Progress in censorship circumvention: overview of Tor and Pluggable transports

Maikel Zweerink

Abstract. Tor provides many internet censorship circumvention techniques via its flexible pluggable transport interface. This article will discuss the most prominent options: Obfs2/3/4, FTE and Meek and discuss its (dis)advantages for internet censorship circumvention.

Keywords. Tor · Censorship · Traffic obfuscation · Pluggable transports · Obfs2/3/4

1. Internet Censorship

Internet Censorship comes in many forms: black listing websites, whitelisting websites, physically shutting it down, denying access based on status (only the Elitist and some scientists are allowed to access the internet in North Korea) or even based on wealth (in Turkmenistan you pay around 33,700 US dollars per month for a 34Mbps internet connection [1]). Internet Censorship in China is one of the (technically) most sophisticated out there. China is the biggest internet censoring country with a vast amount of internet users (670,000,000 as of 30 Nov 2015 [2]).

Chinese citizens have multiple options to circumvent this internet censorship. Common practice is to use a proxy server or VPN outside of China to access blocked websites. A study in 2010 estimated that about half of the censorship circumvention was executed through simple (transparent) proxy servers and the other half via blocking resistant methods like VPNs or Tor. Interesting to note is that the estimation concluded that only about 3% of China's internet users were thought to actively use censorship circumvention as of 2010 [3,4].

2. Tor and internet censorship circumvention

Tor, a way to access the internet anonymously, is not suited to circumvent internet censorship. This becomes clear if you look at the infrastructure. A Tor client downloads a publicly available list with entry and relay nodes to connect with. Blocking Tor would be as simple as automating the downloading of these lists and adding these relays to a blacklist, as done by the Chinese Firewall.

As the Tor project became aware of this, they created Tor Bridges [5], a project which allows the existence of Tor nodes which are not in the public list. These lists are shared in small bits via private networks (such as mail or via the Tor Bridges website after entering a CAPTCHA). The Chinese Firewall can no longer identify Tor relays via a public list.

But as the Chinese government keeps improving the Chinese Firewall, more advanced filtering techniques are adapted. One of the most recent filtering techniques involves active probing for Tor relays [6]. This method, intended to filter out Tor bridges, uses active computer systems to verify if a TLS

connection setup by a citizen within China is to a Tor relay. Tor makes use of TLS, but the handshake can be identified as Tor due to a specific set of parameters (and is detected by the Chinese Firewall by using Deep Packet Inspection). This active probing is done with a computer system of the Chinse government that uses an arbitrary IP address (suspected to be supplied by Chinese ISPs) to connect to the same host and port that the citizen connected to. After this, the system simulates a Tor handshake and if this succeeds it will drop the connection of the citizen and add the server IP to a blacklist.

Identifying the Tor TLS handshake can be done with easy. If you open Wireshark and initiate a Tor connection, you can see a Client Hello coming by with a distinct set of Cipher Suits supported (compared to a normal browser, although this has been improved to look like Firefox on the client and as Apache on the server side). The handshake also includes a fake domain name (as seen in figure 1).

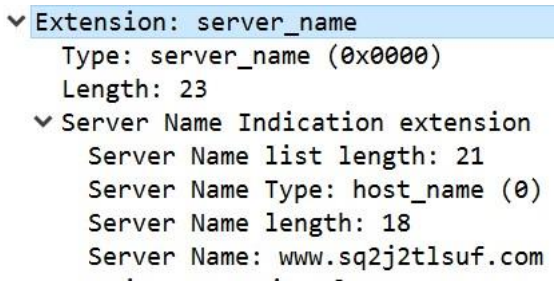

Fig. 1. *The Tor TLS handshake includes the server_name extension with a fake domain name.*

Although Tor tries to mimic website TLS traffic (acting like you visit sq2j2tlsuf.com), this does not stop the Chinese government from identifying and blocking Tor traffic. China actually identifies Tor TLS traffic by a specific set of Cipher Suites used by Tor. After the (possible) identification of a Tor TLS connection, the Chinese Firewall will actively probe the Tor Entry Node to verify the server is indeed a Tor Node. The identification of Tor TLS traffic is actually a crucial part in the internet censorship, since it would not be cost effective to probe all TLS connections.

