Einführung in SSL mit Wireshark

Chemnitzer Linux-Tage
16. März 2014

Martin Kaiser

What?

- SSL/TLS is the most widely used security protocol on the Internet
- there's lots of parameters, options, extensions that make it difficult to understand SSL/TLS
  - create simple test scenarios to get started
- Wireshark can help analyze and understand SSL/TLS
  - in some cases, it's possible to decrypt captured traffic

Overview

- purpose of TLS
- record layer
- handshake
- test setup
- Wireshark and TLS
- decrypting TLS traffic with Wireshark

About me

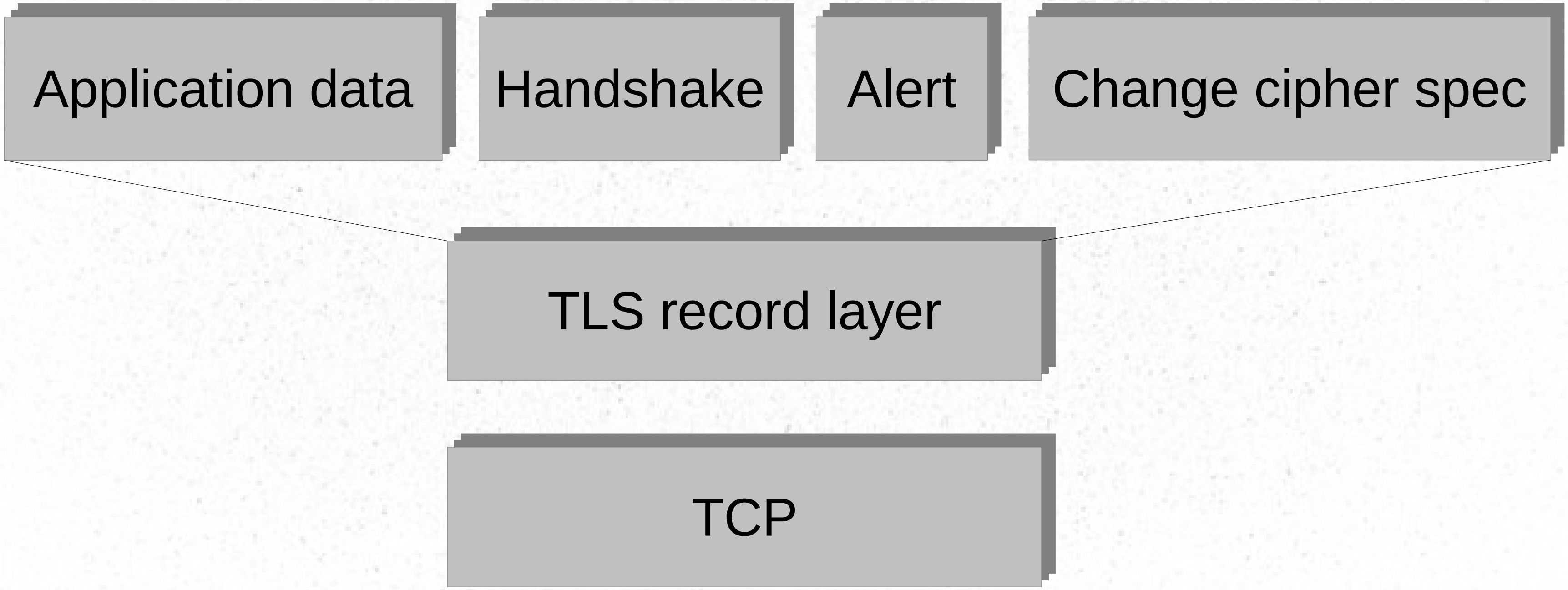
- writing embedded software for Digital TVs
- involved in creating the CI+ Pay-TV standard
- Wireshark Core Developer
- http://www.kaiser.cx

