Browser-Based Attacks on Tor

Timothy G. Abbott, Katherine J. Lai, Michael R. Lieberman, Eric C. Price
{tabbott, k_lai, mathmike, ecprice}@mit.edu

Abstract. This paper describes a new attack on the anonymity of web browsing with Tor. The attack tricks a user's web browser into sending a distinctive signal over the Tor network that can be detected using traffic analysis. It is delivered by a malicious exit node using a man-in-the-middle attack on HTTP. Both the attack and the traffic analysis can be performed by an adversary with limited resources. While the attack can only succeed if the attacker controls one of the victim's entry guards, the method reduces the time required for a traffic analysis attack on Tor from $O(nk)$ to $O(n + k)$, where n is the number of exit nodes and k is the number of entry guards. This paper presents techniques that exploit the Tor exit policy system to greatly simplify the traffic analysis. The fundamental vulnerability exposed by this paper is not specific to Tor but rather to the problem of anonymous web browsing itself. This paper also describes a related attack on users who toggle the use of Tor with the popular Firefox extension Torbutton.

1 Introduction

The Internet was not designed with anonymity in mind; in fact, one of the original design goals was accountability [3]. Every packet sent by established protocols identifies both parties. However, most users expect that their Internet communications are and should remain anonymous. As was recently highlighted by the uproar over AOL's release of a large body of "anonymized" search query data [10], this disparity violates the security principle that systems meet the security expectations of their users. Some countries have taken a policy of arresting people for expressing dissident opinions on the Internet. Anonymity prevents these opinions from being traced back to their originators, increasing freedom of speech.

For applications that can tolerate high latencies, such as electronic mail, there are systems that achieve nearly perfect anonymity [1]. Such anonymity is difficult to achieve with low latency systems such as web browsing, however, because of the conflict between preventing traffic analysis on the flow of packets through the network and delivering packets in an efficient and timely fashion.

Because of the obvious importance of the problem, there has been a great deal of recent research on low-latency anonymity systems. Tor, the second-generation onion router, is the largest anonymity network in existence today.

In this paper we describe a new scheme for executing a practical timing attack on browsing the web anonymously with Tor. Using this attack, an adversary can identify a fraction of the Tor users who use a malicious exit node and then leave

a browser window open for an hour. With current entry guard implementations, the attack requires the adversary to control only a single Tor server in order to identify as much as 0.4% of Tor users targeted by the malicious node (and this probability can be increased roughly linearly by adding more machines). The targeting can be done based on the potential victim's HTTP traffic (so, for example, one could eventually identify 0.4% of Tor users who read Slashdot).

2 How Tor Works

Tor [5] is an anonymizing protocol that uses *onion routing* to hide the source of TCP traffic. Onion routing is a scheme based on layered encryption, which was first developed for anonymizing electronic mail [1]. As of December 15, 2006, Tor was used by approximately 200,000 users and contained about 750 nodes (also sometimes referred to as "servers" or "routers") [4].

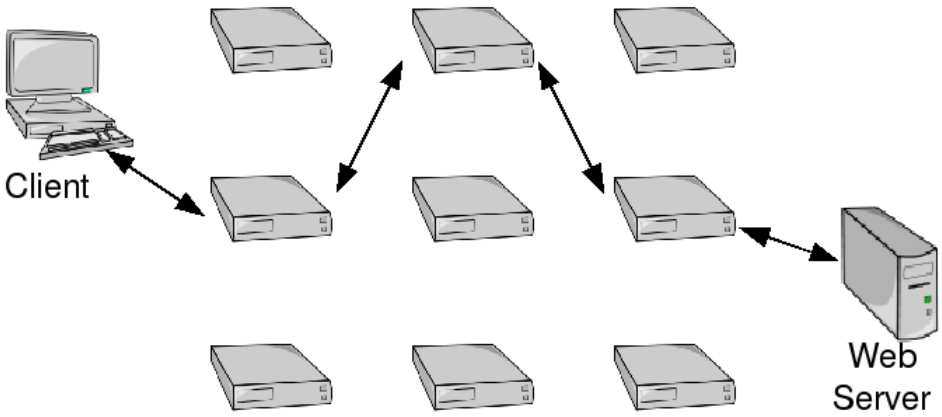

Fig. 1. A Tor circuit. The client chooses an entry node, a middle node, and an exit node, allowing the exit node to fetch content from a web server.

In Tor, a client routes his traffic through a chain of three nodes that he selects from the Tor network, as shown in Figure 1. A client constructs this path of nodes, or "circuit", by performing Diffie-Hellman handshakes with each of the nodes to exchange symmetric keys for encryption and decryption. These Tor nodes are picked from a list of current servers that are published by a signed

directory service.[1] To send TCP data through the circuit, the client starts by breaking the stream into fixed sized cells that are successively encrypted with the keys that have been negotiated with each of the nodes in the path, starting with the exit node's key and ending with the entry node's key. Fixed size cells are important so that anyone reading the encrypted traffic cannot use cell size to help identify a client [7][9].

