The Design and Implementation of the Tor Browser [DRAFT]

Mike Perry

<mikeperry#torproject org>

Erinn Clark

<erinn#torproject org>

Steven Murdoch

<sjmurdoch#torproject org>

Georg Koppen

<gk#torproject org>

June 15, 2018

Table of Contents

1. Introduction

This document describes the adversary model, design requirements, and implementation of the Tor

Browser. It is current as of Tor Browser 7.0.11.

This document is also meant to serve as a set of design requirements and to describe a reference implementation of a Private Browsing Mode that defends against active network adversaries, in addition to the passive forensic local adversary currently addressed by the major browsers.

For more practical information regarding Tor Browser development, please consult the Tor Browser Hacking Guide.

1.1. Browser Component Overview

The Tor Browser is based on Mozilla's Extended Support Release (ESR) Firefox branch. We have a series of patches against this browser to enhance privacy and security. Browser behavior is additionally augmented through the Torbutton extension, though we are in the process of moving this functionality into direct Firefox patches. We also change a number of Firefox preferences from their defaults.

Tor process management and configuration is accomplished through the Tor Launcher addon, which provides the initial Tor configuration splash screen and bootstrap progress bar. Tor Launcher is also compatible with Thunderbird, Instantbird, and XULRunner.

To help protect against potential Tor Exit Node eavesdroppers, we include HTTPS-Everywhere. To provide users with optional defense-in-depth against JavaScript and other potential exploit vectors, we also include NoScript. We also modify several extension preferences from their defaults.

To provide censorship circumvention in areas where the public Tor network is blocked either by IP, or by protocol fingerprint, we include several Pluggable Transports in the distribution. As of this writing, we include Obfs3proxy, Obfs4proxy, meek, and FTE.

2. Design Requirements and Philosophy

The Tor Browser Design Requirements are meant to describe the properties of a Private Browsing Mode that defends against both network and local forensic adversaries.

There are two main categories of requirements: Security Requirements, and Privacy Requirements. Security Requirements are the minimum properties in order for a browser to be able to support Tor and similar privacy proxies safely. Privacy requirements are the set of properties that cause us to prefer one browser over another.

While we will endorse the use of browsers that meet the security requirements, it is primarily the privacy requirements that cause us to maintain our own browser distribution.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

2.1. Security Requirements

The security requirements are primarily concerned with ensuring the safe use of Tor. Violations in these properties typically result in serious risk for the user in terms of immediate deanonymization and/or observability. With respect to browser support, security requirements are the minimum properties in order for Tor to support the use of a particular browser.

1. **Proxy Obedience**

   The browser MUST NOT bypass Tor proxy settings for any content.

2. **State Separation**

   The browser MUST NOT provide the content window with any state from any other browsers or any non-Tor browsing modes. This includes shared state from independent plugins, and shared state from operating system implementations of TLS and other support libraries.

3. **Disk Avoidance**

   The browser MUST NOT write any information that is derived from or that reveals browsing activity to the disk, or store it in memory beyond the duration of one browsing session, unless the user has explicitly opted to store their browsing history information to disk.

4. **Application Data Isolation**

   The components involved in providing private browsing MUST be self-contained, or MUST provide a mechanism for rapid, complete removal of all evidence of the use of the mode. In other words, the browser MUST NOT write or cause the operating system to write *any information* about the use of private browsing to disk outside of the application's control. The user must be able to ensure that secure deletion of the software is sufficient to remove evidence of the use of the software. All exceptions and shortcomings due to operating system behavior MUST be wiped by an uninstaller. However, due to permissions issues with access to swap, implementations MAY choose to leave it out of scope, and/or leave it to the operating system/platform to implement ephemeral-keyed encrypted swap.

2.2. Privacy Requirements

The privacy requirements are primarily concerned with reducing linkability: the ability for a user's activity on one site to be linked with their activity on another site without their knowledge or explicit consent. With respect to browser support, privacy requirements are the set of properties that cause us to prefer one browser over another.

For the purposes of the unlinkability requirements of this section as well as the descriptions in the implementation section, a **URL bar origin** means at least the second-level DNS name. For example, for mail.google.com, the origin would be google.com. Implementations MAY, at their option, restrict the URL bar origin to be the entire fully qualified domain name.

1. **Cross-Origin Identifier Unlinkability**

   User activity on one URL bar origin MUST NOT be linkable to their activity in any other URL bar origin by any third party automatically or without user interaction or approval. This requirement specifically applies to linkability from stored browser identifiers, authentication tokens, and shared state. The requirement does not apply to linkable information the user manually submits to sites, or due to information submitted during manual link traversal. This functionality SHOULD NOT interfere with interactive, click-driven federated login in a substantial way.

2. **Cross-Origin Fingerprinting Unlinkability**

   User activity on one URL bar origin MUST NOT be linkable to their activity in any other URL bar origin by any third party. This property specifically applies to linkability from fingerprinting browser behavior.

3. **Long-Term Unlinkability**

   The browser MUST provide an obvious, easy way for the user to remove all of its authentication tokens and browser state and obtain a fresh identity. Additionally, the browser SHOULD clear linkable state by default automatically upon browser restart, except at user option.

2.3. Philosophy

In addition to the above design requirements, the technology decisions about Tor Browser are also guided by some philosophical positions about technology.

1. **Preserve existing user model**

   The existing way that the user expects to use a browser must be preserved. If the user has to maintain a different mental model of how the sites they are using behave depending on tab, browser state, or anything else that would not normally be what they experience in their default browser, the user will inevitably be confused. They will make mistakes and reduce their privacy as a result. Worse, they may just stop using the browser, assuming it is broken.

   User model breakage was one of the failures of Torbutton: Even if users managed to install everything properly, the toggle model was too hard for the average user to understand, especially in the face of accumulating tabs from multiple states crossed with the current Tor-state of the browser.

2. **Favor the implementation mechanism least likely to break sites**

   In general, we try to find solutions to privacy issues that will not induce site breakage, though this is not always possible.

3. **Plugins must be restricted**

   Even if plugins always properly used the browser proxy settings (which none of them do) and could not be induced to bypass them (which all of them can), the activities of closed-source plugins are very difficult to audit and control. They can obtain and transmit all manner of system information to websites, often have their own identifier storage for tracking users, and also contribute to fingerprinting.

   Therefore, if plugins are to be enabled in private browsing modes, they must be restricted from running automatically on every page (via click-to-play placeholders), and/or be sandboxed to restrict the types of system calls they can execute. If the user agent allows the user to craft an exemption to allow a plugin to be used automatically, it must only apply to the top level URL bar domain, and not to all sites, to reduce cross-origin fingerprinting linkability.

4. **Minimize Global Privacy Options**

   Another failure of Torbutton was the options panel. Each option that detectably alters browser behavior can be used as a fingerprinting tool. Similarly, all extensions should be disabled in the mode except as an opt-in basis. We should not load system-wide and/or operating system provided addons or plugins.

   Instead of global browser privacy options, privacy decisions should be made per URL bar origin to eliminate the possibility of linkability between domains. For example, when a plugin object (or a JavaScript access of window.plugins) is present in a page, the user should be given the choice of allowing that plugin object for that URL bar origin only. The same goes for exemptions to third party cookie policy, geolocation, and any other privacy permissions.

   If the user has indicated they wish to record local history storage, these permissions can be written to disk. Otherwise, they should remain memory-only.

5. **No filters**

   Site-specific or filter-based addons such as AdBlock Plus, Request Policy, Ghostery, Priv3, and Sharemenot are to be avoided. We believe that these addons do not add any real privacy to a proper implementation of the above privacy requirements, and that development efforts should be focused

on general solutions that prevent tracking by all third parties, rather than a list of specific URLs or hosts.

Implementing filter-based blocking directly into the browser, such as done with Firefox' Tracking Protection, does not alleviate the concerns mentioned in the previous paragraph. There is still just a list containing specific URLs and hosts which, in this case, are assembled by Disconnect and adapted by Mozilla.

Trying to resort to filter methods based on machine learning does not solve the problem either: they don't provide a general solution to the tracking problem as they are working probabilistically. Even with a precision rate at 99% and a false positive rate at 0.1% trackers would be missed and sites would be wrongly blocked.

Filter-based solutions in general can also introduce strange breakage and cause usability nightmares. For instance, there is a trend to observe that websites start detecting filer extensions and block access to content on them. Coping with this fallout easily leads to just whitelisting the affected domains, hoping that this helps, defeating the purpose of the filter in the first place. Filters will also fail to do their job if an adversary simply registers a new domain or creates a new URL path. Worse still, the unique filter sets that each user creates or installs will provide a wealth of fingerprinting targets.

As a general matter, we are also generally opposed to shipping an always-on Ad blocker with Tor Browser. We feel that this would damage our credibility in terms of demonstrating that we are providing privacy through a sound design alone, as well as damage the acceptance of Tor users by sites that support themselves through advertising revenue.

Users are free to install these addons if they wish, but doing so is not recommended, as it will alter the browser request fingerprint.

6. **Stay Current**

   We believe that if we do not stay current with the support of new web technologies, we cannot hope to substantially influence or be involved in their proper deployment or privacy realization. However, we will likely disable high-risk features pending analysis, audit, and mitigation.

3. Adversary Model

A Tor web browser adversary has a number of goals, capabilities, and attack types that can be used to illustrate the design requirements for the Tor Browser. Let's start with the goals.

3.1. Adversary Goals

1. **Bypassing proxy settings**

   The adversary's primary goal is direct compromise and bypass of Tor, causing the user to directly connect to an IP of the adversary's choosing.

2. **Correlation of Tor vs Non-Tor Activity**

   If direct proxy bypass is not possible, the adversary will likely happily settle for the ability to correlate something a user did via Tor with their non-Tor activity. This can be done with cookies, cache identifiers, JavaScript events, and even CSS. Sometimes the fact that a user uses Tor may be enough for some authorities.

3. **History disclosure**

   The adversary may also be interested in history disclosure: the ability to query a user's history to see

if they have issued certain censored search queries, or visited censored sites.

4. **Correlate activity across multiple sites**

   The primary goal of the advertising networks is to know that the user who visited siteX.com is the same user that visited siteY.com to serve them targeted ads. The advertising networks become our adversary insofar as they attempt to perform this correlation without the user's explicit consent.

5. **Fingerprinting/anonymity set reduction**

   Fingerprinting (more generally: "anonymity set reduction") is used to attempt to gather identifying information on a particular individual without the use of tracking identifiers. If the dissident's or whistleblower's timezone is available, and they are using a rare build of Firefox for an obscure operating system, and they have a specific display resolution only used on one type of laptop, this can be very useful information for tracking them down, or at least tracking their activities.

6. **History records and other on-disk information**

   In some cases, the adversary may opt for a heavy-handed approach, such as seizing the computers of all Tor users in an area (especially after narrowing the field by the above two pieces of information). History records and cache data are the primary goals here. Secondary goals may include confirming on-disk identifiers (such as hostname and disk-logged spoofed MAC address history) obtained by other means.

3.2. Adversary Capabilities - Positioning

The adversary can position themselves at a number of different locations in order to execute their attacks.

1. **Exit Node or Upstream Router**

   The adversary can run exit nodes, or alternatively, they may control routers upstream of exit nodes. Both of these scenarios have been observed in the wild.

2. **Ad servers and/or Malicious Websites**

   The adversary can also run websites, or more likely, they can contract out ad space from a number of different ad servers and inject content that way. For some users, the adversary may be the ad servers themselves. It is not inconceivable that ad servers may try to subvert or reduce a user's anonymity through Tor for marketing purposes.

3. **Local Network/ISP/Upstream Router**

   The adversary can also inject malicious content at the user's upstream router when they have Tor disabled, in an attempt to correlate their Tor and Non-Tor activity.

   Additionally, at this position the adversary can block Tor, or attempt to recognize the traffic patterns of specific web pages at the entrance to the Tor network.

4. **Physical Access**

   Some users face adversaries with intermittent or constant physical access. Users in Internet cafes, for example, face such a threat. In addition, in countries where simply using tools like Tor is illegal, users may face confiscation of their computer equipment for excessive Tor usage or just general suspicion.

