Internet Engineering Task Force
Internet-Draft
Expires: January 26, 2012

D. Balfanz, Ed.
D. Smetters
M. Upadhyay
A. Barth
Google Inc.

July 25, 2011

TLS Origin-Bound Certificates
draft-balfanz-tls-obc-00

Abstract

This document specifies a Transport Layer Security (TLS) extension and associated semantics that allow clients and servers to negotiate the use of origin-bound, self-signed certificates for TLS client authentication.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on January 26, 2012.

Copyright Notice

Table of Contents

1. Introduction

Transport Layer Security (TLS [RFC5246]) allows clients to authenticate themselves using Client Certificates. In practice, clients tend to ask for user consent before using Client Certificates. This is to allay privacy concerns about user-identifying information in the Client Certificate, and also to let the user choose among possibly many certificates that can be used by the client for the TLS session.

The user experience of obtaining this consent, along with that of obtaining the certificates in the first place, has traditionally presented a hurdle to user adoption. Additionally, operational constraints on the server side can make it difficult for service providers to switch from a cookie-based authentication scheme to certificate-based TLS client authentication.

The TLS Origin-Bound Certificates extension (TLS-OBC) is a TLS extension [RFC6066] that allows clients to use certificate-based client authentication without having to obtain user consent before using certificates. A client creates at most one (self-signed) certificate of any given type per web origin [WebOrigin], and does not include user-identifying information into such _origin-bound certificates_, thus making user consent and user-assisted certificate selection unnecessary.

1.1. Requirements Language

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].

1.2. Extension Dependencies

The client SHOULD use TLS-OBC during a TLS session only if it also uses the TLS-MessageReordering [CertMessageReordering] extension during that TLS session.

2. Origin-Bound Certificates

An origin-bound certificate is a self-issued certificate that the client uses for TLS client authentication for a particular web origin [WebOrigin]. Origin-bound certificates MUST be self-signed, i.e., a private key corresponding to the certified public key MUST be used to sign the certificate.

Clients MUST NOT re-use the same origin-bound certificate for more

than one web origin.

2.1. Certificate Creation

Clients create origin-bound certificates when asked to perform TLS client authentication after negotiating the "ob_cert" extension with a server from that origin (see Section 3.5), and they do not already have a valid origin-bound certificate for that origin available for use.

Clients SHOULD NOT re-use the same key pair for more than one origin-bound certificate. To do so would violate the privacy properties required by origin-bound certificates.

When creating an origin-bound certificate, it is RECOMMENDED that clients use the PrintableString representation of "anonymous.invalid" as the common name component of the Distinguished Name of both the certificate's Issuer and Subject, with no other name components present.

It is RECOMMENDED that clients pick a lifetime for the new certificate between one month and one year (perhaps depending on an assessment of how well the private key is protected on the host platform). The client SHALL use the notBefore and notAfter fields in the new certificate to record the lifetime of the new certificate.

2.1.1. Origin-Bound Certificate Extension

Origin-bound certificates MUST be X.509 v3 certificates, and MUST include the origin-bound certificate extension (OID 1.3.6.1.4.111129.2.1.6), shown below. This extension MUST be marked critical.

The data of the extension will be the ASCII Serialization [WebOrigin] of the certificate's web origin [WebOrigin] as an IA5String.

```
id-ce-originBoundCertificate OBJECT IDENTIFIER ::=
        { enterprises Google 2 1 6 }

OriginBoundCertificate ::=
        IA5String
```

Figure 1

2.2. Certificate and Key Rotation

After a client has created an origin-bound certificate for a certain web origin, the client SHOULD re-use the certificate for a period of time, and then discard it.

The client MUST discard existing origin-bound certificate at or before the notAfter time indicated in each certificate.

The client MAY opt to discard an existing origin-bound certificate at any time of its choosing prior to the notAfter time indicated in the certificate. In particular, the client may delete an existing origin-bound certificate in response to a user request to clear privacy-sensitive state from their user-agent, ensuring that a fresh certificate will be generated the next time a user visits that origin.

The client SHOULD use a new key pair when it generates a new origin-bound certificate.

3. Changes to The Handshake Message Contents

3.1. Extension Definition

This document defines a new TLS extension, "ob_cert" (with tentative extension type 0xff0f), which indicates that client and server agree to allow the client to use an origin-bound certificate to authenticate itself to the server.

