MTAT.07.017

Applied Cryptography

Transport Layer Security (TLS)
Advanced Features

University of Tartu

Spring 2019
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 his identity by signing and not by decrypting
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 server to renegotiate stronger cipher suite or to request client certificate authentication if on application level client tries to access resources 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
Diffie-Hellman Key Exchange

- \((2^5)^4 = 2^{5 \cdot 4} = (2^4)^5\)

- In practice multiplicative group of integers modulo \(p\) is used

- Discrete logarithm problem
  - hard to find \(x\), given \(2^x = 32 \mod p\)

- Secure against passive eavesdropping

\[
\begin{align*}
\text{random number} &= 4 \\
2^4 &= 16 \\
32^4 &= 1,048,576 \\
\text{random number} &= 5 \\
2^5 &= 32 \\
16^5 &= 1,048,576
\end{align*}
\]
• 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)

• Used by TLS_(EC)DHE_RSA_WITH_* cipher suites

• Achieves perfect forward secrecy
Perfect Forward Secrecy

Benefits:

- Attacker who has compromised RSA private key cannot decrypt previous TLS traffic
- Attacker who has compromised RSA private key has to execute active MITM attack
- Attacker has to crack $x$ asymmetric keys to decrypt $x$ sessions made to the server

PFS is achieved by using the long-term private key to authenticate a short-term key that is used to encrypt the actual data.
Extensions

- ClientHello can contain length-prefixed extensions
- ServerHello will contain response to client’s extensions
- Most popular extensions:
  - Server Name Indication (SNI) extension (RFC 3546)
  - TLS Session Tickets (RFC 5077)
  - Elliptic Curves (RFC 4492)
  - Heartbeat (RFC 6520)
Task: TLS client – 5p

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

$ ./tls_client.py https://127.0.0.1:4433/
--> client_hello()
<--- handshake()
  <--- server_hello()
    [+] server randomness: 57359448EF20879409852D451B1A3089D620A95944BF8092
    [+] server timestamp: 2019-04-26 11:46:00
    [+] TLS session ID:
    [+] Cipher suite: TLS_RSA_WITH_RC4_128_SHA
<--- handshake()
  <--- certificate()
    [+] Server certificate length: 554
<--- handshake()
  <--- server_hello_done()

--> client_key_exchange()
--> change_cipher_spec()
--> finished()
<--- change_cipher_spec()
<--- handshake()
  <--- finished()

--> application_data()
GET / HTTP/1.0
<--- application_data()
HTTP/1.0 200 OK
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 you provide correct inputs to these functions (!!!)
- Your TLS client should work on www.facebook.com
- Grading:
  - 2 points if a server accepts your ClientKeyExchange message
  - 2 points if a server accepts your Finished message
  - 1 point if your code can show HTTP response
- You can use tls_server binary for development (port 4433)
- Wireshark “Decode As” – “TCP Destination 4433” – “SSL”
$ ./tls_server --port 4433
[+] Connection from 127.0.0.1:38452

---------

[---] handshake()
  
  [---] client_hello()
  [+] version: 0303
  [+]'client randomness: 5AE1C2C0A89495A695EFD7945EEBE629CE3AE6E42673172266072BF54EEE1BB9
  [+]'TLS session ID:
  [+]'Cipher suites:
    TLS_RSA_WITH_RC4_128_SHA
  [+]'Compression methods:
    null
  [+]'Extensions length: 0

  [---] server_hello()
  [+] server randomness: 5AE1C2C0B7A4C11ABF845064B3EC52D188A936C12DEC1FCEDF8BE5DA551F1
  [+] server timestamp: 2019-04-26 15:14:56
  [+] TLS session ID:
  [+] Cipher suite: TLS_RSA_WITH_RC4_128_SHA

  [---] certificate()
  [+] Server certificate length: 554

  [---] server_hello_done()

[---] handshake()
  [---] client_key_exchange()
  [+] PreMaster length: 128
  [+] PreMaster (encrypted): ....
  [+] PreMaster: 030362b7dc1497d02d377d34c30a446839214f32d48f5163a2979d614019ed8778048ff8c60cd97757b88a8bd6afdc5a

[---] change_cipher_spec()
  [+] Applying cipher suite:
    [+] master_secret = PRF(030362b7dc1497d02d377d34c30a446839214f32d48f5163a2979d614019ed8778048ff8c60cd97757b88a8b)
    [+] master_secret: c182ff31961f326b777b9ec627ba4b17b2ea9b0a606ba1c04be2d0b8347aa3a3d2f7de13880f07dbbf9909fbd
    [+] client_mac_key: 0c5dc9428c8f56fa6e62df3b2f837ce866623
    [+] server_mac_key: a304d7ae33435a757e0eb4efb2ca062354aefb
    [+] client_enc_key: 82f955c772a4e9b39c009188a149976f
    [+] server_enc_key: e9caad52b25f872a96b8d2d5657c7835

[---] handshake()
  [---] finished()
  [---] finished()

  [---] application_data()
  GET / HTTP/1.0

  [---] application_data()
  HTTP/1.0 200 OK

Hello!
[+] Closing TCP connection!
RC4 (TLS_RSA_WITH_RC4_128_SHA)

$ ./tls_client.py https://www.swedbank.ee/
--> client_hello()
    ---- alert()
        [-] fatal: 40

$ ./tls_client.py https://www.nordea.ee/
--> client_hello()
    ---- alert()
        [-] fatal: 40

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

$ ./tls_client.py https://www.facebook.com/
--> client_hello()
    ---- handshake()
        --- server_hello()
            [+] server randomness: 5D8ED830872A31CCE442E042D9B057657329122E6FC28B7F8EA1EFFDBF145F34
            [+] server timestamp: 2019-09-28 06:49:04
            [+] TLS session ID: E3279B254961A66F89C90D9CC419A564EDC9D795C9405F6A50574BA1036B1DD
            [+] Cipher suite: TLS_RSA_WITH_RC4_128_SHA
    ---- handshake()
        --- certificate()
            [+] Server certificate length: 1792
    ---- handshake()
        --- server_hello_done()
--> client_key_exchange()
--> change_cipher_spec()
--> finished()
    ---- change_cipher_spec()
    ---- handshake()
        ---- 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"
Date: Wed, 01 May 2019 15:37:46 GMT
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`.

- Server returns fatal *alert* “decryption failed” after receiving client’s *finished* message
  - Make sure the server did not choose non-RC4 cipher suite.