Using this protocol, the entry node is the only node that is told the client's identity, the exit node is the only node that is told the destination's identity and the actual TCP data sent, and the middle node simply exchanges encrypted cells between the entry node and the exit node along a particular circuit. The nodes are selected approximately randomly using an algorithm dependent on various Tor node statistics distributed by the directory server, some client history, and client preferences.

3 Related Work

In May 2006, S. Murdoch and G. Danezis discussed how a website can include "traffic analysis bugs"—invisible signal generators which are used to shape traffic in the Tor network [11]. Our attack uses similar signal generators to attack a Tor client. We rely on the ideas of the papers discussed in the next two sections to deliver the attack and to identify the Tor client.

3.1 Browser Attacks

To browse the Internet anonymously using Tor, a user must use an HTTP proxy such as Privoxy so that traffic will be diverted through Tor rather than sent directly over the Internet. This is especially important because browsers will not automatically send DNS queries through a SOCKS proxy. However, pieces of software that plug into the browser, such as Flash, Java, and ActiveX Controls, do not necessarily use the browser's proxy for their network traffic. Thus, when any of these programs are downloaded and subsequently executed by the web browser, any Internet connections that the programs make will not go through Tor first. Instead, they will establish direct TCP connections, compromising the user's anonymity, as shown in Figure 2. This attack allows a website to identify its visitors but does not allow a third party to identify Tor users visiting a given website. These active content systems are well-known problems in anonymous web-browsing, and most anonymizing systems warn users to disable active content systems in their browsers.

In October 2006, FortConsult Security [2] described how to extend this attack so that parties could identify Tor users visiting a website they do not control. The attacker uses a malicious exit node to modify HTTP traffic and thus conduct a man-in-the-middle attack, as shown in Figure 3. In particular, it inserts an

[1] While the directory service is signed, anyone can add an entry, and claim to have a long uptime and high bandwidth. This makes getting users to use a malicious node a little easier, because clients prefer to use servers with good statistics.

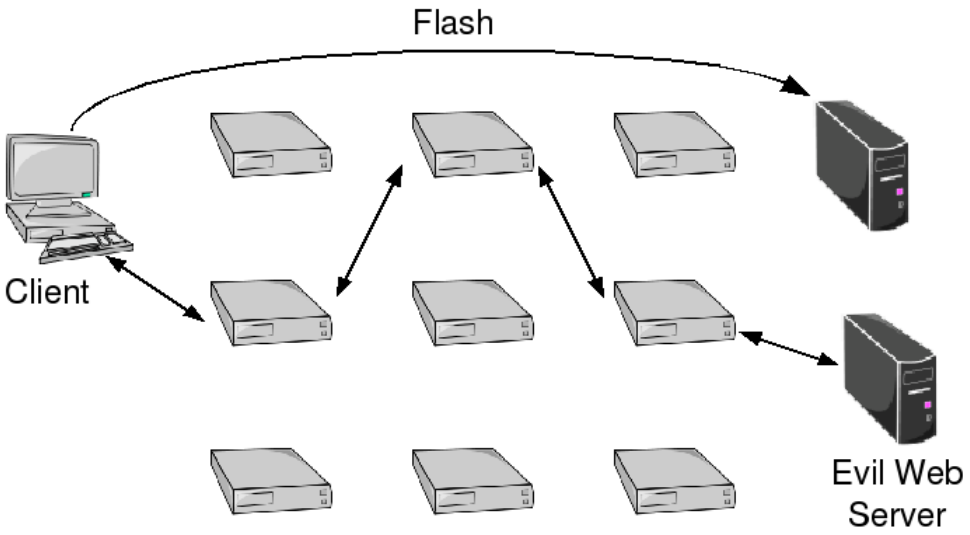

Fig. 2. Prior work: a browser attack using Flash included in a website. The client's web browser executes a Flash program, which then opens a direct connection to a logger machine, compromising the client's anonymity.

invisible iframe with a reference to some malicious web server and a unique cookie. In rendering the page, the web browser will make a request to the web server and will retrieve a malicious Flash application. If Flash is enabled in the browser, then the Flash movie is played invisibly. The Flash application sends the cookie given to the user directly to the evil web server, circumventing Tor. The web server can then identify which webpages were sent to which users by matching the cookies with the Flash connections. In other words, all Tor users who use HTTP through that exit node while Flash is enabled will have their HTTP traffic associated with their respective IP addresses. However, if we assume that the number of malicious Tor servers is small compared to the total number of Tor servers, a normal user will get a malicious exit node only once in a while. As a result, this attack only works to associate traffic with the particular user for the length of time that the user keeps the same Tor circuit, or at most ten minutes by default.

