INTERNET-DRAFT
Updates (if approved): 5246, 4366, 4346, 2246
Intended Status: Standards Track
Expires: June 18, 2010

Martin Rex
(SAP AG)
Stefan Santesson
(3xA Security)

December 15, 2009

Transport Layer Security (TLS) Secure Renegotiation
<draft-mrex-tls-secure-renegotiation-04.txt>

Status of this Memo

Copyright and License Notice

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 and SSLv3 that will fix this vulnerability for all communication between updated TLS clients and updated TLS servers.

Table of Contents

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 the communication of many application protocols, such as HTTP over TLS [RFC2818], WebDAV, CalDAV, LDAP, SOAP, SIP, IPP, IMAP/POP, NNTP, SMTP, XMPP, BEEP and also SSL-VPNs.

Today you find SSL and its successor TLS not only in web servers and web browsers, but in many communication software for PCs, networking equipment, appliances, PDAs, SmartPhones and other small devices.

2.1 TLS handshake terminology

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 connection, 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 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 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 it 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, and are quite effective for protocols such as HTTP over TLS. When client certificates are used, type (3) attacks are also attractive. For type (2) scenarios, no attractive exploits have been described so far, but it would be unwise to assume that they do not exist.

At the TLS protocol level, all these renegotiations look perfectly OK. Server Endpoint Identification performed by clients (as in Section 3.1 of [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 can not 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. It provides a high level of interoperability with the installed base of old TLS communication peers, while protecting communication between updated TLS peers from downgrade attacks.

4.2 Solution brief

This solution applies equally to TLS and SSLv3. All further references to TLS without protocol version applies to SSLv3 as well.

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 value TLS_RENEGO_PROTECTION_REQUEST is assigned and must be included in all ClientHello handshake messages from updated clients.

3. For Server to Client signaling, a new TLS extension "renego_protection" is defined, that an updated Server must send back as a ServerHello extension whenever it finds the cipher suite value TLS_RENEGO_PROTECTION_REQUEST in ClientHello.

4.3 Additional connection and session state

In order to implement secure TLS renegotiation, it is necessary to memorize additional TLS connection state: the verify_data from the finished messages, a state variable "protection_available" for the signaling, and optionally a session 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 TLSv1.x 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)  protection_available  /*Boolean, for handshake signaling   */

(4)  allow_old_renego      /*Boolean, OPTIONAL session attribute*/
                          /*for renegotiation with old TLS peers*/
```

For every initial TLS handshake (see Section 2.1), the values for (1a)(1b)(2a)(2b) are empty/initial. The the optional session state "allow_old_renego" is left unchanged when a session resume is performed, and initialized with the setting of a system-wide or application-supplied value for support of old renegotiation (distinguishing client and server) for a full initial TLS handshake. "protection_available" is initialized to False for every TLS handshake.

TLS servers and clients that implement renegotiation MUST memorize the verify_data of Client and Server Finished messages, so this can be used in a later renegotiation handshake to authenticate the enclosing TLS session. These could be memorized when building one's 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 any system-wide configuration option(s) MUST distinguish between TLS server and 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 New protocol elements

This document defines a new cipher suite value

```
TLS_RENEGO_PROTECTION_REQUEST = { TBD, TBD }
```

to be used as for Client to Server signaling in ClientHello. This cipher suite value 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. TLS servers MUST NOT select this cipher suite value as the common cipher suite with the client.

This document also defines a new TLS extension "renego_protection"

```
enum {
    renego_protection(TBD), (65535)
} ExtensionType;
```

and contains _no_ extension_data (zero-length vector)

Implementations of SSLv3 and TLSv1.x, which do not implement TLS extensions, might use the following simplified approach to process the Server to Client signaling in ServerHello. Properly encoded, the above TLS extension is represented with the following static sequence of 6 octets as a single TLS extension in ServerHello after compression_method:

