Network Working Group
Internet-Draft
Expires: October 20, 2014

K. Bhargavan
A. Delignat-Lavaud
A. Pironti
Inria Paris-Rocquencourt
A. Langley
Google Inc.
M. Ray
Microsoft Corp.
April 18, 2014

Transport Layer Security (TLS) Resumption Indication Extension
draft-bhargavan-tls-resumption-indication-00

Abstract

When a TLS session is resumed via an abbreviated handshake, the knowledge of the master secret is used to implicitly mutually authenticate the two peers. However, an attacker can synchronize two different TLS sessions, so that they share the same master secret, breaking the resumption authentication property. This specification defines a TLS extension that cryptographically binds the resumption abbreviated handshake with its original session, thus preventing this attack.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on October 20, 2014.

Copyright Notice

Table of Contents

1. Introduction

In TLS [RFC5246], a session is established by a full handshake, and it can be resumed via abbreviated handshakes. Furthermore, several full or abbreviated hansdshakes can follow over the same connection. It is well known that, without the secure_renegotiation extension [RFC5746], handshakes performed over the same connection are not cryptographically bound: this means that an attacker can initiate a communication with a server, then ask for renegotiation and plug a connection originating from a victim client. The server will treat this as a renegotiation, while the victim client will believe it is the first handshake over the connection. The secure_renegotiation extension fixes this by cryptographically binding each handshake happening on a connection with the previous handshake that happened on the same connection. Technically, according to [RFC5746], the Client and Server Hello messages contain the client and server verify_data generated by the previous handshake in the same

connection: if these data do not match at the client and server side, then a renegotiation attack is detected, and the connection is aborted.

Complementary, an existing session can be resumed via an abbreviated handshake as the first handshake over a connection. In this case, one needs to make sure that the peers resuming the session are indeed the same as the ones who originated such session. In an abbreviated TLS handshake, this is achieved by proving the knowledge of the session master_secret, via the generation of the correct verify_data content (and its encryption within the Finished message).

However, especially with the RSA key exchange method, an attacker can easily synchronize two TLS sessions, so that they share the same master_secret [TRIPLE-HS]. Suppose a client, C, is connecting to an attacker, A. The attacker wishes to synchronize the client and a victim server, S, so that both have a session cached with a master secret and session ID that are known to the attacker.

1. C sends its "ClientHello.random" value to A.

2. A connects to S, using C's "ClientHello.random" value.

3. S responds to A, sending its "ServerHello.random", "ServerHello.session_id" and certificate chain.

4. A responds to C with its own certificates, but using the server's "ServerHello.random" and "ServerHello.session_id" values.

5. C proceeds with the key exchange, sending to A the "pre_master_secret" value, encrypted with A's public key.

6. A decrypts the "pre_master_secret", re-encrypts it with the server's public key and sends it on to S.

At this point, both sessions (between C and A, and between A and S) share the same "pre_master_secret", "ClientHello.random" and "ServerHello.random". Hence, the "master_secret" value will be equal for the two sessions and it will be associated both at C and S with the same session ID.

Note that the secure_renegotiation extension does not help in this case, because both client and server are resuming a session as their first handshake over the new connection, and hence the secure_renegotiation values (empty values in this case) will also match. Indeed, this resumption attack is dual to the renegotiation one, and as such requires a dual extension to fix the problem.

2. Requirements Notation

This document uses the same notation and terminology used in the TLS Protocol specification [RFC5246].

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. The TLS Session Hash

When a full handshake takes place, and thus a new TLS session is generated, implementations complying with this document MUST compute the "session_hash", as defined in [session-hash].

Additionally, the session_hash MUST be stored along with the other session data in the session database, or it MUST be included in the session ticket, where applicable.

4. The secure_resumption Extension

4.1. Overwiev

This specification introduces a new TLS extension, called "secure_resumption", that prevents the resumption attack described above. Basically, this extension cryptographically binds any abbreviated handshake with the original session the handshake is trying to resume. Technically, this is achieved by adding to the Client and Server Hello messages a "session_hash" associated to the session being resumed.

4.2. Extension definition

The "secure_resumption" extension has type TBD. The "extension data" field of this extension contains a "SecureResumption" structure:

```
struct {
  opaque secure_resumption<0..255>;
} SecureResumption;
```

The content of this extension is explained below, together with the different use case scenarios.

4.3. Client behavior: no resumption desired

When a client sends a Client Hello with empty session_id (and no session ticket), it means it has no session to resume and is only willing to establish a new session with the server. In this case,

the client MUST NOT send the secure_resumption extension in its Client Hello message.

With such a Client Hello message, the server will start a new session and, not seeing any secure_resumption extension, will not include it in its Server Hello message.