3.2 Finding Hidden Servers

Along with hiding the locations of clients, Tor also supports location-hidden servers, where the clients of a service (for example, visitors to a website) are not able to identify the machine hosting the service. To connect to a hidden server, a client sends a message through an introduction point that is advertised as

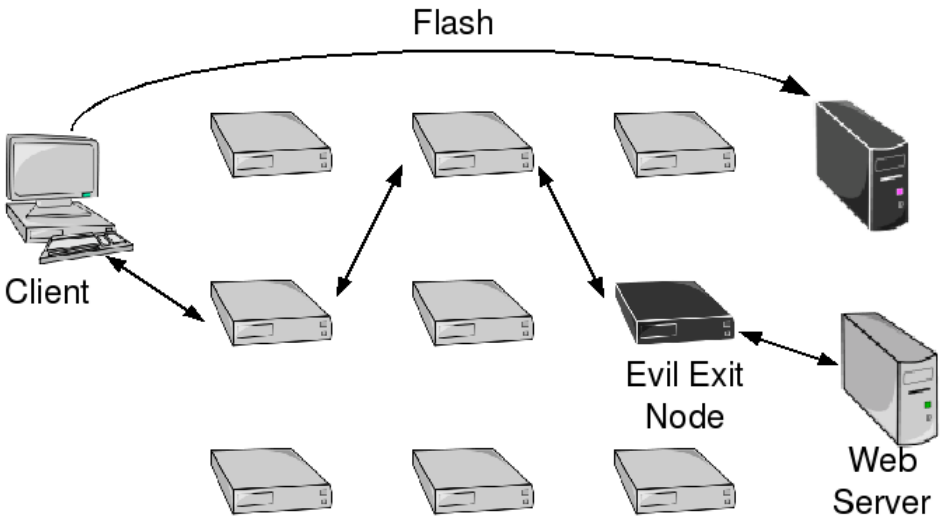

Fig. 3. Prior work: a browser attack executed by an exit node. The client's web browser executes a Flash program inserted into a webpage by the exit node, which opens a direct connection to a logger machine.

being associated with the hidden service by the Tor directory. A clever anonymous interaction results in the hidden client and hidden server both opening Tor connections to a rendezvous point (chosen by the client). The rendezvous point patches the connections together to form an anonymous channel between the hidden client and hidden server.

In May 2006, L. Øverlier and P. Syverson [12] described an attack to locate hidden servers in Tor. The attacker begins by inserting a malicious Tor node into the Tor network and using a Tor client to repeatedly connect to the targeted hidden server, sending a distinctive signal over each Tor connection. Since the hidden server cannot distinguish this from a wave of legitimate clients, each connection forces the hidden server to construct a new Tor circuit. The attacker can do traffic analysis to determine when his Tor node is in the hidden server's rendezvous circuit. He can then identify the hidden server by using a predecessor attack [18].

The paper states that their attack should apply to other clients using an anonymity network, but gives no details for how to do so. In particular, the attack does not immediately apply to clients because they don't make new circuits on demand. The attack relied on requesting a large number of new connections with a hidden server, which is not easy to do with a hidden client.

4 A Browser-Based Timing Attack

We describe a new attack that combines and builds upon the attacks discussed in Section 3. The attack, shown in Figure 4, attempts to discover a Tor client without using invasive plugins like Java or Flash but with JavaScript instead. JavaScript alone is not powerful enough to discover the client's IP address, but combined with a timing attack similar to the one presented by Øverlier and Syverson [12], an adversary has a non-trivial chance of discovering a client in a reasonable amount of time. In Section 4.2 we discuss how to implement this attack using only the HTML meta refresh tag, but the JavaScript version is simpler so we discuss it first. This attack is partially mitigated by *entry guards*, which has become a standard feature of Tor. For clarity, we will defer discussion of the role of entry guards until Section 4.7, after we have explained the basic plan of attack.

4.1 The Attack

Like the FortConsult Security attack [2], our attack uses a malicious Tor exit node that modifies HTTP traffic passing through it, inserting an invisible iframe containing JavaScript into requested webpages. The JavaScript repeatedly contacts a malicious web server, posting a unique ID. This JavaScript continues to run as long as the client leaves the "bugged" browser tab open. The complete attack is as follows:

1. The attacker first sets up the necessary resources.
   (a) The attacker inserts two malicious nodes into the Tor network: one to act as an entry node, and the other to act as an exit node.
   (b) The attacker sets up a web server that receives and logs JavaScript connections.
2. The malicious exit node modifies all HTTP traffic destined for Tor clients to include an invisible JavaScript signal generator that generates a unique signal for each Tor client.
3. The Tor client's web browser executes the JavaScript code, sending a distinctive signal to the web server. This traffic passes through the Tor circuit, and the client is still anonymous.
4. Approximately every ten minutes, the Tor client chooses a new circuit. Eventually, an unlucky Tor client picks and uses the malicious entry node.
5. The attacker performs traffic analysis to compare the signals on each circuit passing through his entry node with the various signals received by the web server. A match reveals the Tor client's identity and its corresponding traffic history during the time it used the malicious exit node.