3.3. Adversary Capabilities - Attacks

The adversary can perform the following attacks from a number of different positions to accomplish

various aspects of their goals. It should be noted that many of these attacks (especially those involving IP address leakage) are often performed by accident by websites that simply have JavaScript, dynamic CSS elements, and plugins. Others are performed by ad servers seeking to correlate users' activity across different IP addresses, and still others are performed by malicious agents on the Tor network and at national firewalls.

1. **Read and insert identifiers**

   The browser contains multiple facilities for storing identifiers that the adversary creates for the purposes of tracking users. These identifiers are most obviously cookies, but also include HTTP auth, DOM storage, cached scripts and other elements with embedded identifiers, client certificates, and even TLS Session IDs.

   An adversary in a position to perform MITM content alteration can inject document content elements to both read and inject cookies for arbitrary domains. In fact, even many "SSL secured" websites are vulnerable to this sort of active sidejacking. In addition, the ad networks of course perform tracking with cookies as well.

   These types of attacks are attempts at subverting our Cross-Origin Identifier Unlinkability and Long-Term Unlinkability design requirements.

2. **Fingerprint users based on browser attributes**

   There is an absurd amount of information available to websites via attributes of the browser. This information can be used to reduce the anonymity set, or even uniquely fingerprint individual users. Attacks of this nature are typically aimed at tracking users across sites without their consent, in an attempt to subvert our Cross-Origin Fingerprinting Unlinkability and Long-Term Unlinkability design requirements.

   Fingerprinting is an intimidating problem to attempt to tackle, especially without a metric to determine or at least intuitively understand and estimate which features will most contribute to linkability between visits.

   The Panopticlick study done by the EFF uses the Shannon entropy - the number of identifying bits of information encoded in browser properties - as this metric. Their result data is definitely useful, and the metric is probably the appropriate one for determining how identifying a particular browser property is. However, some quirks of their study means that they do not extract as much information as they could from display information: they only use desktop resolution and do not attempt to infer the size of toolbars. In the other direction, they may be over-counting in some areas, as they did not compute joint entropy over multiple attributes that may exhibit a high degree of correlation. Also, new browser features are added regularly, so the data should not be taken as final.

   Despite the uncertainty, all fingerprinting attacks leverage the following attack vectors:

   a. **Observing Request Behavior**

      Properties of the user's request behavior comprise the bulk of low-hanging fingerprinting targets. These include: User agent, Accept-* headers, pipeline usage, and request ordering. Additionally, the use of custom filters such as AdBlock and other privacy filters can be used to fingerprint request patterns (as an extreme example).

   b. **Inserting JavaScript**

      JavaScript can reveal a lot of fingerprinting information. It provides DOM objects such as window.screen and window.navigator to extract information about the user agent. Also, JavaScript can be used to query the user's timezone via the `Date()` object, WebGL can reveal information about the video card in use, and high precision timing information can be used to

fingerprint the CPU and interpreter speed. JavaScript features such as Resource Timing may leak an unknown amount of network timing related information. And, moreover, JavaScript is able to extract available fonts on a device with high precision.

c. **Inserting Plugins**

The Panopticlick project found that the mere list of installed plugins (in navigator.plugins) was sufficient to provide a large degree of fingerprintability. Additionally, plugins are capable of extracting font lists, interface addresses, and other machine information that is beyond what the browser would normally provide to content. In addition, plugins can be used to store unique identifiers that are more difficult to clear than standard cookies. Flash-based cookies fall into this category, but there are likely numerous other examples. Beyond fingerprinting, plugins are also abysmal at obeying the proxy settings of the browser.

d. **Inserting CSS**

CSS media queries can be inserted to gather information about the desktop size, widget size, display type, DPI, user agent type, and other information that was formerly available only to JavaScript.

3. **Website traffic fingerprinting**

Website traffic fingerprinting is an attempt by the adversary to recognize the encrypted traffic patterns of specific websites. In the case of Tor, this attack would take place between the user and the Guard node, or at the Guard node itself.

The most comprehensive study of the statistical properties of this attack against Tor was done by Panchenko et al. Unfortunately, the publication bias in academia has encouraged the production of a number of follow-on attack papers claiming "improved" success rates, in some cases even claiming to completely invalidate any attempt at defense. These "improvements" are actually enabled primarily by taking a number of shortcuts (such as classifying only very small numbers of web pages, neglecting to publish ROC curves or at least false positive rates, and/or omitting the effects of dataset size on their results). Despite these subsequent "improvements", we are skeptical of the efficacy of this attack in a real world scenario, *especially* in the face of any defenses.

In general, with machine learning, as you increase the number and/or complexity of categories to classify while maintaining a limit on reliable feature information you can extract, you eventually run out of descriptive feature information, and either true positive accuracy goes down or the false positive rate goes up. This error is called the bias in your hypothesis space. In fact, even for unbiased hypothesis spaces, the number of training examples required to achieve a reasonable error bound is a function of the complexity of the categories you need to classify.

In the case of this attack, the key factors that increase the classification complexity (and thus hinder a real world adversary who attempts this attack) are large numbers of dynamically generated pages, partially cached content, and also the non-web activity of the entire Tor network. This yields an effective number of "web pages" many orders of magnitude larger than even Panchenko's "Open World" scenario, which suffered continuous near-constant decline in the true positive rate as the "Open World" size grew (see figure 4). This large level of classification complexity is further confounded by a noisy and low resolution featureset - one which is also relatively easy for the defender to manipulate at low cost.

To make matters worse for a real-world adversary, the ocean of Tor Internet activity (at least, when compared to a lab setting) makes it a certainty that an adversary attempting examine large amounts of Tor traffic will ultimately be overwhelmed by false positives (even after making heavy tradeoffs on the ROC curve to minimize false positives to below 0.01%). This problem is known in the IDS literature as the Base Rate Fallacy, and it is the primary reason that anomaly and activity

classification-based IDS and antivirus systems have failed to materialize in the marketplace (despite early success in academic literature).

Still, we do not believe that these issues are enough to dismiss the attack outright. But we do believe these factors make it both worthwhile and effective to deploy light-weight defenses that reduce the accuracy of this attack by further contributing noise to hinder successful feature extraction.

4. **Remotely or locally exploit browser and/or OS**

   Last, but definitely not least, the adversary can exploit either general browser vulnerabilities, plugin vulnerabilities, or OS vulnerabilities to install malware and surveillance software. An adversary with physical access can perform similar actions.

   For the purposes of the browser itself, we limit the scope of this adversary to one that has passive forensic access to the disk after browsing activity has taken place. This adversary motivates our Disk Avoidance defenses.

   An adversary with arbitrary code execution typically has more power, though. It can be quite hard to really significantly limit the capabilities of such an adversary. The Tails system can provide some defense against this adversary through the use of readonly media and frequent reboots, but even this can be circumvented on machines without Secure Boot through the use of BIOS rootkits.

4. Implementation

The Implementation section is divided into subsections, each of which corresponds to a Design Requirement. Each subsection is divided into specific web technologies or properties. The implementation is then described for that property.

In some cases, the implementation meets the design requirements in a non-ideal way (for example, by disabling features). In rare cases, there may be no implementation at all. Both of these cases are denoted by differentiating between the **Design Goal** and the **Implementation Status** for each property. Corresponding bugs in the Tor bug tracker are typically linked for these cases.

4.1. Proxy Obedience

Proxy obedience is assured through the following:

1. **Firefox proxy settings, patches, and build flags**

   Our Firefox preferences file sets the Firefox proxy settings to use Tor directly as a SOCKS proxy. It sets **network.proxy.socks_remote_dns**, **network.proxy.socks_version**, **network.proxy.socks_port**, and **network.dns.disablePrefetch**.

   To prevent proxy bypass by WebRTC calls, we disable WebRTC at compile time with the **--disable-webrtc** configure switch, as well as set the pref **media.peerconnection.enabled** to false.

   We also patch Firefox in order to provide several defense-in-depth mechanisms for proxy safety. Notably, we patch the DNS service to prevent any browser or addon DNS resolution, and we also remove the DNS lookup for the profile lock signature. Furhermore, we patch OCSP and PKIX code to prevent any use of the non-proxied command-line tool utility functions from being functional while linked in to the browser. In both cases, we could find no direct paths to these routines in the browser, but it seemed better safe than sorry.

   For further defense-in-depth we disable WebIDE because it can bypass proxy settings for remote debugging, and also because it downloads extensions we have not reviewed. We are doing this by setting **devtools.webide.autoinstallADBHelper**, **devtools.webide.autoinstallFxdtAdapters**,

devtools.webide.enabled, and **devtools.appmanager.enabled** to **false**. Moreover, we removed the Roku Screen Sharing and screencaster code with a Firefox patch as these features can bypass proxy settings as well.

Further down on our road to proxy safety we disable the network tickler as it has the capability to send UDP traffic and we disable mDNS support, since mDNS uses UDP packets as well. We also disable Mozilla's TCPSocket by setting **dom.mozTCPSocket.enabled** to **false**. We intend to rip out the TCPSocket code in the future to have an even more solid guarantee that it won't be used by accident.

Finally, we remove potentially unsafe Rust code.

During every Extended Support Release transition, we perform in-depth code audits to verify that there were no system calls or XPCOM activity in the source tree that did not use the browser proxy settings.

We have verified that these settings and patches properly proxy HTTPS, OCSP, HTTP, FTP, gopher (now defunct), DNS, SafeBrowsing Queries, all JavaScript activity, including HTML5 audio and video objects, addon updates, WiFi geolocation queries, searchbox queries, XPCOM addon HTTPS/HTTP activity, WebSockets, and live bookmark updates. We have also verified that external protocol helpers, such as SMB URLs and other custom protocol handlers are all blocked.

2. **Disabling plugins**

   Plugins, like Flash, have the ability to make arbitrary OS system calls and bypass proxy settings. This includes the ability to make UDP sockets and send arbitrary data independent of the browser proxy settings.

   Torbutton disables plugins by using the **@mozilla.org/plugin/host;1** service to mark the plugin tags as disabled. This block can be undone through both the Torbutton Security UI, and the Firefox Plugin Preferences.

   If the user does enable plugins in this way, plugin-handled objects are still restricted from automatic load through Firefox's click-to-play preference **plugins.click_to_play**.

   In addition, to reduce any unproxied activity by arbitrary plugins at load time, and to reduce the fingerprintability of the installed plugin list, we also patch the Firefox source code to prevent the load of any plugins except for Flash and Gnash. Even for Flash and Gnash, we also patch Firefox to prevent loading them into the address space until they are explicitly enabled.

   With Gecko Media Plugins (GMPs) a second type of plugins is available. They are mainly third party codecs and EME content decryption modules. We currently disable these plugins as they either can't be built reproducibly or are binary blobs which we are not allowed to audit (or both). For the EME case we use the **--disable-eme** configure switch and set **browser.eme.ui.enabled**, **media.gmp-eme-adobe.visible**, **media.gmp-eme-adobe.enabled**, **media.gmp-widevinecdm.visible**, **media.gmp-widevinecdm.enabled**, **media.eme.enabled**, and **media.eme.apiVisible** to **false** to indicate to the user that this feature is disabled. For GMPs in general we make sure that the external server is not even pinged for updates/downloads in the first place by setting **media.gmp-manager.url.override** to **data:text/plain,** and avoid any UI with **media.gmp-provider.enabled** set to **false**. Moreover, we disable GMP downloads via local fallback by setting **media.gmp-manager.updateEnabled** to **false**. To reduce our attack surface we exclude the ClearKey EME system, too.

3. **External App Blocking and Drag Event Filtering**

   External apps can be induced to load files that perform network activity. Unfortunately, there are cases where such apps can be launched automatically with little to no user input. In order to prevent this, we ship Firefox patches and Torbutton installs a component to provide the user with a popup

whenever the browser attempts to launch a helper application.

Furthermore, we ship a patch for Linux users that makes sure sftp:// and smb:// URLs are not passed along to the operating system as this can lead to proxy bypasses on systems that have GIO/GnomeVFS support. And proxy bypass risks due to file:// URLs should be mitigated for macOS and Linux users by two further patches.

