Identifying TLS abnormalities in Tor

Anders Olaus Granerud

Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2010

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Identifying TLS abnormalities in Tor

Anders Olaus Granerud

2010/12/1

Abstract

This master thesis will examine Tor, which is an open-source anonymizing service used on the Internet. Tor gives the user an anonymous connection to the Internet in real-time. The research on Tor has been considerable and it is a mature project.

Tor uses a network of relays to communicate anonymously. The client software establishes a circuit across the network and makes the final connection from the exit node. The data is relayed back to the user over the hidden circuit. The circuit is build using separate encrypted transport layer security (TLS) connections with public key infrastructure. If these TLS streams can be identified an adversary could filter or block Tor traffic. This could prevent people living under censorship from using Tor and risk prosecution from Internet usage. We have investigated TLS traffic generated by Tor and compared with common TLS traffic. We have studied normal TLS traffic behavior in order to compare with Tor. Our results identifies where Tor diverges from normal TLS traffic and proves that Tor can be detected. In particular we investigated the TLS handshake and certificate which revealed Tor circuit creation. Finally we propose a new layer to Tor which can even out the differences and make Tor more censorship resistant.

Keywords

Security and privacy protection, Public key cryptosystems, Traffic analysis, Censorship, Internet

Sammendrag

Denne masteroppgaven undersøker Tor som er en åpen kildekode anonymiseringstjeneste brukt på Internett. Tor tilbyr brukerne en anonym tilkobling til Internett i sanntid. Forskningen på Tor gjennom årene har vært betydlig og gjort Tor til et modent prosjekt.

Tor bruker et nettverk av releèr for å kommunisere anonymt. Klient-programvaren etablerer en forbindelse over nettverket som videresender trafikk til og fra brukeren, samt gjør den siste forbindelsen fra utgangsnoden til destinasjonen. Den skjulte forbindelsen mellom nodene bygges over separate krypterte TLS-forbindelser med offentlig nøkkel-kryptografi. Disse TLS forbindelsene kan bli identifisert av en motstander som kan filterere eller blokkere Tor trafikk. Dette kan hindre mennesker som lever under sensur å bruke Tor og dermed risikere påtale fordi de brukte Internett. Vi har studert TLS-trafikk generert av Tor og sammenliknet med vanlig TLS-trafikk. Vi har også studert vanlig TLS-trafikk for å gjøre sammenlikningen. Våre resultater identifiserer hvor Tor divergerer fra normal TLS-trafikk og beviser at Tor kan detekteres. Mer spesifikt undersøkte vi TLS-oppkoblingen samt sertifikatet og avslørte oppkobling av en Tor-forbindelse. Til slutt foreslår vi et nytt lag i Tor som kan jevne ut forskjellene og gjøre Tor mer usynlig på nettverket.

Preface

I would like to thank my supervisor Lasse Øverlier for insight and motivation, my wife for her patience, friends and family for keeping me motivated, my colleges for feedback and discussions. Finally, thanks to the Tor project for serving democracy and free speech.
Anders Olaus Granerud, 2010/12/1

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Several governments, organizations and other institutions have the ability and resources to control Internet access. This power can be abused to filter content on the Internet, shutdown servers, reveal the identity of whistle blowers, dissidents or other activists. This could be fatal in some circumstances and countries. Thus there is a demand for anonymous Internet access for people living under censorship, surveillance and fear. Even individuals living a free country may wish to protect their privacy from the government or other powers. The key issue is to protect the individual against a lager force for whatever reason. Anonymity makes it possible to use the Internet without risk of reprisals.

Some countries execute censorship on the Internet to deny the population access to certain content. The censorship differ between nations depending on the government and its laws. One reason for censorship is to keep the opposition under control and from gaining support in the population. This is unfortunate because a democracy depends on freedom of speech. Tor is a service that provides anonymous connection to the Internet and is capable of circumventing censorship. By doing this master thesis we seek solutions to the situation and propose changes that could make Tor slip through filter devices and providing the individuals with anonymity. A message from China appeared on the tor-relays email list 25.5.2010 stating that China is getting more strict in filtering traffic and using Tor more difficult [1]. This shows that there is indeed a need for anonymity and that Tor needs to improve censorship resistance.

1.2 Privacy

To have privacy can be said to deny an organization, government or anyone access to information about you. Privacy comes from latin, *privatus*, and translates to "separated from the rest" or "deprived of something" [2].

Privacy is a fundamental human right as stated in article 12 of the United Nation's Universal Declaration of Human Rights [3]. In democratic countries the human rights are generally respected. This is not always the case in countries without democracy. Our society is getting more digitalized as time passes. This has a positive effect by making the society more efficient and flexible. But digitalization can also make surveillance easier for governments and organizations. The process of capturing and storing privacy related data is less resource demanding and more omnipresent than ever before. With new services such as Facebook and Twitter it is possible to monitor a person and his/her network of friends, coworkers and family more easily. Consider how many events each day that can be traced back to you. The amount of monitoring and logging present in our everyday life is astonishing.

The OpenNet Initiative has taken on the task of documenting and identifying Internet filtering

and surveillance [4]. This effort tries to shed light on this sensitive problem by keeping track of states and regions executing censorship or surveillance. As stated earlier the technology today enables logging of almost everything a person does in course of a day and even a lifetime. This is provided one has access to all sources of course. This is a great power and comes with great responsibility, monitored by the OpenNet Initiative.

Privacy seems to be harder and harder to achieve in the digital age. In the last two decades it has been a reoccurring topic mainly due to the war on terror and its side effects for the general population. Several groups have interests in collecting privacy information for various reasons. Large corporations collect information to present pin pointed advertisement to the user. This creates more revenue as the advertisement are more likely to address the user. The more they know about your person, the more likely it is that you buy a service or product. The development makes collecting more privacy data on the individual profitable. Privacy data can also be used for surveillance or other purposes and not just for profit. The individual must trust the corporations and their handling of privacy data. Most governments collect privacy data to conduct surveillance and keep an eye on the public in hope of revealing threats against society. Revealing threats is a huge task and can be very demanding. Large amounts of data must be collected and scrutinized. This must contain privacy data to be able to detect people with evil intentions from innocent people. More surveillance can give better security which drives the development.

1.3 Torproject

Tor [5] is a service that provides a low-latency anonymzing network. Tor uses a network of so called onion routers (OR) to hide who made the connection. The onion router principle is designed to prevent the transport medium from knowing who is communicating with who. To utilize this a person installs the Tor software on the computer. The Tor software can then provide anonymity to the user on that computer and hide the true user identity. The software builds a circuit over the Tor network and the client uses this connection. The connection originate from random exit nodes in the Tor network and hides the true origin by using layers of Public key encryption [6]. Tor is dependent of volunteers that run Tor servers (onion routers) around the world to serve the users and providing them with anonymity.

Tor uses the encryption protocol TLS to provide anonymity. The TLS streams from Tor could be identified by an adversary if they differ from other TLS traffic. This is due to the fact that Tor uses TLS in a non-standard way in order to make Tor function[7]. How Tor use TLS in a non standard way will be covered later in the report. The adversary could look for volume, number of packets, certain strings, number of connections in a stream to identify Tor usage on a network. This is called traffic analysis. If the TLS streams were identified the adversary could block this stream and prevent Tor usage on the network. This would in turn deny anonymity to the user and could possibly cause harm. The user would be forced to give up Internet access or face the consequences possible identification.

This master thesis will investigate in what ways Tor's TLS traffic differ or resembles TLS traffic created by more common services. These services could be HTTPS, IMAPS or instant messaging for example. Traffic analysis will be applied to the streams to compare them. The thesis will coherently explore the Tor specification to understand how Tor implements TLS and where it resembles or differ from other TLS services. In order to do this the thesis must also define what normal TLS traffic consists of and decide which parameters to analyze in the two streams. The traffic analysis outcome of Tor TLS versus normal TLS will give a answer to how we can change Tor and make it more censorship resistant. We will not implement, but rather suggest reasonable changes.

It is important to match Tor TLS traffic and normal TLS traffic as good as possible so that an adversary believes it is normal network traffic. It would make Tor less detectable and keep the Tor users safe. Many users rely on Tor keep them anonymous and it would be a step backwards if an adversary could block Tor traffic. This could degrade human rights because users under censorship cannot access Internet services of their choosing and it could in turn influence their free will. By writing this thesis we hope Tor will become more censorship resistant and even more usable. The Tor developers have made some efforts to normalize the Tor traffic, but little scientific approaches has been used [7].

1.4 Research questions

1. *What are the characteristics of a TLS stream in terms of traffic analysis?*
2. *How does Tor implement TLS and how is it different from other TLS services?*
3. *Are there any changes that could make Tor look like a common TLS stream?*

1.5 Summary of claimed contributions

- We have established a baseline for a standard TLS handshake. The baseline was established with data from a isolated test environment and in a field experiment on the Internet.
- We have discovered several differences between a Tor TLS handshake and a normal TLS handshake.
- We implemented the differences as signatures in Snort [8] and show that Tor can be detected. It could make the bridge feature in Tor useless if such filter was deployed country wide for example.
- We suggest a new layer to Tor. The layer would even out changes and shape traffic to resemble a particular service.

2 Background

We start of by explaining some concepts for anonymity and controlling Internet access. In the next section we describe in depth the TLS protocol. The last section describes the Tor project, how it works and some design choices.

2.1 Anonymity

Anonymity on the Internet is not possible today without using special techniques. Every connection gets terminated somewhere and can be traced. The anonymity community tries to make the anonymity better by attacking the service and proposing changes as needed. This is a cat and mouse game where the anonymity is getting better and attacks are getting more advanced. There are several types and implementations of anonymizing services available on the Internet today. The ambitions and level of anonymity differs between them.

The word anonymity means to be without name in Greek and can be divided into linkable anonymity and unlinkable anonymity. With linkable anonymity it possible to correlate one event with an other event. An example could be a disposable cell phone. Calls from the disposable cell phone originates from the same number and can be said to give linkable anonymity. To obtain unlinkable anonymity the user must use a different number and cell phone for every call. For example a true anonymous purchase can only be done with cash. It is not possible to link one purchase with an other purchase when cash is used. The cash cannot be traced to the buyer after the purchase because the cash is not linked to a person. This is true anonymity and it is not possible to link one event with an other. In true anonymity it must be impossible to link events together and it would require to use a connection that cannot be linked with any previous and future connections.

There are different levels of identity disclosure. Goldberg presents the Nymity slider in his Ph.D thesis [9]. The slider divide identity disclosure in levels. Verinymity is the state where one can be certain that an entity belongs to an identity. Such as paying with a credit card and using a PIN. Pseudonymity is the state where one can choose to reveal ones identity. Examples could be using a nickname on IRC or other social services. The Pseudonym creates a level of abstraction between you and the other party preventing them from disclosing your identity.

An anonymity service can be divided into low-latency or high-latency service. High-latency will delay the traffic and make it unsuitable for real-time Internet access. Anonymizing services which utilizes high-latency techniques send the traffic over a time span that makes correlation is difficult. Examples of high-latency anonymity services are Mixmaster[10] and Mixminion[11]. This is typically used for email which doesn't require real-time service. These utilizes the mix principle. All connections travel via proxy which delays each packet in a stream an arbitrary time. Since many connections originates from the same proxy an attacker would have problems corresponding streams with end-to-end timing attacks. High-latency anonymity is not very popular since it does not correlate very well with the bandwidth requirement today. Browsing the

web would be impossible and remote access to a computer painfully slow.

2.2 Internet filtering

The OpenNet Initiative research show that states which utilizes Internet filtering is increasing [4]. By Internet filtering we understand controlling access to the Internet by technical means. This way one can control the flow of certain topics and content on the Internet. The reasons for this can be diverging and motivations diverse. For countries with democratic governments the rationale is often national security and protection of intellectual property. While other countries may use cultural norm and religious views to justify Internet filtering.

2.2.1 Points of control

The Internet infrastructure makes it possible to control Internet access centrally by filter on the international gateway of a country. This gateway typically connects the country to the rest of the world. It can be used to control what information the people has access to and prevent information from leaving the country. An advantage to this method is that one has a limited set of these gateways which makes it less resource demanding to administer.

It takes a little more effort to filter at the Internet Service Provider (ISP). The ISP is often a private company but under law to implement the national filtering rules. This is the most common way of filtering according to the OpenNet Initiative [4]. Institutions can also implement filtering in government departments, schools and other public access points. This doesn't prevent people from accessing the Internet censorship-free at home but it will non the less limit access.

The most invasive place to control access is by placing filtering software or hardware at the end user. This can be done by creating a law which demands that all computers with Internet access must run the government software for example. This is easier to do on public computers in libraries, government departments and such.

There are several ways of doing Internet filtering and controlling Internet access. The traffic volume can be immense and thus the filter must be very efficient. The next four sections describes ways to conduct Internet filtering.

2.2.2 Technical blocking

There are three principal ways to do technical blocking. IP blocking is a technique where a IP address is denied access to the Internet to exclude an website. An other less brutal way to filter websites is by URL blocking. URL blocking only blocks certain URL's or URL's with certain keywords in them. This way can be more specific and filter certain topics without blocking the whole website. By tampering the DNS requests from clients the filter can intercept requests to certain services on the Internet and make them hard to find. This technique uses the DNS protocol and exploits its importance to conduct filtering. All these techniques are in effect a Denial of Service (DOS) but done with other means that a DOS attack normally uses.

2.2.3 Search result removal

By removing and tampering search results a site can be very hard to find. The search provider must cooperate with the government or requests must some how be filtered. When the user searches for a filtered keyword the search will come up empty or with certain sites removed.

This can be frustrating but it will not make the site disappear from the Internet. This is a less invasive way of filtering because the site is still accessible.

2.2.4 Take down

By using legal and jurisdictional means a regulator can threat or get a court order to take down a site. This would commonly happen between private parties where illegal fileshares are threatened with a cease and desist notice and expensive lawsuits. If the regulator controls the Domain Name Servers it can directly deregister the domain and make it invisible to the browser. Whether deregistration makes the site unaccessible by typing IP addresses directly depends on the configuration.

2.2.5 Induced self-censorship

Indirect filtering can appear when the government or some other force intimidates the individuals from running a service on the Internet or the right to execute freedom of speech. This method can be more informal by threating the individuals directly or by legal notice to decease his/her activities. Regardless of the threat, the filtering is executed by forcing the persons behind the information to limit or give up their activities.

2.3 Transport Layer Security Protocol

The Secure Socket Layer [12] protocol was developed by Netscape in 1994, mainly to provide confidentiality but also integrity and authenticity. When the Internet was established there were very few people online and everybody could trust each other. Hence there was no purpose in using encryption or authentication. This changed as more and more people connected to the Internet in the 90's and very soon there was a demand to encrypt communication. Since there was no standard for encrypting communications between a web client and web server Netscape decided to make one. SSL became widely used and several new versions were developed. The first version, SSL 1.0, was never publicly released, but SSL 2.0 was released in 1995 [13]. SSL 2.0 had a number of security issues and a version 3.0 was developed to mitigate this [12]. The Internet Engineering Task Force (IETF) [14] eventually took over the responsibility for the protocol after SSL 3.0 and renamed it TLS. TLS 1.0 came as an upgrade to SSL 3.0 and these two does not interoperate [15]. But TLS 1.0 has the possibility to downgrade to SSL 3.0 connection. TLS has become an industry standard and is currently at version 1.2 [16].

TLS assumes that a connection-oriented transport protocol is used. This is typically Transmission Control Protocol (TCP) and a client/server architecture. TLS can detect message tampering, message interception and message forgery[17]. TLS introduced a new communication layer between the transport layer and the application layer dedicated to security. TLS is actually a collection of protocols: The Record Protocol, Handshake Protocol, Alert Protocol, ChangeCipher Spec Protocol and Application Protocol. As shown in figure 1 the Record Protocol is on top of the Transport Layer and the other TLS protocols located over the Record Protocol[16].

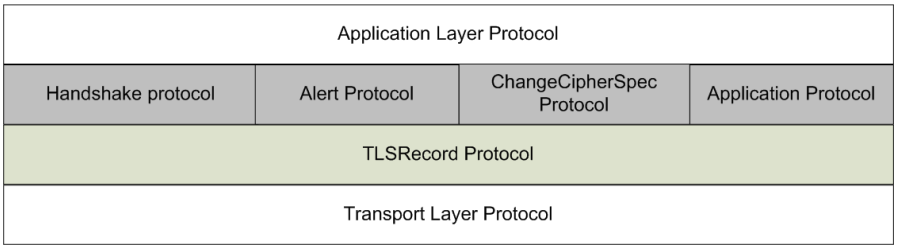

Figure 1: TLS Protocol Layers

2.3.1 TLS Record Protocol

The main responsibility for the Record Protocol is verifying the integrity and origin of the application data. Integrity checking on outgoing and incoming messages with Hashed Message Authentication Codes (HMAC) is used to provide integrity. Further it divides outgoing messages into blocks and reassembling incoming messages. TLS has an option to compress data. If this is the case the Record Protocol handles it. The final task is encrypting incoming and decrypting outgoing messages. After the Record Protocol is finished the data is passed to TCP for transport if it's an outgoing message. The Record protocol header is shown on figure 2 below[16].

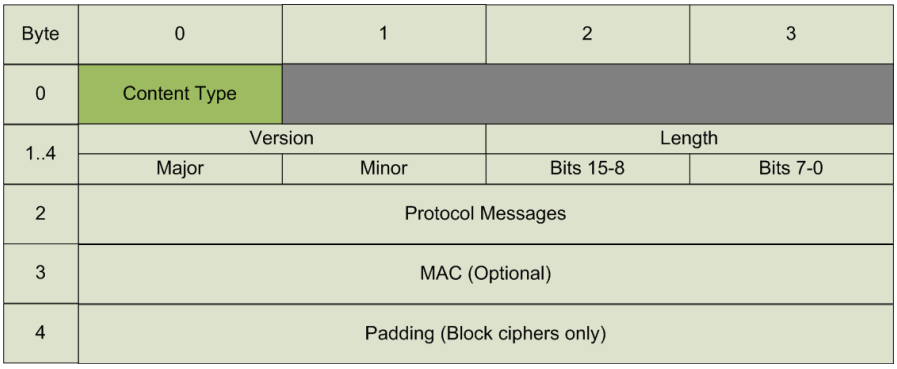

Figure 2: TLS Record Protocol[18]

2.3.2 TLS Handshake Protocol

The TLS Handshake Protocol is responsible for setting up the connection. We focus on server authenticated TLS only and not mutual authentication. In a mutual authenticated session the client will also present a client certificate to authenticate against the server. In a typical TLS connection there are four steps[17]:

1. Handshake and cipher suite negotiation
2. Authentication
3. Key-related information exchange
4. Application data exchange

Step one and two takes place within the Handshake Protocol. Step three take place in the ChangeCipher Spec Protocol. The final step take place in the Application Protocol.

Now that we have an overview lets move on to the details in in the Handshake Protocol. This layer take care of authentication and key exchange in order to establish new or resume TLS connections. The three major tasks for the Handshake protocol are cipher suite negotiation, authentication of the server, and optionally the client, and session key information exchange[19].