TLS

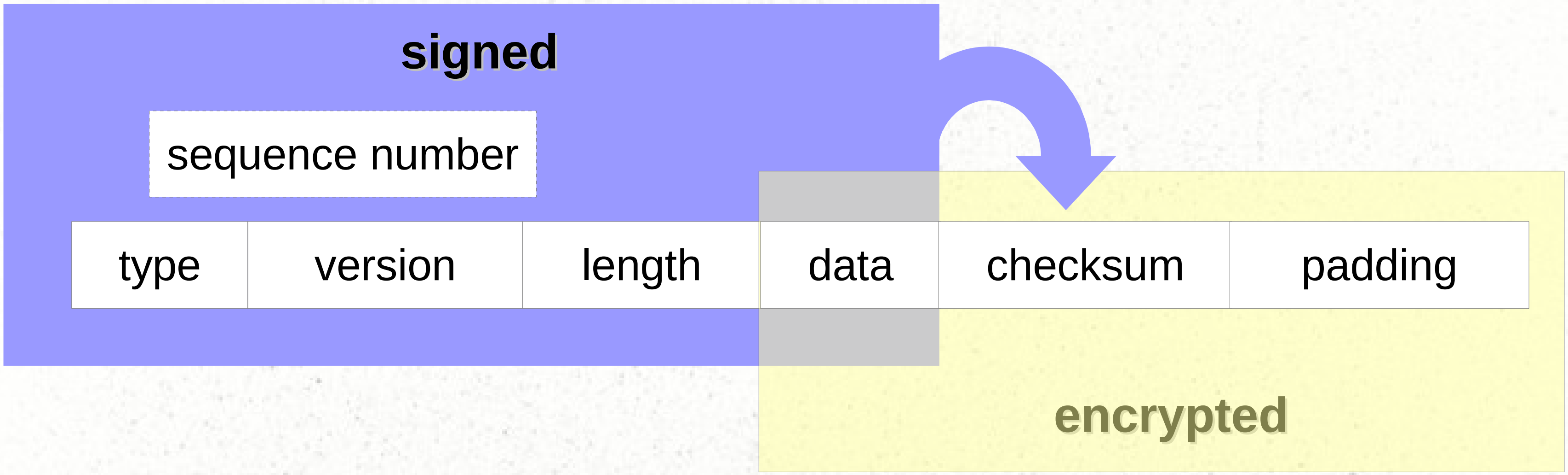
- Transport Layer Security
  - successor of SSL
  - TLS 1.2 defined in 2008, not widely deployed
- client and server
- runs on top of TCP
- transparent secure channel
  - encryption
  - authentication
  - compression

Freedigitalphots/photostock

TLS overview



Record layer



- the sequence number is not part of the message
- type is Application Data, Handshake, ...
- checksum is HMAC (key, hash algorithm)
- (compress-then-) sign-then-encrypt

