Network Working Group
Internet-Draft
Expires: May 16, 2015

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

November 12, 2014

Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension
draft-ietf-tls-session-hash-03

Abstract

The Transport Layer Security (TLS) master secret is not cryptographically bound to important session parameters. Consequently, it is possible for an active attacker to set up two sessions, one with a client and another with a server, such that the master secrets on the two sessions are the same. Thereafter, any mechanism that relies on the master secret for authentication, including session resumption, becomes vulnerable to a man-in-the-middle attack, where the attacker can simply forward messages back and forth between the client and server. This specification defines a TLS extension that contextually binds the master secret to a log of the full handshake that computes it, thus preventing such attacks.

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 May 16, 2015.

Table of Contents

1. Introduction

In TLS [RFC5246], every session has a "master_secret" computed as:

```
master_secret = PRF(pre_master_secret, "master secret",
                    ClientHello.random + ServerHello.random)
                    [0..47];
```

where the "pre_master_secret" is the result of some key exchange protocol. For example, when the handshake uses an RSA ciphersuite, this value is generated uniformly at random by the client, whereas

for DHE ciphersuites, it is the result of a Diffie-Hellman key agreement.

As described in [TRIPLE-HS], in both the RSA and DHE key exchanges, an active attacker can synchronize two TLS sessions so that they share the same "master_secret". For an RSA key exchange where the client is unauthenticated, this is achieved as follows. Suppose a client, C, connects to a malicious server, A. A then connects to a server, S, and completes both handshakes. For simplicity, assume that C and S only use RSA ciphersuites. (Note that C thinks it is connecting to A and is oblivious of S's involvement.)

1. C sends a "ClientHello" to A, and A forwards it to S.

2. S sends a "ServerHello" to A, and A forwards it to C.

3. S sends a "Certificate", containing its certificate chain, to A. A replaces it with its own certificate chain and sends it to C.

4. S sends a "ServerHelloDone" to A, and A forwards it to C.

5. C sends a "ClientKeyExchange" to A, containing the "pre_master_secret", encrypted with A's public key. A decrypts the "pre_master_secret", re-encrypts it with S's public key and sends it on to S.

6. C sends a "Finished" to A. A computes a "Finished" for its connection with S, and sends it to S.

7. S sends a "Finished" to A. A computes a "Finished" for its connection with C, and sends it to C.

At this point, both connections (between C and A, and between A and S) have new sessions that share the same "pre_master_secret", "ClientHello.random", "ServerHello.random", as well as other session parameters, including the session identifier and, optionally, the

session ticket. 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, even though the server identities on the two connections are different. Moreover, the record keys on the two connections will also be the same.

Similar scenarios can be achieved when the handshake uses a DHE ciphersuite, or an ECDHE ciphersuite with an arbitrary explicit curve. Even if the client or server does not prefer using RSA or DHE, the attacker can force them to use it by offering only RSA or DHE in its hello messages. Other key exchanges may also be vulnerable. If client authentication is used, the attack still

works, except that the two sessions now differ on both client and server identities.

Once A has synchronized the two connections, since the keys are the same on the two sides, it can step away and transparently forward messages between C and S, reading and modifying when it desires. In the key exchange literature, such occurrences are called unknown key-share attacks, since C and S share a secret but they both think that their secret is shared only with A. In themselves, these attacks do not break integrity or confidentiality between honest parties, but they offer a useful starting point from which to mount impersonation attacks on C and S.

Suppose C tries to resume its session on a new connection with A. A can then resume its session with S on a new connection and forward the abbreviated handshake messages unchanged between C and S. Since the abbreviated handshake only relies on the master secret for authentication, and does not mention client or server identities, both handshakes complete successfully, resulting in the same session keys and the same handshake log. A still knows the connection keys and can send messages to both C and S.

Critically, on the new connection, even the handshake log is the same on C and S, thus defeating any man-in-the-middle protection scheme that relies on the uniqueness of finished messages, such as the secure renegotiation indication extension [RFC5746] or TLS channel bindings [RFC5929]. [TRIPLE-HS] describes several exploits based on such session synchronization attacks. In particular, it describes a man-in-the-middle attack that circumvents the protections of

[RFC5746] to break client-authenticated TLS renegotiation after session resumption. Similar attacks apply to application-level authentication mechanisms that rely on channel bindings [RFC5929] or on key material exported from TLS [RFC5705].

