FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Correlation Attacks on TOR
Student:	Jan Fajfer
Supervisor:	Ing. Josef Kokeš
Study Programme:	Informatics
Study Branch:	Computer Security and Information technology
Department:	Department of Computer Systems
Validity:	Until the end of summer semester 2018/19

Instructions

1) Study the TOR network - its purpose, principles, strong and weak points.
2) Research the current status of correlation attacks on TOR.
3) Demonstrate the execution of a correlation attack.
4) Discuss your results.
5) Propose countermeasures against correlation attacks.

References

Will be provided by the supervisor.

prof. Ing. Róbert Lórencz, CSc.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague January 20, 2018

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Correlation Attacks on Tor

Jan Fajfer

Department of Computer Systems
Supervisor: Ing. Josef Kokeš

May 14, 2018

Acknowledgements

I take this opportunity to express my sincere gratitude to my supervisor, Ing. Josef Kokeš, for his support and valuable guidance. I also wish to thank my friends and family for their continuous support and encouragement.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 14, 2018 ……

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Fajfer, Jan. *Correlation Attacks on Tor*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018.

Abstract

The primary goal of this bachelor's thesis is to describe and execute a correlation attack on the anonymity network Tor. It starts with an analysis of Tor's design, threat model, and attack vectors and continues with the analysis of the current state of correlation attacks on Tor and notable work in this field. The practical part contains execution of an attack using the Levine et al. and a modified Sun et al. method on the live Tor network. The main tests showed an overall good correlation with an average correlation coefficient r greater than 0.87 for both methods. The error rate was 5% with no false positives. The thesis continues with an analysis of factors influencing these attacks and concludes with a description and an analysis of countermeasures against end-to-end correlation attacks.

Keywords onion routing, Tor, correlation attacks, attack execution, countermeasures

Abstrakt

Primárním cílem této bakalářské práce je popsat a provést korelační útok na anonymizační síť Tor. První kapitola analyzuje design Toru, model a vektory útoku. Druhá kapitola popisuje aktuální stav korelačních útoků na Tor a významné práce v této oblasti. Praktická část popisuje testy, které byly uskutečněny pomocí dvou různých metod na korelační útoky. Hlavní testy ukázaly poměrně dobrou korelaci s průměrným korelačním koeficientem r větším než 0,87 pro obě metody. Chybovost byla 5 % s žádnými výsledky, které by špatně identifikovaly vybraného klienta připojeného k Toru. Kapitola pokračuje analýzou faktorů ovlivňujících tyto útoky. Práci zakončuje kapitola popisující a analyzující opatření proti korelačním útokům na Tor.

Klíčová slova onion routing, Tor, korelační útoky, provedení útoku, protiopatření

Contents

List of Figures

Introduction

With the rise of the Internet, there is also a high demand for anonymity when communicating online. An anonymity network called Tor introduced a simple and accessible option for hiding one's identity.

Tor is maintaining its popularity and provides anonymity for a wide variety of users. It allows anyone to share and access information on the Internet anonymously. It enables users to access websites blocked by their internet service providers. This anonymity network provides a relatively safe place for whistleblowers, dissidents, journalists, users sharing sensitive information, but also for non-ethical hackers, drug dealers or human traffickers. With this range of Tor users, it is inevitable for them or this network to be attacked with the intentions of revealing someone's identity.

There are many possible attack vectors. Tor covers some of them, but there are also several known attacks that can expose Tor users identity. A group of these attacks is called correlation attacks. These attacks are based on statistical correlation and traffic analysis. If an adversary can eavesdrop on specific spots where her victim's communication flows, she can analyze the traffic and guess the identity of someone using Tor even though the traffic is encrypted. However, these attacks could be difficult to execute and may require a significant amount of resources. There are also ways of defending against such attacks that could help protect Tor users.

Tor is evolving every day and so do the attacks on it, so it is necessary to keep track of new ways that adversaries could exploit someone's anonymity. Studying further attacks and their feasibility helps Tor developers in keeping this network anonymous and safe to use.

The goal of the analytical part is to describe the Tor network, its purpose, principles, strong and weak points, analyze its design, its security features, possible threats and attack vectors. The aim is also to research the current state of correlation attacks on Tor and to describe known methods that can

be used to execute a correlation attack.

The goal of the practical part is to demonstrate a correlation attack, discuss its result and factors that affect these types of attacks. And lastly, propose countermeasures against correlation attacks.

In the first chapter, I describe the Tor network, its purpose, and principles. I continue with explaining how communication through Tor works, how a circuit is constructed, the use of directory servers and the principle of rendezvous points and onion services. I end this chapter by discussing possible attack vectors on Tor.

In the next chapter, I analyze the current state of correlation attacks on Tor, methods of correlation and other notable work in this field.

In the practical part, I demonstrate the execution of a correlation attack using two of the methods described in the previous chapter. I continue this chapter with a description of the results and the factors affecting these attacks. In the end, I discuss resources needed to perform a successful correlation attack and the possible development of these attacks.

I continue with a chapter called countermeasures. I discuss measures to counter correlation attacks from the point of Tor developers and also from the position of a Tor user. I propose options and analyze the effectiveness of some of these countermeasures. Finally, I suggest and test an effective dummy traffic pattern that protects Tor users.

Chapter 1

Tor

Tor is a low-latency anonymous communication service that protects against basic traffic analysis attacks. Tor is now an open-source application maintained by a nonprofit organization called The Tor Project. The name Tor is an acronym for "The Onion Router" and is based on the principle of onion routing.[1]

Tor is suitable for a wide variety of users. It provides a solution for users of the Internet who want to keep their online activity hidden or users who do not want to share their Internet history with their ISP.[2] Tor also enables users to access sites which are restricted in the country the user is connecting from. Other examples are dissidents or whistleblowers sharing sensitive information with journalists. A branch of United States Navy and Law enforcement in the United States also uses this anonymous network. [1]

Not only clients but also servers can stay anonymous and not leak their IP addresses while still being accessible from the Internet through Tor. Onion services formerly known as hidden services offer a solution so Tor users can access a given server through so-called "rendezvous points." [2]

In 1995 a group of military scientist funded by the Naval Research Laboratory started working on the idea of onion routing, a plan which was to help protect the United States' intelligence from traffic analysis attacks. A year later, a proof of concept was launched. It was a single machine simulating five onion routers which mixed the incoming traffic before sending it to its destination to confuse the origins of the connection. In 1997 DARPA (The Defense Advanced Research Projects Agency) started funding the development of onion routing. In 1999 the team at Naval Research Laboratory published a paper describing specifications of the so-called generation 1 onion routing. After that, the work on onion routing was suspended for a little while,

[1] defined in section 1.2

[2] Internet service provider

and the generation 0 proof of concept was shut down. During its existence, it processed over 20 million requests from over 60 countries. In 2002, work on generation 2 onion routing began. The research continued while being funded mostly by the Office of Naval Research and DARPA. In 2003 the Tor network was deployed, and the Tor code was released under the MIT license. The next year, a paper called *Tor: The Second-Generation Onion Router* was published, describing the Tor design and the concept that is used until today. After that, the funding from the Office of Naval Research and DARPA ended. Nevertheless, the Naval Research Laboratory continued with the funding of only the work on location hidden services. The Electronic Frontier Foundation became the primary sponsor. By the end of 2004, the Tor network consisted of over 100 nodes located on three continents. [3, 4]

As of the time of writing, slight changes have been made but the design principles have stayed the same. The network's popularity rose and today contains over 6000 Tor relays [5] with a total bandwidth of almost 250 Gbit/s [6] and around 100 Gbits/s of it consumed every day for the past year [7].

Tor is now sponsored by many individual supporters but also organizations such as Google, Mozilla, National Science Foundation, United States Department of State Bureau of Democracy, Human Rights, and Labor, and others [8].

1.1 Goals and purpose

The initial goal of developing Tor was to create a low latency network which could provide anonymous online communication. It was designed to protect mainly TCP communication. When communicating using the TCP/IP protocols, the packet contains data, IP addresses, and ports of the source and the destination. And even if the data is encrypted, an attacker can still detect who both partners of the communication are, and based on the traffic analysis, can reveal more information such as the nature of their communication. Tor's primary goal is to prevent this from happening. In addition to that, Tor developers proposed some design goals in their 2004 paper [9].

- *Deployability*: Because the design of Tor relies on volunteers running individual relays, it should not be expensive and should be easy to install and to operate. Tor should not require the relay operators to state their identity. Furthermore, running a relay should not present a big burden to the operator.

- *Usability*: The Tor network gets safer with more users and more relay operators. Therefore it is essential for Tor to be user-friendly, as a hard-to-use system would possibly discourage potential users. As a client, it should also be easy to connect to Tor from any standard operating system.

- *Flexibility*: A protocol that is used in the network to communicate should be well specified and flexible so it could potentially be applied in other research regarding low latency anonymity networks.
- *Simple Design*: The design of the protocol should be easy to comprehend. It should distance itself from using unproven and complex features to maintain readability.

On the other hand, Tor **distances** itself from:

- Being *steganographic*. Tor does not aim to hide the fact that someone is connected to Tor.
- Being able to *provide protocol normalization*. Tor does not provide anonymity for someone communicating through this network using complex and variable protocols. Mainly because there are other tools, such as Privoxy, that can be used with Tor to provide this service.
- Being able to *protect against end-to-end attacks*. When an adversary is in a position of observing the connection between the client and the Tor network as well as the communication coming to the server, Tor does not guarantee the client's safety.
- Tor is not a *peer-to-peer network*, and as stated in the design papers [9], the solution that some other decentralized environments bring is "appealing, but still has many open problems."

  [1, 9, 10]

1.2 Design

The design of Tor is based on the idea of onion routing. Onion routing is a method where a client does not send data directly to a server but chooses a path through an onion network. First, the client connects to an entry node which is a part of the network. The communication between the client and the entry node is encrypted. Then the client connects all of the nodes in the path, creating a so-called circuit. The data that the client wants to send is encrypted several times wrapping the message into "layers." Along the way, onion routers unwrap the message layer by layer. That is why this type of routing is called an onion routing. The information that a server receives does not contain information about the client as it is sent from the last onion router in the circuit previously constructed.

Tor works very similarly. Let us say that a client C wants to access a server S without the server, her ISP or anyone listening to the communication knowing who is she connecting to. C sets up a connection to an entry onion

router (OR) in the Tor network. For C to identify which particular ORs are entry ORs, the client previously fetches a list of all nodes in the Tor network from a more trusted OR called a directory server which has a list of ORs with information about them. C then continues establishing a connection and negotiating keys for encryption with the middle OR in the circuit and lastly does the same with the exit OR. The default circuit that a client chooses through the Tor network consists of three ORs, but that can be modified. C then prepares a message or a request she wants to send the server, encrypts it using all the symmetric keys that she negotiated with the ORs in the circuit, and sends it to the first OR in the circuit, the entry node.