The entry node only needs to log the traffic pattern that passes through it on each circuit, and the exit node only needs to perform the code injections in the HTTP traffic. Although for clarity we described the attack with multiple machines, the malicious Tor nodes and the web server can all be implemented

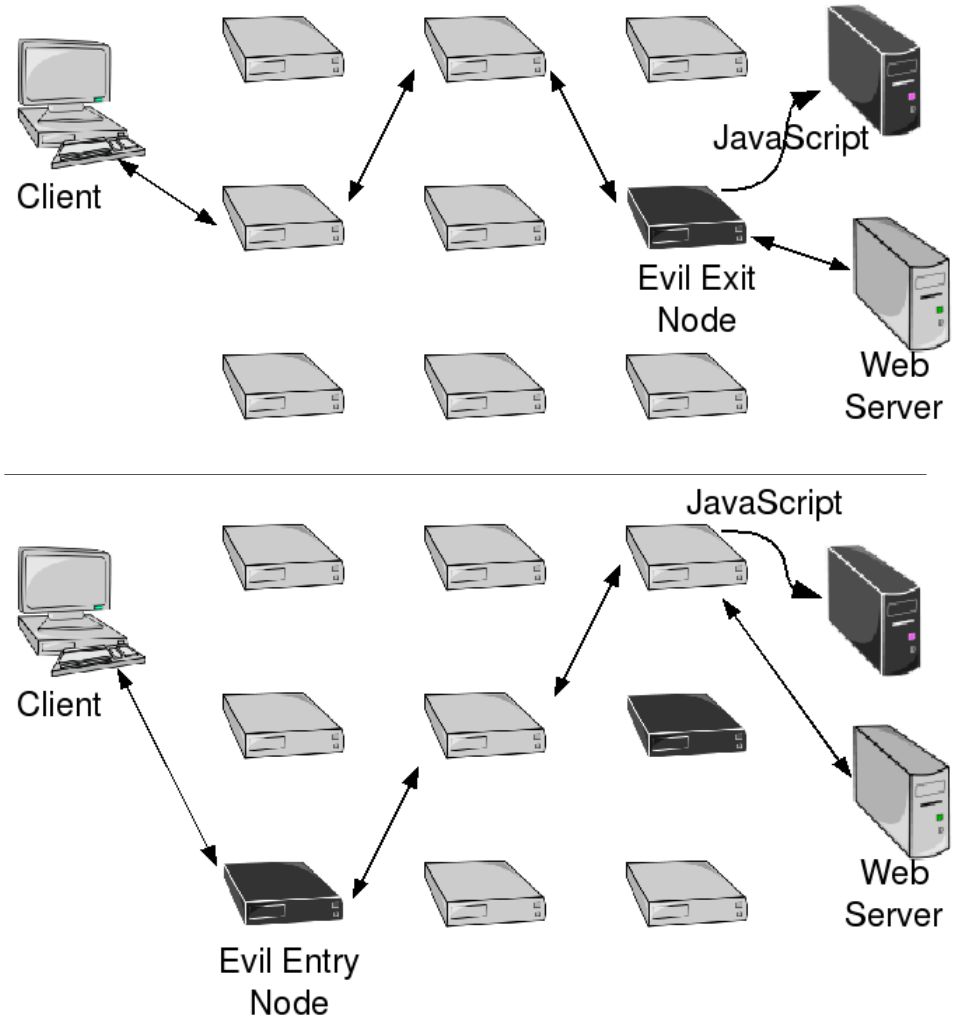

Fig. 4. Our new attack. A malicious exit node modifies webpages, inserting JavaScript code that repeatedly connects to a logger server, sending a distinctive signal along the link (top). If the client then uses a malicious entry node while that JavaScript is still executing, the entry node can detect the signal, and the attacker can thus associate the client with his communications (bottom).

on the same machine. If the user is browsing the web while using the malicious entry node, the traffic analysis can be difficult because the additional traffic introduces "noise" to the signal. However, if the user stops browsing the web, the channel contains little "noise" and the traffic analysis is easy. A method for simplifying the timing attack even if the user does continue browsing the web is discussed in Section 4.4.

For most traffic analysis attacks, the attacker must control both the exit node and entry node at the same time. For our attack, if a client leaves a browser window open running the JavaScript signal generator, and at any later point chooses a malicious entry node, then the timing attack can reveal his identity. Since this only requires the right choice of an entry node, the probability that the client is compromised each time he chooses a new circuit is roughly $\frac{1}{n_e}$, where n_e is the number of available entry nodes. If the attacker had to get control of both the entry and exit nodes at the same time, the probability would then be $\frac{1}{n_e n_x}$, where n_x is the number of available exit nodes. The signal generator allows us to decouple the need to control an exit node and an entry node at the same time, decreasing the expected time to compromise the client. As with any such traffic analysis attack, the adversary can further decrease the time the attack takes by increasing the number of malicious Tor entry nodes [16].

4.2 A Browser-Based Timing Attack Using Only HTML

The attack we just described relies on the victim having JavaScript enabled. This requirement is unnecessary. The same attack can be implemented by using the HTML meta refresh tag. In this version of the attack, the webpage is modified such that it will automatically be refreshed by the web browser after a period of time. The attacker generates the desired traffic signal by dynamically varying the refresh delays or the page size each time the webpage is refreshed.

The HTML meta refresh version of the attack is more conspicuous than the JavaScript version because browsers generally indicate when they are reloading a webpage but not when executing JavaScript XMLHttpRequests. Thus, it is easier for the user to observe the meta refresh version of the attack than the JavaScript one. This could be mitigated by only performing this attack on sites that already have a meta-refresh tag. Even on pages that would not normally have the tag, the HTML meta refresh attack could be made less obvious if the first refresh happens with a large delay, when the user is less likely to be still in front of his computer. After an initial delay of a few hours, subsequent refreshes could happen every few seconds to generate the signal for a timing attack.