```
0x00  0x04  MSB  LSB  0x00  0x00
```

where (MSB*256)+LSB is equal to the extension type assigned by IANA.

Conforming servers that do not implement TLS extensions may add this static sequence of 6 octets into the ServerHello handshake message after compression_methods as a response to a ClientHello that includes TLS_RENEGO_PROTECTION_REQUEST. (this increases the length of the ServerHello handshake message from 70 to 76 octets, in case of a 32-octet session_id).

Conforming clients that do not implement TLS extensions will have to check whether the received ServerHello handshake message contains 6 additional octets after the compression_method and whether these match the above static 6-octet sequence representing the TLS extension "renego_protection".

4.5 Reconnaissance

All conforming TLS clients MUST include the cipher suite value TLS_RENEGO_PROTECTION_REQUEST in the cipher_suites list of _every_ ClientHello handshake message they send. This includes clients that do not implement renegotiation or have it disabled (see Section 3.1 type (1) attacks). This cipher suite value MAY appear anywhere in the cipher_suites list.

Conforming clients that compose a ClientHello handshake messages with other TLS extensions, MAY include the TLS extension "renego_protection" defined in 4.4.

When receiving a ClientHello that contains the cipher suite value TLS_RENEGO_PROTECTION_REQUEST, conforming servers MUST treat this exactly like the receipt of the TLS extension "protection_available" and MUST add this TLS extension into the (Extended)ServerHello reply to the client. This applies to full handshakes as well as session resume, and includes servers that do not implement renegotiation or have it disabled (see Section 3.1 type (2) attacks).

Such server behavior is an explicit exception to the prohibition of "unsolicited" ServerHello extensions in Section 7.4.1.4 [RFC5246] and Section 2.3 [RFC4366] and is only permitted when the client requests this TLS extension by including the TLS_RENEGO_PROTECTION_REQUEST cipher suite value in ClientHello. The special cipher suite value is a request for the renego_protection extension that can be combined with extension-less ClientHello and initial SSLv2 ClientHello that are still in use by conservative clients and for re-connect fallbacks of some web browsers for interoperability with old servers.

A conforming server which receives TLS_RENEGO_PROTECTION_REQUEST asserts the "protection_available" flag in the connection state and sets the optional "allow_old_renego" state to False.

A conforming client that receives an (Extended)ServerHello containing the "renego_protection" extension asserts the "protection_available" flag in the connection state and sets the optional "allow_old_renego" state to False for the current session.

4.6 Backwards interoperability with old peers

Conforming TLS clients receiving a ServerHello without the TLS extension "renego_protection" assume an old server. If the current handshake is a renegotiation for the TLS client, but "allow_old_renego" is False for the enclosing TLS session, then the client MUST abort the handshake with a fatal handshake failure alert.

Otherwise the client MAY proceed with the old handshake.

Conforming TLS servers receiving a ClientHello without the cipher suite value TLS_RENEGO_PROTECTION_REQUEST assume an old client. If the current handshake is a renegotiation for the TLS server, but "allow_old_renego" is False for the enclosing TLS session, then the server MUST abort the handshake with a fatal handshake failure alert. Otherwise the server MAY proceed with the old handshake.

4.7 Updated Handshake message hash calculation

For all TLS handshakes between updated clients and updated servers the following updated definition of the handshake message hash is used. This applies to the handshake message hash used in Client.Finished and Server.Finished and in the optional CertificateVerify handshake message.

The updated handshake message hash will ensure that initial and renegotiation handshakes are properly distinguished from each other and that renegotiation handshakes must authenticate the enclosing TLS session.

Conforming clients and servers, which have received the confirmation about renego protection availability from their peer, MUST add the following data directly to their handshake message hash function, immediately following the ServerHello handshake message:

on every initial TLS handshake with an updated peer:

```
4 static octets:    0x14 0x00 0x00 0x0b
```

on every TLS renegotiation handshake with an updated peer:

