Transport-Level Security

Tecnologie Internet

a.a. 2022/2023

Summary

- Web Security
- Transport Layer Security (TLS)
- OpenSSL
- HTTPS

Web Security

Virtually all businesses, most government agencies, and many individuals now have **web sites**.

The **World Wide Web** is fundamentally a client/server application running over the Internet and TCP/IP intranets.

A Web server can be exploited as a launching pad into the corporation's or agency's entire computer complex.

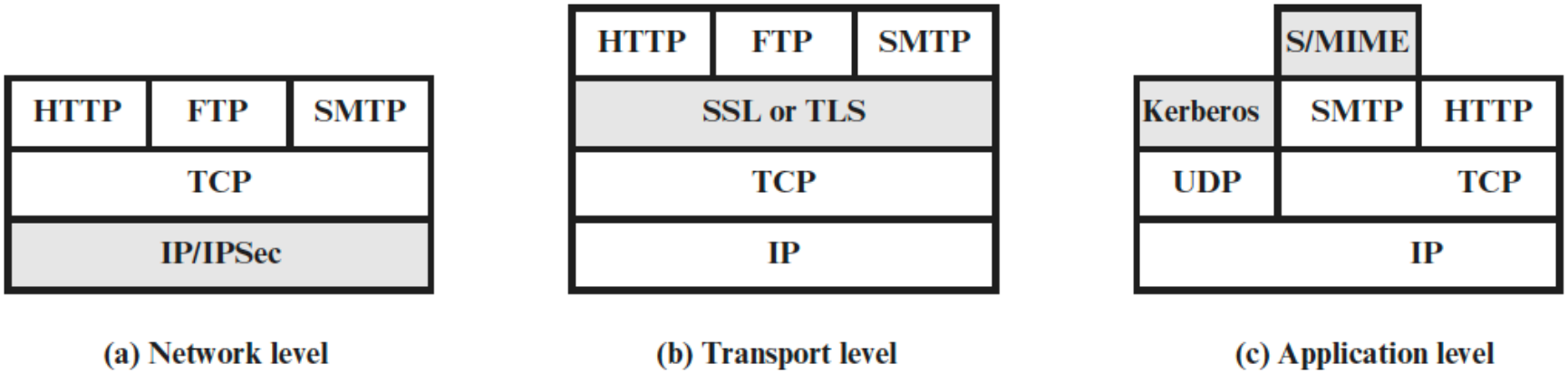
Once the Web server is subverted, an attacker may be able to gain access to data and systems not part of the Web itself but connected to the server at the local site.

Web Security Threats

What: passive vs. active threats.
Where: on the Web server, on the Web browser, or in the network.

	Threats	Consequences	Countermeasures
Integrity	• Modification of user data • Trojan horse browser • Modification of memory • Modification of message traffic in transit	• Loss of information • Compromise of machine • Vulnerability to all other threats	Cryptographic checksums
Confidentiality	• Eavesdropping on the net • Theft of info from server • Theft of data from client • Info about network configuration • Info about which client talks to server	• Loss of information • Loss of privacy	Encryption, Web proxies
Denial of Service	• Killing of user threads • Flooding machine with bogus requests • Filling up disk or memory • Isolating machine by DNS attacks	• Disruptive • Annoying • Prevent user from getting work done	Difficult to prevent
Authentication	• Impersonation of legitimate users • Data forgery	• Misrepresentation of user • Belief that false information is valid	Cryptographic techniques

Web Traffic Security Approaches



IP security (IPsec): general purpose, transparent to end users and applications.

Secure Sockets Layer (SSL) or Transport Layer Security (TLS) are general-purpose as well.

Transport Layer Security (TLS)

Defined in RFC 8446, TLS 1.3 is an Internet **standard** that evolved from a commercial protocol known as Secure Sockets Layer (SSL).

