Internet Engineering Task Force (IETF)
Request for Comments: 7918
Category: Informational
ISSN: 2070-1721

A. Langley
N. Modadugu
B. Moeller
Google

August 2016

Transport Layer Security (TLS) False Start

Abstract

This document specifies an optional behavior of Transport Layer Security (TLS) client implementations, dubbed "False Start". It affects only protocol timing, not on-the-wire protocol data, and can be implemented unilaterally. A TLS False Start reduces handshake latency to one round trip.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7918.

Copyright Notice

Table of Contents

1. Introduction

A full handshake in TLS protocol versions up to TLS 1.2 [RFC5246] requires two full protocol rounds (four flights) before the handshake is complete and the protocol parties may begin to send application data. Thus, using TLS can add a latency penalty of two network round-trip times for application protocols in which the client sends data first, such as HTTP [RFC7230]. Figure 1 (copied from [RFC5246]) shows the message flow for a full handshake.

```
Client                                               Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                         [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data
```

Figure 1: Message Flow for a Full Handshake

This document describes a technique that alleviates the latency burden imposed by TLS: the client-side TLS False Start. If certain conditions are met, the client can start to send application data when the full handshake is only partially complete, namely, when the client has sent its own ChangeCipherSpec and Finished messages (thus having updated its TLS Record Protocol write state as negotiated in the handshake) but has yet to receive the server’s ChangeCipherSpec and Finished messages. (Per Section 7.4.9 of [RFC5246], after a full handshake, the client would have to delay sending application data until it has received and validated the server’s Finished message.) Accordingly, the latency penalty for using TLS with HTTP can be kept at one round-trip time.

Note that in practice, the TCP three-way handshake [RFC0793] typically adds one round-trip time before the client can even send the ClientHello. See [RFC7413] for a latency improvement at that level.

When an earlier TLS session is resumed, TLS uses an abbreviated handshake with only three protocol flights. For application protocols in which the client sends data first, this abbreviated handshake adds just one round-trip time to begin with, so there is no need for a client-side False Start. However, if the server sends application data first, the abbreviated handshake adds two round-trip times, and this could be reduced to just one added round-trip time by doing a server-side False Start. There is little need for this in practice, so this document does not consider server-side False Starts further.

Note also that TLS versions 1.3 [TLS13] and beyond are out of scope for this document. False Start will not be needed with these newer versions since protocol flows minimizing the number of round trips have become a first-order design goal.

In a False Start, when the client sends application data before it has received and verified the server’s Finished message, there are two possible outcomes:

- The handshake completes successfully: The handshake is retroactively validated when both Finished messages have been received and verified. This retroactively validates the handshake. In this case, the transcript of protocol data carried over the transport underlying TLS will look as usual, apart from the different timing.

- The handshake fails: If a party does not receive the other side’s Finished message or if the Finished message’s contents are not correct, the handshake never gets validated. This means that an attacker may have removed, changed, or injected handshake messages. In this case, data has been sent over the underlying transport that would not have been sent without the False Start.

The latter scenario makes it necessary to restrict when a False Start is allowed, as described in this document. Section 3 considers basic requirements for using False Start. Section 4 specifies the behavior for clients, referring to important security considerations in Section 5.

2. Requirements Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT","RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. False Start Compatibility

TLS False Start as described in detail in the subsequent sections, if implemented, is an optional feature.

A TLS server implementation is defined to be "False Start compatible" if it tolerates receiving TLS records on the transport connection early, before the protocol has reached the state to process them. For successful use of client-side False Start in a TLS connection, the server has to be False Start compatible. Out-of-band knowledge that the server is False Start compatible may be available, e.g., if this is mandated by specific application profile standards. As discussed in Appendix A, the requirement for False Start compatibility generally does not pose a hindrance in practice.

4. Client-Side False Start

This section specifies a change to the behavior of TLS client implementations in full TLS handshakes.

When the client has sent its ChangeCipherSpec and Finished messages, its default behavior per [RFC5246] is to not send application data until it has received the server’s ChangeCipherSpec and Finished messages, which completes the handshake. With the False Start protocol modification, the client MAY send application data earlier (under the new Cipher Spec) if each of the following conditions is satisfied:

- The application layer has requested the TLS False Start option.

- The symmetric cipher defined by the cipher suite negotiated in this handshake has been whitelisted for use with False Start according to the Security Considerations in Section 5.1.

