


Technische Universität München
Department of Informatics

Interdisciplinary Project in Electrical Engineering

Detecting TLS interception in ISP networks

Johannes Schleger

Technische Universität München

Department of Informatics

Interdisciplinary Project in Electrical Engineering

Detecting TLS interception in ISP networks

Erkennung von aufgebrochenen TLS-Verbindungen in ISP-Netzwerken

Author	Johannes Schleger
Supervisor	Prof. Dr.-Ing. Georg Carle
Advisor	Florian Wohlfart, M. Sc.
Date	July 26, 2017

Informatik VIII
Chair for Network Architectures and Services

Abstract

The goal of this work is to not only find TLS interception in ISP networks, but also to provide a detailed characterization including the impact on security and triggers of the proxy responsible for the interception. To obtain insight in a variety of networks and different deployments of proxies a crowd-based measurement approach is chosen. This approach requires three components, namely TLS Client, TLS Capture Server and TLS Analysis Server. This work defines the requirements for those, the interface they are communicating with each other and gives details for the implementation and deployment.

Contents

1 Introduction 1
1.1 Research Question . . . 1
1.2 Goals of the thesis . . . 1
1.3 Outline . . . 2

2 Approach 3
2.1 Test workflow . . . 3
2.2 Detection of TLS interception . . . 4
2.3 Proxy characterization . . . 4
2.4 Design decision . . . 6

3 Requirements 7
3.1 TLS Capture Server . . . 7
3.2 TLS Analysis Server . . . 9
3.3 TLS Client . . . 10
3.3.1 User Stories . . . 10
3.3.2 Client requirements . . . 10

4 System Design and Implementation 12
4.1 Test workflow . . . 12
4.2 TLS Capture . . . 13
4.3 TLS Capture Server . . . 14
4.4 TLS Analysis Server . . . 14
4.4.1 Rest-API . . . 14
4.4.2 TLS JSON Parser . . . 15
4.4.3 Persistence Layer . . . 15
4.5 TLS Android Client . . . 15

5 Deployment 18
5.1 LRZ-Cloud . . . 18
5.2 TLS Analysis Server . . . 18
5.2.1 Requirements . . . 18

5.2.2 Source . . . 18
5.2.3 Build . . . 19
5.2.4 Extract . . . 19
5.2.5 Configuration . . . 19
5.2.6 Start . . . 20
5.2.7 iptables . . . 20
5.3 TLS Capture Server . . . 20
5.3.1 Requirements . . . 20
5.3.2 Source . . . 20
5.3.3 Build . . . 20
5.3.4 Extract . . . 21
5.3.5 Configuration . . . 21
5.3.6 Start . . . 21
5.3.7 iptables . . . 21
5.3.8 Interception . . . 21
5.4 TLS Android Client . . . 22
5.4.1 Requirements . . . 22
5.4.2 Source . . . 22
5.4.3 Build and Deployment . . . 22

6 Conclusion 23

Bibliography 24

List of Figures

2.1 Mobile client connection to own tls server. 4
2.2 Mobile client connection to other tls server. 5
2.3 Tls server acting as tls client. 5

4.1 Overview of the main components 12
4.2 Workflow for the detection of a TLS interception. 13

List of Tables

3.1 REQ-CAP-1 7
3.2 REQ-CAP-2 7
3.3 REQ-CAP-3 8
3.4 REQ-CAP-4 8
3.5 REQ-CAP-5 8
3.6 REQ-CAP-6 8
3.7 REQ-ANALYSIS-1 9
3.8 REQ-ANALYSIS-2 9
3.9 REQ-ANALYSIS-3 9
3.10 REQ-CLIENT-1 10
3.11 REQ-CLIENT-2 10
3.12 REQ-CLIENT-3 11

4.1 API for creating a new sessionID 15
4.2 API for uploading the result of a test 16
4.3 API for analysing a test 17

Chapter 1

Introduction

The goal of Transport Layer Security (TLS) is to ensure end-to-end encryption between two systems. Nevertheless, recent papers have revealed that TLS connections are intercepted [1], [2], [3]. Such interception takes place in proxies or so called middleboxes. In general, research takes either the client- or the server-side view of the connection into consideration to detect TLS interception. Because they look at single connections only, they assume that either all communication from that client is intercepted or none.

1.1 Research Question

We believe, that there are more complex situations. Transparent proxies can be deployed such that not all connections are intercepted, but only some of them. We aim to find not only transparent proxies, but also selective proxies and characterize them by identifying their triggers in case of selective interception (e.g. domain name, destination IP, client fingerprints). Therefore, we want to deeply inspect the deployment and configuration of the proxies.

