Laboratory Assignment 3:
Certificates, PKI, and Secure Communication

1 Usage of the Virtual Machines

Students are recommended to use their own equipment to do the lab assignments as that will make it easier for them to prepare for the lab. It is also possible to run the virtual machines in the computers at the lab. A more detailed description about the necessary tools, instructions and links to download the virtual machines is available on Canvas.

Follow the steps below before starting with the lab assignment:

1. Download the lab machines and extract them using 7zip
2. Start *Oracle VM VirtualBox Manager*
3. Click on the menu Machine → Add
4. Go to the location where you have downloaded and unzipped the virtual machines, and select the *.vbox* file.
5. Repeat for all VMs needed in the lab.
6. If a snapshot called "lab-start" or similar does not exist yet, create a Snapshot by following the instructions below:
   (a) Select the VM in *Oracle VM VirtualBox Manager*
   (b) Click on *Machine Tools* → Snapshots
   (c) Right-click on *Current State* → Take
   (d) Name the Snapshot *START_EDA491* and click OK.
7. Follow the lab instructions.

If using the lab machines, remember to delete, when you are done, your local copies of the virtual machines (from virtual box first and then from the extracted folder) to ensure the system has free space for the next student.

2 How to ask questions and demonstrate the lab

Labs will be performed on room ED4225. Attending the room physically to do the demonstration is required to pass the labs. All students in the group need to: be physically present; and have

understood, and be able to answer all the questions. Demos can take from 15 minutes to half an hour. Since there may be other groups before you, you should never consider coming to do your demonstration less than one hour before the lab pass ends.

You are also welcome to come to the labs to do the assignment, ask us questions, and even perform your demo once you are done. That said, other than for the demo, it is up to you to decide where and how to perform the assignment. During the lab, TAs will prioritize attention to students doing their demonstration during the time they booked.

If you need to do demos or plan on joining the lab session after 18:00 or before 10:00, make sure to perform your booking at least 24 hours in advance to ensure that a TA will wait for you. Please remove your bookings (or ask us to do it for you if it is already locked) if you cannot attend your booking. We have to plan our workload based on the number of students signing up and we would rather leave early than wait an hour for an student which will not come.

3 Introduction

A major problem with the Internet is that it was not designed with security in mind. We rely heavily on cryptography to enable authentication when connecting to unknown entities, and to provide confidentiality of our communication.

This lab focuses on digital certificates (X.509) and the *Transport Layer Security (TLS)* protocol which is supported by all modern web browsers. It is an IETF standard (RFC 5280 and RFC 8446) which builds on the *Secure Sockets Layer (SSL)*, and the terms are often used interchangeably, although technically that is incorrect. SSLv3 preceded TLSv1, and all versions of SSL are considered insecure[1], so use later versions of TLS whenever possible.

4 Purpose and Scope

This lab consists of four parts:

Part 1: Authentication, digital certificates and chains of trust. In Part 1 you will investigate digital certificates and how they can be applied to provide trust when connecting to known or unknown systems. This will be done in a web browser such as Mozilla Firefox or Google Chrome. You should gain a better understanding of the uses and limitations of certificates in the Internet.

Part 2: Confidentiality and Secure Sockets Layer operation. Part 2 focuses on confidentiality and integrity and the operation of Transport Layer Security (TLS). You will use the OpenSSL package to investigate the TLS handshake, and you will look into the ciphers supported by OpenSSL. In Part 2 you will also use Wireshark to capture and investigate TLS packets.

Part 3: Creating and signing certificates. Part 3 focuses on *creating* certificates. You will learn how to use OpenSSL to create certificate requests and public/private key pairs. You will use a local Certificate Authority (CA) to sign the certificates. The aim of Part 3 is to exemplify how server certificates can be created and managed, and to clarify the notion of trusting a CA.

Part 4: Comparing TLS1.2 and TLS1.3 Handshakes. TLS1.3 was released in 2018 and has gained approval from researchers and industry for being more secure as well as more efficient. In this part you will compare the TLS1.2 handshake with TLS1.3 and learn how they differ.

