Securing the Tor Network

Black Hat USA 2007 Supplementary Handout

Mike Perry

This handout is not a standalone document. It is intended to be used in conjunction with the presentation slides, and together the two documents should be sufficient to serve as a comprehensive artifact of the presentation. The speaker will refer to some figures and pages by number during the presentation.

Tor Routing

Tor traffic is routed through 3 nodes by default: Guard, relay, and exit. In normal operation, a given user only has 2 guard nodes that they use exclusively. The relay node is chosen arbitrarily, and the exit node is chosen according to its exit policy, which specifies which IPs and ports it will connect to. Paths are changed roughly every 10 minutes.

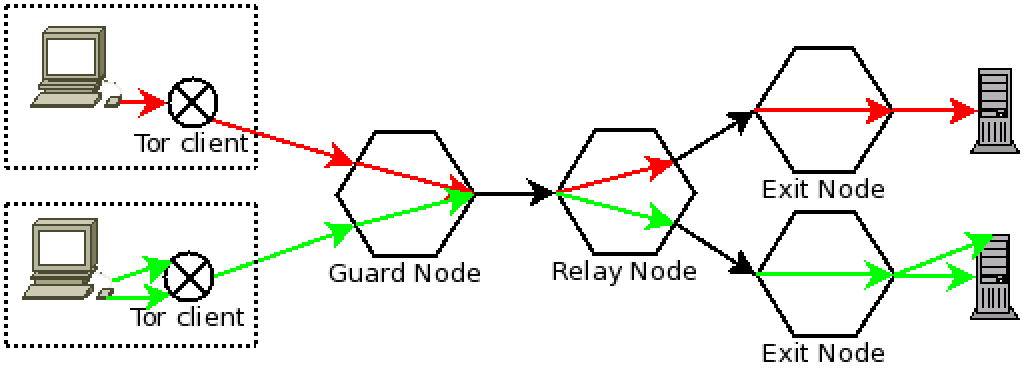

Illustration 1: Routing Diagram

This diagram illustrates two important properties of Tor. First, the two clients illustrate the multiplexing of multiple "circuits" over a single node-to-node TLS connection. Second, the bottom client illustrates the stream multiplexing capability: multiple TCP connections can be carried over a single Tor circuit. Each node knows only the source and destination pairing for a circuit. It does not know the whole path.

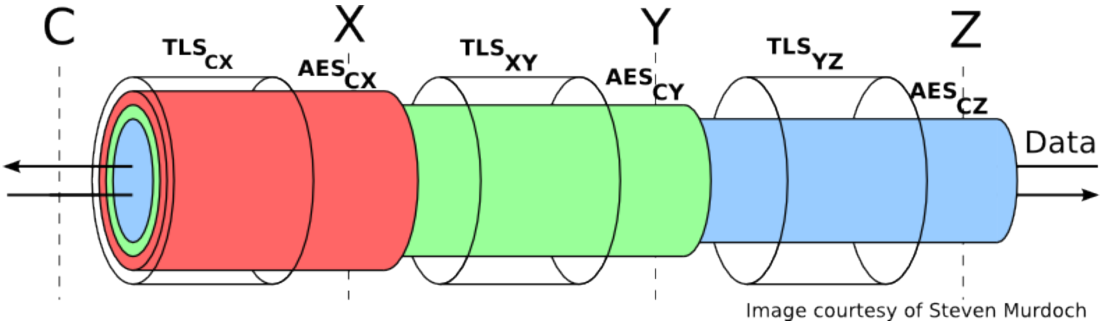

Image courtesy of Steven Murdoch

Illustration 2: Tor Encryption

This diagram illustrates the Tor circuit level encryption mechanism. Node-level communications are encrypted with TLS connection. Inside this, the Tor Directory lists public keys that the client uses to establish the three secret AES session keys between itself and each successive hop in the circuit.