1.2 Goals of the thesis

The goal of this thesis is to build a working prototype, which is capable of controlling both the client- and server-side view of the TLS handshake. Like Netalyzr [4], a crowd-based measurement approach will be chosen to obtain insight in a variety of proxies and their deployments.

1.3 Outline

The thesis is structured as follows. In Chapter 2 our approach to the problem is described. The requirements for the independent components are described in Chapter 3. Details of the implementation are given in Chapter 4. The required steps to deploy each component are listed in Chapter 5. Final remarks are given in Chapter 6.

Chapter 2

Approach

We get the full picture of TLS proxies when comparing the client and server-side data-stream. Therefore, our approach requires an TLS Client as well as an TLS Capture Server. In addition, an TLS Analysis Server is used to analyse both views and generate the measurement report.

2.1 Test workflow

The workflow for a new session is as following.

1. TLS Client conducts several tests with varying destination addresses, such that certain white- or blacklist approaches are circumvented. Details are given in Section 2.2.
2. The information gathered during the tests by TLS Client and TLS Capture Server are sent to the TLS Analysis Server.
3. TLS Analysis Server generates a report.
4. TLS Client requests the report which includes the analysis and information whether the connection was intercepted
5. If and only if there was an interception the TLS Client performs a characterization of the proxy. Details are given in Section 2.3.
6. TLS Client sends the proxy characteristics to the TLS Analysis Server.

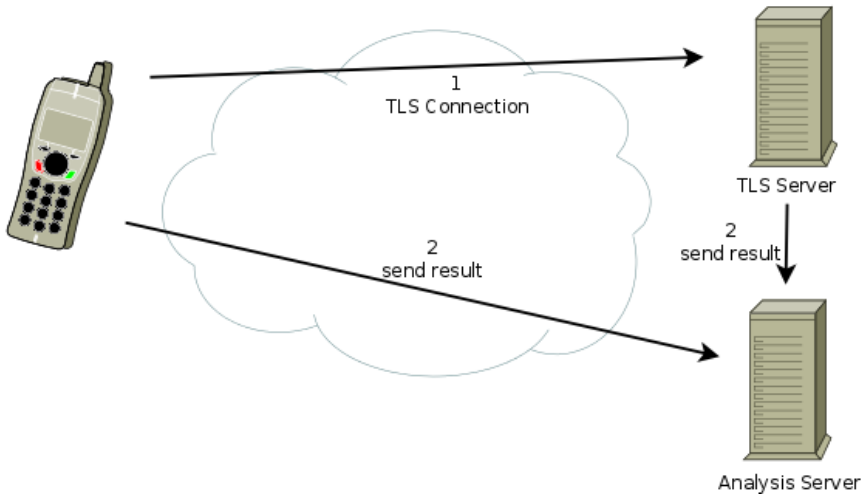

Figure 2.1: Mobile client connection to own tls server.

2.2 Detection of TLS interception

In the first step we perform multiple measurements. These measurements are illustrated in Figure 2.1, 2.2 and 2.3.

In Figure 2.1, the interception is detected by capturing both the client and the server side handshake. Differences in the handshakes are then analysed by the TLS Analysis Server.

Figure 2.2 and 2.3 are required to identify triggers for the interception. A possible trigger might be, that all connections are intercepted, but a whitelist, e.g. banking websites, are not intercepted. Therefore, those tests check whether the destination IP is a possible trigger for interception.

2.3 Proxy characterization

Proxies can be characterized by the following traits.

Localization Find out where the proxy is located.

Application Layer Analyse the effect of the proxy on protocols in the application layer such as HTTP. For example, detect changes in HTTP header fields.

TLS The impact of the proxy on the security of TLS connections. Possible modifications include, but not limited to, the used TLS version, the imitation of other software's fingerprints and using no or invalid Server Name Indications (SNI).

