Design Requirements for a Tor Censorship Analysis Tool

Philipp Winter
phw@torproject.org

Tor Tech Report 2013-02-001
February 6, 2013

1 Introduction

The Tor network is documented to be blocked in several countries [1]. Analyzing and circumventing these blocks typically requires detailed *packet traces* or access to *machines inside censoring countries*. Both, however, are not always easy to acquire:

1. Network traces are problematic for two reasons. First, they are difficult to obtain since they require the cooperation of users within censoring countries. Second, they are hard to anonymize and must not fall into wrong hands. Derived information, such as flow diagrams[1], are typically safe to publish but frequently lack important information.

2. The alternative to network traces is to gain access to machines inside the censoring regime. This approach turns out to be difficult as well; mostly due to the lack of volunteers who could provide machines or the lack of VPS providers and open SOCKS proxies.

These problems show that there is a strong need for a lightweight tool which can assist in analyzing censorship events. This tool should be run by censored users and perform several tests to gain a rough understanding of how and if Tor could be blocked in the respective network. The results of these tests should make it back to the Tor project and are used to improve circumvention technology such as obfsproxy [7] and to document censorship [1].

This technical report discusses the design requirements for such a censorship analysis tool. We list the desired features, discuss how they can be implemented and we give a rough overview of the software design. After all, this technical report should serve as basis for the development and deployment of the censorship analysis tool.

[1] See, for example, the blog post discussing a Tor block in Iran [5].

2 Feature Requirements

The following list enumerates the features which are desirable in a censorship analyzer. Naturally, certain features are harder to implement than others, so the list is organized in ascending order based on the difficulty of the respective feature.

1. **Capture debugging process**: The tool should be able to create a pcap file of the network debugging process to allow further inspection. While very handy, pcaps are sensitive data and would require Administrator/root permissions.

2. **User-friendly output**: While the censorship analyzer is meant to assist Tor developers in debugging censorship incidents, user-friendly log messages are easy to add and can give users an idea of why their Tor fails to connect. Based on the gathered data, the analyzer could give the user suggestions on what to try next. This might even slightly reduce the help desk's load.

3. **Obfuscate tests**: Censors might be interested in identifying the Tor censorship analyzer and try to actively falsify the tests. Therefore, the analyzer should make an effort to stay under the radar. In particular, the analyzer should implement:

   (a) Random sleep periods between (and perhaps during) tests.

   (b) Randomize the order of executed tests.

   (c) Use random IP addresses for tests such as the relay reachability discussed below.

   Note that it is not possible to completely hide the analyzer's existence. Rather, this feature should be understood as hiding all too obvious network activity.

4. **Leave no traces behind**: The analyzer should not leave any traces on the user's hard disk. Ideally, the analyzer should generate a single report file which is placed in the same directory as the analyzer itself. That would make it possible for users to conveniently delete all traces. Temporary analysis files should be deleted after the report was generated.

5. **DirAuth reachability**: Try to connect to the directory authorities and download the consensus. If this fails, check if:

   (a) The authorities respond to ICMP echo requests to see if the IP addresses are blocked.

   (b) Run traceroutes to the directory authorities as well as to other—hopefully unblocked—hosts in the same subnet as the directory authorities. This could yield the location of censoring boxes and serve as proof that the IP addresses are, in fact, blocked.

6. **Web site reachability**: Try to connect to `https://www.torproject.org` and fetch the index page. If the web site fails to load, check if:

   (a) One of the official Tor mirrors [8] works[2].

[2] In particular, mirrors without the strings "tor" or "torproject" in the domain should be given a try.

(b) The domain `www.torproject.org` resolves to the correct, non-poisoned IP addresses.

(c) A simple TCP connection to `www.torproject.org` succeeds. If so, in the subsequent step, a TLS session could be established. That way, it is possible to find out whether DPI boxes are inspecting the SNI in the TLS client hello.

(d) The hosts behind `www.torproject.org` respond to ICMP echo requests.

7. **Bridge distribution**: Try to connect to `https://bridges.torproject.org` and fetch the index page. If the web site loads, it is safe to assume that obfsproxy bridges can be fetched as well. If the web site fails to load, check if:

   (a) The domain `bridges.torproject.org` resolves to the correct, non-poisoned IP address.

   (b) A simple TCP connection to `bridges.torproject.org` succeeds. If so, in the subsequent step, a TLS session could be established. That way, it is possible to find out whether DPI boxes are inspecting the SNI in the TLS client hello.

   (c) The host behind `bridges.torproject.org` responds to ICMP echo requests.