The underlying protocol issue is that since the master secret is not guaranteed to be unique across sessions, it cannot be used on its own as an authentication credential. This specification introduces a TLS extension that computes the "master_secret" value from the log of the handshake that computes it, so that different handshakes will, by construction, create different master secrets.

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 TLS handshake takes place, we define

```
session_hash = Hash(handshake_messages)
```

where "handshake_messages" refers to all handshake messages sent or received, starting at the ClientHello up to and including the ClientKeyExchange message, including the type and length fields of the handshake messages. This is the concatenation of all the exchanged Handshake structures, as defined in Section 7.4 of [RFC5246].

For TLS 1.2, the "Hash" function is the one defined in Section 7.4.9 of [RFC5246] for the Finished message computation. For all previous versions of TLS, the "Hash" function computes the concatenation of MD5 and SHA1.

There is no "session_hash" for resumed handshakes, as they do not lead to the creation of a new session.

4. The Extended Master Secret

When the extended master secret extension is negotiated in a TLS session, the "master_secret" is computed as

```
master_secret = PRF(pre_master_secret, "extended master secret",
                    session_hash)
                    [0..47];
```

The extended master secret computation differs from the [RFC5246] in the following ways:

- The "extended master secret" label is used instead of "master secret";
- The "session_hash" is used instead of the "ClientHello.random" and "ServerHello.random".

The "session_hash" depends upon a handshake log that includes "ClientHello.random" and "ServerHello.random", in addition to ciphersuites, key exchange information and client and server certificates. Consequently, the extended master secret depends upon the choice of all these session parameters.

This design reflects the recommendation that keys should be bound to the security contexts that compute them [sp800-108]. The technique of mixing a hash of the key exchange messages into master key derivation is already used in other well-known protocols such as SSH [RFC4251].

Clients and servers SHOULD NOT resume sessions that do not use the extended master secret, especially if they rely on features like compound authentication that fall into the vulnerable cases described in Section 6.1.

5. Extension Negotiation

5.1. Extension Definition

This document defines a new TLS extension, "extended_master_secret" (with extension type 0x0017), which is used to signal both client and server to use the extended master secret computation. The "extension_data" field of this extension is empty. Thus, the entire encoding of the extension is 00 17 00 00.

If client and server agree on this extension and a full handshake takes place, both client and server MUST use the extended master secret derivation algorithm, as defined in Section 4.

If an abbreviated handshake takes place, the extension has no effect. The resumed session is protected by the extended master secret if the extension was negotiated in the full handshake that generated the session.

5.2. Client and Server Behavior

In its ClientHello message, a client implementing this document MUST send the "extended_master_secret" extension.

If a server implementing this document receives the "extended_master_secret" extension, it MUST include the "extended_master_secret" extension in its ServerHello message.

6. Security Considerations

6.1. Triple Handshake Preconditions and Impact

One way to mount a triple handshake attack has been described in Section 1, along with a mention of the security mechanisms that break due to the attack; more in-depth discussion and diagrams can be found in [TRIPLE-HS]. Here, some further discussion is presented about attack preconditions and impact.

To mount a triple handshake attack, it must be possible to force the same master secret on two different sessions. For this to happen, two preconditions must be met:

- The client, C, must be willing to connect to a malicious server, A. In certain contexts, like the web, this can be easily achieved, since a browser can be instructed to load content from an untrusted origin.

- The pre-master secret must be synchronized on the two sessions. This is particularly easy to achieve with the RSA key exchange, but arbitrary DH groups or ECDH curves can be exploited to this effect as well.

Once the master secret is synchronized on two sessions, any security property that relies on the uniqueness of the master secret is compromised. For example, a TLS exporter [RFC5705] no longer provides a unique key bound to the current session.

TLS session resumption also relies on the uniqueness of the master secret to authenticate the resuming peers. Hence, if a synchronized session is resumed, the peers cannot be sure about each other identity, and the attacker knows the connection keys. Clearly, a precondition to this step of the attack is that both client and server support session resumption (either via session identifier or session tickets [RFC5077]).

Additionally, in a synchronized abbreviated handshake, the whole transcript is synchronized, which includes the "verify_data" values. So, after an abbreviated handshake, channel bindings like "tls-unique" [RFC5929] will not identify uniquely the connection anymore.

