TRANSPORT LAYER SECURITY

ADAM HUMPHRIES

2

TRANSPORT LAYER SECURITY

- What is it?
- How does it work?
- What's an example?
3

**TLS: WHAT IS IT?**

- A ubiquitous, versatile protocol that provide security benefits to a connection, including the following:
  - Confidentiality
  - Authentication
  - Integrity
- TLS builds on top of TCP – The data and metadata in a TLS connection are contained within the data section of a TCP record.
  - So TLS has reliable transport “already taken care of.”
  - This also means that IP routing info isn’t hidden by TLS.

4

**TLS: WHAT IS IT?**

- TLS is a protocol, not a specific implementation or piece of software.
- TLS was created from the Secure Socket Layer originally created by Netscape. Upon its adoption as a public standard, SSL was renamed to TLS. People still use both SSL and TLS to refer to this standard.
- TLS is defined in documents called RFCs, or requests for comments. The latest major version of TLS currently live is TLS 1.2 (RFC5246), and is a text document.
12/9/2017

TLS: WHAT IS IT?

- TLS is complicated.
- Not every TLS session provides all possible security guarantees, and many options are available for the implementer to select.
  - Some of or all of confidentiality, integrity, authentication
  - CBC, stream ciphers, or AEAD ciphers for data encryption
  - Key exchange and authentication variants involving combinations of RSA and Diffie-Hellman, with and without elliptic curve crypto (and many others)

TLS: WHAT IS IT?

- Here's a very small example of the cipher suites supported
- Many more exist. The list on the right is only for server authenticated Diffie-Hellman key exchange.
TLS: HOW DOES IT WORK?

• TLS Defines something called the “Record Protocol”
• Each record has a type – for instance, handshake, application data, alert, etc
• We’ll be covering the handshake and application data message types today

Connections start with the handshake protocol (see right)

Goals of the handshake are

• Authenticate, if necessary
• Negotiate cypher suite and connection details
• Produce a shared key
CLIENT HELLO

- First message in the handshake
- Client specifies a couple of preferences for TLS version, choice of cipher suite, etc.
- Note that a client-generated random value is also provided.

```c
struct {
    ProtocolVersion client_version;
    random random;
    SessionID session_ID;
    CipherSuite cipher_suite;
    CompressionMethod compression_method;
    select (extension_present) {
        case false:
            struct {};
        case true:
            Extension extension extremism..
    };
} clientHello;
```

SERVER HELLO

- Based on preferences suggested by client, server decides the cipher suite and compression method.
- Note that a server-generated random value is also provided.

```c
struct {
    ProtocolVersion server_version;
    random random;
    SessionID session_ID;
    CipherSuite cipher_suite;
    CompressionMethod compression_method;
    select (extension_present) {
        case false:
            struct {};
        case true:
            Extension extension extremism;
    };
} serverHello;
```
### Optional Server Messages and Server HelloDone

Optionally, the server may send a Certificate to authenticate itself. If DH key exchange has been agreed upon, the server will send its piece of the key material.

### Client Key Exchange

Regardless of the cipher suite used, the client must follow a ServerHelloDone with the ClientKeyExchange method.

After this message has been sent, both sides should have access to the “premaster key”.

```c
struct {  
    select (keyExchangeAlgorithm) {  
        case rsa:  
            encryptedPremasterSecret;  
        case dh_05s:  
            case dh_rsa:  
                case dh_dh:  
                    case dh_anon;  
                    ClientHello嫌弃Public;  
                exchange_keys;  
            ClientKeyExchange;  
        }  
}
```
### 13 ChangeCipherSpec and Finished

Both the server and client send ChangeCipherSpec messages to indicate they'll begin encrypted transmission.

Interestingly, the Finished message contains a hashed/encrypted copy of each party's communication to prevent tampering.

```c
typedef struct {
    uint8 verify_data[verify_data_length];
} finished;

finished = PRF(premaster_secret, finished_label, mask(random_messages))

finished_label
```

For Finished messages sent by the client, the string "client finished". For Finished messages sent by the server, the string "server finished".

![Message flow for a full handshake](image)

---

### 14 Key Calculation

- Both sides have the premaster secret from the handshake. So what?
- Basically, both sides create a PRF using the premaster secret and the random values from the hellos earlier
- This PRF is used to generate key material
- The PRF is not a PRF in the sense we covered in class; it's really more of a pseudorandom generator
KEY CALCULATION

P_HASH AND PRF DEFINITION

\[
P_{\text{hash}}(\text{secret}, \text{seed}) = \text{HMAC}_{\text{hash}}(\text{secret}, A(1) \oplus \text{seed}) \\
\text{where } \oplus \text{ indicates concatenation.}
\]

A(1) is defined as:
\[
A(1) = \text{seed} \\
A(2) = \text{HMAC}_{\text{hash}}(\text{secret}, A(1) \oplus \text{seed})
\]

\[
\text{PRF}(\text{secret}, \text{label}, \text{seed}) = P_{\text{hash}}(\text{secret}, \text{label} \oplus \text{seed})
\]

MASTER SECRET CALCULATION

A.1. Computing the master secret

For all key exchange methods, the same algorithm is used to convert the pre-master_secret into the master_secret. The pre_master_secret should be deleted from memory once the master_secret has been computed.

\[
\text{master_secret} = \text{PRF}(\text{pre_master_secret}, "master_secret", \\
\text{(ClientHello.random + ServerHello.random)} \oplus [0..47])
\]

The master secret is always exactly 48 bytes in length. The length of the premaster secret will vary depending on key exchange method.

KEY CALCULATION CONTINUED

- Finally, we produce the actual key material using the PRF keyed on the master secret and the server/client randomly provided values.

- Note that we covered HMAC in a previous class

- HMAC is implemented with a hash function specified in the handshake
APPLICATION DATA FLOW

- After handshake is completed, data can be sent
- Data is MAC’d and then encrypted by default TLS 1.2 standards; extensions now available for encrypt-then-MAC (RFC 7366)
- Exact layout of records will differ based on the type of cipher used, but messages with type "Application Data" and length and version are sent along with ciphertext.
- MAC typically will run on sequence number and text
- Cipher suites defined in the TLS RFC use HMAC (RFC 2104)

EXAMPLE - HTTPS

- What would the TLS portion of a common HTTPS connection look like?
- It could potentially use the cipher suite TLS_RSA_WITH_RC4_128_MD5\(^1\)
- RSA used for server authentication and key exchange, plus stream cipher RC4 with key size 128 and hash function MD5
- This is an older example of a cipher suite; don’t use this in production!
- What about client authentication?

SOURCES

• Gutmann, P., "Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7366, September 2014