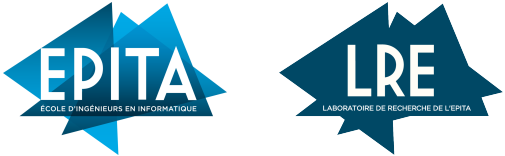
Basics of Networking 2: Security – P2

Daniel STAN

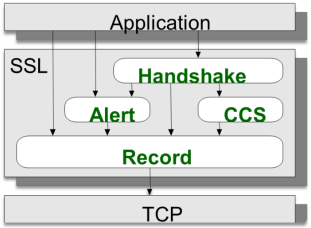


June 10, 2024

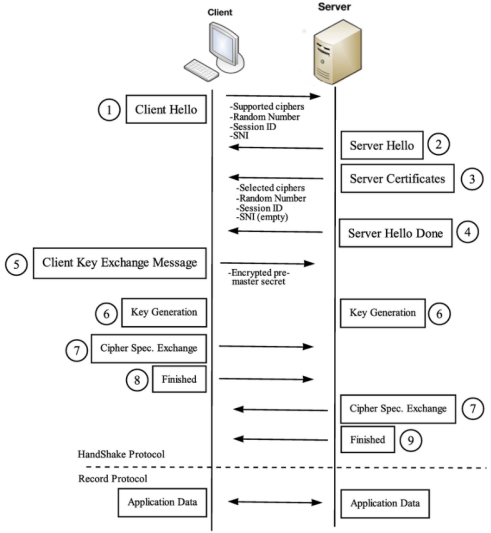
Éléments de réseau 2 Contrib and Speakers: Daniel Stan, Eric Milard, Nidà Meddouri, Elloh Adja, Suzana Dedefa, Christian Diaconu
Version: v1.0, feedback and remarks to: `daniel.stan@epita.fr`

TLS: Transport Security Layer

- The main goal of TLS is to provide a secure channel between two communicating peers
- Provides:
  - Authentication:
    - ★ Through asymmetric encryption
    - ★ The server side is always authenticated
    - ★ Client can also authenticate in some circumstances
  - Confidentiality
  - Integrity
- It is inserted between the **Transport** and the **Application** layer



TLS: Handshake



- Server presents its certificates to authenticate itself.
- Client validates this certificate (PKI).
- Client and servers are also negociating what cryptographic algorithms to use (supported vs selected ciphers).
- Handshake goal: establish a secure bidirectional channel with a **symmetric encryption**

TLS: Versions

Ancestor name: *Secure Socket Layer* (SSL). Many versions, many security flaws[1]
`https://fr.wikipedia.org/wiki/Transport_Layer_Security`

- 1994 : SSL 1.0. (Netscape)
- February 95: SSL 2.0
- March 96: SSL 3.0

[1] these are security flaws in the design of the protocol, even when perfectly implemented by libraries

TLS: Versions

Ancestor name: *Secure Socket Layer* (SSL). Many versions, many security flaws[1]
`https://fr.wikipedia.org/wiki/Transport_Layer_Security`

- 1994 : SSL 1.0. (Netscape)
- February 95: SSL 2.0
- March 96: SSL 3.0

Then SSL got renamed to TLS

- January 99: TLS 1.0
- ...

[1] these are security flaws in the design of the protocol, even when perfectly implemented by libraries

TLS: Versions

Ancestor name: *Secure Socket Layer* (SSL). Many versions, many security flaws[1]
`https://fr.wikipedia.org/wiki/Transport_Layer_Security`

- 1994 : SSL 1.0. (Netscape)
- February 95: SSL 2.0
- March 96: SSL 3.0

Then SSL got renamed to TLS

- January 99: TLS 1.0
- ...
- August 2018: TLS 1.3 ← **current version**

NB: you may still find the name "SSL" everywhere, even though it's TLS now.

[1] these are security flaws in the design of the protocol, even when perfectly implemented by libraries

TLS Implementations

`https://en.wikipedia.org/wiki/Comparison_of_TLS_implementations`

- **OpenSSL**
- LibreSSL
- WolfSSL
- ...

Heartbleed vulnerability in OpenSSL exposed private keys of 12% of the internet (2012).

