Anonymity with onion routing and Tor

Jan Šubelj

University of Ljubljana
Faculty of Computer and Information Science
E-mail: js3445@student.uni-lj.si

Abstract. Today, the onion routing service is one of the most used anonymizer services. Readers would usually associate Tor with illegal activities but in reality, as much as 95 % of the Tor network is used for normal day to day operations. This article grants the reader an in-depth overview of the onion routing protocol and its implementation in Tor. I will also touch on the implementation of hidden services in Tor and some attacks on Tor.

Keywords: Onion routing, Tor, Anonymity, Security

Anonimnost s čebulnim usmerjanjem in Tor

Storitev čebulnega usmerjanja, ki je implementirana v Tor-u, je trenutno ena izmed najbolj uporabljenih storitev anonimizacije. Čeprav veliko bralcev asociira Tor z ilegalnimi aktivnostmi, v resnici 95 % uporabnikov Tor uporablja za normalne vsakodnevne aktivnosti. Ta članek omogoča bralcu pregled čebulnega usmerjanja in implementacijo v Tor-u. Dotaknil se bom tudi implementacije skritih storitev v Tor-u in par napadov na Tor.

1 INTRODUCTION

We are bombarded with information about governments tracking our every online move on a daily basis. Nowadays, almost every company tracks both the online and social media activity of their employees, especially in the workplace. It is not unheard of for employees being fired because their political views differ from that of the leadership [1]. Therefore, more and more people are looking for ways to stay truly anonymous on the internet. The solution: anonymity providing services. One of the most used anonymity providing service is Tor (The Onion Router), which is based on the onion routing protocol. In this article, I will cover onion routing as described in [2] and how is it implemented in Tor. I will also cover hidden services in Tor and some attacks on Tor.

2 ONION ROUTING PROTOCOL (ORP)

In this chapter, I will cover the onion routing protocol as described in the article Anonymous Connections and Onion Routing from Naval Research Laboratory [2].
The first part of the Onion routing protocol is establishing a connection. Instead of sockets making connections to the resource provider directly, Onion routing makes a connection to the service through several special types of routers called Onion routers (OR). These routers are located throughout the world and in essence work as proxies.

The other part of the protocol is hiding information in an envelope kind-of way. This is done through layers of encryption, which is where the ORP gets its name from.

2.1 Protocol entities

For reader to understand the ORP, we first have to define entities through which the data stream is anonymized. I will start with the innermost interface, the application proxy.

2.1.1 Application proxy

Connection between the application proxy and an application wanting to use the onion routing network is application specific. Connection between the application proxy and the onion proxy is defined as follows. When the application proxy receives data to be sent via the onion network it first checks whether it supports the needed protocol, if not, it returns an error to the application. If it supports the protocol, it sends the standard structure to the onion proxy with regard to the following data:

- version of the onion routing protocol (1 B)
- the transmission protocol (e. g. STMP, HTTP etc.) (1 B)
- the retry count which states how many times the end node should retry connecting with the ultimate destination (1 B)
- the type of address format that will be forwarded to the onion proxy (1 B)

After this standard structure, the IP address and port number are sent in the format specified in the last field. Application proxy then waits for a 1-byte error code before sending the data to the onion proxy.

2.1.2 Onion proxy

Receiving the standard structure from the application proxy, the onion proxy can decide to accept or reject data. Its decision is sent to the application proxy with a 1-byte error code. If the request is accepted, it starts to build a packet called onion that is used to build the onion routing circuit. Each layer of an onion is encrypted with a public key of a receiving onion router. In an onion layer there is data for the receiving onion router regarding information concerning data exchange. More information about onions and onion routers is provided below.

2.1.3 Onion router

Like with proxies the onion routers forward data, however, in the onion routing protocol they also encrypt or decrypt received data. We could view the onion routing network as chaining proxies with additional cryptographic functions.

