Network Working Group
Internet-Draft
Updates: 5246,6347 (if approved)
Intended status: Standards Track
Expires: May 21, 2014

N. Mavrogiannopoulos
Red Hat
A. Pironti
INRIA Paris-Rocquencourt

November 17, 2013

A new TLS record padding mechanism
draft-mavrogiannopoulos-new-tls-padding-01

Abstract

This memo proposes a new padding mechanism the TLS and DTLS record protocols. It defines a TLS extension to allow arbitrary amount of padding in any ciphersuite. The new padding mechanism eliminates any known padding oracle attacks on the CBC ciphersuites and allows novel length hiding techniques.

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 21, 2014.

Table of Contents

1. Introduction

The ciphersuites of the TLS protocol that utilize CBC block ciphers, [RFC5246][RFC6347] require the plaintext to be padded to a multiple of the cipher's block size prior to encryption. However, this padding, as implemented in TLS, has the following issues.

1. The plaintext is padded after it is authenticated, allowing for padding oracle attacks [CBCTIME][DTLS-ATTACK][LH-PADDING].

2. It is limited to 255 bytes.

3. No other than the CBC ciphersuites can take advantage of padding to hide the length of the records.

To overcome these limitations, the TLS extension proposed in this document enables a new padding mechanism for for both block and stream ciphers. The proposed extension eliminates padding oracles (both in errors and timing) that have been plaguing standard TLS block ciphers, without modifying critical aspects of the protocol such as the order of encryption and authentication.

2. Terminology

This document uses the same notation and terminology used in the TLS and DTLS protocol specifications [RFC5246][RFC6347].

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 [RFC2119].

3. TLS Extension: Extended Record Padding

The TLS extended record padding is a variant of the TLS record protocol where every record can be padded up to 2^14 bytes, regardless of the cipher being used.

3.1. Extension Negotiation

In order to indicate the support of the new record padding, clients MUST include an extension of type "extended_record_padding" to the extended client hello message. The "extended_record_padding" TLS extension is assigned the value of TDB-BY-IANA from the TLS ExtensionType registry. This value is used as the extension number for the extensions in both the client hello message and the server hello message. The hello extension mechanism is described in [RFC5246].

This extension carries no payload and indicates support for the extended record padding. The "extension_data" field of this extension are of zero length in both the client and the server.

The negotiated record padding applies for the duration of the session, including session resumption. A client wishing to resume a session where the extended record padding was negotiated SHOULD include the "extended_record_padding" extension in the client hello.

3.2. Record Payload

The translation of the TLSCompressed structure into TLSCiphertext remains the same as in [RFC5246]. When the cipher is BulkCipherAlgorithm.null, the 'fragment' structure of TLSCiphertext also remains unchanged. That is, for the TLS_NULL_WITH_NULL_NULL ciphersuite and for MAC-only ciphersuites this extension has no effect. For all other ciphersuites, the 'fragment' structure of TLSCiphertext is modified as follows.

```
stream-ciphered struct {
    opaque pad<0..2^14>;
    opaque content[TLSCompressed.length];
    opaque MAC[SecurityParameters.mac_length];
} GenericStreamCipher;
```

```
        struct {
            opaque IV[SecurityParameters.record_iv_length];
            block-ciphered ciphered struct {
                opaque pad<0..2^14>;
                opaque content[TLSCompressed.length];
                opaque MAC[CipherSpec.hash_size];
            };
        } GenericBlockCipher;

        struct {
            opaque nonce_explicit[SecurityParameters.record_iv_length];
            aead-ciphered struct {
                opaque pad<0..2^14>;
                opaque content[TLSCompressed.length];
            };
        } GenericAEADCipher;
```

The padding can be filled with arbitrary data, and it is authenticated as part of the MAC. For block ciphers, the length of the pad MUST be such that the total length (i.e., the pad, the content and the MAC) are a multiple of the block size.

Note: In typical applications that take no steps to hide the length of the record and are not using block ciphers, the size of the pad will be zero.

For the various ciphers the data are authenticated as follows.

Standard Stream Ciphers:

```
        MAC(MAC_write_key, seq_num +
            TLSCompressed.type +
            TLSCompressed.version +
            length +
            TLSCiphertext.fragment.GenericStreamCipher.pad +
            TLSCompressed.fragment);
```

Block Ciphers:

```
        MAC(MAC_write_key, seq_num +
            TLSCompressed.type +
            TLSCompressed.version +
            length +
            TLSCiphertext.fragment.GenericBlockCipher.pad +
            TLSCompressed.fragment);
```

AEAD Ciphers:

```
additional_data = seq_num + TLSCompressed.type +
                  TLSCompressed.version + length;

AEADEncrypted = AEAD-Encrypt(write_key, nonce,
                             pad + plaintext,
                             additional_data);
```

length
: For all the above cases, a uint16 containing the sum of the padding length and the content length.

Implementation note: With block and stream ciphers, in order to avoid padding oracles, decryption, MAC verification and payload decoding MUST be executed in the following order.

1. Decrypt TLSCiphertext.fragment.

2. Verify the MAC.

3. Split plaintext from pad.

4. Security Considerations

Since the padding is always included in the MAC computation, attacks that utilize the CBC-padding timing channel (e.g., [DTLS-ATTACK]) are not applicable.

In a way, the extended record padding can be seen as a special way of encoding application data before encryption (where application data given by the user are prefixed by some padding). Hence, previous security results on standard TLS block and stream ciphers still apply to the extended record padding.

5. IANA Considerations

This document defines a new TLS extension, "extended_record_padding", assigned a value of TBD-BY-IANA (the value 48015 is suggested) from the TLS ExtensionType registry defined in [RFC5246]. This value is used as the extension number for the extensions in both the client hello message and the server hello message.

6. Normative References

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

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

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

[DTLS-ATTACK] Nadhem, N. and K. Paterson, "Plaintext-recovery attacks against datagram TLS.", Network and Distributed System Security Symposium , 2012.

[LH-PADDING] Pironti, A., Strub, P., and K. Bhargavan, "Identifying Website Users by TLS Traffic Analysis: New Attacks and Effective Countermeasures.", INRIA Research Report 8067 , 2012.

[CBCTIME] Canvel, B., Hiltgen, A., Vaudenay, S., and M. Vuagnoux, "Password Interception in a SSL/TLS Channel", Advances in Cryptology -- CRYPTO , 2003.

Appendix A. Acknowledgements

The authors wish to thank Kenny Paterson for his suggestions on improving this document.

Authors' Addresses

Nikos Mavrogiannopoulos
Red Hat
Purkynova 99/75a
Brno 612 00
Czech Republic

Email: nmav@redhat.com

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

Email: alfredo.pironti@inria.fr