- The protocol version chosen by ServerHello.server_version has been whitelisted for use with False Start according to the Security Considerations in Section 5.2.

- The key exchange method defined by the cipher suite negotiated in this handshake and, if applicable, its parameters have been whitelisted for use with False Start according to the Security Considerations in Section 5.3.

- In the case of a handshake with client authentication, the client certificate type has been whitelisted for use with False Start according to the Security Considerations in Section 5.3.

The rules for receiving data from the server remain unchanged.

Note that the TLS client cannot infer the presence of an authenticated server until all handshake messages have been received. With False Start, unlike with the default handshake behavior, applications are able to send data before this point has been reached: from an application point of view, being able to send data does not imply that an authenticated peer is present. Accordingly, it is recommended that TLS implementations allow the application layer to query whether the handshake has completed.

5. Security Considerations

In a TLS handshake, the Finished messages serve to validate the entire handshake. These messages are based on a hash of the handshake so far processed by a Pseudorandom Function (PRF) keyed with the new master secret (serving as a Message Authentication Code (MAC)) and are also sent under the new Cipher Spec with its keyed MAC, where the MAC key again is derived from the master secret. The protocol design relies on the assumption that any server and/or client authentication done during the handshake carries over to this. While an attacker could, for example, have changed the cipher suite list sent by the client to the server and thus influenced cipher suite selection (presumably towards a less secure choice) or could have made other modifications to handshake messages in transmission, the attacker would not be able to round off the modified handshake with a valid Finished message: every TLS cipher suite is presumed to key the PRF appropriately to ensure unforgeability. Verifying the Finished messages validates the handshake and confirms that the handshake has not been tampered with; thus, secure encryption is bootstrapped from secure authentication.

Using False Start interferes with this approach of bootstrapping secure encryption from secure authentication, as application data may have already been sent before Finished validation confirms that the handshake has not been tampered with -- so there is generally no way to be sure that communication with the expected peer is indeed taking place during the False Start. Instead, the security goal is to ensure that if anyone at all can decrypt the application data sent in a False Start, it must be the legitimate peer. While an attacker could be influencing the handshake (restricting cipher suite selection, modifying key exchange messages, etc.), the attacker should not be able to benefit from this. The TLS protocol already relies on such a security property for authentication; with False Start, the same is needed for encryption. This motivates the rules put forth in the following subsections.

It is prudent for applications to be even more restrictive. If heuristically a small list of cipher suites and a single protocol version is found to be sufficient for the majority of TLS handshakes in practice, it could make sense to forego False Start for any handshake that does not match this expected pattern, even if there is no concrete reason to assume a cryptographic weakness. Similarly, if handshakes almost always use ephemeral Elliptic Curve Diffie-Hellman (ECDH) over one of a few named curves, it could make sense to disallow False Start with any other supported curve.

5.1. Symmetric Cipher

Clients MUST NOT use the False Start protocol modification in a handshake unless the cipher suite uses a symmetric cipher that is considered cryptographically strong.

Implementations may have their own classification of ciphers (and may additionally allow the application layer to provide a classification), but generally only symmetric ciphers with an effective key length of 128 bits or more can be considered strong. Also, various ciphers specified for use with TLS are known to have cryptographic weaknesses regardless of key length (none of the ciphers specified in [RFC4492] and [RFC5246] can be recommended for use with False Start). The AES_128_GCM_SHA256 or AES_256_GCM_SHA384 ciphers specified in [RFC5288] and [RFC5289] can be considered sufficiently strong for most uses. Implementations that support additional cipher suites have to be careful to whitelist only suitable symmetric ciphers; if in doubt, False Start should not be used with a given symmetric cipher.

While an attacker can change handshake messages to force a downgrade to a less secure symmetric cipher than otherwise would have been chosen, this rule ensures that in such a downgrade attack, no application data will be sent under an insecure symmetric cipher.

5.2. Protocol Version

Clients MUST NOT use the False Start protocol modification in a handshake unless the protocol version chosen by ServerHello.server_version has been whitelisted for this use.

Generally, to avoid potential protocol downgrade attacks, implementations should whitelist only their latest (highest-valued) supported TLS protocol version (and, if applicable, any earlier protocol versions that they would use in fallback retries without TLS_FALLBACK_SCSV [RFC7507]).