4.3 Torbutton

Torbutton is a simple Firefox extension that allows a user to toggle whether their web browser is using the Tor proxy with a single click. This convenient interface makes it possible for users who are frustrated with Tor's slow speed to turn Tor off while browsing websites that they do not feel requires anonymity. It is a popular extension, with more than 251,000 downloads as of February

22, 2007 [19]. Since this number only counts downloads of Torbutton from the official website, it underestimates the number of Torbutton users.

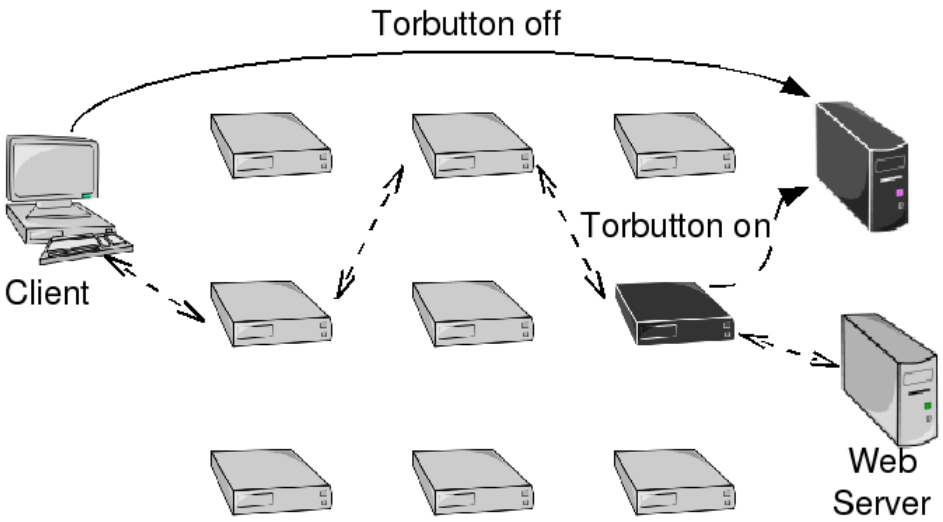

Fig. 5. Our Torbutton attack. A malicious exit node modifies webpages, inserting JavaScript code that repeatedly connects to a logger server, sending an ID number (seen above in dashed lines). If the client later configures their browser to stop using Tor while that JavaScript is still executing, he will connect directly to the logger server.

As shown in Figure 5, if a user toggles the Tor proxy off using Torbutton but leaves a tab open with one of our JavaScript signal generators, then he will be discovered the next time the signal generator contacts the adversary's server. In practice, this relatively simple attack is effective at but limited to discovering Tor clients who stop using the Tor proxy while the browser is still open. Torbutton makes it easy for users to be careless in this way.

Torbutton could easily be modified such that when the user chooses to stop using Tor, all JavaScript and automatically reloading webpages are stopped before changing the proxy settings. This may inconvenience the user if he is using sites that heavily depend on JavaScript, but it will protect the user from discovery. A Tor user who wants to browse the web both with and without Tor could also choose to either completely close his browser between uses or use two completely separate browsers for anonymous and nonanonymous communications.

4.4 Tor Exit Policies

Our attack works by adding traffic to a Tor circuit so that a Tor node can identify whether it is, in fact, the entry node of the Tor circuit. If the Tor circuit is carrying no other traffic, this detection is fairly easy—the entry node knows exactly what traffic pattern to look for. On the other hand, a Tor circuit full of unrelated traffic is hard to test for presence of a signal because the entry node does not know what other traffic to expect. For this reason, it is easier to identify a victim during a break than during active browsing. In this section we present a novel method of using exit policies to create a clean circuit dedicated to the identifying signal.

A common concern among Tor server operators is the issue of abuse: Tor can be used to anonymously send spam or viruses as well as to anonymously browse the web. In order to make it more attractive to run a Tor server, Tor's protocol dictates that each server advertises an *exit policy* that specifies which (IP address, port) pairs the server is willing to exit traffic to. Because few server operators are willing to exit spam or viruses, there are certain ports that almost every Tor server refuses to exit, as shown in Figure 6.

Port	Number of Exit Nodes
22	211
53	216
80	226
110	210
143	208
443	238
5190	184
6667	172

Port	Number of Exit Nodes
25	4
119	25
135–139	6
445	6
465	12
587	13
1214	7
4661–4666	5
6699	9

Fig. 6. Number of Tor servers exiting various ports as of December 15, 2006

4.5 Using Tor Exit Policies to Simplify the Timing Attack

Suppose that the signal generator connects to a malicious server over an unpopular port. Most web browsers will refuse to connect to some of the unpopular ports. For example, Mozilla Firefox resists connecting to the SMTP port 25 but not the filesharing ports 4661 through 4666. If the signal generator connects over an unpopular port, the client's existing circuits probably do not have an exit node willing to serve the signal generator's traffic. This likely forces the Tor client to open a new circuit for the signal generator's traffic. In fact, the Tor algorithm for routing traffic over circuits prefers older circuits, so for several minutes, other traffic may use a different circuit than the signal generator's traffic, even if the

new exit node is willing to exit other traffic. An attacker can improve the odds that the attack traffic will have a dedicated circuit by inserting exit nodes into the network that will only exit unpopular ports. As we have remarked before, having a dedicated circuit simplifies the traffic analysis substantially.