[1] `https://www.openssl.org/~bodo/ssl-poodle.pdf`

5 Preparations prior to the lab

You are expected to read the following chapters **before** doing to the lab.

In the course book, "William Stallings: Cryptography and Network Security":

- Chap 14.4 (X.509 Certificates)
- Chap 17.2 and 17.3 (Secure Socket Layer and Transport Layer Security)

Try to be well-prepared before coming to the lab, you will save much time!

See also the "Additional reading resources" sub-heading for this assignment in Canvas. Those may be helpful, but are not strictly necessary for the lab.

6 Reporting

At the end of the lab session, you must discuss the answers to the questions stated throughout this document with a teaching assistant. Remember to make notes for your final discussion.

7 Lab setup

Use the VM `lab-kali-linux` – **don't forget to restore to the lab-start snapshot**. The name, ip address, username and password for the VM is shown in Table 1 together with a short description.

VM name	IP-Address	Description	username/password
kali-linux	10.0.0.1/24	Host to run website and openssl	eda491/EDA491!

Table 1: Name of the needed VM, including its ip address and username/password combination.

The files required for part 2 and 3 are located in `/home/eda491/netsec-lab3/`. The folder includes the setup for a local OpenSSL CA, and an old certificate for theoden, a web server you are going to access in this assignment, and its private key.

The passwords are:

- private key of root certificate (CA): **eda491-2015**.
- private key of theoden's certificate: **eda491-2015**.

Starting the webserver. For part 1 and 4 you need to start the locally installed webserver *apache2*. Start the webserver by entering `systemctl start apache2` in the terminal.

8 Lab assignments

This lab consists of four parts: certificates, openssl, creating a certificate request to a certificate authority and comparing the handshake of TLS1.2 and TLS1.3. They will be covered in sequence in the following subsections.

8.1 Part 1: Authentication using certificates

In electronic commerce it is important for clients to be confident that transactions are secure. One way of providing authentication is to use a *Public Key Infrastructure* (PKI) and to let the servers authenticate themselves by providing trusted certificates to the clients. In this assignment, we will use a web browser to investigate certificates and their implications for trust. The instructions are written for Mozilla's Firefox, but any browser of your choice will do (as long as it supports certificates).

NOTE: *In the following, take extra care whether you are supposed to use* **http** *or* **https**.

Direct your browser (use Firefox) to `https://theoden.ce.chalmers.se/`[2]. Theoden provides security for the connecting client, but when you enter the page, you are directed to a security warning page. Inspect the security warning, but **do not import the certificate!**

Q1: What are the possible reasons for this security warning?

The security warning holds information about the web server's certificate. Answer the following questions by viewing the certificate (`"Advanced"` → `"Add Exception..."` → `"View..."`).

Q2: Who is the issuer of the certificate? (answer with the Common Name)

Q3: To whom is the certificate issued? (answer with the Common Name)

When you have inspected the certificate, close the window and press `"Cancel"` to get back to the security warning page. *If you added the certificate by mistake, remove it again before continuing. Ask for help if you do not know how.*

Mozilla keeps a list of publicly trusted certificates[3]. These certificates are issued by trusted issuers, e.g., DigiCert or Amazon, and are used for authenticating websites. If a user decides to trust a certain issuer, the user can import the trusted issuer's certificate.

You will now import the issuer's certificate into the browser and add it to the list of trusted certificates: Navigate your web browser to `http://theoden.ce.chalmers.se/` (*Note the protocol!)*. Download and save the root certificate in your home directory (right-click → save as), and import the certificate manually: `Preferences` → `Advanced` → `Certificates` → `View Certificates`. (What tab should you use? Think about the purpose of the certificate to use the correct tab).

Direct the browser to `https://theoden.ce.chalmers.se/` again. The security warning should have disappeared.

Q4: Why is there no security warning any more? Explain using the following terms: web browser, root certificate, web server certificate, signature, verification.