Additionally, modern desktops now preemptively fetch any URLs in Drag and Drop events as soon as the drag is initiated. This download happens independent of the browser's Tor settings, and can be triggered by something as simple as holding the mouse button down for slightly too long while clicking on an image link. We filter drag and drop events events from Torbutton before the OS downloads the URLs the events contained.

4. **Disabling system extensions and clearing the addon whitelist**

   Firefox addons can perform arbitrary activity on your computer, including bypassing Tor. It is for this reason we disable the addon whitelist (**xpinstall.whitelist.add**), so that users are prompted before installing addons regardless of the source. We also exclude system-level addons from the browser through the use of **extensions.enabledScopes** and **extensions.autoDisableScopes**. Furthermore, we set **extensions.systemAddon.update.url** and **extensions.hotfix.id** to an empty string in order to avoid the risk of getting extensions installed by Mozilla into Tor Browser, and remove unused system extensions with a Firefox patch. In order to make it harder for users to accidentally install extensions which Mozilla presents to them on the *about:addons* page, we hide the *Get Addons* option on it by setting **extensions.getAddons.showPane** to **false**.

4.2. State Separation

Tor Browser State is separated from existing browser state through use of a custom Firefox profile, and by setting the $HOME environment variable to the root of the bundle's directory. The browser also does not load any system-wide extensions (through the use of **extensions.enabledScopes** and **extensions.autoDisableScopes**). Furthermore, plugins are disabled, which prevents Flash cookies from leaking from a pre-existing Flash directory.

4.3. Disk Avoidance

Design Goal:

> The User Agent MUST (at user option) prevent all disk records of browser activity. The user SHOULD be able to optionally enable URL history and other history features if they so desire.

Implementation Status:

> We are working towards this goal through several mechanisms. First, we set the Firefox Private Browsing preference **browser.privatebrowsing.autostart** to **true**. We also had to disable the media cache with the pref **media.cache_size**, to prevent HTML5 videos from being written to the OS temporary directory, which happened regardless of the private browsing mode setting. Finally, we set **security.nocertdb** to **true** to make the intermediate certificate store memory-only.
>
> Moreover, we prevent text leaking from the web console to the /tmp directory with a direct Firefox patch.
>
> As an additional defense-in-depth measure, we set **browser.cache.disk.enable**, **browser.cache.offline.enable**, **signon.rememberSignons**, **browser.formfill.enable** to **true**, **browser.download.manager.retention** to **1**, and both **browser.sessionstore.privacy_level**

and **network.cookie.lifetimePolicy** to **2**. Many of these preferences are likely redundant with **browser.privatebrowsing.autostart** enabled, but we have not done the auditing work to ensure that yet.

For more details on disk leak bugs and enhancements, see the tbb-disk-leak tag in our bugtracker

4.4. Application Data Isolation

Tor Browser MUST NOT cause any information to be written outside of the bundle directory. This is to ensure that the user is able to completely and safely remove it without leaving other traces of Tor usage on their computer.

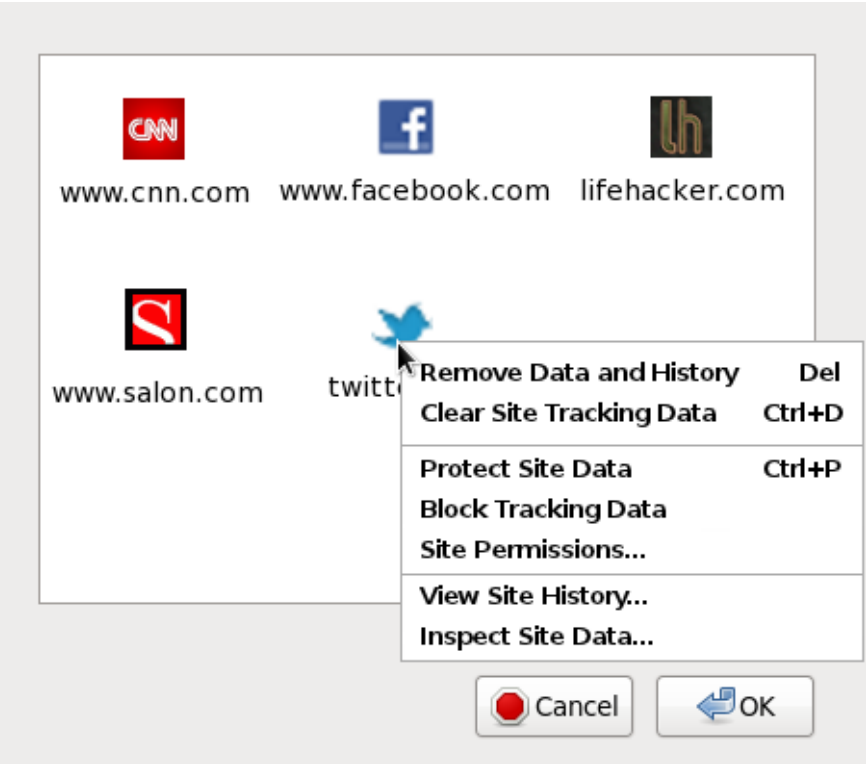
To ensure Tor Browser directory isolation, we set **browser.download.useDownloadDir**, **browser.shell.checkDefaultBrowser**, and **browser.download.manager.addToRecentDocs**. We also set the $HOME environment variable to be the Tor Browser extraction directory.

4.5. Cross-Origin Identifier Unlinkability

The Cross-Origin Identifier Unlinkability design requirement is satisfied through first party isolation of all browser identifier sources. First party isolation means that all identifier sources and browser state are scoped (isolated) using the URL bar domain. This scoping is performed in combination with any additional third party scope. When first party isolation is used with explicit identifier storage that already has a constrained third party scope (such as cookies and DOM storage), this approach is referred to as "double-keying".

The benefit of this approach comes not only in the form of reduced linkability, but also in terms of simplified privacy UI. If all stored browser state and permissions become associated with the URL bar origin, the six or seven different pieces of privacy UI governing these identifiers and permissions can become just one piece of UI. For instance, a window that lists the URL bar origin for which browser state exists, possibly with a context-menu option to drill down into specific types of state or permissions. An example of this simplification can be seen in Figure 1.

Figure 1. Improving the Privacy UI



This example UI is a mock-up of how isolating identifiers to the URL bar domain can simplify the privacy UI for all data - not just cookies. Once browser identifiers and site permissions operate on a URL bar basis, the same privacy window can represent browsing history, DOM Storage, HTTP Auth, search form history, login values, and so on within a context menu for each site.

Identifier Unlinkability Defenses in the Tor Browser

Unfortunately, many aspects of browser state can serve as identifier storage, and no other browser vendor or standards body had invested the effort to enumerate or otherwise deal with these vectors for third party tracking. As such, we have had to enumerate and isolate these identifier sources on a piecemeal basis. This has gotten better lately with Mozilla stepping up and helping us with uplifting our patches, and with contributing their own patches where we lacked proper fixes. However, we are not done yet with our unlinkability defense as new identifier sources are still getting added to the web platform. Here is the list that we have discovered and dealt with to date:

1. **Cookies**

   Design Goal: All cookies MUST be double-keyed to the URL bar origin and third-party origin.

   Implementation Status: Double-keying cookies should just work by setting **privacy.firstparty.isolate** to **true**. However, we have not audited that yet and there is still the UI part missing for managing cookies in Private Browsing Mode. We therefore opted to keep third-party cookies disabled for now by setting **network.cookie.cookieBehavior** to **1**.

2. **Cache**

   Design Goal: All cache entries MUST be isolated to the URL bar domain.

   Implementation Status: We isolate the content and image cache to the URL bar domain by setting **privacy.firstparty.isolate** to **true**.

   Furthermore there is the CacheStorage API. That one is currently not available in Tor Browser as we do not allow third party cookies and are in Private Browsing Mode by default. As the cache entries are written to disk the CacheStorage API got disabled in that mode in Firefox, similar to how IndexedDB is handled. There are thoughts about enabling it by providing a memory-only database but that is still work in progress. But even if users are leaving the Private Browsing Mode and are enabling third party cookies the storage is isolated to the URL bar domain by **privacy.firstparty.isolate** set to **true**.

   Finally, we have the asm.js cache. The cache entry of the script is (among others things, like type of CPU, build ID, source characters of the asm.js module etc.) keyed to the origin of the script. Lacking a good solution for binding it to the URL bar domain instead we decided to disable asm.js for the time being by setting **javascript.options.asmjs** to **false**. It remains to be seen whether keying the cache entry e.g. to the source characters of the asm.js module helps to avoid using it for cross-origin tracking of users. We did not investigate that yet.

3. **HTTP Authentication**

   HTTP Authorization headers can be used to encode silent third party tracking identifiers. To prevent this, we set **privacy.firstparty.isolate** to **true**.

4. **DOM Storage**

   DOM storage for third party domains MUST be isolated to the URL bar domain, to prevent linkability between sites. We achieve this by setting **privacy.firstparty.isolate** to **true**.

5. **IndexedDB Storage**

   IndexedDB storage for third party domains MUST be isolated to the URL bar domain, to prevent linkability between sites. By default IndexedDB storage is disabled as Tor Browser is using Firefox's Private Browsing Mode and does not allow third party cookies. There are thoughts about enabling this API in Private Browsing Mode as well but that is still work in progress. However, if users are leaving this mode and are enabling third party cookies, isolation to the URL bar is achieved, though, by **privacy.firstparty.isolate** set to **true**.

6. **Flash cookies**

   Design Goal: Users should be able to click-to-play flash objects from trusted sites. To make this behavior unlinkable, we wish to include a settings file for all platforms that disables flash cookies using the Flash settings manager.

   Implementation Status: We are currently having difficulties causing Flash player to use this settings file on Windows, so Flash remains difficult to enable.

7. **SSL+TLS session resumption**

   Design Goal: TLS session resumption tickets and SSL Session IDs MUST be limited to the URL bar domain.

   Implementation Status: We disable TLS Session Tickets and SSL Session IDs by setting **security.ssl.disable_session_identifiers** to **true**. To compensate for the increased round trip latency from disabling these performance optimizations, we also enable TLS False Start via the Firefox Pref **security.ssl.enable_false_start**. However, URL bar domain isolation should be working both for session tickets and session IDs but we have not verified that yet.

8. **Tor circuit and HTTP connection linkability**

   Design Goal: Tor circuits and HTTP connections from a third party in one URL bar origin MUST NOT be reused for that same third party in another URL bar origin.

   Implementation Status: The isolation functionality is provided by a Torbutton component that sets the SOCKS username and password for each request. The Tor client has logic to prevent connections with different SOCKS usernames and passwords from using the same Tor circuit. Firefox has existing logic to ensure that connections with SOCKS proxies do not re-use existing HTTP Keep-Alive connections unless the proxy settings match. We extended this logic to cover SOCKS username and password authentication, providing us with HTTP Keep-Alive unlinkability.

   While the vast majority of web requests adheres to the circuit and connection unlinkability requirement there are still corner cases we need to treat separately or that lack a fix altogether.

9. **SharedWorkers**

   SharedWorkers are a special form of JavaScript Worker threads that have a shared scope between all threads from the same Javascript origin. They MUST be isolated to the URL bar domain. I.e. a SharedWorker launched from a third party from one URL bar domain MUST NOT have access to the objects created by that same third party loaded under another URL bar domain. This functionality is provided by setting **privacy.firstparty.isolate** to **true**.

10. **blob: URIs (URL.createObjectURL)**

    The URL.createObjectURL API allows a site to load arbitrary content into a random UUID that is stored in the user's browser, and this content can be accessed via a URL of the form **blob:UUID** from any other content element anywhere on the web. While this UUID value is neither under control of

the site nor predictable, it can still be used to tag a set of users that are of high interest to an adversary.

URIs created with URL.createObjectURL MUST be limited in scope to the first party URL bar domain that created them. We provide the isolation in Tor Browser by setting **privacy.firstparty.isolate** to **true**.

11. **SPDY and HTTP/2**

    Design Goal: SPDY and HTTP/2 connections MUST be isolated to the URL bar domain. Furthermore, all associated means that could be used for cross-domain user tracking (alt-svc headers come to mind) MUST adhere to this design principle as well.