The Tor exit policy can also be used to decrease the time required for the attack. Suppose that there were zero nodes willing to exit k different ports. Then the attacker could insert k different servers into the Tor network, each of which only exits on one of the k ports. A signal generator that tried to connect on all k ports would force the Tor client to create k new circuits, each dedicated to the signal generator's traffic. Hence the client would have k different entry nodes at a time, rather than only one. This would speed up the attack by a factor of k.

In reality, there are no ports that have zero Tor nodes willing to exit on them, so one would expect a smaller speed increase. Some ports do come close; only five nodes offer ports 4661 through 4666, and at times only one or two of those are operating. If an attacker performed a denial of service attack on these nodes, he could create a situation where these ports do have zero Tor nodes willing to exit on them, and obtain the full factor of six.

Experiments showed that Tor's algorithm was slow to open a circuit to these unpopular ports. It often took several minutes to make a circuit, which can cause browser timeouts in connecting. While this wouldn't be a problem with the JavaScript version of the attack, this error would cause the HTML meta refresh version to fail sometimes.

We believe that this is the first reported method for exploiting the Tor exit policy system in an attack. A reasonable solution to this vulnerability would be to have a client-side exit policy. A client would only send data into Tor destined for selected ports—requests to send data on other ports would be refused. This exit policy should default to only allowing the client to send data via Tor that is destined for popular ports.

4.6 Using TCP Streams to Simplify the Timing Attack

Tor stops sending new connections through a circuit after the first 10 minutes of using the circuit. However, it does not close a circuit until there are no longer any open TCP sessions on the circuit. Thus, an attacker can hold a circuit open for more than 10 minutes by maintaining an open TCP stream on the circuit. After the first 10 minutes have passed, the attacker can start sending a unique signal to the client using that TCP connection. Unless there are other lingering TCP connections on the circuit, the attacker's traffic will now be the only traffic in that direction on the Tor circuit, allowing the attacker to detect the signal using simple traffic analysis techniques.

A defense against this attack is to forcibly close all TCP connections on a circuit after the 10 minutes are up. However, this is perhaps impractical for a couple of reasons. First, if a client is trying to download something that will take more than 10 minutes, the download will always fail. Second, various web applications utilize a perpetually open TCP connection in order to push changes to a

webpage (avoiding the delays associated with polling). Killing TCP connections when the circuit becomes stale could prove annoying to users.

An attacker that intentionally keeps a TCP connection open for an extended period of time is difficult to distinguish from the applications that are broken by the defense we just described. We know of no effective way to defend against this version of the attack without losing functionality.

4.7 Entry Guards

Our attack relies on the assumption that eventually, one of the malicious Tor routers will act as the entry node for a client. Since many attacks rely on this assumption, Wright et al. [17] and later Øverlier and Syverson [12] proposed selecting the entry nodes from a small subset of Tor nodes called *entry guards*. This feature is now standard in Tor [20]. By default, each Tor client chooses 3 random Tor nodes to be its entry guards. Thus if none of the entry guards are malicious, the client will never have a malicious entry node. If one or more of them is malicious, the client can be compromised much more quickly than if this feature were not used. Without entry guards, however, our attack would eventually expose all clients if there were even one malicious entry node in the entire Tor network.

Rather than selecting random entry guards, the user can choose a specific set of trusted nodes. This has benefits and drawbacks, which Øverlier and Syverson [12] discuss in detail. We suspect that most users will use the default random choices, so we will assume that henceforth.

One interesting feature of using entry guards is that a timing attack that would find the hidden client will instead find the entry guards. After the attacker identifies the entry guards that a targeted victim is using, he can attempt to execute a denial of service attack against those Tor servers in order to cause the victim to fall back to different entry nodes that the attacker might control.

Entry guards change the probability distribution so that the probability that a malicious server will ever be an entry node for a particular client is $O(\frac{3}{n})$, where n is the number of possible entry nodes in the network. On the other hand, those clients who are unlucky enough to select a malicious router as one of their entry guards will more quickly be discovered. If, for example, the attack targets the visitors of a particular website, the attack will affect up to an expected $\frac{3}{n}$ of the visitors. The use of entry guards then serves to speed up the attack on that fraction of that population. If we take today's numbers into account, n is around 700, one Tor node will be an entry guard for around 0.4% of a targeted population. Adding more malicious Tor nodes to the network is easy and would increase that proportion roughly linearly.

5 Methods

We developed a prototype implementation of this attack, expanding on the techniques discussed in the FortConsult Security paper [2]. FortConsult Security's

attack used Linux's iptables filtering to modify the payloads of TCP packets at the exit node. This was a convenient mechanism because it did not require modifications to the Tor source code itself. However, it resulted in a restriction on their attack: they could not change the number of bytes in any TCP packet, because TCP sequence numbers are a byte count. Thus their insertions into the webpages also required overwriting some part of those webpages. We sidestepped this issue by changing webpages at the HTTP level.