```
4 static octets:    0x14 0xff 0xff 0xff
verify_data from Client.Finished of enclosing TLS session
verify_data from Server.Finished of enclosing TLS session
```

This applies to full TLS handshake as well as TLS session resumes.

The verify_data from Client.Finished MUST be added before the verify_data from Server.Finished. There MUST NOT be any length fields included in verify_data, only the verify_data itself (so for TLSv1.0-1.2 it is 12 octets each, for SSLv3 36 octets each).

The optional state "allow_old_renego" must be transferred from the enclosing TLS session to the newly renegotiated session.

4.8 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 the optional client certificate authentication.

As discussed in Section 3.1, 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 [RFC5246], TLSv1.1 [RFC4346], TLSv1.0 [RFC2246] and SSLv3 [SSLv3] to fix a serious security vulnerability in the TLS renegotiation algorithm.

The original SSL and TLS protocol does not distinguish an initial TLS handshake from 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 3.1, 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.

Usually TLS renegotiation involves a full TLS handshake, so all of the security parameters and cryptographic properties are newly negotiated and authenticated, including the identities of the communication peers. Some applications actively use this feature, e.g. web servers deciding to request client certificates in a server-initiated TLS renegotiation after having seen the client's HTTP request. Some applications may be actively using the possibility to switch identities multiple times through TLS renegotiation for an existing connection.

To a number of applications, however, the change of a previously authenticated identity during TLS renegotiation comes as a surprise, and this may have serious security implications. Some TLS implementations will automatically perform certificate path validation, but the interpretation what a peer's certificate means is left entirely to the calling application ([RFC5246] Section 1. last paragraph).

It is RECOMMENDED that TLS implementations offer to applications the option to either disable renegotiation or to abort renegotiations when the remote peer tries to replace a previously authenticated certificate with a different one during renegotiation.

6 IANA Considerations

IANA has assigned the following TLS Cipher Suite value and the following TLS ExtensionType value for use with this specification (see Section 4.4):

TLS Cipher Suite	TLS_RENEGO_PROTECTION_REQUEST = { TBD, TBD }
TLS ExtensionType	renego_protection = { 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 channel bindings discussions on the IETF TLS WG mailing list.

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 continuous implementer's feedback, Marsh Ray, Nicolas Williams, Nasko Oskov, David-Sarah Hopwood and Eric Rescorla for elaborate discussions and input.

8 References

8.1 Normative References

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

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

[RFC4346] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006

[RFC2246] T. Dierks and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999

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

[SSLv3] Alan O. Freier, Philip Karlton, Paul C. Kocher, "The SSL Protocol Version 3.0", Internet Draft, November 1996, http://tools.ietf.org/html/draft-ietf-tls-ssl-version3-00

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

[RFC4366] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4466, April 2006

Appendix A Implementation Considerations

A.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 server implementations of SSLv3 and TLSv1.0 in the installed base.

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 (the extensibility used by 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 this fix, it is highly advisable to also check these two issues.

A.2 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 forward compatibility of protocol versions and TLS extension in the initial ClientHello handshake message.

Some of the installed base is quite old, some might be out of maintenance, and some will be difficult to patch, let alone upgrade.

The production of software patches with the security fix for TLS/SSLv3 described in this document 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 problems and risks each patch 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 will likely 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, updated handshake message hash

Here is an example, very loosely based on OpenSSL, for the server-side of the updated handshake message algorithm.

The final statement in 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 )
    {
    if (s->s3->protection_available)
       {
       if (NULL == s->enc_read_ctx)
          {
          /* add distinct prefix for initial handshake */
          ssl3_finished_mac(s,"\x14\x00\x00\x0b", 4);
          }
       else
          {
          /* add static prefix for renegotiation */
          ssl3_finished_mac(s,"\x14\xff\xff\xff", 4);
          /* add previous verify_data of Client.Finished */
          ssl3_finish_mac(s,s->s3->previous_client_finished,
                          s->s3->previous_client_finished_len);
          /* add previous verify_data of Server.Finished */
          ssl3_finish_mac(s,s->s3->previous_server_finished,
                          s->s3->previous_server_finished_len);
          }
       }
    else
       {
       if (NULL != s->enc_read_ctx)
          if (!s->session->allow_old_renego)
             {
             ssl3_send_alert(s,SSL3_AL_FATAL,
                             SSL_AD_HANDSHAKE_FAILURE);
             ret = -1;
             }
          }
       }
    }
 return(ret);
```

Author's Addresses

Martin Rex
SAP AG
EMail: mrex@sap.com

Stefan Santesson
3xA Security
EMail: sts@aaa-sec.com