WHITE PAPER

Transport Layer Security (TLS) 1.3

Impact on web gateways

Transport Layer Security (TLS) 1.3

Impact on web gateways

New transport layer security (TLS) protocols are usually an expected evolution of encryption protocols. However, no version in the past has caused such confusion as TLS 1.3. The working group finished their work in mid-2017. Usually, rapid adoption as part of crypto-providers such as OpenSSL is expected. However, an initial non-beta implementation is still pending.

Surveying various resources, it appears that the changes that are part of TLS 1.3 are significant—more than just the inclusion of better encryption and minor changes on handshakes. Therefore, thorough testing is required before the final version is released.

Looking at the timeframes of the versions of secure sockets layer (SSL)/TLS and the gap since the last version of TLS and the TLS 1.3 version, it is evident that something has to happen quickly to avoid the SSL attacks we have witnessed in the last few years, such as POODLE, Heartbleed, and others—and new ones that come along.

SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3
1995	1996	1999	2006	2008	2018

This report was written by:

Michael Schneider, CISSP|CCSK, Senior Product Manager

Scope of TLS 1.3

The RFC lists the following items as major differences to TLS 1.2:

1.2. Major Differences from TLS 1.2

- Port the CFRG curves and signatures work from RFC4492bis.
- Remove sequence number and version from additional_data, which is now empty.
- Reorder values in HkdfLabel.
- Add support for version anti-downgrade mechanism.
- Update IANA considerations section and relax some of the policies.
- Unify authentication modes. Add post-handshake client authentication.
- Remove early_handshake content type. Terminate 0-RTT data with an alert.
- Reset sequence number upon key change (as proposed by Fournet et al).
- Remove ClientCertificateTypes field from CertificateRequest and add extensions.
- Merge client and server key shares into a single extension.
- Change to RSA-PSS signatures for handshake messages.
- Remove support for DSA.
- Update key schedule, per suggestions by Hugo, Hoeteck, and Bjoern Tackmann.
- Add support for per-record padding.
- Switch to encrypted record ContentType.
- Change HKDF labeling to include protocol version and value lengths.
- Shift the final decision to abort a handshake due to incompatible certificates to the client rather than having servers abort early.
- Deprecate SHA-1 with signatures.
- Add MTI algorithms.
- Remove support for weak and lesser-used named curves.
- Remove support for MD5 and SHA-224 hashes with signatures.
- Update lists of available AEAD cipher suites and error alerts.
- Reduce maximum permitted record expansion for AEAD from 2048 to 256 octets.
- Require digital signatures even when a previous configuration is used.
- Merge EarlyDataIndication and KnownConfiguration.
- Change code point for server_configuration to avoid collision with server_hello_done.
- Relax certificate_list ordering requirement to match current practice.

- Integration of semi-ephemeral DH proposal.
- Add initial 0-RTT support.
- Remove resumption and replace with PSK + tickets.
- Move ClientKeyShare into an extension.
- Move to HKDF.
- Prohibit RC4 negotiation for backwards compatibility.
- Freeze and deprecate record layer version field.
- Update format of signatures with context.
- Remove explicit IV.
- Prohibit SSL negotiation for backwards compatibility.
- Fix which MS is used for exporters.
- Modify key computations to include session hash.
- Remove ChangeCipherSpec.
- Renumber the new handshake messages to be somewhat more consistent with existing convention and to remove a duplicate registration.
- Remove renegotiation.
- Remove point format negotiation.
- Remove GMT time.
- Merge in support for ECC from RFC 4492 but without explicit curves.
- Remove the unnecessary length field from the AD input to AEAD ciphers.
- Rename {Client,Server}KeyExchange to {Client,Server} KeyShare.
- Add an explicit HelloRetryRequest to reject the client's.
- Increment version number.
- Rework handshake to provide 1-RTT mode.
- Remove custom DHE groups.
- Remove support for compression.
- Remove support for static RSA and DH key exchange.
- Remove support for non-AEAD ciphers.

(For more information, visit: https://tools.ietf.org/html/draft-ietf-tls-tls13-21 for full current RFC.)

This is an impressive list of enhancements and changes that will be incorporated into the spec. But, it's worth mentioning that man-in-the-middle (MITM) attacks are still possible with TLS 1.3. Even more important, these types of attacks will be the only reliable way to get access to encrypted data. Rapid7 raises that concern here: https://blog.rapid7.com/2016/11/10/conflicting-perspectives-on-the-tls-13-draft/. (See the section entitled "The request.")

Connection Creation: TLS 1.2 and 1.3

Two features in the list above stand out: RTT-1 and RTT-0 support. RTT stands for Round Trip Time and means that there is only one RTT needed to establish the encrypted communication and best case zero RTT.

Let's take a look at the communication in TLS 1.2.

