Lecture 18: Transport Layer Security Protocol Part 2

COSC362 Data and Network Security

Book 1: Chapter 17 – Book 2: Chapter 22

Spring Semester, 2021

Motivation Reminder

- TLS is the most widely used security protocol.
- TLS is used to secure communications with banks, online shops, email providers, etc.
- TLS uses most of the mainstream cryptographic algorithms.
- TLS is a very complex protocol.
- TLS has been subject of many attacks, and subsequent repairs.

Outline

Summary of Lecture 17

Attacks on TLS

TLS 1.3

TLS Protocols

1. Handshake protocol
2. Record protocol
3. Alert protocol

Handhsake Protocol

Process to start a communication session between a server and a client:

- Specify which version of TLS they will use (mostly TLS 1.2 or 1.3).
- Decide on which cipher suites they will use.
- Authenticate the identity of the server via the server's public key and the certificate authority's digital signature.
- Generate session keys in order to use symmetric encryption after the handshake is complete.

Steps with RSA Key Exchange

1. *"client hello" message:* TLS version and cipher suites supported by the client + N_C.
2. *"server hello" message:* certificate + chosen cipher suite + N_S.
3. *Authentication:* client checks certificate.
4. *Premaster secret using key transport:*
   - chosen by client and encrypted using server's public key.
   - decrypted using server's private key.
5. *Session keys:* computed using PRF, on each side.
6. *"client finished" message:* encrypted with a session key.
7. *"server finished" message:* encrypted with a session key.

The handshake is completed and communication continues using the session keys.

Steps with Diffie-Hellman Key Exchange

1. *"client hello" message:* TLS version and cipher suites supported by the client + N_C.
2. *"server hello" message:* certificate + chosen cipher suite + N_S.
3. *server's signature:* on N_C, N_S and server's Diffie-Hellman parameters using server's private key.
4. *Signature verification:* client checks signature + sends client's Diffie-Hellman parameters.
5. *Premaster secret using key agreement:* using exchanged Diffie-Hellman parameters.
6. *Session keys:* computed using PRF, on each side.
7. *"client finished" message:* encrypted with a session key.
8. *"server finished" message:* encrypted with a session key.

The handshake is completed and communication continues using the session keys.

Record Protocol

Guarantee confidentiality and integrity of application data using the session keys created during the handshake:

- Divide outgoing messages into manageable blocks and re-assemble incoming messages.
- (optional) Compress outgoing blocks and decompress incoming blocks.
- Apply a MAC to outgoing messages and verify incoming messages using the MAC.
- Encrypt outgoing messages and decrypting incoming messages.

When the Record Protocol is complete, the outgoing encrypted data is passed down to the TCP layer for transport.

Outline

Summary of Lecture 17

Attacks on TLS

TLS 1.3

Backward Compatibility

Backward compatibility is a problem:

- SSL 3.0 was deprecated in 2015.
- End of life for TLS 1.0 and 1.1 only in 2020.
- TLS 1.2 is still the most widely supported:
  - Supported by 99.5% of websites (August 2021).
- TLS 1.3 is slowly adopted:
  - Supported by 47.8% of websites (August 2021).

Limitations

- Many practical attacks on TLS in the past few years.
- Many servers:
  - do not support the latest TLS versions.
  - are not protected against known attacks.
- Example:
  - Recent attacks show that RC4 is vulnerable.
  - RC4 is offered in TLS 1.2.
  - TLS 1.3 has discarded it, but not widely supported.
- SSL Pulse gives an up-to-date picture:
  - `https://www.ssllabs.com/ssl-pulse/`
- Good coverage of attacks is given on Matt Green's blog:
  - `http://blog.cryptographyengineering.com`

BEAST Attack

- Browser Exploit Against SSL/TLS (BEAST).
- Exploiting non-standard use of IV in CBC mode encryption:
  - IVs are chained from the previous ciphertexts.
  - Allowing the attacker to recover the plaintext byte by byte.
- Stages:
  - 2002: theoretical weakness
  - 2011: practical weakness
  - Only random IV from TLS 1.1
  - No longer considered as a realistic threat
- Most browsers implement a mitigation strategy:
  - Splitting the plaintext into first byte and remainder to force a randomized IV including a MAC computation.

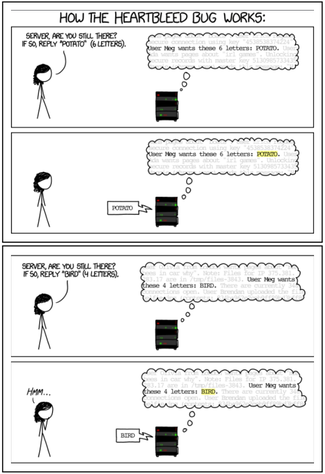
CRIME and BREACH Attacks

- Compression Ratio Info-leak Made Easy (CRIME).
- Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext (BREACH).
- Side channel attacks based on *compression*:
  - Different inputs result in different amounts of compression.
  - CRIME exploits compression in TLS.
  - BREACH exploits compression in HTTP.
- 2002: idea of the attack.
- Commonly recommended to switch off compression in TLS:
  - Compression not available in TLS 1.3.
- Switching off in HTTP results in big performance hit.