OpenSSL in CLI: analyze certificate

`openssl x509`

```
openssl x509 -text -in mycert.pem
```

NB: actually the subcommand `x509` has nothing to do with TLS, it's just a PKI management tool.

OpenSSL in CLI: analyze certificate

`openssl x509`

```
openssl x509 -text -in mycert.pem
```

OpenSSL: verify

```
openssl verify -CAfile RootCert.pem -untrusted Intermediate.pem \
    UserCert.pem
```

NB: again, nothing to do with TLS, it's just x509 certificate manipulation.

StartTLS vs New Protocol

TLS is used on top on an existing transport session, typically TCP. There are two ways to implement it:

- Start with a clear text session, then in your current protocol, implement a new command to switch communication to an encrypted session: "`startssl`", "`starttls`".
  - Examples: IMAP, SMTP, POP
  - **Advantages**: backward compatible, not a new protocol port.
  - **Drawbacks**: sensitive to downgrade attacks; a MITM can prevent going to the encrypted session.

StartTLS vs New Protocol

TLS is used on top on an existing transport session, typically TCP. There are two ways to implement it:

- Start with a clear text session, then in your current protocol, implement a new command to switch communication to an encrypted session: "`startssl`", "`starttls`".
  - Examples: IMAP, SMTP, POP
  - **Advantages**: backward compatible, not a new protocol port.
  - **Drawbacks**: sensitive to downgrade attacks; a MITM can prevent going to the encrypted session.
- A completely new application protocol, on top of TLS:
  - HTTP is 80/tcp → HTTP**S** is TLS/443/tcp
  - SMTP is 25/tcp → SMTP**S** is TLS/465/tcp
  - IMAP is 143/tcp → IMAP**S** is TLS/993/tcp
  - FTP is 143/tcp → FTP**S** is TLS/993/tcp

  NB:  FTPS ≠ SFTP (SSH subprotocol) 

A new net cat: openssl connect

Let's get an HTTP**S** webpage! Initiate a TLS session, on port 443, then talk clear HTTP:

```
openssl s_client -connect epita.fr:443
```

A new net cat: openssl connect

Let's get an HTTP**S** webpage! Initiate a TLS session, on port 443, then talk clear HTTP:

```
openssl s_client -connect epita.fr:443


depth=2 C = US, O = Google Trust Services LLC, CN = GTS Root R1
verify return:1
depth=1 C = US, O = Google Trust Services LLC, CN = GTS CA 1P5
verify return:1
depth=0 CN = epita.fr
...
GET / HTTP/1.1
Host: www.epita.fr
```

OpenSSL: it really checks the certificates!

Let's connect to the IP of `epita.fr` instead. Authentication **fails** because server's address doesn't match the presented certificate subject:

```
dstan@flan:~$ openssl s_client -connect 104.26.7.225:443
CONNECTED(00000003)
40C7D94FB4730000:error:0A000410:SSL routines:ssl3_read_bytes:sslv3 alert handshake failure:../ssl/record/rec_layer
---
no peer certificate available
---
```

"104.26.7.225" $\neq$ *"epita.fr"*

Exercise: showcerts

One can list the presented certificate (chain) with `-showcerts` option:

```
$ openssl s_client -connect www.epita.fr:443 -showcerts
[...]
-----BEGIN CERTIFICATE-----
MIIFXzCCBEegAwIBAgIRAKG/1Ew1oUu8E8TPtjDUREowDQYJKoZIhvcNAQELBQAw
...
```

Exercise: why can I connect to *www.epita.fr* while the certificate is for *epita.fr*?

Exercise: showcerts

One can list the presented certificate (chain) with `-showcerts` option:

```
$ openssl s_client -connect www.epita.fr:443 -showcerts
[...]
-----BEGIN CERTIFICATE-----
MIIFXzCCBEegAwIBAgIRAKG/1Ew1oUu8E8TPtjDUREowDQYJKoZIhvcNAQELBQAw
...
```

Exercise: why can I connect to *www.epita.fr* while the certificate is for *epita.fr*?

```
$ openssl x509 -text
...[paste your certificate here]...
X509v3 Subject Alternative Name:
    DNS:epita.fr, DNS:*.epita.fr
```

The HTTPS problem.

