SSL/TLS Decryption
uncovering secrets

Wednesday April 11th, 2018

Peter Wu
Wireshark Core Developer
peter@lekensteyn.nl
About me

- Wireshark contributor since 2013, core developer since 2015.
- Areas of interest: TLS, Lua, security, ...
- Developed a VoIP product based on WebRTC.
- InfoSec Master’s student @ TU/e (NL).
- Cloudflare crypto intern in 2017.
Things that people care about: pictures, videos, documents, email conversations, passwords, ... 

Application Data: cookies, API keys, Request URI, User Agent, form data, response body, ... 

How to keep these safe when sending it over the internet or over your local Wi-Fi network?
Transport Layer Security (TLS)

- Provides secure communication channel between two endpoints (client and server).
- Network protocol with two components:
  - Record Protocol: carries messages and protects application data fragments.
Secure Sockets Layer (SSL) versus Transport Layer Security (TLS)

- SSLv3: old (RFC 6101, 1996) and deprecated (RFC 7568, 2015). Do not use it!
- TLS 1.0 (RFC 2246, 1999), 1.1 (RFC 4346, 2006), 1.2 (RFC 5246, 2008).
- Changes:
  - New versions are generally fixing weaknesses due to new attacks.
  - TLS 1.0 (RFC 3546, 2003) and up allow for extensions, like Server Name Indication (SNI) to support virtual hosts.
  - TLS 1.2: new authenticated encryption with additional data (AEAD) mode.
- “SSL” term still stuck, e.g. “SSL certificate”, “SSL library” and field names in Wireshark (e.g. ssl.record.content_type).
- Mail protocols: TLS often refers to STARTTLS while SSL directly starts with the handshake.
Secure communication channel

- Symmetric-key algorithms: encrypt/decrypt bulk (application) data using a single (secret) symmetric key. Examples: AES, 3DES, RC4.
- How to create such a secret? For example, AES-256 needs a 256-bit key.
- Public-key cryptography: a (secret) private key and a related public key.
  - Mathematically hard to compute private key from public key.
  - Encrypt data with public key, decrypt with private key.
  - Limitation: maximum data size for RSA is equal to modulus size, 2048-4096 bits.
  - Idea: generate a random premaster secret and encrypt it with the RSA public key.
- Where to retrieve this RSA public key from?
Certificates and trust

- Public key is embedded in an X.509 certificate.
- How can this certificate be trusted?
- A Certificate Authority (issuer) signs the certificate with its private key.
- Public-key cryptography: use a private (secret) key and a public key with small data.
  - Compress data using a hash function. Examples: SHA256, SHA1, MD5.
  - Sign hash with private key, verify with public key. Examples: RSA, ECDSA.
- Root CAs are self-signed and installed by the OS vendor or local admin (Group Policy, etc.).
TLS handshake with RSA key exchange method

- Client Hello advertises supported parameters, Server Hello decides.
- Server picks RSA key exchange: `TLS_RSA_WITH_AES_128_CBC_SHA`.

+ Certificate (with RSA public key)
+ `ServerHelloDone`
TLS handshake with RSA key exchange method - ClientKeyExchange

- Client received Server Hello and now knows protocol version and cipher suite.
- Client generates a new random 48-byte **premaster secret**, encrypts it using the **public key** from the Certificate and sends the encrypted result to the server in a **ClientKeyExchange** message.
- Using the private RSA key, server (or anyone else!) decrypts the premaster secret.

<table>
<thead>
<tr>
<th>Handshake Protocol: Client Key Exchange</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handshake Type: Client Key Exchange (16)</td>
</tr>
<tr>
<td>Length: 130</td>
</tr>
<tr>
<td>RSA Encrypted PreMaster Secret</td>
</tr>
<tr>
<td>- Encrypted PreMaster length: 128</td>
</tr>
<tr>
<td>- Encrypted PreMaster: 6714b8c800549d2857d2484f7d184a6d7e2d186b7e4322b0...</td>
</tr>
</tbody>
</table>
Both sides calculate the 48-byte **master secret** based on the Client Random, Server Random and the premaster secret.

Both sides derive symmetric keys from this master secret, send the *ChangeCipherSpec* message to start record protection.

Finally they both finish the Handshake protocol by sending a *Finished* Handshake message over the encrypted record layer.

