TLS 1.3: changes and impact on detection

Pierre Chifflier
pierre.chifflier@ssi.gouv.fr

Agence Nationale de la Sécurité des Systèmes d’Information

2018
Introduction
Who

- Pierre Chifflier
- Head of the Detection Research lab at ANSSI
- Security, compilers and languages
- Rust enthusiast *(parse all the things!)*
- @pollux7

Note

- Thanks to the OISF + Suricata teams
  - also for accepting strange languages pull requests
This talk will not be about Rust

- TLS 1.3: Background & History
- What’s New/Changed/Removed
- Impacts on Detection / New detection rules proposals
TLS 1.3: Changes
Transport Layer Security

“Secure Communications”

- Confidentiality
  - Encryption
  - Threat: wiretapping
- Integrity
  - Detect message tampering
  - Threat: MITM
- Authentication
  - Make sure you talk with the intended server
  - Threat: impersonation
TLS 1.3: History

- 1993: SSL v1 (never publicly released)
- 1994: SSL v2
- 1995: SSL v3
- 1999: TLS 1.0 (RFC2246)
- 2008: TLS 1.2 (RFC5246)
- 2011: SSL 2.0 prohibited (RFC6176)
- 2015: SSL 3.0 prohibited (RFC7568)
- 2015: Let’s Encrypt
- 2018 (August): TLS 1.3 ([RFC8446])

See [SSL/TLS and PKI History]
### Key Goals of TLS 1.3

<table>
<thead>
<tr>
<th>Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clean up: remove unsafe or unused features</td>
</tr>
<tr>
<td>Security: better algorithms (and less choices)</td>
</tr>
<tr>
<td>Privacy: encrypt more elements</td>
</tr>
<tr>
<td>Performance: reduce handshake (0-RTT and 1-RTT)</td>
</tr>
<tr>
<td>Backward compatible</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>More than 50% of internet traffic is encrypted!</td>
</tr>
<tr>
<td>Lots of repeated sessions to same servers</td>
</tr>
</tbody>
</table>
Clean up: removing unsafe/unused features

Is a small Diffie-Hellman group a Diffie-Hellboy?

<table>
<thead>
<tr>
<th>Deprecated/removed elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key exchange: RSA (Bleichenbacher, DROWN, weak Diffie-Hellman groups)</td>
</tr>
<tr>
<td>Key exchange: custom (EC)DHE groups (CVE-2016-0701)</td>
</tr>
<tr>
<td>Encryption ciphers: Export ciphers, DES, 3DES (Sweet32), RC4 (Bar-mitzvah, NOMORE), Camellia</td>
</tr>
<tr>
<td>Hash algorithms: MD5, SHA-1 (SLOTH, SHAttered)</td>
</tr>
<tr>
<td>Cipher modes (CBC) (Vaudenay, BEAST, Lucky13, POODLE)</td>
</tr>
<tr>
<td>Compression (CRIME)</td>
</tr>
<tr>
<td>Session renegotiation (Marsh Ray, Renegotiation DoS, Triple Handshake)</td>
</tr>
</tbody>
</table>
The handshake has three goals:

- Agree a cipher suite
- Agree a master secret
- Establish trust between client & server

Optimizations for most common use cases:

- Choosing fast & secure parameters
- Same cipher suites used across websites repeatedly
- Client connect to the same sites repeatedly
Changes: Handshake

TLS 1.2

TLS 1.3

Source: cloudflare
Parsing ServerHello:

```rust
match parse_tls_record(&bytes) {
    Err(e) => { err!("parsing failed: {:?}", e); },
    Ok((rem, msg)) => {
        match msg {
            Handshake(ServerHelloV13(ref sh)) => {
                log!("ciphersuite: {:x}", sh.cipher);
                // ...
            }
        }
    }
}
```
Changes: Cipher Suites

cipher? sweet!

Cipher Suites

- Only five ciphersuites (all AEAD)
  - TLS_AES_128_GCM_SHA256 (mandatory)
  - TLS_AES_256_GCM_SHA384
  - TLS_CHACHA20_POLY1305_SHA256
  - TLS_AES_128_CCM_8_SHA256
  - TLS_AES_128_CCM_SHA256

- DHE and ECDHE (5 each, but mostly X22519 and P-256)
Changes: Cipher Suites (2)

Hell, man

Removal of Static RSA and Diffie-Hellman Cipher Suites

- PFS now mandatory
- Private Key no more useable for middle-boxes
  - Network monitoring, compliance, …
  - NCSC stands against
    [TLS 1.3: better for individuals - harder for enterprises]
- Reply from [Adam Langley]
- Proposal for static DH [I-D.green-tls-static-dh-in-tls13]
Changes: Cipher Suites (3)

TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
  Hash algorithm
  Encryption algorithm
  Authentication algorithm
  Key exchange algorithm
  Protocol

TLS 1.2 cipher suite naming convention

TLS_AES_128_GCM_SHA256
  Hash algorithm
  Encryption algorithm
  Protocol

TLS 1.3 cipher suite naming convention
Other Handshake changes

Fall back and regroup

- Full handshake signature
- Downgrade protection (improve TLS_FALLBACK_SCSV)
- Curve 25519 and 448
**Anti-Downgrade Protection**