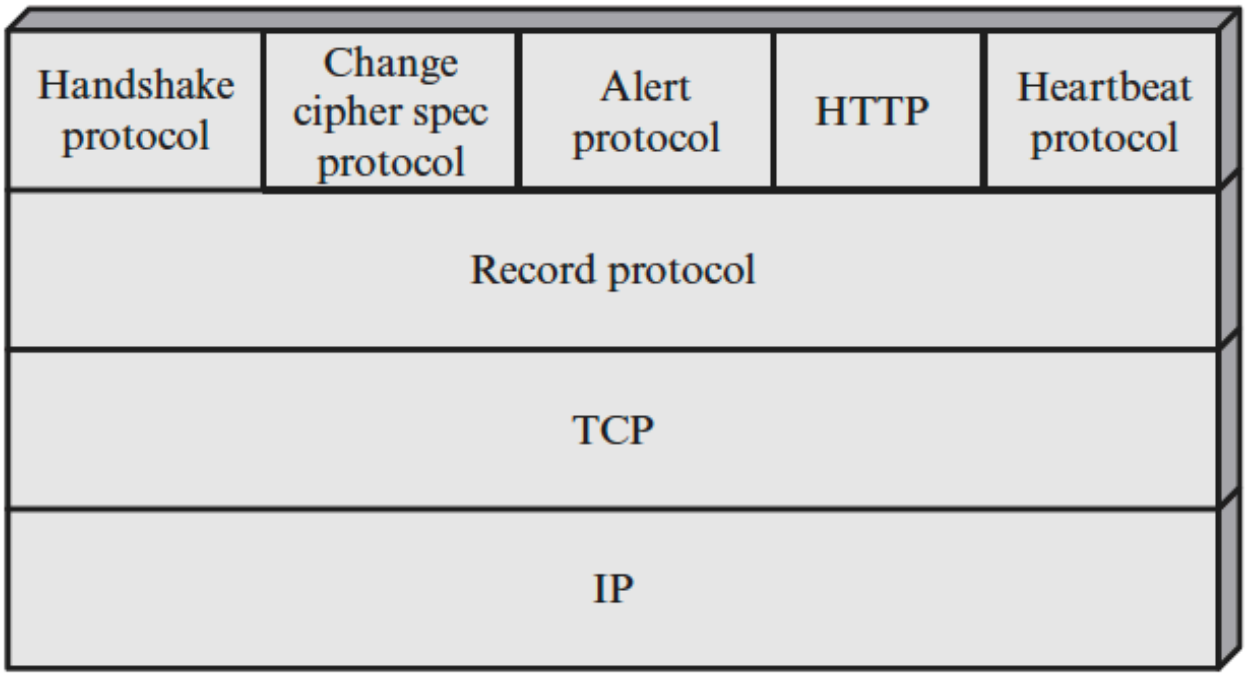
Although SSL implementations are still around, it has been deprecated by IETF and is disabled by most corporations offering TLS software.

TLS is a **general-purpose service** implemented as a set of protocols that rely on **TCP**.

Virtually all browsers come equipped with TLS and most Web servers have implemented the protocol.

TLS Architecture

The **TLS Record Protocol** provides basic security services to various higher-layer protocols, such as HTTP. The TLS specification includes also the **Handshake**, **Change Cipher Spec** and **Alert** protocols. The **Heartbeat** protocol is defined in a separate RFC.



TLS Architecture

client: The endpoint initiating the TLS connection.

connection: A transport-layer connection between two endpoints.

endpoint: Either the client or server of the connection.

handshake: An initial negotiation between client and server that establishes the parameters of their subsequent interactions within TLS.

peer: An endpoint. When discussing a particular endpoint, "peer" refers to the endpoint that is not the primary subject of discussion.

receiver: An endpoint that is receiving records.

sender: An endpoint that is transmitting records.

server: The endpoint that did not initiate the TLS connection.

TLS Architecture

Session: an association between a client and a server. Sessions are created by the Handshake Protocol. Sessions define a set of cryptographic security parameters.

