Secure Systems and Networks

INEA 00014 W

Tomasz Surmacz, PhD

6. SSL

SSL – Secure Socket Layer / TLS – Transport Layer Security

SSL

- encrypts data
- protects against Man-in-the-middle attacks
- uses certificates to verify the identity of communication peer

Certificates

- peer identity verification
- spoofing protection
- automatically retrieved from peer on connection setup
- certificates are signed – a system of 'Certificate Authorities'

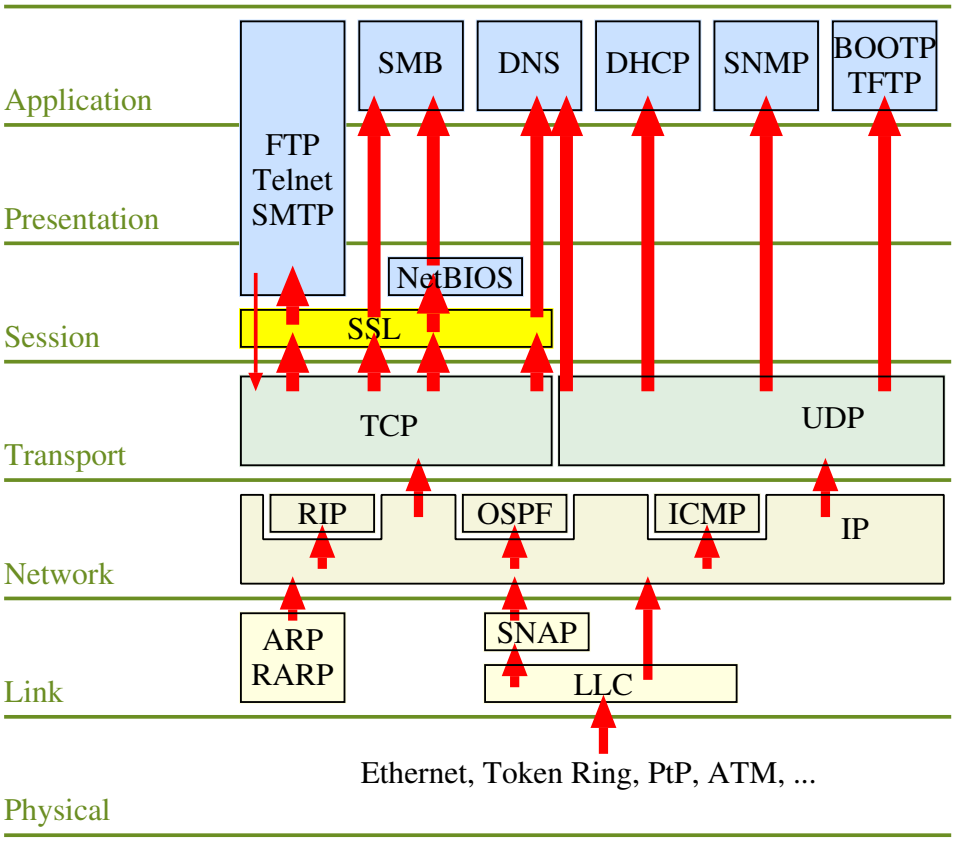
Using SSL

- Explicit SSL
  - by using special protocol, such as `HTTPS`
  - by using appropriate functions from the SSL library
- Implicit SSL
  - by using `tlswrap`

SSL – Secure Socket Layer / TLS – Transport Layer Security

Two basic reasons to use SSL/TLS:

- two-way authentication
- securing the communication channel which can be used by higher layers protocols