[2] Make sure that you have started the local webserver as described in Section 7.
[3] `https://wiki.mozilla.org/CA/Included_Certificates`

8.2 Part 2: Confidentiality using TLS

In the previous part we illustrated one part of a secure connection, *authentication*. In this step, you will have a closer look at another property of a secure connection, *confidentiality*. This is achieved with encryption.

You will now investigate how TLS ensures confidentiality by studying the *handshake* procedure. There is a tool for debugging SSL/TLS connections which has many useful options: `s_client`, a sub-command of `openssl`. By using the appropriate options and flags, you will be able to answer the questions. Look at the manual page for `s_client`[4] to find the correct options.

Use `s_client` to connect to theoden on the https port (443). Make sure you use the parameter `-tls1_2` After you connected, at the end of the output, you should see that `s_client` failed to verify the server's identity.

Q5: What is the reason the verification failed?

Perform the necessary steps to correctly verify the server's identity. (*Hint:* recap from Part 1 how the browser verified the identity of theoden.)

Q6: Once theoden's identity is correctly verified, the printout should differ slightly: What did you do, and why did this solve the problem?

During the TLS handshake, several messages are exchanged in order to decide upon the parameters to use. Some messages are always present while some are optional and depend on the authentication procedure. Use `s_client` to find out what messages are used when connecting to theoden. (*Hint:* Using `openssl ciphers -v` will provide more information about the algorithms.)

Q7: What algorithms are used for key exchange, authentication of server, application message encryption, MAC calculation?

Q8: RSA with a sufficient keylength is a strong cipher. Why is RSA not used to encrypt application messages?

Open a new terminal window, cd into the `/home/eda491/netsec-lab3` folder (has to be `~/netsec-lab3`), and start the **s_server** application on your local host (port 40123), by issuing the following command (see Section 7 for the password):

```
# openssl s_server -accept 40123 -cert theoden2015.crt -key theoden2015.key
```

Start Wireshark (`sudo wireshark`) and capture packets on the loopback interface (`lo`). (*Hint:* Use appropriate filters.) Then start the `s_client` application.

```
# openssl s_client -tls1_2 -connect localhost:40123 -CAfile ~/netsec-lab3/cahome/cacert
```

[4] `https://www.openssl.org/docs/man1.1.1/man1/s_client.html`

Q9: At this point you should have an established connection: is there anything strange about the TLS output? (*Hint: Do not waste too much time here, ask a TA if you do not find anything after a minute or two.)*

Use the open connection (with `s_client`) to send the string `TESTMESSAGE` to the server.

Q10: Was the string transmitted successfully? Did you expect this to work? Why or why not?

Now close the connection on the client side with `Ctrl-D` (EOF), stop the server with `Ctrl-C` (SIGINT) and stop the packet capturing in Wireshark.

NOTE: Wireshark decodes (*not decrypts*) well known port numbers to the corresponding protocols by default. For example, traffic to port 80 is decoded to the http protocol. If a service deviates from the default port, Wireshark may not know how to decode (interpret) the traffic. Therefore you should tell Wireshark that you are using port 40123 for TLS. *Hint: check the protocol options for http(s).*

Q11: Identify the different TLS messages in Wireshark: which messages are sent? Illustrate your answer with a diagram.

Q12: Which ciphers are proposed by your client. Where can we find this information?

Q13: Which cipher is selected by the server? Why is it selected? Where can we find this information?

Earlier, you sent the string `"TESTMESSAGE"` from the client to the server.

Q14: Which TLS message do you think contains the string "TESTMESSAGE"?

This concludes the sniffing and the second part of the lab.

8.3 Part 3: Creating and signing certificates

In Part 1 the term *issuer* was used to denote a trusted third party. Such a trusted third party in a network may also be called a *Certificate Authority*, or CA.

In the following you will use OpenSSL to create certificates to authenticate a server. You will also gain some insight into how a Public Key Infrastructure can provide trust by using a trusted CA to sign a server certificate.

