SSL/TLS

EJ Jung
Early Version of SSL (Simplified)

Bob’s reasoning: I must be talking to Alice because...

- Whoever signed $N_B$ knows Alice’s private key... Only Alice knows her private key... Alice must have signed $N_B$... $N_B$ is fresh and random and I sent it encrypted under $K_{AB}$... Alice could have learned $N_B$ only if she knows $K_{AB}$... She must be the person who sent me $K_{AB}$ in the first message...
Breaking Early SSL

Charlie uses his legitimate conversation with Alice to impersonate Alice to Bob

- Information signed by Alice is not sufficiently explicit
What is SSL / TLS?

- Transport Layer Security protocol, version 1.0
  - De facto standard for Internet security
  - “The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications”
  - In practice, used to protect information transmitted between browsers and Web servers
- Based on Secure Sockets Layers protocol, ver 3.0
  - Same protocol design, different algorithms
- Deployed in nearly every Web browser
SSL / TLS in the Real World
Application-Level Protection

Protects against application-level threats (e.g., server impersonation), **NOT** against IP-level threats (spoofing, SYN flood, DDoS by data flood)

application, presentation, session, transport, network, data link, physical

email, Web, NFS
History of the Protocol

- **SSL 1.0**
  - Internal Netscape design, early 1994?
  - Lost in the mists of time

- **SSL 2.0**
  - Published by Netscape, November 1994
  - Several weaknesses

- **SSL 3.0**
  - Designed by Netscape and Paul Kocher, November 1996

- **TLS 1.0**
  - Internet standard based on SSL 3.0, January 1999
  - **Not** interoperable with SSL 3.0
    - TLS uses HMAC instead of MAC; can run on any port
Network protocols are usually disseminated in the form of an RFC

TLS version 1.0 is described in RFC 2246

Intended to be a self-contained definition of the protocol

- Describes the protocol in sufficient detail for readers who will be implementing it and those who will be doing protocol analysis
- Mixture of informal prose and pseudo-code
Evolution of the SSL/TLS RFC
TLS Basics

- TLS consists of two protocols
  - Familiar pattern for key exchange protocols
- Handshake protocol
  - Use public-key cryptography to establish a shared secret key between the client and the server
- Record protocol
  - Use the secret key established in the handshake protocol to protect communication between the client and the server
- We will focus on the handshake protocol
TLS Handshake Protocol

- Two parties: client and server
- Negotiate version of the protocol and the set of cryptographic algorithms to be used
  - Interoperability between different implementations of the protocol
- Authenticate client and server (optional)
  - Use digital certificates to learn each other’s public keys and verify each other’s identity
- Use public keys to establish a shared secret
Handshake Protocol Structure

ClientHello

ServerHello,
[Certificate],
[ServerKeyExchange],
[CertificateRequest],
ServerHelloDone

[Certificate],
ClientKeyExchange,
[CertificateVerify]

Finished

switch to negotiated cipher

Record of all sent and received handshake messages

switch to negotiated cipher

Finished
ClientHello

Client announces (in plaintext):
Protocol version he is running
Cryptographic algorithms he supports
ClientHello (RFC)

```c
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites;
    CompressionMethod compression_methods;
} ClientHello
```

- **ProtocolVersion**: Highest version of the protocol supported by the client.
- **Random**: Session id (if the client wants to resume an old session).
- **CipherSuite**: Set of cryptographic algorithms supported by the client (e.g., RSA or Diffie-Hellman).
C, \text{Version}_c, \text{suite}_c, N_c

Server responds (in plaintext) with:
- Highest protocol version supported by both client and server
- Strongest cryptographic suite selected from those offered by the client
Server sends his public-key certificate containing either his RSA, or his Diffie-Hellman public key (depending on chosen crypto suite).
ClientKeyExchange

C, Version<sub>c</sub>, suite<sub>c</sub>, N<sub>c</sub>

Version<sub>s</sub>, suite<sub>s</sub>, N<sub>s</sub>,
\( \text{sig}_{ca}(S,K_s) \),
“ServerHelloDone”

Client generates some secret key material and sends it to the server encrypted with the server’s public key (if using RSA)
struct {
    select (KeyExchangeAlgorithm) {
        case rsa: EncryptedPreMasterSecret;
        case diffie_hellman: ClientDiffieHellmanPublic;
    } exchange_keys
} ClientKeyExchange

struct {
    ProtocolVersion client_version;
    opaque random[46];
} PreMasterSecret

Random bits from which symmetric keys will be derived (by hashing them with nonces)
“Core” SSL 3.0 Handshake

C, Version\textsubscript{c}=3.0, suite\textsubscript{c}, N\textsubscript{c}

Version\textsubscript{s}=3.0, suite\textsubscript{s}, N\textsubscript{s}, sig\textsubscript{ca}(S, K\textsubscript{s}), “ServerHelloDone”

\{Secret\textsubscript{c}\}\textsubscript{Ks}

If the protocol is correct, C and S share some secret key material (secret\textsubscript{c}) at this point

switch to key derived from secret\textsubscript{c}, N\textsubscript{c}, N\textsubscript{s}  
switch to key derived from secret\textsubscript{c}, N\textsubscript{c}, N\textsubscript{s}
Version Rollback Attack

C, Version\(_c\) = 2.0, suite\(_c\), N\(_c\)

Server is fooled into thinking he is communicating with a client who supports only SSL 2.0

Version\(_s\) = 2.0, suite\(_s\), N\(_s\), sig\(_{ca}(S,K_s)\), “ServerHelloDone”

\{Secret\(_c\)\}\(_K_s\)

C and S end up communicating using SSL 2.0 (weaker earlier version of the protocol that does not include “Finished” messages)
SSL 2.0 Weaknesses (Fixed in 3.0)

- Cipher suite preferences are not authenticated
  - “Cipher suite rollback” attack is possible
- Weak MAC construction
- SSL 2.0 uses padding when computing MAC in block cipher modes, but padding length field is not authenticated
  - Attacker can delete bytes from the end of messages
- MAC hash uses only 40 bits in export mode
- No support for certificate chains or non-RSA algorithms, no handshake while session is open
“Chosen-Protocol” Attacks

- Why do people release new versions of security protocols? Because the old version got broken!
- New version must be *backward-compatible*
  - Not everybody upgrades right away
- Attacker can fool someone into using the old, broken version and exploit known vulnerability
  - Similar: fool victim into using weak crypto algorithms
- Defense is hard: must authenticate version early
- Many protocols had “version rollback” attacks
  - SSL, SSH, GSM (cell phones)
Version Check in SSL 3.0

If the protocol is correct, C and S share some secret key material secret_c at this point

$\{\text{Version}_c, \text{Secret}_c\}_{K_S}$

"Embed" version number into secret

Check that received version is equal to the version in ClientHello

$\text{Version}_s=3.0, \text{suite}_s, N_s, \text{sig}_{\text{ca}}(S, K_s)$

“ServerHelloDone”

$\text{Version}_c=3.0, \text{suite}_c, N_c$

switch to key derived from $\text{secret}_c, N_c, N_s$

switch to key derived from $\text{secret}_c, N_c, N_s$
SSL/TLS Record Protection

Application Data

Fragment

Compress

Add MAC

Encrypt

Append SSL Record Header

Use symmetric keys established in handshake protocol