Internet Engineering Task Force (IETF)
Request for Comments: 5746
Updates: 5246, 4366, 4347, 4346, 2246
Category: Standards Track
ISSN: 2070-1721

E. Rescorla
RTFM, Inc.
M. Ray
S. Dispensa
PhoneFactor
N. Oskov
Microsoft

February 2010

Transport Layer Security (TLS) Renegotiation Indication Extension

Abstract

Secure Socket Layer (SSL) and Transport Layer Security (TLS) renegotiation are vulnerable to an attack in which the attacker forms a TLS connection with the target server, injects content of his choice, and then splices in a new TLS connection from a client. The server treats the client's initial TLS handshake as a renegotiation and thus believes that the initial data transmitted by the attacker is from the same entity as the subsequent client data. This specification defines a TLS extension to cryptographically tie renegotiations to the TLS connections they are being performed over, thus preventing this attack.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

TLS [RFC5246] allows either the client or the server to initiate renegotiation -- a new handshake that establishes new cryptographic parameters. Unfortunately, although the new handshake is carried out using the cryptographic parameters established by the original handshake, there is no cryptographic binding between the two. This creates the opportunity for an attack in which the attacker who can intercept a client's transport layer connection can inject traffic of his own as a prefix to the client's interaction with the server. One form of this attack [Ray09] proceeds as shown below:

```
Client                        Attacker                        Server
------                        --------                        ------
                                  <----------- Handshake ---------->
                                  <======= Initial Traffic ========>
<--------------------------  Handshake ============================>
<======================== Client Traffic ==========================>
```

To start the attack, the attacker forms a TLS connection to the server (perhaps in response to an initial intercepted connection from the client). He then sends any traffic of his choice to the server. This may involve multiple requests and responses at the application layer, or may simply be a partial application layer request intended to prefix the client's data. This traffic is shown with == to indicate it is encrypted. He then allows the client's TLS handshake to proceed with the server. The handshake is in the clear to the attacker but encrypted over the attacker's TLS connection to the server. Once the handshake has completed, the client communicates with the server over the newly established security parameters with the server. The attacker cannot read this traffic, but the server believes that the initial traffic to and from the attacker is the same as that to and from the client.

If certificate-based client authentication is used, the server will see a stream of bytes where the initial bytes are protected but unauthenticated by TLS and subsequent bytes are authenticated by TLS and bound to the client's certificate. In some protocols (notably HTTPS), no distinction is made between pre- and post-authentication stages and the bytes are handled uniformly, resulting in the server believing that the initial traffic corresponds to the authenticated client identity. Even without certificate-based authentication, a variety of attacks may be possible in which the attacker convinces the server to accept data from it as data from the client. For instance, if HTTPS [RFC2818] is in use with HTTP cookies [RFC2965], the attacker may be able to generate a request of his choice validated by the client's cookie.

Some protocols -- such as IMAP or SMTP -- have more explicit transitions between authenticated and unauthenticated phases and require that the protocol state machine be partly or fully reset at such transitions. If strictly followed, these rules may limit the effect of attacks. Unfortunately, there is no requirement for state machine resets at TLS renegotiation, and thus there is still a potential window of vulnerability, for instance, by prefixing a command that writes to an area visible by the attacker with a command by the client that includes his password, thus making the client's password visible to the attacker (note that this precise attack does not work with challenge-response authentication schemes, but other attacks may be possible). Similar attacks are available with SMTP, and in fact do not necessarily require the attacker to have an account on the target server.

It is important to note that in both cases these attacks are possible because the client sends unsolicited authentication information without requiring any specific data from the server over the TLS connection. Protocols that require a round trip to the server over TLS before the client sends sensitive information are likely to be less vulnerable.

These attacks can be prevented by cryptographically binding renegotiation handshakes to the enclosing TLS cryptographic parameters, thus allowing the server to differentiate renegotiation from initial negotiation, as well as preventing renegotiations from being spliced in between connections. An attempt by an attacker to inject himself as described above will result in a mismatch of the cryptographic binding and can thus be detected. The data used in the extension is similar to, but not the same as, the data used in the tls-unique and/or tls-unique-for-telnet channel bindings described in [TLS-CHANNEL-BINDINGS]; however, this extension is not a general-purpose RFC 5056 [RFC5056] channel binding facility.