NOTE: The following is a warning from the "openssl ca" manpage: *The ca utility was originally meant as an example of how to do things in a CA. It was not supposed to be used as a full blown CA itself: nevertheless some people are using it for this purpose.*

A CA signs certificates and provides certificate revocation lists. Servers and clients in the network rely on the CA for authentication. The provided zip file contains a local OpenSSL Certificate Authority (`~/netsec-lab3/cahome/`). The CA's root certificate is stored in `~/netsec-lab3/cahome/cacert.pem`, and the private key can be found at `~/netsec-lab3/cahome/private/cakey.pem` (password: see Section 7).

NOTE: Public announcement of a private key is *very* dangerous and should *never* be done. We do it only to save you the time to create a new certificate authority yourself, and to easily illustrate how the process works.

Q15: What are the implications of making the CA's private key public?

Q16: Why would you encourage public sharing of the CA's certificate when the private key must be kept secret?

In general, when a certificate is created, a public key is bound to a specific name (the `common name (CN)` of the subject). When creating server certificates, the name to use is the name of the computer on which the server resides. For this lab, it will be the hostname of the computer where you create the certificate.

The client application (e.g., a browser) can match the `CN` in the certificate to match the fully qualified domain name (FQDN) and give a warning if they do not match.

Q17: What is verified when using the openssl library when a TLS connection is being established? (*Hint: It checks five related things, provide at least three.*)

Three different methods exist for creating a certificate. The methods are called *Self-signed certificate, Certificate signed by a trusted CA* and *Certificate signed by a local CA*. In this assignment we are going to use a local CA to sign our server certificate.[5]

When using a local CA, the procedure has three steps:

1. Create a public/private key pair and a *certificate request*.
2. Send the request to the CA, who then signs the request with its private key (from the root certificate), which creates the real certificate.
3. Receive the certificate from the CA.

OpenSSL supports generation of certificate requests by using the command `openssl req`. The request and the private key should be generated according to the following specifications (in nsecXX, replace XX with your group number):

- The request will be a *new* request. i.e. it should not be loaded from any file.
- The request should be signed with a *sha256* message digest.
- The private key should be a *4096 bits rsa* key.
- The private key shall be password protected.
- The certificate fields **subject** should define the following items: Organisation (O), Organisational Unit (OU) and Common Name (CN). Use NETSEC for Organisation, nsecXX for Organisational Unit and the hostname of your computer as Common Name. If you choose to include this information in the command itself, be sure to enclose the entire subject in apostrophes ('), i.e. 'examplestring'.
- Use nsecXX.key as key name and nsecXX.csr as certificate name (csr = certificate signature request).

Hint: Check your hostname with `hostname -A`.

[5] See `https://www.symantec.com/connect/articles/apache-2-ssltls-step-step-part-2` for more info.

Use the following command to print the certificate request in human readable form, and verify that you provided the correct parameters:

```
# openssl req -in nsecXX.csr -text
```

Save the output temporarily, you will need it later. You now have a certificate request and a private key, and you will assume the role of the Certificate Authority to sign the request.

NOTE: Before proceeding, make sure that nsecXX.csr and nsecXX.key are stored in the `netsec-lab3` directory.

When signing a certificate request, the OpenSSL CA uses two files to store internal information about the certificate: `serial` and `index.txt`. `serial` contains the serial number to be used in the *next* certificate that the CA will sign. `index.txt` contains info about the signed certificates.

To sign the certificate, issue the following command:

```
# openssl ca -config  ~/netsec-lab3/cahome/openssl.cnf
   -policy policy_anything -md sha256 -out ~/netsec-lab3/nsecXXsign.pem
   -infiles ~/netsec-lab3/nsecXX.csr
```

NOTE1: `openssl.cnf` assumes the directory structure laid out in this manual. If you chose a different directory structure, adapt the "`dir`" variable in `openssl.cnf` accordingly.

NOTE2: *policy_anything* refers to a field in the CA's config file, `openssl.cnf`. It provides more information about the policy and extensions if you are interested.

