MTAT.07.017  
Applied Cryptography

Transport Layer Security (TLS)

University of Tartu

Spring 2017
Transport Layer Security

“TLS is cryptographic protocol that provides communication security over the Internet.”

- Provides confidentiality, integrity and server authentication
- Most successful and widely used cryptographic protocol (!!!)
- Any application protocol can be encapsulated in TLS
TLS History

- SSL 1.0 – never publicly released
- SSL 2.0 – Netscape (1995)
- SSL 3.0 – Netscape (1996)
- **TLS 1.0** (SSL 3.1) – RFC 2246 (1999)
- TLS 1.1 – RFC 4346 (2006)
- TLS 1.2 – RFC 5246 (2008)
- TLS 1.3 – RFC draft (2016)

No fundamental changes between versions

http://www.ietf.org/rfc/rfc2246.txt
TLS Record Layer

[Type] [Version] [Length] [Data]

- **Type**: content type of encapsulated data:
  - Handshake message (0x16)
  - Change cipher spec message (0x14)
  - Alert message (0x15)
  - Application data (0x17)
- **Protocol version**: 0x0301 (for TLS 1.0)
- **Length**: length of the data (2 bytes)
- **Data**: encapsulated content
  - Can contain several same type messages

Header is never encrypted!
TLS Handshake

ClientHello

ServerHello, Certificate, ServerHelloDone

ClientKeyExchange

[ChangeCipherSpec], Finished

[ChangeCipherSpec], Finished

[Application Data]

- Client verifies server’s X.509 certificate
- Client extracts from the certificate server’s public key
- Client encrypts random symmetric key using public key
- Only the server can decrypt symmetric key
- Now the client and server share the same symmetric key
- Symmetric key used for actual data encryption/authentication
Dissecting TLS with Wireshark
Alert Message

Signals about TLS related issues to other party

[Level][Description]

- Level (1 byte):
  - Warning (0x01)
  - Fatal (0x02)
- Description (1 byte):
  
  close_notify(0),
  unexpected_message(10),
  bad_record_mac(20),
  decryption_failed(21),
  handshake_failure(40),
  bad_certificate(42),
  unsupported_certificate(43),
  certificate_revoked(44),
  certificate_expired(45),
  illegal_parameter(47),
  unknown_ca(48),
  access_denied(49),
  decrypt_error(51),
  user_canceled(90),
  ...

TLSv1 Record Layer: Alert (Level: Fatal, Description: Certificate Unknown)

Content Type: Alert (21)
Version: TLS 1.0 (0x0301)
Length: 2

Alert Message
Level: Fatal (2)
Description: Certificate Unknown (46)
Change Cipher Spec Message

Signals to other party that from now on the negotiated cipher suite will be used to protect outgoing messages
[0x01]
Application Data

Contains (most likely encrypted) application data in a form as required by the application protocol (e.g., HTTP request/response etc.)

[Application Data]
Handshake Message

Negotiates TLS protocol security parameters
[Type] [Length] [Body]

- **Type**: message type:
  
  - hello_request(0), client_hello(1), server_hello(2), certificate(11), server_key_exchange (12), certificate_request(13), server_hello_done(14), certificate_verify(15), client_key_exchange(16), finished(20)

- **Length**: length of the body (3 bytes)

- **Body**: message body
  
  - Can be split over several records
Handshake Message: client_hello

- Highest TLS version supported (2 bytes)
- Client randomness (32 bytes)
  - Timestamp in first 4 bytes
- Session ID length (1 byte) + session ID
- Cipher suite length (2 bytes)
- List of cipher suites supported:
  - 0x0005 – TLS_RSA_WITH_RC4_128_SHA
  - 0x002f – TLS_RSA_WITH_AES_128_CBC_SHA
  - 0x0035 – TLS_RSA_WITH_AES_256_CBC_SHA
  - 0x0039 – TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- Compression method length (1 byte)
- List of compression methods supported:
  - 0x00 – null (mandatory)
  - 0x01 – DEFLATE (gzip)
- Extensions (optional)
Handshake Message: server_hello

- TLS version selected (2 bytes)
- Server randomness (32 bytes)
  - Timestamp in first 4 bytes
- Session ID length (1 byte) + session ID
- Cipher suite selected (2 bytes)
- Compression method selected (1 byte)
- Extensions (optional)
Handshake Message: certificate

- Length of certificate list (3 bytes)
- List of certificates
  - Certificate length (3 bytes)
  - DER encoded certificate
- The first is server’s certificate
- Other certificates are optional
  - Usually intermediate CA certificates
Handshake Message: server_hello_done

- Empty message body
- Tells that there will be no more messages from the server in this protocol round
Handshake Message: client_key_exchange

Contains (two byte length-prefixed) encrypted 48 byte random “pre-master secret”

