Censorship avoidance for Tor

W.J.B. Beukema

University of Twente, PO Box 217, 7500 AE Enschede, The Netherlands
`w.j.b.beukema@student.utwente.nl` (0E9B5E91)

1 Introduction

The Union Router, better known as Tor [3], is a circuit-based low-latency communication service that provides anonymity. It directs Internet traffic through a free, worldwide network of volunteers to conceal a user's location and communication partners from outside attackers.

Tor is often used in regimes where values such as free speech and freedom of press are suppressed or absent. It has been discovered that nations such as Iran, China and Vietnam actively monitor and control domestic Internet traffic [9].

Internet traffic might be monitored on nation-scale, in an effort to block certain traffic or in order to locate the users participating in it. Through its powerful mechanism, Tor provides anonymity on the Internet that makes it at least substantially harder for external attackers[1] to link users to their communication partners.

1.1 Problem definition

Over the last few years, Tor has gained much popularity. This has also caught the attention of those who have interests in monitoring and blocking Internet traffic. By design, Tor does not encrypt its traffic end-to-end, nor does it hide its own protocol messages by means of e.g. steganography. As a result of the latter, censors might try to identify Tor traffic and sabotage it, forcing users to use 'normal' ways of communication, which might be subject to monitoring and blocking again. This development obviously threatens Tor's raison d'être in its very core.

The purpose of this paper is to analyse to what extent censorship of Tor traffic can be avoided. We do this by investigating the literature on this topic. This paper is organised as follows: we first briefly discuss the design of Tor and what is known about Tor specific censorship techniques. In the third section, we describe the suggested improvements/enhancements of Tor to achieve censorship avoidance. Finally, we sum up our findings and make recommendations on the future of censorship avoidance with regards to Tor.

[1] The term *external attacker* or *censor* is used as a general term for instances, such as governments, intelligence services, but also other groups, entities or persons who are interested in monitoring the user's communication and willing to frustrate Tor traffic for that cause.

2 Tor technology

The main feature of Tor is the traffic routing through a multiply encrypted circuit of Tor participants [3]. These relay[2] points are chosen randomly by the client. Each of the relay points can remove only a single layer of encryption and can hence only learn about the next hop. Therefore, this mechanism ensures that a single malicious relay cannot know both the source and the destination of Tor traffic he receives.

A specifically described non-goal of Tor is steganography: "Tor does not try to conceal who is connected to the network" [3, p. 4]. Therefore, it might be possible to identify traffic as Tor traffic. The actual contents of the traffic are, however, encrypted from the client to the last relay point. Encryption is applied on the application level, using TLS. Thus, external attackers observing Tor traffic simply (up to the last relay point) see TLS traffic.

2.1 Current methods of Tor censorship

As mentioned before, there are parties who have an interest in blocking Tor. Especially governments monitoring and blocking Internet traffic might have the right resources to try hinder people using Tor.

An obvious start would be to block the website of The Tor Project[3]. Although this might scare some users off, it is an ineffective approach as there are too many mirrors and alternative download locations to be able to completely ban the distribution of the software necessary to run Tor.

Most commonly, censors try to block access to the directory authorities that provide the lists that contain public relays. By blocking the directory authorities, clients will not be able to select clients to route their traffic through. Alternatively, all Tor relay IP addresses in the directory authorities might be blocked as well. This makes it even harder for the client to find relays to run the protocol. Note that these measures are at the end not really flexible, as it relies on blocking services based on advertised addresses. For instance, private relays (which are thus not advertised by the directory authorities) will not be discovered and hence not blocked, as we will see further on.

A somewhat more flexible approach for the external attacker is to detect Tor traffic and block the involved parties. This involves the use of Deep Packet Inspection (DPI) in order to classify Internet traffic flows by protocol. Through DPI, the censor will try to recognise and filter Tor traffic flows even when they connect to relays that are unknown to the censor. In the past, Tor used a quite distinctive set of supported ciphers that were advertised in the TLS handshake while setting up the Tor connection. China has been using this flaw to fingerprint Tor connections and subsequently temporarily deny the involved parties Internet access [9].

A rather extreme way to block Tor would be to completely block the underlying protocol that Tor uses to communicate between nodes within the network, TLS. Although this method has been used in practice (e.g. in Iran, 2012 [5]), there is no way to

[2] We distinguish *bridge relays* (those who give clients access to the Tor network), *non-exit relays* (those who transfer between Tor relays), and *exit relays* (those who output the 'real' request to the destination).