The entry node then decrypts the message using the negotiated key. The message still needs to be decrypted two more times to be readable by the server. Then the entry OR sends the message to the middle OR. The middle OR decrypts the message and sends it to the exit OR, which later decrypts the data which is now readable and sends it over to the server.

In the beginning, the message was encrypted three times over, so anyone eavesdropping between the client and the entry node can only see that client C is communicating with some onion router, but can not directly guess IP address or any other information about the other ORs or the server. The same applies to the communications between the remaining ORs involved in the circuit. Each OR only knows the identity of the directly adjacent ORs in the circuit, while the server S only knows the address of the exit node. And even though an attacker eavesdropping between the exit OR and the server S can read the message sent to the server, she cannot track the origins of that message as the TCP headers of packets she could capture only contain the IP address and ports from the exit OR. The final hop in the communication is therefore not encrypted by Tor. The connection can still be encrypted using HTTPS protocol. [9, 1] See figure 1.1.

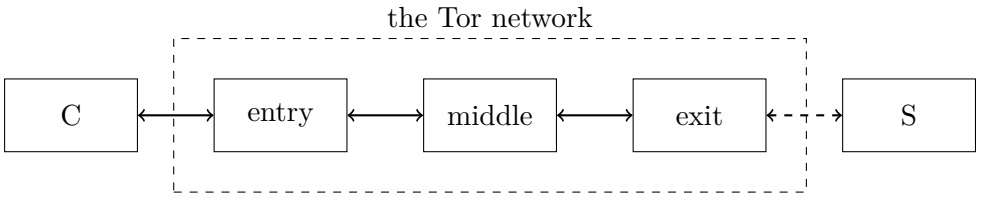

Figure 1.1: A client C connected through Tor to communicate with a server S anonymously. The dashed arrow from exit OR to S indicates traffic which is not necessarily encrypted.

1.3 Browser

The Tor browser is a software developed by The Tor Project. It is a modified Firefox ESR[3] web browser that helps users run a Tor client on their computer. With the help of the TorLauncher, the browser takes care of connecting to the Tor network, choosing a circuit and negotiating keys. It also contains several features that help users stay secure, such as NoScript or HTTPS Everywhere. [11]

The Tor Project provides other software and services helping users stay secure and anonymous, such as Tor messenger, Onionoo or a live operating system distribution called Tails. [12]

1.4 Protocol

Tor is an overlay network, which means, that ORs together create a virtual network that is built on top of another physical network, in this case, the Internet [13]. Individual ORs communicate with one another using the TLS protocol [14].

Each client runs a so-called onion proxy (OP) that handles the connections to ORs and communicates with the user applications using the SOCKS[4] protocol. Directory servers are relays that keep information about the network and provide it to OPs and other ORs, so they can connect and create a valid path through the network. [9]

1.4.1 Cells

The Tor protocol defines cells as a unit of communication. The primary goal of sending data through cells is to prevent basic traffic analysis attacks.

The developers at The Tor Project made some changes since the Tor protocol version 1, but the core of the design stayed the same. Moreover, new Tor protocols are compatible with the old ones.

All cells used to have a fixed size, but now the lengths may vary. All cells have a common format. The header starts with CircID, an identification to a communication stream between two ORs or an OP and an OR. A command that states the role of a given cell follows the identifier. Some of these commands and their values are CREATE (create a circuit), RELAY (send end-to-end data), DESTROY (stop using a circuit).

In a version 2 or higher, the command field is followed by a field that specifies the length of a cell's payload. [14]

[3] Extended Support Release
[4] Socket Secure

1.4.2 Circuits

Every Onion router uses several keys for its identification and communication. The "Identity key" is a 1024-bit RSA key used exclusively to sign certificates and other documents. It is used together with the Ed25519 "master identity key" as a unique identification of an OR.

There are two other keys used to manage operations such as the circuit creation and expansion called the "Onion keys." Every OR also keeps several other medium and short-term keys to manage TLS connections.

In the first design of Onion routing the OP created a new circuit for every TCP stream, but in the Tor design [9], several TCP streams share one circuit. This approach is better because it speeds up the connection, and helps to protect from traffic analysis and correlation attacks which are described in chapter 2.

In order to construct a circuit, the OP chooses ORs distinct from one another. These ORs will form the circuit. If the OP is not already connected to the first OR in the circuit (entry OR), the OP chooses an appropriate CircID and sends a CREATE cell to the entry OR.

The CREATE cell or the newer version CREATE2 cell is used to negotiate a symmetric key. The initial work on Tor [9] proposed a DH type handshake for negotiating the key. In this original, "TAP" handshake, a create cell carries the first part of the DH handshake (g^x). The entry OR responds with the second part of the DH handshake (g^y) and a hash of the final key (g^{xy}) encapsulated in a created cell.

Another type of handshake, called the "ntor" handshake, was introduced later on. This method uses several handshakes to create a set of shared keys.

After receiving the CREATED cell, the OP finishes the handshake and continues with the circuit creation.

Every circuit is constructed "one hop at a time," so when the initial handshake between OP and the entry OR (OR1) is completed, the OP sends an EXTEND cell to a subsequent OR in the circuit. The OP sends an EXTEND or EXTEND2 cell to OR1. Both of the extend-type cells carry the address of the node that the circuit should be extended to and data necessary for a handshake. OR1 extracts an "onion skin" from the received EXTENDED cell, which is a payload for a create cell. OR1 sends a create-type cell to the next OR (OR2). OR2 responds with a created-type cell. OR1 then wraps the data received by OR2 in an extended-type cell and sends it to OP.

Using this method, an OR only needs to know about the ORs next to it, it doesn't need to see an address of any other OR in the circuit or the OP, therefore, any OR can track the data received back only to its adjacent ORs.

As stated before, the whole circuit is always constructed one hop at a time,

and another OR can be added using the same method as described above.

Once the OP receives a CREATED cell from the exit OR indicating that the circuit has been constructed, actual communication between the client and the server can start.

The OP and the exit OR use RELAY cells to send data through the circuit. RELAY cells contain the specification of a relay command, a "Recognized" field that implies if the cell is at the OR it is destined for, a StreamID chosen by the OP and a digest which is used to check the validity of a cell. The cell also contains data to be delivered and its length without padding.

When constructing a RELAY cell, the OP encrypts both the relay header and the payload iteratively using the keys negotiated with the ORs in a given circuit. Therefore a digest set by the OP will have a meaningful value only when it reaches the targeted OR. When an OR sends a relay cell back to the OP, it encrypts its relay header and payload using the key shared with the OP. Thus when an exit OR sends a relay cell back to the OP, along the way, each OR encrypts the data, creating layers of encryption.

To initiate a TCP connection, an OP sends a RELAY_BEGIN cell, which contains the address and a port of the server. When an OR receives a relay cell, it decrypts it or encrypts it, depending on whether it is a communication flowing from client to server or in the opposite direction.

When the client decides to destroy the circuit, the OP sends a DESTROY cell to the entry OR. This OR closes the streams of that circuit and passes a destroy cell further down the circuit if it is not the exit OR. If the client wants to tear down just a part of the circuit, the OP sends a RELAY_TRUNCATE cell, which sends a DESTROY cell to the next OR in the circuit and replies with a RELAY_TRUNCATED cell. If a single OR in a circuit goes down, the OR closer to OP sends a RELAY_TRUNCATED cell and the OR further from OP sends a DESTROY cell to the next OR in the circuit. [9, 14]

1.4.3 Directory servers

Directory servers are a small group of trusted ORs in the Tor network. Each of these directory authorities serves as an HTTP server so that OPs can fetch information about the network and ORs can share their status with other authorities. The list of these authorities is hardcoded in the Tor software. There are ten currently running directory authorities [15].

Each of these has a long-term "Authority Identity key" which is distinct from the Identity key that each OR has. This key is used to sign "key certificate" documents, which contain an "authority signing key". The authority uses this signing key to sign other information.

These directory authorities get the information about each OR in the network from "router descriptors" which are signed by ORs and uploaded to the directory authorities. These router descriptors contain public keys, capabilities such as bandwidth and other information such as the exit policy of a given OR. ORs also serve as caches for directory servers.

Each directory authority periodically sends a "status vote" which contains the status of ORs in the network, and their descriptors to the other authorities. The authorities then compute a "consensus status" from the votes previously received. ORs serving as directory caches then download these consensus documents and store them. All OPs and ORs use the consensus documents to check whether their document is out-of-date. If it is, they download the needed "router descriptors." [16, 17, 9]

1.4.4 Onion services and rendezvouz points

Not only can Tor users connect to servers through Tor, keeping their identity secret, but they can also provide services which clients can access without them or the DNS server they are using knowing the IP address of a given server. Clients can use "rendezvous points" to communicate with a server or so-called "onion service." Six steps take place to set up an onion service and initiate a communication with a client as defined in [2].

1. The onion service generates a public and a private key. The public key will be used for its identification. The onion service randomly chooses some ORs as "introduction points" and creates a circuit to each one of them. It can add more introduction points later on. After that, the onion service tells its introduction points to forward connection requests from users to it.

2. After that, the onion service assembles a set of "onion service descriptors," signs them and uploads them to "hidden service directories," which are ORs that host signed onion service descriptors. These onion service descriptors include a public key of the onion service, a summary of each introduction point and a description of how to contact the onion service, which is now ready to communicate with Tor users.

3. When a client wants to contact the onion service, it first needs to set up a rendezvous point. A rendezvous point is a randomly chosen OR that the onion service builds a circuit to and sends it a one-time secret which is used to confirm the identity of a given onion service. The client then requests an onion service descriptor from a hidden service directory if it does not have it already or the one it has is not up-to-date.

4. The client then creates a circuit to one of the introduction points of the onion service and follows up with sending the OR an introduction

request to be passed to the onion service. The introduction request consists of the address of the target rendezvous point, the first part of a handshake and the one-time secret previously sent to the rendezvous point.

5. When the onion service receives the introduction request, it builds a circuit to the rendezvous point, sends it the one-time secret received with the introduction request and the second part of the handshake.

6. The rendezvous point then checks whether the one-time secret it received is the one from the client. If it is, the rendezvous point then connects the two circuits and verifies to the client that the onion service it is talking to is the correct one.