In the first step the client and server exchange Hello messages and decide the cipher suite that will be used throughout their message exchange. A cipher suite is the type of crypto algorithm which will be used for this session. The Cipher suite specifies the asymmetric crypto, symmetric crypto with mode and the hashing algorithm used. In the authentication step the server proves its identity to the client. If requested the client must also prove its identity to the server. PKI and certificates are used to authenticate the parties, the exact cipher suite is negotiated in the previous step. The last step is where the client and server exchange random numbers and a Pre-Master Secret. The communicating parties use these numbers to calculate their Master Secret which is shared between them. The Master Secret is used to generate the session keys. Sessions keys consist of a MAC secret for hashing and a key for encrypting. Figure 3 shows the header for TLS Handshake Protocol[16].

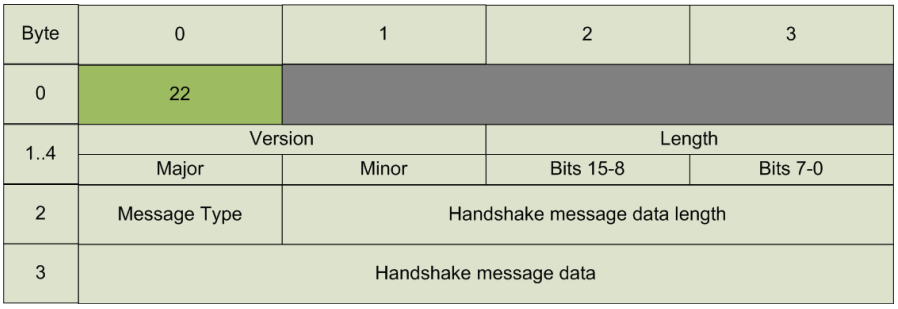

Figure 3: TLS Handshake Protocol[18]

We will now present the steps to create a new secure session with TLS. In the TLS Handshake Protocol we have these steps[19]:

1. The client sends a "Client hello" message to the server, along with the client's random value and supported cipher suites.
2. The server responds by sending a "Server hello" message to the client, along with the server's random value.
3. The server sends its certificate to the client for authentication and may request a certificate from the client. The server sends the "Server hello done" message.
4. If the server has requested a certificate from the client, the client sends it.
5. The client creates a random Pre-Master Secret and encrypts it with the public key from the server's certificate, sending the encrypted Pre-Master Secret to the server.

6. The server receives the Pre-Master Secret. The server and client each generate the Master Secret and session keys based on the Pre-Master Secret.
7. The client sends "Change cipher spec" notification to server to indicate that the client will start using the new session keys for hashing and encrypting messages. Client also sends "Client finished" message.
8. Server receives "Change cipher spec" and switches its record layer security state to symmetric encryption using the session keys. Server sends "Server finished" message to the client.
9. Client and server can now exchange application data over the secured channel they have established. All messages sent from client to server and from server to client are encrypted using session key.

To visualize the process of making a new TLS handshake we have created figure 4[16]:

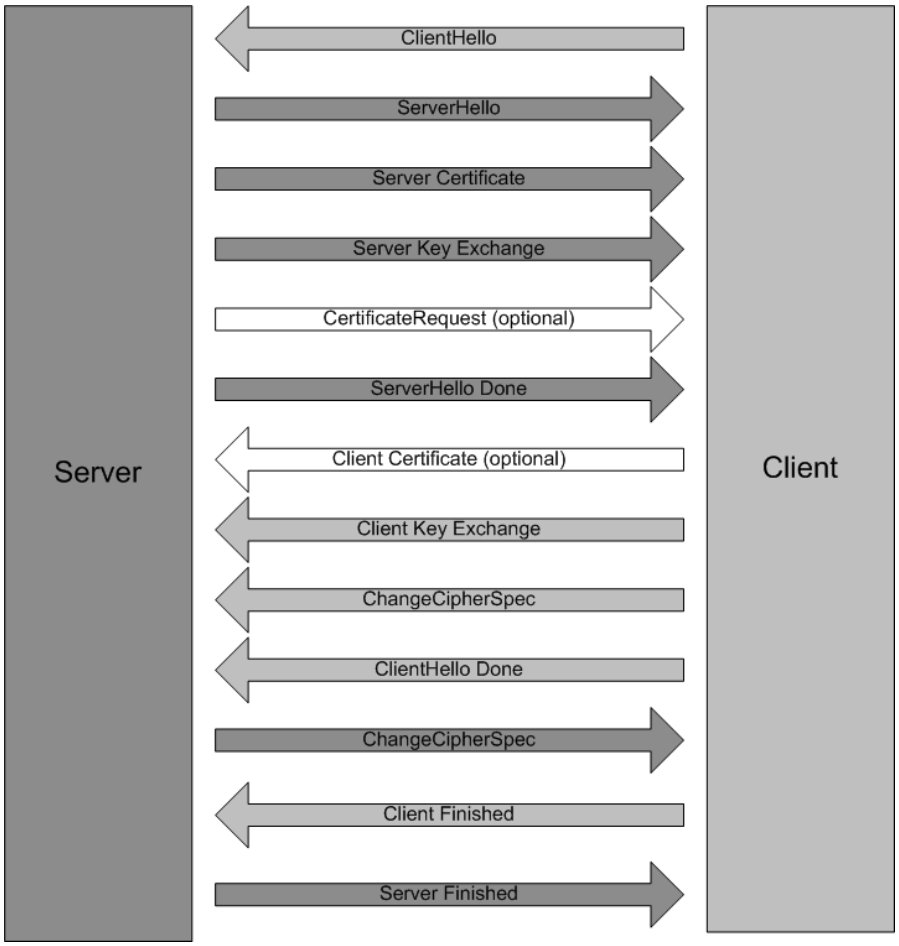

Figure 4: TLS Connection

To resume a TLS connection there are fewer steps:

1. The client sends a "Client hello" message using the Session ID of the session to be resumed.
2. The server checks its session cache for a matching Session ID. If a match is found, and the server is able to resume the session, it sends a "Server hello" message with the Session ID. If a session ID match is not found, the server generates a new session ID and the TLS client and server perform a full handshake.
3. Client and server must exchange "Change cipher spec" messages and send "Client finished" and "Server finished" messages.
4. Client and server can now resume application data exchange over the secure channel.

The process of resuming a TLS connection can be seen on figure 5 below[16]:

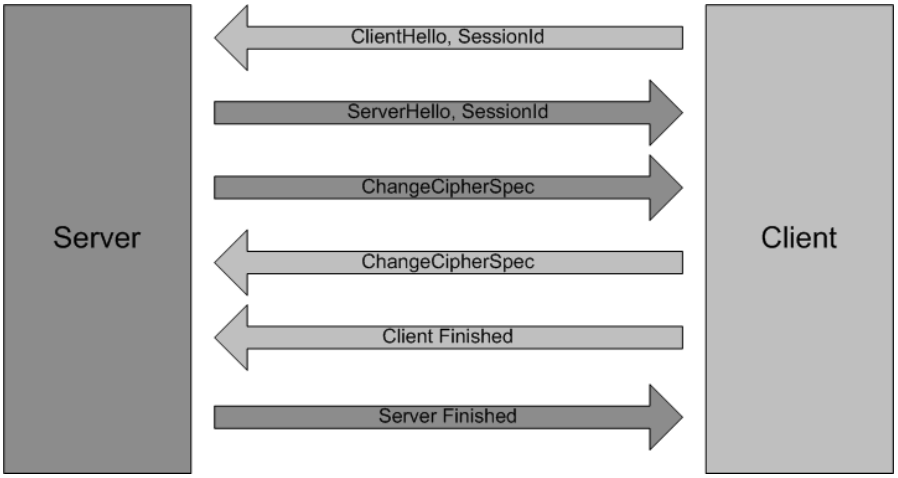

Figure 5: TLS Resume

2.3.3 Change Cipher Spec Protocol

This is perhaps the smallest protocol ever made and it has only a single message. The message is to inform the other end of the connection that all subsequent messages are encrypted. Both parties send the ChangeCipherSpec message and instructs the record layer to start using the negotiated encrypted channel. Figure 6 show the header of the ChangeCipherSpecProtocol.

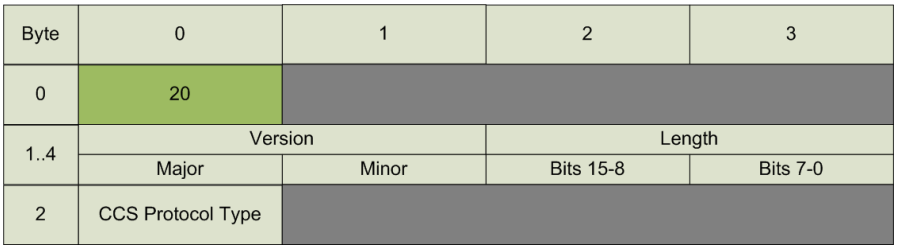

Figure 6: TLS ChangeCipherSpec Protocol[18]

2.3.4 Alert Protocol

The Alert Protocol gives TLS the ability to send messages to the other end of the connection if certain events occur. The alert messages can be fatal which means that both parties must tear down the connection and reconnect. In the case where an alert message is at warning level the receiver makes a decision if the connection can be used further. The different alert messages are described in RFC 5246 [16]. They describe different states such as decryption failed, decompression failure, certificates errors, close notify et cetera. Figure 7 show the TLS Alert Protocol header[16].

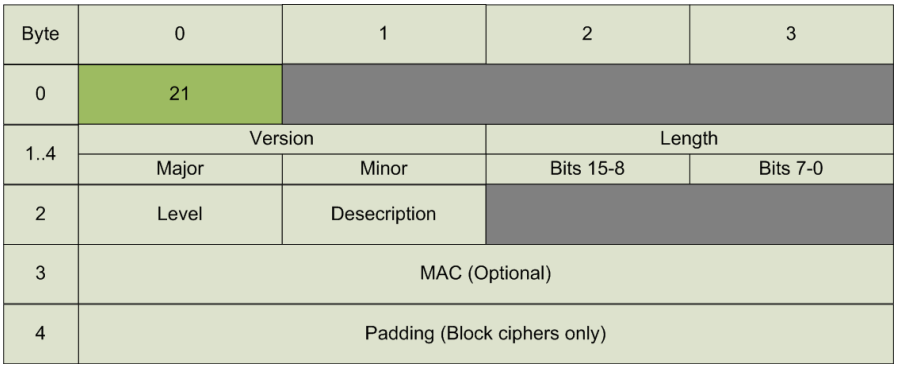

Figure 7: TLS Alert Protocol[18]

2.3.5 Application Data Protocol

This protocol is where the actual data resides. The data sent and received are transported in the Application Data Protocol Messages. The messages are appended with MAC and padding. The header for the Application protocol is shown on figure 8[16].

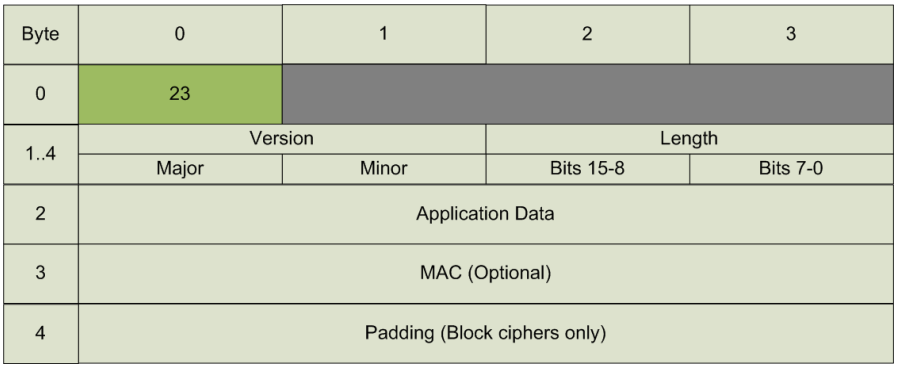

Figure 8: TLS Application Data Protocol[18]

2.4 Tor

Tor evolved from the Onion Routing project at US Naval Laboratories [20]. The goal of the project is to anonymize Internet traffic for low-latency or interactive applications. The Tor servers are run by generous volunteers donating bandwidth and time. It is perhaps the most successful service that provides low-latency Internet anonymity. Tor has been deployed since 2003 and is still growing.

2.4.1 Tor users

Tor can be used by people with good or evil intentions like any type of freedom. One can therefor argue that Tor can be used to support dubious persons, governments or organizations. This was not the intentions of the Tor project, but since Tor is an open and free service it cannot choose its users. However most Tor usage is by legitimate users that appreciates anonymity. The list below is taken from the Tor project website and presents different groups that uses Tor [21].

- Normal people - who want privacy for various reasons.
- Military -want anonymity for intelligence reasons.
- Journalists - want privacy to execute freedom of speech.
- Law enforcement - to conduct surveillance and anonymous tip service.
- Activists and whistleblowers- to avoid being prosecuted.
- High profile people - to have true privacy when traveling
- Business executives -to have confidentiality.
- Bloggers -to avoid being sued or fired.
- IT Professionals - for technical reasons.

The motivation to use Tor for each group differ and these are only some examples. There are probably almost as many reasons for using Tor as there are users. For example governments need to conceal their traffic to prevent counter-intelligence by an adversary conducting traffic

analysis. For businesses the motivation might be to have better network security. This would leverage the bar for an attacker trying to hack the business. For the individual at home the motivation could be privacy. Maybe the the user doesn't want the ISP to be able to read the email from the users computer. The user might also use a public wireless hotspot and doesn't want to be eavesdropped by attackers in the vicinity.

2.4.2 Tor operation

To use Tor the user installs a Tor client on his or her computer. This client runs an Onion Proxy (OP) that builds a circuit through the Tor Network. A circuit is an ordered list of routers that the clients traffic will travel over[22]. Circuits are made of encrypted layers wrapped in each other like an onion. The OP is also responsible for fetching directories and handle connections from user applications. The OP accepts TCP streams and multiplex them across circuits.

Each Onion Router (OR) along the circuit only knows the OR directly previous and successive of it self. The first node on the circuit is called a entry node which knows about the sender. The middle node is responsible for forwarding any data they receive. The last node on the circuit is called the exit node. The exit node is responsible for communicating with the destination chosen by the client and forward the traffic back to the client over the circuit.

To be able to build the circuit the OP must first have list of possible OR's. This is obtained from the directory servers. The directory servers acts as HTTP servers where OP's can fetch current network status and list of available OR's. OR's can upload their state information to the directory servers which maintains a complete view of the Tor network. The collected information given from the OR's to the directory servers is processed and network information can be distributed to OP's and OR's. Some of the directory server IP addresses are hard-coded into the Tor client to prevent misuse[23]. These addresses work as a bootstrap to download information from less trusted directory servers. This is important in order to increase performance in the start up of a Tor instance and prevent spoofing the directory servers[22].

On figure 9 one can see the different layers of encryption in action. Each color represents a encryption layer and the black arrow in the middle represents a request over the circuit to Bob. The yellow arrow represents Alice making a request to the directory servers. This returns a list of all Tor servers in the network.

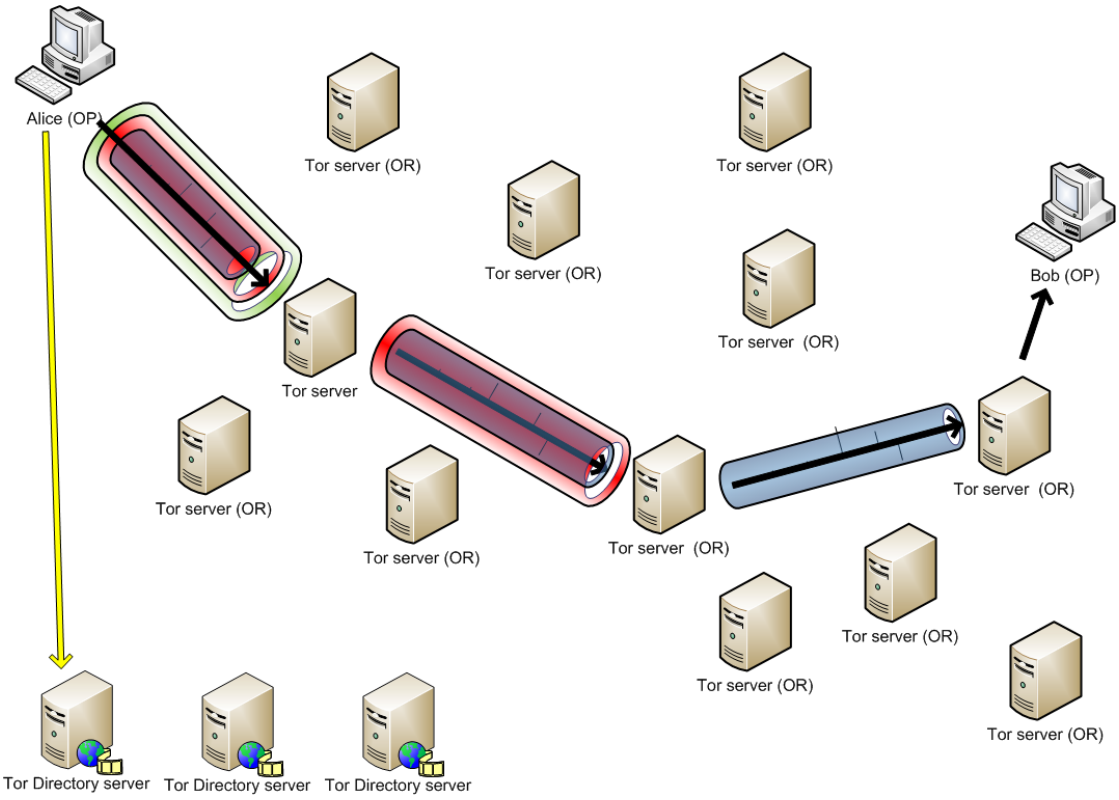

Figure 9: Tor Circuit Creation

To construct a circuit the OP incrementally negotiates a symmetric key with each OR on the circuit. The steps to make a two hop circuit can be seen on figure 10. Alice first selects random OR's to create the circuit over. The default is three hops, but two is used in this example. She starts a TLS connection to the first OR. Alice sends a create cell over the TLS connection which is protected with the public key of OR1. Cell types are explained in detail the next section (2.4.3). The created cell contains the first half of the Diffie-Hellman handshake. OR1 responds with the second part of the Diffie-Helman handshake and a secure hash of the symmetric key. They now share a secret tunnel to exchange information.

To extend the circuit further Alice sends a relay extend cell to OR1 with the address of the next OR (OR2) and the encrypted key for this connection. The message content is encrypted with OR2's public key an cannot be viewed by OR1. OR1 is now responsible for making a TLS connection to OR2. OR1 copies the half-handshake into a create cell, encrypts with OR2's public key and passes it to OR2 to extend the circuit. A created cell is received from OR2 with the second part the Diffie-Helman key and secure hash of the symmetric key. OR1 wraps it into a relay extended cell and passes it back to Alice. The circuit is now extended, Alice and OR2 share a common key. The circuit can again be extended by repeating the steps above. Alice has a

working circuit to OR2 and shares keys with each OR on the circuit. The default circuit length is three to be sure that exit node doesn't communicate directly with the entry node[22].