8. **Relay reachability**: Try to connect to a number of Tor relays listed in the consensus. Typically, clients connect to entry guards. However, it would also be interesting to learn whether connections to pure middle or exit relays succeed[3]. If this fails, check if:

   (a) A Tor-specific TLS client hello can be sent to `mail.google.com:443` – assuming that this host is reachable. If the connection is closed in a non-clean fashion, this could be an indicator that fields in the TLS client hello are subject to filtering.

   (b) (Private) bridges and their censorship-resistant variants (brdgrd, obfs2, obfs3, flashproxies) are reachable.

9. **Gather debug information**: Censorship is typically not homogeneous across a country and varies depending on provinces, autonomous systems or ISPs [9]. As a result, we are interested in information which can help shed light on the respective censorship infrastructure. Also, this would help ruling out interferences and prevent jumping to wrong conclusions. Of interest would be:

   (a) What ISP does the user have?

   (b) What is the autonomous system number?

   (c) Is the user behind a captive portal?

   (d) Is all traffic forced to go through an HTTP proxy?

10. **Debug the TLS handshake**: Tor is frequently blocked based on identifying information in its TLS handshake [4, 1]. Debugging the exact fingerprint used by DPI boxes to identify Tor can be of great value. This is, however, a very hard problem which requires a client server architecture to infer fingerprints. The tool daphne was started with this goal in mind [2].

[3] This could be an indicator that a censor is blindly blacklisting all IP addresses found in the consensus.

3 Software Architecture

The following list enumerates software-specific aspects of the censorship analyzer.

1. **Ease of use**: It is crucial that the analyzer is as easy to use as possible. Ideally, it should be a self-contained click-and-go executable, just like the Tor Browser Bundle. After all, the target group consists mostly of ordinary users rather than developers.

   Ease of use also involves the analyzer's *bundle size*. Ideally, the analyzer would only be a few megabytes in size which would also make is suitable for distribution via GetTor [6].

2. **Configurable during build**: It should be possible to pass configuration parameters to the analyzer during the build process. That is necessary because certain information such as IP addresses of relays to test or of `www.torproject.org` change over time or might be white-listed by censors. As a result, it is not a good idea to hard-code these things in the source code.

3. **Least privilege**: Ideally, the analyzer should not require Administrator/root access.

4. **Existing framework**: There is no need to reinvent the wheel. The analyzer should be implemented as tests for the open observatory of network interference (OONI) [3]. OONI provides a Python API which can be used to develop all of the above mentioned features.

5. **Data delivery**: Eventually, the Tor project has to learn about test results. There are two possible ways:

   (a) The analyzer could automatically transmit gathered data to the Tor project. Automated uploads must require the user's informed consent and the user must be given the choice to review the report prior to submission.

   (b) The analyzer can create a report and then ask the user to send the report to censorship-analyzer@torproject.org (which does not exist yet). This could still be a fallback plan if an automated upload fails.

   Further, the report should contain a *message digest* which is built over the report. This is particularly important when the report is being sent over e-mail since it allows us to detect if the report is incomplete or the user accidentally changed parts of it.

6. **Testability**: All the features discussed in Section 2 should be testable in an automated way. Otherwise, we might end up shipping code which does not work in real environments or we might not notice if improvements break existing code.

Acknowledgments

Arturo Filastò, George Kadianakis, Karsten Loesing and Runa A. Sandvik provided valuable feedback for this technical report.

References

[1] Censorship Wiki. URL: https://censorshipwiki.torproject.org.

[2] daphne. URL: https://trac.torproject.org/projects/tor/wiki/doc/OONI/Tests/daphne.

[3] Arturo Filastò and Jacob Appelbaum. OONI: Open Observatory of Network Interference. In *Free and Open Communications on the Internet*, Bellevue, WA, USA, 2012. USENIX Association. URL: https://www.usenix.org/system/files/conference/foci12/foci12-final12.pdf.

[4] Nick Mathewson. TLSHistory. URL: https://trac.torproject.org/projects/tor/wiki/org/projects/Tor/TLSHistory.

[5] phobos. Update on Internet censorship in Iran. URL: https://blog.torproject.org/blog/update-internet-censorship-iran.

[6] The Tor Project. GetTor e-mail autoresponder. URL: https://www.torproject.org/projects/gettor.html.en.

[7] The Tor Project. obfsproxy. URL: https://www.torproject.org/projects/obfsproxy.html.en.

[8] The Tor Project. Tor: Mirrors. URL: https://www.torproject.org/getinvolved/mirrors.html.en.

[9] Joss Wright, Tulio de Souza, and Ian Brown. Fine-Grained Censorship Mapping: Information Sources, Legality and Ethics. In *Free and Open Communications on the Internet*, San Francisco, CA, USA, 2011. USENIX Association. URL: http://static.usenix.org/event/foci11/tech/final_files/Wright.pdf.