Transport Layer Security (TLS) Secure Renegotiation
<draft-mrex-tls-secure-renegotiation-01.txt>

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Copyright and License Notice

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the BSD License.
Abstract

A protocol design flaw in the TLS renegotiation handshake leaves all currently implemented protocol version of TLS (SSLv3 to TLSv1.2) vulnerable to Man-in-the-Middle (MitM) attacks where the attacker can establish a TLS session with a server, send crafted application data of his choice to the server and then proxy an unsuspecting client’s TLS handshake into the TLS renegotiation handshake of the server. Many applications on top of TLS see the data injected by the attacker and the data sent by the client as a single data stream and assume that an authentication during the TLS renegotiation handshake or contained in the client’s application data applies to the entire data stream received through the TLS-protected communication channel.

This document describes a protocol change for all protocol versions of TLS plus SSLv3 that will fix this vulnerability for all communication between updated TLS clients and updated TLS servers.

Table of Contents

1 Requirements Terminology ........................................ 3
2 Introduction ........................................................ 3
3 The TLS renegotiation vulnerability ............................. 3
   3.1 TLS handshake types ...................................... 3
   3.2 Attack scenarios ........................................... 4
4 The TLS renegotiation fix ......................................... 6
   4.1 Characteristics ............................................. 6
   4.2 Solution brief ............................................... 6
   4.3 Additional session state ................................... 7
   4.4 Reconnaissance ............................................. 8
   4.5 Secure renegotiation handshake message hash .............. 9
   4.6 Rationale ................................................... 10
5 Security Considerations .......................................... 11
6 IANA Considerations ............................................... 11
7 Acknowledgements ................................................ 11
8 References .................................................................. 12
   8.1 Normative References ....................................... 12
   8.2 Informative References ..................................... 12
Appendix A Installed Base Considerations ....................... 13
Appendix B Code example ............................................. 14
   B.1 Server-Side, modified handshake message hash .......... 14
Appendix C Implementation Notes .................................. 15
   C.1 Forward compatibility of SSLv3 and TLSv1.0 ............ 15
Author’s Address ..................................................... 16
1 Requirements Terminology

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].

2 Introduction

The TLS protocol provides communications security over the Internet and allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

TLS is the IETF’s successor to SSLv3 from Netscape. TLSv1.0 [RFC2246] was finalized in January 1999. It is widely deployed and used to protect a large variety of transport and application protocols, such as HTTP over TLS [RFC2818], WebDAV, CalDAV, SIP, IPP, IMAP/POP, SMTP, XMPP, BEEP and also SSL-VPNs.

Today you find TLS in various PC software, networking equipment, appliances, PDAs, SmartPhones and other small devices.

3 The TLS renegotiation vulnerability

All currently existing protocol version of TLS (SSLv3 to TLSv1.2) contain a security vulnerability in the design of the TLS renegotiation algorithm. The newly renegotiated TLS session is completely independent from the previous TLS session that it replaces. Applications using TLS to secure their communication often use TLS for channel authentication. They assume that an authentication performed at the TLS level or within application data coming through the TLS-protected channel is valid for all data received through this channel. The TLS protocol explicitly requires a TLS renegotiation to be mostly transparent to the application data stream. This opens a door to Man-in-the-Middle (MitM) attacks exploiting this weakness in the TLS renegotiation handshake.

3.1 TLS handshake types

The TLS and SSLv3 protocols specify only two types of handshakes (see TLSv1.2 [RFC5246] Section 7.3 Handshake Protocol Overview), a "full handshake" and an "abbreviated handshake" which is also referred to as "session resume".

The distinction "initial TLS handshake" and "TLS renegotiation handshake" is orthogonal to these handshake types.
An initial TLS handshake is the first TLS handshake on a communication channel, i.e. the handshake begins in the clear; the TLS record layer is initialized with the cipher suite TLS_NULL_WITH_NULL_NULL.

A TLS renegotiation handshake is a handshake that is started under the protection of an existing TLS session. With the exchange of the ChangeCipherSuite messages the existing TLS session is entirely replaced with the newly (re)negotiated TLS session.