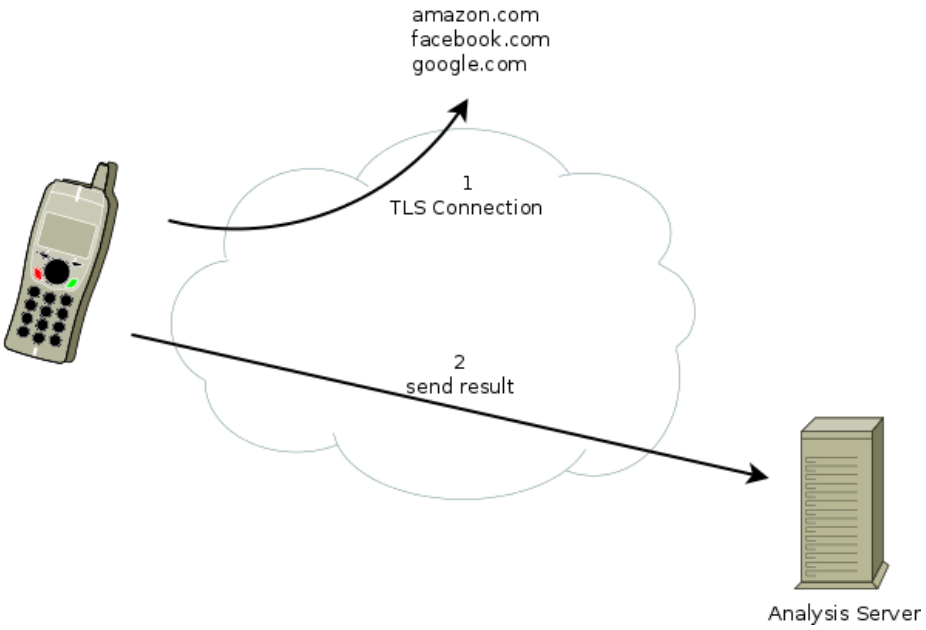

Figure 2.2: Mobile client connection to other tls server.

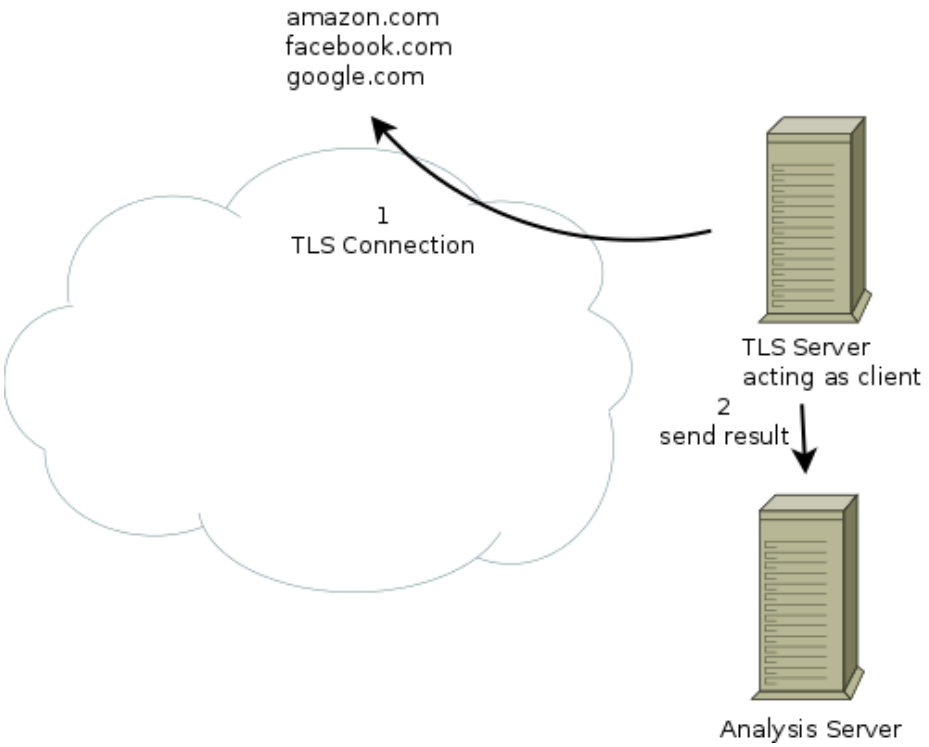

Figure 2.3: Tls server acting as tls client.

2.4 Design decision

The measurement approach requires to be in full control of the TLS layer as well as the appliation layer. Therefore, a decision between a browser-based or mobile-based application is to be made. While a native Android-Client provides full access to raw sockets, web browser deny access to raw sockets. In addition, browser restricts the usage of plugins such as Flash or Java. Therefore, the decision is made to develop a native Android application, because this enables us to be in full control of the TLS layer and the application layer.

Chapter 3

Requirements

Only features (functional requirements) are listed. Other requirements such as external interface or non-functional requirements are not considered.

3.1 TLS Capture Server

The TLS Capture Server is required for capturing the server-side view on the TLS handshake. In addition, we could detect changes in the application layer. The features for the TLS capture server are listed in table 3.1 to 3.6.