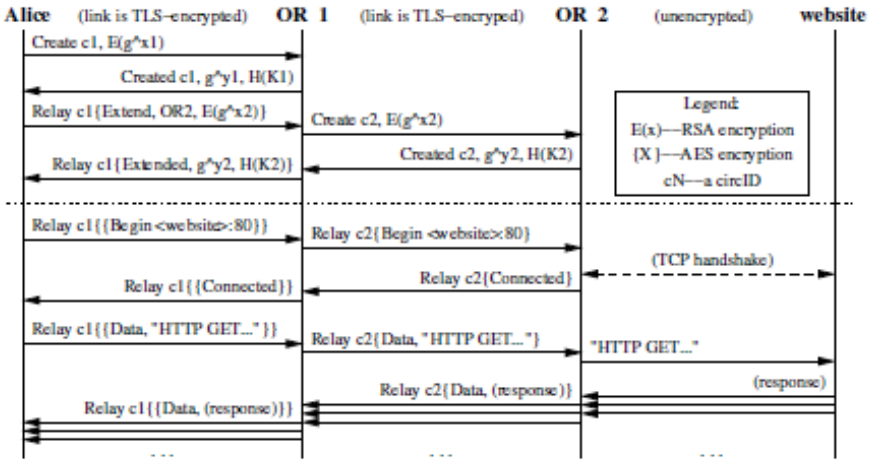

Figure 10: Tor circuit [22]

To use the circuit Alice has created an application asks the OP (via SOCKS or Privoxy) to make a connection. The OP then chooses the newest open circuit and a OR to serve as the exit node. To start sending data Alice sends a relay begin cell to the exit node using a random streamID. When the exit node has made a connection to the service Alice requested it sends a relay connected to her. The OP then notifies the client proxy of the success and the OP can accept data from Alice's application. The data is sent with a relay data cell and received with a relay data cell. When Alice's application is finished using the TCP stream she sends a relay end cell to indicate to close the stream. The exit node replies with the same cell. If the connection is closed abnormally the adjacent node sends a relay teardown cell.

To close the circuit Alice can send a destroy control cell. The destroy packet is decrypted at each node in the circuit and the connection is torn down. The node then passes the destroy cell down the circuit. The relay truncate cell can be used to destroy the circuit from a OR and forward. The receiving OR then responds with a "relay truncated".

To support different protocols Tor requires the use of a proxy. The SOCKS proxy and Privoxy has been used which support most protocols and has the required features. These two support almost all TCP-based protocols[22].

2.4.3 Tor Cells

Each packet of data in the Tor network has a fixed cell size of 512 bytes with header and a payload. This gives two Tor packets per TCP packet in a typical network with a maximum transmission unit of 1500 bytes [23]. The cell size is the only delimiter for traffic in Tor. The header includes a circuit identifier (CircID) which specifies which circuit the cell refers to. Many circuits can be multiplexed over a single TLS connection. The cell can be a control cell or relay cell as

seen on figure 11. A control cell can carry commands such as padding, create, created and destroy. These are interpreted by the OR that receives them [22]. The relay cells carries end-to-end stream data and has an additional header containing a streamID, end-to-end checksum, length of the relay payload and a relay command. The relay commands are[22]:

- relay data - for data flowing down the stream.
- relay begin -to open a stream.
- relay end - to close a stream cleanly.
- relay tear down -to close a broken stream.
- relay connected - a relay begin has succeeded.
- relay extend - to extend the circuit by one hop.
- relay extended - a circuit is extended by one hop.
- relay truncate - tear down this part of the circuit.
- relay truncated - relay truncate acknowledgment.
- relay sendme -is used for congestion control.
- relay drop - to implement long-range dummies.

Figure 10 depicts the Tor packet header.

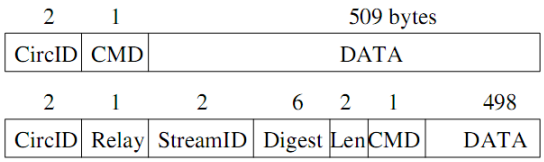

Figure 11: Tor Packet Header [22]

2.4.4 Tor design choices

Lets examine why Tor uses three hops by default. A circuit chooses hops that are spread over geographical, juridical and political regions. This makes it almost unfeasible for an adversary to compromise all servers along a circuit. Three hops is the minimum for anonymous operation. The first OR can see the sender and second OR, the second OR sees each end of the circuit but not who is communicating. The third node can see the second node and the destination. Each relay is only aware of its adjacent relays through a circuit.

Creating circuits in Tor can be time consuming because public-key cryptography is CPU intensive and network latency is slow[22]. To mitigate network latency each circuit is shared by many TCP streams and circuits are built preemptively. The circuits are created periodically, old circuits are expired and torn down.

If a Tor instance uses a corrupted first node the attacker can see the identity of the user by their IP address. But the attacker cannot see what user is doing online because it is encrypted. It

is possible to prevent this by using a trusted first node that you run yourself for example. If the last node in circuit is corrupted the attacker can tell that someone is accessing a service. But not who is requesting it, just that someone is using Tor to access a site.

Even though it is possible to have a longer circuit it would increase overhead and degrade performance further. It could make the anonymity better by adding even more layers of encryption. But the performance penalty would be larger and undesirable.

Tor makes use of the perfect forward secrecy principle where each hop down the circuit is protected with encryption from initiator [5]. This can be compared the layers of an onion, which is Tor's logo. This approach is reliable because the initiator knows when a hop fails and compromised hosts cannot decrypt traffic once the keys have been deleted[22]. Tor rotates keys regularly and makes it is pointless to compromise a Tor server in order to decrypt recorded traffic. Tor uses Diffie-Hellman with RSA cryptography as the public key infrastructure to obtain perfect forward secrecy [22]. If Tor used only RSA cryptography and the private key had been compromised all previous communication as well as the future traffic would been compromised because of this static key. By using Diffie-Hellman with RSA perfect forward secrecy is achieved and gives Tor the property it needs. There are a limited set of cipher suites to choose from which has this property.

2.4.5 Tor Bridges

To improve blocking resistance Tor introduced Bridges [7]. The intended use is when access to the directory servers and all known OR's are blocked. The bridges were invented To serve users which experience this. A bridge is an unlisted OR which delivers the same functionality as the OR's listed in the directory servers. But since they are unlisted an adversary will have problems keeping track of all bridges and hence not be able to block them. This improves the blocking resistance but makes the start up of Tor slower. The bridges announce their presence over an encrypted connection so an adversary cannot collect bridges by observing them. A Tor client can request bridges but will only receive three to prevent the adversary from enumerating many bridges. The bridge technology is still in development [24].

2.4.6 Tor Hidden services

Tor also has the ability to run hidden services. This gives the publishers ability to hide the servers location and running services anonymously. When a user wants to access a hidden service the user and service meet at a Tor server called an "rendezvous point". This server enables the origin of the service remain hidden from the user. Hidden services are not relevant to this thesis and further explanation therefore omitted.

3 Related Work

The next chapter describes some general traffic analysis attacks first. We then explain attacks regarding TLS and finally, attacks specific to Tor.

3.1 Traffic analysis

Traffic analysis attacks is when a side-channel is used to reveal information. Something is exploited in a way the developer didn't intend. Traffic analysis uses unencrypted fields in the header, packet size, latency, round trip times, statistics et cetera to reveal information about the communicating parties. Traffic analysis does not try to recover the key in order to decrypt the messages. The amount of information deduced from traffic analysis can be astonishing and the field of research has grown. Privacy is often compromised by traffic analysis as much of the research in this report describes.

There are obviously differences in the traffic of Tor and TLS in general. There have been several papers on these differences and how Tor could even them out[25]. Timing and latency are important metrics in traffic analysis as we shall see later. Timing and latency is not the only metrics in traffic analysis. For instance packet counting and volume could used to identify Tor. If a TLS session has a very high packet count it could indicate Tor usage because Tor generates many packets. Packet size is fixed to 512 bytes by Tor and could also be a metric to indicate Tor usage[6].

An interesting talk was given on Black Hat Las Vegas in 2007 where researchers had made a protocol classification tool called PISA. The tool would observe the network traffic and based on several metrics guess the protocol used in the encrypted traffic[26]. The tool used ten metrics, such as packet size, packet count, timing, traffic volume differences and tried to identify the protocol used. The tool was successful in identifying Skype, NetBios, Voice data and other traffic types. This show that is possible to create a tool which could identify Tor and degrade the blocking resistance of Tor. Unfortunately the source code is no longer available to the public.

Murdoch and Zielinski extended a traffic analysis attack to anonymity networks in 2007 [27]. The level of anonymity depends on geographically and politically spreading the traffic stream. This is done to maximize legal and cooperation problems for an attacker. They showed that ISP's often connect to many other ISP's in a single location called an Internet exchange (IX). The research shows that it is possible to conduct traffic analysis of low latency anonymity networks with sampled traffic from these IX's. This means an attacker can use sampled traffic and doesn't need to process the huge amount of traffic from the IX.

The amount of creativity and innovation is traditionally the only limit when doing traffic analysis. There are many clever attacks which reveal astonishing things. Some are general to encrypted communication and some are specific to TLS. Fu et al [28] uses round trip times from ping packets to recover information about the payload traffic rate. By constantly sending ping packets to the target and measure the performance a baseline is established. When the

performance reduces one can assume that that the target is communicating. This could reveal that something is going on between the two communicating parties. Increased chatter often implies that something is about to happen or has happened. It shows that a simple ping packet can lead to partial loss of confidentiality.

A more recent research paper from 2010 takes advantage of the development towards software-as-a-service [29]. Web applications are becoming popular, even in governmental departments and this could give an attacker access to privacy data. Chen et al investigated several high-profile web applications and found that several of them leaked information about the user if an attacker had access to the encrypted traffic. Information about healthcare, income and investments could be deduced just by observing the encrypted traffic. Web applications use a very stateful communication combined with a low entropy which makes them vulnerable to traffic analysis. In this paper a web application with many states, forms and options were studied. The authors noticed what each choice in the web application generated in traffic flow. This made them able to deduce information.

3.2 TLS attacks

3.2.1 Threat model

There are many ways to attack TLS and the threat model is complicated. To organize and catalog the threats a mind map has been developed by SSL Labs [30]. Some of the threats are dated or not of an technical matter, but still exists. The major threats are against authentication as we will show later. The figure 12 is presented below and gives a very organized view of the threats that TLS has to address.

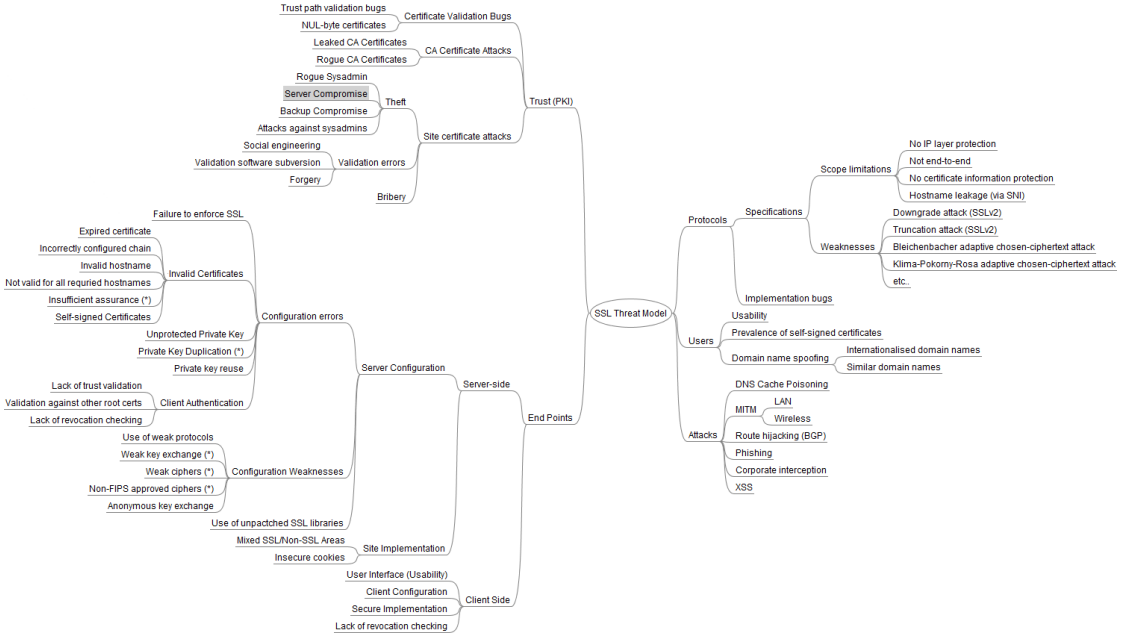

Figure 12: TLS Threat Model [30]

TLS provides confidentiality over an open communication channel. Cryptoanalysis have become harder since SSL 3.0, requiring a great effort and determination to break them [31]. Several researchers have hammered and tested the ciphers to discover weaknesses and flaws to make them as secure as possible. Since cryptoanalysis has become very demanding the research has evolved to more easier targets such as implementation flaws and traffic analysis [31]. We choose to focus on the attacks which we think are relevant to this report and present them. This is because the threat model of SSL is very large and complicated as we can see on figure 12. We focus on traffic analysis rather than implementation and design issues.

3.2.2 Certificate related attacks

The most recent attack against TLS exploits a MD5 collision and rogue CA generation. This was discovered by Sotirov et al [32]. Because some CA's still use the weak MD5 hash to generate certificates the researchers were available to create rouge CA certificate which was trusted by all common web browsers. This makes an attacker able to impersonate any website on the Internet with the help of redirecting requests and this rouge certificate.

The x.509 certificate standard has been around since 1988 and is currently at revision three [33]. Because x.509 is old it has many revisions and extensions. There are many x.509 extensions which adds necessary features. They are used frequently as we will show later in the report. These extensions are appended to the certificate between the public key and the signature. One common extension is the Basic Constraint field. This field is used to indicate that this is a root CA certificate and hence stop the further validation up the certificate chain [34]. It also tells that this certificate may not be used to create further CA certificates. NULL byte certificate attacks by Moxie Marlinspike was discovered in 2009 [35]. This made an attacker able to create a certificate which was valid for all sites on the Internet with usage of the * character as the common name in a certificate. It was possible to make wildcard certificate which was valid for any site on the Internet and combined with a Man-in-the-Middle attack could compromise a user completely. Moxie Marlinspike published an attack tool which exploits the fact that many browsers do not check the Basic Constraint field on intermediate nodes and that it was possible create wild card certificate [35].

The authentication gap vulnerability was discovered by Marsh Ray and Steve Dipensa in 2009 [36]. It exploits the way a client certificate induce a renegotiation. This in turn could make an attacker able to inject himself in an authenticated TLS session. When executed the vulnerability partially disables the padlock icon and the user must usually click pass a warning. Websites which uses client certificates can be vulnerable. This concerns smart card deployments in most cases.

3.2.3 Traffic analysis attacks

Fingerprinting websites by using traffic analysis is also possible [37]. The attack can reveal if a user is viewing a certain webpage. Certain countries denies citizens to view nongovernmental information and punishes this act [4]. This attack generate fingerprints on websites from the size of each TLS connection. When a user visits a website there are several connections to serve every picture, animation and style sheet et cetera. Each of these connections have unique download size. So based on the number of connections and size of each connection a unique fingerprint

can be built. This fingerprint is then correlated to the traffic to the encrypted traffic stream. This attack suffers from the fact that websites are becoming more dynamic and the fingerprints must be updated accordingly.

Another traffic analysis attack was described by Danezis in 2009[38]. A website tries to protect confidentiality by using HTTP over TLS. The attack used the length of resources to identify if an item was viewed. The content was encrypted but if the attacker accesses the data length can be noticed and used as an identifier for a specific page. The attack collected the length of an encrypted GET request to a certain secure website and correlated this information when observing traffic elsewhere to see if any traffic matched. The research focus further on how unique the resource length is as an identifier. The attack can also be conducted on log files, if the server logs the requests. This could be very damaging if dissident website is ordered to reveal log files from the server. Dissident identities could be revealed against their will.

An attack specific to SSL was implemented by Cheng and Avnur in 1998[39]. They used the HTML size and object size to identify if traffic correlated to a certain website. They indexed several websites and created a large database to test the sniffed traffic against. The attack showed that websites have unique fingerprints and is was possible to reveal if a user was viewing a specific website. The mitigation proposed to use random padding in order to mangle the fingerprint or use an anonmyzing service. This would of course impact the performance and experience negatively.

3.2.4 Cipher suite fingerprinting

Since TLS is just a standard, developers could implement it differently from each other. The RFC states how the protocol works and functions but some choices are left to the developer because it would be impossible to specify everything. OpenSSL is widely used and implements TLS. It is likely that OpenSSL implements TLS differently from Microsoft. SSL Labs has done some efforts to fingerprint such differences [30]. Because there are so many cipher suites to choose from the Client hello packet in a TLS handshake will look different between implementations. SSL Labs has developed an addon to Apache Web server which has a fingerprint database and can guess the HTTPS client which is used.

Foundstone has also made an effort to test SSL servers. They have made an application called SSLDigger which testes the available ciphers a SSL server supports. It also reports the version, validity of the certificate, key length et cetera [40]. The tool is very useful for reconnaissance of SSL servers and to fingerprint them.

3.3 Tor attacks

3.3.1 Tor Threat Model

There exists several attacks against Tor. Some will not be addressed by the Tor developers because it is in the nature of low latency anonymity systems [22]. This includes traffic analysis attacks such as end-to-end timing attacks [41], latency [42], denial of service attack on the directory servers [22] and hostile Onion Routers[22]. The developers have made some of these attacks more difficult, but Tor is still vulnerable by design.

The Tor threat model assumes a non-global active adversary [22]. This limitation is a trade

off between performance and anonymity. The Tor design assumes an limited adversary that can control a only small segment of the network. For instance compromise some fraction of the onion routers and operate a limited set of own onion routers [22]. By compromise we mean observing, modify, generate, delete or delay traffic. Tor depends on the assumption that nodes along a circuit do not collude [23]. If nodes collude they can correlate when a client uses a circuit together with a exit node making a request and reveal the user by end-to-end timing attack.

Trade off between anonymity and performance is a very important issue. If Tor becomes to slow it will be hard to attract users and claim to be a low-latency service. It is therefore important that any changes Tor doesn't degrade performance to an extent which drives users away.

3.3.2 Attacks against Tor

The end-to-end attack can be preformed by a global adversary. A global adversary can view traffic over the entire network and makes it possible to observe the first node and the last node on a circuit. When traffic emits from the first node the attacker looks for a node where traffic exits. The exit node must emit traffic after it entered the network plus some latency. The adversary can correlate when traffic enters the first node and leaving the last node by knowing this latency. When the attacker knows the first node and exit node he/she can know who is doing what online. This attack can be effectively mitigated by running your own trusted onion router and always use this as a first hop. Latency is also a threat to Tor because an attacker could measure the latency over a traffic stream on the first hop of an circuit and correlate to other nodes in the Tor network. This requires a global adversary as well.