Points of Attack

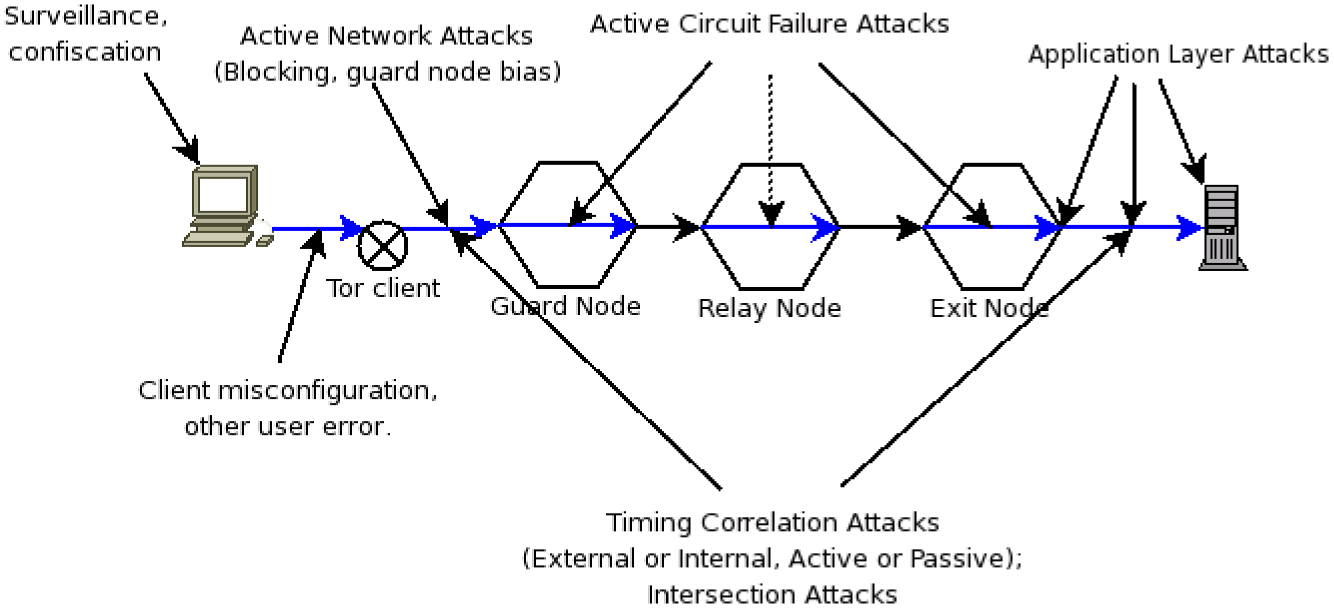
Surveillance, confiscation
Active Network Attacks (Blocking, guard node bias)
Active Circuit Failure Attacks
Application Layer Attacks
Tor client
Guard Node
Relay Node
Exit Node
Client misconfiguration, other user error.
Timing Correlation Attacks (External or Internal, Active or Passive); Intersection Attacks

Illustration 3: Attack Points

There are obviously a lot of attack points here, some more serious than others, and some can be performed in a few different ways. The following are worth discussing in extended detail:

1. **Application Layer Attacks.**

   These attacks revolve around feeding the application some type of data that either causes it to bypass proxy settings, reveal crucial user information, or otherwise outright exploit it. Note that there are actually three positions that this type of attack can be performed (exit, transit, and destination), and all three have been observed in the wild.

   A particularly surprising example of this type of attack is an exit node spoofing a content element from mail.google.com and fetching your gmail account cookie and downloading all your mail, even though you were not visiting any Google-related sites at the time. Google's insistence (and Yahoo's and Hotmail's too for that matter) on allowing non-https access makes this attack an easy target from not only Tor, but your local coffee shop also (or the Defcon network!). Yet another great reason to clear your cookies regularly.

2. **Intersection Attacks.**

   In my opinion, intersection attacks are currently the second most dangerous attack against Tor, behind application layer attacks. Intersection attacks essentially rely on correlating several distinct properties of Tor users to match pseudonymous Tor activity to that user, and function best if there are few Tor users using the network. This obviously covers a wide range of concrete attacks, but the most surprising instance I am aware of is the case of a university student who was conducting an online scam somehow related to his or her university.[1] Since few people are currently willing to put up with Tor's slow (and irregular) connection speeds, it was a trivial matter for local network administrators to interrogate the only two regular Tor users on campus. This is not to say that running scams via Tor is something we would like to encourage or protect, but this underscores the fact that Tor network popularity is actually a key component of Tor network security for others who need more legitimate anonymity.[2]

3. **Active Circuit Failure Attacks**.