Name	Server accepts new TLS connections
Description	The server accepts new TLS connections from clients.
Priority	high
Preconditions	None
Sequence	1. Client initiates the TLS handshake
Postconditions	Client and server are connected via TLS.

Table 3.1: REQ-CAP-1

Name	Server reads the test ID
Description	The TLS capture server reads the test ID.
Priority	high
Preconditions	An TLS connection is established.
Sequence	1. Client sends the test ID
Postconditions	The server received the test ID.

Table 3.2: REQ-CAP-2

Name	Server closes the TLS connection
Description	An established tls connection is closed.
Priority	high
Preconditions	The server read the test ID successfully.
Sequence	1. Server closes the TLS connection
Postconditions	The TLS connection is closed.

Table 3.3: REQ-CAP-3

Name	Server captures the TLS handshake
Description	The server captures the raw TLS handshake.
Priority	high
Preconditions	None
Sequence	1. A TLS connection is established 2. Data is transferred (optional) 3. TLS connection is closed
Postconditions	The TLS handshake is captured.

Table 3.4: REQ-CAP-4

Name	Server sends the captured TLS handshake
Description	The captured TLS handshake is sent as JSON via HTTP to another server.
Priority	high
Preconditions	An TLS connection was established and the test ID successfully read.
Sequence	1. The Server sends the captured handshake to another server.
Postconditions	The result was sent.

Table 3.5: REQ-CAP-5

Name	Capture
Description	The server acts as client and captures handshakes from its own view.
Priority	low
Preconditions	None
Sequence	1. The server initiates a TLS connection. 2. The server closes the TLS connection. 3. The captured TLS handshake is sent to another server.
Postconditions	The captured handshake was sent.

Table 3.6: REQ-CAP-6

3.2 TLS Analysis Server

The TLS Analysis Server is used to generate a report, which is based on the test results obtained by the TLS Capture Server and the TLS Client. The features for the TLS analysis server are listed in table 3.7 to 3.9.

Name	Receive and store TLS handshakes
Description	The captured TLS handshake is sent as JSON via HTTP to the server.
Priority	high
Preconditions	None
Sequence	1. The server receives the captured handshake. 2. The server stores the handshake in a non-volatile fashion.
Postconditions	The result is stored.

Table 3.7: REQ-ANALYSIS-1

Name	Determine interception
Description	The analysis server correlates the handshakes and determines whether the connection was intercepted.
Priority	high
Preconditions	Client and Server result were received.
Sequence	1. The server receives the request for an analysis for a specific test ID. 2. The server analyses the handshakes. 3. The server returns the analysis result.
Postconditions	The analysis result is returned.

Table 3.8: REQ-ANALYSIS-2

Name	Proxy characterization
Description	The server receives and stores the detected proxies and their characterization
Priority	low
Preconditions	TBD
Sequence	TBD
Postconditions	TBD

Table 3.9: REQ-ANALYSIS-3

3.3 TLS Client

The TLS Client captures the client-side view on both the TLS handshake and application layer. Therefore, the TLS Client must be in full control of those layers.

3.3.1 User Stories

1. I want to customize the list of targets.
2. I want to get instant feedback when an interception was discovered.
3. I want to view test reports.
4. I want to know if the interception was discovered before.

3.3.2 Client requirements

The requirements are listed in Table 3.10 to 3.12.

Name	Execute tests
Description	The client performs several tests.
Priority	high
Preconditions	Internet connection
Sequence	1. Execute tests
Postconditions	The tests have been executed.

Table 3.10: REQ-CLIENT-1

Name	Publish results of tests
Description	The client publishes the test results to the analysis server.
Priority	high
Preconditions	Internet connection, executed tests
Sequence	1. Publish results
Postconditions	The tests have been executed.

Table 3.11: REQ-CLIENT-2

Name	Get and show the test report
Description	The client requests the test report from the analysis server.
Priority	high
Preconditions	Internet connection, executed tests
Sequence	1. Request test report 2. Display test report
Postconditions	The user can see the test report.

Table 3.12: REQ-CLIENT-3

Chapter 4

System Design and Implementation

In this chapter, details of the technical implementation are given. Figure [?] shows an overview of the technology used in each component and the connection between them.