The client and onion service can now share a key and a circuit. To open a stream, the client sends a RELAY_BEGIN cell and continues with the communication as described above in 1.4.2. [18, 9, 2]

1.5 Strengths

There is a wide variety of users on Tor, and each has a reason why she wants to hide her identity. And whether the activity is lawful or not, with each user there is also a potential adversary who might wish to decrypt the user's communication or discover the final destination of the given connection. This chapter describes strengths of Tor and some attacks that the design should protect from.

The fact that Tor's design changed so little from the original is only adding to the point that the ideas outlined by Dingledine, Mathewson and Syverson in [9] are solid and the principles introduced serve its purpose.

Section 1.1 describes some of the initial goals of the Tor design which are now some of the strengths of the model. With its *deployability* Tor makes it easy for the volunteers to set up an OR and keep it running with minimal cost. *Usability* allows users to use the network without complications. A big help in this regard is the Tor browser. *Flexibility* and a *simple design of the protocol* allowed easier upgrades and backward compatibility. [9]

An attacker observing traffic in between a Tor user and a server should not be able to get any information from it as it is encrypted. Therefore the ISPs cannot discover what are their clients doing on the internet when using Tor. Also, any server that a user of Tor connects to should not be able to know her identity because of the circuit that is between the user and the server. [9]

Tor should also protect from a variety of active attacks such as the *DoS*[5] *non-observed nodes* attack. With the amount of ORs running every day, it

[5]Denial of Service

would be difficult to perform a DDoS[6] attack where an adversary would take down several relays to make someone connect through the part of the network that the adversary can observe. The integrity checks of cells also prevent *replacing contents of an unauthenticated* protocol such as HTTP or the *tagging attacks* where an adversary controlling an entry OR would mark a cell and match it to a corrupted content on the server's side. [9]

Tor's strength is also its popularity. Because Tor is relying on volunteers running relays and the more relays, the more users and ORs communicate through Tor, the harder it is for possible adversaries to detect user's communication in the network. And as Tor's popularity rose, the security increased as well. The popularity of the system also brings in new investors and funders who support The Tor Project financially. They provide the resources needed for maintenance and bug patching but also for further research and development of this network and other software that The Tor Project creates.

1.6 Weaknesses

Dingledine, Mathewson, and Syverson [9] proposed several attack vectors that Tor is vulnerable to. For the design to meet all of its initial goals, there need to be several compromises to its security that can pose a threat to Tor's users. In this chapter, I describe some of these vectors, mostly proposed in the original Tor design paper [9].

An ISP can guess whether its client is using Tor because the list of Tor relays is public. The solution to that could be using a Tor bridge, an extra entry node which is not listed in the public directory of ORs. These bridges are used mainly when an ISP blocks all Tor relays with the intention of blocking access to the anonymity network.

Website fingerprinting is an attack, where an adversary observes the pattern of traffic between the OP and the entry OR and based on that guesses with what website is the user communicating.

Correlation attacks exploit the low latency of the network. More about these attacks can be found in chapter 2.

An adversary can easily eavesdrop between an exit OR and a server and observe the communication. Therefore it is up to every user to make sure the content it is sending to the server is encrypted or cannot serve as a clue to revealing the user's identity.

An adversary who controls an exit node can *modify HTTP content* and misuse the traffic in any way. Moreover, Tor reuses a circuit for several TCP connections, so an adversary with possession of the given exit node can link anonymous and non-anonymous traffic together.

Learning OR's identity key could be a problem because an adversary could impersonate the OR by sending forged descriptors to the directory servers.

[6] Distributed Denial of Service

The possibility of this attack increases since Tor uses only 1024-bit key which has been declared not suitable by NIST[7] [19].

By *subverting majority of directory servers* an adversary could easily include her malicious ORs into the final "consensus status." However, this attack would be very difficult as the operators of directory servers expect an attack and are likely to be prepared. [9, 20]

1.7 Notable attacks

Over the years, Tor has been a target for many attacks. Some of the successful and notable ones are:

- The *Bad Apple attack* described by Le Blond et al. [21] exploits P2P applications to trace and profile Tor users. They took advantage of the linkability of multiple Tor streams sharing one circuit. The insecure application they used was BitTorrent.
- Murdoch and Danezis proposed an attack exploiting the low latency and linkability of two streams in their paper called *Low-Cost Traffic Analysis of Tor*. [22]
- Kwon et al. show how to detect users' involvement with an onion service using circuit fingerprinting. [23]
- *FOXACID* was an attack reportedly executed by NSA on users of Tor by modifying the browser bundle and distributing its malicious version. [24]
- Tor was also affected by the *OpenSSL Heartbleed Bug* since some of the ORs were running the vulnerable version of OpenSSL. [25]

These attacks did not directly exploit mistakes in the Tor design. They exploited low latency and other features that are a part of the design. Therefore the developers do not intend to change them. [20]

[7] National Institute of Standards and Technology

Chapter 2

Correlation attacks on Tor

Correlation attacks on Tor aim at uncovering someone's identity. The goal is to discover the originator of a communication stream. An adversary who observes the traffic on both ends of a connection can perform an end-to-end correlation attack. These attacks use statistical methods of correlation in order to confirm whether a stream originating with a client is the same stream reaching the destination that the adversary observes. That is why such attacks are also called the traffic confirmation attacks. [26]

2.1 Correlation

Before I describe some of the methods used when performing end-to-end correlation attacks, it is necessary to define several terms.

- *Covariance* determines the relation between two variables.
- *Correlation* is used when determining a relation between two variables. It is defined as "Tendency of two variables to increase or decrease together." [27]
- The *Pearson's correlation coefficient* gives a "measure of the strength of a linear association between two variables" [28] X and Y and is defined as

$$r_{X,Y} = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}} \tag{2.1}$$

  In this case measure the correaltion of two vectors of integers both with n elements, X_i is the i-th integer in vector X and Y_i is the i-th integer in the vector Y. $\overline{X}$ is the mean of vector X and $\overline{Y}$ is the mean of vector Y.

  Pearson's correlation coefficient can also be defined as covariance divided by the product of the standard deviations [29].

To determine whether the resulting correlation coefficient is statistically significant, the null hypothesis (H_0) that the two vectors have no correlation ($r_{X,Y} = 0$), is tested against an alternative hypothesis (H_A) that $r_{X,Y} \neq 0$.

When observing independent normally distributed pairs, in the case of H_0, the statistic

$$T = \frac{r_{X,Y}}{\sqrt{1 - r_{X,Y}^2}} \sqrt{n-2} \tag{2.2}$$

has the Student's t-distribution with $n - 2$ degrees of freedom.

Null hypothesis H_0 is then tested and rejected in a favor of alternate hypothesis H_A on significance level α if $|T| > t_{\frac{\alpha}{2},n-2}$. [30]

2.2 Motivation

In this section, I describe reasons why an attacker would want to execute an end-to-end correlation attack and the advantages of such attack.

When attacking Tor, the adversary faces several obstacles which make the attacks harder.

Tor prevents attacks using basic traffic analysis. When an adversary tries to find a relation between two given streams and at least one of them is within the Tor network, then the streams are going to differ. The reason for that is the fact that traffic within Tor is encrypted and encapsulated into cells. The data is therefore unreadable and the size of the packets also differ. There are also several hops before a packet sent by a client reaches a server. Due to several layers of encryption, an attacker can not guess the relationship between these two streams merely by inspecting the data themselves. Moreover, when observing the traffic flow, the attacker can see only the two adjacent ORs of the given stream and not the whole path of the traffic.

That is why end-to-end correlation attacks are preffered for attacking Tor. The data on each side that the attacker eavesdrops does not need to be the same. There just needs to be a positive correlation between the two streams.

There are several methods on how to correlate traffic. The adversary can essentially use any data from the data streams to try and compute a correlation between them. The most used methods use the timing of packets, the quantity of sent and received packets or the sizes of each one of them to perform a confirmation attack.

Another advantage is the fact that Tor does not intend to change its design goals and try to integrate some protection against these attacks.

Lastly, for a successful attack the adversary only needs to *observe* traffic in Tor. There is no need for active attacks on ORs, the client or the server if the attacker is in the right position.

2.3 Preconditions

Several conditions have to be met for an end-to-end correlation to be feasible.

Whether the attacker uses timing or size of the packets to perform an analysis, she always needs to be in the right position. At first, the attacker needs to be able to observe both ends of a communication. For example, an adversary observes traffic between a client C and a server S who are communicating using a circuit through the Tor network. The circuit consists of three ORs, OR1, the entry OR that C is directly connected to, OR2, the middle OR, and OR3, the exit OR which connects directly to S. The adversary needs to see the communication from C to OR1 and the traffic from OR3 to S. However, the attacker does not always need to observe both incoming and outgoing streams on both sides; this is described in section 2.4.3.

Timing correlation attacks are feasible when the latency of the network is low so that the time it takes for the packet to get from OR1 to OR3 is not significant enough to tamper with the statistical analysis. Therefore these types of attacks are ideal when eavesdropping on someone who is web browsing, which requires low latency. [31]

Correlation attacks that determine their success rate using the size of packets to correlate also rely on the fact that the size of a packet leaving C is not too different from the amount of data that S receives.

2.4 Methods

Since the launch of the Tor router, there have been several researchers issuing papers with new methods, ideas, and algorithms for end-to-end correlation attacks. In this section, I will describe some of them using the setup with client C, server S, and a circuit OR1-3 (onion routers 1-3) defined in section 2.3.

2.4.1 Packet counting

Back, Möller and Stiglic proposed a packet counting attack [32]. This attack suggests discovering the parts of a circuit one by one. In this method, an adversary counts the number of packets transmitted from C to OR1. After that, the adversary begins eavesdropping on connections from OR1, till she finds the second node in the circuit by counting the packets and comparing them to the connection from C to OR1. This method continues until reaching the server.[32]

This attack might have been feasible in the early version of Tor, because of the small amount of ORs in the network and their location not being distributed all around the world as it is today. However, it would be difficult to perform such an attack today as the attacker would have to be able to observe a significant amount of ORs all over the world. Moreover, with the number

of Tor users, the attacker would have to possess a greater computing power when attacking today as opposed to the time when Tor has launched. Finally, this attack suggested a great attack vector, but the method is not as efficient as other attacks I will describe further. [32]

2.4.2 Active attack

The difference between an active and a passive correlation attack is that when performing a passive attack, the adversary only observes the network and analyses the data collected, while an active correlation attack includes active interventions and manipulations with the traffic or the network.