   Guard nodes can actively fail circuits if they do not extend to their colluding peers. If they are able to find a valid side channel either within the Tor protocol or just using timing information, they may be able to perform this attack at the guard and exit position only. In the degenerate case, however, they can continue to fail each successive circuit extend until a colluding peer is chosen for that hop. This attack underscores the fact that reliability is yet another key factor of security in anonymity networks.

   Note that various properties of Tor make this attack slightly less straightforward in practice, however. The user typically has at least 2, but usually more guards, and the Tor client will switch to a randomly selected guard each time the circuit fails. However, lying about bandwidth to a degree proportional to the percentage of circuits failed before one succeeds can allow nodes to maximize their potential for damage, since they will be carrying less traffic due to failing circuits causing the clients to sometimes move elsewhere.

4. **Timing Correlation Attacks.**

   Timing correlation attacks attempt to use connection time, duration, and flow characteristics to correlate the client's connection to a guard node to an external exiting connection. An active adversary can also introduce their own timing patterns into this traffic. Academic studies have shown that this attack can be extremely effective in simulation[3]. However, it still remains to be seen how particular details of the Tor network affect the ability to carry out this attack. In particular, unanswered and interesting research questions include:

   - To what degree does Tor's stream multiplexing frustrate this?
   - At what point does the number of users exceed the bits of information that can be reliably obtained from timing information in a passive attack?
   - How much more difficult is it to perform this attack externally than internally?
   - Running Tor as both a client and a node should greatly improve your resistance to both internal and external versions of this attack. Is this quantifiable, though?

Centralized Bandwidth and Load Scanning

Centralized bandwidth scanning divides the network into groups of nodes based on their reported speed, and fetches large files via paths that consist entirely of nodes with reported speeds close to theirs. The locally observed bandwidth is then recorded for each node in the path, and divided into its reported bandwidth to obtain an estimated number of concurrent full-capacity streams passing through that node. **S=B/L**.

Since Tor attempts to load balance proportional to node's claimed bandwidth, nodes that lie about their capacity will end up carrying many more streams than they usually would, and will be greatly overloaded, and will exhibit very low observed values for **L**, and thus high values for **S**.

Like all centralized scanning approaches, centralized bandwidth scanning is not without its limitations. Scanner IPs will exhibit very obvious repetitive traffic patterns to guard nodes, and exit nodes can come to recognize test URLs. Once the scanner activity is recognized, scanner traffic can be prioritized to give the illusion of higher capacity.

Workarounds for these issues include infrequent, short duration scanning, as well as only counting statistics for the middle node, who is unable to easily recognize scanned content.

Interesting Items Found During Network Scanning

1. Chinese ISP doing MITM on SSL
2. Regional ISPs inserting popup blocker Javascript into HTTP streams
3. DNS spoofing
4. Upstream caches caching stale content
5. Verified reports of MITM on SSH, SSL
6. Disproved incorrect accusations of bandwidth lying
7. Instances of overloaded nodes due to strange exit policies and load balancing bugs

Decentralized Network Scanning

Decentralized failure scanning is still in proposal form as part of the Two Hop Paths Tor Proposal #115.[4] Google Summer of Code participant Johannes Renner is also conducting research on the effectiveness of node-based scanning as a part of his Master's Thesis on improving Tor's path selection.

Tor's Web Attack Profile

1. Ways to bypass Tor/Proxy Settings
   - (a) Java 1.5 Socket API allows code to override Java proxy settings
     - i. Does this sound like a good feature for secured environments in general??
   - (b) Plugins handle their own network code and settings
   - (c) Delayed execution until Tor is disabled
     - i. meta-refresh tags
     - ii. Timers and Events
     - iii. Busy waiting
2. Correlation of Tor vs Non-Tor
   - (a) Cookies
     - i. Exits can spoof elements from fruitful SSL-sloppy domains such as mail.google.com
   - (b) Cache Data
     - i. Cached objects can contain deliberately constructed unique identifiers
3. History Disclosure
   - (a) Enumeration of link tag style attributes in Javascript
     - i. http://ha.ckers.org/weird/CSS-history-hack.html
   - (b) Using CSS to fetch specific background images for visited styles. No Javascript needed.
     - i. http://ha.ckers.org/weird/CSS-history.cgi
4. Location artifacts
   - (a) Timezone
   - (b) Locale, charset
5. Misc anonymity set reduction
   - (a) OS version/user agent
6. History records/disk state
   - (a) Some users are at risk simply for using Tor. Their computers may be confiscated/examined.