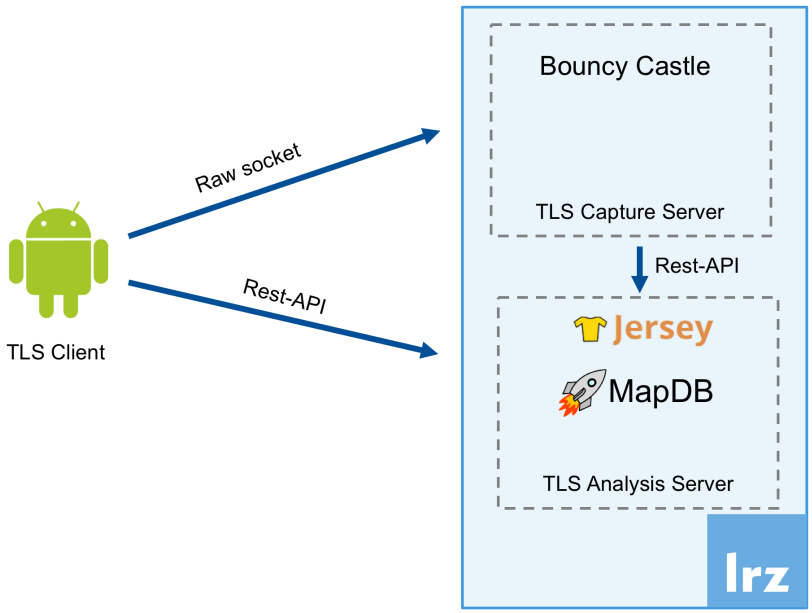

Figure 4.1: Overview of the main components. The TLS Capture Server and TLS Analysis Server are both deployed within the LRZ Cloud.

4.1 Test workflow

An overview of the test workflow is illustrated in Figure 4.2. First, the TLS clients request a new unique sessionID from the TLS Analysis Server. Second, the TLS Client

creates a new TLS connection to the TLS Caption Server, which may be used to transfer additional content in the application layer. Third, the TLS Client and TLS Capture Server transmit their captured results to the TLS Analysis Server. As a last step, the TLS Client request the report from the TLS Analysis Server.

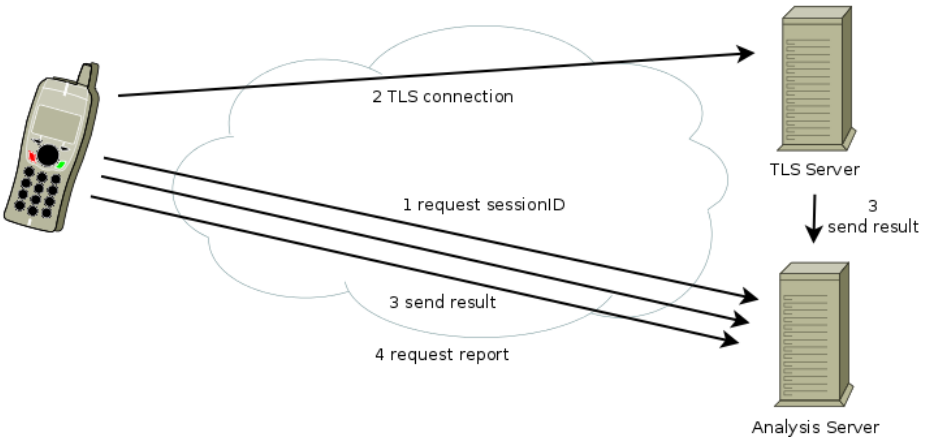

Figure 4.2: Workflow for the detection of a TLS interception.

4.2 TLS Capture

The number of different components, and the availability of a fully customizable crypto library (Bouncy Castle) led the choice to Java as common language for all three components. Therefore, the common functionality is provided by the core project *tls-capture*. It provides the core functionality, which is used by the other components.

TLS client The TLS client contains currently two different client. An default client called *TrimmedTlsClient*, which accepts all server certificates and has a handshake similar to firefox, and a client (SniTlsClient) which is extended with the Server Name Indication (SNI).

TLS server Despite only the TLS Capture Server utilised the functionality, it is within the core package to ease development.

Analysis-API The API for the TLS Analysis Server is utilized by both, the TLS Capture Server and the TLS Client. Therefore, the functionality is within the core package. This includes the creation of new sessions.

Scenario-Framework The goal was to be able to easily create new TLS clients, which in turn execute a specific test case. New clients must implement the *Scenario* interface, which expects to return a *ScenarioResult*. In general it is expected, that it is enough to use the *DefaultClientScenario*, which accepts a *TlsClient* as parameter which is responsible for the TLS handshake.