There have been a few attacks on Tor over the years. For example Murdoch and Danezis [42] has showed in 2005 that it is possible to enumerate nodes which are involved in a transaction, called a congestion attack. They could identify which relays that were on a Tor users circuit. First they build circuits through every Tor relay, one at the time, and at the same time introducing congestion on the OR. This is done by flooding relays one by one and at the same time observing latency. When the observed latency drops one can assume that the a relay in the circuit is hit by the flooding. Evans et al later showed in 2009 that this attack is no longer practical since Tor has 1500 heavily loaded relays[43]. The attack does not scale well because an attacker must have a tremendous bandwidth to measure enough relays and false positives are generated by other users. In stead an other attack was described that used long circuits that loop back to themselves enhance the needed bandwidth.

Another attack by Hopper in 2007 [44] proved that it is possible to use latency to confirm that two connections from the same Tor exit node are using the same circuit. This is done by two colluding websites which measure latency on the local segment. The same article also describes an attack where a website can gain several bits of information about a client each time the site is visited. Hopper et al shows that latency can reveal the network location in less than 50 visits. Even though these attacks are resource demanding they are interesting because it shows that traffic analysis can be done on Tor.

Blocking Tor can be quite trivial and the developers have tried to address this [7]. It is possible to block Tor by blocking the directory authorities, blocking all the server IP addresses in the directory and to fingerprint Tor from the network traffic. This report concerns the latter problem.

This could be searching for strings in TCP packets, intercepting DNS requests to give bogus answers or traffic analysis. Some efforts have been implemented to mitigate this. By using a bridge the Tor client which is blocked by some country or firewall access the Tor network. A bridge is a client that volunteers to help censored users access the Tor network by serving an unlisted first-hop relay. To use a bridge the client must obtain a directory record of a bridge by using an out of band channel. Lately Tor has implemented the option to ask for bridges over the Tor network. This could lead to some attacks where an attacker enumerates many bridges and the adversary can then block all bridges [45].

Øverlier et al [46] showed that it is possible to locate the hidden servers in 2006. Tor provides this feature to let users host services without exposing geographical location. The attack requires a malicious client and a Tor node. When many connections are made to the hidden service the circuit and eventually the malicious Tor node is used as an entry node to the rendezvous point. The malicious Tor node can use packet counting to discover when this happens. The attack has been mitigated by introducing guard nodes which every circuit starts with. The Tor client chooses three guard nodes which serves as an entry node for all circuits the client creates. This prevents enumerating the node which connects the hidden server. As a side note the hidden service also protects against distributed DoS attacks[22]. Hidden services is not the main interest in this thesis and we choose to not explore the attack further.

An other attack against hidden services was proposes by Murdoch in 2006 [47]. The attack is based on clock skew and timestamps. By observing timestamps Murdoch was able to detect if node had a heavy load. This could be used as a side-channel to identify nodes in a circuit. This show that the side-channels can be hard to spot and pretty obscure.

The literature on countermeasures against traffic analysis is not complete. One has to evaluate padding, delaying, cover traffic and such for mitigation. Obviously will this degrade the performance of Tor. This is not desirable since Tor already cause a delay on the performance of the Internet connection. Tor users in less developed countries frequently have modem lines which has a very limited bandwidth. Most traffic analysis counter measures will degrade performance because they introduce overhead.

4 Choice of Methods

In this project we try to describe ways Tor reveals its presence by pointing out differences from normal TLS traffic. This requires us to describe normal TLS traffic as well. Because of this difference it is possible to filter the directory servers or onion relays and deny access. By using bridges, which doesn't advertise their presence the user could still access the Tor network if servers are blocked. We try to identify Tor just by investigating the induced TLS traffic. This could be used by an adversary as a way to block Tor. We focus on this method because it will be valid even if bridges are used.

If Tor is trivial to block it would be to easy for a larger force to prevent anonymity services and preventing information from reaching the public. For example it's crucial to use Tor for dissidents because they risk prosecution from the government if they are exposed. If a government blocks Tor a dissident can be forced to expose his/her identity by using other means of communication. Or even worse use his/her the true identity on the Internet which could have serious consequences. This is the threat model we use as our starting point when experimenting with ways to reveal Tor. If Tor could pass as ordinary and legitimate traffic it would we very hard to block, provided that bridges are used. By increasing censorship resistance Tor can be used in countries with strong censorship.

4.1 Methodology

4.1.1 Reliability, validity and metrics

Reliability and validity are important aspects to consider when choosing metrics and conducting research. Reliability describes if the measurements we are doing are correctly executed[48]. An important factor for increasing reliability can be to eliminate random errors. We try to protect ourself against random errors by doing experiments several times, remove variables and assessing the process. In this experiment we carefully choose each metric in order to give reliability to the results in our report.

Validity can be said to be the accuracy, meaningfulness and creditability of the research[49]. Putting it an other way, can we be sure that we are measuring what we think we are measuring? A simple example can be if we are trying to count vehicles passing a bridge. We choose to count all vehicles with four wheels. This would give a errors since we exclude some buses and motorcycles in our survey. This leads to failure because we are counting incorrectly. We believe this research has validity because we have sufficient controls to conclude from the results. To give external validity we have tested both in a closed environment and in a real Internet situation.

We are trying to find different metrics to be able to identify Tor usage. There are many metrics to choose from and we need to define some criteria to evaluate these against each other. The information assurance research community has done several efforts to come up with metrics to measure security. To better understand metrics they have categorized and quantized security metrics [50].

Metrics can be divided into categories [51]:

- Technical: These metrics measure or compare technical objects. Such as number of packets, round trip time, cipher algorithm. This is a quantifiable metric which is relatively easy to measure.
- Organizational: These metrics measure on the programs and processes within an organization. Examples are policy management, support to programs et cetera.
- Operational: This is a metric for measuring the properties of a system in operation, its operating properties and measures specific to the environment.

We use Technical metrics only in this thesis. Inspired by the information assurance metric research we choose metrics with maximum reliability and validity and leave out poor candidates. Metrics which could indicate Tor usage must be useful as indicator in a filter with intention of blocking Tor. Further the metric must be able to exclude or include a reasonable amount of elements in the set. If the indicator exclude or include very few elements it will be inefficient as a test in a blocking device. Or it could give to many false positives or false negatives to contribute to the performance.

4.1.2 Initial test

The chosen method in this project is qualitative in the quest to compare the signature and behavior of Tor TLS traffic and other TLS traffic. Our strategy was to use as simple means as possible to identify differences. This was because the filter must be very efficient to handle large traffic volumes according to our threat model. For a filter to be efficient it cannot calculate statistics and use CPU intensive computations. Hence we investigated the handshake and unencrypted parts of the communications. As a starting point in our research we inspected the traffic which a Tor instance generates in the startup of it is lifetime. We investigated this traffic and took notice of oddities and implementation choices. This was our initial test to make a basis for further research. The initial test put us in a direction where we had to define metrics for normal TLS traffic and behavior. From this initial test we could choose test metrics both in live Internet traffic and in our test environment.

4.1.3 TLS baseline test

This test took metrics found in the initial test as a starting point. From the chosen metrics we tested if TLS traffic generated on the Internet had the same properties as Tor in our initial test. We tested 40 different websites and recorded the results. We connected to the TLS enabled sites, observed the traffic and noted the values of the parameters. This analysis gave an impression of the characteristics that TLS traffic generates.

4.1.4 TLS cipher suite selection

To have more control and to test other services than HTTPS we set up a test environment with different TLS services. The test environment is described in the next section. In the test environment we could test how the different services responded to different input. This added validity

to the research and we could test how the system reacted in a closed environment.

4.2 Test environment and tools

In order to investigate what other TLS services generate in terms of traffic we created a test environment. Here we could inspect TLS traffic in a closed environment without tampering from other sources. In the experiment we set up different TLS services on different test machines. To save time and money we used VMware workstation 7.0 to virtualize the machines and network [52]. The experiment was done on a Laptop with an Intel Core 2 Duo T9400 Processor 2.53GHz and 4GB RAM. The operating system used was Ubuntu Server 9.10, Linux based, [53] because it is free and well documented. In the Windows environment we used Windows 2008 R2 Server Standard with Exchange 2010 as the mail server and Internet Information Server 7 as the webserver [54][55][56]. The purpose of this environment was to compare different TLS services on different platforms against Tor's TLS traffic. We choose two of the most common operating systems to have some diversity. We set up the virtual machines as shown in figure 13.

- Tor 0.2.2.1-alpha on Ubuntu Server 9.10
- Apache 2.2 with Openssl 0.9.8g on Ubuntu Server 9.10
- Postfix 2.6.5 with Openssl 0.9.8g on Ubuntu Server 9.10
- Courier-imap4 4.4.1 with Openssl 0.9.8g on Ubuntu Server 9.10
- Courier-pop3 0.61.2 with Openssl 0.9.8g on Ubuntu Server 9.10
- Windows 2008 R2 Server with Exchange 2010 and IIS 7.

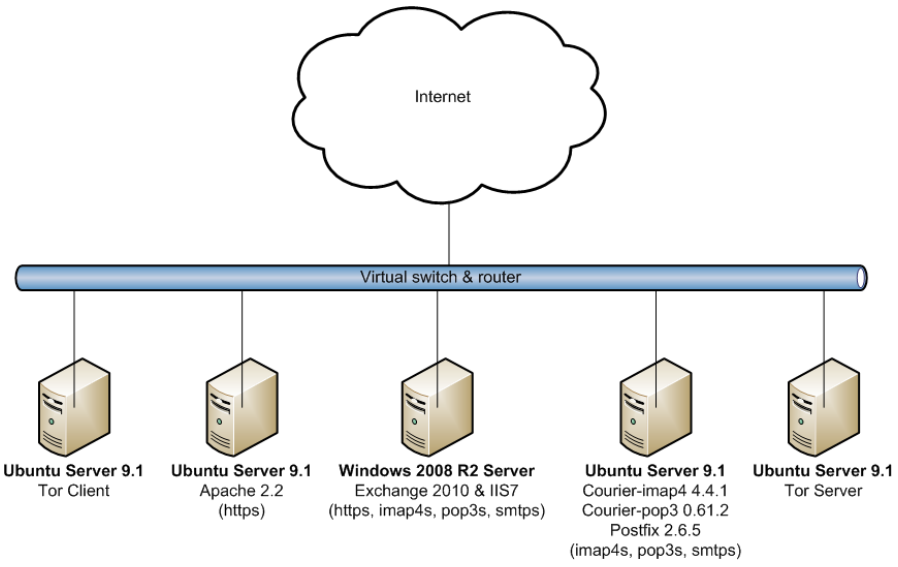

Figure 13: Test setup

Tor comes as bundle with all the tools needed to set it up. To have the latest version of Tor we used the alpha repository. APT, which is the package manager in Ubuntu, conveniently sets up Tor, Polipo and Vidalia. Polipo is a SOCKS proxy and works between Tor, the Internet and the internal TCP stack of the operating system. The Polipo configuration file was downloaded from the Torproject webserver to set up the SOCKS proxy. Vidalia is the GUI frontend to Tor and asks questions to the user in order to set up Tor the first time [57]. It can even setup firewalls if it has uPnP enabled. In Vidalia it is possible to tweak the configuration easily by using a GUI. Vidalia is designed to be easy to use but has severe restrictions compared to the commandline configuration possibilities in Tor. We used Vidalia to single out the different streams and circuits when capturing traffic in Tor. The control port is a command line interface accessible over telnet on port 9051 with a password. The control port is only enabled on the local interface by default. It gives the ability control almost every behavior of Tor such as circuit creation. The control port can for example be used to create circuits of to certain nodes and circuits with custom length.

To set up Apache, Postfix, Courier-imap4 and Courier-pop3 we used the documentation provided by the Ubuntu community [58]. The configuration is mainly default and only minor changes was necessary to make the services work. We deliberately did as little configuration as possible to make the behavior of the services as little distinct as possible. OpenSSL provides a set of tools to create a certificate. The certificate we used was created by the author and thus self-signed. The steps to a make self-signed certificate is also documented by the Ubuntu community [58]. When making a certificate it is important to set the common name correctly. The common name must match the URL of the service. We only tested the newest stable version of the services, no other versions were tested.

To capture and examine traffic we used Tcpdump 4.0.0 and Wireshark 1.2.5 [59][60]. These tools gave us the ability to capture and examine the network traffic. To test the servers we used Mozilla Thunderbird and Mozilla Firefox for clients. We also tested with Internet Explorer where applicable to get diversity in the experiment. The Mozilla software has extensive options for tweaking different options. Especially we were interested in customizing the available cipher suites given by the client. This was easily accessible by the graphical interface of both Thunderbird and Firefox. To test the Tor server an other approach had to be used. Here we used Scapy which is a packet crafter API in Python. Then we could create a raw Client hello and test the TLS handling in Tor [61]. The script we developed is included in appendix C.

The traffic capture was conducted one step at a the time. First we set up the service and made sure everything was prepared to make process as fast as possible to limit background traffic. We stated the traffic capture and induced traffic. The capture was then stopped, the traffic was examined and results was recorded. Each test was done two times to limit any random errors.

To test Tor traffic we used the Control port to start and shutdown circuits. We could then easily examine the TLS traffic without any unnecessary traffic.

5 Results

First we present the characteristics of TLS traffic and move on to explaining the characteristics of Tor TLS traffic. Then we elaborate on the differences and evaluate their potential as an identifier. Finally we implement some of the identifiers and propose a new layer to Tor.

5.1 TLS Traffic Characteristics

There are many options to choose from when implementing TLS. TLS has many different versions to begin with. But only SSL 3.0 and up are considered secure. SSL 2.0 has several known weaknesses [13]. The author has the impression that most implementations uses TLS 1.0 and up.

One obvious and crude characteristic for TLS traffic is the TCP port used in the connection. The port would in most cases imply the service used. The default port for webtraffic over TLS is TCP port 443, POP3S has port 995, SMPTS has port 465 while port 993 is used by IMAP4S. These are the default ports for TLS traffic and will often identify the traffic type. But it is up to the administrator of the service to choose which port to bind to. This could be simple way to obfuscate the service.

Traveling further up the layers, the handshake protocol has several options, especially specifying the available cipher suites. A typical Client hello packet offers 11-33 cipher suites when negotiating a TLS connection, by our observation. Chrome announces 11 ciphers, Firefox announces 33 ciphers, Internet Explorer announces 12 ciphers, Opera announces 27 ciphers and Tor announces 28 ciphers. The order of the list of available cipher suites can also be different between Client hello packets.

The Client hello can contain extensions, which is good because several extensions can contribute to better security [62]. Extensions could act as a unique identifier because they are numerous and have many options. The extensions can provide extra information if the IP address hosts several webpages or elliptic curve crypto information for example. Firefox sends four extensions by default which could be used as an identifier in some circumstances. The reply to a Client hello is the Server Hello which specifies which cipher the communication should use. The client certificate is usually sent after the Server Hello. We have not examined Client certificates in this project because it is not used by Tor.

Following a new TLS negotiation, (not renegotiation) the next step is to exchange server certificates. The certificates can reveal a lot of information. Such as location, email, contact person and of course the public key [33]. x.509 is very old in digital standards and has many extensions to provide functionality and compability. Since there are many extensions to choose from and many ways to combine it could lead to unique signatures which are easy to check for.

When we examined the TLS negotiation of POP3S, SMTPS, IMAPS and HTTPS, described later in the report, we saw that a typical handshake uses four packets. This is also used by Tor as shown in the TLS Handshake figure 15. First the client send a Client hello packet. The server

responds with a packet with several TLS messages embedded: Server Hello, Certificate, Server Key Exchange and Server Hello Done. This packet contains all parameters for the client to start an encrypted connection to the server. So now the client responds with a Client Key Exchange together with a Change Cipher Spec. The packet also contains a Client hello Done, but this is encrypted. This indicates to the server that all traffic from now on will be encrypted. The server responds with a Change Cipher Spec also indicating that all traffic from now on will be encrypted.

Traffic volume and timing can reveal information about the encrypted traffic as well. When the user is browsing (HTTP) the client typically sends a short request and the server responds with a big chunk of data. This is typically for browsing the web and clearly different from services such as VoIP. VoIP traffic typically flows in bursts with approximately equally traffic volume in both directions. The traffic flow directions depends on the persons talking. The traffic volume can reveal information regardless of encryption [29]. For instance one can see the person sending more data and assume based on this observation.

5.1.1 TLS Certificates

TLS and Tor relay heavily on certificates which has potential for traffic analysis. On the Internet a certificate is used to authenticate that a public key belongs to an identity. This is obtained by using a digital signature to sign the certificate which contain the public key and the identity. The certificate usually contains additional information but this is irrelevant to the signing process. The identity will verify that a bank, email provider or a person is bound to a public key included in the certificate. A certificate is usually bound to an identity with a humanly readable name called the Common Name, such as "www.google.com". The Common Name (CN) field in the certificate must correspond to the entity which is being authenticated, commonly a URI. The root certificate authority (CA) is the highest level in a chain of trust scheme and uses a self-signed certificate. Normally a browser, which uses certificates extensively, ships with a number of CA certificates by default to speed up the validation. The certificate can contain lot of useful information such as issuer, address, limitations of the certificate et cetera.

A certificate can only be valid a certain time. A certificate is typically issued for several years as one can see in Appendix B. The x.509 certificates has two fields which describe the time frame the certificate is valid. This is typically two years but up to the creator of the certificate [63].

The certificate chain is a list where each certificate is signed with key of the entity over it in the hierarchy [32]. So when a certificate chain is received from a server it must be verified by the recipient. This dept of this chain can vary between sites and hence it could used as an identifier.

5.2 TLS Baseline

In order to do our study of Tor's TLS traffic we must investigate typical TLS traffic from normal Internet network traffic. This will give us the foundation to claim that TLS traffic generated by Tor deviates from normal TLS traffic. To do this study we chose 40 websites that uses TLS over a variety of businesses such as banking, emailing, law enforcement et cetera. We choose HTTPS as the service because Tor appears to be trying to mimic this type of service [7]. For instance many run their Tor server over port 443 and Tor uses the same cipher suite list in the Client hello as Firefox [64]. From our initial study we have established that there are some differences between

Tor's TLS traffic and the traffic generated in our test environment. We used these metrics to measure if the live traffic has the same characteristic. We only used metrics which can be found in the handshake before the encryption is commenced. We looked for metrics that would give few false positives or false negatives.