Key material

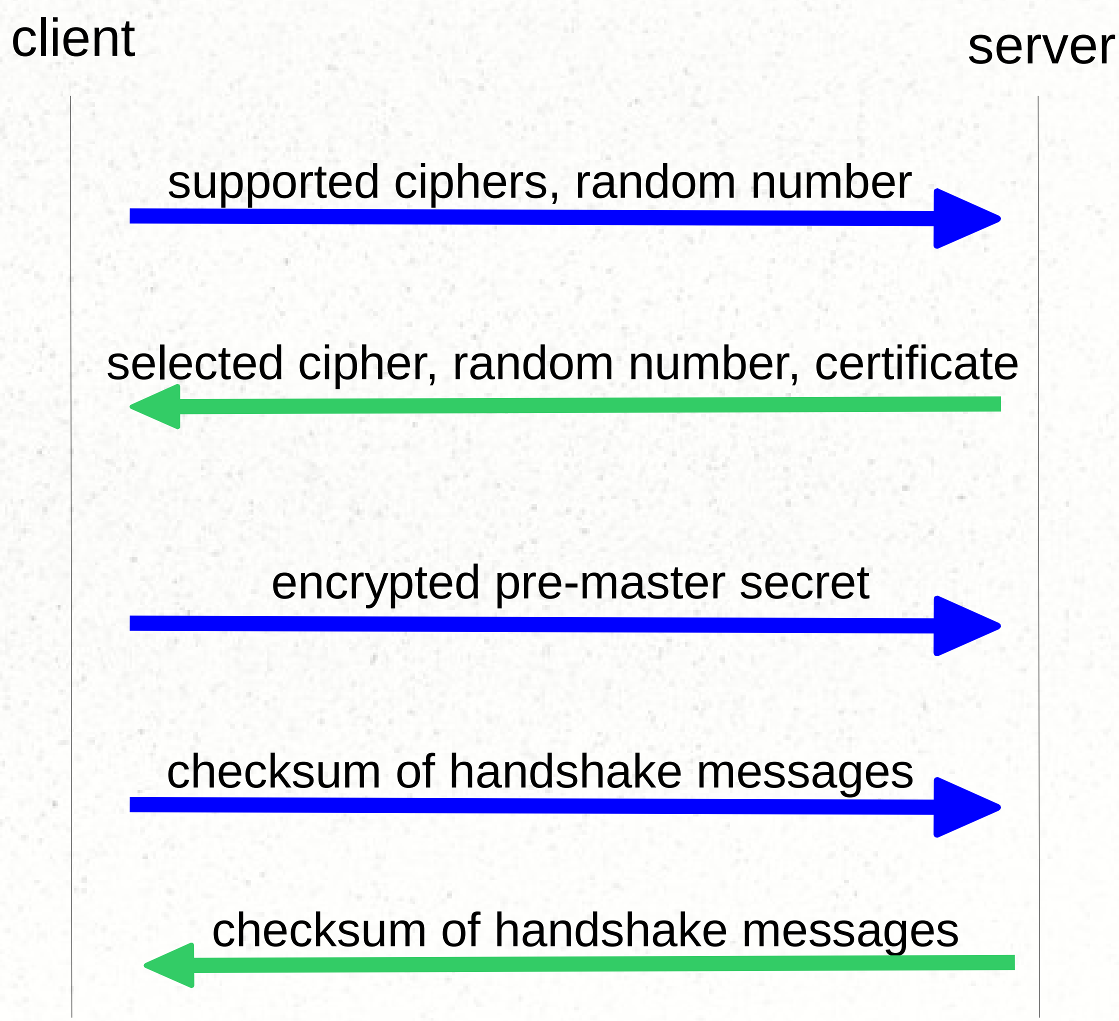
- pseudorandom function (PRF)
- pre-master secret (“result of the handshake”)
- master secret
  = PRF(pre-master secret, client random, server random, ...)
- key block
  = PRF(master secret, client random, server random, ...)
- split the key block into six keys
  - client HMAC key, server HMAC key
  - client encryption key, server encryption key
  - client init vector (IV), server init vector

Handshake

- agree on a set of ciphers
- client verifies the server's identity
- calculate the pre-master secret
- derive master secret and required keys
- verify the integrity of the handshake messages

Handshake

- verify server certificate
- create pre-master secret
- encrypt it with server's public key



Test setup

- a simple TLS client and server
  - OpenSSL command line tools
- server's private key
  - *openssl genrsa -out serverKey.pem 2048*
- server certificate
  - *openssl req -x509 -new -key serverKey.pem *
    *-out serverCert.pem *
    -subj "/C=DE/ST=Hessen/L=Frankfurt/
    O=private/OU=Martin Kaiser's server/
    CN=test.kaiser.cx/emailAddress=test@kaiser.cx"

Test client & server

- serve an info page on port 4433
  - *openssl s_server -accept 4433 \
    -cipher AES256-SHA -no_comp -www \
    -cert serverCert.pem -key serverKey.pem*
  - offer only one set of algorithms
  - don't support compression
- client
  - *openssl s_client -no_ticket -tls1*
  - localhost:4433 is the default target

Wireshark and SSL/TLS

- SSL and TLS up to version 1.2 are supported
- ASN.1 framework
  - dissect the server's X.509 certificate
  - generate protocol dissectors from ASN.1 modules
- decrypt captured TLS traffic
  - using the server's private key
  - using the master secret
  - gnutls, libgcrypt are required for this
    - *wireshark -v*

Demo: capture TLS traffic

Useful Wireshark settings

- in our example, TCP port 4433 is SSL
  → *Decode As*
  - this setting can be saved
- both client and server are on localhost
  - add columns for source and destination port
- Display Filter *ssl*
- *Follow TCP stream, Follow SSL stream*
- Time Shift to see the time difference between TLS messages