2. Conventions Used in This Document

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 [RFC2119].

3. Secure Renegotiation Definition

3.1. Additional Connection State

Both client and server need to store three additional values for each TLS connection state (see RFC 5246, Section 6.1). Note that these values are specific to connection (not a TLS session cache entry).

o a "secure_renegotiation" flag, indicating whether secure renegotiation is in use for this connection.

o "client_verify_data": the verify_data from the Finished message sent by the client on the immediately previous handshake. For currently defined TLS versions and cipher suites, this will be a 12-byte value; for SSLv3, this will be a 36-byte value.

o "server_verify_data": the verify_data from the Finished message sent by the server on the immediately previous handshake.

3.2. Extension Definition

This document defines a new TLS extension, "renegotiation_info" (with extension type 0xff01), which contains a cryptographic binding to the enclosing TLS connection (if any) for which the renegotiation is being performed. The "extension data" field of this extension contains a "RenegotiationInfo" structure:

```
struct {
    opaque renegotiated_connection<0..255>;
} RenegotiationInfo;
```

The contents of this extension are specified as follows.

- o If this is the initial handshake for a connection, then the "renegotiated_connection" field is of zero length in both the ClientHello and the ServerHello. Thus, the entire encoding of the extension is ff 01 00 01 00. The first two octets represent the extension type, the third and fourth octets the length of the extension itself, and the final octet the zero length byte for the "renegotiated_connection" field.

- o For ClientHellos that are renegotiating, this field contains the "client_verify_data" specified in Section 3.1.

- o For ServerHellos that are renegotiating, this field contains the concatenation of client_verify_data and server_verify_data. For current versions of TLS, this will be a 24-byte value (for SSLv3, it will be a 72-byte value).

This extension also can be used with Datagram TLS (DTLS) [RFC4347]. Although, for editorial simplicity, this document refers to TLS, all requirements in this document apply equally to DTLS.

3.3. Renegotiation Protection Request Signaling Cipher Suite Value

Both the SSLv3 and TLS 1.0/TLS 1.1 specifications require implementations to ignore data following the ClientHello (i.e., extensions) if they do not understand it. However, some SSLv3 and TLS 1.0 implementations incorrectly fail the handshake in such a case. This means that clients that offer the "renegotiation_info" extension may encounter handshake failures. In order to enhance compatibility with such servers, this document defines a second signaling mechanism via a special Signaling Cipher Suite Value (SCSV) "TLS_EMPTY_RENEGOTIATION_INFO_SCSV", with code point {0x00, 0xFF}. This SCSV is not a true cipher suite (it does not correspond to any valid set of algorithms) and cannot be negotiated. Instead, it has the same semantics as an empty "renegotiation_info" extension, as described in the following sections. Because SSLv3 and TLS implementations reliably ignore unknown cipher suites, the SCSV may be safely sent to any server. The SCSV can also be included in the SSLv2 backward compatible CLIENT-HELLO (see Appendix E.2 of [RFC5246]).

Note: a minimal client that does not support renegotiation at all can simply use the SCSV in all initial handshakes. The rules in the following sections will cause any compliant server to abort the handshake when it sees an apparent attempt at renegotiation by such a client.

3.4. Client Behavior: Initial Handshake

Note that this section and Section 3.5 apply to both full handshakes and session resumption handshakes.

o The client MUST include either an empty "renegotiation_info" extension, or the TLS_EMPTY_RENEGOTIATION_INFO_SCSV signaling cipher suite value in the ClientHello. Including both is NOT RECOMMENDED.

o When a ServerHello is received, the client MUST check if it includes the "renegotiation_info" extension:

  * If the extension is not present, the server does not support secure renegotiation; set secure_renegotiation flag to FALSE. In this case, some clients may want to terminate the handshake instead of continuing; see Section 4.1 for discussion.

  * If the extension is present, set the secure_renegotiation flag to TRUE. The client MUST then verify that the length of the "renegotiated_connection" field is zero, and if it is not, MUST abort the handshake (by sending a fatal handshake_failure alert).

    Note: later in Section 3, "abort the handshake" is used as shorthand for "send a fatal handshake_failure alert and terminate the connection".

- o When the handshake has completed, the client needs to save the client_verify_data and server_verify_data values for future use.