To help mitigate some of extra latency and computational costs of the full TLS handshake, **TLS Session Resumption** provides a mechanism to resume or share the same negotiated secret key data between multiple connections.

Session state defined by:

- Session identifier (arbitrary byte sequence chosen by the server)
- Peer certificate (X.509 certificate of the peer)
- Cipher spec (data encryption algorithm and more)
- Master secret (48-byte secret between client and server)
- …

TLS Architecture

A TLS request to establish a connection begins with the establishment of a **TCP connection** between the TCP entity on the client side and the TCP entity on the server side.

Connection state defined by:

- Server write MAC secret
- Client write MAC secret
- Server write key (symmetric encryption key used for data encrypted by the server and decrypted by the client)
- Client write key (symmetric encryption key used for data encrypted by the client and decrypted by the server)
- Initialization vectors (IVs)
- Sequence numbers

TLS Record Protocol

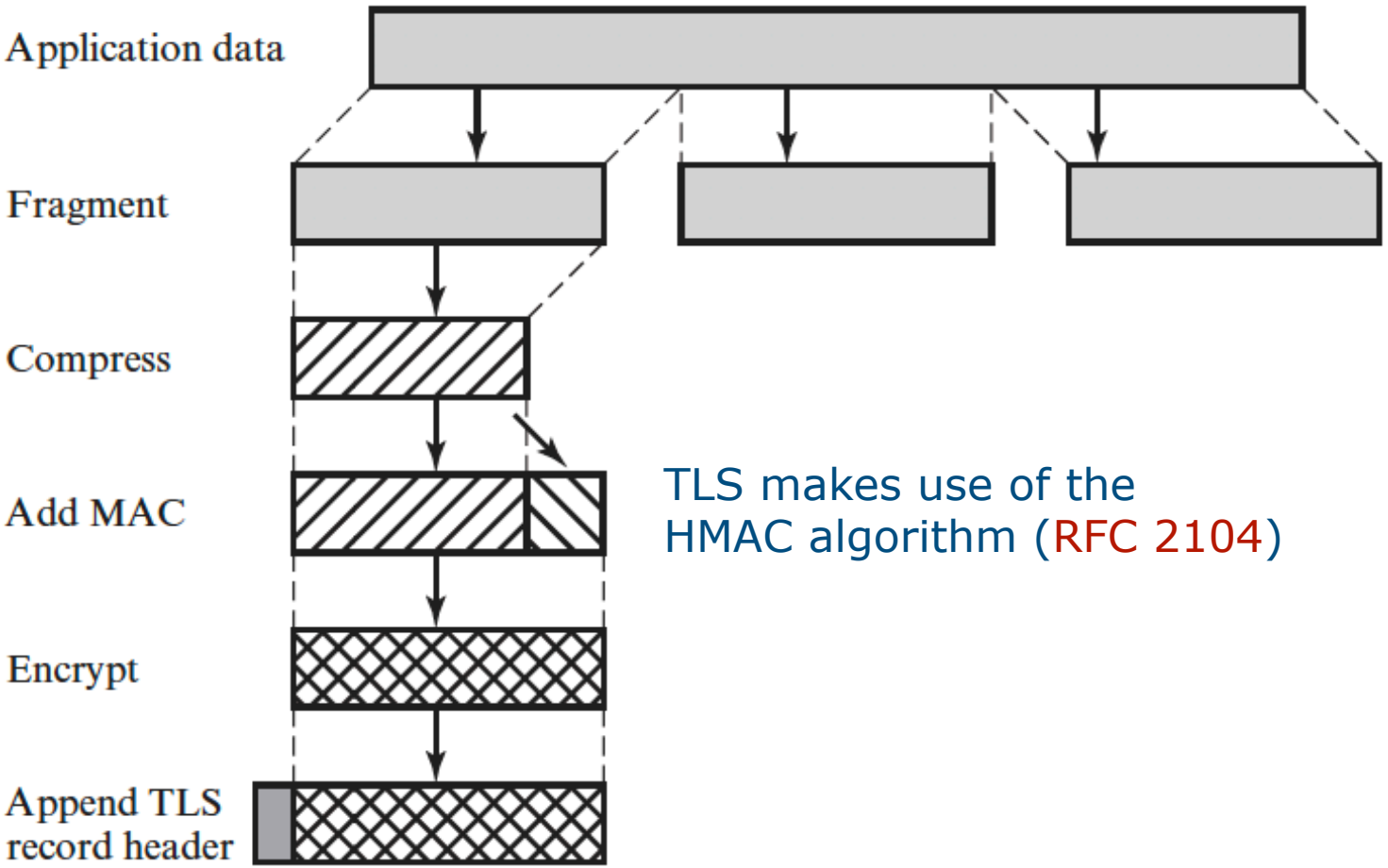
Provides two services for TLS connections:

- **Confidentiality**: The Handshake Protocol defines a shared secret key that is used for conventional encryption of TLS payloads.
- **Message Integrity**: The Handshake Protocol also defines a shared secret key that is used to form a message authentication code (MAC).

The Record Protocol takes an application message to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, adds a header, and transmits the resulting unit in a TCP segment.

Received data are decrypted, verified, decompressed, and reassembled before being delivered to higher-level users.

Record Protocol

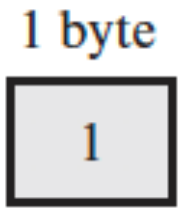


TLS makes use of the
HMAC algorithm (RFC 2104)

Change Cipher Spec Protocol

This protocol consists of a single message, which consists of a single byte with the value 1.

The CCS message **tells the peer that the sender wants to change to a new set of keys**, which are then created from information exchanged by the Handshake Protocol.



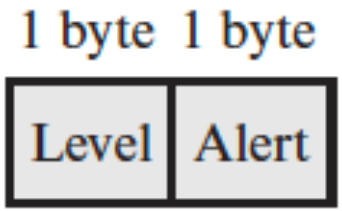
Alert Protocol

The Alert Protocol is used to convey TLS-related alerts to the peer entity.

Each message in this protocol consists of two bytes.

The first byte takes the value **warning (1)** or **fatal (2)** to convey the severity of the message. If the level is fatal, TLS immediately terminates the connection.

The second byte contains **a code that indicates the specific alert**.



Alert Protocol

Fatal alert examples:

- **unexpected_message**: An inappropriate message was received.
- **bad_record_mac**: An incorrect MAC was received.
- **decompression_failure**: The decompression function received improper input (e.g., unable to decompress or decompress to greater than maximum allowable length).
- **handshake_failure**: Sender was unable to negotiate an acceptable set of security parameters given the options available.
- **illegal_parameter**: A field in a handshake message was out of range or inconsistent with other fields.

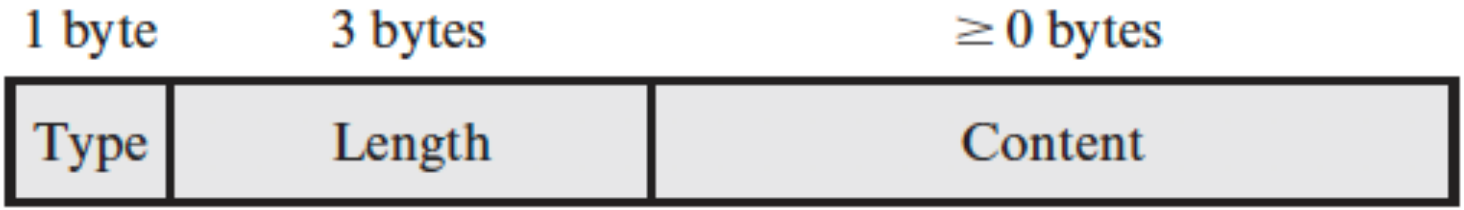
Handshake Protocol

The most complex part of TLS is the Handshake Protocol.