When an onion router joins the network, it contacts its neighbors and exchanges long lasting keys with them. The onion routers are identified by an IP and port number, so that a single machine could run multiple instances of an onion router. In order to exchange keys with a neighboring onion router, an initiating onion router sends its IP and port number to the neighbor to identify itself. The key exchange phase begins with Station-To-Station key agreement which generates two 56-bit DES keys. Communication between onion routers is then done with DES in OFB mode with IV set to 0.

Onion routers communicate with each other through packets called cells. Cells have a fixed length. Its head includes information regarding the anonymous connection identifier (ACI), type of cell and data length. Body of the cell carries the payload. There are four types of cells in onion routing (PADDING, CREATE, DATA, DESTROY). ACI and command fields are always

encrypted with the key agreed between the neighboring onion routers. The length and payload fields are encrypted with that key if cells are of type DESTROY or PADDING. However, if the cell is of type CREATE then only the length field is encrypted with the previously mentioned key. Each onion router also randomly reorders cells received in a fixed time frame, so that it makes it difficult for data analysis to break anonymity. The order of data for each anonymous connection is preserved.

2.1.3.1 Types of cells

The padding cell is used to inject confusion for data analysis into a longstanding communication and is dropped on receipt.

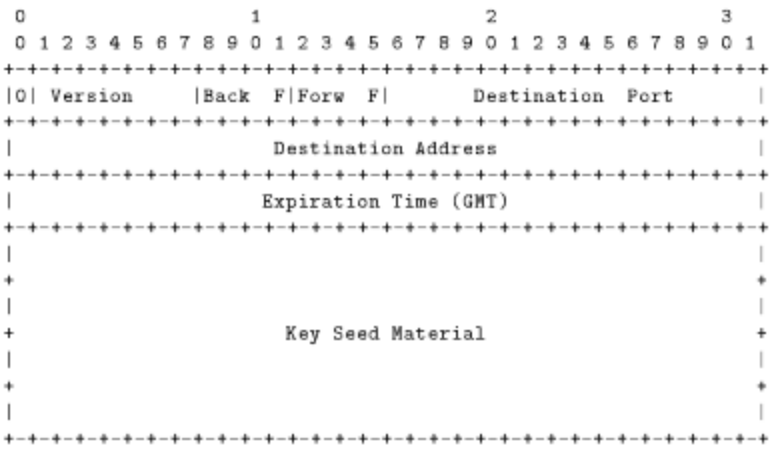
The create cell is used for receiving and transmitting onions through the network. When an onion is received, the onion router decrypts the onion's first 128 bytes with its private key, checks that it is not expired or replayed, and then generates the required keys, incoming and outgoing crypto engines and generates a new ACI for the next router. It stores the incoming crypto engine and key under the received ACI and the outgoing crypto engine and key under the generated ACI for the next router. It also maps the two ACIs together so that it knows where to forward the incoming and outgoing data. Then the onion router decrypts the rest of the onion with the first generated key, randomly pads it to the original length, puts it into the create cell and then forwards it on to the next onion router. Key generation and specific details about onions are described in subchapter *2.1.4*.

The data cell is used to relay data between an onion proxy and the service. If the data cell received by the onion router is sent in the direction from the onion proxy to the end address, it is decrypted with the crypto engine and key initialized with the onion that created the circuit. If the data is sent from the end address to the onion proxy, it is encrypted with the crypto engine and key initialized with the onion. The onion router does not actually need to know the way data is sent, because the crypto engine, the key and the forwarding ACI are stored under the ACI set by the onion. Therefore, in reality, when the onion router receives a data cell, it looks up the cell's ACI and uses its associated crypto engine with the key on its payload and length. It then forwards it on to the associated router (looks up the forwarding ACI).

The destroy cell is used to break the anonymous connection. When the cell is received, the onion router must send the destroy cell to the forwarding address of the designated ACI. When it receives the confirmation that the connection was destroyed, it also purges the data under the designated ACI and sends the confirmation to the sender. From that point on it can use that ACI for other anonymous connections.

2.1.4 Onion

Onions are used to create circuits. An onion has multiple layers. Each layer includes the transmitting and receiving information for an onion router in the circuit. Picture 1 shows what data an onion layer includes.