SSL – Secure Socket Layer / TLS – Transport Layer Security

SSL/TLS – Development history

- In early 1990s IETF chartered a Web Transaction Security (WTS) WG
- This resulted in an application layer protocol named Secure Hypertext Transfer Protocol (S-HTTP), described in experimental RFCs 2659 and 2660
- Independently, Netscape Communications proposed (and patented) a transport layer security protocol named Secure Sockets Layer (SSL)
- SSL is placed on top of a connection oriented and reliable transport layer (TCP or similar)
- Netscape Communications implemented SSL 2.0 in its Netscape Navigator browser in 1994
- Microsoft proposed a similar concept as Private Communication Technology (PCT) and an enhanced Secure Transport Layer Protocol (STLP) in 1995
- In 1996 SSL 3.0 was specified (currently: RFC 6101)
- Transport Layer Security (TLS) WG has been created by IETF in 1999 (RFC 2246) to resolve dispute between Netscape Communications (SSL) and Microsoft (PCT/STLP)
- Further development focused on TLS, resulting in TLS 1.1 (RFC 4346) release in April 2006, as well as Datagram TLS (DTLS) protocol 1.0 (RFC 4347)
- TLS 1.2 has been released in August 2008 (RFC 5246)
- TLS 1.3 has been released in August 2018 (RFC 8446) , removal of MD5 and SHA-224 among other features

SSL/TLS caveats

- SSL/TLS is supposed to achieve point-to-point secure communication
- SSL/TLS is not a complete and universal security platform
- do not solve problems such as:
  - SQL injection
  - Cross-site scripting (XSS)
  - Cross-site request forgery (CSRF)
- Using SSL/TLS may impose problems with implementing security policy (content screening) and may require installing a SSL/TLS proxy

MITM attacks

- in theory – protection against ,,Man in the Middle" attacks
- in practice – problems with certificate control and theauthentication chain

SSL/TLS implementation problems

- Misinterpretations of certificate verification error codes (recent: Polish local elections Nov 2014)
- (8.04.2014) – Heartbleed attack (OpenSSL 1.0.1–1.0.1e – buffer overread)
- (10.2014) – 'poodle attack' ('Padding Oracle On Downgraded Legacy Encryption') on SSL 3.0 – drawbacks of mainaining interoperability between different systems)
- (15.03.2022) – 'Infinite loop reachable when parsing certificates' (CVE-2022-0778)

Definitions and standards

- ANI X.509 – certificates
- RFC 2246 – TLS
- RFC 2487 – SMTP over TLS
- RFC 2818 – HTTPS (`https://...`, port 443)
- RFC 4346 – TLS 1.1
- RFC 5246 – TLS 1.2
- RFC 8446 – TLS 1.3

Using SSL/TLS

- OpenSSL (`http://www.openssl.org/`)
- Apache SSL (`http://www.apache-ssl.org/`)
- mod_ssl (`http://www.modssl.org/`)

SSL – Configuration

- Verification of the SSL version and available encryption methods

```
linux% openssl version -a
OpenSSL 0.9.8o 01 Jun 2010
platform: debian-amd64
options:  bn(64,64) md2(int) rc4(ptr,char) des(idx,cisc,16,int) blowfish(ptr2)
...
linux% openssl ciphers -v
DHE-RSA-AES256-SHA   SSLv3 Kx=DH     Au=RSA   Enc=AES(256)   Mac=SHA1
DHE-DSS-AES256-SHA   SSLv3 Kx=DH     Au=DSS   Enc=AES(256)   Mac=SHA1
AES256-SHA           SSLv3 Kx=RSA    Au=RSA   Enc=AES(256)   Mac=SHA1
DES-CBC3-SHA         SSLv3 Kx=RSA    Au=RSA   Enc=3DES(168)  Mac=SHA1
DES-CBC3-MD5         SSLv2 Kx=RSA    Au=RSA   Enc=3DES(168)  Mac=MD5
AES128-SHA           SSLv3 Kx=RSA    Au=RSA   Enc=AES(128)   Mac=SHA1
...
```

- Verifying where SSL keeps its files: `openssl version -d` (usually */usr/lib/ssl*)
  - *openssl.conf* – CA configuration, certificate directories, default value
  - *cert.pem* – set of known certificates (Comodo, DigiCert, DigiNotar, Thawte, TERENA)
  - *certs* subdirectory – individual *.pem* files with CA certificates
  - symlinks in *certs* directories to certificate files. Symlink names – cert hashes