Table 1: TLS charateristics

Site	Version	DH support	Packet count	Failure	Certificates	Cert validity	Common Name	Certificate extensions
https://Tor server	1.0	Yes	4	Alert	1	2h	www.<Random>.net	0
https://torproject.org	1.0	Yes	4	Alert	1	1y 2d	*.torproject.org	5
https://gmail.com	1.0	No	5	Alert	2	1y	www.google.com	4
https://bankofamerica.com	1.0	No	4	Alert	3	1y 10d	www.bankofamerica.com	8
https://facebook.com	1.0	Yes	5	Alert	3	2y 4d	www.facebook.com	8
https://amazon.com	1.0	No	4	RST	2	1y	www.amazon.com	7
https://wordpress.com	1.0	Yes	5	Alert	3	4y 11m 19d	*.wordpress.no	9
https://continental.com	1.0	No	5	Alert	3	2y 4d	www.continental.com	8
https://ieee.org	1.0	No	4	Alert	2	1y	www.ieee.org	3
https://verisign.com	1.0	Yes	5	Alert	3	2y	www.versign.com	8
https://politi.no	1.0	No	5	Alert	3	1y	www.politi.no	7
https://www.dnbnor.no	1.0	No	4	Alert	3	2y 1d	www.dnbnor.no	8
https://us.hsbc.com	1.0	Yes	5	Alert	3	1y	us.hsbc.com	9
https://twitter.com	1.0	No	5	Alert	3	1y	twitter.com	10
https://www.play.com	1.0	Yes	5	Alert	3	2y 1m	www.play.com	8
https://blackhat.com	1.0	Yes	5	Alert	4	4y 1d	*.blackhat.com	9
https://wordpress.com	1.0	Yes	4	Alert	3	4y 11m 11d	*.wordpress.com	9
https://hig.no	SSL 3.0	Yes	4	Alert	4	3y	www.hig.no	9
https://www.salesforce.com	1.0	Yes	4	FIN	1	2y 17d	login.salesforce.com	7
https://www.vmware.com	SSL3.0	No	4	Alert	3	1y	www.vmware.com	3
https://www.thawte.com	1.0	Yes	4	Alert	3	2y 1d	www.thatwe.com	7
https://www.ietf.org	1.0	Yes	5	Alert	3	1y 10d	*.ietf.org	8
https://www.java.com	1.0	No	5	Alert	3	4y	www.java.com	8
https://www.mozilla.com	1.0	Yes	4	Alert	2	2y 2d	*.mozilla.com	6
https://www.blogger.com	SSL 3.0	No	5	Alert	3	1y	*.blogger.com	6
https://signup.live.com	1.0	Yes	4	FIN	4	1y	signup.live.com	9
https://accounts.craigslist.org	1.0	Yes	4	FIN	2	5y 2m 12d	accounts.craigslist.org	4
https://home.americanexpress.com	SSL 3.0	No	4	Alert	3	1y	home.americanexpress.com	3
https://www.sas.no	1.0	Yes	5	Alert	3	14y 11m 13d	www.sas.no	5
https://www-ssl.bestbuy.com	1.0	No	4	Alert	2	1y	www-ssl.bestbuy.com	8
https://secure-web33.secondlife.com	1.0	Yes	4	Alert	2	1y 2d	*.secondlife.com	5
https://store.apple.com	1.0	Yes	4	Alert	3	2y 1d	store.apple.com	10
https://members.bet365.com/	1.0	No	4	Alert	2	3y	members.bet365.com	4
https://www.nsb.no/	1.0	Yes	5	Alert	2	3y 1m 13d	www.nsb.no	7
https://ssl-no.hotels.com	1.0	No	4	Alert	2	5y 1d	*.hotels.com	5
https://www.fedex.com	1.0	No	4	Alert	4	1y	www.fedex.com	9
https://www.ssllabs.com/	1.0	Yes	4	Alert	3	2y	www.ssllabs.com	9
https://www.ehealthinsurance.com/	1.0	No	4	Alert	3	1y 15d	www.ehealthinsurance.com	7
https://login.salesforce.com/?locale=eu	1.0	Yes	5	FIN	3	2y 27d	login.salesforce.com	7
https://www.sap.com/index.epx?kNtBzmUK9zU=1	1.0	No	4	Alert	3	1y 7d	www.sap.com	9

Table 1 is the result of our experiment with creating a TLS baseline. As we can see almost all the tested sites support TLS version 1.0 just like the Tor server. The DH support field in table 1 measures if the site supports Diffie-Helman cipher suites. More precisely if the site is able to use any of these ciphers:

- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA

These four ciphers are supported by Tor and could indicate that this connection is carrying Tor traffic. From table 1 we can see many of the tested sites does not support these ciphers and in a traffic analysis we could exclude them as Tor traffic.

The packet count in table 1 tells us if the handshake is done over four or five packets which is the two ways of doing the handshake we observed during our research. This shows us that by measuring the packet count we can exclude many TLS streams as Tor traffic.

In the failure metric we tested what happens if the server and client doesn't have common cipher suite to communicate over. Some servers just rip down the connection on the TCP layer. The more common way to do it, as we can see from the table 1, is to send a TLS alert message with a handshake failure code.

The certificates field in table 1 show how long certificate chain we received from the server. This varies quite a bit because banks and other high security sites tends to have a long chain and less security demanding sites use a shorter one.

A certificate has a validity field which tells the time frame this certificate is valid. This field is used to describe how long the CA will maintain information about the certificate validity [33]. After the time has expired the CA will no longer track if a certificate is revoked or valid and the communication is considered insecure. All the certificates we observed has a validity of more than one year and several up to five years. The Tor server presents a certificate with a validity of 2 hours to the Tor client. This stand out from the rest of the certificates. The reason for this behavior can be to rotate the keys often and limit the time available for an attack.

The common name is an important part of the certificate because it describes which URI the certificate authenticates. This is typically the URI which the service resides on. A humanly readable name is almost always used as the URI and also the common name. DNS was invented to make it easier to remember URI's. When Tor uses a random string it is not conform with the rest of the services we tested [65]. There are many URI's on the Internet which is not humanly readable but we claim that not very many are bound to a certificate.

5.2.1 SSL survey

We realize that this experiment has a limited set of tested domains. It cannot be trusted fully without a larger set of tested SSL sites. Fortunately has Ivan Ristic at SSL Labs done a large study concerning the security of SSL sites and their certificates recently [30, 63]. The intention of the study was to investigate the security state of SSL on the Internet. As a starting point they tried to locate all SSL sites that VeriSign had registered at non-national domains such as .com,.net,.org etctera. They also included the 1 million most popular sites from Alexa's [66]. The survey started out with 119 million domain names, but very many had to be excluded because of no response or DNS failure. Further they had to exclude many domains due to invalid certificates and ended up with 867 361 domains to test. SSL Labs believes this is 25-50% of all SSL domains on the Internet. The original dataset has not been made public, only the graphs and tables.

We found that all domains used version 1.0 of the TLS protocol or SSL 3.0. This correlates with SSL Labs results which show that almost every site uses 1.0 as the highest supported version. SSL Labs has provided table 2.

Table 2: TLS versions from SSL Labs [63]

Protocol	Support	Best protocol
SSL v2.0	302,886	-
SSL v3.0	607,249	3,249
TLS v1.0	604,242	603,404
TLS v1.1	838	827
TLS v1.2	11	11

When it comes to cipher suite support we get approximately the same results as SSL Labs. Around half of all domains support Diffie-Helman which is the only key exchange algorithm Tor uses. SSL Labs found that 57% of all domains had support for the Diffie-Helman keyexchange. This is seen in table 3.

Table 3: TLS Key exchange support from SSL Labs [63]

Key exchange	Servers	Percentage
RSA	607,582	99.99%
DHE_RSA	348,557	57.36%
RSA_EXPORT	319,826	52.63%
RSA_EXPORT_1024	193,793	31.89%
DHE_RSA_EXPORT	176,258	29.00%

The certificate chain length was a parameter we measured in the TLS baseline test. This was also tested by SSL Labs. We found that the majority of certificates had a chain length of 2 or 3. SSL Labs found that the chain length was 2 levels deep in 44% of the certificates and 3 levels deep in 55% of the certificates [63]. We see that our results concur in this test.

Perhaps the most interesting result was the certificate validity survey. Tor uses a very short validity time frame for its certificates. According to our results the typical certificate validity is more than 1 year. This is supported by SSL Labs which came up with the graph on figure 14 on the distribution of certificate validity. On the graph we see that most certificates has a typical validity of 12, 24 and 36 months.

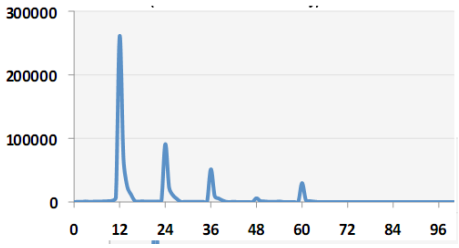

Figure 14: Certificate validity from SSL Labs[63]

5.3 Tor Traffic Characteristics

To examine the traffic to and from the Tor server we installed Tor on a Ubuntu Server 9.10 in VMware [53, 52]. Tor implements TLS 1.0 in version 0.2.2.1 which is slightly older than the newest version 1.2 of TLS [16]. Tor implements TLS fairly standardly when we examine the TLS Handshake. A standard handshake is done over four packets. An extract from a TLS handshake in Tor is seen on figure 15. The handshake will look different between implementations of the TLS protocol.

No. -	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.198.130	192.42.113.248	TCP	52918 > 9001 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 TSV=384667 TSER=0 WS=5
2	0.043982	192.42.113.248	192.168.198.130	TCP	9001 > 52918 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3	0.043998	192.168.198.130	192.42.113.248	TCP	52918 > 9001 [ACK] Seq=1 Ack=1 Win=5840 Len=0
4	0.044353	192.168.198.130	192.42.113.248	TLSv1	Client Hello
5	0.044457	192.42.113.248	192.168.198.130	TCP	9001 > 52918 [ACK] Seq=1 Ack=142 Win=64240 Len=0
6	0.110674	192.42.113.248	192.168.198.130	TLSv1	Server Hello, Certificate, Server Key Exchange, Server Hello Done
7	0.110700	192.168.198.130	192.42.113.248	TCP	52918 > 9001 [ACK] Seq=142 Ack=924 Win=7384 Len=0
8	0.134757	192.168.198.130	192.42.113.248	TLSv1	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
9	0.134892	192.42.113.248	192.168.198.130	TCP	9001 > 52918 [ACK] Seq=924 Ack=340 Win=64240 Len=0
10	0.202916	192.42.113.248	192.168.198.130	TLSv1	Change Cipher Spec, Encrypted Handshake Message
11	0.213579	192.168.198.130	192.42.113.248	TLSv1	Encrypted Handshake Message
12	0.213687	192.42.113.248	192.168.198.130	TCP	9001 > 52918 [ACK] Seq=983 Ack=457 Win=64240 Len=0
13	0.280794	192.42.113.248	192.168.198.130	TLSv1	Encrypted Handshake Message, Encrypted Handshake Message, Encrypted Handshake Message,
14	0.280822	192.42.113.248	192.168.198.130	TLSv1	Encrypted Handshake Message
15	0.280826	192.168.198.130	192.42.113.248	TCP	52918 > 9001 [ACK] Seq=457 Ack=2475 Win=10220 Len=0
16	0.296269	192.168.198.130	192.42.113.248	TLSv1	Encrypted Handshake Message, Encrypted Handshake Message, Encrypted Handshake Message,
17	0.296391	192.42.113.248	192.168.198.130	TCP	9001 > 52918 [ACK] Seq=2475 Ack=1810 Win=64240 Len=0
18	0.363769	192.42.113.248	192.168.198.130	TLSv1	Change Cipher Spec, Encrypted Handshake Message
19	0.376019	192.168.198.130	192.42.113.248	TLSv1	Application Data, Application Data
20	0.376119	192.42.113.248	192.168.198.130	TCP	9001 > 52918 [ACK] Seq=2565 Ack=1884 Win=64240 Len=0

Figure 15: Tor TLS Handshake

When we inspect the TLS handshake more in depth we can see that Tor offers 28 cipher suites in the Client hello as shown on figure 16. This fairly typical when we compare it to the other services we set up, such as HTTPS, SMTPS, IMAPS and POP3S which had around the same number of cipher suites.

Inspecting the Client hello we discover that Tor implements TLS extensions [67]. Specifically Tor implements server_name and SessionTicket TLS. The server_name extension is a hostname that resolves the server as understood by the client. The server_name is used to identify the servers certificate or enforce a security policy. This enables the usage of virtual hosting where several websites share a single IP address.

The SessionTicket enables an easier way for the server to manage TLS sessions [67]. Instead of storing all sessions on the server, and possibly corrupting the sessions, the client receives a session ticket. The Session ticket contains the all the session data such as cipher suite master secret. The session data is encrypted with AES cipher and then sent to the client at the end of the TLS handshake. For integrity checking, the session tickets SHA1-HMAC is calculated and sent along with the ticket. So when a returning client wish to resume communication it can just send the Session Ticket to the TLS server.

5.3.1 Tor Certificates

Tor uses the x.509 version 3 certificate which could reveal information that implies an user is running Tor. A certificate is mandatory in TLS connections and to authenticate Tor relays. This is

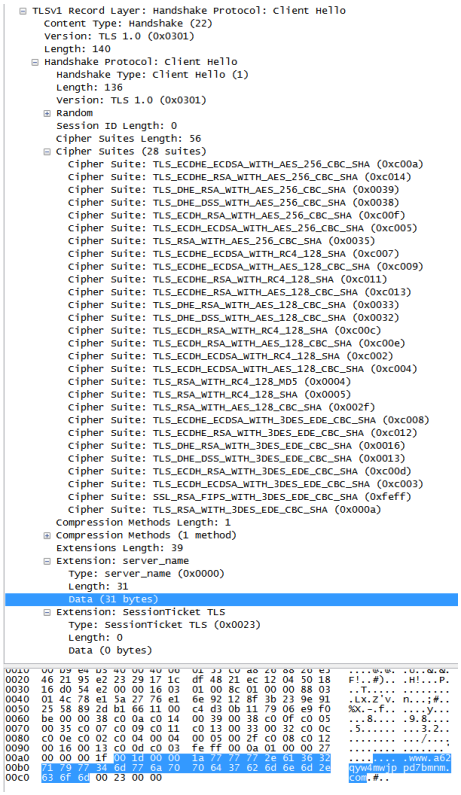

Figure 16: Tor Client hello

important to prevent man-in-the-middle attacks [68]. The certificate that Tor uses is self-signed and the certificate creator also signed its legitimacy. This means that no certificate authority ultimately signs the certificate and you have to trust that this is the correct node without any chain of trust.

Tor uses certificates a bit differently than normal services, which could reveal that the user is running Tor. The issuer and Common Name in the certificate is normally a humanly readable name. This is not case with Tor. In appendix A and B we have provided certificates from Tor and Google. The certificate generated by Tor is valid for two hours. This is a very short time frame for a certificate as we know from the TLS baseline test.

5.4 TLS and Tor TLS differences

There are some obvious differences in how Tor implements TLS and other TLS services. The port number Tor uses default is TCP port 9001 for communication. TCP port 9030 is used for directory authority communication. TLS traffic is often seen on TCP port 443 which is used for HTTPS traffic. The fact that Tor uses port 9001 is revealing because no other services uses this port for TLS traffic. Tor can also bind to other ports and advertise this to the directory servers which distributes this information to the clients. This is used by several Tor servers behind corporate firewalls or simply to obfuscate Tor's presence.

5.4.1 Handshake differences

We observed that Tor completes the handshake in four packets as seen in figure 15, two packets in each direction. This is the same as we observed for HTTPS, SMTPS, POPS and IMAPS in our test environment. This is the minimum amount of packets required to complete the handshake and the most common packet count we observed in our TLS baseline experiment.

When we look closer on the network traffic that Tor generates in the TLS handshake it does not deviate much from other TLS handshake implementations. We saw for example that Thunderbird offers 28 ciphers in the Client hello packet just as Tor. Firefox offered 33 by default which is a little more than the average we observed. The order of the available cipher suites also appear to match, except the that Tor doesn't support as many cipher suites but enough to make it pretty conform with other applications. The extensions that Tor uses in the Client hello are server name and and session ticket TLS. These are always present by our observations.

5.4.2 Cipher suite selection

To further explore the differences in the TLS negotiation process we decided to investigate the cipher suite selection. When a client connects to a server with TLS the client announces its available cipher suites in the Client hello packet. The server responds with a Server Hello and selects the most secure cipher suite which is common between them. In order to test the preferred cipher suites we first observed the traffic and noticed the selected cipher suite. We then removed the chosen cipher suite by configuration and rerun the experiment. We repeated this until TLS reported a handshake failure.

We first ran the experiment against a Tor Server. Since it is not possible to tweak the cipher suites in Tor without editing the source code and do a recompile we created a python script with Scapy [61]. The script allows us to send the Client hello packet in the TLS negotiation with parameters of our choosing. The script had the same cipher suites that Firefox sends in the Client hello packet. The result from this experiment is shown below in tables 4-9. The Python script is located in the Appendix C.

To compare with normal services that run TLS we tested the preferred cipher suites of HTTPS, SMTPS, POP3S and IMAPS. Mozilla Thunderbird and Mozilla Firefox both allows the user to tweak the available cipher suites easily with the graphical user interface. This allowed us to configure the list of cipher suites in the Client hello packet in a TLS negotiation. The tables use the standard abbreviations for cipher suites found in the TLS specification [16]. Each line is divided in groups where each group describes which technology to use. For example Tor uses TLS_DHE_RSA_WITH_AES_256_CBC_SHA by default. This translates to use TLS (not SSL), Diffie-Hellman and RSA for asymmetric crypto. Then use AES with 256 bit key in Cipher block chaining mode for symmetric crypto and SHA1 for integrity checking.

Table 4: Cipher suite selection: Tor

#	**Tor 0.2.2.1-alpha with openssl on Ubuntu Server 9.10 from script**
1	TLS_DHE_RSA_WITH_AES_256_CBC_SHA
2	TLS_DHE_RSA_WITH_AES_128_CBC_SHA
3	TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
4	TLS Handshake failure

Table 5: Cipher suite selection: Apache 2.2

#	**Apache 2.2 with openssl on Ubuntu Server 9.10 from Firefox (HTTPS)**
1	TLS_DHE_RSA_WITH_AES_256_CBC_SHA
2	TLS_RSA_WITH_AES_256_CBC_SHA
3	TLS_DHE_RSA_WITH_AES_128_CBC_SHA
4	TLS_RSA_WITH_RC4_128_MD5
5	TLS_RSA_WITH_RC4_128_SHA
6	TLS_RSA_WITH_AES_128_CBC_SHA
7	TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
8	TLS_RSA_WITH_3DES_EDE_CBC_SHA
9	TLS Handshake failure

Table 6: Cipher suite selection: Postfix

#	**Postfix on Ubuntu Server 9.10 from Thunderbird (SMTPS)**
1	TLS_DHE_RSA_WITH_AES_256_CBC_SHA
2	TLS_RSA_WITH_AES_256_CBC_SHA
3	TLS Handshake failure

Table 7: Cipher suite selection: Courier

#	**Courier on Ubuntu Server 9.10 from Thunderbird (IMAP4S and POP3S)**
1	TLS_RSA_WITH_AES_256_CBC_SHA
2	TLS_RSA_WITH_AES_128_CBC_SHA
3	TLS_RSA_WITH_3DES_EDE_CBC_SHA
4	TLS Handshake failure

Table 8: Cipher suite selection: Exchange 2010

#	**Exchange 2010 on Windows 2008 R2 Server from Thunderbird (SMTPS)**
1	TLS_RSA_WITH_RC4_128_MD5
2	TLS_RSA_WITH_3DES_EDE_CBC_SHA
3	TLS_RSA_WITH_RC4_128_SHA
4	TCP Connection Close

Table 9: Cipher suite selection: Internet Information Server 7 and Exchange 2010

#	**Internet Information Server 7 and Exchange 2010 on Windows 2008 R2 Server from Internet Explorer 8.0 and Thunderbird (HTTPS, IMAP4S, POP3S)**
1	TLS_RSA_WITH_AES_128_CBC_SHA
2	TLS_RSA_WITH_AES_256_CBC_SHA
3	TLS_RSA_WITH_RC4_128_SHA
4	TLS_RSA_WITH_3DES_EDE_CBC_SHA
5	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
6	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
7	TLS_RSA_WITH_RC4_128_MD5
8	TCP Connection Close

Tor has a unique list of preferred cipher suites as one can read from tables 4 to 9. This makes it possible to actively fingerprint a Tor server and distinguish it from other TLS services according to our results. It is possible that other services has the same fingerprint as Tor since we have not tested every service. We can not rely completely on these results alone. Tor uses Diffie-Hellman cipher suites to provide perfect forward secrecy and hence has a very short list of cipher suites to choose from. It also reports failure on the TLS layer when no cipher suites matches the server. This is different from TLS services from Microsoft which close the connection on the TCP layer immediately. This can be seen in table 8 and 9.