In [22] Murdoch and Danezis proposed and described a correlation attack where an adversary controls a server and an OR. This attack aims to discover the relays which are a part of the circuit between the corrupt server and its victim. The corrupt server injects patterns into the communication with the victim. With the corrupt OR, the attacker makes a connection to a Tor relay in the network and using timing correlation techniques tries to decide whether the corrupt node is making a connection to a Tor relay which is a part of the circuit established between the corrupt server and its victim.

The correlation is defined using a function $S(t) = 1$ if the corrupt server was sending at a sample number t, $S(t) = 0$ otherwise.

Data from the probe is the measured latency at a sample t of the Tor relay which the corrupt OR is connected to. The correlation is then computed as a sum of the product of $S(t)$ and a normalized version of the probe data ($L'(t)$) divided by the sum of $S(t)$.

$$c = \frac{\sum S(t)L'(t)}{\sum(S(t))} \tag{2.3}$$

The results of this attack showed "good correlation between probe data in victim traffic." [22] However, the attack was tested on then not so popular Tor network which consisted of only thirteen nodes. Nowadays, with over 6000 Tor relays [5], the attack would be much harder to execute. [22]

2.4.3 Asymmetric traffic analysis

Sun et al. [33] presented a suite of new end-to-end correlation attacks, calling the rest of the attacks just a "tip of the iceberg." These attacks, called Raptor, are designed to be performed by a someone with the power over an Autonomous System[8] (AS). They also display the advantages that an AS level adversary has when possibly eavesdropping on Tor users.

[8] colection of networks administered by a single entity [34]

AS level adversaries can exploit the asymmetric nature of routing on the Internet, meaning that a path of a connection from an OP[9] to an entry node does not have to be always the same as the path from the entry node back to the OP and the same on the other side of the communication. This fact is not a liability for an attacker; it is the exact opposite. When using the proper traffic analysis technique, AS level adversaries have a better chance of executing an end-to-end correlation attack because of this nature.

Moreover, the natural routing updates in the Internet protocol (BGP churns) enable ASes to observe more users over time. When an AS can only observe a path from OP to an entry OR, such a churn can put an AS in the position to observe also the path from the exit OR to a server.

The asymmetric correlation analysis took into account the TCP Sequence number and the TCP Acknowledgment number fields from the captured packets. From these numbers, it is possible to compute a vector of transmitted data over time for the given data trace. And using the Spearman's rank correlation coefficient, it is possible to get the highest correlation between the given computed vectors. The last step is just selecting the two vectors with the highest correlation as a result.

Using live experiments, Sun et al. [33] showed that the asymmetric traffic analysis attacks have a 95% accuracy without any false positives. And with the analysis of historical BGP and Traceroute data, they were able to show that the threat of AS-level attacks increases by 50% for asymmetry and 100% for a routing churn.

In [33] Sun et al. showed that these attacks are feasible and pose a threat to Tor users. On the other hand, these attacks are not feasible for someone with just a partial view of an AS and limited resources.

2.4.4 Timing attacks

Timing attacks are end-to-end correlation attacks based on the timing of packets. These attacks use the timestamp of each packet captured on both sides of a communication. There are several approaches on how to correlate such data.

Levine et al. [35] described a simple attack where the correlation is computed using the difference between the timestamp of a packet and its successor. And if there is a correlation between the data computed on one side and the one on the other side, the two given streams are likely to be related. This approach, however, does not deal well with dropping packets. A dropped packet causes the correlation to be counted between non-matching packets. Therefore the times computed after the packet drop will be off and an otherwise perfect correlation could be marked as a mismatch.

[9] defined in section 1.4

Therefore Levine et al. proposed another attack which combined both counting and timing of packets. This attack includes sorting packets into time windows of the same width. These windows do not overlap and are adjacent to each other. All packets can, therefore, be sorted into these windows. The packets are placed into one time window according to the time stamp that states the time the adversary captured them. The quantity of packets in each window is then put together to form a vector. The result of this analysis is Pearson's correlation coefficient between the two vectors computed on each side.

Levine et al. [35] also made a suggestion that an adversary controlling either the entry node or the exit node can drop packets intentionally. If dropped at the right time, the packets could create a pattern which would increase the overall correlation and help differentiate the given stream from other ones.

The advantage of this attack is that it deals pretty easily with dropped packets and padding of Tor cells as it depends only on the count and timing of packets.

The disadvantage of this attack, as it is with most end-to-end correlation attacks, is that the adversary needs to be in an ideal position to eavesdrop on both ends of a given communication. And that is hard to achieve mainly because of the thousands of Tor relays that are now spread all over the world. It is true that they are not distributed evenly, but still the adversary would sometimes have to eavesdrop a communication on two continents at the same time to perform an end-to-end correlation attack.

Lastly, these attacks are easier to execute for AS level adversaries and attackers with the ability to control a lot of ORs or the ability to observe a significant portion of the Tor network.

Chapter 3

Execution of a correlation attack

In this chapter, I describe an execution of a correlation attack on Tor and its results. The method I chose to perform is the one proposed in Levine et al. [35] and described above. This attack is passive and feasible for anyone with limited resources. It deals well with dropped packets and padding of Tor cells. Moreover, when situated ideally, there is no need for an adversary to control any ORs.

Using a fixed traffic pattern I compare this method to a simple version of the traffic analysis method proposed by Sun et al. in [33].

My goal is to execute a correlation attack and show its applicability and relevance to other situations. Another goal is to compare the two methods for correlating traffic. And the last goal is to discover factors that influence end-to-end correlation attacks.

First, I define the methods I used. Next, I describe the tests' set-up and the situation it simulates. Next, I illustrate, how I performed the tests. In the following section, I analyze the results and discuss the conditions that affected the attacks. Finally, I discuss the feasibility of these attacks and propose several improvements.

There are two main categories of the tests I performed. The first one is just a simple client-server setup. The other one uses multiple clients to see if the correlation changes and is distinguishable between clients connecting to the server. After that, I performed few additional attacks to analyze the factors affecting the correlation.

Finally, I comment on the conditions that affected the testing and the results, their relevance and applicability.

3.1 Definitions of the methods

In this section, I describe the methods used for the correlation attacks. First I define the information, that is different for each method, and then I continue with definitions which apply to both methods.

3.1.1 L method

The first method I use to execute tests in this chapter is the Levine et al. method (L) which uses packet counting and the timing of each packet to create vectors.[10] I then compute the correlation of the vectors using the Pearson's correlation coefficient as proposed in [35]. Based on the result, I state whether there is a statistically significant positive correlation between the two given communication streams or not.

3.1.2 S method

The original method for traffic analysis by Sun et al. [33] (S) used Acknowledge and Sequence numbers in the TCP header to determine how much data was transferred over a specific stream. The tests I performed measured only the size of the data of each packet, which is the method from [33] assuming we see both directions of the communication. I extract the size of the TCP layer of each packet and add it to the i-th widow of a given vector according to the timestamp of the given packet.

3.1.3 Both methods

The following applies to both method L and S.

I chose the size of each time window (W) to be 10 s as proposed in Levine et al. The total time of eavesdropping (E) in seconds is divided by 10, creating a vector of $E/10 + 1$ elements.

The first element of the vector represents a W_0 from the start time of eavesdropping (B) to $B + 10$. The i-th time window W_i represents the time from $B + i * 10$ to $B + (i + 1) * 10$.

For correlating the stream in both methods, I used the method suggested in [35] which uses the Pearson's correlation coefficient. And even though Sun et al. in [33] compute the correlation using the Spearman's correlation rank, as they state in their paper, "other correlation metrics could also be applicable." [33]

The null hypothesis H_0 for two communication streams is: "The correlation between the two given streams is either zero or negative." The alternate hypothesis H_1 for two communication streams is: "There is a positive correlation between the two given streams." The significance level α I chose for

[10] see section 2.4.4

both methods is 10^{-4}. It is so low, because of the pattern of communication I choose to test. I need to distinguish traffic streams that match from those that are just similar.

I also only consider a positive correlation coefficient to indicate two matching streams. With the increase of packet count or size of the transmitted data, there should be an increase of the same quantity on the other side of the communication if the two streams are matching, and certainly not the exact opposite which represents a negative correlation. Therefore, the statistics $T > t_{\frac{\alpha}{2},n-2}$ needs to be true in order to reject H_0. Negative r will be displayed as 0.

3.2 Set-up

The tests should simulate a correlation attack on Tor. The setup is simple, an adversary eavesdrops a communication between a client and a server on each side of the Tor network. The attacker does not control any OR, the client or the server, and can only see encrypted traffic flowing from the client to OR1 and from OR3 to the server.

To simulate this, I used two separate notebooks (Toshiba Portégé) to simulate a client and a server. The server runs the Apache HTTP Server serving a 20MB file. The client communicates with the server using the Tor browser which is commonly used among Tor users. The communication between the client and the server was encrypted using a self-signed SSL certificate to simulate a connection to a secure server.

I performed three main sets of tests. The first one is one client communicating with a server. The second and the third sets are two and three clients communicating with the server. The goal of these tests is to match the client's communication stream to a stream between OR3 and the server, therefore confirming the identity of the client.

For most of the tests, I have not changed the entry node mainly because of the convenience. An OP keeps the same entry node usually for nine months for security reasons[11]. The test results should not be affected by this as I created a new circuit for every single test, which means OR2 and OR3 changed after every test.

The fact that I capture the traffic on the same devices that communicate with one another should not tamper with the correlation results too much as I expect the main effect on latency to be the connection through Tor. The same applies to the fact that both the client and the server are in the same local network.

[11] see section 4.2.2

3.3 Traffic capture and analysis

I captured the traffic on each machine separately using Wireshark on each of the devices and saved each communication in a .pcap file. This method allowed me to get back to specific recorded traffic and inspect it with different techniques and try computing a correlation with a different method on the same communication. On the other hand, when correlating, it would be more efficient to filter out the traffic and not store the unnecessary data. Filtering out the data while eavesdropping could be achieved using, for example, tcpdump or when sniffing the traffic using a router, features such as Cisco's NetFlow protocol should allow the same.

After that, I analyzed the traffic using the Scapy module in Python. At first I filtered out the packets that were not a part of the communication that I was correlating. Then I split the connection on each side into streams. Finally, I transferred each stream into a vector of numbers. For the method L, I counted packets in each time window from the first packet recorded to the last one. And for the S method, I computed the amount of data transferred in a window of time.

I then computed the correlation between these vectors using Pearson's correlation coefficient as defined in section 2.1 using no delay as suggested in Levine [35]. The Python script (analyze.py) which contains a detailed description of this procedure is provided on the enclosed CD.