The details of nominally identical cipher suites can differ between protocol versions, so this reinforces Section 5.1.

5.3. Key Exchange and Client Certificate Type

Clients MUST NOT use the False Start protocol modification in a handshake unless the cipher suite uses a key exchange method that has been whitelisted for this use. Also, clients MUST NOT use the False Start protocol modification unless any parameters to the key exchange methods (such as ServerDHParams or ServerECDHParams) have been whitelisted for this use. Furthermore, when using client authentication, clients MUST NOT use the False Start protocol modification unless the client certificate type has been whitelisted for this use.

Implementations may have their own whitelists of key exchange methods, parameters, and client certificate types (and may additionally allow the application layer to specify whitelists). Generally, out of the options from [RFC5246] and [RFC4492], the following whitelists are recommended:

- Key exchange methods: DHE_RSA, ECDHE_RSA, DHE_DSS, ECDHE_ECDSA

- Parameters: well-known DH groups (at least 3,072 bits), named curves (at least 256 bits)

- Client certificate types: none

However, if an implementation that supports only key exchange methods from [RFC5246] and [RFC4492] does not support any of the above key exchange methods, all of its supported key exchange methods can be whitelisted for False Start use. Care is required with any additional key exchange methods, as these may not have similar properties.

The recommended whitelists are such that if cryptographic algorithms suitable for forward secrecy would possibly be negotiated, no False Start will take place if the current handshake fails to provide forward secrecy. (Forward secrecy can be achieved using ephemeral Diffie-Hellman or ephemeral Elliptic Curve Diffie-Hellman; there is no forward secrecy when using a key exchange method of RSA, RSA_PSK, DH_DSS, DH_RSA, ECDH_ECDSA, or ECDH_RSA, or a client certificate type of rsa_fixed_dh, dss_fixed_dh, rsa_fixed_ecdh, or ecdsa_fixed_ecdh.) As usual, the benefits of forward secrecy may need to be balanced against efficiency, and accordingly, even implementations that support the above key exchange methods might whitelist further key exchange methods and client certificate types.

Client certificate types rsa_sign, dss_sign, and ecdsa_sign do allow forward security, but using False Start with any of these means sending application data tied to the client’s signature before the server’s authenticity (and thus the CertificateRequest message) has been completely verified, so these too are not generally suitable for the client certificate type whitelist.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, DOI 10.17487/RFC4492, May 2006, <http://www.rfc-editor.org/info/rfc4492>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5288] Salowey, J., Choudhury, A., and D. McGrew, "AES Galois Counter Mode (GCM) Cipher Suites for TLS", RFC 5288, DOI 10.17487/RFC5288, August 2008, <http://www.rfc-editor.org/info/rfc5288>.

[RFC5289] Rescorla, E., "TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)", RFC 5289, DOI 10.17487/RFC5289, August 2008, <http://www.rfc-editor.org/info/rfc5289>.

6.2. Informative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <http://www.rfc-editor.org/info/rfc7413>.

[RFC7507] Moeller, B. and A. Langley, "TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks", RFC 7507, DOI 10.17487/RFC7507, April 2015, <http://www.rfc-editor.org/info/rfc7507>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", Work in Progress, draft-ietf-tls-tls13-14, July 2016.

Appendix A. Implementation Notes

TLS False Start is a modification to the TLS protocol, and some implementations that conform to [RFC5246] may have problems interacting with implementations that use the False Start modification. If the peer uses a False Start, application data records may be received directly following the peer’s Finished message, before the TLS implementation has sent its own Finished message. False Start compatibility as defined in Section 3 ensures that these records with application data will simply remain buffered for later processing.

A False Start compatible TLS implementation does not have to be aware of the False Start concept and is certainly not expected to detect whether a False Start handshake is currently taking place: thanks to transport layer buffering, typical implementations will be False Start compatible without having been designed for it.

Acknowledgments

The authors wish to thank Wan-Teh Chang, Ben Laurie, Martin Thomson, Eric Rescorla, and Brian Smith for their input.

Authors' Addresses

Adam Langley
Google Inc.
345 Spear St
San Francisco, CA 94105
United States of America

Email: agl@google.com

Nagendra Modadugu
Google Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America

Email: nagendra@cs.stanford.edu

Bodo Moeller
Google Switzerland GmbH
Brandschenkestrasse 110
Zurich 8002
Switzerland

Email: bmoeller@acm.org