4.3 TLS Capture Server

The TLS Capture Server is located within the Maven project *tls-capture-server*. It utilises the functionality provided by the core package *tls-capture* and extends it with a configuration and result publisher, which sends the results to the TLS Analysis Server. The procedure of the server is as following.

1. Read configuration
2. Start server
   (a) Accept new connections
   (b) Handle connection
   (c) Close connection
   (d) Publish result of connection

4.4 TLS Analysis Server

The TLS Analysis Server is located within the Maven project *tls-analysis-server*. It utilises the functionality provided by the core package *tls-capture* and extends it with the Rest-API, the TLS JSON Parser, a configuration and the persistence layer.

4.4.1 Rest-API

The Rest-API for the TLS Analysis Server is defined in table 4.1 to 4.3. Because the unique identification of sessions and tests is of utmost importance, the name scheme of those are defined in Definition 4.1 to 4.3. Definition 4.4 to 4.6 define several types used by the Rest-API.

Definition 4.1 *SessionID := [A-Za-z0-9]+*
Definition 4.2 *Counter := [0-9]+*
Definition 4.3 *TestID := SessionID-Counter*
Definition 4.4 *Node := [CLIENT, SERVER]*
Definition 4.5 (State of a test) *State := [CONNECTED, NO_CONNECTION, ERROR]*
Definition 4.6 (Result of the analysis) *Type := [INTERCEPTION, NO_INTERCEPTION, NO_CLIENT_RESULT, NO_SERVER_RESULT, NO_CLIENT_NO_SERVER_RESULT]*

Title	**Create a new unique sessionID**
URL	/session
Method	POST
URL Params	-
Data Params	-
Success Response	Code: 201 (Created) Content: { sessionID : “jKD273jF“ }
Error Response	-
Note	The parameter *sessionID* is defined in Definition 4.1.

Table 4.1: API for creating a new sessionID. A new sessionID is to requested by the TLS Client before the start of test execution.

4.4.2 TLS JSON Parser

The raw captured handshake is parsed by the external library *tls-parser* [5]. The project *tls-json-parser* extends the capability, to convert the internal data structure from the *tls-parser* to JSON and write it to STDOUT. The *tls-json-parser* converts not all messages, but only the for the analysis required messages such as *ClientHello*, *ServerHello* and *Certificate*. The procedure of the *tls-json-parser* is as following.

1. Read captured TLS data encoded as Base64
2. Parse TLS messages
3. Convert internal data structure to JSON
4. Write JSON to STDOUT

4.4.3 Persistence Layer

The data received via the Rest-API is stored by the in-memory database MapDB [6], which is backed up by a file named *file.db*.

4.5 TLS Android Client

The chosen compatibility level of Android is version 4.4 (API Level 19). One reason for that is that this level is supported by more than 87% of the devices [7]. Another reason is, that Spongy Castle is obsolete and Bouncy Castle itself can be used [8].

In the following the most important aspects of the implementation are briefly described.

- *MainActivity* provides an overview of the performed sessions.
- *SessionViewActivity* provides the overview of the tests performed within a session.

Title	**Upload a captured handshake**
URL	/result/:testID
Method	PUT
URL Params	testID (see Definition 4.3)
Data Params	Content: { node: "CLIENT", source: "10.83.81.4:32788", destination: "10.83.81.4:32789", state: "CONNECTED", receivedBytes: "YWJjCg==", sentBytes: "ZGVmCg==", error: "", msg: "" }
Success Response	Code: 204 (No Content)
Error Response	Code: 400 (Bad Request) Content: TBD
Note	The parameters *receivedBytes*, *sentBytes*, *error* and *msg* are optional. The captured handshake is Base64 encoded. For accepted values for the parameters *node* and *state* see Definition 4.4 and 4.5.

Table 4.2: API for uploading the result of a test. Each test result shall be uploaded by the TLS Client to the TLS Analysis Server. The test result contains metadata such as source and destination ip-address and port, as well as the captured TLS handshake.

- *TestViewActivity* displays the result of a test.
- *ProgressActivity* shows the progress while the tests of a session are executed.
- *ConfigurationReader* provides methods for loading and storing the configuration, as well as executed sessions and tests.

