Internet Engineering Task Force (IETF)
Request for Comments: 7366
Category: Standards Track
ISSN: 2070-1721

P. Gutmann
University of Auckland

September 2014

Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)

Abstract

This document describes a means of negotiating the use of the encrypt-then-MAC security mechanism in place of the existing MAC-then-encrypt mechanism in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). The MAC-then-encrypt mechanism has been the subject of a number of security vulnerabilities over a period of many years.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7366.

Table of Contents

1. Introduction

TLS [2] and DTLS [4] use a MAC-then-encrypt construction that was regarded as secure at the time the original Secure Socket Layer (SSL) protocol was specified in the mid-1990s, but that is no longer regarded as secure [5] [6]. This construction, as used in TLS and later DTLS, has been the subject of numerous security vulnerabilities and attacks stretching over a period of many years. This document specifies a means of switching to the more secure encrypt-then-MAC construction as part of the TLS/DTLS handshake, replacing the current MAC-then-encrypt construction. (In this document, "MAC" refers to "Message Authentication Code".)

1.1. 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 [1].

2. Negotiating Encrypt-then-MAC

The use of encrypt-then-MAC is negotiated via TLS/DTLS extensions as defined in TLS [2]. On connecting, the client includes the encrypt_then_mac extension in its client_hello if it wishes to use encrypt-then-MAC rather than the default MAC-then-encrypt. If the server is capable of meeting this requirement, it responds with an encrypt_then_mac in its server_hello. The "extension_type" value for this extension SHALL be 22 (0x16), and the "extension_data" field of this extension SHALL be empty. The client and server MUST NOT use encrypt-then-MAC unless both sides have successfully exchanged encrypt_then_mac extensions.

2.1. Rationale

The use of TLS/DTLS extensions to negotiate an overall switch is preferable to defining new ciphersuites because the latter would result in a Cartesian explosion of suites, potentially requiring duplicating every single existing suite with a new one that uses encrypt-then-MAC. In contrast, the approach presented here requires just a single new extension type with a corresponding minimal-length extension sent by client and server.

Another possibility for introducing encrypt-then-MAC would be to make it part of TLS 1.3; however, this would require the implementation and deployment of all of TLS 1.2 just to support a trivial code change in the order of encryption and MAC'ing. In contrast, deploying encrypt-then-MAC via the TLS/DTLS extension mechanism required changing less than a dozen lines of code in one implementation (not including the handling for the new extension type, which was a further 50 or so lines of code).

The use of extensions precludes use with SSL 3.0, but then it's likely that anything still using that protocol, which is nearly two decades old, will be vulnerable to any number of other attacks anyway, so there seems little point in bending over backwards to accommodate SSL 3.0.

3. Applying Encrypt-then-MAC

Once the use of encrypt-then-MAC has been negotiated, processing of TLS/DTLS packets switches from the standard:

```
encrypt( data || MAC || pad )
```

to the new:

```
encrypt( data || pad ) || MAC
```

with the MAC covering the entire packet up to the start of the MAC value. In TLS [2] notation, the MAC calculation for TLS 1.0 without the explicit Initialization Vector (IV) is:

```
MAC(MAC_write_key, seq_num +
    TLSCipherText.type +
    TLSCipherText.version +
    TLSCipherText.length +
    ENC(content + padding + padding_length));
```

and for TLS 1.1 and greater with an explicit IV is:

```
MAC(MAC_write_key, seq_num +
    TLSCipherText.type +
    TLSCipherText.version +
    TLSCipherText.length +
    IV +
    ENC(content + padding + padding_length));
```

(For DTLS, the sequence number is replaced by the combined epoch and sequence number as per DTLS [4].) The final MAC value is then appended to the encrypted data and padding. This calculation is identical to the existing one, with the exception that the MAC calculation is run over the payload ciphertext (the TLSCipherText PDU) rather than the plaintext (the TLSCompressed PDU).

The overall TLS packet [2] is then:

```
struct {
       ContentType type;
       ProtocolVersion version;
       uint16 length;
       GenericBlockCipher fragment;
       opaque MAC;
       } TLSCiphertext;
```

The equivalent DTLS packet [4] is then:

```
struct {
       ContentType type;
       ProtocolVersion version;
       uint16 epoch;
       uint48 sequence_number;
       uint16 length;
       GenericBlockCipher fragment;
       opaque MAC;
       } TLSCiphertext;
```

This is identical to the existing TLS/DTLS layout, with the only difference being that the MAC value is moved outside the encrypted data.