When building a connection between client and server TLS 1.2 uses 2 RTT to connect, exchange keys, agree on the encryption, and, finally, exchange data over an encrypted channel. When a client has connected to a server previously, it is able—also in TLS 1.2—to decrease the RTT by one and do an RTT-1-based reconnect. TLS 1.2 client uses a so-called session ticket/ID to reconnect to servers that are sent in the client "hello" and helps the server simply "safe" the exchange, as both parties have done that already and simply have to reuse the data they had agreed on previously.

TLS .1.3, in contrast, uses an RTT-1 while building the connection and sends key data, such as the supported ciphers. In addition, the client tries to make an educated guess on what key the server could be using and sends that within the initial process. All the server has to do is to agree and send back the matching key material such as the certificate, which is now encrypted as well, as it has already acquired the key in the initial request. After the server responds, the client simply needs to acknowledge that all is well, and, after that, the encrypted connection is established.

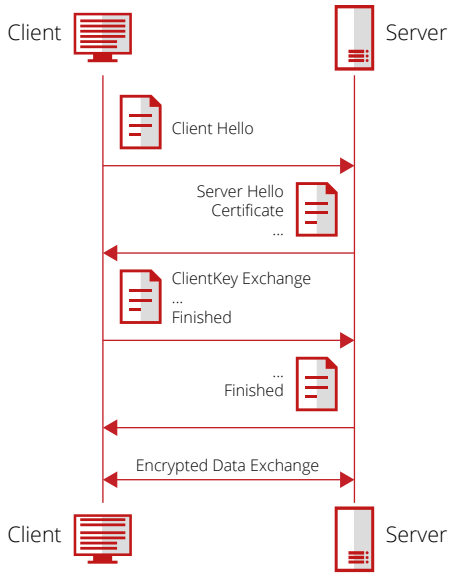

Figure 1. Initial TLS 1.2 connection.

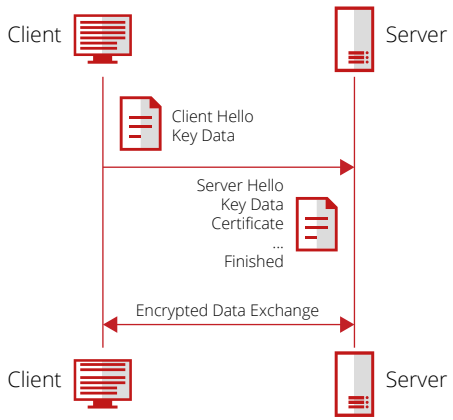

Figure 2. TLS 1.3 connection.

If you now add the method for session resumption—explained above for TLS 1.2—you see that the RTT is reduced to essentially zero. The only difference is that during the connection of a TLS 1.3 server and a TLS 1.3 client, both agree on a pre-shared key—PSK—that is used later to resume the connection. It is no longer the session ID. This PSK is also encrypted due to the previous key exchange.

If a TLS client is not trying to resume or reuse the connection, it will send the PSK along with the HTTP request. The server will respond with its equivalents and the HTTP response.

What About Man-in-the-Middle Attacks Now?

Based on the previously provided explanation on how a TLS connection is established in T:S 1.3, it becomes evident that an offline decryption is no longer possible. The deprecation of RSA hashes removed the ability to see a clear text hash as part of the encryption negotiation. The lack of this clear text information does not allow attackers to create keys in a brute-force manner and then compare their hashes against the one in the connection in order to find the matching one.

Active participation in the connection negotiation and creation is required in order to decrypt the data and secure the payload in the encrypted section.

Given that, the urban legend of the inability of an MITM proxy to scan TLS 1.3 remains just that. This might have been spawned when Chrome, in early 2017, added support for TLS 1.3, which then disabled certain proxies from scanning the data stream or even blocking the connections. But, in general, TLS 1.3 will enhance security through stronger keys and ciphers. It will increase performance by reducing RTT but will not remove the option of scanning traffic.

McAfee Web Gateway and McAfee Web Gateway Cloud Service

With the release of McAfee® Web Gateway version 8.2, the proxy fully supports TLS 1.3 on the client as well as on the server side. This bidirectional implementation enables customers to avoid downgrading TLS 1.3 to TLS 1.2 and thereby weaken their encryption and security posture. HTTPS content scanning will continue, as with previous versions of TLS, and content security filters are available to TLS 1.3-based traffic as well.

For more information, visit McAfee Web Gateway and McAfee® Web Gateway Cloud Service.

About McAfee

McAfee is the device-to-cloud cybersecurity company. Inspired by the power of working together, McAfee creates business and consumer solutions that make our world a safer place. By building solutions that work with other companies' products, McAfee helps businesses orchestrate cyber environments that are truly integrated, where protection, detection, and correction of threats happen simultaneously and collaboratively. By protecting consumers across all their devices, McAfee secures their digital lifestyle at home and away. By working with other security players, McAfee is leading the effort to unite against cybercriminals for the benefit of all.

www.mcafee.com.

2821 Mission College Blvd.
Santa Clara, CA 95054
888.847.8766
www.mcafee.com