This protocol allows the server and client to authenticate each other and to negotiate an encryption and MAC algorithm and cryptographic keys to be used to protect data sent in a TLS record.

The Handshake Protocol is used before any application data is transmitted.

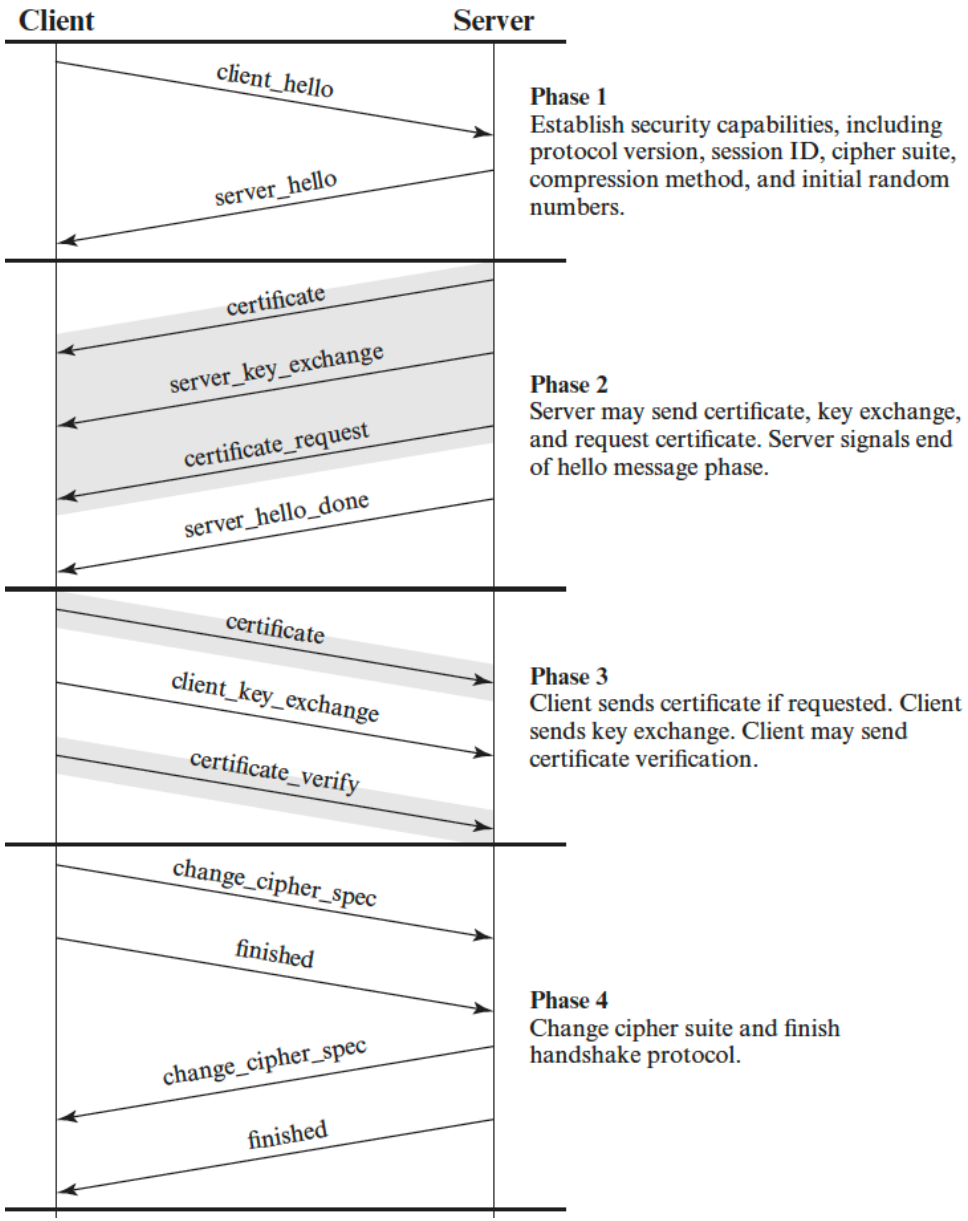
Handshake messages have the following structure:



Handshake Protocol

Handshake phases:

1. **Establish Security Capabilities**
2. **Server authentication and key exchange**
3. **Client authentication and key exchange**
4. **Finish**



Handshake Protocol

Message types:

Message Type	Parameters
hello_request	null
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

Handshake Protocol

Supported **key exchange methods**: (EC)DHE (Diffie-Hellman over either finite fields or elliptic curves), PSK-only (pre-shared keys), or PSK with (EC)DHE

Supported **cipher algorithms**: Authenticated Encryption with Associated Data (AEAD) RFC 5116

Supported **MAC algorithms**: HMAC-based

Heartbeat Protocol

A heartbeat protocol is typically used to monitor the availability of a protocol entity.

The Heartbeat protocol runs on top of the TLS Record Protocol and consists of two message types: **heartbeat_request** and **heartbeat_response**.

A heartbeat_request message can be sent at any time. Whenever a request message is received, it should be answered promptly with a corresponding heartbeat_response message.

The **payload** is a random content between 16 B and 64 KB in length. The corresponding heartbeat_response message must include an exact copy of the received payload.

RFC 6520

OpenSSL

OpenSSL is a robust, commercial-grade, and full-featured toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It is also a general-purpose cryptography library.

https://www.openssl.org/

Command line tool:

openssl *command* [*options* ...] [*parameters* ...]

openssl list -standard-commands | **-digest-commands** | **-cipher-commands** | **-cipher-algorithms** | **-digest-algorithms** | **-mac-algorithms** | **-public-key-algorithms**

OpenSSL

How to create a self-signed server certificate:

openssl req -new -newkey rsa:2048 -days 365 -nodes -x509 -keyout server.key -out server.crt

Option -nodes means “no DES” (the private key will not be encrypted).

File server.key contains the RSA key pair, it must be kept secret!

File server.crt is the self-signed certificate.

Normally we would use a server certificate from a Certificate Authority such as Let’s Encrypt (https://letsencrypt.org/).

Let's Encrypt is a free, automated, and open certificate authority brought to you by the nonprofit Internet Security Research Group (ISRG).

HTTPS

HTTPS (**HTTP over SSL/TLS**) refers to the combination of HTTP and SSL/TLS to implement secure communication between a Web client (in particular, a Web browser) and a Web server.

The HTTPS capability is built into all modern Web browsers. Its use depends on the Web server supporting HTTPS communication.

The principal difference seen by a user of a Web browser is that URL (uniform resource locator) addresses begin with **https://** rather than http://.

A normal HTTP connection uses port 80. If HTTPS is specified, **port 443** is used.

RFC 2818

HTTPS

When HTTPS is used, the following elements of the communication are encrypted:

- URL of the requested document
- Contents of the document
- Contents of browser forms (filled in by browser user)
- Cookies sent from browser to server and from server to browser
- Contents of HTTP header

HTTPS

For HTTPS, the agent acting as the HTTP client also acts as the TLS client.

HTTP connection request
—> TLS connection request
—> TCP connection request

Then, the TLS client sends the TLS ClientHello to begin the TLS handshake. When the TLS handshake has finished, the client may then initiate the first HTTP request.

All HTTP data is to be sent as TLS application data.

The closure of an HTTPS connection requires that TLS close the connection with the peer TLS entity on the remote side, which will involve closing the underlying TCP connection.

References

William Stallings, ***Cryptography and Network Security - Principles and Practice***, 7th edition, Pearson 2017