SSL/TLS – Certificates

- Certificate Name – the name that will identify the newly created certificate.
- Expiry Date – allows controlling the time period for which this certificate can be used.
- Common Name – This represents the 'issued to' identifier for the certificate. Because the certificate will be self signed, this will also be the 'issued by' identifier.
- Email Address – the email address that any queries or other contact related to the certificate should be directed to.
- Organisation – the name of the organisation that the certificate is intended for.
- Department – the department that the certificate is intended for within an organisation.
- City / Town – the city or town where this organisation is based.
- State / Province – the state or province where the organisation using the certificate is based.
- Country – the country where the organisation using the certificate is based.
- Private Key Length – the size of the private key that will be used with the certificate. A 2048-bit key will provide stronger security than a 1024-bit key.

SSL/TLS – Certificates

- Certificate creation

```
openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout mycert.pem -out
    mycert.pem
```

  - Most questions do not need explanation (Country Name, State, City, itd.)
  - ,,Common Name" – DNS name of the serveror its CNAME
  - Certificate creation can be automated thanks to `-subj` option:

```
openssl req  -x509 -nodes -days 365  -newkey rsa:1024 \
  -keyout mycert.pem -out  mycert.pem \
  -subj '/C=PL/ST=Dolnoslaskie/L=Wroclaw/CN=www.jakasfirma.pl'
```

  - Minimalistic certificate must contain the CN field, the rest is optional.
- Using the `-subj` option:
  - `/C=`*PL* – country, `/ST=`*Dolnoslaskie* – state, voivodship, region, etc.
  - `/O=`*Wroclaw University of Technology* – organization
  - `/OU=`*Electronics Dept* – organizational unit, department, etc.
  - `/CN=`**www.pwr.wroc.pl** – common name – DNS name
  - `/emailAddress=`*Some.Address@pwr.wroc.pl* – contact person's email address
- Checking the certificate:

```
linux% openssl verify mycert.pem
mycert.pem: /C=PL/ST=Dln/L=Wroclaw/O=PWr/OU=Electronics/CN=www.pwr.wroc.pl/emailAddress=S
error 18 at 0 depth lookup:self signed certificate
OK
```

SSL/TLS – certificate signing

1. Certificate creation means generating ,,certificate signing request"

```
openssl req -new -key privkey.pem -out cert.csr
```

✪ Checking the contents of this request:

```
openssl req -in cert.csr -noout -text
```

2. Such a ,certificate reqest" (plik *cert.csr*) may be sent to the CA for signing or one can sign it oneself (especially when having someone's own CA)

```
# Individual CA in demoCA subdir (demoCA/cacert.pem, demoCA/private/cakey.pem)
openssl ca -policy policy_anything -out newcert2.pem -infiles newreq.pem
```

1. Creation of a self-signed certificate:

```
openssl genrsa -des3 -out privkey.pem 2048
openssl req -new -x509 -key privkey.pem -out cacert.pem -days 1095
```

- First command generates the key, the second one - the certificate.
- The `-des3` option locks cert with a password (otherwise – a paswordless certificate)
- If the password-protected certificate is to be used by the WWW server, it will be necessary to manually unlock the certificate each time the server is restarted.

✪ Verification:

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

- Selective checking: – with `-issuer`, `-subject`, `-dates`, `-hash`, `-fingerprint`

SSL/TLS – Individual CA

- To create one's own CA it is best to use `CA.pl` or `CA.sh` scripts (located in */usr/lib/ssl/misc/* directory by default)
- The default */usr/lib/ssl/openssl.cnf* file points to *./demoCA* directory– CA certificates, database of singed certificates and other files.
- A different config may be used by setting `OPENSSL_CONF` environment variable.

Basic usage:

- `CA -newca` – Creation of a new CA
- `CA -newreq [-nodes]` – New certificate request (unsigned certificate)
- `CA -sign` – certificate signing
  Signs cert request file *newreq.pem* and stores the signed result in *newcert.pem*

Newly signed certificate is also placed in the CA's database (*demoCA/newcerts/* directory)

- Error message ,,failed to update database" means, that the certificate being signed is already in the database (it can be removed by `openssl ca -revoke xyz.crt`)
- Default values (e.g. `countryName_default=AU`) used by CA and when creating new cert signing requests may be modified in */usr/lib/ssl/openssl.cnf* file.