Our implementation also uses iptables, but only to redirect Tor's outgoing HTTP requests to a local port. Transproxy, a transparent proxy daemon, binds to this port and adds the appropriate headers to the requests so that a regular HTTP proxy works properly. The requests proceed through a proxy built on the Twisted Python libraries that modify every webpage to insert an iframe. This iframe downloads, from the adversary's web server, a page that contains a simple (25 line) JavaScript program that connects with this server. The size and frequency of these connections can be dynamically modified by the adversary's server in order to produce a distinctive signal.

Because the connections are to a URL containing the unique identifier, the server can ensure that each JavaScript instance sends a unique signal associated with its identifier. This allows the entry node to identify the client that downloaded a specific webpage, rather than one of the clients that downloaded any of the webpages with inserted signal generators. Not only can the attacker attribute the one webpage to this client, but he also knows that client was responsible for all the other traffic sent over its circuit ID at the time that it downloaded the bugged webpage.

When setting up a number of exit nodes to exit ports 4661-4666, we were able to put them all on one machine simply by configuring each Tor instance to use and advertise a different port and IP address. Upcoming Tor releases will not use two routers from the same Class B subnet on a single path, as a weak defense against the Sybil attack [6]. For our attack, the adversary can simply run two computers on different Class B networks, one running a large number of entry nodes, and the other running many exit nodes. Since Tor clients can choose any pair of routers from different subnets, the changes of selecting both entry node and exit node from the adversary's set will still be quadratic in the number of fake routers being used. Since colocation is easily available commercially, we do not believe that this new feature will present an effective defense against a determined attacker with limited resources.

Our experimental entry node required minor modifications of the Tor source to increase logging, most of which were used in Øverlier and Syverson [12]. We tried a Fourier transform to identify circuits with the signal, but could not find a strong enough signal (past the noise of legitimate web traffic) to identify them if they were browsing the web. We then implemented a much simpler recognition system that could find users when their circuit was only carrying attack traffic.

6 Defenses Against Browser-Based Attacks

We have considered a few defenses against these browser-based attacks.

6.1 Disabling Active Content Systems

The most obvious defense against these browser-based attacks is to disable all active content systems, such as Java, Flash, ActiveX Controls, and JavaScript in the browser. The disadvantage of this defense, however, is that disabling the active content systems would preclude the use of many popular web services in the process. Our HTML-only attack using the meta refresh tag also shows that this only exacerbates the problem since it can't be turned off without significant modification to a web browser.

6.2 HTTPS

Modifying a website at the exit node is a man-in-the-middle attack on HTTP. Because HTTPS is secure against man-in-the-middle attacks (assuming that the user has a chain of trust to the website), tunneling HTTP over SSL prevents a malicious exit node from either reading or modifying the data it is transporting.

In practice, this defense is less effective than it might seem, because users will often accept self-signed certificates as valid despite the browser warning. A malicious exit node could thus trick careless users by replacing webpages with malicious versions that are also signed, but with forged certificates.

Using HTTPS provides reasonable security against this attack so long as the client can trust the servers serving the sites he visits and correctly verifies certificates. If the server is not trustworthy, it can include the malicious JavaScript attack code in the website itself, and sign it with a valid SSL certificate. Using the methods we have described, the server could then identify its visitors.

Unfortunately, this defense is not something a Tor client can implement unilaterally; every website that he visits must allow the client to do so. Many sites do not allow a user to communicate with them in a secure fashion; for example, `https://www.google.com` currently (May 2007) redirects to `http://www.google.com`. Using SSL for all web traffic also has performance concerns, which is perhaps the reason why many sites do not support it.

7 Analysis and Results

Let us estimate the probability that our attack will be successful. Suppose that Tor uses k entry guards in a network of n nodes, m of which exit port 80. In our basic attack, the client uses one circuit at a time that changes every ten minutes. Further suppose that the attacker inserts u evil nodes in the network, of which v are exit nodes that modify HTTP traffic. The v exit nodes can be noticed by Tor users, but the other $u - v$ servers only log data, and give no indication of malice. Assume that all Tor nodes are equal—an unrealistic assumption, since

some Tor nodes have much better bandwidth than others, but not that relevant if the attacker has average bandwidth. At the moment, the Tor network has $k = 3, n = 700, m = 200$. Setting up an attack with $u = 1$ and $v = 1$ is fairly easy to accomplish, so we will use these values to approximate. We will also assume $n \gg k, u, v$.

Then $\frac{v}{m} \approx 0.5\%$ of all Tor circuits will insert signal generators into webpages, and approximately $\frac{ku}{n} \approx 0.4\%$ of all Tor clients will choose an evil server for an entry guard. Any given bugged page will use one entry guard every ten minutes, so for any Tor user that has an evil entry guard the chance of being discovered in any ten-minute interval is $\frac{1}{k} \approx 33\%$, and the probability of remaining anonymous over time is approximately the exponential distribution $P(t) \approx \left(\frac{k-1}{k}\right)^{t/10\text{ min}} \approx 0.66^{t/10\text{ min}}$.