TorButton Options

- **Disable plugins on Tor Usage**
  This option is key to Tor security. Plugins perform their own networking independent of the browser, and many plugins only partially obey even their own settings.
- **Isolate Dynamic Content to Tor State**
  Another crucial option, this setting causes the plugin to disable Javascript on tabs that are loaded during a Tor state different than the current one, to prevent delayed fetches of injected URLs that contain unique identifiers, and to kill meta-refresh tags. It also enables an nsIContentPolicy that prevents all fetches from tabs with an opposite Tor state to block non-Javascript dynamic content such as CSS popups.
- **Hook Dangerous Javascript**
  This function enables the Javascript hooking code. Javascript is injected into the DOM (and then removed immediately after executing) to hook the Date object to mask timezone, and to hook the navigator object to mask OS and user agent properties not handled by the standard Firefox user agent override settings.
- **Disable Updates During Tor**
  Many extension authors do not update their extensions from SSL-enabled websites. It is possible for malicious Tor nodes to hijack these extensions and replace them with malicious ones, or add malicious code to existing extensions.
- **Disable Search Suggestions during Tor**
  This optional setting governs if you get Google search suggestions during Tor usage. Since no cookie is transmitted during search suggestions, this is a relatively benign behavior.
- **Block History Reads during Tor/Non-Tor**
  Based on code contributed by Collin Jackson[5], this is actually two settings, but are combined here for brevity since the effect is the same. When enabled and the corresponding Tor state is active, this setting prevents the rendering engine from knowing if certain links were visited. This mechanism defeats all document-based history disclosure attacks, including CSS-only attacks.
- **Block History Writes during Tor/Non-Tor**
  This setting prevents the rendering engine from recording visited URLs, and also disables download manager history, form field history, and disables remembering login information. Note that if you allow writing of Tor history, it is recommended that you disable non-Tor history reads, since malicious websites you visit without Tor can query your history for .onion sites and other history recorded during Tor usage (such as Google queries).

- **Disable Session Saving**
  This option disables the session store, which stores your session in the event of browser upgrades and crashes. Since the session store can be written at random times and a browser crash or upgrade can cause you to refetch many Tor urls outside of Tor, currently this is an all-or-nothing setting for both Tor and Non-Tor.
- **Clear History During Tor Toggle**
  This is an alternate setting to use instead of (or in addition to) blocking history reads or writes.
- **Block Tor disk cache and clear all cache on Tor Toggle**
  Since the browser cache can be leveraged to store unique identifiers, cache must not persist across Tor sessions. This option keeps the memory cache active during Tor usage for performance, but blocks disk access for caching.
- **Block disk and memory cache during Tor**
  This setting entirely blocks the cache during Tor, but preserves it for Non-Tor usage.
- **Clear Cookies on Tor Toggle**
  Fully clears all cookies on Tor toggle.
- **Store Non-Tor cookies in a protected jar**
  This option stores your persistent Non-Tor cookies in a special cookie jar file, in case you wish to preserve some cookies. Contributed by Collin Jackson.[5]
- **Manage My Own Cookies**
  This setting allows you to manage your own cookies with an alternate extension, such as CookieCuller[6]. Note that this is particularly dangerous, since malicious exit nodes can spoof document elements that appear to be from sites you have preserved cookies for.
- **Clear cookies on Tor/Non-Tor shutdown**
  This setting uses the Firefox Private Data settings to clear cookies on Tor and/or Non-Tor browser shutdown.
- **Set user agent during Tor usage**
  User agent masking is done with the idea of making all Tor users appear uniform. A recent Firefox 2.0.0.4 Windows build was chosen to mimic for this string and supporting navigator.* properties, and this version will remain the same for all TorButton versions until such time as specific incompatibility issues are demonstrated. Uniformity of this value is obviously very important to anonymity. Note that for this option to have full effectiveness, the user must also allow **Hook Dangerous Javascript** ensure that the navigator.* properties are reset correctly. The browser does not set some of them via the exposed user agent override preferences.