Now the actual encrypted *Application Data* can be sent and received.
Handshake overview

Client

ClientHello ---> ServerHello
Certificate*
ServerKeyExchange*
<-------- ServerHelloDone

ClientKeyExchange

[ChangeCipherSpec]

Finished ---> [ChangeCipherSpec]
<-------- Finished

Application Data <--------> Application Data

Simplified TLS handshake (adapted from RFC 5246 (TLS 1.2))

#sf18asia ● NEC, Nanyang Technological University, Singapore ● April 9-11
Server administrators can check application logs.

Web browsers provide developer tools.

What if the information is not logged?

What if you want to know what this third-party Android app is doing?

What if the application under investigation is poorly documented?

What if you want to debug your new HTTP/2 feature?

Solution: packet capture plus SSL/TLS secrets!
Configure Wireshark with a RSA private key file:\n
- IP address is unused and ignored. Port + Protocol can be empty. These three fields will be removed in future.
- Specify (passwordless) PEM-encoded key file or PKCS#12 key file + password.

\[-----BEGIN PRIVATE KEY-----\nMIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQDSejtB5QbSkaLM...NsRXfSXTvphoograxijG/RfKcTmi0c0nuckopyKDwBsdY3HnPrTB/h7FuKmew0bWgn4GfGdwuvP9C+FoaG8+s=\n\[-----END PRIVATE KEY-----\n
\[See https://wiki.wireshark.org/SSL#Preference_Settings\]
Limitations of RSA private key

- Clients usually do not have access to the RSA key, only server operators can use it.
- In case of mutual authentication (client certificates), the private key is only used for *signing*. The client private RSA key cannot decrypt.
- Encrypted premaster secret is not sent with resumed sessions.

Diagram:

```
Client                     Server

ClientHello                ServerHello
<----- [ChangeCipherSpec]
[ChangeCipherSpec]          [ChangeCipherSpec]
Finished                   Finished

Application Data           Application Data
```

Message flow for an abbreviated handshake (RFC 5246, Figure 2)
Ephemeral (Elliptic Curve) Diffie-Hellman (ECDHE)

- Decryption using RSA private key not possible with cipher suites like TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 and TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256.
- Although it has RSA in its name, it is not used for encryption, but signing.
- Instead it uses Diffie-Hellman to establish a shared secret (the **premaster secret**) based on ephemeral secrets (different secrets for every session).
- Server chooses a group/curve, generates private value and its related public value and sends it to the client. Client uses same group/curve and also generates a pair.
- Computationally hard to find the private value given the public one.
TLS secrets summary

▶ Any of these can be used for decryption with passive captures:
  ▶ premaster secret: RSA-encrypted or output from DH key exchange.
  ▶ Master secret: derived from premaster secret and handshake messages. Also used for session resumption.
  ▶ Symmetric encryption key for record encryption.
  ▶ RSA private key file (for RSA key exchange, covered before).

▶ So how to use master secrets?
SSL key log file

- Text file with master secrets\(^2\).
- Works for any cipher, including RSA and DHE.
- Clients can use this too!
- Set environment variable SSLKEYLOGFILE before starting Firefox or Chrome. The variable is only read during startup, so restart if necessary.
- Format: `CLIENT_RANDOM <Client Hello Random> <master secret>`.

# SSL/TLS secrets log file, generated by NSS

```
CLIENT_RANDOM 5f4dad779789bc5142cacf54f5dafba0a06235640796f40048ce4d0d1df63ad8 a4d69a3fa4222d6b6f2492e66dca2b1fc4e2bc143df849ad45ef9f
CLIENT_RANDOM c2407d5ba931798e3a35f775725fb3e5aeefcb5804bb50271fe3bd5fb19c90061 e419759e7b44f766df6defe6b656eda3d430754044773b6fc0a19e1b
CLIENT_RANDOM abc6cf83ea1dcb135b21fd94b0c1206da37dca0fcd96ef8989d05c1cc3ab 5b4d525dfe3168132d388810336333c2aba99346c25ae8163f2191f
CLIENT_RANDOM dffe2c857a7d6f3c3e34ba52ea710f0f1649e58afa02f9824d983ea74f07900e fdb5d849482f876f200ce680b9d698743e3a6a54d203fc57cc5888
CLIENT_RANDOM fbf40ada961093cd917fba97dfbe7e4b0bbf57a0cf90626dede417d3d12b3755 6b4e31366be9316c42f47ddd3ceef9743825bd3c3bb25ec9ac73c9
CLIENT_RANDOM 2b818474624df4bb5979ad9a623690b0f8f392deb94f6b4b00d7dc78b711638b dfeb3f9f46f99eeaa02489e3b92c8d7770c12928becaf0ac1e34edf
CLIENT_RANDOM 7e4340c76c720d39c98e761697be0f32e1c79c6c04ade05a3f29325ac9cae612 1de402b85560048ae278b78febe83ee1640785b969c328d94a785a
```