Title	**Retrieve the analysis for a specific test session**
URL	/analysis/:testID
Method	GET
URL Params	testID (see Definition 4.3)
Data Params	-

Success Response	Code: 200 (OK)
	Content: { type: "INTERCEPTION",
	clientHello: {
	version: { expected: "771", actual: "771" },
	ciphers: { expected: "[1, 2, 3]", actual: "[2, 3, 4]" }
	},
	serverHello: {
	version: { expected: "771", actual: "771" },
	ciphers: { expected: "2", actual: "4" }
	},
	certificate: {
	certChain: { expected: "c1", actual: "c2" }
	},
	}

Error Response	Code: 400 (Bad Request) Content: TBD
Note	-

Table 4.3: API for analysing a test. The result contains the diff between the expected (what the client sent) and the actual (what the server received) message. The status indicates, whether an interception was detected. Analysed messages are *ClientHello*, *ServerHello* and *Certificate*.

Chapter 5

Deployment

In this chapter, the steps required to deploy each of the three components are described.

5.1 LRZ-Cloud

Both, the TLS Analysis Server and the TLS Capture Server are deployed on a single virtual machine within the LRZ-Cloud, which has the IP-Address 141.40.254.119. The operating system is Debian in version 7 (wheezy). For the login, the public key must be stored in */root/.ssh/authorized_keys*. Then, it is possible to login as shown in Listing 5.1.

```
ssh root@141.40.254.119
```

Listing 5.1: Login to the virtual machine

5.2 TLS Analysis Server

5.2.1 Requirements

- Maven, version 3 or later
- Java, version 8 or later
- Rust, tested with version 1.18.0

5.2.2 Source

The source can be obtained via git (see listing 5.2). Once the repository is cloned, it is enough to pull the latest changes (see listing 5.3). After the successful download of the sources the server can be built.

```
git clone https://gitlab.lrz.de/wohlfart/tls-interception.git
cd tls-interception
```

Listing 5.2: Clone the git repository

```
git pull
```

Listing 5.3: Pull the latest changes from the repository

5.2.3 Build

```
cd tls-capture
mvn clean install
cd ../tls-analysis-server
mvn clean install
```

Listing 5.4: Build the TLS Analysis Server

The successful build generates archives as artefacts. Those two are *tls-analysis-server-<version>-bin.tar.gz* and *tls-analysis-server-<version>-bin.zip*, where *<version>* is replaced with the version number of the build (e. g. 1.0-SNAPSHOT).

5.2.4 Extract

```
unzip tls-analysis-server-<version>-bin.zip
cd tls-analysis-server-<version>
```

Listing 5.5: Build the TLS Analysis Server

5.2.5 Configuration

The configuration resides within the *conf* directory.

- **tls.p12** is the PKCS12 keystore containing the certificate (chain) and the corresponding private key.
- **server.properties**
  - **tls.keystore.password=password** password for the keystore tls.p12
  - **port=443** The port on which the server is listening

5.2.6 Start

After the configuration the TLS Analysis Server can be started. Because no service is implemented yet, *nohup* is used to avoid stopping the process when logging out.

```
nohup ./start.sh &
```

Listing 5.6: Start the TLS Analysis Server

5.2.7 iptables

At the moment the TLS Analysis server is running as root on port 3000. To allow incoming traffic on the port, an iptables rule must be created and saved (see listing 5.7).

```
iptables -A INPUT -p tcp --dport 3000 -j ACCEPT
iptables-save > /etc/iptables/rules.v4
```

Listing 5.7: Allow incoming traffic on port 3000

5.3 TLS Capture Server

5.3.1 Requirements

- Maven, version 3 or later
- Java, version 8 or later

5.3.2 Source

See section 5.2.2.

5.3.3 Build

```
cd tls-capture
mvn clean install
cd ../tls-capture-server
mvn clean install
```

Listing 5.8: Build the TLS Capture Server

The successful build generates archives as artefacts. Those two are *tls-capture-server-<version>-bin.tar.gz* and *tls-capture-server-<version>-bin.zip*, where *<version>* is replaced with the version number of the build (e. g. 1.0-SNAPSHOT).

5.3.4 Extract

```
unzip tls-capture-server-<version>-bin.zip
cd tls-capture-server-<version>
```

Listing 5.9: Extract the TLS Capture Server artefact

5.3.5 Configuration

The configuration resides within the *conf* directory.

- **cert.pem** is the certificate used for TLS connections. It can be a certificate chain.
- **key.pem** is the corresponding private key. At the moment RSA and ECDSA are supported only.
- **server.properties**
  - **test.session=ONLINE/LOCAL** ONLINE: the results are sent to target.url, LOCAL: the results are logged only
  - **target.url=http://127.0.0.1:3000** where to send the captured handshake. Only applies for test.session=ONLINE
  - **port=443** The port on which the server is listening

5.3.6 Start

After the configuration the TLS Capture Server can be started. Because no service is implemented yet, *nohup* is used to avoid stopping the process when logging out.

```
nohup ./start.sh &
```

Listing 5.10: Start the TLS Capture Server

5.3.7 iptables

The TLS Capture server is running on port 443. To allow incoming traffic on that port see section 5.2.7.