3.2 Attack scenarios

There are three possible types of attack scenarios on TLS renegotiation:

1. Client’s initial TLS handshake is proxied by MitM into Server’s TLS renegotiation

2. Client’s renegotiation handshake is proxied by MitM into Server’s initial TLS handshake

3. Two independent TLS sessions Client<->MitM and MitM<->Server are spliced into one single TLS session Client<->Server through TLS renegotiation where the MitM proxies all communication

The MitM can only inject data into the initial TLS session where the MitM is an original TLS client or server. It is not possible to modify the actual handshake between TLS client and server without breaking the Finished verification. As soon as the ChangeCipherSpec messages are exchanged on the renegotiation handshake, the MitM can no longer inject or read application data exchanged by client and server. So the MitM is unable to read the server’s reply to the injected request(s) that the unsuspecting client is made to authenticate for.

It is impossible for the server to notice that it is being attacked in all three scenarios with the existing TLS protocol. Example exploits for type (1) scenarios have received the most attention so far, and are quite effective for protocols such as HTTP over TLS. When client certificates are used, type (3) attacks can be a problem. No attractive exploits have been described for type (2) scenarios yet, but it would be unwise to assume that they do not exist.

At the TLS/SSLv3 protocol level, all these renegotiations look perfectly OK. Server Endpoint Identification performed by clients (such as Section 3.1 in [RFC2818]) does not necessarily mitigate all of the attacks scenarios of type (2) and (3), where the renegotiation
will usually result in a change of the server identity at the TLS protocol level. The TLS protocol itself does not constrain changes in cryptographic properties and authenticated identities during a renegotiation.

A MitM attack usually leaves behind _two_ victims of the attack. The server is a victim of the attack, because it is made to perform a request issued by the attacker. But the client is also a victim, because the authentication performed by the unsuspecting client is re-purposed to authorize the request of the attacker.

You may notice that TLS clients in type (1) scenarios as well as TLS servers in type (2) scenarios perform only an initial TLS handshake, and they can still become a victim of an attack. This has serious consequences. It means that all TLS implementations, including those that have renegotiation disabled or not even implemented, are at risk from becoming a victim in a MitM attack on the TLS renegotiation vulnerability.
4 The TLS renegotiation fix

4.1 Characteristics

If a TLS client or server wants to be absolutely sure that it cannot become a victim of an attack based on the TLS renegotiation vulnerability, it (a) must be updated and (b) must discontinue talking TLS to peers that are not updated.

The latter is a pretty challenging requirement. The first one getting updated would suddenly have no one else to talk to. In the interest of continuous operation and interoperability with existing usage scenarios in the installed base, the vast majority is likely to embrace a different approach—at least for a transition period, where a lot of communication peers are not yet updated. Unpatched TLS server should have the old renegotiation disabled entirely. TLS clients, which have traditionally been quite trusting to TLS servers and requests for renegotiation, should become much more careful about unpatched TLS servers they handshake with.

This document provides a protocol fix for the TLS renegotiation vulnerability. It secures the TLS renegotiation between updated clients and updated servers. It allows updated clients and servers to determine whether their respective communication peer has also been updated. And it provides the highest possible level of interoperability with the installed base of old TLS communication peers, while still protecting communication between updated TLS peers from downgrade attacks; for a smooth transition period and beyond.

4.2 Solution brief

1. The verify_data from Finished messages of a TLS handshake are memorized in the connection state and will be added into the handshake message hash of the renegotiation handshake, thus authenticating the enclosing TLS session.

2. For Client to Server signaling, the special cipher suite ID TLS_RENEGPROTECTION_REQUEST is assigned and must be included in all ClientHello messages from updated TLS/SSLv3 clients.

3. For Server to Client signaling, the highest bit (0x80) of server_version.major is asserted, but _only_ for the network encoding of the ServerHello handshake message; ProtocolVersions everywhere else in TLS must remain original, including the record layer.
### 4.3 Additional session state

In order to implement secure TLS renegotiation, it is necessary to memorize additional TLS session state: the verify_data from the finished messages, a state variable "peer_is_updated" for the signaling, and optionally the state variable "allow_old_renego" when old renegotiation needs to be supported.