    Implementation status: SPDY and HTTP/2 are currently disabled by setting the Firefox preferences **network.http.spdy.enabled**, **network.http.spdy.enabled.v2**, **network.http.spdy.enabled.v3**, **network.http.spdy.enabled.v3-1**, **network.http.spdy.enabled.http2**, **network.http.spdy.enabled.http2draft**, **network.http.altsvc.enabled**, and **network.http.altsvc.oe** to **false**.

12. **Automated cross-origin redirects**

    Design Goal: To prevent attacks aimed at subverting the Cross-Origin Identifier Unlinkability privacy requirement, the browser MUST NOT store any identifiers (cookies, cache, DOM storage, HTTP auth, etc) for cross-origin redirect intermediaries that do not prompt for user input. For example, if a user clicks on a bit.ly URL that redirects to a doubleclick.net URL that finally redirects to a cnn.com URL, only cookies from cnn.com should be retained after the redirect chain completes.

    Non-automated redirect chains that require user input at some step (such as federated login systems) SHOULD still allow identifiers to persist.

    Implementation status: There are numerous ways for the user to be redirected, and the Firefox API support to detect each of them is poor. We have a trac bug open to implement what we can.

13. **window.name**

    window.name is a magical DOM property that for some reason is allowed to retain a persistent value for the lifespan of a browser tab. It is possible to utilize this property for identifier storage.

    In order to eliminate non-consensual linkability but still allow for sites that utilize this property to function, we reset the window.name property of tabs in Torbutton every time we encounter a blank Referer. This behavior allows window.name to persist for the duration of a click-driven navigation session, but as soon as the user enters a new URL or navigates between HTTPS/HTTP schemes, the property is cleared.

14. **Auto form-fill**

    We disable the password saving functionality in the browser as part of our Disk Avoidance requirement. However, since users may decide to re-enable disk history records and password saving, we also set the signon.autofillForms preference to false to prevent saved values from immediately populating fields upon page load. Since JavaScript can read these values as soon as they appear, setting this preference prevents automatic linkability from stored passwords.

15. **HSTS and HPKP supercookies**

    An extreme (but not impossible) attack to mount is the creation of HSTS supercookies. Since HSTS effectively stores one bit of information per domain name, an adversary in possession of numerous domains can use them to construct cookies based on stored HSTS state.

HPKP provides a mechanism for user tracking across domains as well. It allows abusing the requirement to provide a backup pin and the option to report a pin validation failure. In a tracking scenario every user gets a unique SHA-256 value serving as backup pin. This value is sent back after (deliberate) pin validation failures working in fact as a cookie.

Design Goal: HSTS and HPKP MUST be isolated to the URL bar domain.

Implementation Status: Currently, HSTS and HPKP state is both cleared by New Identity, but we don't defend against the creation and usage of any of these supercookies between **New Identity** invocations.

16. **Broadcast Channels**

    The BroadcastChannel API allows cross-site communication within the same origin. However, to avoid cross-origin linkability broadcast channels MUST instead be isolated to the URL bar domain.

    We provide the isolation in Tor Browser by setting **privacy.firstparty.isolate** to **true**.

17. **OCSP**

    OCSP requests go to Certificate Authorities (CAs) to check for revoked certificates. They are sent once the browser is visiting a website via HTTPS and no cached results are available. Thus, to avoid information leaks, e.g. to exit relays, OCSP requests MUST go over the same circuit as the HTTPS request causing them and MUST therefore be isolated to the URL bar domain. The resulting cache entries MUST be bound to the URL bar domain as well. This functionality is provided by setting **privacy.firstparty.isolate** to **true**.

18. **Favicons**

    Design Goal: When visiting a website its favicon is fetched via a request originating from the browser itself (similar to the OCSP mechanism mentioned in the previous section). Those requests MUST be isolated to the URL bar domain.

    Implementation Status: Favicon requests are isolated to the URL bar domain by setting **privacy.firstparty.isolate** to **true**. However, we need an additional Firefox patch to take care of favicons in tab list menuitems.

19. **mediasource: URIs and MediaStreams**

    Much like blob URLs, mediasource: URIs and MediaStreams can be used to tag users. Therefore, mediasource: URIs and MediaStreams MUST be isolated to the URL bar domain. This functionality is provided by setting **privacy.firstparty.isolate** to **true**.

20. **Speculative and prefetched connections**

    Firefox provides the feature to connect speculatively to remote hosts if that is either indicated in the HTML file (e.g. by link rel="preconnect" and rel="prefetch") or otherwise deemed beneficial.

    Firefox does not support rel="prerender", and Mozilla has disabled speculative connections and rel="preconnect" usage where a proxy is used (see comment 3 in bug 18762 for further details). Explicit prefetching via the rel="prefetch" attribute is still performed, however.

    All pre-loaded links and speculative connections MUST be isolated to the URL bar domain, if enabled. This includes isolating both Tor circuit use, as well as the caching and associate browser state for the prefetched resource.

    For automatic speculative connects and rel="preconnect", we leave them disabled as per the Mozilla default for proxy settings. However, if enabled, speculative connects will be isolated to the proper

first party Tor circuit by the same mechanism as is used for HTTP Keep-Alive. This is true for rel="prefetch" requests as well. For rel="preconnect", we set **privacy.firstparty.isolate** to **true**. This isolation makes both preconnecting and cache warming via rel="prefetch" ineffective for links to domains other than the current URL bar domain. For links to the same domain as the URL bar domain, the full cache warming benefit is obtained. As an optimization, any preconnecting to domains other than the current URL bar domain can thus be disabled (perhaps with the exception of frames), but we do not do this. We allow these requests to proceed, but we isolate them.

21. **Permissions API**

    The Permissions API allows a website to query the status of different permissions. Although permissions are keyed to the origin, that is not enough to alleviate cross-linkability concerns: the combined permission state could work like an identifier given more and more permissions and their state being accessible under this API.

    Design Goal: Permissions MUST be isolated to the URL bar domain.

    Implementation Status: Right now we provide a Firefox patch that makes sure permissions are isolated to the URL bar domain.

For more details on identifier linkability bugs and enhancements, see the tbb-linkability tag in our bugtracker

4.6. Cross-Origin Fingerprinting Unlinkability

Browser fingerprinting is the act of inspecting browser behaviors and features in an attempt to differentiate and track individual users.

Fingerprinting attacks are typically broken up into passive and active vectors. Passive fingerprinting makes use of any information the browser provides automatically to a website without any specific action on the part of the website. Active fingerprinting makes use of any information that can be extracted from the browser by some specific website action, usually involving JavaScript. Some definitions of browser fingerprinting also include supercookies and cookie-like identifier storage, but we deal with those issues separately in the preceding section on identifier linkability.

For the most part, however, we do not differentiate between passive or active fingerprinting sources, since many active fingerprinting mechanisms are very rapid, and can be obfuscated or disguised as legitimate functionality.

Instead, we believe fingerprinting can only be rationally addressed if we understand where the problem comes from, what sources of issues are the most severe, what types of defenses are suitable for which sources, and have a consistent strategy for designing defenses that maximizes our ability to study defense efficacy. The following subsections address these issues from a high level, and we then conclude with a list of our current specific defenses.

Sources of Fingerprinting Issues

All browser fingerprinting issues arise from one of four primary sources: end-user configuration details, device and hardware characteristics, operating system vendor and version differences, and browser vendor and version differences. Additionally, user behavior itself provides one more source of potential fingerprinting.

In order to help prioritize and inform defenses, we now list these sources in order from most severe to least severe in terms of the amount of information they reveal, and describe them in more detail.

1. **End-user Configuration Details**

End-user configuration details are by far the most severe threat to fingerprinting, as they will quickly provide enough information to uniquely identify a user. We believe it is essential to avoid exposing platform configuration details to website content at all costs. We also discourage excessive fine-grained customization of Tor Browser by minimizing and aggregating user-facing privacy and security options, as well as by discouraging the use of additional plugins and addons. When it is necessary to expose configuration details in the course of providing functionality, we strive to do so only on a per-site basis via site permissions, to avoid linkability.

2. **Device and Hardware Characteristics**

   Device and hardware characteristics can be determined in three ways: they can be reported explicitly by the browser, they can be inferred through browser functionality, or they can be extracted through statistical measurements of system performance. We are most concerned with the cases where this information is either directly reported or can be determined via a single use of an API or feature, and prefer to either alter functionality to prevent exposing the most variable aspects of these characteristics, place such features behind site permissions, or disable them entirely.

   On the other hand, because statistical inference of system performance requires many iterations to achieve accuracy in the face of noise and concurrent activity, we are less concerned with this mechanism of extracting this information. We also expect that reducing the resolution of JavaScript's time sources will significantly increase the duration of execution required to extract accurate results, and thus make statistical approaches both unattractive and highly noticeable due to excessive resource consumption.

3. **Operating System Vendor and Version Differences**

   Operating system vendor and version differences permeate many different aspects of the browser. While it is possible to address these issues with some effort, the relative lack of diversity in operating systems causes us to primarily focus our efforts on passive operating system fingerprinting mechanisms at this point in time. For the purposes of protecting user anonymity, it is not strictly essential that the operating system be completely concealed, though we recognize that it is useful to reduce this differentiation ability where possible, especially for cases where the specific version of a system can be inferred.

4. **User Behavior**

   While somewhat outside the scope of browser fingerprinting, for completeness it is important to mention that users themselves theoretically might be fingerprinted through their behavior while interacting with a website. This behavior includes e.g. keystrokes, mouse movements, click speed, and writing style. Basic vectors such as keystroke and mouse usage fingerprinting can be mitigated by altering JavaScript's notion of time. More advanced issues like writing style fingerprinting are the domain of other tools.

5. **Browser Vendor and Version Differences**

   Due to vast differences in feature set and implementation behavior even between different (minor) versions of the same browser, browser vendor and version differences are simply not possible to conceal in any realistic way. It is only possible to minimize the differences among different installations of the same browser vendor and version. We make no effort to mimic any other major browser vendor, and in fact most of our fingerprinting defenses serve to differentiate Tor Browser users from normal Firefox users. Because of this, any study that lumps browser vendor and version differences into its analysis of the fingerprintability of a population is largely useless for evaluating either attacks or defenses. Unfortunately, this includes popular large-scale studies such as Panopticlick and Am I Unique. To gather usable data about Tor Browser's fingerprinting defenses we launched a Google Summer of Code project in 2016, called FPCentral, with the aim to provide us an own testbed. We set this up during 2017 and have it available now for further integration into our

quality assurance efforts and possible research into improving our fingerprinting defenses and measuring their effectiveness.

General Fingerprinting Defenses

To date, the Tor Browser team has concerned itself only with developing defenses for APIs that have already been standardized and deployed. Once an API or feature has been standardized and widely deployed, defenses to the associated fingerprinting issues tend to have only a few options available to compensate for the lack of up-front privacy design. In our experience, so far these options have been limited to value spoofing, subsystem modification or reimplementation, virtualization, site permissions, and feature removal. We will now describe these options and the fingerprinting sources they tend to work best with.

1. **Value Spoofing**

   Value spoofing can be used for simple cases where the browser provides some aspect of the user's configuration details, devices, hardware, or operating system directly to a website. It becomes less useful when the fingerprinting method relies on behavior to infer aspects of the hardware or operating system, rather than obtain them directly.

2. **Subsystem Modification or Reimplementation**

   In cases where simple spoofing is not enough to properly conceal underlying device characteristics or operating system details, the underlying subsystem that provides the functionality for a feature or API may need to be modified or completely reimplemented. This is most common in cases where customizable or version-specific aspects of the user's operating system are visible through the browser's featureset or APIs, usually because the browser directly exposes OS-provided implementations of underlying features. In these cases, such OS-provided implementations must be replaced by a generic implementation, or at least modified by an implementation wrapper layer that makes effort to conceal any user-customized aspects of the system.

3. **Virtualization**

   Virtualization is needed when simply reimplementing a feature in a different way is insufficient to fully conceal the underlying behavior. This is most common in instances of device and hardware fingerprinting, but since the notion of time can also be virtualized, virtualization also can apply to any instance where an accurate measurement of wall clock time is required for a fingerprinting vector to attain high accuracy.