NOTE3: If the signing succeeded, and you wish to sign the same certificate again (e.g. because you forgot a parameter), you first need to revoke the previously signed certificate stored in `newcerts` (see `ca -revoke`).

Use the following command to check the signed certificate:

```
# openssl x509 -in ~/netsec-lab3/nsecXXsign.pem -text
```

Q18: Compare the output to the unsigned request you saved earlier. In what ways do the outputs differ?

Q19: Consider the following scenario: You need to buy something and you search the web for more information. After a while you find a small and unknown (at least to you) web-shop which has the lowest price. What a great price, you think! You also see that they have an https connection, and the little lock in your browser is closed (i.e., the server certificate is validated, with a signature by some CA). When you are about to enter your credit card details, you ask yourself...

1. **Does the CA's signature imply that you can trust the person who holds the server certificate (the web shop)?**
2. **Can you trust the server that holds the certificate? For instance, do you believe the information you enter will be kept secure?**

8.4 Part 4: Comparing TLS1.2 and TLS1.3 Handshakes

In this part you are going to compare the handshakes of TLS1.2 and TLS1.3. You will access `https://theoden.ce.chalmers.se` and allow TLS1.3 connections. Please be aware that most popular browsers, such as Mozilla Firefox and Google Chrome, already support TLS1.3 by default, however, Firefox in this virtual machines was manually limited to TLS1.2.

Start Firefox and sniff traffic passing the loopback interface on Wireshark. When you access `https://theoden.ce.chalmers.se` you will be able to see the TLS1.2 handshake. As soon as the handshake is finished, enable TLS1.3 in firefox as follows: Enter `about:config` in the address bar and continue by accepting the risk. Enter `security.tls` in the search bar in order to find the relevant preferences. Change the value of `security.tls.version.max` to 4 instead of 3. Access `https://theoden.ce.chalmers.se` again[6], as you have enabled TLS1.3 now. Compare the traces in Wireshark with each other and find the differences in the handshake.

Q20: Which differences can you observe?

Q21: When does each version start sending encrypted messages?

Q22: Finally, if you remember only one thing from this lab, what should it be? (There is no right answer here.)

When you have answered all the questions, and all students of the group are able to explain the answers, **contact a teaching assistant to discuss your results**.

[6] Restart firefox in case you don't see a difference in Wireshark

9 Optional task(s)

The following task(s) are completely and absolutely optional and are only provided as a reference to students who want to increase their knowledge further than the contents of the course. The TAs will never ask you any questions regarding these tasks. Similarly, the exam will be created assuming these tasks never existed. Consequently, most TAs will not be able to help you with the optional tasks in which case you should ask Francisco about them. Francisco will also be very thankful if you provide him feedback about the tasks and your experience performing them.

9.1 Rationale

TLS is the most common solution used to secure network traffic nowadays. TLS can keep traffic confidential and prevent modification by external attackers. TLS can also authenticate client and server to each other. This authentication, if used, ensures that each system knows who they are speaking to: i.e., the client knowas that the server is legitimate and, when using user certificates, the server knows which client is connecting to it. To be useful, certificates have to be issued by an entity that the other side trusts. Learning how to create certificates trusted by most people and how to secure TLS are both critical tasks when securing systems.

9.2 Task details

In this optional task you will create a certificate issued by Let's Encrypt, a CA trusted by many modern browsers. You will also learn how to check your server's TLS configuration to improve its security (at the cost of reducing the number of systems that can connect with the server).

To perform this task you will need a web server with TLS support, a public IP address, and a domain you control pointing to that web server's public IP address.

To perform this task:

1. Use a tool like certbot or acme.sh to request and issue a Let's Encrypt certificate for your domain.
2. If possible, automate the issuing of new certificates so that you do not need to keep track of when the certificate expires.
3. Configure your web server to use the certificate you created.
4. Use *Qualys SSL server test*[7] and/or *testssl.sh*[8] to check your TLS configuration.
5. Update your configuration to avoid the issues you consider relevant.

[7] `https://www.ssllabs.com/ssltest/`
[8] `https://testssl.sh/`