The length of the verify_data in the Finished messages differs between protocol versions of TLS and SSLv3:

- **TLSv1.0 & TLSv1.1:** 12 octets
- **TLSv1.2:** default 12 octets --but can be defined by cipher suite
- **SSLv3:** 36 octets --it is a concatenation of two elements "md5_hash" and "sha_hash"

The additional state that TLS client and servers have to memorize:

- (1a) plaintext verify_data of Client.Finished
- (1b) length of (1a)
- (2a) plaintext verify_data of Server.Finished
- (2b) length of (2a)
- (3) peer_is_updated /* only for handshake signaling */
- (4) allow_old_renego /* OPTIONAL, sticky session attribute */
  /* for interop with old renegotiation */

For new TLS handshakes on a communication channel (i.e. under TLS_NULL_WITH_NULL_NULL), the the values for (1a)(1b)(2a)(2b) are empty/initial, "peer_is_updated" is initialized to False. The the optional session state "allow_old_reneg"o is left unchanged when a session resume is performed, and initialized to the configuration parameter setting for support of old renegotiation when an initial full handshake is performed.

TLS servers and Clients MUST memorize the verify_data of the Finished messages if they implement renegotiation, so that this data can be used in a later renegotiation handshake to authenticate the enclosing TLS session. The easiest might be to memorize it when building their own Finished message and when processing the peer’s Finished message.

If TLS implementations want to offer support for old renegotiation, at least for the transition period, then they MUST offer separate configuration options for the TLS server and the TLS client side. TLS servers SHOULD NOT allow old renegotiation, TLS client MAY allow old renegotiation for a transition period, after which they SHOULD NOT allow old renegotiation.
4.4 Reconnaissance

On every new TLS handshake (including renegotiation), "peer_is_updated" in the connection state is initialized to False.

All updated TLS clients MUST include the special cipher suite ID TLS_RENEGO_PROTECTION_REQUEST (see Section 4.2) in the cipher_suites list of every ClientHello handshake message that they send. This applies to all updated TLS clients, including those that do not implement renegotiation or have it disabled (see Section 3.2 type (1) attacks). It is RECOMMENDED that this special cipher suite be listed first in ClientHello.cipher_suites list, but it MAY appear anywhere in the list. This special cipher suite ID does not represent a real cipher suite and should not be configurable by, and not made visible to, regular cipher suite configuration APIs and UIs.

An updated Server that receives a ClientHello must search the cipher_suites list for presence of the special cipher suite ID TLS_RENEGO_PROTECTION_REQUEST. When found, the server asserts the "peer_is_updated" flag in the connection state and it MUST set optional "allow_old_renego" state to False. The TLS server MUST NOT select the special cipher suite ID as the common cipher suite with the client.

When the Server has asserted "peer_is_updated", then it MUST signal back to the client that the server is updated by asserting the highest bit of server_version.major when composing the ServerHello handshake message for the client. This applies to _all_ updated TLS servers, including those that do not implement renegotiation or have it disabled (see Section 3.2 type (2) attacks).

Currently known values for ProtocolVersion fields in TLS:

<table>
<thead>
<tr>
<th>ProtocolVersion</th>
<th>SSLv3</th>
<th>TLSv1.0</th>
<th>TLSv1.1</th>
<th>TLSv1.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8 major</td>
<td>0x03</td>
<td>0x03</td>
<td>0x03</td>
<td>0x03</td>
</tr>
<tr>
<td>uint8 minor</td>
<td>0x00</td>
<td>0x01</td>
<td>0x02</td>
<td>0x03</td>
</tr>
</tbody>
</table>

Modified server_version when an updated TLS server is signaling back to an updated TLS client.

<table>
<thead>
<tr>
<th>modified server_version</th>
<th>SSLv3p</th>
<th>TLSv1.0p</th>
<th>TLSv1.1p</th>
<th>TLSv1.2p</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8 major</td>
<td>0x83</td>
<td>0x83</td>
<td>0x83</td>
<td>0x83</td>
</tr>
<tr>
<td>uint8 minor</td>
<td>0x00</td>
<td>0x01</td>
<td>0x02</td>
<td>0x03</td>
</tr>
</tbody>
</table>
The modified server_version.major value MUST be used for exactly those handshakes, where the server received the special cipher suite ID in ClientHello. It MUST NOT be sent otherwise. The modified server_version value MUST NOT be used elsewhere in the server code for protocol versions, and in particular, the modified server_version MUST NOT be used for the TLS record layer protocol version.

An updated TLS client that receives a ServerHello with a server_version.major that has the highest bit asserted, MUST assert the "peer_is_updated" flag and MUST set the optional "allow_old_renego" to False for the current session.

The TLS client MUST remove the highest bit from server_version.major before using the value as the TLS protocol version for the communication with the TLS server. The modified server_version MUST NOT be used for the record layer protocol version, and should not be made visible to code outside of the parser for the ServerHello handshake message.