3.4 Tests

In this section I describe the test's results and explain anomalies that occurred. Finally, I perform several additional tests to illustrate some of the factors affecting correlation. All of the tests' results are on the enclosed CD.

To simulate communication between a client and a server I chose a standard pattern that I use in all of the following tests if not stated otherwise.

The attacker starts eavesdropping. The adversary only listens to the communication of the targeted client (C1) on the side of the client. I suppose that the adversary tries to match the stream from C1 to a stream entering the server rather than identifying anyone communicating with the server. After approximately 10 s, the clients open up the Tor browser and the OPs construct circuits through Tor to the server. After all the circuits are created, the clients start to download a 20MB file. The clients start the download simultaneously. When the download finishes, the clients close the browser and the OPs close the circuits. Each client shuts the connection down after the download is finished and does not wait for the other clients. Approximately 10 seconds after the last client closes the connection to the server, the adversary stops listening to the traffic and analyzes the captured traffic.

3.4.1 Initial Tests

The first three tests did not involve a connection over Tor. The goal of them was to test how both methods deal with regular connections and see the level of correlation I get. The clients connected directly to the server. The only differences in the set-up were the location of the client and the browser that the client used. The client and server were in different local networks but still in the same country. The client used a regular internet browser and the communication pattern was the same as described above.

The results showed an almost perfect correlation in each test. The r was over 0.999 and H_0 was rejected in favor of H_1 on level of confidence 0.01% for both methods and tests 0_0, 0_1 and 0_2.

Because the latency was very low and the data packets did not change in size or count, the correlation was very high.

3.4.2 One client

The goal of this group of tests was to analyze the impact of latency in the Tor network on the correlation. Secondly, the goal was to compare the rate of correlation that I get using the two different methods.

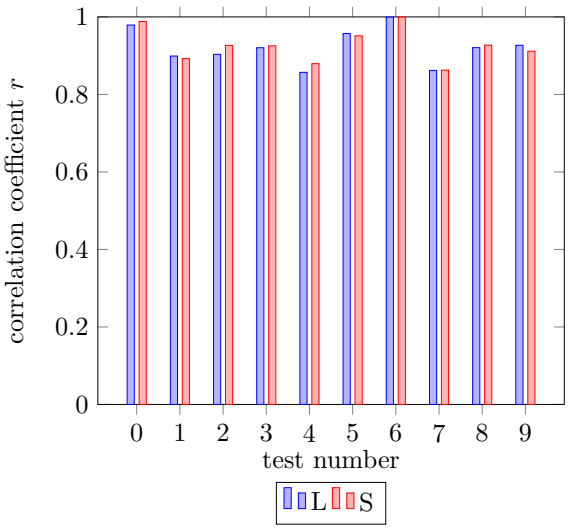

Figure 3.1: Tests 1_0 - 1_9 for C1

The connection through Tor made the correlation definitely worse than in the initial tests; however, the correlation is still good with the average values $r = 0.92$ for method L and $r = 0.93$ for the method S. H_0 was rejected in favor of H_1 on level of confidence 0.01% for both methods and all of the tests. Figure 3.1 shows results of the executed tests and a comparison between the two methods used.

The latency varied for all of the tests as I chose a different circuit for each of them. The circuits were always consisting of ORs from different countries and sometimes from different continents as well.

There was a significant difference in the bandwidth and latency depending on the chosen circuit. The download speeds ranged from 50 to 300 kB/s which significantly affected the total download time of the file.

There appears to be a relationship between the bandwidth and latency of a circuit and the correlation measured while communicating through the circuit. However, it is not significant as there also appears to be a relationship between the total time of eavesdropping and the correlation.

Overall, both methods yielded similar results, showing good correlation with no anomalies in all of the ten tests.

3.4.3 Two clients

The goal of this group of tests was to see if the models for correlating traffic can distinguish between two clients.

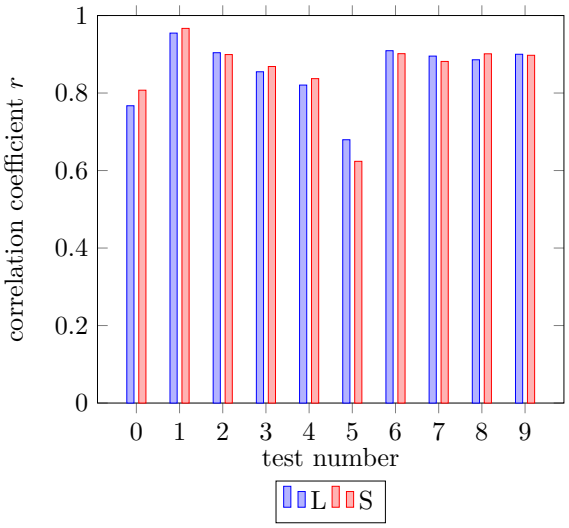

Figure 3.2: Tests 2_0 - 2_9 for C1

The communication pattern of both clients was similar and executed at the same time which should make the distinction harder. Despite that, the results showed a good correlation on the majority of the tests. The latency of the network created a pattern in the communication which made the two vectors distinct from one another, which led to a high correlation with C1 and significantly lower with the other client (C2). The distinctions in the resulted values for C1 and C2 were significant for a majority of the tests, with no false positives, which would be showing better correlation for the wrong client. The average r for C1 was 0.86 for both of the correlating methods and the average

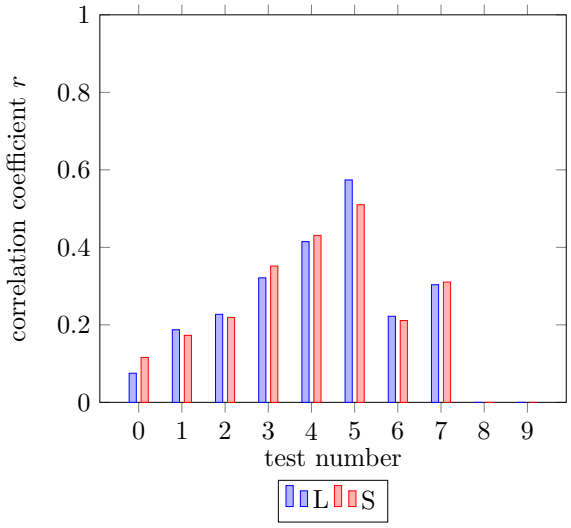

Figure 3.3: Tests 2_0 - 2_9 for C2

r for the dummy client was 0.21 for both of the methods as well. H_0 for C1 was rejected in favor of H_1 on a level of confidence 0.01% for both methods and all of the tests except for test 2_5. H_0 for C2 was not rejected in favor of H_1 on a level of confidence 0.01% for both methods and all of the tests including the test 2_5. Figures 3.2 and 3.3 show results for the two-client setup. In the figure 3.3 negative r is displayed as $r = 0$, this applies for all the following figures.

There were two tests, 2_0 and 2_5, where r for C1 was significantly lower than in other tests. This anomaly was probably due to the higher latency of the circuits created for these tests.

High latency modifies the pattern of the traffic, which could result in the following. The start of the download happens in the time window W_i on client's side, but on the server's side, the beginning of the download matches a time window W_{i+1}. The same could happen with the end of the download. Such a significant increase or decrease in traffic that matches a different time window can cause the resulted r to be lower.

Another factor was probably the slow creation of the circuit which resulted in traffic flow on the client's side and no traffic on the side of the server. And because of the small time frame that I measured the traffic in, the circuit creation took a significant time, which could tamper with the resulted correlation.

In test 2_5, the correlation coefficient r for C2 was higher than 0.5 for both of the methods. This was due to the fact that both downloads finished within 1 second of one another, making the traffic pattern a little more similar to one another than it was with the other tests.

3.4.4 Three clients

The goal of this group of tests was to see how does the attack deal with an additional client (C3). Another goal was to test if the circuit creation and keeping the circuit it alive affect the correlation. Therefore, for this attack, I did not always close the circuit after the download finished.

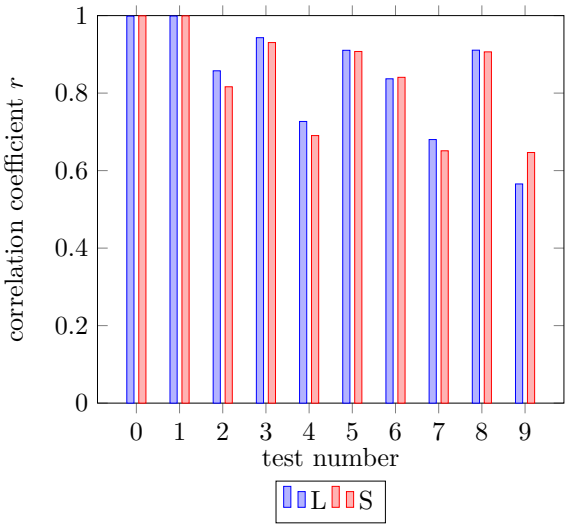

Figure 3.4: Tests 3_0 - 3_9 for C1

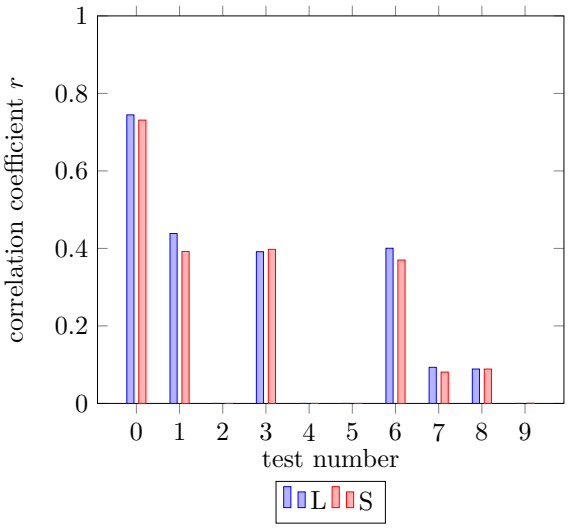

Figure 3.5: Tests 3_0 - 3_9 for C2

The results show a high correlation on the majority of the tests for both methods. The average r for the target client is 0.84 for both methods. The

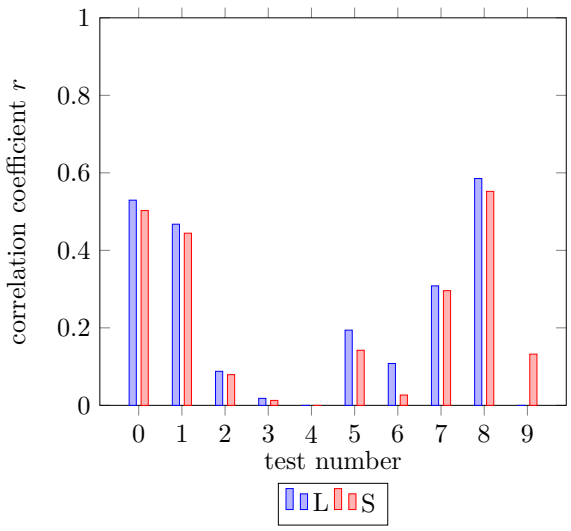

Figure 3.6: Tests 3_0 - 3_9 for C3

average r for C2 and C3 is less than 0.25 for both methods. H_0 for C1 was rejected in favor of H_1 on a level of confidence 0.01% for both methods and all of the tests. H_0 for C2 and C3 was not rejected in favor of H_1 on a level of confidence 0.01% for both methods and all of the tests except for C2's test 3_0 and C3's test 3_8. Figures 3.4, 3.5 and 3.6 show results for the three-client setup.