Synchronization of the "verify_data" in abbreviated handshakes also undermines the security guarantees of the renegotiation indication extension [RFC5746], re-enabling a prefix-injection flaw similar to the renegotiation attack [Ray09]. However, in a triple handshake attack, the client sees the server certificate changing across different full handshakes. Hence, a precondition to mount this stage of the attack is that the client accepts different certificates at each handshake, even if their common names do not match. Before the triple handshake attack was discovered, this used to be widespread behavior, at least among some web browsers, that where hence vulnerable to the attack.

The extended master secret extension thwarts triple handshake attacks at their first stage, by ensuring that different sessions necessarily end up with different master secret values. Hence, all security properties relying on the uniqueness of the master secret are now

expected to hold. In particular, if a TLS session is protected by

the extended master secret extension, it is safe to resume it, to use its channel bindings, and to allow for certificate changes across renegotiation, meaning that all certificates are controlled by the same peer.

6.2. Cryptographic Properties of the Hash Function

The session hashes of two different sessions need to be distinct, hence the "Hash" function used to compute the "session_hash" needs to be collision resistant. As such, hash functions such as MD5 or SHA1 are NOT RECOMMENDED.

We observe that the "Hash" function used in the Finished message computation already needs to be collision resistant, for the renegotiation indication extension [RFC5746] to work: a collision on the verify_data (and hence on the hash function computing the handshake messages hash) defeats the renegotiation indication countermeasure.

As a matter of fact, all current ciphersuites defined for TLS 1.2 use SHA256 or better. For earlier versions of the protocol, only MD5 and SHA1 can be assumed to be supported, and this document does not require legacy implementations to add support for new hash functions.

6.3. Session Hash Handshake Coverage

The "session_hash" is designed to encompass all relevant session information, including ciphersuite negotiation, key exchange messages and client and server identities.

This document sets the "session_hash" to cover all handshake messages up to and including the ClientKeyExchange. In this way, on one hand, all the relevant session information is included; on the other hand, the master secret can be computed right after the ClientKeyExchange message, allowing implementations to shred the pre-master secret from memory as soon as possible.

It is crucial that any message sent after the ClientKeyExchange does not alter the session information. This is the case for the Finished messages, as well as for the client CertificateVerify in client-authenticated sessions. This also applies to session ticket messages [RFC5077]. Any protocol extension that adds protocol messages after the Client Key Exchange MUST either ensure that such messages do not alter the session information, or it MUST analyze the impact of the protocol changes with respect to the handshake coverage of the session hash.

6.4. No SSL 3.0 Support

SSL 3.0 [RFC6101] is a predecessor of the TLS protocol, and it is equally vulnerable to the triple handshake attacks.

The use of extensions precludes use of the extended master secret with SSL 3.0. Yet, this protocol uses encryption schemes and algorithms that are now considered weak. Furthermore, it seems likely that any system that did not upgrate from SSL 3.0 to any later version of TLS will be exposed to several other vulnerabilties anyway. As a consequence, this document does not provide workarounds to accommodate SSL 3.0.

7. IANA Considerations

IANA has added the extension code point 23 (0x0017), which has been used for prototype implementations, for the "extended_master_secret" extension to the TLS ExtensionType values registry as specified in TLS [RFC5246].

8. Acknowledgments

The triple handshake attacks were originally discovered by Antoine Delignat-Lavaud, Karthikeyan Bhargavan, and Alfredo Pironti, and were further developed by the miTLS team: Cedric Fournet, Pierre-Yves Strub, Markulf Kohlweiss, Santiago Zanella-Beguelin. Many of the ideas in this draft emerged from discussions with Martin Abadi, Ben Laurie, Nikos Mavrogiannopoulos, Manuel Pegourie-Gonnard, Eric Rescorla, Martin Rex, Brian Smith.

9. References

9.1. 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.

9.2. Informative References

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

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, March 2010.

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, July 2010.

[RFC4251] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, January 2008.

[RFC6101] Freier, A., Karlton, P., and P. Kocher, "The Secure Sockets Layer (SSL) Protocol Version 3.0", RFC 6101, August 2011.

[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, pages 98-113 , 2014.

[sp800-108] Chen, L., "NIST Special Publication 800-108: Recommendation for Key Derivation Using Pseudorandom Functions", Unpublished draft , 2009.

[Ray09] Ray, M., "Authentication Gap in TLS Renegotiation", 2009.

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