Cipher Suites

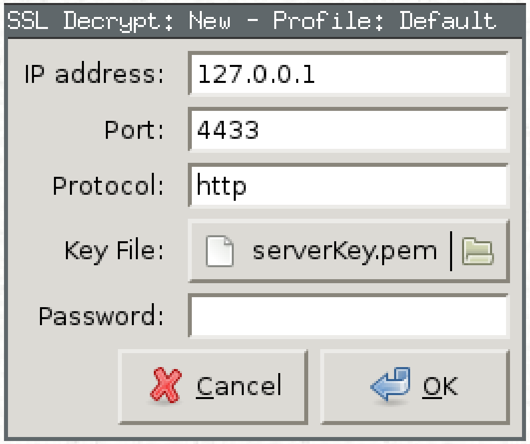
- Cipher Suite == a set of algorithms
  - type of server's keypair
  - algorithm used for negotiating the pre-master secret
    - some cipher suites use server's keypair directly
  - record-layer's encryption algorithm
  - record-layer's MAC algorithm
- *TLS_<key-exchange>_<auth>_WITH_<enc>_<mac>*

Cipher Suite Example

- TLS_RSA_WITH_AES_256_CBC_SHA
  - server has an RSA keypair
  - RSA is used for pre-master secret calculation
  - record layer encryption uses AES 256 in CBC mode for encryption
  - record layer uses HMAC-SHA-1 for message authentication

Decrypt TLS traffic using the server's private key

- Edit / Preferences / Protocols / SSL / RSA keys list



- Protocol *data* simply shows the decrypted bytes
- Wireshark decrypts the pre-master secret, calculates the master secret and the key block

Demo: decrypt TLS traffic

Export PDU mechanism

- strip off all layers below the TLS payload
- the resulting packets can be interpreted without any key material
- *File / Export PDUs to file*
- experimental

Session resumption

- speed up the handshake, skip the public key calculations
- initial connection
  - server assigns a *session ID*
  - client and server cache the master secret
- subsequent connection
  - client sends the session ID to resume the session
  - client and server use the cached master secret
    - new random numbers
      → unique key material for each connection
- decryption with the server's private key requires a capture with the initial handshake

Session resumption in practice

- *openssl s_client -no_ticket -tls1 -sess_out s1.dat*
  - cache information for session resumption
- *openssl sess_id -in s1.dat -noout -text*
  - display the cached session info
- *openssl s_client -no_ticket -tls1 -sess_in s1.dat*
  - resume a session based on cached information

Demo: session resumption

Ephemeral cipher suites

- use an ephemeral (short-lived) key for generating the pre-master secret
  - server's key pair is not used directly
  - ephemeral key is linked to the server's key pair
- additional handshake message *ServerKeyExchange*
- *forward secrecy*:
  if the server's private key is compromised, it can't be used for decrypting captured TLS traffic

Testing an ephemeral cipher suite

- *DHE-RSA-AES256-SHA*
  - server certificate contains an RSA keypair
  - Diffie-Hellman is used for calculating the pre-master secret
  - the server signs its Diffie-Hellman public key with its RSA private key
  - the record layer uses AES-256 in CBC mode, HMAC-SHA1
- *openssl s_server -accept 4433 \\
  -no_comp -cipher DHE-RSA-AES256-SHA -www \\
  -cert serverCert.pem -key serverKey.pem*

Demo:
ephemeral cipher suite

Decrypt TLS traffic using the master secret

- session resumption, ephemeral keys
  - the server's private key is not sufficient to decrypt TLS traffic
- provide the master secret to Wireshark directly
- key file
  - RSA Session-ID:<sess_id> Master-Key:<master secret>
  - CLIENT_RANDOM <client_random> <master secret>
  - RSA <8 bytes enc pre-master secret> <pre-master secret>

How to create a key file

- Wireshark
  - *File / Export SSL session keys*
  - only when Wireshark can already decrypt the TLS traffic
    - e.g. because it has the server's private key
- use OpenSSL's cached session info
  - *openssl sess_id -in s1.dat -noout -text*
  - some tweaking is required to get the data into the correct format
- applications based on NSS (e.g. chrome, firefox)
  - *export SSLKEYLOGFILE=./out.log && firefox*

Demo: TLS decryption using the master secret

Summary

- to understand TLS, start with simple scenarios
- Wireshark can decrypt TLS traffic
  - using the server's private key
  - using the master secret
- please let us know if you have some TLS traces that Wireshark doesn't fully support

Thank you for your attention.

Freedigitalphotos/Master Isolated Images