MTAT.07.017
Applied Cryptography

Transport Layer Security (TLS)
Advanced Features

University of Tartu

Spring 2020
Server-authenticated TLS

Client usually is authenticated on the application level by some shared secret (e.g., password). This can go wrong:

- Server can be impersonated
- Server can be compromised
- Password can be reused in another service
- Password can be guessed
- Password can be phished
Client Certificate Authentication

- CertificateVerify – signature over all handshake messages
- Can CertificateVerify be reused in another handshake?
- Why CertificateVerify is after ClientKeyExchange?
- Client’s Certificate is sent before ChangeCipherSpec
- Client proves its identity by signing and not by decrypting
- Solves most of the problems of password authentication
Renegotiation

• Any party can initiate negotiation of a new TLS session:
  • Client by sending ClientHello
  • Server by sending HelloRequest

• Handshake messages of the new TLS session are protected by the cipher suite negotiated in the previous TLS session

• Used by the server to renegotiate stronger cipher suite or to request a client certificate authentication if on the application level a client tries to access some resource that require such security measure

• Client-initiated renegotiation usually disabled on the server
Certificate request on renegotiation

Client

ClientHello

ServerHello, Certificate, ServerHelloDone

ClientKeyExchange

[ChangeCipherSpec], Finished

[ChangeCipherSpec], Finished

Application Data (GET /auth HTTP/1.1)

HelloRequest

ClientHello

ServerHello, Certificate, **CertificateRequest**, ServerHelloDone

**Certificate**, ClientKeyExchange, **CertificateVerify**

[ChangeCipherSpec], Finished

...
If the Key Exchange type is RSA:

- If we can get a hold of the server’s RSA private key, we can decrypt the Client Key Exchange message and read the pre-master secret key. No other heavy work need be done.
- Valid for life of certificate

Can we prevent it?
Perfect Forward Secrecy

PFS is achieved by using the server’s long-term private key to authenticate a short-term/ephemeral asymmetric key that is used to encrypt the actual data.

Benefits:
- Attacker who has compromised server’s private key cannot decrypt network traffic
  - Attacker has to execute active MITM attacks
- Attacker has to crack $x$ asymmetric keys to decrypt $x$ sessions made to the server

Used in TLS cipher suites: TLS_(EC)DHE_RSA_WITH_*
Diffie-Hellman (DH) Key Exchange

\[ (2^5)^4 = 2^{5\cdot4} = (2^4)^5 \]

- In practice multiplicative group of integers modulo p is used
- Discrete logarithm problem:
  - hard to find \( x \), given \( 2^x = 32 \mod p \)
- ElGamal and DSA based on DL problem
- Secure against passive eavesdropping
(EC)Diffie-Hellman Key Exchange

- ServerKeyExchange contains DH group, server's DH public key and server's RSA signature over DH public key, client randomness and server randomness
- ClientKeyExchange contains client's DH public key
- How is “pre-master secret” calculated?
- Handshake requires two public key operations (DH+RSA)
- Achieves perfect forward secrecy
TLS extensions

• ClientHello can contain length-prefixed extensions
• ServerHello will contain a response to client’s extensions
• Most popular extensions:
  • Server Name Indication (SNI) extension (RFC 3546)
    ```
    Extension: server_name
    Type: server_name (0x0000)
    Length: 17
    Server Name Indication extension
    Server Name list length: 15
    Server Name Type: host_name (0)
    Server Name length: 12
    Server Name: www.eesti.ee
    ```
  • TLS Session Tickets (RFC 5077)
    ```
    Extension: SessionTicket TLS
    Type: SessionTicket TLS (0x0023)
    Length: 180
    Data (180 bytes)
    ```
  • Elliptic Curves (RFC 4492)
    ```
    Extension: elliptic_curves
    Type: elliptic_curves (0x000a)
    Length: 8
    Elliptic Curves Length: 6
    Elliptic curves (3 curves)
    Elliptic curve: secp256r1 (0x0017)
    Elliptic curve: secp384r1 (0x0018)
    Elliptic curve: secp521r1 (0x0019)
    ```
Task: TLS client – 6p

Implement TLS v1.2 client that can obtain HTTP GET response:

$ ./tls_client.py https://127.0.0.1:4433/
---> ClientHello()
<--- Handshake()
    ----> ServerHello()
        [+] server randomness: 5EA61AB07ED9CFF9312B60CB06CC62F779636AFD8CB28C0DC295FB7210AB9
        [+] server timestamp: 2020-04-27 02:35:39
        [+] TLS session ID:
        [+] Cipher suite: TLS_RSA_WITH_RC4_128_SHA
<--- Handshake()
    ----> Certificate()
        [+] Server certificate length: 554
<--- Handshake()
    ----> ServerHelloDone()
---> ClientKeyExchange()
---> ChangeCipherSpec()
---> Finished()
<--- ChangeCipherSpec()
<--- Handshake()
    ----> Finished()
---> Application_data()
GET / HTTP/1.0
<--- Application_data()
HTTP/1.0 200 OK
Content-Length: 6
Hello!
[+] Closing TCP connection!
Task: TLS client