The first two tests 3_0 and 3_1 showed an almost perfect correlation, the download speed for these tests was fast, and each client closed the circuit immediately after the download finished.

In the rest of the tests, all the clients closed their circuits simultaneously after they all finished downloading, which could make the distinction between them worse. However, probably the main cause for the lower correlations in tests 3_4, 3_7 and 3_9 was the higher latency and lower bandwidth of the circuits created. These factors cause a slow download. Therefore, the simultaneous increase or decrease in traffic on both sides was not as significant which could cause the lower correlation.

The reason that H_0 was rejected for C2's test 3_0 and C3's test 3_8 was possibly the low latency of the given clients' circuit combined with the fact, that the differences in the total download times were within one time window, which could cause a higher correlation even for streams with no connection to one another.

3.4.5 Additional tests

Usually, when the latency of a circuit would be extremely slow that I would have to wait several minutes just to start the download, I would start the

test over and change the circuit. However, I performed two tests (4_0 and 4_1) where I continued with the test even when the latency of C2's circuit was significantly lower than the latency of the other circuits created. The slow download resulted in C1 finishing the download earlier and then just keeping the circuit alive but not communicating with the server. Thus C1 was creating traffic on one side of the communication and not on the other side. And because the download speed was so low, the impact of the circuit management on the correlation was even higher. This was one of the reasons why r for C1 and the test 4_0 was as low as 0.24 and 0.20 for method L and S respectively.

Test 4_1 for C1 had negative r for both methods. H_0 was not rejected for either of these tests and either of the methods.

Another cause for this low correlation could have been a number of dropped packets resulting in an increase of packet count and data transmitted on one side and not on the other side.

I performed several additional tests to see how much does just keeping the circuit alive affect the correlation. The setup was the same as in section 3.4.2 except that when the download concluded, C1 kept the circuit alive for ten minutes and after that ended the communication. The results showed a very good correlation with $r > 0.9$ for all the tests (5_0, 5_1, 5_2). The size of the data transmitted and the count of packets used to keep a circuit alive is not significant compared to the 20MB file download. Therefore the correlation was not affected. Also, the file download was fast for all of these tests, so the packets for circuit maintenance were irrelevant to the correlation in the small time frame.

The goal of tests 5_3 and 5_4 was the same as for test 5_0-2, only the communication pattern was a little different. C1 created and torn down three circuits before creating the one which would be used for the file download. After that, C1 torn down the original circuit. Lastly, C1 created and torn down a new circuit three times, after which the eavesdropping stopped. The OP, therefore, generated more traffic on the client's side which did not match any traffic on the server's side.

The results for tests 5_3 and 5_4 showed a very good correlation with $r > 0.9$ for both methods. H_0 for C1 was rejected in favor of H_1 on level of confidence 0.01% for both tests.

To see if a longer period of time can increase the correlation, I performed one circa two-hour long test (6_0). The test consisted of four 20MB file downloads. It started by downloading a file, then 5 minutes of keeping the circuit alive and then C1 destroyed the circuit and followed up with 20 minutes of waiting. C1 repeated this pattern four times. The results showed an almost perfect correlation for both methods. H_0 was rejected for test 6_0 and client C1 in favor of H_1 on a level of confidence 0.01%.

All of the tests' results are on the enclosed CD.

3.5 Results

The results showed an overall good correlation for most of the tests. Considering only the three main groups of tests ([1-3]_[0-9]), the overall average r was 0.874 and 0.875 for method L and S respectively. H_0 for C1 was rejected in favor of H_1 on a level of confidence 0.01% for 29 out of the 30 main tests. The error rate was 5% for both methods, which means, there were three times out of 60 where the given method supported the wrong hypothesis for a given client. However, there were no false positives.

The average r slightly decreased with the number of clients, which was probably due to the combination of more traffic on the server and circuits with higher overall latency. However, the number of clients should not affect the resulted correlation directly, because both method L and S dissect the traffic into streams. With the increase of clients, the adversary only needs to compute a correlation for more combinations of streams. On the other hand, a higher traffic on a server can definitely affect latency which can cause a lower correlation.

Methods L and S showed overall very similar results mainly because the traffic pattern that I used resulted in a majority of the cells sent by OP to be the same size. Thus there was a relatively linear relationship between the packet count and the size transmitted. In a more variable communication pattern, the relation between packet count and overall size of transmitted data would be a little different.

Finally, I tried to recreate what happened in tests 4_0 and 4_1 to get a better understanding of such low correlation results, but I was not able to achieve it even with slow connections and high latency circuits. And in the scope of the overall success rate, these two results seem to be more of an anomaly.

3.5.1 Feasibility and applicability

The results suggest that an attacker in an ideal position can perform an end-to-end correlation attack on Tor's users. However, depending on the position of the attacker and the traffic on the server, the amount of traffic to be analyzed can require a lot of computing power.

For an attacker with limited resources, it is crucial to find the best position for the attack so she does not have to correlate a lot of traffic for a long time. If the attacker does not have access to the traffic flowing directly to the server and the traffic flowing from the client, it could be difficult to get into the right position. Especially now, with the size of Tor's network, the probability for the targeted client to choose the attacker's corrupt node as an exit node is small.

For an AS-level adversary with a lot of resources the ability to observe a large amount of the Tor network can pose a significant threat to Tor users.

Especially if the adversary is not looking for a specific client-server pair but instead looks for anyone connecting to a particular server, the probability of finding such a client is significantly higher than for an attacker with limited resources.

3.5.2 Factors affecting the correlation

There are several factors affecting the end-to-end correlation attacks on Tor but it is difficult to evaluate exactly how much does each of them influence the correlation of the two given streams.

- The first one is **the Tor cells**, which tamper with the size and the count of a packet leaving the client. That results in a different size and number of packets that enter the server compared to the traffic leaving a client, which can lead to a decrease in correlation depending on the method used. However, depending on the nature of the traffic, sometimes there can be observed a linear relationship between the number and size of packets leaving a client and packets reaching a server, which was the case with the tests that I executed.

- **Circuits** that are used for communication through Tor can cause a higher **latency**. This factor possibly affects the correlation of the two given streams the most. As described in 3.4.3, high latency tampers with the correlation of two streams, especially when the latency changes throughout the connection.

- Operations **handling the circuit** proved not to be a significant factor when correlating streams of a file download. On the other hand, when the attacker does not download as much data, operations regarding the circuits can lower the correlation.

- **Time** can help correlation significantly as shown with the test 6_0.

- A traffic that is not linear but has a distinctive **pattern** is easier to correlate and better distinguishes the given stream from others.

- There are several factors affecting the **speed of the connection** through Tor. And as described in 3.4.4, download speed is also a factor that can influence the correlation.

To specify how much does each factor affect the correlation attacks on Tor would require further testing using different patterns and set-ups which is beyond the scope of this work.

3.5.3 Improvements

With the nature of Tor and its low latency as one of the key design features, the end-to-end attacks do have a future. In this section, I describe several ideas for future work and improvements of these attacks. However, I did not test any of these since it is beyond the scope of my work.

To study a pattern in connection with ORs could help automatically recognize traffic that operating the circuit creates. Cutting out this traffic from the overall computations could increase the success rate of correlation attacks, trying to correlate traffic that does not transfer a lot of data.

Study of correlation methods L, S and others using different types of traffic patterns such as simple web browsing, file download, upload or video streaming could help combine the advantages of each method and create one. This method could adjust to the type of the given connection and therefore be more durable to various kinds of connections.

Methods L and S could take into account the ORs that are involved in the targeted circuit. With the public information about all ORs, the method could adjust the time window size based on the circuit's latency or incorporate a delay into the correlation coefficient calculation according to the information gathered about the given OR.

If the adversary had control over several ORs, she could improve the methods for correlating. The attacker could filter out the streams used for correlation as she would use only the ones with matching middle ORs. The packets used for the circuit management would not be included in the correlation computation. The adversary would have a more accurate idea about the overall latency and bandwidth of the circuit, which could be used to adjust the methods for correlating as well.

Chapter 4

Countermeasures

The goal of this chapter is to propose countermeasures against end-to-end correlation attacks described in chapters 2 and 3.

I divided the countermeasures against correlation attacks into several categories. The first two categories consider the entity trying to prevent these attacks. It can be either an ordinary user of Tor or the Tor's developers themselves. Another way of analyzing these countermeasures is whether the protection is aiming at tampering the correlation itself or whether it aims at preventing the adversary from getting to the ideal position from which she could execute the attack.

I analyze the effectiveness of dummy traffic protection using the same setup as in chapter 3. Finally, I propose a pattern of dummy traffic that Tor users can use to protect themselves against end-to-end correlation attacks.

4.1 Users

There are several ways how an ordinary Tor user can prevent a successful end-to-end correlation attack.

4.1.1 Correlation

To prevent an attacker from executing a successful correlation attack, a client can try to tamper with the correlation of two streams.

The first thing that can be done is taking advantage of the fact that Tor does not split connections to different destination into different streams. Therefore a dummy traffic distorting the original communication pattern could decrease the correlation. It is hard for the adversary to distinguish the additional dummy traffic from the communication with the targeted server. However, if the adversary controls the given entry node, the client would have to make sure that the dummy traffic follows the communication stream to the

next hop in the circuit because otherwise the entry OR would recognize the targeted traffic from the dummy traffic.

Another way of protecting against these attacks is using busy nodes. As suggested by Adam Billman [36], choosing busy nodes as a part of the circuit increases the difficulty of correlating and requires more computing power from the given attacker.