TLS 1.3 has a downgrade protection mechanism embedded in the server’s random value. TLS 1.3 servers which negotiate TLS 1.2 or below in response to a ClientHello MUST set the last 8 bytes of their Random value specially in their ServerHello. If negotiating TLS 1.2, TLS 1.3 servers MUST set the last 8 bytes of their Random value to the bytes:

44 4F 57 4D 47 52 44 01
D O W N G R D 01

- Value stored in the ServerRandom
- Last byte to 00 if negotiating TLS 1.1 or below
- ServerRandom covered by signature
Changes: Resumption / Session Handling

TLS 1.2

- Client Hello
- Session ID / Ticket
- Finished
- HTTP GET

TLS 1.3 (0-RTT)

- Client Hello
- Session Ticket (PSK)
- Key share
- Server Hello
- Key share
- Finished
- HTTP GET
- HTTP Answer

Source: cloudflare

- Session ID/Ticket ⇒ Ticket+PSK
- Interesting for mobile/low-resources devices
# Changes: Resumption and o-RTT

## o-RTT caveats

- **Replayable**
- No forward secrecy
- Trust server’s storage
  - for security
  - for Session Tickets duration

## Workarounds

- Short Session Tickets
- Allow o-RTT only for GET requests
- Requires to trust both client and server
Super cool “Secret” handshake

Handshake Encryption

- Most of the handshake is now encrypted
- Including the server certificate
- Including many extensions
Nice curves makes TLS sexy

**Changes: Key Schedule Generation**

**TLS 1.3: HKDF**

- TLS <= 1.2 defines PRF algorithm
- TLS 1.3 replaces this with HKDF
  - HKDF encapsulates how TLS uses HMAC
  - Re-used in other protocols
  - Separate cryptographic analysis already done
- HMAC is integral to HKDF
  - HMAC requires the Cryptographic Hash algorithm specified in the cipher suite (SHA256 or SHA384)
TLS 1.3: Impacts on Detection
Certificate Encryption

- Extracting certificates now impossible
  - Except for active methods
- No more visibility from network
- Certificate was useful for detection 😞
- Using SNI still possible
Handshake Encryption

The Knights Who Say Ni

Server Name Indication

- SNI can be used, but
  - encrypted SNI being added
    [SNI Encryption in TLS Through Tunneling]
  - eSNI now in Firefox nightly
- We can still use Client/Server profiles
  - Detect new/changed profiles
  - tlsfingerprint.io
  - JA3 + JA3S hashes
Interesting Items for Detection

- o-RTT is a security risk (replay, no PFS)
- ⇒ Detect/alert on early data extension
Interesting Items for Detection

Downgrade attempts

- Could indicate a MITM
- \( \Rightarrow \) Detect/alert ServerRandom magic bytes
Interesting Items for Detection

Other items

- Many messages have been removed
- ⇒ Detect/alert other messages
Interlude: sleeping meerkat

At this point I hope everyone is not asleep
Backwards compatibility

Other items

- Clients and Servers must be able to talk to previous versions
- ClientHello compatible with TLS <= 1.2
  - Some unused fields
  - Tons of extensions
  - Risks of data exfiltration / C2 commands
- ⇒ Check values
Handshake Encryption

Extensions

- 21 extensions in main RFC, many in others
- Cleartext instead of encrypted (and reverse)
- Invalid extensions combination
- Unexpected extensions
Parsing extensions:

```rust
match parse_tls_extensions(&ch.ext) {
    Err(e) => { err!("parsing failed: {?:?}" , e); },
    Ok((rem, msg)) => {
        match msg {
            Encrypted(_) => { ... },
            SNI(_) => { ... },
            EarlyData(_) => { ... },
            PreSharedKey(ref psk) => {
                log!("session resumption");
                // ...
        }
    }
```
The case of version

Jen, “This" is the Internet

**TLS Version**
- Negotiation of version
- TLS version is stored in:
  - the record header
  - the ClientHello
  - the `supported_versions` extension
- Usually, ClientHello claims 1.2, and record 1.0
- The extension specifies the supported (CH) / selected value (SH)
- ⇒ checking the version is much more complex
**O’Browser, Where Art Thou?**

**Deployment status**

- First push in browsers (Chrome 56), then pulled back ("Bluecoat and other proxies hang up during TLS 1.3")
- Most browsers now implements it
  - except that old OS/browsers will stay for long in companies
- OpenSSL 1.1.1, …
  - and custom implementations
- Will take longer for embedded, IoT, …
- Reductive answer to non-deployment: middleboxes
Conclusion
Have you tried turning it off and on again?

**TLS 1.3**
- Huge changes (more 2.0 than 1.3)
  - Lots of improvements
  - Focused on privacy
- Reduce observable data
  - Still, many detection rules can be written
  - TODO: write them
- Upgrade will take some time!
Questions?

More resources

- [RFC8446]
- https://tls13.ulfheim.net/
- Filippo Valsorda’s CCC talk
- Cloudflare blog