Certificates – samples

- Creating cert-request, signing it by CA:

```
% openssl req -new -key privkey.pem -out new4.csr
% openssl ca -policy policy_anything -out newcert4.pem -infiles new4.csr

Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for ./demoCA/private/cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 10527901524476118302 (0x921a9ea844b1451e)
        Validity  Not Before: Nov 30 12:50:35 2016 GMT,  Not After: Nov 30 12:50:35 2017 GMT
        Subject:
            countryName               = PL              organizationName          = ccc
            stateOrProvinceName       = aaa             organizationalUnitName    = ddd
            localityName              = bbb             commonName                = eee
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                E0:66:B6:4A:19:7C:88:52:ED:7C:1B:6E:A5:40:86:0C:74:DA:71:B0
            X509v3 Authority Key Identifier:
                keyid:F2:31:E2:3D:1A:38:F9:E9:34:A2:E8:E1:FB:67:33:33:ED:FF:B3:AE
Certificate is to be certified until Nov 30 12:50:35 2017 GMT (365 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
% a signed certificate also goes to the database:
% -rw-r--r-- 1 ts ts 3378 2011-11-01 19:32 A7AAD5EE0ECE77CE.pem
% -rw-r--r-- 1 ts ts 3876 2011-11-01 21:36 A7AAD5EE0ECE77D0.pem  <--
% -rw-rw-r-- 1 ts ts 4536 lis 30 12:34 921A9EA844B1451D.pem
```

Certificates – samples

- Verification of a signed certificate:

```
demoCA% openssl x509 -text -in newcerts/A7AAD5EE0ECE77D0.pem
Certificate:
  Data: Version: 3 (0x2)
        Serial Number:    a7:aa:d5:ee:0e:ce:77:d0
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=PL, ST=Dolnoslaskie, O=Demo CA org, CN=laptok CA/emailAddress=laptok@laptok
        Validity:     Not Before: Nov  1 20:36:03 2011 GMT, Not After : Oct 31 20:36:03 2012 GMT
        Subject: C=PL, ST=aaa, L=bbb, O=ccc, OU=ddd, CN=eee
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:d4:be:ee:2a:ef:df:e7:4a:aa:9b:0d:65:7b:f7:
        ...
                    2d:28:ae:ad:7e:08:b0:70:a6:28:7d:8c:e5:3f:ac:15:5d
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:          CA:FALSE
            X509v3 Subject Key Identifier:     CA:B1:FD:89:DD:04:5F:A0:E3:9E:C7:50:3A:01:88:1F:8F:9E:8F:E9
            X509v3 Authority Key Identifier:   keyid:BC:7D:A2:9B:2D:FA:F9:51:6E:73:82:2B:30:02:EC:E3:33:2A:D7:40
    Signature Algorithm: sha1WithRSAEncryption
        a8:ea:f2:52:ea:83:45:44:0b:06:6b:9d:30:6e:ae:2c:7b:be:
        ...
        03:98:74:0f:90:d5:43:08:93:c6:b6:09:b4:ad:9c:08:d5:aa:cb:6c
-----BEGIN CERTIFICATE-----
MIIDPDCCAqWgAwIBAgIJAKeq1e4OznfQMA0GCSqGSIb3DQEBBQUAMGwxCzAJBgNV
...
nEzxzqMbRtwVhbl5IhEo2tK8fwSpm6iiI1zWbO+euHBJyFpOJaL9jjLMJ1gDmHQP
kNVDCJPGtgm0rZwI1arLbA==
-----END CERTIFICATE-----
```

SSL/TLS – Using certificates

In the Apache server:

- */etc/apache/mod_ssl.conf* config file:

```
SSLCertificateKeyFile /etc/apache/ssl.key/server.key
SSLCertificateFile /etc/apache/ssl.crt/server.crt
```

These 2 files should contain the previously generated key and the signed certificate (*.pem* files).