5.4.3 Certificate differences

The next place we observed differences were in the certificates. Tor uses only one certificate (certificate chain length one) in the handshake to connect to Tor relays. This is not normal according to our TLS baseline test. A public service where the certificate is self-singed and more importantly has no certificate chain is unusual. For a public service is it important to establish authenticity to make sure that one is communicating with the correct service. This is normally done by a chain of trust with certificates and a certificate authority at the top. Where the certificate of the root CA is self-signed. The certificate chain is not the only suspicious thing with Tor's TLS traffic. The certificate itself has some revealing deviations from other TLS certificates.

The certificates used in Tor are only valid for two hours. This is a very short time frame for a certificate. A typical certificate could be valid for one to five years. The options when buying a certificate from CA Thawte is one to five years [69]. It would be plausible to have a short lived certificate for testing purposes, but for long lived services such as email and web it appears strange. The validity time could be revealing, especially if Tor tries to mimic a public service.

The x.509 certificate has a Issued and Common Name field which normally is a humanly readable name. The Issuer field describes who signed and obviously who issued the certificate. This is random string of letters in the certificates that Tor uses and does not resolve to any IP address. The field follows a standard where "xyz" is a random string of letters: "www.xyz.net". This looks very odd because one of the points of DNS is to have strings that are easy to remember [65]. This is a very uncommon way to use a certificate and could reveal a Tor user on a corporate network for example. The next field which could raise suspicion is the Subject field. The Subject

consists of several sub fields which holds information about the organization and location. The only sub field that has any useful information for the authentication process is the Common Name field. This field binds the host name to the public key in the certificate. The Common Name field corresponds to the host name which it authenticates. On Tor certificates the field follows the same pattern of random string as the issuer field and could reveal that the traffic originates from Tor. The Tor certificate communicates that it shares very little information because it has omitted any sub fields except the Common Name. An example of a Tor certificate and a Google certificate for comparison are presented in appendix A and B.

5.4.4 Identifiers

To sum up the identifiers we provide the list below. There are two principal ways to do traffic analysis: active and passive. Passive traffic analysis doesn't send any traffic to the target and has the advantage that it is impossible to detect. Active traffic analysis will normally provide more information but reveals the attackers presence because he/she probes the target with traffic.

- Passive Identifiers
  - TCP port
  - Cipher suites in Client hello
  - Extension types in Client hello
  - Length of certificate chain
  - Certificate validity time frame
  - Certificate common name
  - Number of certificate extensions

- Active Identifiers
  - Cipher suite selection
  - Diffie-Helman support
  - Handshake failure behavior

5.5 Evaluation

We have discovered several indicators of Tor usage with different performance and usability. In the next section we evaluate each metric with respect to error rate and our results of the TLS baseline. We argue the need to combine several metrics to make sound decision regarding the TLS traffic.

5.5.1 Passive metrics

The default port for circuit creation in Tor is TCP port 9001 and 9030 for server descriptors. It would be a easy solution to close these ports for an adversary with intension to block Tor and Tor bridges. Fortunately it is possible to use Tor over port 443 or 80 which is open in almost every scenario. We claim that the TCP port as a identifier has a poor performance. It is not certain that Tor runs on port 9001, so no traffic on port 9001 doesn't necessary imply a lack of Tor and could give false negatives. If we observe traffic on port 9001 it could be some other service accidentally running on the Tor port and give false positives. The administrator can inmost cases set up a service on any port he/she likes.

In the Client hello the TLS client announces available ciphers. This list is usually different between implementations of TLS clients. The list ordering is also different and could be used to identify differences. We have not investigated every TLS implementation to find out if Tor is unique. Since there are so many variations we believe this identifier would give few false positives. False negatives would occur seldom because the the cipher suite list must match exactly. It could exist an implementation which is exactly that same as Tor's Client hello cipher suite list and one cannot relay on the cipher suite list alone as an identifier. There are very many cipher suite algorithms to choose from and the prospect of making a unique announcement is pretty good. Tor has stated that it tries to use the same list as Firefox, but this has not been maintained [64]. Firefox has 33 ciphers now while Tor still has 28 in the Client hello.

Tor uses a self-signed certificate chain of length one in the initial handshake. The only website we observed that used a certificate chain length of one was the Tor project site. This could be random because we observed a small set of sites and hence the metric has a poor reliability. The fact that we didn't observe a website with a certificate chain length one doesn't imply it doesn't exists and could give false positives. It is not possible to rely on this metric alone to conclude because it could be a coincident.

We extracted several metrics from the certificate of Tor. Tor's certificate has a time frame of two hours. We observed that this was uncommon because Thawte, a certificate authority, issues certificates with a life span of 1-5 years [69]. All websites we tested in the TLS baseline experiment had this time frame. False positives or false negatives seems unlikely from this result. We both observed and found documentation that Tor uses a unusual time frame in the certificate. The plausible explanation for having a certificate with such a short validity can be to rotate the keys often.

Tor uses a self-signed certificate and a common name which doesn't resolve to any site. The common name is a random string of letters which starts with "www" and ends with ".net" in Tor's certificate. A TLS service might use a random string as the Fully Qualified Domain Name but it would be very hard to remember and forfeit the point of DNS. This metric would be hard to implement because the filter must spot a random string. A computer would have difficulties deciding if the string is random. It relies on dictionaries, whitelisting or blacklisting to make the decision. It probably exists several websites with the structure of "www.<randomstring>.net" and some might even be a TLS website. If that's the case it would give false positives. The algorithm for deciding if the common name is random could produce errors and give false negatives.

Just like Client hello packets a certificate can also have extensions. The number of extensions

depends on the usage of the certificate. The certificate which a Tor server gives to the user has zero extensions while all the websites we observed has three or more extension. Again, since we observed a very small set of websites we could have picked servers with many extensions by random. There are many extensions to x.509 certificates which makes it suitable as an identifier because the number of variations that can occur. Since there might exist many certificates with the exact same extensions as Tor certificates it can only indicate and not conclude Tor usage. It is possible that false positives occur, but false negatives seem unlikely. The metric can still be used efficiently to exclude traffic as non Tor.

5.5.2 Active metrics

Active identifiers are not as interesting as the passive identifiers because the threat model we have defined uses a passive attacker. An active identifier can still be useful if one tries to identify Tor on the corporate network and it contributes to the overall normalization of Tor traffic. We experimented with the cipher suite selection algorithm in the TLS handshake to see which were preferred ciphers. We believe this metric has high reliability because we tested in isolated environment and tested several operating systems. Tor turned out to have unique fingerprint compared to the other implantations we tested and we believe it would give few false positives. Tor only supported Diffie-Helman ciphers which is pretty uncommon according to our results. The services we tested on the Linux machines used the OpenSSL library to facilitate TLS in the application [70]. OpenSSL is a open source library which implements SSL and TLS. It can be configured in numerous ways. It's a possibility that someone has configured their server to resemble a Tor server. Tor server only completes a handshake with the one of the four Diffie-Helman cipher suites. It would be bad for availability if this was a public service because many clients would not be supported. We argue that this behavior is uncommon for a server on the Internet. To extend the validity we also tested the same services on Microsoft Windows which has implemented TLS differently than OpenSSL.

We observed that the Tor server only complete a handshake if the Client hello presents a Diffie-Helman cipher suite. Tor must use Diffie-Helman with RSA to provide perfect forward secrecy which is important to stay anonymous if a node is comprised [22]. It is possible to indicate that one is communicating with a Tor server in just one packet by using this metric. If the server completes the handshake when given only Diffie-Helman cipher suites it raises the probability for identifying a Tor server. If we receive a handshake failure, it's not a Tor server. We believe this identifier has a reasonable false positive rate. We tested several times and in a isolated environment. If we observe that the TLS handshake completes we know that this could be a Tor server, but many servers use Diffie-Helman ciphers. Many high security sites such as Internet banking uses strong encryption. This application of this metric should be for excluding that we are talking to a Tor server.

We also observed that some TLS implementations close the communication on the TCP layer and not the TLS layer with a Handshake failure message. Only one of the servers in the TLS baseline test had this behavior. If we observe this we can conclude that this is not a Tor server. The metric would give many false positives because a lot of servers has this behavior. Table 10 sums up our evaluation of the metrics and their performance.

Table 10: Metric evalutaion

Metric	False positives	False negatives	Threat model	Filter type
TCP port	Unlikely	Likely	Passive	Include
Cipher suites	Unlikely	Unlikely	Passive	Include
TLS Extensions	Likely	Unlikely	Passive	Exclude
Certificate chain length	Unlikely	Unlikely	Passive	Exclude
Certificate validity time frame	Unlikely	Unlikely	Passive	Include
Certificate common name	Likely	Likely	Passive	Include
Number of certificate extensions	Unlikely	Unlikely	Passive	Include
Cipher suite selection	Unlikely	Unlikely	Active	Include
Diffie-Hellman support	Likely	Unlikely	Active	Include
Handshake failure	Likely	Unlikely	Active	Exclude

The table shows our evaluation of false positives and false negatives and how likely it is that they occur. Further it describes if the metric can be measured by passive or active traffic analysis. The last column states if the metric work best to exclude or include the traffic as Tor.

5.6 Snort detection

We have unveiled several revealing identifiers during the experimentation. Some are easy to implement and test while others could be hard and perhaps unfeasible to implement. The threat in our scenario was an authority conducting censorship by filtering Internet access in a central places. The filter needs to be efficient because it must potentially handle a very large traffic volume. The typical place for a such filter can be border gateways. It cannot do complex statistical analysis or any CPU intensive computations because it would degrade performance beyond acceptable limits. The identifiers we have found are not particular resource demanding and we argue they could be used by an adversary.

We tried to implement some of the identifiers which might imply Tor usage. To do this we used the open source intrusion detection system called Snort [8]. Snort has been around a long time and can be said to be a mature and reliable IDS. Snort uses preprocessors and signatures to scrutinize network traffic. It typically runs on a Linux operating system and requires very little configuration to work. We deployed Snort to a machine with Tor installed to see if we could detect Tor. Snort uses the signatures to identify possible malicious traffic. In order for Snort to detect Tor we must first create signatures that triggers on Tor usage. To add new signatures a simple language is used where one specifies the direction of traffic and the bytes which identify the traffic. We wrote signatures based on some of the identifiers we found. The signatures are available in appendix D.

The first signature detects the exact cipher suites used in the Client hello by the Tor handshake. It checks that the Client hello has the 28 cipher suites which Tor uses and has the right order. The second signature detects a certificate chain length of one. This is pretty uncommon for normal TLS traffic and perhaps specific to Tor as we discovered during the project [30]. The certificate validity was also relatively easy to implement in Snort. This signature checks if the "not

valid before" or "not valid after" parameter is before or after a specific date. The date must of course reflect the current date we are testing on. The last signature checks the certificate extensions. If there are zero extensions the alarm triggers. We choose to implement these signatures because they were feasible and quick to write in Snort. The other Tor identifiers we found would require more complex checking. For example checking the common name and deciding whether it's humanly readable can be difficult implement. We omitted the active identifiers because Snort is a passive tool and transparent to the user.

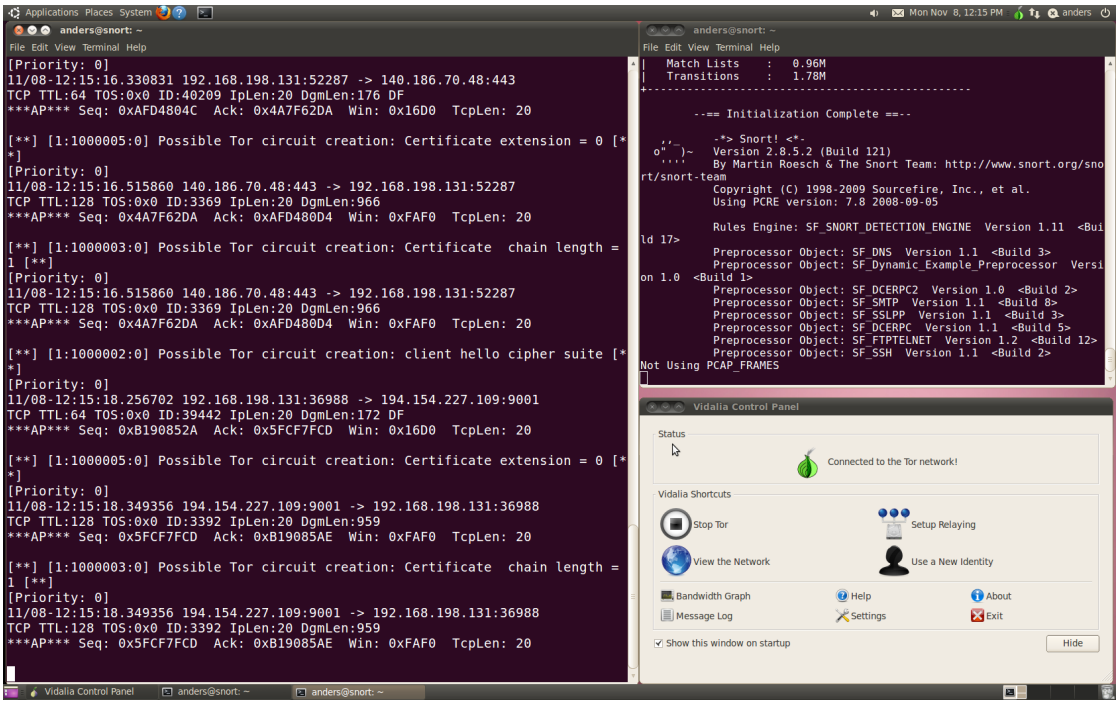

Figure 17: Snort detects Tor

Figure 17 depicts a screen shot from the actual test we see that Snort can detect Tor with the identifiers we have found. On the screen shot at bottom right we can see that Vidalia has a green onion which means that Tor is up and running. In the right top command shell we see Snort is started and running. The left command shell outputs the Snort alert file. The alert file is written to when an alert triggers. We see that all the rules we wrote was triggered and indicates that Tor is used on the network.

5.7 Normalization layer

Traffic mimic is a tool which uses live traffic as a model-base to hide the network fingerprint of Tor without the use of cover traffic [25]. This resulted in minimal loss of performance and making it more difficult to identify Tor traffic. Traffic mimic was implemented as a layer between Tor and the network interface. We argue that Tor needs such a normalization layer. This layer would

adjust, remove or add items to the traffic to ensure that Tor doesn't stand out from normal TLS traffic. The layer would as a minimum normalize the issues discovered as passive identifiers in this report. The normalization layer could mimic different encrypted services. In this suggestion we try to normalize the traffic to HTTPS. This is probably the most extensively used encrypted protocol and we have the best data on this service. We propose that the normalization layer address these differences:

- Normalization layer mimicking HTTPS traffic
  - TCP port - Tor should use same port as HTTPS (443).
  - Cipher suites in the Client hello - The cipher suites should mimic a well known browser, for instance Firefox.
  - Extension types in Client hello - Tor should use the extensions according to a well known browser, for instance Firefox.
  - Length of certificate chain - The chain should be at least two long according to our results.
  - Certificate validity time frame - The time frame should be 1-5 years according to our results.
  - Certificate common name - Common name should be humanly readable and look like plausible website.
  - Number of certificate extensions - There should be several extensions. The extensions doesn't need to have any function other than normalization.
  - Model-based traffic limiting - Trafficmimic can use live HTTPS traffic to mimic HTTPS streams and should be used[25].

Some of the issues would be fairly simple to implement in Tor. Especially normalizing the TCP port, cipher suites and extensions could make a difference. Normalizing the certificate issues could be harder because they are closely connected to Tor's implementation. Adding certificate extensions could make a difference and make Tor more resilient. We elaborate further in the discussion.

A normalization layer in Tor should be implemented with as little performance penalty as possible. Tor's performance is already slow and throws off many users. On the other hand it keeps bittorrent users downloading illegal material away from Tor. We argue that the normalization layer will make a very small contribution to degrade performance. Most of the work is done on the client. A normalization layer is independent of Tor's architecture and the developers doesn't need to change the design.

6 Discussion

In this thesis we have investigated the network traffic and behavior of Tor, an anonymity service, and compared the traffic to normal TLS network traffic. We show that there are several patterns that reveal Tor usage. It is very hard to detect Tor by measuring one parameter, but put together the evidence is strong. When Tor diverges too much from normal TLS traffic it will become trivial for an adversary to block Tor, even when bridges are used. It is important that Tor has a strong blocking resistance to maintain usability and security of the users. Tor is used by people with different needs, but some need protection [21]. Specially people like dissidents, whisteblowers and people living under stringent regimes. Tor offers to protect their identities and has an obligation to do it securely.

To be able to deduce information from encrypted traffic there have been much research in the area of traffic analysis. These techniques can been used on Tor to degrade anonymity of the users [44, 42, 46]. A protocol classification tool called PISA demonstrated that it is possible to identify traffic types from encrypted traffic. The motivation was to identify bittorrent traffic which can have huge impact on network performance. PISA used ten metrics to preform traffic analysis and could identify different applications [26].

We have chosen to use less complicated detection techniques, but instead identify simple and performance efficient identifiers for Tor traffic. This brought us to reveal Tor traffic indicators in the TLS handshake. We could have missed important and definite identifiers that could be found using statistics and more advanced traffic analysis. These can be subtle and we choose to focus on less CPU intensive differences that could be implemented in a large volume traffic filter. A recent paper also proposes a new way of normalizing such differences by using live network traffic as a model in which Tor should send packets [25]. The tool is called Trafficmimic and can use live data to mimic HTTPS network traffic for example. The technique levels out differences in timing and latency especially.

It seems like the Tor project has done some efforts to normalize the traffic which Tor usage generates. For example a proposal describes the Client hello packet in a TLS handshake should be a close match to Mozilla Firefox's implementation [64]. The Tor project has also made it possible to run a Tor server or bridge on port 443 or any other port to omit firewall rules. With that said we have the impression that there are not very many variations to choose from when implementing TLS. When a TLS traffic pattern deviates from the norm it becomes very obvious that the traffic is something different.