Servers receiving an invalid Client Hello message containing an empty ClientHello.session_id and a secure_resumption extension MUST NOT send the secure_resumption extension back in the Server Hello. Servers MAY abort the connection, or decide to continue ignoring the secure_resumption extension given by the client.

4.4. Client behavior: resumption desired

When a client wishes to resume a session, it fills the ClientHello.session_id (or sends a session ticket). In this case, a client implementing this specification MUST also send a secure_resumption extension, with SecureResumption.secure_resumption filled with the session_hash value of the session being resumed.

4.4.1. Server behavior: resumption rejected

If the server rejects the client request to resume a session, it provides a new ServerHello.session_id and proceeds with a full handshake. In this case, a server implementing this specification MUST NOT send a secure_resumption extension, and MUST ignore the value of the secure_resumption extension sent by the client.

Clients receiving an invalid ServerHello containing a new ServerHello.session_id value together with a secure_resumption extension MUST ignore the content of the server provided secure_resumption extension. Such clients MAY disconnect or continue with a full handshake.

4.4.2. Server behavior: resumption accepted

If the server accepts to resume the session it MUST check that the value contained in the ClientHello.secure_resumption extension matches the locally stored session_hash for the session being resumed.

If the check fails, the server MUST NOT continue with session resumption; instead the server MAY abort the connection or start a full handshake to generate a new session.

If the check succeeds, the server MAY continue with session resumption. In this case, the server MUST include a

ServerHello.secure_resumption extension, filled with the session_hash for the session being resumed.

4.4.2.1. Client behavior: resumption accepted

When the server accepts resumption, the client MUST check that a ServerHello.secure_resumption is present, and it MUST check that its content matches the locally stored session_hash for the session being resumed.

If the match fails, the client MUST abort the connection. (At this stage of the handshake, the client cannot ask anymore for a full handshake, and the server already committed to an abbreviated one, hence the only solution is to abort and re-start.)

If the match succeeds, the client continues with a normal abbreviated handshake.

5. Backward compatibility

5.1. Client not supporting secure_resumption

It is easy for servers to identify clients not supporting the secure_resumption extension: the ClientHello.session_id will be filled, but no secure_resumption extension will be present. In such cases, servers implementing this specification MUST refuse the resumption request and hence continue with a full handshake. Note that in practice, this disables resumption for all un-patched clients.

5.2. Server not supporting secure_resumption

With the current definition of the extension, a client gets to know whether a server supports or not the secure_resumption extension only after the server has already committed to an abbreviated handshake. If a client detects an un-patched server wishing to resume, it MUST abort the session with a handshake_failure fatal alert, and re-start a new connection proposing a full handshake.

6. Security Considerations

Without this extension, authentication over a resumed session is based only on the uniqueness of the master_sercret. However, an attacker can carefully craft two TLS sessions so that they share the same master_secret, breaking the authentication properties of TLS in case of resumed sessions.

This specification introduces a secure_resumption extension which cryptographically binds an abbreviated handshake to the session being resumed, by means of its session_hash. The session_hash value is unique to each session, as it depends on all the data exchanged to generate the session, including client and server randomness, their identities, and the choices of the pre_master_secret.

In principle, the Client and Server Finished.verify_data of the full handshake generating the session could be used instead of the session_hash, because both the verify_data and the session_hash depend on all the data that lead to the session context. However, the verify_data is typically very short (12 bytes for all currently defined cipher suites), and so collisions among verify_data of different sessions are relatively easy to find. In this document, by using the session_hash, the collision probability reduces to the collision resistance of the chosen hash algorithm (ciper suite-dependent for TLS 1.2, and concatenation of MD5 and SHA1 for all previous TLS versions and SSL 3.0).

7. References

7.1. 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.

[RFC5746] Rescorla, E., Ray, M., Dispensa, S., and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, February 2010.

7.2. Informative References

[TRIPLE-HS] Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Pironti, A., and P. Strub, "Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS", IEEE Symposium on Security and Privacy, to appear , 2014.

[session-hash] Bhargavan, K., Delignat-Lavaud, A., Langley, A., Pironti, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", Internet Draft , 2014.

Authors' Addresses

Karthikeyan Bhargavan
Inria Paris-Rocquencourt
23, Avenue d'Italie
Paris 75214 CEDEX 13
France

Email: karthikeyan.bhargavan@inria.fr

Antoine Delignat-Lavaud
Inria Paris-Rocquencourt
23, Avenue d'Italie
Paris 75214 CEDEX 13
France

Email: antoine.delignat-lavaud@inria.fr

Alfredo Pironti
Inria Paris-Rocquencourt
23, Avenue d'Italie
Paris 75214 CEDEX 13
France

Email: alfredo.pironti@inria.fr

Adam Langley
Google Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043
USA

Email: agl@google.com

Marsh Ray
Microsoft Corp.
1 Microsoft Way
Redmond, WA 98052
USA

Email: maray@microsoft.com