- Client has to support TLS_RSA_WITH_RC4_128_SHA cipher suite
- Template contains fully implemented PRF(), derive_master_secret(), derive_keys(), encrypt(), decrypt() and client/server Finished hash calculation code
  - Make sure that the correct inputs are provided to these functions (!!!)
- TLS client should work on www.facebook.com
- Grading:
  - 2 points if the server accepts your ClientKeyExchange message
  - 2 points if the server accepts your Finished message
  - 2 points if your code can show HTTP response
- tls_server binary can be used for development (port 4433)
- Wireshark “Decode As” – “TCP Destination 4433” – “SSL”
$ ./tls_server --port 4433
[+] Connection from 127.0.0.1:36098

--- Handshake()

--- ClientHello()

[+] version: 0303
[+] client randomness: 5EA61ACBF1615730E2F11004E6F2588A74D35C2A6B879F7B57CDEEB049B3C9CB
[+] client timestamp: 2020-04-27 02:35:39

--- ServerHello()

[+] server randomness: 5EA61ACB07EED9CFFE9312B60CB06CC62F779636AFD8CB28C0DC295FB7210AB9
[+] server timestamp: 2020-04-27 02:35:39

--- Certificate()

[+] Server certificate length: 554

--- ServerHelloDone()

--- Handshake()

--- ClientKeyExchange()

[+] PreMaster length: 128
[+] PreMaster (encrypted): 6dccc4be09228d6cf824b472148d94f83f3e7f95c1bf5aad382fb97afccfbeb4c830a1a00db45c7eff85258e40f9389be3b213cc0a18d94ab47ab054a0f787596e259d3c9b
[+] PreMaster: 03033a509170b0c36bad0a1ac4f371c51dabf6f75cd6988304e482894c456450d82e7ab031ddeebe4d5e666724d60f

--- ChangeCipherSpec()

[+] Applying cipher suite:

--- Handshake()

--- Finished()

[+] client_verify (received): 2612c639ae9ac7eb5dc84881
[+] client_verify (calculated): 2612c639ae9ac7eb5dc84881

--- ChangeCipherSpec()

--- Finished()

--- Application_data()

GET / HTTP/1.0

--- Application_data()

Hello!

--- Finished()
$ ./tls_client.py https://www.swedbank.ee/
--> ClientHello()
        ---- Alert()
        [-] fatal: 40

$ ./tls_client.py https://www.eesti.ee/
--> ClientHello()
        ---- Alert()
        [-] fatal: 40

$ ./tls_client.py https://www.facebook.com/
--> ClientHello()
        ---- Handshake()
          ---- ServerHello()
          [+] server randomness: E2F4273E8D8F964B08EA1290CF93
          [+] server timestamp: 2090-08-29 04:00:46
          [+] TLS session ID: AFBE4C852286AE87864B7DCCE510E6
          [+] Cipher suite: TLS_RSA_WITH_RC4_128_SHA
        ---- Handshake()
          [+] Server certificate length: 1798
        ---- ServerHelloDone()
--> ClientKeyExchange()
--> ChangeCipherSpec()
--> Finished()
        ---- ChangeCipherSpec()
        ---- Finished()
--> Application_data()
GET / HTTP/1.0
        ---- Application_data()
HTTP/1.1 301 Moved Permanently
Vary: Accept-Encoding
Location: https://www.facebook.com/
Content-Type: text/html; charset="utf-8"
X-FB-Debug: 301zzVa6cieErTa/IN7GigNt7c8r2cKUkrv4ap3wifQI6xoM/Chmv/v7OXXWhuwd4APl+FU+gI2wJaZ7vmw==
Date: Sun, 26 Apr 2020 22:26:34 GMT
Alt-Svc: h3-27=":443"; ma=3600
Connection: close
Content-Length: 0
        [+] Closing TCP connection!
Most common pitfalls

• Server fails to verify MAC of client’s Finished message
  • Make sure client’s Finished message is encrypted using the correct keys. Compare keys – if they are different make sure key derivation receives the correct values of premaster secret, client and server randomness

• Server fails to verify hash in client’s Finished message
  • Make sure all handshake messages sent and received are appended to the handshake_messages variable

• Client fails to verify hash in server’s Finished message
  • Plaintext version of client’s Finished message must be appended to the handshake_messages variable

• Server returns fatal Alert “decryption failed” after receiving client’s Finished message
  • Make sure the server did not choose non-RC4 cipher suite