodes [25]. Using the backup and layering techniques in combination with a non-random selection, e.g. based on some kind of trust, or experience, with the nodes, may slow the attack even more or may even prevent it entirely.

We have demonstrated attacks that surprisingly require just one or two hostile nodes. What is possible by an adversary that controls several nodes, or even, e.g., two percent of the network? We will investigate this in future work. Other future work includes testing implemented countermeasures for vulnerabilities using the described attack scenarios, as well as new ones. We will also be investigating improved performance by shrinking the path length between the Client and the Hidden Server. We speculate that it may be possible to do so with adequate security when using our suggested countermeasures, and possibly others. We will also turn our attack on its head: testing to locate a client by having an attractive hidden service.

7 Acknowledgements

Thanks to Tor developers Roger Dingledine and Nick Mathewson for talks on the functionality of Tor. Thanks to the reviewers for their constructive comments on how to improve and emphasize the important points of the paper. This work supported by ONR.

References

[1] A. Acquisti, R. Dingledine, and P. Syverson. On the economics of anonymity. In R. N. Wright, editor, *Financial Cryptography, 7th International Conference, FC 2003*, pages 84–102. Springer-Verlag, LNCS 2742, 2003.

[2] R. J. Anderson. The eternity service. In *Proceedings of Pragocrypt '96*, 1996.

[3] Bibliography of anonymous communication literature. http://freehaven.net/anonbib/.

[4] Anonymizer. http://www.anonymizer.com/.

[5] A. Back, U. Möller, and A. Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In I. S. Moskowitz, editor, *Information Hiding (IH 2001)*, pages 245–257. Springer-Verlag, LNCS 2137, 2001.

[6] O. Berthold, H. Federrath, and S. Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 115–129. Springer-Verlag, LNCS 2009, July 2000.

[7] O. Berthold and H. Langos. Dummy traffic against long term intersection attacks. In R. Dingledine and P. Syverson, editors, *Proceedings of Privacy Enhancing Technologies workshop (PET 2002)*. Springer-Verlag, LNCS 2482, April 2002.

[8] P. Boucher, A. Shostack, and I. Goldberg. Freedom system 2.0 architecture. White paper, Zero Knowledge Systems, Inc., December 2000.

[9] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1981.

[10] W. Dai. Pipenet 1.1. Usenet post, August 1996.

[11] R. Dingledine, M. J. Freedman, and D. Molnar. The Free Haven Project: Distributed anonymous storage service. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, July 2000.

[12] R. Dingledine, N. Mathewson, and P. Syverson. Challenges in deploying low-latency anonymity (DRAFT). Unpublished Manuscript. http://tor.eff.org/cvs/tor/doc/design-paper/challenges.pdf.

[13] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[14] R. Dingledine and P. Syverson. Reliable MIX Cascade Networks through Reputation. In M. Blaze, editor, *Financial Cryptography, 6th International Conference, FC2002*, pages 253–268. Springer-Verlag, LNCS 2357, 2002.

[15] I. Goldberg. *A Pseudonymous Communications Infrastructure for the Internet*. PhD thesis, UC Berkeley, December 2000.

[16] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Hiding Routing Information. In R. Anderson, editor, *Proceedings of Information Hiding: First International Workshop*, pages 137–150. Springer-Verlag, LNCS 1174, May 1996.

[17] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright. Timing attacks in low-latency mix-based systems (extended abstract). In A. Juels, editor, *Financial Cryptography, 8th International Conference, FC 2004*, pages 251–265. Springer-Verlag, LNCS 3110, 2004.

[18] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of Tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*. IEEE CS Press, May 2005.

[19] R. M. Needham. Denial of service. In *CCS '93: Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 151–153, New York, NY, USA, 1993. ACM Press.

[20] R. M. Needham. Denial of service: an example. *Communications of the ACM*, 37(11):42–46, 1994.

[21] L. Øverlier, T. Brekne, and A. Årnes. Non-expanding Transaction Specific Pseudonymization for IP Traffic Monitoring. In *Cryptology and Network Security: 4th International Conference (CANS 2005)*, pages 261–273. Springer-Verlag, LNCS 3810, December 2005.

[22] A. Pfitzmann, B. Pfitzmann, and M. Waidner. ISDN-MIXes: Untraceable communication with very small bandwidth overhead. In *Proceedings of the GI/ITG Conference on Communication in Distributed Systems*, pages 451–463, February 1991.

[23] M. Reiter and A. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1), June 1998.

[24] A. Serjantov and P. Sewell. Passive attack analysis for connection-based anonymity systems. In E. Snekkenes and D. Gollmann, editors, *Computer Security – ESORICS 2003*, pages 116 – 131. Springer-Verlag, LNCS 2808, October 2003.

[25] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. Towards an Analysis of Onion Routing Security. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: Workshop on Design Issue in Anonymity and Unobservability*, pages 96–114. Springer-Verlag, LNCS 2009, July 2000.

[26] *Tor Manual*. http://tor.eff.org/tor-manual.html.en.

[27] Tor wiki. http://wiki.noreply.org/noreply/TheOnionRouter/TorFAQ.

[28] M. Waldman and D. Mazières. Tangler: a censorship-resistant publishing system based on document entanglements. In *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS 2001)*, pages 126–135, November 2001.

[29] M. Wright, M. Adler, B. N. Levine, and C. Shields. An analysis of the degradation of anonymous protocols. In *Proceedings of the Network and Distributed Security Symposium - NDSS '02*. IEEE CS Press, February 2002.

[30] M. Wright, M. Adler, B. N. Levine, and C. Shields. Defending anonymous communication against passive logging attacks. In *IEEE Symposium on Security and Privacy*, pages 28–41. IEEE CS Press, May 2003.

[31] M. K. Wright, M. Adler, B. N. Levine, and C. Shields. The predecessor attack: An analysis of a threat to anonymous communications systems. *ACM Trans. Inf. Syst. Secur.*, 7(4):489–522, 2004. A preliminary version of this paper appeared in [29].