- Reminder (**Virtual Host**): in HTTP (cleartext), multiple websites can be hosted on the same IP address/port: the server process picks which site to serve depending on the `Host` field value sent by the client.
- Reminder: HTTPS works with an encrypted session from the beginning, so a single certificate can be presented in the negociation.
  - $\rightarrow$ x509 certificates with **multiple domain names** (alternate DNS) are super useful.
  - **Drawback 1** lots of websites may be hosted on the same server so a large certificate might be there.
  - **Drawback 2**: this discloses the set of websites hosted on a server.

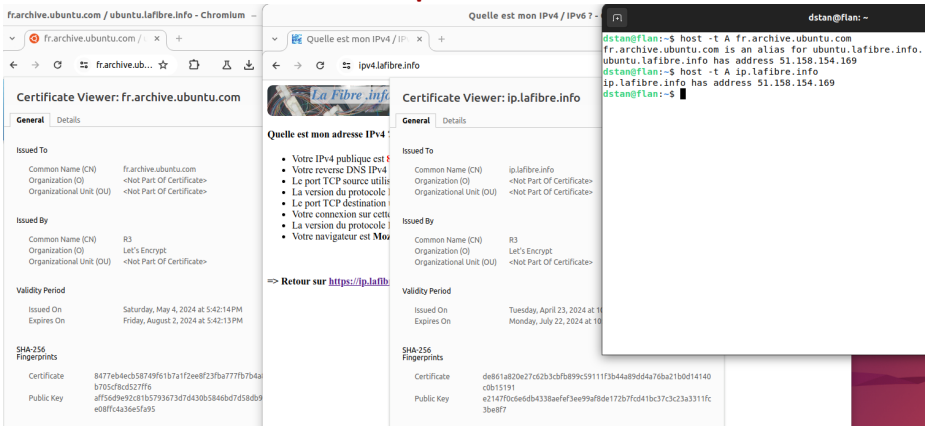
The HTTPS problem.

- Reminder (**Virtual Host**): in HTTP (cleartext), multiple websites can be hosted on the same IP address/port: the server process picks which site to serve depending on the `Host` field value sent by the client.
- Reminder: HTTPS works with an encrypted session from the beginning, so a single certificate can be presented in the negociation.
  - $\rightarrow$ x509 certificates with **multiple domain names** (alternate DNS) are super useful.
  - **Drawback 1** lots of websites may be hosted on the same server so a large certificate might be there.
  - **Drawback 2**: this discloses the set of websites hosted on a server.

  - $\rightarrow$ **Wildcard certificates** (For example: *.epita.fr) :
  - **Drawback**: harder to make, only a solution for FQDNs of a same domain (www., intra.).

HTTPS: Virtual Host vs SNI

Two websites, **same IP** so **same process**, **different domains**, **different certificates**



HTTPS: Virtual Host vs SNI

Two websites, **same IP** so **same process**, **different domains**, **different certificates**
A bit puzzling, isn't it? How can the process server guess what certificate to present, **before** knowing what website will be requested?!!

HTTPS: Virtual Host vs SNI

Two websites, **same IP** so **same process**, **different domains**, **different certificates**
A bit puzzling, isn't it? How can the process server guess what certificate to present, **before** knowing what website will be requested?!!
TLS has a mechanism similar to HTTP's Virtual Host feature: **Server Name Indication** (SNI).

```
openssl s_client -showcerts -servername www.example.com   \
                 -connect www.example.com:443
```

HTTPS: Virtual Host vs SNI

Two websites, **same IP** so **same process**, **different domains**, **different certificates**
A bit puzzling, isn't it? How can the process server guess what certificate to present, **before** knowing what website will be requested?!!
TLS has a mechanism similar to HTTP's Virtual Host feature: **Server Name Indication** (SNI).

```
openssl s_client -showcerts -servername www.example.com   \
                 -connect www.example.com:443
```

Note that this problem would not have happened:

- With a "starttls" style protocol (**less secure**)
- If only a single website was hosted behind an IP address: these "fixes" (Virtual Host and SNI) are needed due to the **IP shortage problem**.

TLS: summary

- Protocol to secure communication: **authentication** and **confidentiality**
- Based on x509 certificates and PKI
- **backward-compatible**: just wrap your clear text application protocol into a TLS session, instead of a TCP session.
- Some "hack" to preserve the "Virtual Host" feature of HTTP: SNI.
- Major (widespread) implementation of the protocol: **OpenSSL**.