4.5 Secure renegotiation handshake message hash

An updated TLS client performing a renegotiation handshake and receiving a ServerHello without the highest bit of server_version.major asserted ("peer_is_updated" is False), assumes an old server requesting old renegotiation. If the updated TLS client does not support old renegotiation, it MUST abort the handshake. If the updated TLS client supports old renegotiation and "allow_old_renego" is False for the enclosing TLS session, then it MUST abort the handshake. If "allow_old_renego" is still asserted in the enclosing TLS session, then the TLS client MAY proceed with old renegotiation.

An updated TLS server performing a renegotiation handshake having received a ClientHello without the special cipher suite TLS_RENEGOT_PROTECTION_REQUEST ("peer_is_updated" is False), assumes an old client requesting old renegotiation. If the server does not support old renegotiation, it MUST end the handshake sending the no renegotiation alert, and MAY abort the communication. If the server supports old renegotiation and "allow_old_renego" is False for the enclosing TLS session, then the server MUST end the handshake sending the no renegotiation alert and MAY abort the communication. If "allow_old_renego" is still asserted for the enclosing TLS session, then the TLS client MAY proceed with old renegotiation.

Otherwise, servers and clients MUST proceed with the secure TLS renegotiation handshake, which authenticates the enclosing TLS session in the following fashion:
TLS client and TLS server MUST individually add their memorized verify_data from the enclosing TLS session to the handshake message hash as "virtual" handshake messages. The verify_data MUST be added to the handshake message hash immediately following the ServerHello handshake message. The verify_data from the Client.Finished MUST be added first, the verify_data from the Server.Finished MUST be added second. There MUST NOT be any length fields included, only the verify_data (so for TLSv1.0-1.2 it will normally be 12 octets each).

The optional state "allow_old_renego" MUST BE transferred from the enclosing TLS session to the newly renegotiated session.

4.6 Rationale

The renegotiation vulnerability is removed by cryptographically binding the renegotiation handshake to the enclosing TLS session. This is accomplished by both sides adding the Finished.verify_data, which authenticated the enclosing TLS session, to the handshake message hash of the renegotiation handshake. The handshake authentication performed by the Finished message verification will fail if client and server do not share the exact same memories about the previous Finished messages, and thus protect renegotiation handshakes from MitM attacks. The same applies to the CertificateVerify signature verification in an optional client certificate authentication.

As discussed in Section 3.2, only communication between updated clients and updated servers can be reliably protected from type (1) and (2) attacks. Clients and servers need a bidirectional signaling scheme as part of the TLS handshake to determine whether the peer, they are handshaking with, is also updated.

The chosen signaling scheme is a compromise due to a non-negligible amount of intolerance of old servers to TLS extensions in the ClientHello handshake message. Various workarounds currently in use to remedy this interoperability problem (see [RFC5246] Appendix E) can not be simply ignored. The chosen signaling scheme works for extension-less SSLv3 ClientHello and even SSLv2 ClientHello on the initial TLS handshake. This enables secure renegotiation in all existing usage scenarios, including conservative clients and application-level reconnect fallbacks.
5 Security Considerations

This document describes a protocol change for all currently existing versions of the TLS protocol: TLSv1.2 {0x03,0x03} [RFC5246], TLSv1.1 {0x03,0x02} [RFC4346], TLSv1.0 {0x03,0x01} [RFC2246] and SSLv3 {0x03,0x00} [SSLv3] to fix a serious security vulnerability in the TLS renegotiation algorithm.

In the original SSLv3 and TLS protocol there is no difference between an initial TLS handshake and a TLS renegotiation handshake. Every pair of old TLS clients and servers of the installed base can potentially become a victim in a Man-in-the-Middle (MitM) attack through TLS renegotiation in one or more of the attack scenarios described in Section 3.2, provided that one of the two implements TLS renegotiation and can be coerced, lured, or simply asked to perform a TLS renegotiation.

Only TLS communication between updated clients and updated servers is reliably protected from the risk of attack.

6 IANA Considerations

IANA has assigned the following cipher suite value for the Client-to-Server signaling scheme, enabling clients to request that its handshake message are not to be used in old TLS renegotiations:

CipherSuite TLS_RENEGO_PROTECTION_REQUEST = {TBD,TBD}

7 Acknowledgements

The TLS renegotiation vulnerability was first discovered by Marsh Ray in August 2009. The MitM susceptibility of the TLS renegotiation was independently discovered by Martin Rex in November 2009 during discussions on the IETF TLS working group mailing list about channel bindings in TLS.