Picture 1: Structure of a layer of an onion

For an onion to be created, the onion proxy must first choose onion routers (at least three) with which it will build the circuit. Data regarding onion routers and their public keys is stored in a public table. The onion proxy then creates a layer for each onion router. In the layer, it stores a value that denotes the cryptographic function for data moving from it (Forw F) and towards it (Back F). It then stores the next node destination address and port number. The rest of the onion will be forwarded to that address which will be used by the onion router to forward and receive data. If the IP and port number are set to zero, it means that the receiving onion router is last in the circuit and its job is to relay data to the ultimate destination with the specified protocol. Then the expiration date is set. In the last field, the random key seed material is set.

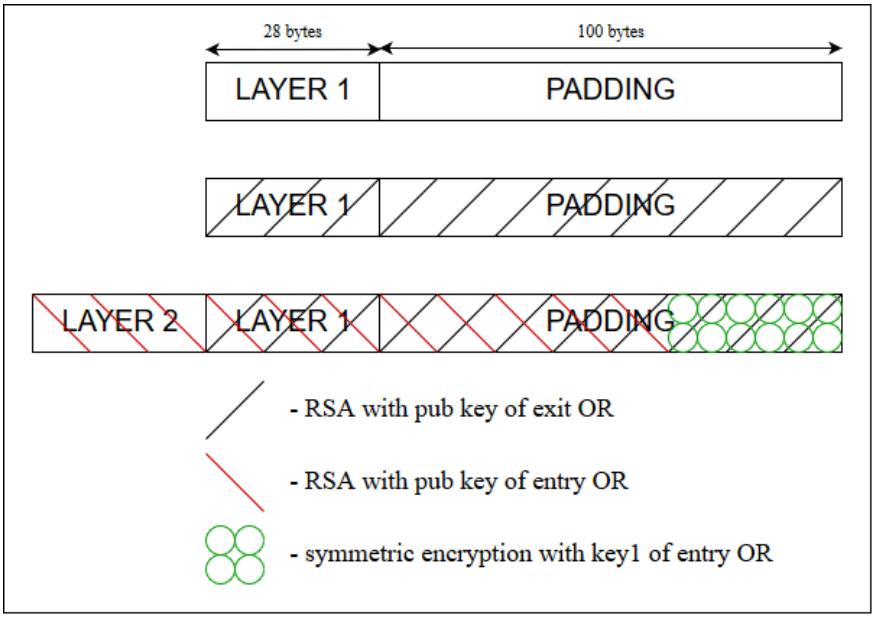
2.1.4.1 Key derivation

Keys are derived from the key seed material. Each onion router needs two keys in order to relay data. For relaying an onion additional key must be used. The three keys are derived from hashing the seed three times. The first hash (key1) is used for encrypting and decrypting the rest of an onion, the second hash (key2) is used for a key to encrypt data before sending it back to the onion proxy. The third hash (key3) is used to decrypt data before forwarding it to the next onion router.

2.1.4.2 Onion creation

To create an onion, we must know the public key of an onion router and its successor in a circuit. First, we create a layer for the outermost router. We generate the required structure of an onion layer and set the destination IP address and port number to zero. Then we pad the onion layer with 100 bytes (this is done only to the layer for the end node). We then encrypt the layer with the public key of the end node. It must be noted that the first bit of an onion is always set to zero, to enforce that the value encrypted is always numerically lower than the modulus of the public key encryption.

Now that we have the initial onion (first layer) we add other layers. For each onion router we generate a layer and set the IP address and port to the address and port of the routers successor. We then take the first 100 bytes of the already existing onion and append them to the generated layer. We then encrypt the layer and appended data with the public key of the onion router for which that layer is intended. We encrypt the remainder of an onion with the first key (key1) of the generated layer. Encrypted onion with two layers can be seen in Picture 2.



Picture 2: Encrypted two layer onion

2.2 Basic example

I will try to provide the reader with a basic understanding of how the onion protocol works through a simple example.