- Encrypted using public key from the server certificate.
- Encrypted according to PKCS#1 v1.5
- First two bytes in premaster secret contain TLS version
  - Must be checked by the server
  - Prevents some attacks (?)
- Next 46 bytes are truly random bytes
Handshake Message: finished

- The first encrypted message
- Serves to verify if encryption works
- Contains hash of concatenation of all previous handshake messages (excluding TLS record header)
  - Must be verified by other party to detect downgrade attacks
Encryption

- Plaintext compression leaks information (CRIME attack)
- How many symmetric keys are needed?
  - MAC & encrypt (+ IV for block ciphers)
    - Separate keys for both directions

How to derive these keys from 48 byte pre-master secret?
Key Derivation

- TLS defines PRF() (pseudo-random function)
  - Uses MD5() and SHA1()
  - Produces indefinitely long pseudo-random output
- 48 byte “master secret” is derived:
  \[
  \text{PRF}(\text{premaster} + "\text{master secret}") + \text{client_random} + \text{server_random}, 48)
  \]
- From “master secret” is derived key block in the size needed:
  \[
  \text{PRF}(\text{master_secret} + "\text{key expansion} + \text{server_random} + \text{client_random}, 136)\]
- Key block is split into keys needed:
  - \text{client_mac_key} = \text{key_block}[0:20]
  - \text{server_mac_key} = \text{key_block}[20:40]
  - \text{client_enc_key} = \text{key_block}[40:56]
  - \text{server_enc_key} = \text{key_block}[56:72]
  - \text{client_iv} = ...
  ...
  ...

MAC Calculation

\[
\text{HMAC\_digest(key, seq + type + version + length + data)}
\]

- \text{digest: digest algorithm from negotiated cipher suite}
- \text{key: [client/server] MAC key}
- \text{seq: [client/server] Sequence number (8 bytes)}
  - Starts from 0
  - Increased for every encrypted TLS record
- \text{type: TLS record content type}
- \text{version: TLS protocol version (2 bytes)}
- \text{length: length of the content (2 bytes)}
- \text{data: content}

Type, version and length are fields from TLS record header! This way integrity for TLS record header is also provided!
Task

Implement TLS 1.0 client that can retrieve server’s certificate.

$ ./tls_getcert.py https://www.eesti.ee/ --certificate server.pem
--> client_hello()
<--- handshake()
  <--- server_hello()
    [+] server randomness: 0A801E9E809F15D7BCDB3A4F9640A3395480E7EF41FC9E6BD9B9438ECD67
    [+] server timestamp: 1975-08-02 02:43:26
    [+] TLS session ID: B268B48206AC28679B315CAB6CF5D0EEB5B0E50A973097EF1AFE20C23E8520D
    [+] Cipher suite: TLS_RSA_WITH_AES_256_CBC_SHA
<--- handshake()
  <--- certificate()
    [+] Server certificate length: 1636
    [+] Server certificate saved in: server.pem
<--- handshake()
  <--- server_hello_done()
--> alert()
    [+] Closing TCP connection!

$ openssl x509 -in server.pem -text | grep 'Subject:'
Subject: C=EE, ST=Harjumaa, L=Tallinn, O=Estonian Information System’s Authority, CN=*.eesti.ee
$ ./tls_getcert.py https://www.e-toimik.ee/
--> client_hello()
<--- handshake()
    <--- server_hello()
        [+] server randomness: 572C59A68A3479C28E3C0A5C369AA5C2EB72CABA690B7B22E480EE07F40A
        [+] server timestamp: 2017-05-05 11:03:48
        [+] TLS session ID: 7C15B1658AFB6768668C86228591E1F5633D553372305F4A276EBCBECA628B4
        [+ ]Cipher suite: TLS_RSA_WITH_AES_128_CBC_SHA
    <--- handshake()
    <--- certificate()
        [+ ]Server certificate length: 1363
    <--- handshake()
--> server_hello_done()
--> alert()
[+] Closing TCP connection!

$ ./tls_getcert.py https://www.nordea.ee/
--> client_hello()
<--- handshake()
    <--- server_hello()
        [+ ]server randomness: 572C59EAC608E3F0446204AB3303A57E9D53553761992107354E971F6A12
        [+ ]server timestamp: 2017-05-05 11:04:31
        [+ ]TLS session ID: 00000000000000000000000000000000100003111572C5C4200000000053666B
        [+ ]Cipher suite: TLS_RSA_WITH_AES_128_CBC_SHA
    <--- certificate()
        [+ ]Server certificate length: 1521
    <--- server_hello_done()
--> alert()
[+] Closing TCP connection!
Task: Hints

- Compare your parsed output with output from Wireshark
  - Use capture filters 'host 1.2.3.4 and port 443'
- NB! One TLS record can contain several handshake messages
- Unix timestamp can be obtained using `int(time.time())`
- Unix timestamp can be printed using:
  ```
  datetime.datetime.fromtimestamp(int(time.time())).strftime('%Y-%m-%d %H:%M:%S')
  ```