\(^2\)File format at https://developer.mozilla.org/NSS_Key_Log_Format

#sf18asia ● NEC, Nanyang Technological University, Singapore ● April 9-11
Configure file in Wireshark preferences: Edit → Preferences; Protocols → SSL; (Pre-)Master Secret log filename.

Key log file is also read during a live capture. And if the file is removed and a new file is written, the new key log file is automatically read.

Caveat: key log is read while processing ChangeCipherSpec. If key is written too late, trigger a redissection (e.g. change a preference or (Un)ignore a packet).
Any application built using NSS and GnuTLS enable key logging via the `SSLKEYLOGFILE` environment variable.

Applications using OpenSSL 1.1.1 or BoringSSL d28f59c27bac (2015-11-19) can be configured to dump keys:

```c
void SSL_CTX_set_keylog_callback(SSL_CTX *ctx,
    void (*cb)(const SSL *ssl, const char *line));
```

ARM Mbed TLS using a debug callback\(^3\).

cURL supports many TLS backends, including NSS, GnuTLS and OpenSSL. Key logging with OpenSSL/BoringSSL is possible since curl 7.56.0\(^4\).

Java applications can use jSSLKeyLog\(^5\).

---

\(^3\)https://github.com/Lekensteyn/mbedtls/commit/68aea15
\(^4\)Requires a build time option, see https://curl.haxx.se/bug/?i=1866
\(^5\)http://jsslkeylog.sourceforge.net
Key log with OpenSSL 1.1.0 and before

- Why: many applications (including Python) use OpenSSL.
- Problem: older OpenSSL versions have no key log callback.
- Solution: intercept library calls using a debugger or an interposing library (LD_PRELOAD) and dump keys.
- Example with OpenSSL 1.1.0f using an intercepting library:

```bash
$ export SSLKEYLOGFILE=some.keys LD_PRELOAD=./libsslkeylog.so
$ curl https://example.com
...
$ cat some.keys
CLIENT_RANDOM 12E0F5085A89004291A679ABE8EE1508193878AB9E909745CA032212FCA24B89 148AF5875F83E083115AA1AB16F97C4F097B5AFBE4B948C5077FD05A0F8FE1E2A68502923B259A20FAF7A9FD915A38AA
```

---

6https://security.stackexchange.com/q/80158/2630
7https://git.lekensteyn.nl/peter/wireshark-notes/tree/src
Windows native TLS library is Secure Channel (SChannel). Feature request for Microsoft Edge browser is pending\(^8\).

Extracting secrets from SChannel is not impossible (but neither easy) though\(^9\).

Apple macOS applications use SecureTransport, also not supported.

---

\(^8\)https://wpdev.uservoice.com/forums/257854-microsoft-edge-developer/suggestions/16310230-ssl-key-logging-aka-sslkeylogfile

Alternative ways to get the secret

▶ Force RSA key exchange (disable forward-secret cipher suites).
▶ Setup a fake CA and force traffic through a proxy like mitmproxy\(^{10}\), OWASP Zap, Fiddler or Burp Suite.
▶ All of these methods can be detected by the client. Certificate pinning can also defeat the custom CA method.
▶ The proxy interception method may also weaken security\(^{11}\).
▶ If you are really serious about a passive, nearly undetectable attack from a hypervisor, see the TeLeScope experiment\(^{12}\).

---

Display the contents of the decrypted application data.

Right-click in the packet list or details view, **Follow → SSL Stream**.

Great for text-based protocols like SMTP. For binary data, try the **Hex Dump** option.

Click on data to jump to related packet (in packet list). Note that a display filter can hide packets, clear the filter to avoid that.
After decryption is enabled, HTTP payloads within TLS (HTTPS) can be exported.

File → Export Objects → HTTP...

Click on an item to select it in the packet list.