POODLE Attack

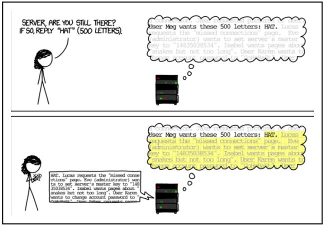
- Padding oracle: enabling an attacker to know if a message in a ciphertext was correctly padded.
- 2002: theoretical idea.
  - Encryption in CBC mode can provide a padding oracle due to its error propagation properties.
  - Applied to TLS in a variety of attacks.
- Main mitigation: having a uniform error response, so that the attacker cannot distinguish padding errors from MAC errors.
- Padding Oracle On Downgraded Legacy Encryption (POODLE):
  - 2014: attack is published.
  - Forcing downgrade to SSL 3.0, and then running padding oracle attack.

Heartbleed Bug



- Implementation error in toolkit OpenSSL.
- Result from improper input validation based on missing bounds check in *heartbeat* messages:
  - Allowing memory leakage from the server which is likely to include session keys and long-term keys.

Heartbleed Bug

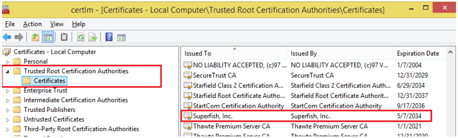


- 2014: error is discovered.
- Required updating of many server keys after the bug was fixed.
- Is it *reasonnable* that big companies use a *free* software for securing important transactions?

https://www.vox.com/2014/6/19/18076318/heartbleed

MITM Attack

- Man-In-The-Middle (MITM):
  - 2015: attack is found.
  - Attack relying on issuing a new certificate and installing a root certificate in the browser.
- Superfish is a media company whose software was bundled with some Lenovo computers:
  - Users expressed concerns about scans of SSL-encrypted web traffic pre-installed on Lenovo machines.
  - US department of Homeland Security warned users to remove the root certificate.
- May 2015: Superfish closed.



https://pkic.org/2015/02/20/lenovo-enables-man-in-the-middle-attacks-via-superfish-adware/

Other Attacks

- STARTTLS command injection attack
- Sweet32 attack
- Triple Handshake attack
- RC4 attacks
- Lucky Thirteen attack (padding oracle attack)
- Renegotiation attack

Outline

Summary of Lecture 17

Attacks on TLS

TLS 1.3

History and Overview

- 2014: first draft version.
- January 2018: Internet draft version.
- August 2018: RFC 8446 is published.
- Browser support by default:
  - Draft version since Chrome 65 and final version (for outgoing connections) since Chrome 70.
  - Draft version in Firefox 52 and above (including Quantum) and final version since Firefox 63.
  - Since Microsoft Edge version 76, and Safari 12.1 on macOS 10.14.4.

Big Removals

The following items were suppressed in TLS 1.3:

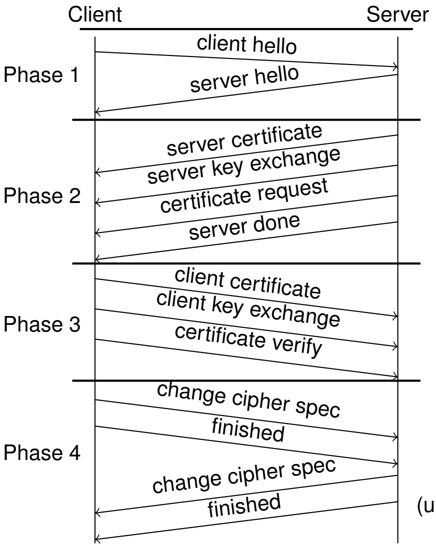
- Static RSA and Diffie-Hellman key exchange
- Renegotiation
- SSL negotiation
- DSA
- Data compression
- Non-AEAD cipher suites
- MD5 and SHA-224 hash functions
- Change Cipher Spec protocol

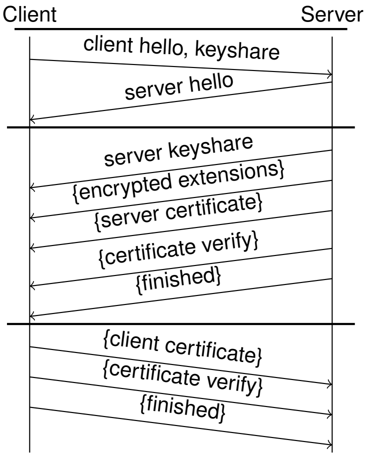
Big Supplements

The following properties/items were added in TLS 1.3:

- Only authenticated encryption with associated data (AEAD) cipher suites.
- Separating key agreement and authentication algorithms from cipher suites.
- Mandating perfect forward secrecy:
  - Using ephemeral keys during (EC) Diffie-Hellman key agreement.
- Encrypting the content type.
- Introducing 0-RTT mode (from pre-shared key).
- Introducing post-handshake client authentication.
- ChaCha20 stream cipher with Poly1305 MAC.