4.1.2 Position

To prevent an adversary from even getting into the right position to correlate or make the eavesdropping harder, Tor users can apply several tips.

To lower the chances of using a corrupt exit node for a significant time, a client should switch the circuit frequently.

A client could only choose "more trusted" ORs to route the traffic through them, however, this poses more threat if one of these ORs would get corrupted.

Attackers who want to increase the chances for correlation attacks often target entry ORs. After that, they get the identity of a client connecting to Tor. Therefore using a proxy to connect to Tor could be helpful as the attacker cannot easily identify the client connecting through the proxy.

4.2 Tor Developers

Even though Tor distances itself from protecting against end-to-end correlation attacks, it already contains several features that help protect against these attacks.

4.2.1 Correlation

Tor could incorporate dummy traffic into the protocol. OPs[12] would generate dummy traffic automatically. The additional traffic between a client and an entry node in the given circuit could lower the correlation that an attacker would compute for two streams. However, on a small scale the dummy traffic may not be efficient, and on a bigger scale it could significantly slow down the connection.

Another modification to the Tor protocol could be a little random latent time added to each packet in the Tor network. It could make the correlation attacks less successful, but this adjustment may not be as efficient using just small delays. And longer packet delays could compromise one of the main goals of Tor, the low latency.

Defensive packet dropping or adaptive padding are methods suggested by Müller [37] that could help defend Tor's users from end-to-end correlation attacks.

[12] defined in section 1.4

4.2.2 Position

By keeping the same entry node by a Tor client for nine months decreases the chances of several traffic analysis attacks including end-to-end correlation attacks. [38]

According to the Tor project [39], Tor switches circuits to prevent a user to route traffic through a possibly corrupt node for a long time. The OP uses a single circuit for ten minutes before switching to a different one. However, Tor does not create multiple circuits for a single TCP stream but waits until the stream ends and switches to a new circuit immediately after that.

However, any kind of additional traffic or additional delay of packets that would Tor add to its network would go against its initial goals which made this anonymity network so popular.

Finally, perhaps the best defense is Tor's popularity. The fact that the Tor network consists of thousands of ORs and millions of users connect to it regularly [40] dramatically increases its overall security. The ability to route traffic over several continents can help protect even from AS-level adversaries. And the volume of traffic that is routed through the network makes it harder for a potential attacker to find the correct stream.

4.3 Dummy traffic tests

As described above in subsection 4.2.1, dummy traffic can help Tor users defend themselves against correlation attacks. I tested the effectiveness of several types of dummy traffic and proposed a specific pattern that is effective against these attacks.

For the following groups of tests I used the same setup as in section 3.2. I considered an adversary who does not control an OR but is in the right position to eavesdrop the traffic on client's side (X) and the traffic on server's side (Y). The adversary uses methods L[13] and S[14] to execute the attack.

4.3.1 Linear

To see how does dummy traffic produced by a client affect the correlation, I executed five tests. The primary goal was to test a simple linear dummy traffic that any user could generate. The setup is the same as in chapter 3, except that the client is additionally streaming a video from the start to finish of the test, which should simulate additional dummy traffic.

The results of tests 7_0 - 4 showed lower correlation than in the previous tests. However, the average was $r = 0.79$ with method L and $r = 0.80$ with method S which is still a good correlation. And H_0 for all tests and both

[13] defined in subsection 3.1.1
[14] defined in subsection 3.1.2

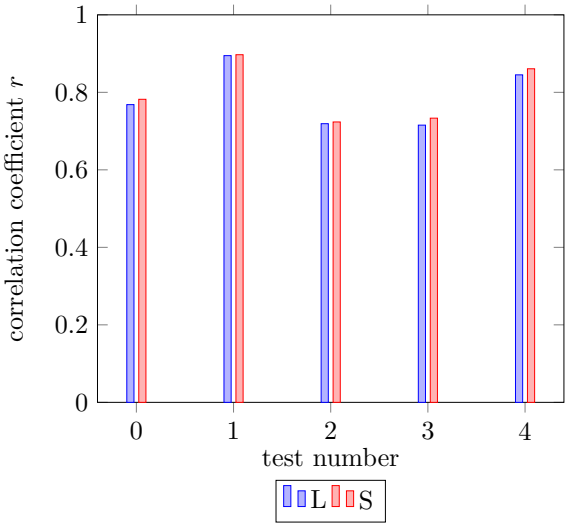

Figure 4.1: Tests 7_0 - 7_4 for C1 with linear dummy traffic

methods was rejected in favor of H_1 on a level of confidence 0.01%. Figure 4.1 shows results of tests with the linear dummy traffic.

4.3.2 Nonlinear

I executed another five tests 7_5 - 9. This communication pattern was the same as above with the exception that after the download finished, the client waited for 20 s, then began streaming a video for 20 s and did this three times over. The goal of these tests was to see whether such a pattern of traffic would affect the resulted correlation coefficient more than just a relatively linear dummy traffic.

The results showed an overall lower correlation than in the test above. This communication pattern resulted in avarage $r = 0.66$ with method L and $r = 0.67$ with method S. Moreover, H_0 for both methods and tests 7_6, 7_8 and 7_9 was *not rejected* in favor of H_1 on a level of confidence 0.01%. Figure 4.2 shows results of tests with the nonlinear dummy traffic.

The test above shows that a dummy traffic which is not linear can lower the correlation. The volume of the dummy traffic is also important, as a significantly lower volume of dummy traffic might not have any results.

On the other hand, the adversary who has the ability to observe both ends of a given communication could use the longer period of time to perform a correlation attack successfully if the dummy traffic would be used only while the client would communicate with a given server.

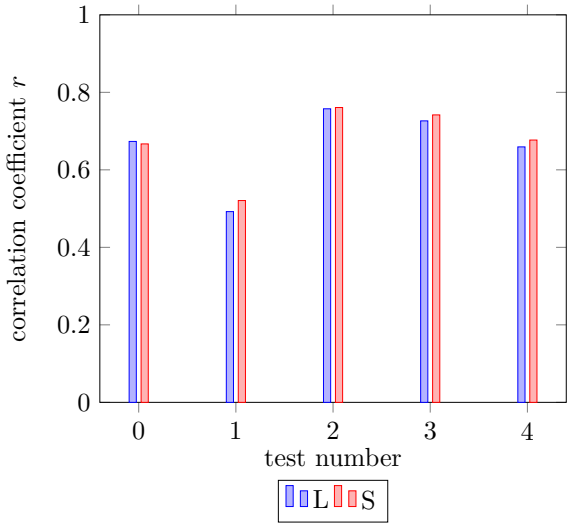

Figure 4.2: Tests 7_5 - 7_9 for C1 with non linear dummy traffic

4.3.3 Effective

Dummy traffic can protect against end-to-end correlation attacks and every Tor user has the possibility to use this protection. However, as shown above, not every pattern of dummy traffic ruins the possibility of correlating the two given streams. I will describe a dummy traffic pattern that could significantly increase a client's protection against correlation attacks.

Goal of the dummy traffic is to modify the traffic pattern of X and balance the overall size of data and number of packets transmitted so that the two streams X and Y on each side have a correlation close to zero.

To achieve that, the traffic patterns on each side should not appear to have any relation with one another. Therefore, with and increase of traffic flow on one side, there should not be a noticible increase in traffic on the other side. It would be dangerous if the dummy traffic created a completely opposite traffic pattern, as the adversary could notice it and suspect the use of dummy traffic by C.

The ideal dummy traffic can be simmulated by creating several dummy connections as follows.

1. At first, C creates a dummy connection D1 which transmits a similar or more significant amount of data and packets on avarage than the connection between C and S is going to generate. D1 should execute some random pattern.

2. After that, C creates another dummy connection D2, which should generate similar traffic as the upcoming connection between C and S. The longer these two dummy connections are maintained, the worse it is for

the adversary to succesfully execute the attack. However, it should last at least for several time windows and at least half of the expected time of communication between C and S to increase the impact of the dummy conections on the attack.

3. After that, C connects to S and at the same time ends D2. There should not be a significant increase in data and packets transmitted between C and OR1, however, betweeen OR3 and S the increase should be noticible.

4. C establishes a dummy connection D3 and ends connection with S at the same time. This should result in a decrease in data and packets transmitted over Y as opposed to X where the traffic flow should appear more linear.

5. Finally, the connection D1 should continue with the random pattern for several time windows and at least half of the time of communication between C and S. After that C ends all conections.

I executed tests 8_0 - 4 to analyze the effectiveness of this method. The connection D1 was a three times interrupted file download from a dummy server. D2 and D3 were 10MB file downloads from a dummy server.

On the server's side there usually occurred a short stream of just 20 packets within a one time window. I ignored streams with such a small lifespan as their relevancy was low considering the short-term attacks I performed.

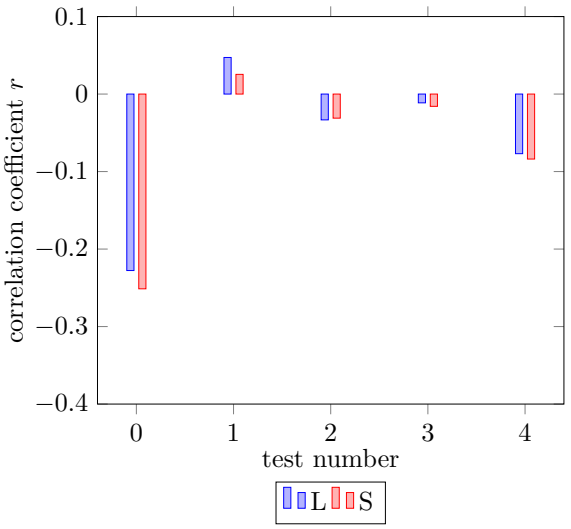

Figure 4.3: Tests 8_0 - 8_4 for C1 with effective dummy traffic protection

The results showed almost no correlation with an average r = -0.06 with method L and r = -0.07 with method S. As shown on figure 4.3 the r was

close to 0 for all of the tests. H_0 for both methods and tests 8_0 - 4 was *not rejected* in favor of H_1 on a level of confidence 0.01%.

The correlation was so low because D2 and D3 masked the beginning and the end of the download on the client's side. D1 created a random pattern on top of this linear connection that started with D2, continued with the download and ended with D3.

The results of these tests showed that dummy traffic in short-term eavesdropping attacks can be very effective. However, as stated above, attacks lasting for a longer period of time could find a better correlation. When selecting time series of hours, this dummy traffic pattern would not be as effective.

Therefore, if a user is connecting to a particular server frequently, she should consider switching the location she is connecting from which would make it harder for the adversary to match the given users' connections together.