Another way to identify the Tor TLS handshake, is via DNS. The agent could simply query each server_name within a TLS handshake to verify it is a legitimate domain name. This method is more efficient that probing all TLS connections, but would still require large (probably cost ineffective) infrastructure to check all TLS handshakes. This method was also rumored to be used in 2012 by the Iranian government, but there was no (reconstructing) proof of the concept [7].

The main reason why abroad TLS connections within an internet censoring country are not being blocked, are the economic dependencies on these kind of technologies. Especially China who has an extremely big export market, relies on the business with foreign companies.

As you can conclude, the Tor network has already come a far way to become better in internet censorship circumvention. Things like the Tor Bridges project already improved the core infrastructure of Tor to become more resistant against basic Tor blocking.

However, the “attack vector” for countries that like to ban Tor is still quite large. Core problems include the identifiable TLS connection from Tor which allows Tor traffic to be singled out due to distinctive deep packet inspection techniques. The second security problem is the possibility for the government to actively probe a Tor relay to verify it is indeed Tor traffic. These core security problems will be addressed further in this document, since traffic obfuscation is aiming to solve these Tor identification problems.

3. Pluggable transports

3.1 Pluggable transports and censorship circumvention

Pluggable transports within Tor are a fundamental tool to circumvent internet censorship with Tor [8]. Pluggable transports is a system which allows a Tor client to communicate to a Tor Entry Node via protocols which are much harder to identify and thus to censor. The pluggable transport is a separate interface in which all Tor traffic is tunneled through (much like the idea of a VPN connection). As the figure below explains, Pluggable transports are meant for obfuscation in the first link of a Tor connection: between the Tor Client and Tor Entry Node. The model assumes that the Tor Entry Node is outside the reach of the internet censoring agent and is stationed in a non-censoring network.