4. **Site Permissions**

   In the event that reimplementation or virtualization is too expensive in terms of performance or engineering effort, and the relative expected usage of a feature is rare, site permissions can be used to prevent the usage of a feature for cross-site tracking. Unfortunately, site permissions become less effective once a feature is already widely overused and abused by many websites, since warning fatigue typically sets in for most users after just a few permission requests.

5. **Feature or Functionality Removal**

   Due to the current bias in favor of invasive APIs that expose the maximum amount of platform information, some features and APIs are simply not salvageable in their current form. When such invasive features serve only a narrow domain or use case, or when there are alternate ways of accomplishing the same task, these features and/or certain aspects of their functionality may be simply removed.

Strategies for Defense: Randomization versus Uniformity

When applying a form of defense to a specific fingerprinting vector or source, there are two general strategies available: either the implementation for all users of a single browser version can be made to behave as uniformly as possible, or the user agent can attempt to randomize its behavior so that each interaction between a user and a site provides a different fingerprint.

Although some research suggests that randomization can be effective, so far striving for uniformity has generally proved to be a better strategy for Tor Browser for the following reasons:

1. **Evaluation and measurement difficulties**

   The fact that randomization causes behaviors to differ slightly with every site visit makes it appealing at first glance, but this same property makes it very difficult to objectively measure its effectiveness. By contrast, an implementation that strives for uniformity is very simple to evaluate. Despite their current flaws, a properly designed version of Panopticlick or Am I Unique could report the entropy and uniqueness rates for all users of a single user agent version, without the need for complicated statistics about the variance of the measured behaviors. FPCentral is trying to achieve that for Tor Browser by providing feedback on acceptable browser properties and giving guidance on possible improvements.

   Randomization (especially incomplete randomization) may also provide a false sense of security. When a fingerprinting attempt makes naive use of randomized information, a fingerprint will appear unstable, but may not actually be sufficiently randomized to impede a dedicated adversary. Sophisticated fingerprinting mechanisms may either ignore randomized information, or incorporate knowledge of the distribution and range of randomized values into the creation of a more stable fingerprint (by either removing the randomness, modeling it, or averaging it out).

2. **Randomization is not a shortcut**

   While many end-user configuration details that the browser currently exposes may be safely replaced by false information, randomization of these details must be just as exhaustive as an approach that seeks to make these behaviors uniform. When confronting either strategy, the adversary can still make use of any details which have not been altered to be either sufficiently uniform or sufficiently random.

   Furthermore, the randomization approach seems to break down when it is applied to deeper issues where underlying system functionality is directly exposed. In particular, it is not clear how to randomize the capabilities of hardware attached to a computer in such a way that it either convincingly behaves like other hardware, or such that the exact properties of the hardware that vary from user to user are sufficiently randomized. Similarly, truly concealing operating system version differences through randomization may require multiple reimplementations of the underlying operating system functionality to ensure that every operating system version is covered by the range of possible behaviors.

3. **Usability issues**

   When randomization is introduced to features that affect site behavior, it can be very distracting for this behavior to change between visits of a given site. For the simplest cases, this will lead to minor visual nuisances. However, when this information affects reported functionality or hardware characteristics, sometimes a site will function one way on one visit, and another way on a subsequent visit.

4. **Performance costs**

   Randomizing involves performance costs. This is especially true if the fingerprinting surface is large (like in a modern browser) and one needs more elaborate randomizing strategies (including randomized virtualization) to ensure that the randomization fully conceals the true behavior. Many

calls to a cryptographically secure random number generator during the course of a page load will both serve to exhaust available entropy pools, as well as lead to increased computation while loading a page.

5. **Increased vulnerability surface**

   Improper randomization might introduce a new fingerprinting vector, as the process of generating the values for the fingerprintable attributes could be itself susceptible to side-channel attacks, analysis, or exploitation.

Specific Fingerprinting Defenses in the Tor Browser

The following defenses are listed roughly in order of most severe fingerprinting threat first. This ordering is based on the above intuition that user configurable aspects of the computer are the most severe source of fingerprintability, followed by device characteristics and hardware, and then finally operating system vendor and version information.

Where our actual implementation differs from an ideal solution, we separately describe our **Design Goal** and our **Implementation Status**.

1. **Plugins**

   Plugins add to fingerprinting risk via two main vectors: their mere presence in window.navigator.plugins (because they are optional, end-user installed third party software), as well as their internal functionality.

   Design Goal: All plugins that have not been specifically audited or sandboxed MUST be disabled. To reduce linkability potential, even sandboxed plugins SHOULD NOT be allowed to load objects until the user has clicked through a click-to-play barrier. Additionally, version information SHOULD be reduced or obfuscated until the plugin object is loaded. For Flash, we wish to provide a settings.sol file to disable Flash cookies, and to restrict P2P features that are likely to bypass proxy settings. We'd also like to restrict access to fonts and other system information (such as IP address and MAC address) in such a sandbox.

   Implementation Status: Currently, we entirely disable all plugins in Tor Browser. However, as a compromise due to the popularity of Flash, we allow users to re-enable Flash, and flash objects are blocked behind a click-to-play barrier that is available only after the user has specifically enabled plugins. Flash is the only plugin available, the rest are entirely blocked from loading by the Firefox patches mentioned in the Proxy Obedience section. We also set the Firefox preference **plugin.expose_full_path** to **false**, to avoid leaking plugin installation information.

2. **HTML5 Canvas Image Extraction**

   After plugins and plugin-provided information, we believe that the HTML5 Canvas is the single largest fingerprinting threat browsers face today. Studies show that the Canvas can provide an easy-access fingerprinting target: The adversary simply renders WebGL, font, and named color data to a Canvas element, extracts the image buffer, and computes a hash of that image data. Subtle differences in the video card, font packs, and even font and graphics library versions allow the adversary to produce a stable, simple, high-entropy fingerprint of a computer. In fact, the hash of the rendered image can be used almost identically to a tracking cookie by the web server.

   In some sense, the canvas can be seen as the union of many other fingerprinting vectors. If WebGL is normalized through software rendering, system colors were standardized, and the browser shipped a fixed collection of fonts (see later points in this list), it might not be necessary to create a canvas permission. However, until then, to reduce the threat from this vector, we have patched Firefox to prompt before returning valid image data to the Canvas APIs, and for access to isPointInPath and

related functions. Moreover, we put media streams on a canvas behind the site permission in that patch as well. If the user hasn't previously allowed the site in the URL bar to access Canvas image data, pure white image data is returned to the JavaScript APIs. Extracting canvas image data by third parties is not allowed, though.

3. **Open TCP Port and Local Network Fingerprinting**

   In Firefox, by using either WebSockets or XHR, it is possible for remote content to enumerate the list of TCP ports open on 127.0.0.1, as well as on any other machines on the local network. In other browsers, this can be accomplished by DOM events on image or script tags. This open vs filtered vs closed port list can provide a very unique fingerprint of a machine, because it essentially enables the detection of many different popular third party applications and optional system services (Skype, Bitcoin, Bittorrent and other P2P software, SSH ports, SMB and related LAN services, CUPS and printer daemon config ports, mail servers, and so on). It is also possible to determine when ports are closed versus filtered/blocked (and thus probe custom firewall configuration).

   In Tor Browser, we prevent access to 127.0.0.1/localhost by ensuring that even these requests are still sent by Firefox to our SOCKS proxy (ie we set **network.proxy.no_proxies_on** to the empty string). The local Tor client then rejects them, since it is configured to proxy for internal IP addresses by default. Access to the local network is forbidden via the same mechanism. We also disable the WebRTC API as mentioned previously, since even if it were usable over Tor, it still currently provides the local IP address and associated network information to websites. Additionally, we rip out the option to collect local IP addresses via the NetworkInfoService.

4. **Invasive Authentication Mechanisms (NTLM and SPNEGO)**

   Both NTLM and SPNEGO authentication mechanisms can leak the hostname, and in some cases the current username. The only reason why these aren't a more serious problem is that they typically involve user interaction, and likely aren't an attractive vector for this reason. However, because it is not clear if certain carefully-crafted error conditions in these protocols could cause them to reveal machine information and still fail silently prior to the password prompt, these authentication mechanisms should either be disabled, or placed behind a site permission before their use. We simply disable them with a patch.

5. **USB Device ID Enumeration via the GamePad API**

   The GamePad API provides web pages with the USB device id, product id, and driver name of all connected game controllers, as well as detailed information about their capabilities.

   It's our opinion that this API needs to be completely redesigned to provide an abstract notion of a game controller rather than offloading all of the complexity associated with handling specific game controller models to web content authors. For systems without a game controller, a standard controller can be virtualized through the keyboard, which will serve to both improve usability by normalizing user interaction with different games, as well as eliminate fingerprinting vectors. Barring that, this API should be behind a site permission in Private Browsing Modes. For now though, we simply disable it via the pref **dom.gamepad.enabled**.

6. **Fonts**

   According to the Panopticlick study, fonts provide the most linkability when they are available as an enumerable list in file system order, via either the Flash or Java plugins. However, it is still possible to use CSS and/or JavaScript to query for the existence of specific fonts. With a large enough pre-built list to query, a large amount of fingerprintable information may still be available, especially given that additional fonts often end up installed by third party software and for multilingual support.

Design Goal: Font-based fingerprinting MUST be rendered ineffective

Implementation Status: We investigated shipping a predefined set of fonts to all of our users allowing only those fonts to be used by websites at the exclusion of system fonts. We are currently following this approach, which has been suggested by researchers previously. This defense is available for all three supported platforms: Windows, macOS, and Linux, although the implementations vary in detail.

For Windows and macOS we use a preference, **font.system.whitelist**, to restrict fonts being used to those in the whitelist. This functionality is provided by setting **privacy.resistFingerprinting** to **true**. The whitelist for Windows and macOS contains both a set of Noto fonts which we bundle and fonts provided by the operating system. For Linux systems we only bundle fonts and deploy a **fonts.conf** file to restrict the browser to use those fonts exclusively. In addition to that we set the **font.name*** preferences for macOS and Linux to make sure that a given code point is always displayed with the same font. This is not guaranteed even if we bundle all the fonts Tor Browser uses as it can happen that fonts are loaded in a different order on different systems. Setting the above mentioned preferences works around this issue by specifying the font to use explicitly.

Allowing fonts provided by the operating system for Windows and macOS users is currently a compromise between fingerprintability resistance and usability concerns. We are still investigating the right balance between them and have created a ticket in our bug tracker to summarize the current state of our defense and future work that remains to be done.

7. **Monitor, Widget, and OS Desktop Resolution**

Both CSS and JavaScript have access to a lot of information about the screen resolution, usable desktop size, OS widget size, toolbar size, title bar size, and OS desktop widget sizing information that are not at all relevant to rendering and serve only to provide information for fingerprinting. Since many aspects of desktop widget positioning and size are user configurable, these properties yield customized information about the computer, even beyond the monitor size.

Design Goal: Our design goal here is to reduce the resolution information down to the bare minimum required for properly rendering inside a content window. We intend to report all rendering information correctly with respect to the size and properties of the content window, but report an effective size of 0 for all border material, and also report that the desktop is only as big as the inner content window. Additionally, new browser windows are sized such that their content windows are one of a few fixed sizes based on the user's desktop resolution. In addition, to further reduce resolution-based fingerprinting, we are investigating zoom/viewport-based mechanisms that might allow us to always report the same desktop resolution regardless of the actual size of the content window, and simply scale to make up the difference. As an alternative to zoom-based solutions we are testing a different approach in our alpha series that tries to round the browser window at all times to a multiple 200x100 pixels. Regardless which solution we finally pick, until it will be available the user should also be informed that maximizing their windows can lead to fingerprintability under the current scheme.

Implementation Status: We automatically resize new browser windows to a 200x100 pixel multiple based on desktop resolution by backporting patches from bug 1330882 and setting **privacy.resistfingerprinting** to **true**. To minimize the effect of the long tail of large monitor sizes, we also cap the window size at 1000 pixels in each direction. In addition to that we set **privacy.resistFingerprinting** to **true** to use the client content window size for window.screen, and to report a window.devicePixelRatio of 1.0. Similarly, we use that preference to return content window relative points for DOM events. We also force popups to open in new tabs (via **browser.link.open_newwindow.restriction**), to avoid full-screen popups inferring information about the browser resolution. In addition, we prevent auto-maximizing on browser start, and inform users that maximized windows are detrimental to privacy in this mode.