6.1 Fixable identifiers

Lets take a look at the identifiers we found and try to determine if the issues can be fixed. Tor runs by default on port 9001 and use port 9030 to get status from directory servers. If we oberve this it is a strong indicator for Tor usage. But Tor can also be run on port 443 which is harder to detect. This is less obvious because port 443 normally carries TLS traffic. Which port the Tor

server runs at is trasmitted to and advertised by the directory servers. Changing Tor's default port to 443 should be possible. It would protect the users better. An adversary might choose to block port 443 or other ports carrying encrypted traffic in order to ban encrypted traffic completely. Blindly blocking ports will not make Tor useless. One would eventually end up blocking all ports.

The Client hello packet is sent by the Tor client in the TLS handshake. In this packet the client advertise the available ciphers it is capable of. This differs among TLS clients. By passively looking for the same cipher suites and their order in the list we can indicate Tor usage. This behavior can be fixed without altering how Tor works. Tor should decide on a service to mimic and be consistent according to this service. If the service is HTTPS it must decide on a browser to mimic. For example Chrome advertises 11 ciphers while Firefox advertises 33 at the time of the experiment. The behavior must be changed as the browsers change to make it conform. We believe the best approach is to pick an application and try to mimic it as best as possible.

We saw that Tor uses the Server name and Session ticket extensions in the TLS handshake. Some TLS clients use more than these two extensions. If the adversary observes TLS handshakes with more than these extensions it is not likely Tor. We think the behavior is acceptable because many other clients also uses these two extensions. If Tor should include more extensions requires more research. More data is needed to find out what extensions are common for which service.

The number of extensions in Tor's certificate are zero. Our results show that this is uncommon for TLS, the number of extensions is normally around 5-10. By adding fake extensions and not use them could make Tor harder to detect. We believe this is easy to fix. One could add pseudo extensions and omit parsing them.

6.2 Possibly fixable identifiers

SSL Labs and our results show that most certificate chain lengths are longer than one [63]. To fix this can be difficult because Tor doesn't use a trusted certificate chain but rather a self-signed certificate. The certificate is self-signed and it may not serve any purpose to make the certificate chain longer because the self-signed certificate already causes attention. We argue that the sum of deviations from the norm makes Tor stand out and not some single indicator. Since the certificate doesn't give any trust relationship it should be possible to make the certificate chain longer by adding a pseudo certificate chain. It should be added that Tor uses a two-certificate chain after the initial handshake [6]. This is encrypted and not visible to anybody reviewing the encrypted traffic.

We found that the Tor certificate validity time deviated from the norm in the SSL survey [63]. SSL Labs and this project found that the validity is normally around 1-5 years. This is because the Tor development team wanted to rotate the keys fairly often. If a Tor server was comprised this would limit the attack because the keys are rotated and changed. The attacker could only decrypt traffic 2 hours old in theoretical attack. If the validity time increases the time available for the attacker goes up. The decision to increase the validity time must be seen in comparison with this problem.

6.3 Unfixable identifiers

The common name in the Tor certificate is not humanly readable. The common name describes the URl which the certificate is bound to. This is not easy to fix because this name is used to identify the Tor server. Fixing this issue would require to change the way Tor works and could cause a larger effort. Since it could be difficult to implement this identifier in a filter an adversary might choose lower hanging fruit.

Tor supports Diffie-Helman with RSA ciphers only. This is very characteristic for Tor as we experienced in active testing of the TLS handshake. The other services POP3S, IMAPS et cetera used by default a greater variety of cipher suites. We think fixing this issue in Tor is not as important as fixing the passive issues. It will contribute to the Tor normalization should eventually be addressed.

6.4 Application

The identifiers can be used to include or exclude Tor TLS traffic. There might be many applications which resemble the traffic that Tor generates. It is therefore impossible to say with certainty that this is Tor traffic or not Tor traffic. Filter creators are torn between whether making false positives or false negatives. It not recommended to conclude or rely on a single identifier alone. We have discovered several good identifiers that together could indicate Tor usage, but it cannot with certainty identify Tor.

By the course of this study we have come to the understanding that is possible to identify traffic from an application [26, 25]. We have also done a analysis of Tor to find revealing identifiers. Tor uses TLS in an unusual way which makes it possible to indicate Tor usage by looking at that the encrypted traffic alone. By deploying these metrics as identifiers we could create a filter which blocks Tor. Censorship resistance is important in Tor to increase usability and security of the users.

7 Conclusion

We have studied Tor, an anonymity service, to find out if the TLS traffic generated can be identified. If the traffic can be easily identified it would be trivial to block the traffic and deny access to Tor. People living under censorship would not be able to freely access the Internet. The rest of the conclusion revisits the research questions and summarizes our findings.

What are the characteristics of a TLS stream in terms of traffic analysis?
We found that a TLS stream has certain characteristics. Especially in the Client hello packet and the certificate. The TLS protocol has several potential places where implementation differences might occur. We looked at the TLS handshake implementation of Tor and other TLS services. A field experiment was conducted to locate typical parameters in TLS streams on the Internet. The experiment was backed up by relevant research [63]. We then compared this baseline to Tor TLS traffic.

How does Tor implement TLS and how is it different from other TLS services?
We have discovered several differences between Tor TLS traffic and normal TLS traffic. The most surprising differences were discovered in the certificate. Tor uses a validity of two hours while the norm lies between 1-5 years. The indicators cannot be used alone but together they make a strong implication that Tor is used on the network. Some indicators work best in excluding Tor while others work best in including Tor traffic. Both types should be used to get the most precise filter. We discovered both passive and active ways to fingerprint Tor. We found that Tor had a unique fingerprint when it comes to cipher suite selection algorithm.

Are there any changes that could make Tor look more like a common TLS stream?
At the end of this thesis we propose to the Tor developers to add a normalization layer that even out differences between Tor TLS and normal TLS streams. The normalization layer must mimic a service realistically. If Tor tries to mimic HTTPS the TCP port should be 443. We found that most revealing factors of Tor lies in the Client hello packet and the certificate. The certificate should have a validity of 1-5 years, humanly readable common name, at least 3 extensions and be in a chain of at least 2 certificates.

As anticipated we found that it is possible to identify Tor traffic quite easily. We have quantified normal TLS traffic and found that Tor diverges from normal TLS traffic. This means an adversary could make a filter to block Tor. It doesn't matter if the user takes advantage of the bridge feature in Tor. Tor has a poential to become more censorship resistant. The users could be forced to give up Tor and face a censored Internet experience.

8 Further Work

We have mainly investigated the differences in the TLS handshake. We have not applied traffic analysis to the encrypted traffic generated by normal TLS or Tor TLS traffic. It would be interesting to discover the differences in the encrypted traffic. Traffic mimic has the potential to level out the differences but to verify the results we need to know the differences [25]. The differences in encrypted TLS traffic beyond the handshake could be things like traffic volume flow, latency, packet size and other statistical data. PISA, the tool presented at Black hat Las Vegas in 2007, implements many of the statistical tests [26]. It would be very interesting to see if the tool could detect Tor easily and what the revealing factors were. SSL Labs' study of TLS traffic looked mainly into the handshake and certificate. To make sure that Tor doesn't reveal itself, the properties of encrypted TLS traffic should be researched. This would enable the Tor project to tune in the Tor TLS traffic to normal TLS traffic completely.

Our identifiers differs in level of performance for indicating Tor usage. We have not weighed the identifiers as to how much certainty each give in identifying Tor. In OS fingerprinting one can use probability and get a more accurate result [71]. This was done in Xprobe2 which is an OS fingerprinting program. We believe many of the same principles in can also be used in fingerprinting Tor.

Bibliography

[1] Torproject. 2010. Tor-relay mailing list; problem with bridges and a suggestion. http://archives.seul.org/tor/relays/May-2010/msg00037.html.

[2] Marriam-Webster. 2010. The marriam-webster dictionary. http://www.merriam-webster.com/dictionary/privacy.

[3] United Nations. 2010. Universal declaration of human rights. http://www.unhchr.ch/udhr/lang/eng.htm.

[4] OpenNet Initiative. 2010. Opennet initiative. http://opennet.net/.

[5] Torproject. 2009. Tor project. https://www.torproject.org/.

[6] Dingledine, R. & Mathewson, N. 2009. Tor protocol specification. https://git.torproject.org/checkout/tor/master/doc/spec/tor-spec.txt.

[7] Dingledine, R. 2007. Defcon 15: Tor and blocking- resistance. http://video.google.com/videoplay?docid=-9081582671026610093.

[8] Zalewski, M. 2010. Snort. http://www.snort.org/.

[9] Goldberg, I. A. & Chair-Brewer, E. *A pseudonymous communications infrastructure for the internet*. PhD thesis, 2000.

[10] Sassaman, L., Moller, U., Tuckley, C., Arneson, E., Kirk, A., & Palfrader, P. 2008. Mixmaster. http://mixmaster.sourceforge.net/.

[11] Danezis, G. 2007. Mixinion. http://mixminion.net/.

[12] Freier, A. O., Karlton, P., & Kocher, P. C. 1996. Ssl 3.0 specification. http://tools.ietf.org/html/draft-ietf-tls-ssl-version3-00.

[13] Gnu. 2010. Gnu. http://www.gnu.org/software/gnutls/manual/html_node/On-SSL-2-and-older-protocols.html.

[14] IETF. 2009. The internet engineering task force. http://www.ietf.org/.

[15] Dierks, T. & Allen, C. 1999. The tls protocol version 1.0. http://www.ietf.org/rfc/rfc2246.txt.

[16] Dierks, T. & Rescorla, E. 2008. The transport layer security (tls) protocol version 1.2. http://tools.ietf.org/html/rfc5246.

[17] Microsoft. 2010. Transport layer security. http://msdn.microsoft.com/en-us/library/aa380516(VS.85).aspx.

[18] Wikipedia. 2010. Transport layer security pictures. http://en.wikipedia.org/wiki/Transport_Layer_Security.

[19] Microsoft. 2010. Tls handshake protocol. http://msdn.microsoft.com/en-us/library/aa380513(VS.85).aspx.

[20] Syverson, P. F., Reed, M. G., & Goldschlag, D. M. 2000. Onion routing access configurations. In *DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, 34–40. IEEE CS Press.

[21] Torproject. 2009. Tor project. https://www.torproject.org/torusers.html.en.

[22] Dingledine, R., Mathewson, N., & Syverson, P. 2004. Tor: The second-generation onion router. In *In Proceedings of the 13 th Usenix Security Symposium*.

[23] Reardon, J. 2008. Improving tor using a tcp-over-dtls tunnel. http://uwspace.uwaterloo.ca/bitstream/10012/4011/1/thesis.pdf.

[24] Dingledine, R. & Mathewson, N. 2009. Tor bridges specification. http://gitweb.torproject.org/tor.git?a=blo0b_plain;hb=HEAD;f=doc/spec/bridges-spec.txt.

[25] Schear, N. & Nicol, D. 2010. Defending against encrypted traffic analysis using protocol behavior models. N/A.

[26] Dhamankar, R. & King, R. 2010. Protocol identification via statistical analysis. https://www.blackhat.com/presentations/bh-usa-07/Dhamankar_and_King/Whitepaper/bh-usa-07-dhamankar_and_king-WP.pdf.

[27] Murdoch, S. J. & Zieliński, P. 2007. Sampled traffic analysis by internet-exchange-level adversaries. In *PET'07: Proceedings of the 7th international conference on Privacy enhancing technologies*. Springer-Verlag.

[28] Fu, X., Graham, B., Bettati, R., & Zhao, W. 2003. Active traffic analysis attacks and countermeasures. In *ICCNMC '03: Proceedings of the 2003 International Conference on Computer Networks and Mobile Computing*. IEEE Computer Society.

[29] Chen, S., Wang, R., Wang, X. F., & Zhang, K. 2010. Side-channel leaks in web applications: a reality today, a challenge tomorrow. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*. IEEE Computer Society.

[30] SSL & Labs. 2010. Http client fingerprinting using ssl handshake analysis. https://www.ssllabs.com/projects/index.html.

[31] Wagner, D. & Schneier, B. 1996. Analysis of the ssl 3.0 protocol. In *WOEC'96: Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce*. USENIX Association.

[32] Sotirov, A., Stevens, M., Appelbaum, J., Arjen Lenstra, D. M., Osvik, D. A., & de Weger, B. 2008. Creating a rogue ca certificate. http://www.win.tue.nl/hashclash/rogue-ca/.

[33] Housley, R., Ford, W., Plok, W., & Solo, D. 1999. Internet x.509 public key infrastructure certificate and crl profile. http://tools.ietf.org/html/rfc2459.txt.

[34] Marlinspike, M. 2002. Ie ssl vulnerability. http://www.thoughtcrime.org/ie-ssl-chain.txt.

[35] Marlinspike, M. 2010. sslsniff. http://www.thoughtcrime.org/software/sslsniff/.

[36] Marsh Ray, S. D. 2009. Renegotiating tls. http://extendedsubset.com/?p=8.

[37] Hintz, A. 2002. Fingerprinting websites using traffic analysis. In *Workshop on Privacy Enhancing Technologies*.

[38] Danezis, G. 2009. Traffic analysis of the http protocol over tls. https://gitweb.torproject.org/tor.git/blob/HEAD:/doc/spec/proposals/124-tls-certificates.txt.

[39] Cheng, H. & Avnur, R. 1998. Traffic analysis of ssl encrypted web browsing.

[40] Foundstone. 2010. Ssldigger. http://www.foundstone.com/us/resources/proddesc/ssldigger.htm.

[41] Cheng, H., Cheng, H., & Avnur, R. 1998. Traffic analysis of ssl encrypted web browsing. http://www.cs.berkeley.edu/ daw/teaching/cs261-f98/projects/final-reports/ronathan-heyning.ps.

[42] Murdoch, S. J. & Danezis, G. 2005. Low-cost traffic analysis of tor. In *In Proceedings of the 2005 IEEE Symposium on Security and Privacy. IEEE CS*, 183–195.

[43] Evans, N. S., Dingledine, R., & Grothoff, C. 2009. A practical congestion attack on tor using long paths.

[44] Hopper, N., Vasserman, E. Y., & Chan-tin, E. 2007. How much anonymity does network latency leak. http://www-users.cs.umn.edu/ hopper/ccs-latency-leak.pdf.

[45] McLachlan, J. & Hopper, N. 2009. On the risks of serving whenever you surf: Vulnerabilities in tor's blocking resistance design.

[46] Øverlier, L. 2006. Locating hidden servers. In *In Proceedings of the 2006 IEEE Symposium on Security and Privacy. IEEE CS*, 100–114.

[47] Murdoch, S. J. 2006. Hot or not: Revealing hidden services by their clock skew. In *In 13th ACM Conference on Computer and Communications Security (CCS 2006)*, 27–36. ACM Press.

[48] Thuren, T. 1993. *Vitenskapsteori for Nybegynnere*. Universitets Forlaget.

[49] Leedy, P. D. 1974. *Practical research: planning and design [by] Paul D. Leedy 8th edition*. Macmillan New York.

[50] Chew, E., Swanson, M., Stine, K., Bartol, N., Brown, A., Robinson, W., & Gutierrez, C. M. 2008. Nist special publication 800-55 revision 1 performance measurement guide for information security. http://csrc.nist.gov/publications/PubsSPs.html.

[51] Vaughn, R. B. 2003. Information assurance measures and metrics - state of practice and proposed taxonomy. In *In Proc. of Hawaii International Conference on System Sciences*. IEEE Computer Society Press.

[52] VMware. 2010. Vmware workstation. http://www.vmware.com/.

[53] Canonical & Ltd. 2010. Ubuntu. http://www.ubuntu.com.

[54] Microsoft. 2010. Windows 2008 server. http://www.microsoft.com/windowsserver2008/en/us/default.aspx

[55] Microsoft. 2010. Exchange 2010. http://www.microsoft.com/exchange/2010/en/us/default.aspx.

[56] Microsoft. 2010. Internet information server 7. http://www.microsoft.com/windowsserver2008.

[57] TorProject. 2010. Vidalia. http://www.torproject.org/vidalia/.

[58] Ubuntu. 2010. Ubuntu community. https://wiki.ubuntu.com/DocumentationTeam.

[59] Tcpdump. 2010. Tcpdump. http://www.tcpdump.org.

[60] Wireshark. 2009. Wireshark. http://www.wireshark.org/.

[61] Biondi, P. 2010. Scapy. http://www.secdev.org/projects/scapy/doc/.

[62] M. Nystrom, D. Hopwood, J. M. & Wright, T. 2003. The transport layer security (tls) protocol extensions. http://www.ietf.org/rfc/rfc3546.txt.

[63] Ristic, I. 2010. Internet ssl survey 2010. http://blog.ivanristic.com/2010/07/internet-ssl-survey-2010-is-here.html.

[64] Torproject. 2007. Proposal 124: Blocking resistant tls certificate usage. https://gitweb.torproject.org/tor.git.

[65] Mockapetris, P. 1987. Domain names - concepts and facilities. http://tools.ietf.org/html/rfc1034.txt.

[66] Alexa. 2010. Alexa topsites. http://www.alexa.com/topsites.

[67] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., & Wright, T. 2003. Transport layer security extensions. http://www.ietf.org/rfc/rfc3546.txt.

[68] Marlinspike, M. 2009. Null prefix attacks against ssl/tls certificates. http://www.thoughtcrime.org/papers/null-prefix-attacks.pdf.

[69] Thawte. 2010. Thawte certificate buying. http://www.thawte.com/ssl/web-server-ssl-certificates/index.html.

[70] Cox, M. J., Engelschall, R. S., Henson, S., & Laurie, B. 2010. Openssl project. https://www.ssllabs.com/.

[71] Arkin, O. & Yarochkin, F. 2002. Xprobe2 – a 'fuzzy' approach to remote active operating system fingerprinting. http://ofirarkin.files.wordpress.com/2008/11/xprobe2.pdf.