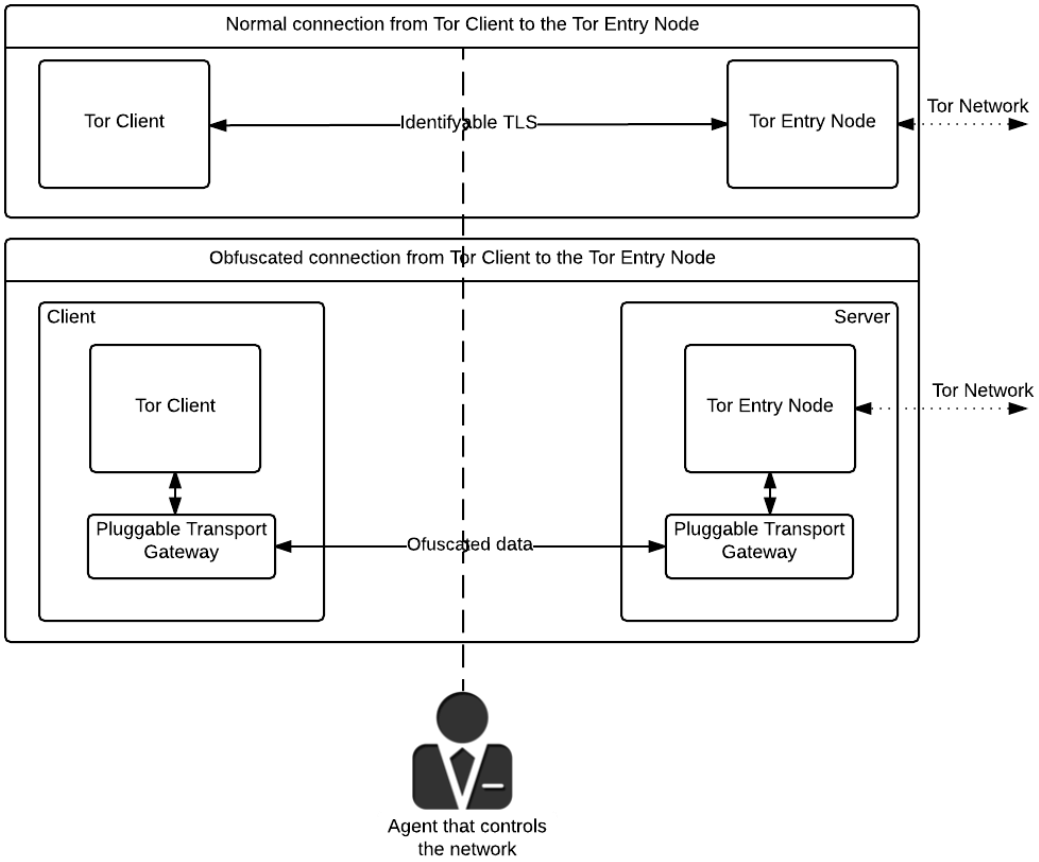

Fig. 2. *A pluggable transport tries to obfuscate the traffic between the Tor Client and the Tor Entry Node.*

The pluggable transports is an interface which can be used for various protocol implementations. Note that both the Tor client and the Tor Entry Node must support the pluggable transport implementation for it to work. We will discuss the currently deployed and most promising implementations in the next chapters.

3.2 Obfs2

Obfs2, designed in 2012 was designed to combat the DPI done by the Iranian government [9]. The obfuscation design was based on the encryption of the TLS connection from Tor with a Stream Cipher (AES-CTR-128).

It exists of two parts: the handshake and transfer. The handshake starts with the following properties:

You have a initiator (Tor Client) and the responder (Tor Entry Node), both have a secure random seed of 16 bytes. Both also create a padding key used in the handshake in the following matter: `MAC("Responder obfuscated data", INIT_SEED)[:16]` (note that "Responder" is replaced with "Initiator" depending on the role). MAC is defined as hashing the values in the following manner: `MAC(s, x) = SHA256(s | x | s).` Note that only the first 16 bytes of the hash is used as a padding key.

After this the initiator and responder sends the following:

```
INIT_SEED | E(INIT_PAD_KEY, UINT32(MAGIC_VALUE) | UINT32(PADLEN) |
WR(PADLEN))
RESP_SEED | E(RESP_PAD_KEY, UINT32(MAGIC_VALUE) | UINT32(PADLEN) |
WR(PADLEN))
```

After this exchange, both parties derives the padding key and verifies that the magic value and padding lengths are correct. After this the keys for the keys for the stream cipher are generated:

```
INIT_SECRET = MAC("Initiator obfuscated data", INIT_SEED|RESP_SEED)
RESP_SECRET = MAC("Responder obfuscated data", INIT_SEED|RESP_SEED)
INIT_KEY = INIT_SECRET[:16]
INIT_IV = INIT_SECRET[16:]
RESP_KEY = RESP_SECRET[:16]
RESP_IV = RESP_SECRET[16:]
```

After this the `INIT_KEY` and `INIT_IV` are used in AES-CTR-218 by the Tor Client and `RESP_KEY` and `RESP_IV` by the Tor Entry Node.

As the first pluggable transport it was effective enough to stop the Iranian government from censoring, but this method no longer succeeds in the Chinese Firewall [6]. This is because the two most important security goals of this obfuscation protocol are not met. As described above the first problem is an identifiable connection. Obfs2 can be easily detected because of the handshake. An agent that watches the key exchange can also determine the secrets (you can determine all secrets and IV's used by using `INIT_SEED` and `RESP_SEED` of the initiator and responder) used in the symmetric stream cipher. By simply decrypting the `MAGIC_VALUE` you can determine obfs2 is used. If you want to be sure Tor is used, you can decrypt the first few bytes of the transfer phase and check for a TLS handshake.