- If the key is locked with an additional password, each server startup will require manual password unlocking.
- Unlocking the key and storing a password-free version:

```
openssl rsa < server.key > unencrypted.key
```

From the client side (remote access to the SSL-enabled HTTP server and its certificate):

- Fetching a certificate from the remote WWW server:

```
openssl s_client -connect ad.res.serwe.ra:443 | \
  sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > cert.pem
```

- Checking the certificate contents:

```
openssl x509 -noout -subject -dates -in cert.pem
openssl x509 -noout -text -in cert.pem
```

SSL – Using client certificates

Client certificates:

- Client certificates are created the same way as server's
- Must be signed by CA recognized by the server
- Authentication is based on signature verification and matching of signed client data
- Authentication rules – in `mod_ssl` module configuration

Using client cerificates

- Apache can force all the clients to authenticate using certificates.
- Some URLs may require clients to authenticate using certificates, while the rest of the WWW server may be accessed by anymone.
- Access can be given based on user name in certiticate, or any other info (e.g. – organization name) to provide group access

More info: (http://httpd.apache.org/docs/2.2/ssl/ssl_howto.html.en#upgradeenc)

SSL/TLS – client certificates

Client certificate is nothing else than a typical certificate generated for a user, signed by CA and written in PKCS12 format:

1. Key (*user.key*) and certificate (*user.csr*) creation means generating the ,,cert signing requests". This key will be password-protected (disable with `-nodes`)

```
openssl req -new  -newkey rsa:1024 -keyout user.key -out user.csr
```

- Verification of request contents:

```
openssl req -in user.csr -noout -text
```

2. Signing this request with CA's certificate:

```
# Your own CA in demoCA/ (demoCA/cacert.pem, demoCA/private/cakey.pem)
openssl ca -policy policy_anything -out user.pem -infiles user.csr
```

3. Merging the signed certificate (*user.pem*) with the key (*user.key*) into one file:

```
cat user.key user.pem  > user2.pem
```

4. Exporting user certificate to a PKCS12 file:

```
openssl pkcs12 -export -out user.pfx -in user2.pem -name "User Cert"
Enter pass phrase for user2.pem: ******
Enter Export Password: ******
Verifying - Enter Export Password: ******
```

- The *user.pfx* file holds user's cert and key in PKCS12 format:

```
openssl pkcs12 -in  user.pfx -nodes
Enter Import Password:
MAC verified OK
   friendlyName: User Cert
   localKeyID: 84 EB F1 51 33 40 55 62 08 EC 99 86 FC E5 68 70 77 87 59 E4
subject=/C=PL/ST=aaaa/L=BBBB/O=cccc/OU=DDDD/CN=User Name/emailAddress=user@here
issuer=/C=PL/ST=Dolnoslaskie/O=Demo CA org/CN=laptok CA/emailAddress=ts@laptok
-----BEGIN CERTIFICATE-----
MIIC3DCCAkWgAwIBAgIJAKeq1e4OznfRMA0GCSqGSIb3DQEBBQUAMGwxCzAJBgNV
...
Wo4MSI19bUnc3Fojovg8CQ==
-----END CERTIFICATE-----
Bag Attributes
   friendlyName: User Cert
   localKeyID: 84 EB F1 51 33 40 55 62 08 EC 99 86 FC E5 68 70 77 87 59 E4
Key Attributes: <No Attributes>
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,971347A4DFA8E807
Bo5YL9BPbiI7iIyppQNnXe3QJCpscKNVQFs3ZyHPlExvZZ+L55V6aIeD2IRF8hkq
...
-----END RSA PRIVATE KEY-----
```

- The *user.pem* file (step ❷) contains CA-signed user certificate:

```
Issuer: C=PL, ST=Dolnoslaskie, O=Demo CA org, CN=laptok CA/emailAddress=
     laptok@laptok
Subject: C=PL, ST=aaaa, L=BBBB, O=cccc, OU=DDDD, CN=User Name/emailAddress=
     user@here
```