8. **Display Media information**

   Beyond simple resolution information, a large amount of so-called "Media" information is also exported to content. Even without JavaScript, CSS has access to a lot of information about the device orientation, system theme colors, and other desktop and display features that are not at all relevant to rendering and also user configurable. Most of this information comes from CSS Media Queries, but Mozilla has exposed several user and OS theme defined color values to CSS as well.

   Design Goal: A website MUST NOT be able infer anything that the user has configured about their computer. Additionally, it SHOULD NOT be able to infer machine-specific details such as screen orientation or type.

   Implementation Status: We set **ui.use_standins_for_native_colors** to **true** and provide a Firefox patch to report a fixed set of system colors to content window CSS, and prevent detection of font smoothing on macOS with the help of **privacy.resistFingerprinting** set to **true**. We use the same preference, too, to always report landscape-primary for the screen orientation.

9. **WebGL**

   WebGL is fingerprintable both through information that is exposed about the underlying driver and optimizations, as well as through performance fingerprinting.

   Because of the large amount of potential fingerprinting vectors and the previously unexposed vulnerability surface, we deploy a similar strategy against WebGL as for plugins. First, WebGL Canvases have click-to-play placeholders (provided by NoScript), and do not run until authorized by the user. Second, we obfuscate driver information by setting the Firefox preferences **webgl.disable-extensions**, **webgl.min_capability_mode**, and **webgl.disable-fail-if-major-performance-caveat** to **true** which reduces the information provided by the following WebGL API calls: **getParameter()**, **getSupportedExtensions()**, and **getExtension()**. Furthermore, WebGL2 is disabled by setting **webgl.enable-webgl2** to **false**. To make the minimal WebGL mode usable we additionally normalize its properties with a Firefox patch.

   Another option for WebGL might be to use software-only rendering, using a library such as Mesa. The use of such a library would avoid hardware-specific rendering differences.

10. **MediaDevices API**

    The MediaDevices API provides access to connected media input devices like cameras and microphones, as well as screen sharing. In particular, it allows web content to easily enumerate those devices with **MediaDevices.enumerateDevices()**. This relies on WebRTC being compiled in which we currently don't do. Nevertheless, we disable this feature for now as a defense-in-depth by setting **media.peerconnection.enabled** and **media.navigator.enabled** to **false**.

11. **MIME Types**

    Which MIME Types are registered with an operating system depends to a great deal on the application software and/or drivers a user chose to install. Web pages can not only estimate the amount of MIME types registered by checking **navigator.mimetypes.length**. Rather, they are even able to test whether particular MIME types are available which can have a non-negligible impact on a user's fingerprint. We prevent both of these information leaks by setting **privacy.resistfingerprinting** to **true**.

12. **Web Speech API**

    The Web Speech API consists of two parts: SpeechSynthesis (Text-to-Speech) and SpeechRecognition (Asynchronous Speech Recognition). The latter is still disabled in Firefox. However, the former is enabled by default and there is the risk that **speechSynthesis.getVoices()** has access to computer-

specific speech packages making them available in an enumerable fashion. Moreover, there are callbacks that would allow JavaScript to time how long a phrase takes to be "uttered". To prevent both we set **media.webspeech.synth.enabled** to **false**.

13. **Touch API**

    Touch events are able to reveal the absolute screen coordinates of a device which would defeat our approach to mitigate leaking the screen size as described above. In order to prevent that we implemented two defenses: first we disable the Touch API by setting **dom.w3c_touch_events.enabled** to **false**. Second, for those user that really need or want to have this API available we patched the code to give content-window related coordinates back. Furthermore, we made sure that the touch area described by **Touch.radiusX**, **Touch.radiusY**, and **Touch.rotationAngle** does not leak further information and **Touch.force** does not reveal how much pressure a user applied to the surface. That is achieved by a direct Firefox patch which reports back **1** for the first two properties and **0.0** for the two last ones.

14. **Battery Status API**

    The Battery Status API provides access to information about the system's battery charge level. From Firefox 52 on it is disabled for web content. Initially, it was possible on Linux to get a double-precision floating point value for the charge level, which means there was a large number of possible values making it almost behave like an identifier allowing to track a user cross-origin. But still after that got fixed (and on other platforms where the precision was just two significant digits anyway) the risk for tracking users remained as combined with the **chargingTime** and **dischargingTime** the possible values got estimated to be in the millions under normal conditions. We avoid all those possible issues with disabling the Battery Status API by setting **dom.battery.enabled** to **false**.

15. **System Uptime**

    It is possible to get the system uptime of a Tor Browser user by querying the **Event.timestamp** property. We avoid this by setting **dom.event.highrestimestamp.enabled** to **true**. This might seem to be counterintuitive at first glance but the effect of setting that preference to `true` is a normalization of `evt.timestamp` and `new Event('').timeStamp`. Together with clamping the timer resolution to 100ms this provides an effective means against system uptime fingerprinting.

16. **Keyboard Layout Fingerprinting**

    KeyboardEvents provide a way for a website to find out information about the keyboard layout of its visitors. In fact there are several dimensions to this fingerprinting vector. The **KeyboardEvent.code** property represents a physical key that can't be changed by the keyboard layout nor by the modifier state. On the other hand the **KeyboardEvent.key** property contains the character that is generated by that key. This is dependent on things like keyboard layout, locale and modifier keys.

    Design Goal: Websites MUST NOT be able to infer any information about the keyboard of a Tor Browser user.

    Implementation Status: We provide two Firefox patches that take care of spoofing **KeyboardEvent.code** and **KeyboardEvent.keyCode** by providing consensus (US-English-style) fake properties. This is achieved by hiding the user's use of the numpad, and any non-QWERTY US English keyboard. Characters from non-en-US languages are currently returning an empty **KeyboardEvent.code** and a **KeyboardEvent.keyCode** of **0**. Moreover, neither **Alt** or **Shift**, or **AltGr** keyboard events are reported to content.

    We are currently not taking the actually deployed browser locale or the locale indicated by a loaded

document into account when spoofing the keyboard layout. We think that would be the right thing to do in the longer run, to mitigate possible usability issues and broken functionality on websites. Similarily to how users of non-english Tor Browser bundles right now can choose between keeping the Accept header spoofed or not they would then be able to keep a spoofed english keyboard or a spoofed one depending on the actual Tor Browser locale or language of the document.

17. **User Agent and HTTP Headers**

    Design Goal: All Tor Browser users MUST provide websites with an identical user agent and HTTP header set for a given request type. We omit the Firefox minor revision, and report a popular Windows platform. If the software is kept up to date, these headers should remain identical across the population even when updated.

    Implementation Status: Firefox provides several options for controlling the browser user agent string which we leverage. We also set similar prefs for controlling the Accept-Language and Accept-Charset headers, which we spoof to English by default. Additionally, we remove content script access to Components.interfaces, which can be used to fingerprint OS, platform, and Firefox minor version.

18. **Timing-based Side Channels**

    Attacks based on timing side channels are nothing new in the browser context. Cache-based, cross-site timing, and pixel stealing, to name just a few, got investigated in the past. While their fingerprinting potential varies all timing-based attacks have in common that they need sufficiently fine-grained clocks.

    Design Goal: Websites MUST NOT be able to fingerprint a Tor Browser user by exploiting timing-based side channels.

    Implementation Status: The cleanest solution to timing-based side channels would be to get rid of them. This has been proposed in the research community. However, we remain skeptical as it does not seem to be trivial even considering just a single side channel and more and more potential side channels are showing up. Thus, we rely on disabling all possible timing sources or making them coarse-grained enough in order to render timing side channels unsuitable as a means for fingerprinting browser users.

    We set **dom.enable_user_timing** and **dom.enable_resource_timing** to **false** to disable these explicit timing sources. Furthermore, we clamp the resolution of explicit clocks to 100ms with two Firefox patches. This includes **performance.now()**, **new Date().getTime()** , **audioContext.currentTime**, **canvasStream.currentTime**, **video.currentTime**, **audio.currentTime**, **new File([], "").lastModified** , **new File([], "").lastModifiedDate.getTime()**, **animation.startTime**, **animation.currentTime**, **animation.timeline.currentTime**, and **document.timeline.currentTime**.

    While clamping the clock resolution to 100ms is a step towards mitigating timing-based side channel fingerprinting, it is by no means sufficient. It turns out that it is possible to subvert our clamping of explicit clocks by using implicit ones, e.g. extrapolating the true time by running a busy loop with a predictable operation in it. We are tracking this problem in our bug tracker and are working with the research community and Mozilla to develop and test a proper solution to this part of our defense against timing-based side channel fingerprinting risks.

19. **resource:// and chrome:// URIs Leaks**

    Due to bugs in Firefox it is possible to detect the locale and the platform of a Tor Browser user. Moreover, it is possible to find out the extensions a user has installed. This is done by including resource:// and/or chrome:// URIs into web content, which point to resources included in Tor Browser itself or in installed extensions, and exploiting the different behavior resulting out of that:

the browser raises an exception if a webpage requests a resource but the extension is not installed. This does not happen if the extension is indeed installed but the resource path does not exist.

We believe that it should be impossible for web content to extract information out of a Tor Browser user by deploying resource:// and/or chrome:// URIs. Until this is fixed in Firefox we filter resource:// and chrome:// requests done by web content denying them by default. We need a whitelist of resource:// and chrome:// URIs, though, to avoid breaking parts of Firefox. There are more than a dozen Firefox resources do not aid in fingerprinting Tor Browser users as they are not different on the platforms and in the locales we support.

20. **Locale Fingerprinting**

    In Tor Browser, we provide non-English users the option of concealing their OS and browser locale from websites. It is debatable if this should be as high of a priority as information specific to the user's computer, but for completeness, we attempt to maintain this property.

    Implementation Status: We set the fallback character set to set to windows-1252 for all locales, via **intl.charset.default**. We also set **javascript.use_us_english_locale** to **true** to instruct the JS engine to use en-US as its internal C locale for all Date, Math, and exception handling. Additionally, we provide a patch to use an en-US label for the **isindex**HTML element instead of letting the label leak the browser's UI locale.

21. **Timezone and Clock Offset**

    While the latency in Tor connections varies anywhere from milliseconds to a few seconds, it is still possible for the remote site to detect large differences between the user's clock and an official reference time source.

    Design Goal: All Tor Browser users MUST report the same timezone to websites. Currently, we choose UTC for this purpose, although an equally valid argument could be made for EDT/EST due to the large English-speaking population density (coupled with the fact that we spoof a US English user agent). Additionally, the Tor software should detect if the user's clock is significantly divergent from the clocks of the relays that it connects to, and use this to reset the clock values used in Tor Browser to something reasonably accurate. Alternatively, the browser can obtain this clock skew via a mechanism similar to that used in tlsdate.

    Implementation Status: We set the timezone to UTC with a Firefox patch using the TZ environment variable, which is supported on all platforms. Moreover, with an additional patch just needed for the Windows platform, we make sure the TZ environment variable is respected by the ICU library as well.

22. **JavaScript Performance Fingerprinting**

    JavaScript performance fingerprinting is the act of profiling the performance of various JavaScript functions for the purpose of fingerprinting the JavaScript engine and the CPU.

    Design Goal: We have several potential mitigation approaches to reduce the accuracy of performance fingerprinting without risking too much damage to functionality. Our current favorite is to reduce the resolution of the Event.timeStamp and the JavaScript Date() object, while also introducing jitter. We believe that JavaScript time resolution may be reduced all the way up to the second before it seriously impacts site operation. Our goal with this quantization is to increase the amount of time it takes to mount a successful attack. Mowery et al found that even with the default precision in most browsers, they required up to 120 seconds of amortization and repeated trials to get stable results from their feature set. We intend to work with the research community to establish the optimum trade-off between quantization+jitter and amortization time, as well as identify highly variable JavaScript operations. As long as these attacks take several seconds or more to execute, they

are unlikely to be appealing to advertisers, and are also very likely to be noticed if deployed against a large number of people.

Implementation Status: Currently, our mitigation against performance fingerprinting is to disable Navigation Timing by setting the Firefox preference **dom.enable_performance** to **false**, and to disable the Mozilla Video Statistics API extensions by setting the preference **media.video_stats.enabled** to **false**, too.