The second security problem is the ability for the agent to actively probe the Tor Relay Node to check if it responds to the initial obfs2 handshake, just like the Chinese Firewall has implemented at the moment. These two security problems are also the main reason why newer methods have risen such as obfs3 and obfs4 (Scramble Suit).

3.3 Obfs3

Designed as a follow up of obfs2, obfs3 solves the ability for an agent to passively decrypt a part of the stream to determine Tor is used. Obfs3 uses an anonymous Diffie and Hellman key exchange (no authentication in a PKI) to securely exchange a secret used in the symmetric AES-CTR-128 encryption method (note again that different keys are used for the initiator stream and responder stream) [10].

The obfs3 specification describes a 1536 bits group 5 Modular Exponential (MODP) Diffie and Hellman. Interesting to note is that RFC 3526 advises that for AES 128 bits a much stronger DH should be chosen, because 1536-bits group 5 will only give comparable results of a 70 to 80 bits symmetric key [12]. A 3200-bit group is advised to let the DH exchange not become the weak link. During a mail conversation with Ian Goldberg it became clear that this is a tradeoff between efficiency and security.

Obfs3 includes a custom version of the DH protocol called UniformDH [10], because the default DH key exchange is not truly random in bytes sent to each other. Because of the use of a fixed group, the first few bytes of the initiator will be in the same group as the first few bytes of the responder. To avoid this identification of the protocol a custom method is used [11]. This custom method involves the randomly choosing to send either `X` or `p-X` (which will both be valid because `(p-X)^x = X^x mod p`, `x` is private key and `X` is the public key). Also note that there is only 1535-bits of randomness because the lowest bit needs to be 0 to prevent the small subgroup attack.

After the key exchange the following properties are determined in the following way (almost similar to obfs2):

```
INIT_SECRET      = HMAC(SHARED_SECRET, "Initiator obfuscated
                 data")
RESP_SECRET      = HMAC(SHARED_SECRET, "Responder obfuscated
                 data")
INIT_KEY         = INIT_SECRET[:16] (last 16 bytes)
INIT_COUNTER     = INIT_SECRET[16:] (last 16 bytes)
RESP_KEY         = RESP_SECRET[:16]  (first 16 bytes)
RESP_COUNTER     = RESP_SECRET[16:] (first 16 bytes)
```

After this the `INIT_KEY` and `INIT_COUNTER` are used in `AES-CTR-128` by the Tor Client and `RESP_KEY` and `RESP_COUNTER` by the Tor Entry Node.

After the handshake is complete, there is still one step to do. Before the initiator and responder can send data, they must once send some random data ending in the HMAC with either “Initiator magic” or “Responder magic” based on their respective role. `WR(PADLEN2)` indicates weak random data with the length of `PADLEN2`. `PADLEN2` is in `[0, 4097]`. This step is done to create more protocol obfuscation in data lengths.

```
WR(PADLEN2) | HMAC(SHARED_SECRET, "Initiator magic") | E(INIT_KEY, DATA)
```

Despite being a great improvement over obfs2, obfs3 still lacks an authenticated key exchange, meaning that an agent could MitM the exchange and retrieve the secrets used in the session. This allows the agent to identify Tor traffic. Traffic could also still be heuristically discovered by looking at the data flow, and with the ability to still actively probe the Tor Node Relay single out all Tor connections over obfs3. The main focus of obfs4 is to solve the authentication and flow identification problems.

3.4 ScrambleSuit / Obfs4

Designed as a follow-up of obfs3, obfs4 (which is based on ScrambleSuit with slight modifications) aims to solve the passive identification and active probing attacks. It provides authenticity (PKI via BridgeDB), integrity and confidentiality.

ScrambleSuit was designed as a follow up of obfs3 providing protection against active probing [13]. Obfs4 improves ScrambleSuit by providing the same security properties as ScrambleSuit but also includes an authenticated key exchange via BridgeDB. Querying for obfs4 Tor Entry Nodes via BridgeDB does not only give you IP addresses, but also the public ID (`NODEID`) and public key (Curve25519, denoted as `(B,b)`) of the Node. Curve25519 is a Montgomery `y2 = x3 + 486662x2 + x` over $2^{255} - 19$ curve and was chosen for being one of the fastest ECC schemes available. You need knowledge of these three variables to connect to this node via obfs4.