Note: does not cover HTTP/2 nor QUIC (yet?) as of Wireshark 2.6.
Feature: Export SSL Session Keys

▶ Suppose you have a capture which is decrypted using a RSA private key file. How to allow others to decrypt data without handing over your RSA private key file?

▶ File → Export SSL Session Keys...

▶ Generates a key log file which can be used instead of the private RSA key file.

▶ Note: currently contains all keys. Remove lines which are not needed (match by the second field, the Random field from Client Hello).
Feature: Display Filters

- Display filters can be used for filtering, columns and coloring rules.
- Discover by selecting a field in packet list, look in status bar.
- Recognize TCP/TLS stream in packet list: Right-click *TCP Stream Index* (tcp.stream) field in packet details, *Apply as Column*.
- Right-click field in packet details, *Apply/Prepare as Filter*.
- SNI in Client Hello: ssl.handshake.extensions.server_name
- Change in Wireshark 2.4: ssl.handshake.random selects full Client or Server Random instead of the just the Random Bytes field. Reason: real time is often no longer included, full bytes field is useful for matching with key log file.

```plaintext
Handshake Protocol: Client Hello
  Handshake Type: Client Hello (1)
  Length: 196
  Version: TLS 1.2 (0x0303)
  Random: 10f5a9d24a0b80166789eeeebf261bd89aad614aa93aa5aa8...
  GMT Unix Time: Oct 25, 1983 00:47:53.000000000 GMT
  Random Bytes: a1bfc1d6679eeebf261bd89aad614aa93aa5aa8...
  Session ID Length: 0
  Cipher Suites Length: 30
  Cipher Suites (15 suites)
  - Compression Methods Length: 1
```

*Random values used for deriving keys (ssl.handshake.random), 32 bytes*
Feature: Decode As

- Force dissector for custom ports. Decode as SSL (TCP) or DTLS (UDP).
- Select application data protocol within SSL/TLS layer (since Wireshark 2.4).
- Example: HTTPS on non-standard TCP server port 4433.
  - Right-click TCP layer, Decode As. Change current protocol for TCP Port to SSL.
  - Press OK to apply just for now or Save to persist this port-to-protocol mapping.
  - Right-click SSL layer, Decode As. Change current protocol for SSL Port to HTTP.
- For STARTTLS protocols, select SMTP/IMAP/... instead of SSL for TCP Port.
- Tip: there are many protocols, just select the field, then use arrow keys or type the protocol name (typing H gives HTTP).
Feature: Tshark

- Tshark: command-line tool, useful to extract information as text, especially when the query is repeated multiple times.
- Find all cipher suites as selected by the server: 
  `tshark -r some.pcap -Tfields -e ssl.handshake.ciphersuite -Y ssl.handshake.type==2`
- List all protocol fields: `tshark -G fields`
- Configure keylogfile:
  `tshark -ossl.keylog_file:firefox.keys -r firefox.pcapng.gz`
- Configure RSA keyfile (fields correspond to the RSA keys dialog):
  `tshark -ouat:ssl_keys:’","","","keys/rsasnakeoil2.key","’`
- Decode DNS-over-TLS\(^\text{13}\) on non-standard port:
  `tshark -d tcp.port==53053,ssl -d ssl.port==53053,dns`

Future: TLS 1.3

- Replaces all previous cipher suites with new one. Dropped all old cipher suites (no more CBC, RC4, NULL, export ciphers).
- RSA key exchange is gone, all ciphers are forward secret.
- Encrypted early (0-RTT) data.
- Encrypted server extensions (like ALPN).
- Encrypted server certificate.
- Multiple derived secrets for resumption, handshake encryption, application data encryption. (Safer resumption!)
- Decryption and dissection is supported by Wireshark (drafts 18-23 as of Wireshark 2.4.5, drafts 18-26 as of Wireshark 2.6).
Known issues

- Out-of-Order TCP segments break dissection and decryption (*Ignored Unknown Record*).  
  https://bugs.wireshark.org/bugzilla/show_bug.cgi?id=9461

- Large certificates result in handshake fragmentation. Not displayed because reassembly for handshake messages is not implemented yet.  
  https://bugs.wireshark.org/bugzilla/show_bug.cgi?id=3303
RSA private keys cannot be used for decryption in all cases.

The key log method (SSLKEYLOGFILE) can also be used by clients and works with all cipher suites.

TLS 1.3 debugging is even more difficult without decryption.

Use latest Wireshark version, especially if you are doing any TLS 1.3 work.