A Tor Certificate

Certificate:
Data:
Version: 3 (0x2)
Serial Number:
1268030768 (0x4b949d30)
Signature Algorithm: sha1WithRSAEncryption
Issuer: CN=www.lmbesi7al4okcp53.net
Validity
Not Before: Mar 8 06:46:08 2010 GMT
Not After : Mar 8 08:46:08 2010 GMT
Subject: CN=www.ho6enmoeg4x4quw5xa4x.net
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public Key: (1024 bit)
Modulus (1024 bit):
00:cd:f8:18:4c:ff:c2:6b:0d:16:52:75:ff:94:7f:
7e:26:3d:60:14:00:3e:80:b2:a5:b2:2a:56:5d:93:
00:1d:1f:59:e3:c7:ae:2b:d9:d7:1b:ed:17:f6:ca:
3c:89:83:c0:a3:7e:64:f6:22:ae:90:7c:70:8f:d9:
34:3e:91:98:cd:77:f9:a4:59:52:c9:05:4a:6b:e0:
21:50:28:4e:c8:2e:b2:d2:9e:4d:b5:c3:65:98:da:
29:5b:15:f3:32:a5:12:a4:64:e7:50:b3:0e:97:56:
99:b3:62:9f:84:49:25:42:78:6f:6d:3e:14:b6:5a:
fc:f6:b0:2f:31:67:78:9f:07
Exponent: 65537 (0x10001)
Signature Algorithm: sha1WithRSAEncryption
03:38:9f:25:a1:ff:c9:75:8d:42:38:f9:b5:bc:16:2e:b0:e8:
28:48:40:cf:64:bf:0c:dd:fb:cb:a4:1f:1e:e6:e5:db:1f:ad:
65:54:60:e5:85:9d:7d:d2:ce:ad:3c:4e:d9:8a:1d:70:25:85:
f5:d2:5e:9b:80:37:ba:ee:5f:6d:e1:4b:45:a0:b8:90:81:19:
fd:70:1f:24:03:0f:66:c0:68:86:c5:00:46:9d:4c:22:65:52:
c6:0f:85:b8:da:00:ab:a4:9b:30:f5:41:45:d2:d9:82:9b:fb:
d1:8a:bb:84:f7:05:d3:b7:5c:12:bc:56:5c:ce:9f:d4:0d:b3:
45:e0
—–BEGIN CERTIFICATE—–
MIIBwTCCASqgAwIBAgIES5SdMDANBgkqhkiG9w0BAQUFADAjMSEwHwYDVQQDExh3
d3cubG1iZXNpN2FsNG9rY3A1My5uZXQwHhcNMTAwMzA4MDY0NjA4WhcNMTAwMzA4

MDg0NjA4WjAnMSUwIwYDVQQDExx3d3cuaG82ZW5tb2VnNHg0cXV3NXhhNHgubmV0
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDN+BhM/8JrDRZSdf+Uf34mPWAU
AD6AsqWyKlZdkwAdH1njx64r2dcb7Rf2yjyJg8CjfmT2Iq6QfHCP2TQ+kZjNd/mk
WVLJBUpr4CFQKE7ILrLSnk21w2WY2ilbFfMypRKkZOdQsw6XVpmzYp+ESSVCeG9t
PhS2Wvz2sC8xZ3ifBwIDAQABMA0GCSqGSIb3DQEBBQUAA4GBAAM4nyWh/8l1jUI4
+bW8Fi6w6ChIQM9kvwzd+8ukHx7m5dsfrWVUYOWFnX3Szq08TtmKHXAlhfXSXpuA
N7ruX23hS0WguJCBGf1wHyQDD2bAaIbFAEadTCJlUsYPhbjaAKukmzD1QUXS2YKb
+9GKu4T3BdO3XBK8VlzOn9QNs0Xg
—END CERTIFICATE—

B Google Certificate

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            2f:df:bc:f6:ae:91:52:6d:0f:9a:a3:df:40:34:3e:9a
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=ZA, O=Thawte Consulting (Pty) Ltd., CN=Thawte SGC CA
        Validity
            Not Before: Dec 18 00:00:00 2009 GMT
            Not After : Dec 18 23:59:59 2011 GMT
        Subject: C=US, ST=California, L=Mountain View, O=Google Inc, CN=www.google.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
            Modulus (1024 bit):
                00:e8:f9:86:0f:90:fa:86:d7:df:bd:72:26:b6:d7:
                44:02:83:78:73:d9:02:28:ef:88:45:39:fb:10:e8:
                7c:ae:a9:38:d5:75:c6:38:eb:0a:15:07:9b:83:e8:
                cd:82:d5:e3:f7:15:68:45:a1:0b:19:85:bc:e2:ef:
                84:e7:dd:f2:d7:b8:98:c2:a1:bb:b5:c1:51:df:d4:
                83:02:a7:3d:06:42:5b:e1:22:c3:de:6b:85:5f:1c:
                d6:da:4e:8b:d3:9b:ee:b9:67:22:2a:1d:11:ef:79:
                a4:b3:37:8a:f4:fe:18:fd:bc:f9:46:23:50:97:f3:
                ac:fc:24:46:2b:5c:3b:b7:45
            Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 CRL Distribution Points:
                URI:http://crl.thawte.com/ThawteSGCCA.crl

        X509v3 Extended Key Usage:
            TLS Web Server Authentication, TLS Web Client Authentication, Netscape Server Crypto
        Authority Information Access:
            OCSP - URI:http://ocsp.thawte.com
            CA Issuers - URI:http://www.thawte.com/repository/Thawte_SGC_CA.crt
```

Signature Algorithm: sha1WithRSAEncryption
9f:43:cf:5b:c4:50:29:b1:bf:e2:b0:9a:ff:6a:21:1d:2d:12:
c3:2c:4e:5a:f9:12:e2:ce:b9:82:52:2d:e7:1d:7e:1a:76:96:
90:79:d1:24:52:38:79:bb:63:8d:80:97:7c:23:20:0f:91:4d:
16:b9:ea:ee:f4:6d:89:ca:c6:bd:cc:24:68:d6:43:5b:ce:2a:
58:bf:3c:18:e0:e0:3c:62:cf:96:02:2d:28:47:50:34:e1:27:
ba:cf:99:d1:50:ff:29:25:c0:36:36:15:33:52:70:be:31:8f:
9f:e8:7f:e7:11:0c:8d:bf:84:a0:42:1a:80:89:b0:31:58:41:
07:5f

——BEGIN CERTIFICATE——
MIIDITCCAoqgAwIBAgIQL9+89q6RUm0PmqPfQDQ+mjANBgkqhkiG9w0BAQUFADBM
MQswCQYDVQQGEwJaQTElMCMGA1UEChMcVGhhd3RlIENvbnN1bHRpbmcgKFB0eSkg
THRkLjEWMBQGA1UEAxMNVGhhd3RlIFNHQyBDQTAeFw0wOTEyMTgwMDAwMDBaFw0x
MTEyMTgyMzU5NTlaMGgxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlh
MRYwFAYDVQQHFA1Nb3VudGFpbiBWaWV3MRMwEQYDVQQKFApHb29nbGUgSW5jMRcw
FQYDVQQDFA53d3cuZ29vZ2xlLmNvbTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkC
gYEA6PmGD5D6htffvXImttdEAoN4c9kCKO+IRTn7EOh8rqk41XXGOOsKFQebg+jN
gtXj9xVoRaELGYW84u+E593y17iYwqG7tcFR39SDAqc9BkJb4SLD3muFXxzW2k6L
05vuuWciKh0R73mkszeK9P4Y/bz5RiNQl/Os/CRGK1w7t0UCAwEAAaOB5zCB5DAM
BgNVHRMBAf8EAjAAMDYGA1UdHwQvMC0wK6ApoCeGJWh0dHA6Ly9jcmwudGhhd3Rl
LmNvbS9UaGF3dGVTR0NDQS5jcmwwKAYDVR0lBCEwHwYIKwYBBQUHAwEGCCsGAQUF
BwMCBglghkgBhvhCBAEwcgYIKwYBBQUHAQEEZjBkMCIGCCsGAQUFBzABhhZodHRw
Oi8vb2NzcC50aGF3dGUuY29tMD4GCCsGAQUFBzAChjJodHRwOi8vd3d3LnRoYXd0
ZS5jb20vcmVwb3NpdG9yeS9UaGF3dGVfU0dDX0NBLmNydDANBgkqhkiG9w0BAQUF
AAOBgQCfQ89bxFApsb/isJr/aiEdLRLDLE5a+RLizrmCUi3nHX4adpaQedEkUjh5
u2ONgJd8IyAPkU0Wueru9G2Jysa9zCRo1kNbzipYvzwY4OA8Ys+WAi0oR1A04Se6
z5nRUP8pJcA2NhUzUnC+MY+f6H/nEQyNv4SgQhqAibAxWEEHXw==
——END CERTIFICATE——

C Python Scapy TLS testing script

```
#! /usr/bin/env python
#Author Anders Olaus Granerud
#Master Thesis 2010 at Gjøvik University College
#usage:
# $sudo ./scapy
# $import scapytls
# $test()

    #Import Scapy
from scapy.all import *
from scapy.fields import *
from scapy.packet import *
import time

    #Set highest possible loglevel
import logging
logging.getLogger("scapy").setLevel(1)

#Prevent reset packets produced by the kernel with Iptables
#sudo iptables -A OUTPUT -p tcp –tcp-flags RST RST -s 1.2.3.4 -d 4.3.2.1 –dport 9001 -j DROP
#sudo iptables -A OUTPUT -p tcp –tcp-flags RST RST -s 1.2.3.4 -d 4.3.2.1 –dport 443 -j DROP

#This class creates a Client hello packet in the TLS Handshake
class TLS(Packet):
    #First we describe the packet by using the Scapy interface
    name = ¨TLS 1.0¨
    fields_desc=[
        ByteEnumField("ContentType",22,
            {20: "Change Cipher Spec", 21:"Alert", 22:"Handshake", 25:"Application Data"}),
        ShortEnumField("Version",769,
            {768:"TLS 1.0", 769:"TLS 1.1", 770:"TLS 1.2"}),
        XShortField("RecordLayerLength",None),
        ByteEnumField("HandshakeType",1,
            {0: "Hello Request", 1:"Client hello", 2:"Server Hello", 11:"Certificate", 12: "Server Key
Exchange", 13: "Certificate Request", 14: "Server Hello Done", 15: "Certificate Verify", 16: "Client
Key Exchange", 20: "Finsihed"}),
        XByteField("HandShakeLengthNull",0), #Hack to make handshake length three bytes
```

```
        XShortField("HandShakeLength",None),
        ShortEnumField("Version",769,
            {768:"TLS 1.0", 769:"TLS 1.1", 770:"TLS 1.2"}),
        IntField("GMTUnixTime",None),
        XIntField("Random0",None),
        XIntField("Random1",None),
        XIntField("Random2",None),
        XIntField("Random3",None),
        XIntField("Random4",None),
        XIntField("Random5",None),
        XIntField("Random6",None),
        XByteField("SessionIDLength",0),
        XShortField("CipherSuitesLength",None),
        XShortField("TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA",eval(hex(49162))),
        XShortField("TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA",eval(hex(49172))),
        #XShortField("TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA",eval(hex(136))),  #not
used by tor client
        #XShortField("TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA",eval(hex(135))),  #not
used by tor client
        XShortField("TLS_DHE_RSA_WITH_AES_256_CBC_SHA",eval(hex(57))), #Preferred cipher
suite of Tor
        XShortField("TLS_DHE_DSS_WITH_AES_256_CBC_SHA",eval(hex(56))),
        XShortField("TLS_ECDH_RSA_WITH_AES_256_CBC_SHA",eval(hex(49167))),
        XShortField("TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA",eval(hex(49157))),
        #XShortField("TLS_RSA_WITH_CAMELLIA_256_CBC_SHA",eval(hex(132))),  #not  used
by tor client
        XShortField("TLS_RSA_WITH_AES_256_CBC_SHA",eval(hex(53))),
        XShortField("TLS_ECDHE_ECDSA_WITH_RC4_128_SHA",eval(hex(49159))),
        XShortField("TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA",eval(hex(49161))),
        XShortField("TLS_ECDHE_RSA_WITH_RC4_128_SHA",eval(hex(49169))),
        XShortField("TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA",eval(hex(49171))),
        #XShortField("TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA",eval(hex(69))), #not used
by tor client
        #XShortField("TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA",eval(hex(68))), #not used
by tor client
        XShortField("TLS_DHE_RSA_WITH_AES_128_CBC_SHA",eval(hex(51))), #Preferred cipher
suite of Tor
        XShortField("TLS_DHE_DSS_WITH_AES_128_CBC_SHA",eval(hex(50))),
        XShortField("TLS_ECDH_RSA_WITH_RC4_128_CBC_SHA",eval(hex(49164))),
        XShortField("TLS_ECDH_RSA_WITH_AES_128_CBC_SHA",eval(hex(49166))),
        XShortField("TLS_ECDH_ECDSA_WITH_RC4_128_SHA",eval(hex(49154))),
        XShortField("TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA",eval(hex(49156))),
```

```
        #XShortField("TLS_RSA_WITH_SEED_CBC_SHA",eval(hex(150))), #not used by tor client
        #XShortField("TLS_RSA_WITH_CAMELLIA_128_CBC_SHA",eval(hex(65))), #not used by
tor client
        XShortField("TLS_RSA_WITH_RC4_128_MD5",eval(hex(4))),
        XShortField("TLS_RSA_WITH_RC4_128_SHA",eval(hex(5))),
        XShortField("TLS_RSA_WITH_AES_128_CBC_SHA",eval(hex(47))), #TLS specification states
that implementations must support this
        XShortField("TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA",eval(hex(49160))),
        XShortField("TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA",eval(hex(49170))),
        XShortField("TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA",eval(hex(22))), #Preferred ci-
pher suite of Tor
        XShortField("TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA",eval(hex(19))),
        XShortField("TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA",eval(hex(49165))),
        XShortField("TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA",eval(hex(49155))),
        XShortField("SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA",eval(hex(65279))),
        XShortField("TLS_RSA_ECDSA_WITH_3DES_EDE_CBC_SHA",eval(hex(10))),
        XByteField("CompressionMethodsLength",1),
        XByteField("CompressionMethod",0),
        XShortField("ExtensionLength",0)
        #XShortField("Extension_Type_ServerName",0)
        #XShortField("Extension_ServerName_Length",0)
        ]


#This method processes random and length fields that are dependent on previous fields.
    def post_build(self, p, pay):
        if self.RecordLayerLength is None:
            l = len(p)-5
            p = p[:3]+struct.pack("!H", l)+p[5:]

        if self.HandShakeLength is None:
            l = len(p)-9
            p = p[:7]+struct.pack("!H", l)+p[9:]

        if self.GMTUnixTime is None:
            t = int(time.time())
            p = p[:11]+struct.pack("!I", t)+p[15:]

        if self.Random0 is None:
            r = random.getrandbits(32)
            p = p[:15]+struct.pack("!I", r)+p[19:]
        if self.Random1 is None:
```

```
            r = random.getrandbits(32)
            p = p[:19]+struct.pack("!I", r)+p[23:]
        if self.Random2 is None:
            r = random.getrandbits(32)
            p = p[:23]+struct.pack("!I", r)+p[27:]
        if self.Random3 is None:
            r = random.getrandbits(32)
            p = p[:27]+struct.pack("!I", r)+p[31:]
        if self.Random4 is None:
            r = random.getrandbits(32)
            p = p[:31]+struct.pack("!I", r)+p[35:]
        if self.Random5 is None:
            r = random.getrandbits(32)
            p = p[:35]+struct.pack("!I", r)+p[39:]
        if self.Random6 is None:
            r = random.getrandbits(32)
            p = p[:39]+struct.pack("!I", r)+p[43:]

        if self.CipherSuitesLength is None:
            l = 56 #70
            p = p[:44]+struct.pack("!H", l)+p[46:]

        #if self.ExtensionLength is None:
            #l = len(p)
            #p = p[:121+struct.pack("!H", l)+p[123:]

        return p

#This class creates a Client Key Exchange in the TLS handshake
class TLSC2(Packet):
    #First we describe the packet by using the Scapy interface
    name = T̈LS 1.0¨
    fields_desc=[
        ByteEnumField("ContentType",22,
            {20: "Change Cipher Spec", 21:"Alert", 22:"Handshake", 25:"Application Data"}),
        ShortEnumField("Version",769,
            {768:"TLS 1.0", 769:"TLS 1.1", 770:"TLS 1.2"}),
        XShortField("RecordLayerLength",None),
        ByteEnumField("HandshakeType",16,
            {0: "Hello Request", 1:"Client hello", 2:"Server Hello", 11:"Certificate", 12: "Server Key
Exchange", 13: "Certificate Request", 14: "Server Hello Done", 15:
"Certificate Verify", 16: "Client Key Exchange", 20: "Finsihed"}),
```

```
        XByteField("HandShakeLengthNull",0), #Hack to make Handshake length three bytes
        XShortField("HandShakeLength",None),
        StrField("Key",None),
        ]

        #This method processes random and length fields that are dependent on previous fields.
    def post_build(self, p, pay):
        if self.Key is None:
            r=[]
        for i in range(0,397):
            r.append(hex(random.randint(0,32)))
            "".join(r)
            p = p[:11]+str(r)+p[15:]

        if self.RecordLayerLength is None:
            l = len(p)-5
            p = p[:3]+struct.pack("!H", l)+p[5:]

        if self.HandShakeLength is None:
            l = len(p)-5
            p = p[:7]+struct.pack("!H", l)+p[9:]

        return p
#This tells Scapy to bind TLS to TCP and port number to parse properly
bind_layers(TCP, TLS, sport=443)
bind_layers(TCP, TLS, dport=443)

#This method test the script to a given ip-address
def test():
    #Destination ip-address
    ip=IP(dst="192.168.198.164")
    SYN=TCP(sport=34587, dport=443, flags="S", seq=100)
    #First half of the TCP handshake
    SYNACK=sr1(ip/SYN)

    #Second half of the TCP handshake
    myack = SYNACK.seq + 1
    ACK=TCP(sport=34587, dport=443, flags="A", seq=101, ack=myack)
    send(ip/ACK)

    #Send the TLS Client hello
    PUSH=TCP(sport=34587, dport=443, flags="PA", seq=101, ack=myack)/TLS()
```

```
send(ip/PUSH)

#Wait for the server
time.sleep(0.1)

#Send the TLS client key exchange
PUSH=TCP(sport=34587, dport=443, flags="PA", seq=100, ack=myack)/TLSC2()
send(ip/PUSH)

#Reset the connection to end the conversation
RST=TCP(sport=34587, dport=443, flags="R", seq=103, ack=myack)/TLS()
send(ip/RST)

return None

if __name__ == "__main__":
interact(mydict=globals(), mybanner="ScapyTls")
```

D Snort signature file

$Id: local.rules $ # ——————- # LOCAL RULES # ——————- # This file intentionally does not come with signatures. Put your local # additions here.

#alert tcp any any -> any [80,443,9001] (msg: "Possible Tor circuit creation: Client hello cipher suite"; content: "|c0 0a c0 14 00 39 00 38 c0 0f c0 05 00 35 c0 07 c0 09 c0 11 c0 13 00 33 00 32 c0 0c c0 0e c0 02 c0 04 00 04 00 05 00 2f c0 08 c0 12 00 16 00 13 c0 0d c0 03 fe ff 00 0a|"; sid: 1000002;)

#alert tcp any [443,9001] -> $HOME_NET any (msg: "Possible Tor circuit creation: Certificate chain length = 1"; content: "|30 0d 06 09|"; offset:80; content:!"|30 0d 06 09|"; distance:400; sid: 1000003;)

#Rule triggers on certificate validity not before or after 10-07-22 (2010 july 22) #alert tcp any [443,9001] -> $HOME_NET any (msg: "Possible Tor circuit creation: Tor certificate validity"; content: "|17 0d 31 30 30 37 32 32|"; offset:110; content: "|17 0d 31 30 30 37 32 32|"; distance:7; sid: 1000004;)

alert tcp any [443,9001] -> $HOME_NET any (msg: "Possible Tor circuit creation: Certificate extension = 0"; content: "|30 0d 06 09|"; offset:80; content: !"|55 1d|"; offset:250; sid: 1000005;)