[3] See `https://www.torproject.org/`.

circumvent this kind of censorship as TLS under normal Tor[4], as is essential to provide anonymity in the Tor infrastructure.

3 Established solutions

3.1 Private bridge relays

Private bridge relays are the same as normal bridge relays, with the only difference that their existence is not publicly announced. For instance, a user could ask someone to set up a bridge relay for him. Alternatively, users can get the addresses of ‘semi-public’ bridge relays via alternative channels (instead of the public directory), such as email services and private websites. This approach is however not DPI resistant.

3.2 Pluggable transports

In order to enable developers to make modifications to the network flow, Tor supports a framework called *pluggable transports*. Using this framework, external developers can write additions for Tor that transform the outbound and inbound Tor traffic between the client and the bridge; all other parts in the Tor network flow (e.g. between (non-)exit relays) remains unchanged. This makes it possible to implement steganographic techniques, letting external attackers think they are looking at innocent traffic instead of Tor traffic. Both the client and the bridge relay need to install and configure the same extension to use it for communication.

There have been proposed some implementations, although only a few of them have been deployed to the wide public.

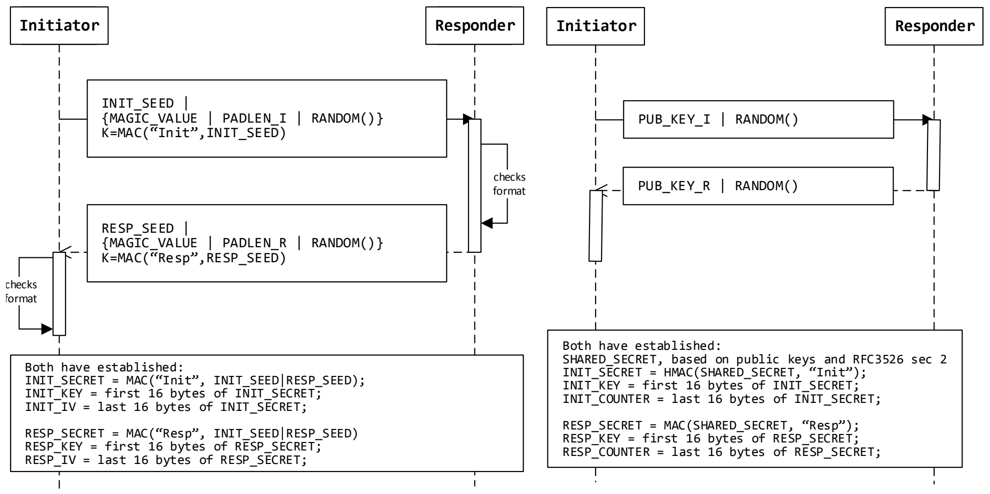
obfsproxy [7] is also included in the official Tor Browser Package. obfsproxy (*obfuscate proxy*) implements the *obfs2* and *obfs3* protocol, both protocol obfuscation layers for TCP protocols. They are *look-like-nothing* protocols: the data that can be observed looks like a random data stream.

Both protocols first establish a shared key, after which the superenciphered traffic takes place.

obfs2 [1] is a rather weak, unsafe protocol. Both parties first generate a random seed. After that, they concatenate a static value (`MAGIC_VALUE`), some random padding length and some random bytes. This is then encrypted using a MAC of the random seed as key. Now each party sends its random seed and the encrypted value to the other party. They derive using a MAC their encryption keys. AES-CTR-128 is used for the encryption. An external observer can easily decrypt the information by executing the same steps as the initiator and the responder do. Hence, although *obfs2* obfuscates the data, it is quite easy to detect and decipher the original Tor packet. See also figure 1a.

obfs3 [2] is substantially improved version of its predecessor. This protocol relies on UniformDH Diffie-Hellman, based on the parameters as set out in RFC 3526 section 2. The private key is an even, randomly chosen 1536-bit number. Both parties exchange their public keys ($2^{privkey} \mod p$) concatenated with some random value. Using standard Diffie-Hellman, they now have a shared key. Using an HMAC, their encryption