The transport over obfs4 also consists of two parts: the key establishment stage and the data transfer phase. The client generates a Curve25519 key pair `(X,x)` and determines the following values:

```
X’  = Elligator 2 representative of X of 256 bits.
P_C = Random padding ranging from 85 bytes to 8128 bytes.
M_C  = HMAC-SHA256(B | NODEID, X’)[128:] (only the first 128
bits count) E  = String containing the Epoch time in hours.
MAC_C  = HMAC-SHA256(B | NODEID, X’ | P_C | M_C |
E)[128:]
```

By default curve points are easy to identify within a data stream because you can check if the coordinates satisfy an elliptic curve equation. Elligator 2 is an obfuscation method to hide Elliptic curve points and make them look like strings of random data. This is why X’ is send to over the line, instead of the X.

After this the Initiator sends this request:

```
X' | P_C | M_C | MAC_C
```

Now the Responder will check the M_C (by computing it himself and matching it in the byte stream of the Initiator). After this it calculates MAC_C in three different variations: E, E-1 and E+1. This is to compensate any clock differences between the Initiator and Responder. If these steps fail in any way, the Responder should drop the connection at a random interval to make probing even more difficult.

Now the Responder can derive X from X’ using reverse Elligator 2. After this a Curve25519 keypair (Y,y) is generated and sends the following back:

```
Y’  = Elligator 2 representative of Y of 256 bits.
AUTH = ntor authentication tag of 256 bits.
P_S = Random padding ranging from 85 bytes to 8128 bytes.
M_S  = HMAC-SHA256(B | NODEID, Y’)[128:] (only the first 128
bits count) E   = The same E that the Initiator used.
MAC_C  = HMAC-SHA256(B | NODEID, Y’ | AUTH | P_S | M_C | E)[128:]
```

The ntor authentication tag is calculated in the following way [14]:

```
SECRET_INPUT     = EXP(X,y) | EXP(X,b) | NODEID | B | X | Y |
                 PROTOID
KEY_SEED         = H(SECRET_INPUT, PROTOID | ":key_extract")
VERIFY           = H(SECRET_INPUT, PROTOID | ":verify")
AUTH_INPUT       = VERIFY | ID | B | Y | X | PROTOID |
                 "Server"
AUTH             = H(AUTH_INPUT, PROTOID | ":mac")
```

Where `PROTOID` is a string telling the current variant used (e.g. ntor-curve25519-sha256-1). H can be any hashing function within ntor. The Responder now sends back to the Initiator:

```
Y' | AUTH | P_S | M_S | MAC_S
```

The Initiator checks if it comes to the same result for `AUTH` as the Responder (by using `EXP(Y,x) | EXP(B, x)`). Both the Initiator and Responder derive with the AUTH a KEY_SEED of 256 bits which is expanded with ntor KDF to 144 bytes. This key material is split into a 256 bit NaCl secretbox key, 128bit nonce prefix for NaCl, 128 bit SipHash-2-4 key and a 64 bit SipHash-2-4 Initialization Vector. NaCl is a library for high speed crypto computations within C. Poly1305 is used for the authenticated encryption of data packets. NaCl has chosen for a stream cipher implementation of Poly1305 with Salsa20. SipHash2-4 is used as a MAC of 64 bits [15].

In the second phase data transfers can take place by using the NaCl secretbox in combination with the SipHash-2-4 algorithm to provide both confidentiality, integrity and authenticity.

Obfs4 seems like an big incremental upgrade over obfs3 satisfying both the obfuscation of the traffic (disallowing identification) and the ability to resist active probing mechanisms (by requiring the knowledge of the NODEID and Cruve25519 public key). The fact that obfs4 also provides authenticity which ScrambleSuit does not, also makes this protocol sound against active Man In the Middle Attacks by the Agent that controls the network.