This means that a Tor user has a 0.4% chance of ever being vulnerable to the attack. Every 10-minute interval during which a vulnerable user leaves a webpage open, he has a 0.5% chance of leaving a signal generator running. If he leaves a bugged page open over an hour-long lunch break, he has a 92% chance of having this signal generator go through an evil entry node. At this point, the adversary can associate the user with all the browsing that he did the circuit that he used to download the signal generator. If he leaves a bugged page open for eight hours of sleep, there is a negligible chance he will not be identified.

The probabilities that users are vulnerable or that they will receive a signal generator are low, but this is under the assumption that the attacker only controls a single Tor node. These probabilities are roughly linear in the number of Tor nodes the attacker runs, so he can amplify his probability of success by running several Tor nodes.

The attacker can decide whether to insert a signal generator into websites based on what other websites the potential victim has visited through the same Tor circuit. This allows a malicious exit node to masquerade as an honest machine to most users, a measure which would help the adversary prevent his exit node from becoming discovered as malicious.

8 Conclusion

Current web design presents fundamental problems for maintaining anonymity while browsing the web. Low latency anonymizing systems cannot easily protect their users from end-to-end traffic analysis. Our attack exploits the web browser code execution environment to perform end-to-end traffic analysis attacks without requiring the attacker to control either party to the target communication.

There are two security problems that our attack exploits: HTTP's vulnerability to man-in-the-middle attacks and web browsers' code execution feature. Tor places the exit node as a man-in-the-middle of clients' communications. Thus, using Tor may actually decrease the anonymity of users by making them vulnerable to man-in-the-middle attacks from adversaries that would otherwise be unable to perform such attacks.

Also fundamental to our attack is the fact that web browsers execute (potentially malicious) code within an imperfect sandbox. This code execution allows for arbitrary communication back to the HTTP server. Such communication can include sending network traffic in a pattern designed to be detected by an external observer using traffic analysis. This danger is particularly important when we consider that recent advances in the web are centered around the use of complex programs executed by the web browser. Even if users are willing to disable these technologies, we have shown that mere HTML (through its meta refresh tag) is a powerful enough language to attack the anonymity of Tor users.

Given the current design of the web, neither of these problems can be readily addressed without sacrificing substantial functionality.

9 Acknowledgements

We thank Lasse Øverlier for sharing his hidden servers timing attack code with us. We also thank Roger Dingledine, Paul Syverson, Frans Kaashoek, and the anonymous reviewers for their helpful suggestions.

References

1. D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM 24(2)*, February 1981.
2. A. Christensen et al. Practical Onion Hacking: Find the real address of Tor clients. *FortConsult*, October 2006. <`http://www.fortconsult.net/images/pdf/Practical_Onion_Hacking.pdf`>
3. D. Clark. Design Philosophy of the DARPA Internet Protocols. In *Proceedings of the ACM Special Interest Group on Data Communications*, pages 106–114, August 1988.
4. R. Dingledine. Tor: anonymity online, November 2006. <`http://tor.eff.org/`>
5. R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.
6. J. Douceur. The Sybil Attack. In *Proceedings of the 1st International Peer To Peer Systems Workshop (IPTPS)*, March 2002.
7. A. Hintz. Fingerprinting Websites Using Traffic Analysis. In *Proceedings of Privacy Enhancing Technologies workshop*, April 2002.
8. B. N. Levine, M. Reiter, C. Wang, and M. Wright. Timing Attacks in Low-Latency Mix Systems (Extended Abstract), In *Proc. Financial Cryptography*, pages 251–265, February 2004.
9. M. Liberatore and B. N. Levine. Inferring the source of encrypted HTTP connections. In *Proceedings of the 13th ACM conference on Computer and communications security*, 2006.
10. K. Martin. AOL search data identified individuals. *SecurityFocus*, August 2006. <`http://www.securityfocus.com/brief/277`>
11. S. J. Murdoch and G. Danezis. Low-Cost Traffic Analysis of Tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, May 2005.

12. L. Øverlier and P. Syverson. Locating Hidden Servers. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, May 2006.
13. J. Raymond. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 10–29, July 2000.
14. A. Serjantov and P. Sewell. Passive Attack Analysis for Connection-Based Anonymity Systems. In *European Symposium on Research in Computer Security (ESORICS)*, pages 116–131, October 2003.
15. P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. Towards an Analysis of Onion Routing Security. *Workshop on Design Issues in Anonymity and Unobservability*, July 2000.
16. M. Wright, M. Adler, B. N. Levine, and C. Shields. An Analysis of the Degradation of Anonymous Protocols. In *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*, pages 38–50, February 2002.
17. M. Wright, M. Adler, B. N. Levine, and C. Shields. Defending Anonymous Communication Against Passive Logging Attacks. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003.
18. M. Wright, M. Adler, B. N. Levine, and C. Shields. The predecessor attack: An analysis of a threat to anonymous communications systems. In *ACM Trans. Inf. Syst. Secur.*, pages 489–522, 2004.
19. S. Squires. Firefox Add-ons: Torbutton. February 2007. <`https://addons.mozilla.org/firefox/2275/`>
20. TheOnionRouter/TorFAQ. November 2006. <`http://wiki.noreply.org/noreply/TheOnionRouter/TorFAQ`>