2.2.1 Initial step

Let us say we want to send message m from A to B. First the onion service initializes a route (also called a circuit) of onion routers. The circuit should always have at least three onion routers in it to provide anonymity. The addresses of onion routers are stored in a publicly accessible table.
In this example, A chooses three onion routers denoted:

- End onion router (EN_OR)
- Middle onion router (M_OR)
- Entry onion router (ENT_OR)

A then generates an onion as is presented in *2.1.4.2* and sends it to ENT_OR.

2.2.2 Creating circuit

When ENT_OR receives an onion, it decrypts its layer and initializes the required fields as described in *2.1.3.1* (create cell). It then decrypts the rest of the onion with key1 derived from its key seed layer, pads it to the original size and forwards it on to M_OR. M_OR does the same and forwards the last layer of the onion to EN_OR. EN_OR processes the remaining layer and waits for data. When the onion is processed, A sends the standard structure, described in *2.1.1*, to EN_OR via the created circuit. This data is already encrypted with the keys derived from the key seed. Now the EN_OR knows with what protocol and to what address and port (B) should it forward the data received from the built circuit.

2.2.3 Sending a message

To send a message, A must first encrypt the data with the key3 of EN_OR then with key3 of M_OR and lastly with key3 of ENT_OR. It sends the data to the ENT_OR which decrypts it with key3 stored under the cells ACI and forwards it on to M_OR. It does the same and forwards it to EN_OR. EN_OR then does the last decryption with its key3 and blindly sends the data to B using the protocol specified when the circuit was created.

2.2.4 Responding to a message

When responding to a message, the process goes in reverse. B sends a response to EN_OR. EN_OR then packages the response into a data cell and encrypts it with key2. It forwards it on to M_OR which does the same and forwards it on to ENT_OR. ENT_OR also encrypts the received cell with key2 and forwards it to A. A must then decrypt the cell with key2 of ENT_OR, then key2 of M_OR and lastly with key2 of EN_OR. The message is sent to the application interface in the specified protocol.

2.3 Dangers

The original protocol lacks integrity due to the fact that the attacker could change the cipher text for malicious intent. The user must also be aware that when using the onion routing service his identity could still be extracted from metadata. Applications could add public IP, email or other identifying data to metadata of a message which could then be extracted to identify the user. This could prove to be a problem if the user does not want to be identified by the server to which he is connecting. The overall connection through the network is still anonymous as long as the user first encrypts the data and then sends it to the onion routing service.

3 IMPLEMENTATION OF ONION ROUTING IN TOR

The onion router is an implementation of the onion routing protocol, however, it differs in order to avoid copyright and patent stealing. Tor actually improved the onion routing protocol and is known as the second generation of onion routing. In this chapter, I will talk about the differences between the first and second generation of ORP as described in [3].

3.1 Network design

In Tor, the onion routers no longer only have shared keys with its neighbors but the whole network is connected with a TLS connection which improves the integrity of cells that traverse the network. The TLS connection is done with ephemeral keys which guaranty the perfect forward secrecy. Tor also implements stronger cryptographic functions like AES in CTR mode instead of DES in OFB mode and agreed upon session keys between onion routers and end user with elliptic curve Diffie Hellman.

3.2 Just cells

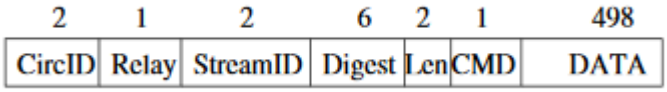
Tor no longer has a special packet called onion for circuit creation. Instead, they extended the types of cells into two categories: control and relay. They also renamed anonymous connection identifier (ACI) to CircID for ease of understanding.

3.2.1 Control cells

The control cells are always processed by the onion router that receives them. They are used for creation and destruction of a circuit. The control cells keep the same layout as cells in original ORP. Commands of the control cells are: padding, create, created and destroy.

3.2.2 Relay cells

The relay cells are used for transporting the control cells or data to the next hop in the circuit. Header of the relay cell is also extended with additional fields. The whole relay cell is shown in Picture 3.