3.5 Format-Transforming Encryption

FTE tries to obfuscate the traffic in a very different way than obfs. Instead of circumvention censorship by making the traffic look as random as possible, FTE tries to do the opposite. FTE is a very dynamic system which hides encrypted traffic within HTTP/SSH or basically any regex that you provide it with [16]. Because of this hiding in clear sight, FTE can even work in networks where a protocol whitelist is enforced. FTE also makes use of the BridgeDB project to share IP addresses and Node ID's with the users.

A Deterministic finite Automata is derived from the regular expression, which is used to rank/unrank the encrypted traffic to fit its particular regular expression [17]. There is no key/pattern establishment phase within FTE, which means that the client just starts talking to a FTE server without actually telling which

pattern it uses [18]. The server needs to decide on the fly which request/response regex is used for it to respond correctly. Tor however, seems to always use an HTTP pattern. This HTTP pattern also does not includes commonly used headers (such as Host and Content-Length).

It is hard to determine the security properties of FTE, due to the lack of research which is available/done on current implementations. According to the first paper, FTE is active probe resistance, but it is not stated why it can resist such an attack. This could be either due to the inability for an Agent to identify possible circumvention traffic or if the protocol resists an actual probing attempt (such as requiring the knowledge of a node ID). I contacted Kevin P. Dyer to clarify this, but as of now did not respond yet. The code in the official FTE repository does not show any kind of Node ID usage within the data exchange [18].

By looking at the FTE traffic flowing by via the Tor browser, there is no way to distinguish the traffic created by FTE beside the lack of proper HTTP headers. This is why current implementations in the Tor browser can be considered to be identifiable by an Agent, but you could already fix this by changing the HTTP regex to contain proper headers.

3.6 Meek

Meek is an implementation that heavily relies on domain fronting [19]. Domain fronting is the idea of having a covert channel via a normal looking TLS connection. The client starts with sending a TLS Client Hello with an SNI that is allowed by the censoring agent. After the handshake succeeds, Meek starts wrapping the actual Tor TLS traffic in a HTTP POST body within the outer TLS stream [19]. This HTTP POST contains the actual server (Host) used to circumvent censorship, which cannot be seen by the Agent. The core principle relies on the fact that big service providers (such as Google, Amazon and Azure) provide multiple services via a front domain (or edge server), which handles all TLS connections and forwards the HTTP requests to the appropriate service.

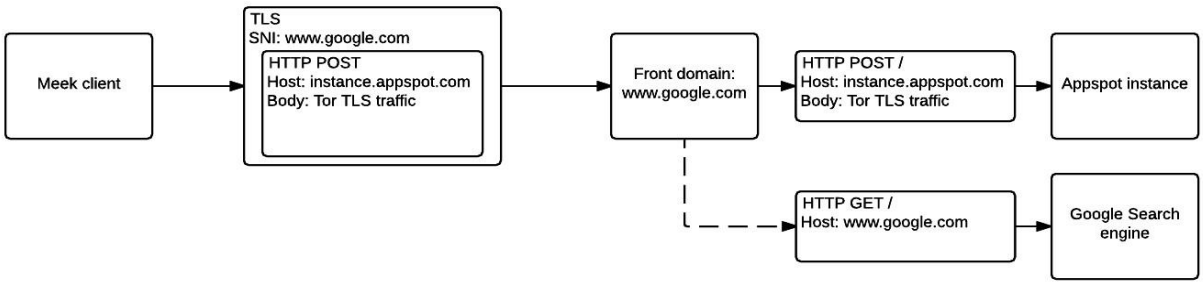

Fig. 3. *The Tor TLS traffic is wrapped within a HTTP POST request within the TLS stream to the front domain (or edge server) of the provider of multiple services.*

As you can see in the figure, it will be hard for an agent to determine if the TLS stream between the Meek client and the front domain is actually requesting www.google.com or if it's sending POST data to a hosted appspot instance. It can only see the SNI of the most outer TLS handshake, which will contain www.google.com. All requests go to the same IP address, the cipher suites within the outer TLS

handshake looks like an ordinary handshake for HTTPS. Meek relies on the fact that a lot of these service providers provide free cloud bases services to host virtual private servers that can be used to run the meek server [20].