Handshake Comparison



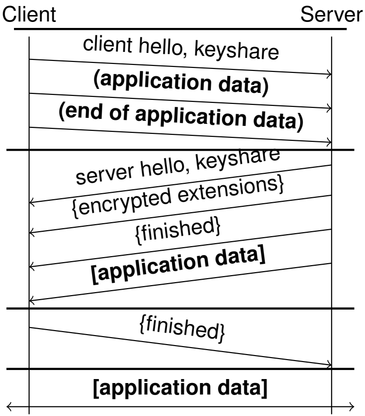


{} protected by handshake traffic keys
(up to) TLS 1.2 on the left, TLS 1.3 on the right

0-RTT Overview

- 0-RTT based on a pre-shared key (resumption master secret):
  - Obtained when server and client complete a handshake for the first time.
  - Using that key when establishing a connection again at a later time:
    - So avoiding to perform the handshake a second time.
- Is this really 0-RTT (zero round trip time)?
  - TLS 1.3 handshake is 1-RTT instead of 2-RTT.
- Limitations:
  - Resumption data require no interaction from the server.
  - An attacker can capture encrypted 0-RTT data and re-send them to the server.
  - If the server is misconfigured, then it may accept replayed requests as valid:
    - Allowing the attacker to perform unsanctioned actions.

0-RTT



() protected by early data keys
{} protected by handshake traffic keys
[] protected by further traffic keys

1-RTT and 0-RTT Comparison

- A user visits a website *not* for the first time:
  - Already visited it *recently*.
  - Resuming a previous connection (which was established using TLS).
- Full handshake in TLS1.3 using ephemeral DH key exchange (resulting in 1-RTT):
  1. $C \rightarrow S$: "client hello"
  2. $S \rightarrow C$: "server hello" + DH key share + encrypted extensions + certificate + key to verify the certificate + "finished"
  3. $C \rightarrow S$: DH key share + certificate + verification data + "finished"
  - Subsequent steps: C and S exchange encrypted application data using the session key obtained from the previous steps.

`https://ldapwiki.com/wiki/0-RTT%20Handshakes`

1-RTT and 0-RTT Comparison

- Session resumption process exists in TLS (mostly 1.2) but still 1-RTT:
  1. $C \rightarrow S$: "client hello" + DH key share + pre-shared key (obtained from a previous connection after a handshake was completed)
  2. $S \rightarrow C$: "server hello" + DH key share + pre-shared key + encrypted extensions + "finished"
     (no certificate and verification key anymore)
  3. $C \rightarrow S$: "finished"
     (no certificate and verification data anymore)

  - Subsequent steps: C and S exchange encrypted application data using the session key obtained from the previous steps.

1-RTT and 0-RTT Comparison

- With 0-RTT, the process is shortened:
  1. $C \rightarrow S$: "client hello" + DH key share + pre-shared key (obtained from a previous connection after a handshake was completed) + early data + "end of early data" (that is an alert)
  2. $S \rightarrow C$: "server hello" + DH key share + pre-shared key + encrypted extensions + application data + "finished" (no certificate and verification data anymore)
  3. $C \rightarrow S$: application data + "finished"

  - Subsequent steps: C and S exchange *more* encrypted application data using the session key obtained from the previous steps.

Example

- *C* and *S* share a pre-shared key *psk* from a previous session/connection/handshake.
- *C* uses *psk* to encrypt everything after "client hello" in 0-RTT.
- While *psk* was created from a previous handshake, it was not created specifically for that previous handshake:
  - It was rather created for future use, when *C* would resume a connection.
- *S* has also *psk* and can thus decrypt what *C* sent.
- *S* replies by using the key used for the previous handshake.
- The DH key shares will serve to create a fresh session key for the rest of the connection (encrypting application data).

Security Summary

- Different kinds of attacks:
  - Implementation errors
  - Poor choice of cryptographic primitives
  - Flaws in protocol
- Backward compatibility is a problem:
  - Downgrade attacks
- Several examples of the principle that “attacks only get better” over time.
- Complexity is a major problem:
  - TLS 1.3 removes many cipher suites and protocol options.
  - TLS 1.3 simplifies the handshake protocol.
  - TLS 1.3 adds new features (e.g. 0-RTT mode) which present new challenges.