Picture 3: Relay cell

Because Tor uses the same TCP stream for many circuits, the StreamID is required to identify to what stream the cell belongs to. The digest field is used for digest and is encrypted along with Len, CMD and data. When a node gets the relay cell, it decrypts it and checks for digest. If digest is correct, then that node processes the cell otherwise it forwards the cell to the next node. If there is no node to forward the cell to, it issues the teardown of a circuit because there could be a man in the middle trying to change cells.

3.2.2.1 Types of relay cells

There are 11 commands for relay cells. We can seperate them by categorys.

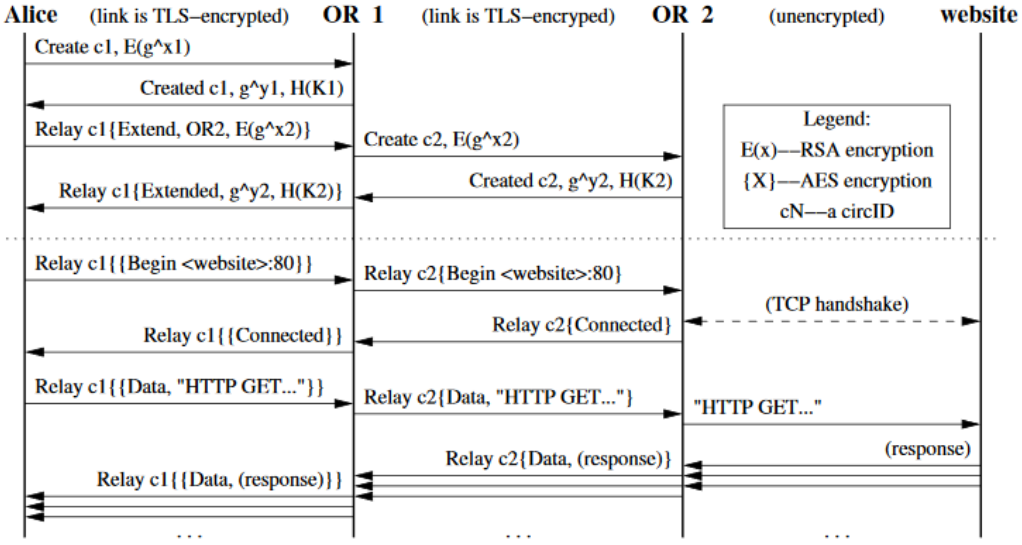
In the data relaying category there are: relay begin, relay connected, relay data, relay end and relay teardown. When relay begin cell is issued it means that the receaving node must connect to the web service with the address and protocol specified in the data field. Relay connected is issued when a node has succesfully connected to the service. Relay data is the most used relay cell and is ment to relay data between the web service and end user through the onion routing network. Relay end and relay teardown are similar commands. Both are used for closing and breaking down a circuit. Relay end is used to correctly close down the working circuit whilst relay teardown is used if the digest of a message does not match or if there are other problems on the circuit.

The control category has following relay cells: relay extend, relay extended, relay truncate, relay truncated, relay sendme and relay drop. Relay extend and relay extended are used to issue a command to extend the circuit by one node with the onion router specified in the data field. Relay truncate and relay truncated are used to teardown one part of the circuit and to confirm the teardown. Relay sendme is used in congestion control. Relay drop is used for implamentation of long range dummies.

3.3 Basic example

Because of the difference between Tor and the original ORP, I will try to provide the reader with a basic example of how Tor communication works. For a more visual reader, I also provided a picture of a simplified communication as seen on Picture 4. I will try to provide the reader with a basic understanding of how Tor works through a simple example. It must be stated that in reality

there are two keys for forward (key1) and backward (key2) communication as there are in the original ORP.



Picture 4: Simplyfied Tor connection

3.3.1 Initial step

Let us say we want to send message m from A to B. First A chooses an entry OR (OR1), preferably a trusted one, and agrees on a session key. Then A sends a message to OR1 to extend the circuit with another OR (OR2). With it, A also agrees on a shared secret, but this time OR2 doesn't know who A is because the message is relayed through OR1. Repeating this step, A sets up a circuit of N ORs and has N session keys for every OR in the circuit.