[4] We might however, as we will see, even in this case be able to use Tor using steganography.



```
INIT_SEED |
{MAGIC_VALUE | PADLEN_I | RANDOM()}
K=MAC("Init",INIT_SEED)
```

```
RESP_SEED |
{MAGIC_VALUE | PADLEN_R | RANDOM()}
K=MAC("Resp",RESP_SEED)
```

```
Both have established:
INIT_SECRET = MAC("Init", INIT_SEED|RESP_SEED);
INIT_KEY = first 16 bytes of INIT_SECRET;
INIT_IV = last 16 bytes of INIT_SECRET;

RESP_SECRET = MAC("Resp", INIT_SEED|RESP_SEED)
RESP_KEY = first 16 bytes of RESP_SECRET;
RESP_IV = last 16 bytes of RESP_SECRET;
```

```
Both have established:
SHARED_SECRET, based on public keys and RFC3526 sec 2
INIT_SECRET = HMAC(SHARED_SECRET, "Init");
INIT_KEY = first 16 bytes of INIT_SECRET;
INIT_COUNTER = last 16 bytes of INIT_SECRET;

RESP_SECRET = MAC(SHARED_SECRET, "Resp");
RESP_KEY = first 16 bytes of RESP_SECRET;
RESP_COUNTER = last 16 bytes of RESP_SECRET;
```

Fig. 1. An general overview of (a) *obfs2* key exchange and (b) *obfs3* key exchange.

keys can be determined. Like its predecessor, AES-CTR-128 is used for the encryption. See figure 1b. At this moment, *obfs3* is the default protocol used by *obfsproxy.*

obfs4 [10] is still in development, and hence not yet included in obfsproxy. It is however stronger than *obfs3*, as it also tries to provide authentication and data integrity, instead of obfuscation only. This is accomplished using the *ntor* protocol (for one-way authentication), Curve25519 keys and Dan Bernstein's Elligator 2. The latter is needed for public key obfuscation, i.e. to make the selected point on the curve look like arbitrary data. Additionally, active probing attacks are no longer possible in *obfs4*, as the bridge will only reply after the client proves knowledge of the secret shared out of band. Also, the packet lengths and timing are randomised by *obfs4*.

StegoTorus [8] is a fork of obfsproxy that splits Tor streams across multiple connections and embeds the traffic flows in traces that look like other protocols. The client hides its data in HTTP requests, using a special encoding, in the `URI` field and the `Cookies` field. The bridge responds by hiding the response data in files that look like PDF files, Flash (SWF) files or minified JavaScript files. The authors mention that video and audio formats might be used as well. Due to the complexity of these formats, the authors reckon it is safe to hide the encrypted Tor data in there.

For an outside observer, the traffic will look like HTTP traffic. Thus, even if TLS would be completely blocked, this approach will pass the filter. Note however that using this method is at the expense of performance: downstream roughly 11 times more data is sent while upstream nearly 16 times more data is sent compared to normal Tor traffic.

SkypeMorph [6] (also known as *Code Talker Tunnel*) is another fork that transforms Tor traffic into a stream that looks like Skype video network traffic.

The setup is generally as follows: the bridge logs in to Skype using the Skype API and starts listening for incoming calls. The bridge publishes its Skype ID in the same way it announces its IP address and port, e.g. via a directory service. If a client wishes to connect, he logs in to Skype as well. In a Skype chat message to the bridge, he announces his public key, his IP and a random UDP port. Note that Skype chat messages are encrypted (using AES) and authenticated (using RSA). The bridge replies its own public key, its own IP and a random UDP port. Both parties now compute their shared key (based on their public keys); the client sends back a hash of the shared key. If it matches the bridge's hash of his computed shared key, he sends '`OKAY`' back. The client now initiates a Skype call to the bridge, but drops the call after a random time. After the bridge noticed that the call has dropped, it starts listening at the port number picked earlier. The client and bridge can now communicate with each other using the IP addresses and port numbers exchanged in the set-up phase.

The transformation of Tor data is performed in such a way that for an outside observer, the generated traffic is indistinguishable from a real Skype call. From a network perspective, a dropped Skype call followed by fake data looks no different than an accepted Skype call followed by call data. The overhead *SkypeMorph* causes is about twice as much as usual Tor traffic.

Note that the design paper of *SkypeMorph* was written in 2012, just after Microsoft acquired Skype Inc. Some time after the acquisition, rigorous changes have been made to the infrastructure of Skype. For instance, the role of P2P within Skype has changed. It is therefore unclear whether the same method would still work today.