3.5. Client Behavior: Secure Renegotiation

This text applies if the connection's "secure_renegotiation" flag is set to TRUE (if it is set to FALSE, see Section 4.2).

- o The client MUST include the "renegotiation_info" extension in the ClientHello, containing the saved client_verify_data. The SCSV MUST NOT be included.

- o When a ServerHello is received, the client MUST verify that the "renegotiation_info" extension is present; if it is not, the client MUST abort the handshake.

- o The client MUST then verify that the first half of the "renegotiated_connection" field is equal to the saved client_verify_data value, and the second half is equal to the saved server_verify_data value. If they are not, the client MUST abort the handshake.

- o When the handshake has completed, the client needs to save the new client_verify_data and server_verify_data values.

3.6. Server Behavior: Initial Handshake

Note that this section and Section 3.7 apply to both full handshakes and session-resumption handshakes.

- o When a ClientHello is received, the server MUST check if it includes the TLS_EMPTY_RENEGOTIATION_INFO_SCSV SCSV. If it does, set the secure_renegotiation flag to TRUE.

- o The server MUST check if the "renegotiation_info" extension is included in the ClientHello. If the extension is present, set secure_renegotiation flag to TRUE. The server MUST then verify that the length of the "renegotiated_connection" field is zero, and if it is not, MUST abort the handshake.

- o If neither the TLS_EMPTY_RENEGOTIATION_INFO_SCSV SCSV nor the "renegotiation_info" extension was included, set the secure_renegotiation flag to FALSE. In this case, some servers may want to terminate the handshake instead of continuing; see Section 4.3 for discussion.

- o If the secure_renegotiation flag is set to TRUE, the server MUST include an empty "renegotiation_info" extension in the ServerHello message.

- o When the handshake has completed, the server needs to save the client_verify_data and server_verify_data values for future use.

TLS servers implementing this specification MUST ignore any unknown extensions offered by the client and they MUST accept version numbers higher than their highest version number and negotiate the highest common version. These two requirements reiterate preexisting requirements in RFC 5246 and are merely stated here in the interest of forward compatibility.

Note that sending a "renegotiation_info" extension in response to a ClientHello containing only the SCSV is an explicit exception to the prohibition in RFC 5246, Section 7.4.1.4, on the server sending unsolicited extensions and is only allowed because the client is signaling its willingness to receive the extension via the TLS_EMPTY_RENEGOTIATION_INFO_SCSV SCSV. TLS implementations MUST continue to comply with Section 7.4.1.4 for all other extensions.

3.7. Server Behavior: Secure Renegotiation

This text applies if the connection's "secure_renegotiation" flag is set to TRUE (if it is set to FALSE, see Section 4.4).

- o When a ClientHello is received, the server MUST verify that it does not contain the TLS_EMPTY_RENEGOTIATION_INFO_SCSV SCSV. If the SCSV is present, the server MUST abort the handshake.

- o The server MUST verify that the "renegotiation_info" extension is present; if it is not, the server MUST abort the handshake.

- The server MUST verify that the value of the "renegotiated_connection" field is equal to the saved client_verify_data value; if it is not, the server MUST abort the handshake.

- The server MUST include a "renegotiation_info" extension containing the saved client_verify_data and server_verify_data in the ServerHello.

- When the handshake has completed, the server needs to save the new client_verify_data and server_verify_data values.

4. Backward Compatibility

Existing implementations that do not support this extension are widely deployed and, in general, must interoperate with newer implementations that do support it. This section describes considerations for backward compatible interoperation.

4.1. Client Considerations

If a client offers the "renegotiation_info" extension or the TLS_EMPTY_RENEGOTIATION_INFO_SCSV SCSV and the server does not reply with "renegotiation_info" in the ServerHello, then this indicates that the server does not support secure renegotiation. Because some attacks (see Section 1) look like a single handshake to the client, the client cannot determine whether or not the connection is under attack. Note, however, that merely because the server does not acknowledge the extension does not mean that it is vulnerable; it might choose to reject all renegotiations and simply not signal it. However, it is not possible for the client to determine purely via TLS mechanisms whether or not this is the case.