3.3.2 Sending a message

When the circuit is built, A encrypts the data with the Nth key then N-1th key and so on until all session keys are used. The encrypted message is then sent to OR1 which decrypts the message with his session key and forwards it to OR2. This process goes on until the last OR (Nth OR) gets the message, decrypts it, and processes it or, in this case, forwards it to B.

3.3.3 Responding to a message

When responding to a message the process goes in reverse. B sends a response to ORN. ORN then encrypts the replied data with its session key and forwards it to the N-1 OR. The latter also encrypts data with its session key and forwards it on. This process goes on until A gets the encrypted data. A then decrypts it in the correct order.

3.4 Integrity checks

One of the biggest problems of the original ORP is that it has no integrity check. This was corrected in Tor. They implemented an end to end integrity check which I already mentioned in *3.2.2*.

For a better understanding, I will use the example from *3.3*. To generate a message integrity code (MIC), A must take the agreed upon key with the last onion router in the circuit, ORN. It then appends the message to the key and hash it. The hash is then put into the digest part of the relay cell. Before A sends the cell, it encrypts it multiple times as stated above. The message is then sent and decrypted node by node. Each node checks the digest value and if it matches then that means that it is the last node in the circuit and that it should process the message. If the digest is incorrect and node has no forwarding address then that node issues a teardown cell. For stronger security, each new message is appended to previously sent messages before hashing. In order for the attacker to generate an authentic MIC, he has to know all of the messages sent including the shared key. Integrity check is done in both directions with different keys. For MIC from end user to service the key1 is used and for MIC from service to end user the key2 is used. Hashing is done with SHA1. Hash length extension attack in this case does not work because whole hash is encrypted when traversing the onion routing network and only the end user and end node have the decrypted value of a hash.

4 HIDDEN SERVICES

When a user wants to put up a service (e. g. a web server) and does not want anyone to know its location (reveal an IP address), the current set up does not provide him with anonymity. In the original ORP only the user was anonymous whilst the IP address of a server was known. To provide anonymity in both directions the researchers included hidden services into Tor.

In essence, a hidden service has set up N number of introduction points. These are normal onion routers and can change (any onion router can be an introduction point). User first chooses an onion router to work as a rendezvous point. User then relays the data about the rendezvous point to the hidden service via the introduction point. Then the user builds a circuit to the rendezvous point and, at the same time, so does the hidden service. When both circuits are built, the communication between the user and the hidden service can proceed. In this configuration, there are at least 7 onion routers in the circuit. Three from user to the rendezvous point, three from rendezvous point to the server and one is the rendezvous point.

4.1 Full example

In this subchapter, I will describe the full example of the connection to the hidden service as is described in the article Tor: The Second-Generation Onion Router [3]. In this example, A denotes the user that wants to talk to the hidden service, HS denotes the hidden service and RP denotes the rendezvous point.

4.1.1 Initial work of the hidden service

The hidden service must first generate a long-term RSA keypair with which it identifies itself and then randomly chooses introduction points. These are normal onion routers with which HS connects via Tor circuit. They start to act as an introduction point when HS passes them its public key. HS then creates an onion service descriptor that contains its public key and a summary of introduction points. The onion service descriptor is then signed with its private key. Descriptor is then stored in the lookup table and is periodically refreshed. Introduction points now wait for requests.

4.1.2 Connecting to a hidden service

For every hidden service a unique domain name is created. Domain name is a 16-character string derived from the public key of HS (e. g. 3g2upl4pq6kufc4m.onion). A gets the domain name of HS out of band, via a website or a HS owner tells the domain name to A. Then it retrives information regarding the introduction points from the lookup table. Afterwards, A randomly selects an onion router to act as an RP. It builds a circuit to it and gives it a randomly chosen rendezvous cookie. Cookie is used to identify and link the HS with A. Next, A connects to the introduction point of a HS, and sends it a message that includes a description of RP, the rendezvous cookie and the first part of Diffie Hellman key agreement. The whole message is encrypted with a public key of HS.