23. **Keystroke Fingerprinting**

    Keystroke fingerprinting is the act of measuring key strike time and key flight time. It is seeing increasing use as a biometric.

    Design Goal: We intend to rely on the same mechanisms for defeating JavaScript performance fingerprinting: timestamp quantization and jitter.

    Implementation Status: We clamp keyboard event resolution to 100ms with a Firefox patch.

24. **Amount of Processor Cores (hardwareConcurrency)**

    Modern computers have multiple physical processor cores available in their CPU. For optimum performance, native code typically attempts to run as many threads as there are cores, and **navigator.hardwareConcurrency** makes the number of those threads (i.e. logical processors) available to web content.

    Design Goal: Websites MUST NOT be able to fingerprint a Tor Browser user taking advantage of the amount of logical processors available.

    Implementation Status: We set **dom.maxHardwareConcurrency** to **1** to report the same amount of logical processors for everyone. However, there are probabilistic ways of determining the same information available which we are not defending against currently. Moreover, we might even want to think about a more elaborate approach defending against this fingerprinting technique by not making all users uniform but rather by following a bucket approach as we currently do in our defense against screen size exfiltration.

25. **Web Audio API**

    The Web Audio API provides several means to aid in fingerprinting users. At the simplest level it allows differentiating between users who have the API available and those who don't by checking for an **AudioContext** or **OscillatorNode** object. However, there are more bits of information that the Web Audio API reveals if audio signals generated with an **OscillatorNode** are processed as hardware and software differences influence those results.

    We disable the Web Audio API by setting **dom.webaudio.enabled** to **false**. That has the positive side effect that it disables one of several means to perform ultrasound cross-device tracking as well, which is based on having **AudioContext** available.

26. **MediaError.message**

    The **MediaError** object allows the user agent to report errors that occurred while handling media, for instance using **audio** or **video** elements. The **message** property provides specific diagnostic information to help understanding the error condition. As a defense-in-depth we make sure that no information aiding in fingerprinting is leaking to websites that way **by returning just an empty string**.

27. **Connection State**

    It is possible to monitor the connection state of a browser over time with navigator.onLine. We

prevent this by setting **network.manage-offline-status** to **false**.

28. **Reader View**

    Reader View is a Firefox feature to view web pages clutter-free and easily adjusted to own needs and preferences. To avoid fingerprintability risks we make Tor Browser users uniform by setting **reader.parse-on-load.enabled** to **false** and **browser.reader.detectedFirstArticle** to **true**. This makes sure that documents are not parsed on load as this is disabled on some devices due to memory consumption and we pretend that everybody has already been using that feature in the past.

29. **Contacting Mozilla Services**

    Tor Browser is based on Firefox which is a Mozilla product. Quite naturally, Mozilla is interested in making users aware of new features and in gathering information to learn about the most pressing needs Firefox users are facing. This is often implemented by contacting Mozilla services, be it for displaying further information about a new feature or by sending (aggregated) data back for analysis. While some of those mechanisms are disabled by default on release channels (such as telemetry data) others are not. We make sure that none of those Mozilla services are contacted to avoid possible fingerprinting risks.

    In particular, we disable GeoIP-based search results by setting **browser.search.countryCode** and **browser.search.region** to **US** and **browser.search.geoip.url** to the empty string. Furthermore, we disable Selfsupport and Unified Telemetry by setting **browser.selfsupport.enabled** and **toolkit.telemetry.unified** to **false** and we make sure no related ping is reaching Mozilla by setting **datareporting.healthreport.about.reportUrlUnified** to **data:text/plain,**. The same is done with **datareporting.healthreport.about.reportUrl** and the new tiles feature related **browser.newtabpage.directory.ping** and **browser.newtabpage.directory.source** preferences. **browser.newtabpage.remote** is set to **false** in this context as well, as a defense-in-depth given that this feature is already of by default. Additionally, we disable the UITour backend by setting **browser.uitour.enabled** to **false** and avoid getting Mozilla experiments installed into Tor Browser by flipping **experiments.enabled** to **false**. On the update side we prevent the browser from pinging the new Kinto service for blocklist updates as it is not used for it yet anyway. This is done by setting **services.blocklist.update_enabled** to **false**. The captive portal detection code is disabled as well as it phones home to Mozilla. We set **network.captive-portal-service.enabled** to **false** to achieve that. Unrelated to that we make sure that Mozilla does not get bothered with TLS error reports from Tor Browser users by hiding the respective checkbox with **security.ssl.errorReporting.enabled** set to **false**. And while we have the Push API disabled as there are no Service Workers available in Tor Browser yet, we remove the value for **dom.push.serverURL** as a defense-in-depth. Finally, we provide a patch to prevent Mozilla's websites from querying whether particular extensions are installed and what their state in Tor Browser is by using the **window.navigator.AddonManager** API. As a defense-in-depth the patch makes sure that not only Mozilla's websites can't get at that information but that the whitelist governing this access is empty in general.

    We have Safebrowsing disabled in Tor Browser. In order to avoid pinging providers for list updates we remove the entries for **browser.safebrowsing.provider.mozilla.updateURL** and **browser.safebrowsing.provider.mozilla.gethashURL** (and the values for Google related preferences as well).

30. **Operating System Type Fingerprinting**

    As we mentioned in the introduction of this section, OS type fingerprinting is currently considered a lower priority, due simply to the numerous ways that characteristics of the operating system type may leak into content, and the comparatively low contribution of OS to overall entropy. In particular, there are likely to be many ways to measure the differences in widget size, scrollbar size, and other

rendered details on a page. Also, directly exported OS routines (such as those from the standard C math library) expose differences in their implementations through their return values.

Design Goal: We intend to reduce or eliminate OS type fingerprinting to the best extent possible, but recognize that the effort for reward on this item is not as high as other areas. The entropy on the current OS distribution is somewhere around 2 bits, which is much lower than other vectors which can also be used to fingerprint configuration and user-specific information. You can see the major areas of OS fingerprinting we're aware of using the tbb-fingerprinting-os tag on our bug tracker.

Implementation Status: At least two HTML5 features have a different implementation status across the major OS vendors and/or the underlying hardware: the Network Connection API, and the Sensor API. We disable these APIs through the Firefox preferences **dom.network.enabled** and **device.sensors.enabled**, setting both to **false**.

For more details on fingerprinting bugs and enhancements, see the tbb-fingerprinting tag in our bug tracker

4.7. Long-Term Unlinkability via "New Identity" button

In order to avoid long-term linkability, we provide a "New Identity" context menu option in Torbutton. This context menu option is active if Torbutton can read the environment variables $TOR_CONTROL_PASSWD and $TOR_CONTROL_PORT.

Design Goal:

All linkable identifiers and browser state MUST be cleared by this feature.

Implementation Status:

First, Torbutton disables JavaScript in all open tabs and windows by using both the browser.docShell.allowJavaScript attribute as well as nsIDOMWindowUtil.suppressEventHandling(). We then stop all page activity for each tab using browser.webNavigation.stop(nsIWebNavigation.STOP_ALL). We then clear the site-specific Zoom by temporarily disabling the preference **browser.zoom.siteSpecific**, and clear the GeoIP wifi token URL **geo.wifi.access_token** and the last opened URL preference (if it exists). Each tab is then closed.

After closing all tabs, we then clear the searchbox and findbox text and emit "browser:purge-session-history" (which instructs addons and various Firefox components to clear their session state). Then we manually clear the following state: HTTP auth, SSL state, crypto tokens, OCSP state, site-specific content preferences (including HSTS state), the undo tab history, content and image cache, offline and memory cache, offline storage, Cache storage, IndexedDB storage, asm.js cache, cookies, DOM storage, the safe browsing key, the Google wifi geolocation token (if it exists), and the domain isolator state. We also clear NoScript's site and temporary permissions, and all other browser site permissions.

After the state is cleared, we then close all remaining HTTP Keep-Alive connections and then send the NEWNYM signal to the Tor control port to cause a new circuit to be created.

Finally, a fresh browser window is opened, and the current browser window is closed (this does not spawn a new Firefox process, only a new window). Upon the close of the final window, an unload handler is fired to invoke the garbage collector, which has the effect of immediately purging any blob:UUID URLs that were created by website content via URL.createObjectURL.

4.8. Other Security Measures

In addition to the above mechanisms that are devoted to preserving privacy while browsing, we also have a number of technical mechanisms to address other privacy and security issues.

1. **Security Slider**

   In order to provide vulnerability surface reduction for users that need high security, we have implemented a "Security Slider" to allow users to make a tradeoff between usability and security while minimizing the total number of choices (to reduce fingerprinting). Using metrics collected from Mozilla's bug tracker, we analyzed the vulnerability counts of core components, and used information gathered from a study performed by iSec Partners to inform which features should be disabled at which security levels.

   The Security Slider consists of three positions:

   - **Low (default)**

     At this security level, the preferences are the Tor Browser defaults. This includes three features that were formerly governed by the slider at higher security levels: **gfx.font_rendering.graphite.enabled** is set to **false** now after Mozilla got convinced that leaving it enabled is too risky. Even though Mozilla reverted that decision after another round of fixing critical Graphite bugs, we remain skeptical and keep that feature disabled for now. **network.jar.block-remote-files** is set to **true**. Mozilla tried to block remote JAR files in Firefox 45 but needed to revert that decision due to breaking IBM's iNotes. While Mozilla is working on getting this disabled again we take the protective stance already now and block remote JAR files even on the low security level. Finally, we exempt asm.js from the security slider and block it on all levels. See the Disk Avoidance and the cache linkability concerns in the Cross-Origin Identifier Unlinkability sections for further details.

   - **Medium**

     At this security level, we disable the ION JIT (**javascript.options.ion**), native regular expressions (**javascript.options.native_regexp**), Baseline JIT (**javascript.options.baselinejit**), WebAudio (**media.webaudio.enabled**), MathML (**mathml.disabled**), SVG Opentype font rendering (**gfx.font_rendering.opentype_svg.enabled**), and make HTML5 audio and video click-to-play via NoScript (**noscript.forbidMedia**). Furthermore, we only allow JavaScript to run if it is loaded over HTTPS and the URL bar is HTTPS (by setting **noscript.global** to false and **noscript.globalHttpsWhitelist** to true).

   - **High**

     This security level inherits the preferences from the Medium level, and additionally disables remote fonts (**noscript.forbidFonts**), completely disables JavaScript (by unsetting **noscript.globalHttpsWhitelist**), and disables SVG images (**svg.in-content.enabled**).

2. **Website Traffic Fingerprinting Defenses**

   Website Traffic Fingerprinting is a statistical attack to attempt to recognize specific encrypted website activity.

   Design Goal:

   > We want to deploy a mechanism that reduces the accuracy of useful features available for classification. This mechanism would either impact the true and false positive

accuracy rates, *or* reduce the number of web pages that could be classified at a given accuracy rate.

Ideally, this mechanism would be as light-weight as possible, and would be tunable in terms of overhead. We suspect that it may even be possible to deploy a mechanism that reduces feature extraction resolution without any network overhead. In the no-overhead category, we have HTTPOS and better use of HTTP pipelining and/or SPDY. In the tunable/low-overhead category, we have Adaptive Padding and Congestion-Sensitive BUFLO. It may be also possible to tune such defenses such that they only use existing spare Guard bandwidth capacity in the Tor network, making them also effectively no-overhead.

Implementation Status:

Currently, we patch Firefox to randomize pipeline order and depth. Unfortunately, pipelining is very fragile. Many sites do not support it, and even sites that advertise support for pipelining may simply return error codes for successive requests, effectively forcing the browser into non-pipelined behavior. Firefox also has code to back off and reduce or eliminate the pipeline if this happens. These shortcomings and fallback behaviors are the primary reason that Google developed SPDY as opposed to simply extending HTTP to improve pipelining. It turns out that we could actually deploy exit-side proxies that allow us to use SPDY from the client to the exit node. This would make our defense not only free, but one that actually *improves* performance.