Conclusion

The goal of the practical part of this thesis was to demonstrate a correlation attack, analyze what affects correlation attacks and propose countermeasures against these attacks.

I demonstrated the execution of a correlation attack using the Levine at al. method and a modified version of Sun et al. method. The execution was successful and showed that these attacks are feasible even with minimal resources. I achieved an average correlation coefficient r of 0.87 for the Levine et al. method and an average r of 0.88 for the modified Sun et al. method. The error rate was 5% for both methods with no false positives.

I analyzed the conditions that influence such attacks. In the short term attacks that I performed, the phase in which an onion proxy connects to an onion router proved to have some effect on the attacks I presented. However, this does not affect long-term attacks as I showed in a two-hour test 6_0 which showed an almost perfect correlation. Moreover, the latency difference that depends on the onion routers in a circuit proved to be among the more significant factors affecting these attacks.

In the last chapter, I proposed several countermeasures which could be executed from the point of a user connecting to Tor as well as countermeasures possibly implemented by the Tor developers. These countermeasures could be useful, especially when an adversary monitors the network for just a small period of time. I executed several additional tests to analyze the effectiveness of dummy traffic on end-to-end correlation attacks. The results showed that linear dummy traffic may not be sufficient enough and that a pattern within the dummy traffic might increase the protection. At the end of this chapter, I suggested and tested an effective dummy traffic pattern which protects Tor users. The results using this method showed minimal correlation with $r < 0.1$ for all of the tests.

Finally, the end-to-end correlation attacks exploit low latency which is a part of the Tor design that Tor's developers will most probably never change. Therefore, as long as the original Tor design does not change, these attacks

will be feasible in some shape or form.

Tor users can limit the chance of someone detecting their activity online by trying to defend against these attacks using dummy traffic, but the Tor's popularity and the ability to route each user's traffic around the world is perhaps the best defense against them.

Bibliography

[1] The Tor Project, Inc. Tor Project: Overview. [online], 2018, (Accessed on 04/14/2018). Available from: https://www.torproject.org/about/overview.html.en

[2] The Tor Project, Inc. Tor: Onion Service Protocol. [online], (Accessed on 04/14/2018). Available from: https://www.torproject.org/docs/onion-services.html.en

[3] The Tor Project, Inc. Onion Routing: History. [online], 2005, (Accessed on 04/14/2018). Available from: https://www.onion-router.net/History.html

[4] Levine, Y. Pando: Almost Everyone Involved in Developing Tor was (or is) Funded by the US Government. [online], jul 2014, (Accessed on 04/14/2018). Available from: https://pando.com/2014/07/16/tor-spooks/

[5] The Tor Project, Inc. Servers – Tor Metrics – Servers. [online], (Accessed on 04/14/2018). Available from: https://metrics.torproject.org/networksize.html?start=2018-03-21&end=2018-03-23

[6] The Tor Project, Inc. Traffic – Tor Metrics – Traffic (2017-03-01 - 2018-03-23). [online], (Accessed on 04/14/2018). Available from: https://metrics.torproject.org/bandwidth.html?start=2018-03-01&end=2018-03-23

[7] The Tor Project, Inc. Traffic – Tor Metrics – Traffic (2018-03-01 - 2018-03-23). [online], (Accessed on 04/14/2018). Available from: https://metrics.torproject.org/bandwidth.html?start=2017-03-01&end=2018-03-23

[8] The Tor Project, Inc. Tor: Sponsors. [online]. Available from: https://www.torproject.org/about/sponsors.html.en

[9] Dingledine, R.; Mathewson, N.; et al. Tor: The Second-Generation Onion Router. [online], 2004. Available from: https://svn.torproject.org/svn/projects/design-paper/tor-design.pdf

[10] The Tor Project, Inc. Who uses Tor? [online], (Accessed on 04/14/2018). Available from: https://www.torproject.org/about/torusers.html.en

[11] The Tor Project, Inc. Tor Browser. [online], (Accessed on 04/14/2018). Available from: https://www.torproject.org/projects/torbrowser.html

[12] The Tor Project, Inc. Tor Project: Projects Overview. [online], (Accessed on 04/14/2018). Available from: https://www.torproject.org/projects/projects.html.en

[13] Stoica, I. Overlay networks. [online], (Accessed on 04/15/2018). Available from: http://www.cs.virginia.edu/~cs757/slidespdf/757-09-overlay.pdf

[14] Dingledine, R. and N. Mathewson. Tor's protocol specifications. [online], (Accessed on 04/15/2018). Available from: https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt

[15] The Tor Project, Inc. Relay Search. [online], apr 2014, (Accessed on 04/15/2018). Available from: https://metrics.torproject.org/rs.html#search/flag:authority

[16] The Tor Project, Inc. Tor directory protocol, version 3. [online], (Accessed on 04/15/2018). Available from: https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt

[17] The Tor Project, Inc. Relay Search: dizum. [online], (Accessed on 04/15/2018). Available from: https://metrics.torproject.org/rs.html#details/7EA6EAD6FD83083C538F44038BBFA077587DD755

[18] The Tor Project, Inc. Tor Rendezvous Specification - Version 3. [online], (Accessed on 04/15/2018). Available from: https://gitweb.torproject.org/torspec.git/tree/rend-spec-v3.txt

[19] Elaine, B.; Roginsky, A. Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths. [online], nov 2015, (Accessed on 04/17/2018). Available from: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-131ar1.pdf

[20] The Tor Project, Inc. Tor Project: FAQ. [online], (Accessed on 04/16/2018). Available from: https://www.torproject.org/docs/faq.html.en#AttacksOnOnionRouting

[21] Le Blond, S. et al. One Bad Apple Spoils the Bunch: Exploiting P2P Applications to Trace and Profile Tor Users. [online], (Accessed on 04/16/2018). Available from: https://www.usenix.org/legacy/events/leet11/tech/full_papers/LeBlond.pdf

[22] Murdoch, S. J. and G. Danezis. Low-Cost Traffic Analysis of Tor. [online], (Accessed on 04/16/2018). Available from: http://sec.cs.ucl.ac.uk/users/smurdoch/papers/oakland05torta.pdf

[23] Kwon, A. et al. Circuit Fingerprinting Attacks: Passive Deanonymization of Tor Hidden Services. [online], (Accessed on 04/16/2018). Available from: https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-kwon.pdf

[24] Schneier, B. How the NSA Attacks Tor/Firefox Users With QUANTUM and FOXACID - Schneier on Security. [online], oct 2013, (Accessed on 04/16/2018). Available from: https://www.schneier.com/blog/archives/2013/10/how_the_nsa_att.html

[25] Paganini, P. The impact of the HeartBleed Bug on Tor Anonymity. [online], apr 2014, (Accessed on 04/16/2018). Available from: https://securityaffairs.co/wordpress/24110/hacking/heartbleed-bug-tor.html

[26] Johnson, A. et al. Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries. [online], nov 2013, (Accessed on 05/03/2018). Available from: https://www.ohmygodel.com/publications/usersrouted-ccs13.pdf

[27] Violeta, I. Introduction to MATLAB: Data Analysis and Statistics. [online], 2007, (Accessed on 05/03/2018). Available from: http://web.mit.edu/acmath/matlab/IAP2007/IntroMatlabStatistics.pdf

[28] Lund Research Ltd. Pearson Product-Moment Correlation. [online], (Accessed on 05/09/2018). Available from: https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php

[29] SPSS Tutorials. Pearson Correlations – Quick Introduction. [online], (Accessed on 05/03/2018). Available from: https://www.spss-tutorials.com/pearson-correlation-coefficient/

[30] Press, H. W. et al. *Numerical Recipes in Fortran 77: The Art of Scientific Computing*. Numerical Recipes Software, 1986-1992, ISBN 052143064X, 14.5 Linear Correlation.

[31] Murdoch, J. S. and P. Zieliński. Sampled Traffic Analysis by Internet-Exchange-Level Adversaries. [online], (Accessed on 05/03/2018). Available from: http://sec.cs.ucl.ac.uk/users/smurdoch/papers/pet07ixanalysis.pdf

[32] Back, A. et al. Traffic Analysis Attacks and Trade-Offs in Anonymity Providing Systems. [online], (Accessed on 04/17/2018). Available from: http://www.cypherspace.org/adam/pubs/traffic.pdf

[33] Sun, Y. et al. RAPTOR: Routing Attacks on Privacy in Tor. [online], 2015, (Accessed on 04/17/2018). Available from: https://www.freehaven.net/anonbib/cache/raptor-sec2015.pdf

[34] Savvius, Inc. Glossary of Network Terms. [online], (Accessed on 05/09/2018). Available from: https://www.savvius.com/networking-glossary/glossary-network-terms/

[35] Levine, N. B. et al. Timing Attacks in Low-Latency Mix Systems. [online], (Accessed on 04/17/2018). Available from: https://www.freehaven.net/anonbib/cache/timing-fc2004.pdf

[36] Billman, A. Use Traffic Analysis to Defeat TOR. [online], 2013, (Accessed on 05/03/2018). Available from: https://null-byte.wonderhowto.com/how-to/use-traffic-analysis-defeat-tor-0149100/

[37] Müller, K. Defending End-to-End Confirmation Attacks against the Tor Network. [online], 2015, (Accessed on 05/03/2018). Available from: https://brage.bibsys.no/xmlui/bitstream/id/355191/KMuller_2015.pdf

[38] Dingledine, R. et al. One Fast Guard for Life (or 9 months). [online], (Accessed on 05/03/2018). Available from: https://petsymposium.org/2014/papers/Dingledine.pdf

[39] The Tor Project, Inc. Tor Project: FAQ. [online], (Accessed on 04/16/2018). Available from: https://www.torproject.org/docs/faq.html.en#ChangePaths

[40] The Tor Project, Inc. Users – Tor Metrics – Users. [online], (Accessed on 05/03/2018). Available from: https://metrics.torproject.org/userstats-relay-country.html

Appendix A

Acronyms

AS Autonomous System

BGP Border Gateway Protocol

DARPA The Defence Advanced Research Projects Agency

DDoS Distributed Denial of Service

DNS Domain Name System

DoS Denial of Service

HTTP Hypertext Transfer Protocol

IP Internet Protocol

ISP Internet service provider

NIST National Institute of Standards and Technology

OP Onion Proxy

OR Onion Router

TCP Transmission Control Protocol

Appendix B

Contents of the enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `analyze.py` ..... the script used for testing in chapters 3 and 4
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
- `test_results` ..... the directory with the results of all the tests