Introduction point then forwards this message to HS. If the HS wants to talk to A, it establishes a circuit to RP and gives it the rendezvous cookie. The RP connects HS to A, and now HS sends to A its part of Diffie Hellman and a hash of an agreed upon key. The whole connection is now secured and A starts the connection with a hidden service with relay begin cell.

5 ATTACKS ON TOR

In this chapter, I will describe a few attacks on Tor in general and attacks on hidden services. Many attacks exist and there is a constant race in finding an attack and preventing it. It should be said that Tor is not meant to provide anonymity against an end-to-end attack where an adversary controls all entry and exit nodes.

5.1 Passive attacks

Passive attacks are eavesdropping attacks where the attacker does not change, discard, replay or modify the data stream in any way. The attacker has only the power to listen to the stream and derive conclusions.

5.1.1 Content vulnerability

Tor is not meant to anonymize the data stream itself. It does not change the stream in any way. Also, it does not provide end-to-end secrecy. Therefore, if a user connects to a service that does not provide data secrecy (e. g. HTTP), the adversary that listens on the data stream between the end point and the service, it could gather data about the user. It could steal data sent from user to the service or the other way around.

If the user identifies to the server via login or any other identifying data (e. g. home address) then the eavesdropper could trivially authenticate the user.

Both of these attacks could be prevented by using end-to-end secrecy like HTTPS. This means that data is encrypted even before it is sent to the onion proxy and is still encrypted when exiting the exit node.

5.1.2 Timing correlation

If an adversary could listen between the onion proxy and the first onion router and between the last onion router and the service, it could correlate the incoming and outgoing packets by timing. Let us say that the user sends three packets to the service. Time between the first and the second is 2 ms and between second and third is 3 ms. The adversary sees 3 packets coming from the onion

proxy with this delay and later (e. g. 500 ms) it sees three packets with almost the same time delay leaving the end node and being delivered to the server. He can then speculate that these three packets are the same as the ones leaving the onion proxy. This means the anonymity is broken.

Even though this attack is possible, it is not necessarily probable because this means that an attacker would have to listen to a lot of connections all over the world. Note that nodes are picked at random and that the entry nodes are chosen from a trusted set of onion routers. Additional protection could be gained by securing the connection from the onion proxy to the first onion router (e. g. by setting it behind a firewall). The attacker would now have to separate cells originating from the onion proxy and cells that use the onion proxy as a relay node.

5.2 Active attacks

In an active attack, an adversary can modify traffic, set up its own onion routers, deploy denial of service attacks etc. These attacks usually need more resources and are generally deployed to authenticate a specific user.

5.2.1 Chain attack

If an attacker has power to get the session keys from onion routers, it could target every onion router in the circuit. If an attacker gets all session keys, then it could decrypt all messages and not only identify the user but also extract data. This could potentially prove to be very harmful for the user and the service.

This attack can work only if an attacker can break every single onion router in a circuit which means that they would have to have legal jurisdiction for all the locations where onion routers are placed, which could provide quite a challenge. Also, the chain has to be broken before the keys expire and a new circuit is generated. So, the attacker has 10 minutes (current lifetime of a circuit) to break every single onion router and get the session keys. This is unlikely to happen because the timing is quite short and onion routers in a chain are usually located in different countries which means that the attacker has to have a large amount of (legal) power.

5.2.2 Hostile onion routers

Because anyone could run onion routers the attacker could also run its own onion routers. Attacker can break the anonymity of a user if he controls the entry and end onion router. If the attacker controls m of N nodes then he can break the anonymity of maximum (m/N)^2 connections.

There is no solution for this attack because anyone could run onion routers. To diminish this attack, Tor runs a service that lists trusted onion routers. Note that it is advised to always use a trusted entry onion router.