Knowing this, we created this defense as an experimental research prototype to help evaluate what could be done in the best case with full server support. Unfortunately, the bias in favor of compelling attack papers has caused academia to ignore this request thus far, instead publishing only cursory (yet "devastating") evaluations that fail to provide even simple statistics such as the rates of actual pipeline utilization during their evaluations, in addition to the other shortcomings and shortcuts mentioned earlier. We can accept that our defense might fail to work as well as others (in fact we expect it), but unfortunately the very same shortcuts that provide excellent attack results also allow the conclusion that all defenses are broken forever. So sadly, we are still left in the dark on this point.

3. **Privacy-preserving update notification**

In order to inform the user when their Tor Browser is out of date, we perform a privacy-preserving update check asynchronously in the background. The check uses Tor to download the file https://www.torproject.org/projects/torbrowser/RecommendedTBBVersions and searches that version list for the current value for the local preference **torbrowser.version**. If the value from our preference is present in the recommended version list, the check is considered to have succeeded and the user is up to date. If not, it is considered to have failed and an update is needed. The check is triggered upon browser launch, new window, and new tab, but is rate limited so as to happen no more frequently than once every 1.5 hours.

If the check fails, we cache this fact, and update the Torbutton graphic to display a flashing warning icon and insert a menu option that provides a link to our download page. Additionally, we reset the value for the browser homepage to point to a page that informs the user that their browser is out of date.

We also make use of the in-browser Mozilla updater, and have patched the updater to avoid sending OS and Kernel version information as part of its update pings.

5. Build Security and Package Integrity

In the age of state-sponsored malware, we believe it is impossible to expect to keep a single build machine or software signing key secure, given the class of adversaries that Tor has to contend with. For this reason, we have deployed a build system that allows anyone to use our source code to reproduce byte-for-byte identical binary packages to the ones that we distribute.

5.1. Achieving Binary Reproducibility

The GNU toolchain has been working on providing reproducible builds for some time, however a large software project such as Firefox typically ends up embedding a large number of details about the machine it was built on, both intentionally and inadvertently. Additionally, manual changes to the build machine configuration can accumulate over time and are difficult for others to replicate externally, which leads to difficulties with binary reproducibility.

For this reason, we decided to leverage the work done by the Gitian Project from the Bitcoin community. Gitian is a wrapper around Ubuntu's virtualization tools that allows you to specify an Ubuntu or Debian version, architecture, a set of additional packages, a set of input files, and a bash build scriptlet in an YAML document called a "Gitian Descriptor". This document is used to install a qemu-kvm image, and execute your build scriptlet inside it.

We have created a set of wrapper scripts around Gitian to automate dependency download and authentication, as well as transfer intermediate build outputs between the stages of the build process. Because Gitian creates a Linux build environment, we must use cross-compilation to create packages for Windows and macOS. For Windows, we use mingw-w64 as our cross compiler. For macOS, we use cctools and clang and a binary redistribution of the Mac OS 10.7 SDK.

The use of the Gitian system eliminates build non-determinism by normalizing the build environment's hostname, username, build path, uname output, toolchain versions, and time. On top of what Gitian provides, we also had to address the following additional sources of non-determinism:

1. **Filesystem and archive reordering**

   The most prevalent source of non-determinism in the components of Tor Browser by far was various ways that archives (such as zip, tar, jar/ja, DMG, and Firefox manifest lists) could be reordered. Many file archivers walk the file system in inode structure order by default, which will result in ordering differences between two different archive invocations, especially on machines of different disk and hardware configurations.

   The fix for this is to perform an additional sorting step on the input list for archives, but care must be taken to instruct libc and other sorting routines to use a fixed locale to determine lexicographic ordering, or machines with different locale settings will produce different sort results. We chose the 'C' locale for this purpose. We created wrapper scripts for tar, zip, and DMG to aid in reproducible archive creation.

2. **Uninitialized memory in toolchain/archivers**

   We ran into difficulties with both binutils and the DMG archive script using uninitialized memory in certain data structures that ended up written to disk. Our binutils fixes were merged upstream, but the DMG archive fix remains an independent patch.

3. **Fine-grained timestamps and timezone leaks**

   The standard way of controlling timestamps in Gitian is to use libfaketime, which hooks time-related library calls to provide a fixed timestamp. However, due to our use of wine to run py2exe for python-based pluggable transports, pyc timestamps had to be addressed with an additional helper script. The timezone leaks were addressed by setting the **TZ** environment variable to UTC in our descriptors.

4. **Deliberately generated entropy**

   In two circumstances, deliberately generated entropy was introduced in various components of the build process. First, the BuildID Debuginfo identifier (which associates detached debug files with their corresponding stripped executables) was introducing entropy from some unknown source. We removed this header using objcopy invocations in our build scriptlets, and opted to use GNU DebugLink instead of BuildID for this association.

   Second, on Linux, Firefox builds detached signatures of its cryptographic libraries using a temporary key for FIPS-140 certification. A rather insane subsection of the FIPS-140 certification standard requires that you distribute signatures for all of your cryptographic libraries. The Firefox build process meets this requirement by generating a temporary key, using it to sign the libraries, and discarding the private portion of that key. Because there are many other ways to intercept the crypto outside of modifying the actual DLL images, we opted to simply remove these signature files from distribution. There simply is no way to verify code integrity on a running system without both OS and co-processor assistance. Download package signatures make sense of course, but we handle those another way (as mentioned above).

5. **LXC-specific leaks**

   Gitian provides an option to use LXC containers instead of full qemu-kvm virtualization. Unfortunately, these containers can allow additional details about the host OS to leak. In particular, umask settings as well as the hostname and Linux kernel version can leak from the host OS into the LXC container. We addressed umask by setting it explicitly in our Gitian descriptor scriptlet, and addressed the hostname and kernel version leaks by directly patching the aspects of the Firefox build process that included this information into the build. It also turns out that some libraries (in particular: libgmp) attempt to detect the current CPU to determine which optimizations to compile in. This CPU type is uniform on our KVM instances, but differs under LXC.

5.2. Package Signatures and Verification

The build process generates a single sha256sums-unsigned-build.txt file that contains a sorted list of the SHA-256 hashes of every package produced for that build version. Each official builder uploads this file and a GPG signature of it to a directory on a Tor Project's web server. The build scripts have an optional matching step that downloads these signatures, verifies them, and ensures that the local builds match this file.

When builds are published officially, the single sha256sums-unsigned-build.txt file is accompanied by a detached GPG signature from each official builder that produced a matching build. The packages are additionally signed with detached GPG signatures from an official signing key.

The fact that the entire set of packages for a given version can be authenticated by a single hash of the sha256sums-unsigned-build.txt file will also allow us to create a number of auxiliary authentication mechanisms for our packages, beyond just trusting a single offline build machine and a single cryptographic key's integrity. Interesting examples include providing multiple independent cryptographic signatures for packages, listing the package hashes in the Tor consensus, and encoding the package hashes in the Bitcoin blockchain.

The Windows releases are also signed by a hardware token provided by Digicert. In order to verify package integrity, the signature must be stripped off using the osslsigncode tool, as described on the Signature Verification page.

5.3. Anonymous Verification

Due to the fact that bit-identical packages can be produced by anyone, the security of this build system

extends beyond the security of the official build machines. In fact, it is still possible for build integrity to be achieved even if all official build machines are compromised.

By default, all tor-specific dependencies and inputs to the build process are downloaded over Tor, which allows build verifiers to remain anonymous and hidden. Because of this, any individual can use our anonymity network to privately download our source code, verify it against public, signed, audited, and mirrored git repositories, and reproduce our builds exactly, without being subject to targeted attacks. If they notice any differences, they can alert the public builders/signers, hopefully using a pseudonym or our anonymous bug tracker account, to avoid revealing the fact that they are a build verifier.

5.4. Update Safety

We make use of the Firefox updater in order to provide automatic updates to users. We make use of certificate pinning to ensure that update checks cannot be tampered with by setting **security.cert_pinning.enforcement_level** to **2**, and we sign the individual MAR update files with keys that get rotated every year.

The Firefox updater also has code to ensure that it can reliably access the update server to prevent availability attacks, and complains to the user after 48 hours go by without a successful response from the server. Additionally, we use Tor's SOCKS username and password isolation to ensure that every new request to the updater (provided the former got issued more than 10 minutes ago) traverses a separate circuit, to avoid holdback attacks by exit nodes.

A. Towards Transparency in Navigation Tracking

The privacy properties of Tor Browser are based upon the assumption that link-click navigation indicates user consent to tracking between the linking site and the destination site. While this definition is sufficient to allow us to eliminate cross-site third party tracking with only minimal site breakage, it is our long-term goal to further reduce cross-origin click navigation tracking to mechanisms that are detectable by attentive users, so they can alert the general public if cross-origin click navigation tracking is happening where it should not be.

In an ideal world, the mechanisms of tracking that can be employed during a link click would be limited to the contents of URL parameters and other properties that are fully visible to the user before they click. However, the entrenched nature of certain archaic web features make it impossible for us to achieve this transparency goal by ourselves without substantial site breakage. So, instead we maintain a Deprecation Wishlist of archaic web technologies that are currently being (ab)used to facilitate federated login and other legitimate click-driven cross-domain activity but that can one day be replaced with more privacy friendly, auditable alternatives.

Because the total elimination of side channels during cross-origin navigation will undoubtedly break federated login as well as destroy ad revenue, we also describe auditable alternatives and promising web draft standards that would preserve this functionality while still providing transparency when tracking is occurring.

A.1. Deprecation Wishlist

1. **The Referer Header**

   When leaving a .onion domain we set the Referer header to an empty string by providing a preference, **network.http.referer.hideOnionSource**, and setting it to **true**. That avoids leaking information which might be especially problematic in the case of transitioning from a .onion domain to one reached over clearnet. Apart from that we haven't disabled or restricted the Referer ourselves because of the non-trivial number of sites that rely on the Referer header to "authenticate"

image requests and deep-link navigation on their sites. Furthermore, there seems to be no real privacy benefit to taking this action by itself in a vacuum, because many sites have begun encoding Referer URL information into GET parameters when they need it to cross HTTP to HTTPS scheme transitions. Google's +1 buttons are the best example of this activity.

Because of the availability of these other explicit vectors, we believe the main risk of the Referer header is through inadvertent and/or covert data leakage. In fact, a great deal of personal data is inadvertently leaked to third parties through the source URL parameters.

We believe the Referer header should be made explicit, and believe that Referrer Policy, which is available since Firefox 52, provides a decent step in this direction. If a site wishes to transmit its URL to third party content elements during load or during link-click, it should have to specify this as a property of the associated HTML tag or in an HTTP response header. With an explicit property or response header, it would then be possible for the user agent to inform the user if they are about to click on a link that will transmit Referer information (perhaps through something as subtle as a different color in the lower toolbar for the destination URL). This same UI notification can also be used for links with the "ping" attribute.

2. **window.name**

   window.name is a DOM property that for some reason is allowed to retain a persistent value for the lifespan of a browser tab. It is possible to utilize this property for identifier storage during click navigation. This is sometimes used for additional CSRF protection and federated login.

   It's our opinion that the contents of window.name should not be preserved for cross-origin navigation, but doing so may break federated login for some sites.

3. **JavaScript link rewriting**

   In general, it should not be possible for onclick handlers to alter the navigation destination of 'a' tags, silently transform them into POST requests, or otherwise create situations where a user believes they are clicking on a link leading to one URL that ends up on another. This functionality is deceptive and is frequently a vector for malware and phishing attacks. Unfortunately, many legitimate sites also employ such transparent link rewriting, and blanket disabling this functionality ourselves will simply cause Tor Browser to fail to navigate properly on these sites.

   Automated cross-origin redirects are one form of this behavior that is possible for us to address ourselves, as they are comparatively rare and can be handled with site permissions.

A.2. Promising Standards

1. Web-Send Introducer

   Web-Send is a browser-based link sharing and federated login widget that is designed to operate without relying on third-party tracking or abusing other cross-origin link-click side channels. It has a compelling list of privacy and security features, especially if used as a "Like button" replacement.

2. Mozilla Persona

   Mozilla's Persona is designed to provide decentralized, cryptographically authenticated federated login in a way that does not expose the user to third party tracking or require browser redirects or side channels. While it does not directly provide the link sharing capabilities that Web-Send does, it is a better solution to the privacy issues associated with federated login than Web-Send is.