5.3.8 Interception

For test purposes we cannot rely on interceptions in ISP networks. Therefore, we have the ability to intercept the traffic on the server itself. A software offering that

functionality is *SSLsplit* [9], which was installed from source. After the successful installation, the necessary steps to start *SSLsplit* are shown in Listing 5.11. The iptables rules for switching the interception on and off are provided in Listing 5.12 and 5.13. At the moment, *SSLsplit* is listening on port 8080.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -A INPUT -p tcp --dport 8080 -j ACCEPT
cd sslsplit
nohup ./start.sh &
```

Listing 5.11: Start SSLsplit

```
iptables -t nat -A PREROUTING -p tcp --destination-port 443 -j REDIRECT --to-ports 8080 ! -s 127.0.0.1/24
```

Listing 5.12: Add iptables rule for TLS interception

```
iptables -t nat -D PREROUTING -p tcp --destination-port 443 -j REDIRECT --to-ports 8080 -s 127.0.0.1/24
```

Listing 5.13: Remove iptables rule for TLS interception

5.4 TLS Android Client

5.4.1 Requirements

- Gradle, version 3.2
- Java, version 7 or later
- Android Studio, version 2.3.3

5.4.2 Source

See section 5.2.2.

5.4.3 Build and Deployment

At the moment, the build and deployment takes place within Android Studio.

Chapter 6

Conclusion

Research in this area focused on the presence of TLS interceptions, but not on the characterization of intercepting devices. Therefore, this work focused on the characterization of proxies. The chosen approach, which relies on crowd-based measurements, involves three components, namely TLS Client, TLS Capture Server and TLS Analysis Server. For those the requirements were defined, and then implemented and deployed.

The necessary components and the corresponding effort to built those had the impact, that at the moment proxies can be characterized by their impact on the TLS connection only. Therefore, this work built the foundation for a thorough detection of TLS interception and must be extended by future work, which should focus on following topics.

Cross-Analysis The detection for interceptions is currently based on single connections only. The analysis of all tests of a session needs to be implemented to detect a possible trigger for selective interception.

Fingerprints The fingerprint of the TLS Client can be modified, that it is recognised as another software such as Firefox. Fingerprints can be found in the repository referenced in [1].

Application Layer At the moment, the proxy can only be characterized by its impact on the TLS connection. Further work needs to be done, to also analyse the impact on the application layer such as HTTP. Therefore, the work should be extended by placing several application layer protocols on top of the TLS connection, such that proxy functionality can be analysed and thus providing a more detailed proxy characterization.

TLS Client The user interface of the TLS Client offers elementary functionality only. It should be extended such that the results and information are presented in a more comprehensible way.

Bibliography

[1] Z. Durumeric *et al.*, "The security impact of https interception," *NDSS*, 2017. [Online]. Available: https://jhalderm.com/pub/papers/interception-ndss17.pdf

[2] X. De Carnavalet *et al.*, "Killed by proxy: Analyzing client-end tls interception software," *NDSS*, 2016. [Online]. Available: https://users.encs.concordia.ca/~mmannan/publications/ssl-interception-ndss2016.pdf

[3] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson, "Analyzing forged ssl certificates in the wild," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, ser. SP '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 83–97. [Online]. Available: http://dx.doi.org/10.1109/SP.2014.13

[4] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson, "Netalyzr: illuminating the edge network," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 246–259.

[5] P. Chifflier, "tls-parser," last visited on 16.07.2017. [Online]. Available: https://github.com/rusticata/tls-parser

[6] MapDB, "Mapdb," last visited on 24.07.2017. [Online]. Available: http://www.mapdb.org/

[7] Google Inc., "Platform versions," last visited on 16.07.2017. [Online]. Available: https://developer.android.com/about/dashboards/index.html

[8] Roberto Tyley, "Spongy castle: is it obsolete?" last visited on 16.07.2017. [Online]. Available: https://github.com/rtyley/spongycastle/issues/34

[9] D. Roethlisberger and contributors, "Sslsplit - transparent ssl/tls interception," last visited on 16.07.2017. [Online]. Available: https://www.roe.ch/SSLsplit