If clients wish to ensure that such attacks are impossible, they need to terminate the connection immediately upon failure to receive the extension without completing the handshake. Such clients MUST generate a fatal "handshake_failure" alert prior to terminating the connection. However, it is expected that many TLS servers that do not support renegotiation (and thus are not vulnerable) will not support this extension either, so in general, clients that implement this behavior will encounter interoperability problems. There is no set of client behaviors that will guarantee security and achieve maximum interoperability during the transition period. Clients need to choose one or the other preference when dealing with potentially un-upgraded servers.

4.2. Client Behavior: Legacy (Insecure) Renegotiation

This text applies if the connection's "secure_renegotiation" flag is set to FALSE.

It is possible that un-upgraded servers will request that the client renegotiate. It is RECOMMENDED that clients refuse this renegotiation request. Clients that do so MUST respond to such requests with a "no_renegotiation" alert (RFC 5246 requires this alert to be at the "warning" level). It is possible that the apparently un-upgraded server is in fact an attacker who is then allowing the client to renegotiate with a different, legitimate, upgraded server. If clients nevertheless choose to renegotiate, they MUST behave as described below.

Clients that choose to renegotiate MUST provide either the TLS_EMPTY_RENEGOTIATION_INFO_SCSV SCSV or "renegotiation_info" in their ClientHello. In a legitimate renegotiation with an un-upgraded server, that server should ignore both of these signals. However, if the server (incorrectly) fails to ignore extensions, sending the "renegotiation_info" extension may cause a handshake failure. Thus, it is permitted, though NOT RECOMMENDED, for the client to simply send the SCSV. This is the only situation in which clients are permitted to not send the "renegotiation_info" extension in a ClientHello that is used for renegotiation.

Note that in the case of a downgrade attack, if this is an initial handshake from the server's perspective, then use of the SCSV from the client precludes detection of this attack by the server (if this is a renegotiation from the server's perspective, then it will detect the attack). However, the attack will be detected by the client when the server sends an empty "renegotiation_info" extension and the client is expecting one containing the previous verify_data. By contrast, if the client sends the "renegotiation_info" extension, then the server will immediately detect the attack.

When the ServerHello is received, the client MUST verify that it does not contain the "renegotiation_info" extension. If it does, the client MUST abort the handshake. (Because the server has already indicated it does not support secure renegotiation, the only way that this can happen is if the server is broken or there is an attack.)

4.3. Server Considerations

If the client does not offer the "renegotiation_info" extension or the TLS_EMPTY_RENEGOTIATION_INFO_SCSV SCSV, then this indicates that the client does not support secure renegotiation. Although the attack described in Section 1 looks like two handshakes to the

server, other attacks may be possible in which the renegotiation is seen only by the client. If servers wish to ensure that such attacks are impossible, they need to terminate the connection immediately upon failure to negotiate the use of secure renegotiation. Servers that do choose to allow connections from unpatched clients can still prevent the attack described in Section 1 by refusing to renegotiate over those connections.

In order to enable clients to probe, even servers that do not support renegotiation MUST implement the minimal version of the extension described in this document for initial handshakes, thus signaling that they have been upgraded.

4.4. Server Behavior: Legacy (Insecure) Renegotiation

This text applies if the connection's "secure_renegotiation" flag is set to FALSE.

It is RECOMMENDED that servers not permit legacy renegotiation. If servers nevertheless do permit it, they MUST follow the requirements in this section.

- o When a ClientHello is received, the server MUST verify that it does not contain the TLS_EMPTY_RENEGOTIATION_INFO_SCSV SCSV. If the SCSV is present, the server MUST abort the handshake.

- o The server MUST verify that the "renegotiation_info" extension is not present; if it is, the server MUST abort the handshake.

4.5. SSLv3

While SSLv3 is not a protocol under IETF change control (see

[SSLv3]), it was the original basis for TLS and most TLS implementations also support SSLv3. The IETF encourages SSLv3 implementations to adopt the "renegotiation_info" extension and SCSV as defined in this document. The semantics of the SCSV and extension are identical to TLS stacks except for the size of the verify_data values, which are 36 bytes long each. Note that this will require adding at least minimal extension processing to such stacks. Clients that support SSLv3 and offer secure renegotiation (either via SCSV or "renegotiation_info") MUST accept the "renegotiation_info" extension from the server, even if the server version is {0x03, 0x00}, and behave as described in this specification. TLS servers that support secure renegotiation and support SSLv3 MUST accept SCSV or the "renegotiation_info" extension and respond as described in this specification even if the offered client version is {0x03, 0x00}. SSLv3 does not define the "no_renegotiation" alert (and does

not offer a way to indicate a refusal to renegotiate at a "warning" level). SSLv3 clients that refuse renegotiation SHOULD use a fatal handshake_failure alert.