TLS connections do not require the SNI to be set, in fact International Computer Science Institute's certificate notary states that 16,5% of all TLS connections lack SNI. This makes it hard for a agent to censor based on TLS SNI, because it has the potential to block a lot of services at once [21].

Meek is a very interesting solution to internet censorship. The usage of existing infrastructure e.g. the TLS connection to a front domain such as amazon.com or google.com provides authenticity which prevent MitM attacks by the agent also provides excellent cover because it blends right in normal traffic. Identification of these kind of data streams will be very costly for an agent, since it is only would be possible by looking at the distinct package sizes which will be hard and must be state full. The requirement of knowledge of the host within the HTTP POST request also provides active probe prevention, because the agent cannot determine what host is accessed within the CDN.

The downside of course is the fact that you need to setup your own Meek server within any of the (free) services. There is also the possibility that the Agent will block the whole CDN, as done with Google in China. However, there are still plenty of alternatives such as Azure or Amazon.

4. Conclusion

Once Iran started applying censorship to its citizens the cat and mouse game started for pluggable transports within Tor to circumvent censorship. Despite being originally not built for circumvention of censorship, the Tor network has the resources and interfaces available to do so. Technically 1 hop would already be enough to circumvent censorship, because you only need a proxy outside the censored network. A lot of these traffic obfuscation protocols are actually deployed and usable without the use of tor, for example VPN connections (obfsproxy) or P2P networks (such as Lantarn which also uses domain fronting [23]).

The core problem is the communication between the Tor Client and Tor Entry Node, which goes over the network that the censoring agent controls. These obfuscation protocols that try to hide the user activities needs to be unidentifiable for the agent, which means that the traffic will look like pure random data or very common allowed network traffic. The second requirement for these circumvention techniques is the ability to become active probing resistant; this means that the agent who tries to enforce censorship cannot verify that a certain server is giving user the ability to circumvent the censorship. The way that the GCF has tried this, was by acting like a Tor Client that was willing to contact the server.

Older protocols such as obfs2 and obfs3 did their job at the time, but no longer succeeds in China because the ability of being identified by traffic patterns and being probed. Current prominent solutions include obfs4, FTE and Meek. Obfs4 tries to look like as random as possible, FTE and Meek try to look as common as possible in terms of network traffic. These protocols could still be identified, but not cost effectively identified and blocked. FTE and obfs4 is hosted by voluntaries, Meek requires to get a (paid) VPS on a popular CDN to host the server.

Apparently many internet censoring countries think that the using of a whitelist system does not work against all the economic / social and political costs involved. As long as this no-whitelist principle is applied, the three most prominent and security sound obfuscation methods will work within that censored network.

5. Opinion

I found out that the pluggable transports interface that Tor provides, is a clean and highly functional system for different communication and obfuscation methods. Information about the inner workings of these obfuscation methods however, is hard to find. I had to derive a lot of information from dumped source code on git servers. It is also interesting to learn that Tor is not a necessary component in internet censorship circumvention in the technical sense (having a server outside the censored network is basically enough), but Tor is mainly used for its large available network with many different entry nodes and IP addresses, meanwhile also providing extra anonymity. I also had a mail conversation with Ian Goldberg about the Obfs3 spec. He explained everything to me in detail.