Javascript Hooking

Firefox extension development is a maze of (extremely powerful) black voodoo magic and cryptic interfaces that allow you to customize just about every aspect of the browser and browser components via Javascript and XML. However, the one thing they lack is the ability to configure your timezone. Hence, regular client-side Javascript can query the Date object to determine your approximate geographic location.

Luckily, it turns out that Javascript is actually a pretty flexible language. Through a combination of object oriented functionality and lexical scoping, it is possible to create wrappers to existing objects while still concealing the original implementation, and to do all this at the same privilege level as client-side Javascript without risk of subversion.

The approach is to create a nsIWebProgressListener to listen for location change events to alert you as soon as a new document's DOM is first created. At this point, you create a custom script tag, insert it into the DOM (this causes the script to immediately run), then remove it.

The key to the whole operation is to make sure that all of your references to the original object exist only as local variables that just happen to be in the same scope as the functions you are hooking (this works because of lexical scoping). In my case, this is what prevents the adversary from just calling the internal wrapped Date implementation.

To make it a bit more difficult to fingerprint TorButton users, all other variables are direct properties of '**window**', and can be deleted from that object using the hash syntax and the **delete** operator. A skeleton sketch of the injected code is below. For the support code that does the injection, see the TorButton .xpi source code (in particular, **torbutton_weblistener** in **torbutton.js**).

```
window.__HookObjects = function() {
 if(__tb_set_uagent) {
   var tmp_oscpu = window.__tb_oscpu;
   navigator.__defineGetter__("oscpu", function() { return tmp_oscpu;}); /* ... */
 }
 var tmp = Date;
 Date = function() {
  var d; /* DO NOT make 'd' a member! EvilCode will use it! */
  var a = arguments;
  if(arguments.length == 0) d=new tmp();   /* ... */

  Date.prototype.getYear=function() {return d.getUTCYear();}
  Date.prototype.getMonth=function(){return d.getUTCMonth();}
  Date.prototype.getDate=function() {return d.getUTCDate();}
  Date.prototype.getDay=function() {return d.getUTCDay();}    /* ... */

  /* Hack to solve the problem of multiple date objects all sharing the same lexically scoped d every
     time a new one is created. This hack creates a fresh new prototype reference for the next object to
     use with a different d binding. It doesn't break stuff because at the start of this function, the
     interpreter grabbed a reference to Date.prototype, and this new assignment does not affect the
     interpreter's reference. During this function we modified Date.prototype to create the new
     methods with the lexically scoped d reference.  */
  Date.prototype = new Object.prototype.toSource();
  return d.toUTCString();
 }
 Date.now=function(){return tmp.now();}
 Date.UTC=function(){return tmp.apply(tmp, arguments); }
}

if (window.__HookObjects) {
  window.__HookObjects();
  delete window['__HookObjects'];
  delete window['__tb_set_uagent'];
  delete window['__tb_oscpu'];
}
```

References

1. p2pnet news. Professor in Tor Trouble. Feb 9, 2007. http://p2pnet.net/story/11279
2. Roger Dingledine and Nick Mathewson. Anonymity Loves Company: Usability and the Network Effect. In the *Proceedings of the Fifth Workshop on the Economics of Information Security (WEIS 2006)*, Cambridge, UK, June 2006.
3. Brian N. Levine, Michael K. Reiter, Chenxi Wang, and Matthew K. Wright. Timing attacks in low-latency mix-based systems. In Ari Juels, editor, Proceedings of Financial Cryptography (FC '04). Springer-Verlag, LNCS 3110, February 2004.
4. Mike Perry. Two Hop Paths Tor Proposal #115. http://tor.eff.org/svn/trunk/doc/spec/proposals/115-two-hop-paths.txt
5. Collin Jackson. Contributed Code to FoxTor. http://cups.cs.cmu.edu/foxtor/
6. Dan Yamaoka. CookieCuller Firefox Extension. https://addons.mozilla.org/en-US/firefox/addon/82