Many participants of the TLS working group provided valuable feedback and comments for improvement, to make the fix easy to implement and have a low risk of causing interoperability problems.

Special thanks to Michael D’Errico for implementer’s feedback, Stefan Santesson and his NroffEdit for helping me write this document, and Marsh Ray, Nicolas Williams and Nasko Oskov for elaborate discussions.
8 References

8.1 Normative References


NOTE to implementers: The protocol specifications of TLSv1.2, TLSv1.1 and TLSv1.0 are individually referenced. Please refer to the protocol specification on which your implementation is based when implementing the fix described in this document. There were a few backwards incompatible changes in the TLS protocol specifications that may not be sufficiently obvious to spot.

8.2 Informative References


Appendix A  Installed Base Considerations

Over the last 14 years SSLv3 and TLS have grown a huge installed base, but differing characteristics with respect to supported protocol versions, and varying levels of forward compatibility for newer protocol version numbers, and presence of TLS extensions [RFC4366] in the initial ClientHello handshake message.

Some of the installed base isn’t exactly new, some of it may already be out of maintenance, and some of it may be difficult to patch, let alone upgrade.

The production of software patches containing the security fix to the TLS and SSLv3 protocols, as described in this document, will probably take a few weeks. It will be followed by a transition period where the patches get individually deployed, resulting in a mix of updated and old TLS client and servers. Adoption speed will likely correspond to the number of interoperability risks and interoperability problems a fix creates for existing usage scenarios.

Implementers, software vendors and suppliers should be careful with providing the update/patch in a fashion that will adversely affect existing usage scenarios. Many consumers of the TLS and SSL technology may need a configuration option that lets them individually determine when to discontinue SSL/TLS-protected communication with unpatched TLS peers, for continued operation through the transition period.
Appendix B  Code example

B.1 Server-Side, modified handshake message hash

Here is an example, very loosely based on OpenSSL, what a code change could look like for the modified handshake message algorithm on renegotiation handshakes:

The final statement in the function ssl3_send_server_hello()
ssl/s3_srvr.c:ssl3_send_server_hello()

    return(ssl3_do_write(s,SSL3_RT_HANDSHAKE));

could be replaced with something like the following:

    ret = ssl3_do_write(s,SSL3_RT_HANDSHAKE);
    if ( ret>0 && s->s3->renegotiate )
    {
        if ( s->s3->new_renego_only_flag )
        {
            /* add previous verify_data of Client.Finished */
            /* to handshake msg hash */
            ssl3_finish_mac(s,s->s3->peer_finished_md,
                            s->s3->peer_finished_md_len);
            /* add previous verify_data of Server.Finished */
            /* to handshake msg hash */
            ssl3_finish_mac(s,s->s3->finished_md,
                            s->s3->finished_md_len);
        }
        else
        {
            /* Servers SHOULD NOT offer old renegotiation anymore */
            /* if (0==(s->options&SSL_OP_ALLOW_INSECURE_RENEGOTIATE))*/

            ssl3_send_alert(s,SSL3_AL_FATAL,SSL_AD_HANDSHAKE_FAILURE);
            ret = -1;
        }
    }
    return(ret);
Appendix C  Implementation Notes

C.1 Forward compatibility of SSLv3 and TLSv1.0

The evolvement of the TLS protocol is facing problems with the interoperability of newer protocol features with some part of the installed base of mainly server implementations of SSLv3 and TLSv1.0.

There are two areas of big concern, where minimal changes to the code might make a huge difference in terms of interoperability. These two issues are described in a little more detail in [RFC5246] Appendix E.

One problem is some servers (lack of) forward compatibility for extra data in the ClientHello handshake message (also called TLS extensions). The other is forward interoperability with TLS protocol version numbers other than SSLv3 (0x03,0x00) or TLSv1.0 (0x03,0x01) in ClientHello.client_version and the relation to protocol versions in other handshake messages (ServerHello, RSA Premaster Secret) and in the SSL/TLS record layer.

When updating SSLv3 or TLSv1.0 code for implementing his fix, it is highly advisable to also check these two issues.
Author’s Address

Martin Rex
SAP AG
EMail: mrex@sap.com