5. Security Considerations

The extension described in this document prevents an attack on TLS. If this extension is not used, TLS renegotiation is subject to an attack in which the attacker can inject their own conversation with the TLS server as a prefix to the client's conversation. This attack is invisible to the client and looks like an ordinary renegotiation to the server. The extension defined in this document allows renegotiation to be performed safely. Servers SHOULD NOT allow clients to renegotiate without using this extension. Many servers can mitigate this attack simply by refusing to renegotiate at all.

While this extension mitigates the man-in-the-middle attack described in the overview, it does not resolve all possible problems an application may face if it is unaware of renegotiation. For example, during renegotiation, either the client or the server can present a different certificate than was used earlier. This may come as a surprise to application developers (who might have expected, for example, that a "getPeerCertificates()" API call returns the same value if called twice), and might be handled in an insecure way.

TLS implementations SHOULD provide a mechanism to disable and enable renegotiation.

TLS implementers are encouraged to clearly document how renegotiation interacts with the APIs offered to applications (for example, which API calls might return different values on different calls, or which callbacks might get called multiple times).

To make life simpler for applications that use renegotiation but do not expect the certificate to change once it has been authenticated, TLS implementations may also wish to offer the applications the option to abort the renegotiation if the peer tries to authenticate with a different certificate and/or different server name (in the server_name extension) than was used earlier. TLS implementations may alternatively offer the option to disable renegotiation once the client certificate has been authenticated. However, enabling these options by default for all applications could break existing applications that depend on using renegotiation to change from one certificate to another. (For example, long-lived TLS connections could change to a renewed certificate; or renegotiation could select a different cipher suite that requires using a different certificate.)

Finally, designers of applications that depend on renegotiation are reminded that many TLS APIs represent application data as a simple octet stream; applications may not be able to determine exactly which application data octets were received before, during, or after renegotiation. Especially if the peer presents a different certificate during renegotiation, care is needed when specifying how the application should handle the data.

6. IANA Considerations

IANA has added the extension code point 65281 (0xff01), which has been used for prototype implementations, for the "renegotiation_info" extension to the TLS ExtensionType values registry.

IANA has added TLS cipher suite number 0x00,0xFF with name TLS_EMPTY_RENEGOTIATION_INFO_SCSV to the TLS Cipher Suite registry.

7. Acknowledgements

This vulnerability was originally discovered by Marsh Ray and independently rediscovered by Martin Rex. The general concept behind the extension described here was independently invented by Steve Dispensa, Nasko Oskov, and Eric Rescorla with refinements from Nelson Bolyard, Pasi Eronen, Michael D'Errico, Stephen Farrell, Michael Gray, David-Sarah Hopwood, Ben Laurie, David Makepeace, Bodo Moeller, Martin Rex, Peter Robinson, Jesse Walker, Nico Williams, and other members of the Project Mogul team and the TLS WG.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

8.2. Informative References

[RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.

[TLS-CHANNEL-BINDINGS]
Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", Work in Progress, October 2009.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC2965] Kristol, D. and L. Montulli, "HTTP State Management Mechanism", RFC 2965, October 2000.

[Ray09] Ray, M., "Authentication Gap in TLS Renegotiation", November 2009, <http://extendedsubset.com/?p=8>.

[SSLv3]   Freier, A., Karlton, P., and P. Kocher, "The SSL Protocol Version 3.0", Work in Progress, November 1996.

Authors' Addresses

Eric Rescorla
RTFM, Inc.

2064 Edgewood Drive
Palo Alto, CA 94303
USA

EMail: ekr@rtfm.com

Marsh Ray
PhoneFactor
7301 W 129th Street
Overland Park, KS 66213
USA

EMail: marsh@extendedsubset.com

Steve Dispensa
PhoneFactor
7301 W 129th Street
Overland Park, KS 66213
USA

EMail: dispensa@phonefactor.com

Nasko Oskov
Microsoft
One Microsoft Way
Redmond, WA 98052
USA

EMail: nasko.oskov@microsoft.com