Note from the GenericBlockCipher annotation that this only applies to standard block ciphers that have distinct encrypt and MAC operations. It does not apply to GenericStreamCiphers or to GenericAEADCiphers that already include integrity protection with the cipher. If a server receives an encrypt-then-MAC request extension from a client and then selects a stream or Authenticated Encryption with Associated

Data (AEAD) ciphersuite, it MUST NOT send an encrypt-then-MAC response extension back to the client.

Decryption reverses this processing. The MAC SHALL be evaluated before any further processing such as decryption is performed, and if the MAC verification fails, then processing SHALL terminate immediately. For TLS, a fatal bad_record_mac MUST be generated [2]. For DTLS, the record MUST be discarded, and a fatal bad_record_mac MAY be generated [4]. This immediate response to a bad MAC eliminates any timing channels that may be available through the use of manipulated packet data.

Some implementations may prefer to use a truncated MAC rather than a full-length one. In this case, they MAY negotiate the use of a truncated MAC through the TLS truncated_hmac extension as defined in TLS-Ext [3].

3.1. Rehandshake Issues

The status of encrypt-then-MAC vs. MAC-then-encrypt can potentially change during one or more rehandshakes. Implementations SHOULD retain the current session state across all rehandshakes for that session. (In other words, if the mechanism for the current session is X, then the renegotiated session should also use X.) Although implementations SHOULD NOT change the state during a rehandshake, if they wish to be more flexible, then the following rules apply:

Current Session	Renegotiated Session	Action to take
MAC-then-encrypt	MAC-then-encrypt	No change
MAC-then-encrypt	Encrypt-then-MAC	Upgrade to Encrypt-then-MAC
Encrypt-then-MAC	MAC-then-encrypt	Error
Encrypt-then-MAC	Encrypt-then-MAC	No change

Table 1: Encrypt-then-MAC with Renegotiation

As the above table points out, implementations MUST NOT renegotiate a downgrade from encrypt-then-MAC to MAC-then-encrypt. Note that a client or server that doesn't wish to implement the mechanism-change-during-rehandshake ability can (as a client) not request a mechanism change and (as a server) deny the mechanism change.

Note that these rules apply across potentially many rehandshakes. For example, if a session were in the encrypt-then-MAC state and a rehandshake selected a GenericAEADCiphers ciphersuite and a subsequent rehandshake then selected a MAC-then-encrypt ciphersuite, this would be an error since the renegotiation process has resulted in a downgrade from encrypt-then-MAC to MAC-then-encrypt (via the AEAD ciphersuite).

(As the text above has already pointed out, implementations SHOULD avoid having to deal with these ciphersuite calisthenics by retaining the initially negotiated mechanism across all rehandshakes.)

If an upgrade from MAC-then-encrypt to encrypt-then-MAC is negotiated as per the second line in the table above, then the change will take place in the first message that follows the Change Cipher Spec (CCS) message. In other words, all messages up to and including the CCS will use MAC-then-encrypt, and then the message that follows will continue with encrypt-then-MAC.

4. Security Considerations

This document defines encrypt-then-MAC, an improved security mechanism to replace the current MAC-then-encrypt one. Encrypt-then-MAC is regarded as more secure than the current mechanism [5] [6] and should mitigate or eliminate a number of attacks on the current mechanism, provided that the instructions on MAC processing given in Section 3 are applied.

An active attacker who can emulate a client or server with extension intolerance may cause some implementations to fall back to older protocol versions that don’t support extensions, which will in turn force a fallback to non-encrypt-then-MAC behaviour. A straightforward solution to this problem is to avoid fallback to older, less secure protocol versions. If fallback behaviour is unavoidable, then mechanisms to address this issue, which affects all capabilities that are negotiated via TLS extensions, are being developed by the TLS working group [7]. Anyone concerned about this type of attack should consult the TLS working group documents for guidance on appropriate defence mechanisms.

5. IANA Considerations

IANA has added the extension code point 22 (0x16) for the encrypt_then_mac extension to the TLS "ExtensionType Values" registry as specified in TLS [2].

6. Acknowledgements

The author would like to thank Martin Rex, Dan Shumow, and the members of the TLS mailing list for their feedback on this document.

7. References

7.1. Normative References

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

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

[3] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

[4] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

7.2. Informative References

[5] Bellare, M. and C. Namprempre, "Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm", Proceedings of AsiaCrypt ’00, Springer-Verlag LNCS No. 1976, p. 531, December 2000.

[6] Krawczyk, H., "The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?)", Proceedings of Crypto ’01, Springer-Verlag LNCS No. 2139, p. 310, August 2001.

[7] Moeller, B. and A. Langley, "TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks", Work in Progress, July 2014.

Author’s Address

Peter Gutmann
University of Auckland
Department of Computer Science
New Zealand

EMail: pgut001@cs.auckland.ac.nz