Appendix A

References

1. Enemies of the Internet. (2014, March 10). Turkmenistan: News black hole. Retrieved from Enemies of the Internet: http://12mars.rsf.org/2014-en/2014/03/10/turkmenistan-news-black-holeturkmentelekom/
2. Internet World Stats. (November, 2015). Internet Users in Asia November 2015. Retrieved from Internet World Stats: http://www.internetworldstats.com/stats3.htm
3. Hal Roberts, Ethan Zuckerman, Jillian York, Robert Faris, and John Palfrey. (October 2010). 2010 Circumvention Tool Usage Report. The Berkman Center for Internet & Society. Retrieved from: http://cyber.law.harvard.edu/sites/cyber.law.harvard.edu/files/2010_Circumvention_Tool_Usage_Report.pdf
4. 32C3. (2015, December 30). Michelle Proksell : CHINTERNET ART. Retrieved from YouTube: https://www.youtube.com/watch?v=mQH-clMdkGs
5. Project, T. T. (n.d.). BridgeDB. Retrieved from BridgeDB: https://bridges.torproject.org
6. Roya Ensafi, D. F. (2015). Examining How the Great Firewall Discovers Hidden Circumvention Servers. Retrieved from: http://conferences2.sigcomm.org/imc/2015/papers/p445.pdf
7. Project Tor. (2012) How is Iran blocking Tor? Retrieved from: https://trac.torproject.org/projects/tor/ticket/7141
8. Project Tor. Tor: Pluggable Transports. Retrieved from Tor: https://www.torproject.org/docs/pluggable-transports.html.en
9. Project Tor. (January 2015). pluggable-transports/obfsproxy obfs2. Retrieved from: https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/tree/doc/obfs2/obfs2-protocolspec.txt
10. Project Tor. (January 2013). pluggable-transports/obfsproxy obfs3. Retrieved from:

https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/tree/doc/obfs3/obfs3-protocolspec.txt
11. Project Tor. (December 2012). RFC on obfs3 pluggable transport. Retrieved from: https://lists.torproject.org/pipermail/tor-dev/2012-December/004245.html
12. T. Kivinen, M. Kojo. (May 2003). More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE). Retrieved from: https://www.ietf.org/rfc/rfc3526.txt
13. Yawning Angel. (January 2015). obfs4-spec.txt. Retrieved from: https://github.com/Yawning/obfs4/blob/master/doc/obfs4-spec.txt
14. Project Tor. (May 2011). Ntor handshake: Improved circuit-creation key exchange. Retrieved from: https://gitweb.torproject.org/torspec.git/tree/proposals/216-ntor-handshake.txt
15. J.P. Aumasson, D.J. Bernstein. SipHash: a fast short-input PRF. Retrieved from: https://131002.net/siphash/siphash.pdf
16. K. P. Dyer, S. E. Coull, T. Ristenpart, T. Shrimpton. (2013). Protocol Misidentification Made Easy with Format-Transforming Encryption. Retrieved from: https://kpdyer.com/publications/ccs2013-fte.pdf
17. K. P. Dyer, S. E. Coull, T. Ristenpart, T. Shrimpton. (2013). Protocol Misidentification Made Easy with Format-Transforming Encryption (Slides). Retrieved from: https://realworldcrypto.files.wordpress.com/2013/06/shrimpton.pdf
18. K. P. Dyer, S. E. Coull, T. Ristenpart, T. Shrimpton. (2013). Gitlab code FTE. Retrieved from: https://github.com/kpdyer/fteproxy
19. Project Tor. (2015). Pluggable Transports explained: Meek. Retrieved from: https://trac.torproject.org/projects/tor/wiki/doc/AChildsGardenOfPluggableTransports#meek
20. D. Fifield, C. Lan, R. Hynes, P. Wegmann, V. Paxson. (2015). Blocking-resistant communication through domain fronting. Retrieved from: http://www.icir.org/vern/papers/meek-PETS2015.pdf
21. D. Fifield. (January 2015).[tor-talk] What to do if meek gets blocked. Retrieved from: https://lists.torproject.org/pipermail/tor-talk/2015-January/036410.html
22. The ICSI Certificate Notary. (January 2016). Total number of certificates. Retrieved from: http://notary.icsi.berkeley.edu/
23. Lantarn. (2015). Github code Lantarn. Retrieved from: https://github.com/getlantern/lantern