Meek [4] takes a slightly different approach, which is called *domain fronting*. The client creates a special HTTPS request and sends it to an intermediate web service with numerous domains behind it, such as the content delivery networks Google, Amazon and Microsoft‘s Azure platform provide. This intermediate service then forwards the client's request to the bridge relay. The core idea behind this method is that a censor is not able to identify the request as being a Tor request, and it is not likely to completely block access to major platforms such as Google and Amazon, as that would mean that all websites relying on those services will be unavailable.

To avoid the fingerprinting issue, *meek* actually uses an actual browser in the background to send the request to the 'front domain'. As a result, the external attacker will not see the difference between a normal browser request and a *meek* request.

3.3 Packet fragmenting

Finally, a technique suggested by Winter et al. is the fragmentation of the packets sent from the client to the bridge relay [9]. This can be done by decreasing the TCP window size advertised by the bridge. The small window size will result in the client splitting its TLS cipher list across two TCP segments, which makes it harder to fingerprint. The authors show that a tool implementing this circumvented the Chinese firewall.

Note though that this is not a sustainable solution, as the censor might implement packet reassembly and as a result be able to analyse the full request.

4 Discussion

We have seen that even though steganography cannot be established by Tor itself, multiple researchers have investigated how the use of Tor can be hidden. There are a number of promising options, although not all solutions have been publicly deployed yet.

It turns out that hiding that one is using Tor always comes at the price of substantial performance loss, on top of the performance loss that Tor already causes. Since the regimes where steganography is necessary often have unreliable Internet connections, any performance loss should be avoided. Some performance loss seems inevitable, although the difference in performance loss between the suggested methods is quite remarkable.

Another observation is that the suggested methods are not completely bullet proof: some might still be recognised as (obfuscated) Tor traffic or are still vulnerable to censorship. *obfsproxy* for instance mainly frustrates DPI, but might still be recognised and blocked if more advanced DPI techniques are used. *SkypeMorph* is very promising with an acceptable performance loss, but if a censor would decide to block all Skype traffic (which is not that unlikely, since Skype itself provides encrypted messaging and phone calls), the whole system is useless.

Therefore, it should be noted that more research in this area is still needed to improve the possibilities for those who live in repressed regimes, being denied access to the open Internet. Every help to let the human rights of freedom of information, freedom of speech and freedom of press prevail, is necessary.

References

1. asn. obfs2 protocol specification. `https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/tree/doc/obfs2/obfs2-protocol-spec.txt` [accessed 3 July 2015], Jan 2015.
2. asn, Philipp Winter, Ian Goldberg. obfs3 protocol specification. `https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/tree/doc/obfs3/obfs3-protocol-spec.txt` [accessed 3 July 2015], Jan 2015.
3. Dingledine, R., Mathewson, N., and Syverson, P. Tor: The Second-generation Onion Router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13* (Berkeley, CA, USA, 2004), SSYM'04, USENIX Association.
4. Fifield, D., Lan, C., Hynes, R., Wegmann, P., and Paxson, V. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies 2015*, 2 (2015), 1–19.
5. M. Price, M. E. Persian cyberspace report: Internet blackouts across Iran. `http://iranmediaresearch.com/en/blog/101/12/02/09/840` [accessed 5 June 2015], 2012.
6. Mohajeri Moghaddam, H., Li, B., Derakhshani, M., and Goldberg, I. Skypemorph: Protocol obfuscation for tor bridges. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (New York, 2012), CCS '12, ACM, pp. 97–108.
7. The Tor Project, Inc. Obfsproxy. `https://www.torproject.org/projects/obfsproxy.html.en` [accessed 2 July 2015], 2015.
8. Weinberg, Z., Wang, J., Yegneswaran, V., Briesemeister, L., Cheung, S., Wang, F., and Boneh, D. Stegotorus: a camouflage proxy for the tor anonymity system. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), ACM, pp. 109–120.
9. Winter, P., and Lindskog, S. How China Is Blocking Tor. *CoRR abs/1204.0447* (2012).
10. Yawning Angel. obfs4 protocol specification. `https://github.com/Yawning/obfs4/blob/master/doc/obfs4-spec.txt` [accessed 3 July 2015], Jan 2015.