5.2.3 Hostile web service

It is possible for a web service to be hostile (a legit website could be spoofed). It can persuade the user to input some kind of authentication and authenticate them that way. It is also possible that a program that is connected to Tor network sends authentication data in the header or metadata.

This kind of attack can be prevented by using additional programs that filter authentication data from data stream (e. g. Privoxy) before sending it to Tor network.

5.3 Attacks on hidden services

One of the greatest features of Tor are hidden services. This is also where most of the "dark web" lies. I will try to explain some of the attacks on hidden services. Even though attacks on hidden services are common, there is not yet a known attack that would disrupt hidden services all together.

5.3.1 Denial of service attacks

These attacks are meant to disrupt a hidden service so that it can no longer serve clients. An attacker could flood an introduction point with request for a hidden service. These requests could be filtered or blocked by the introduction point if a hidden service requires an authentication token. Note that this authentication token is still anonymous in nature because it does not provide any information about the user, only that he can access the hidden service.

Another denial of service attack is when an attacker disables introduction points with DDOS attack or other. This attack can be simply mitigated by a hidden service opening new introduction points. Also, introduction points do not need to be public and can be known only to those few users that are authorized to access the hidden service.

5.3.2 Compromised introduction point

If an attacker owns the introduction point of a hidden service it could flood the hidden service to prevent it from serving the users. This can be easily noticed and a hidden service can close the circuit and open a new introduction point.

Another example of an attack could be that an introduction point does not forward user's request to a hidden service. In this case, the hidden service must periodically check the introduction point so that it sends the request as a normal user and then check if it received a request.

Both of these attacks could be mitigated by opening multiple introduction points so even if one is compromised the chances of others being as well are quite slim.

5.3.3 Compromised rendezvous point

Because the rendezvous point is a normal onion router, an attacker does not gain anything if it controls only the rendezvous point. All the data relayed by the rendezvous point is encrypted by the session key agreed upon between the hidden service and the user so that an attacker cannot read the data stream nor can he identify the hidden service or the user because both are connected via a different anonymizing circuit.

6 CONCLUSION

In this article, I tried to provide the reader an in-depth overview of the onion routing protocol and its implementation in Tor. The attacks described in this article are only the basic attacks and elaborate attacks on Tor, which are being researched daily. Tor is currently still the most used anonymizer service and I doubt this will change anytime soon. For those who want to try browsing the web via Tor they can download the Tor web browser on https://www.torproject.org/. There, the reader can also access many interesting information about Tor usage and anonymity in general. For more intrigued readers I would recommend the articles referenced below.

7 REFERENCES

[1] "Google employee fired over diversity row considers legal action" *The Guardian*. [Online]. Available: https://www.theguardian.com/technology/2017/aug/08/ google-employee-fired-diversity-row-considers-legal-action-james-damore. [Accessed: 12. 1. 2018].
[2] M. G. Reed, P. F. Syverson and D. M. Goldschlag, "Anonymous connections and onion routing," in IEEE Journal on Selected Areas in Communications, vol. 16, no. 4, pp. 482-494, May 1998.
Available:http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=668972&isnumber=14639
[3] R. Dingledine, N. Mathewson, and P. Syverson. "Tor: The Second-Generation Onion Router" in Proceedings of the 13th USENIX Security Symposium, pp. 303-320, August 2004. Available: https://svn.torproject.org/svn/projects/design-paper/tor-design.pdf
[4] R. Dingledine, N. Mathewson, "Tor Protocol Specification", Tor Project [Online],
Available: https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt. [Accessed: 12. 1. 2018].

8 PICTURE REFERENCES

[1] M. G. Reed, P. F. Syverson and D. M. Goldschlag, "Anonymous connections and onion routing," in IEEE Journal on Selected Areas in Communications, vol. 16, no. 4, pp. 482-494, May 1998.
[3], [4] R. Dingledine, N. Mathewson, and P. Syverson. "Tor: The Second-Generation Onion Router" in Proceedings of the 13th USENIX Security Symposium, pp. 303-320, August 2004. Available: https://svn.torproject.org/svn/projects/design-paper/tor-design.pdf