To indicate support for origin-bound certificates, the client includes an extension of type "ob_cert" in the Extended Client Hello message. To request client authentication with an origin-bound certificate, the server includes the same extension in Extended Server Hello message.

The "ob_cert" TLS extension will be assigned a value from the TLS ExtensionType registry, and will contain zero length "extension_data". For testing, we will use the extension type value of 0xff0f. For this type value, the entire encoding of the extension will be ff 0f 00 00.

3.2. Client Hello

A client wishing to indicate support for origin-bound client certificates MUST include the "ob_cert" in the extended Client Hello message to a server from a particular web origin.

3.3. Server Hello

A server that receives a Client Hello message containing the "ob_cert" extension MAY choose to include the same extension in the extended Server Hello message. If it does, the client and server are considered to have negotiated origin-bound client certificates for the session.

3.4. Certificate Request

A server MAY choose to send a Certificate Request message to the client. If the session uses origin-bound client certificates, the server MAY use an empty "certificate_authorities" list.

A client that receives a Certificate Request during a session for which origin-bound client certificates were negotiated MUST ignore the "certificate_authorities" list.

3.5. Client Certificate

TLS [RFC5246] requires that a client that receives a Certificate Request message must send a Client Certificate message in response (which may or may not include client certificates). If client and server negotiated origin-bound client certificates, then the client SHOULD include a certificate in the Client Certificate message. If the "certificate_authorities" list in the Certificate Request is empty, that certificate SHOULD be an origin-bound certificate, and it should be selected by the client without any assistance or approval by the end-user. If the "certificate_authorities" list in the Certificate Request is not empty, the client MAY select a regular certificate for inclusion in the Client Certificate message, and MAY involve the end-user in the certificate selection process.

If a client includes an origin-bound certificate in the Client Certificate message during a session for which origin-bound client certificates were negotiated, the included certificate MUST

- be an origin-bound certificate for the web origin of the server, and
- match a certificate type requested by the server in the Certificate Request message.

If a client doesn’t possess a certificate meeting the above requirements, it SHOULD self-issue a certificate of a type requested by the server in the Certificate Request message and include the newly-generated certificate in the Client Certificate message. Only if the client is not capable of providing a certificate of the

requested type SHOULD it include an empty "certificate_list" in the Client Certificate message.

A server verifying a Client Certificate message in a handshake that negotiated origin-bound client certificates MUST verify that the certificate is self-signed, rather than being signed by any particular CA. The server MUST verify that the public key in the certificate corresponds to the key used to authenticate the client in the handshake. The server SHOULD ignore the Issuer and Subject in the presented certificate. The server MUST ignore the notBefore and notAfter fields in the certificate.

Finally, the server MUST verify that the certificate contains the origin-bound extension, and SHOULD verify that the origin identified in that certificate matches the server. In particular, the scheme MUST be https. The port MUST be the server port to which the client connected for the TLS connection, and the host name MUST be a valid host name for the server. If the client uses SNI, then the hostname in the origin-bound certificate must match the hostname specified in the SNI extension.

4. Security Considerations

4.1. Third-Party Tracking

A third party might be able to track a client across multiple sessions by observing the use of the same origin-bound certificate. To alleviate this concern, clients SHOULD also implement the CertificateMessage Reordering Extension [CertMessageReordering], and use it concurrently with TLS-OBC. The CertificateMessage Reordering Extension [CertMessageReordering]ensures that the origin-bound certificate is sent after TLS established secrecy on the channel between client and server.

4.2. Server Tracking

A client can prevent server tacking by deleting the origin-bound certificate for the server’s Web origin. This could happen, for example, when the user elects to remove all cookies for that origin.

4.3. Cookie Hardening

One way TLS-OBC can be used to strengthen cookie-based authentication is by "binding" cookies to an origin-bound certificate. The server, when issuing a cookie for an HTTP session, would associate the client’s origin-bound certificate with the session (either by encoding information about the certificate unforgeably in the

cookie), or by associating the certificate with the cookie's session through some other means. That way, if and when a cookie gets stolen from a client, it cannot be used over a different TLS connection - the cookie thief would also have to steal the private key associated with the client's origin-bound certificate, a task considerably harder especially when we assume the existence of a Trusted Platform Module or other Secure Element that can store the origin-bound-certificate's private key.

5. References

[CertMessageReordering]
"TODO: Reference for ClientCertificate message reordering", <http://example.com>.

[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.

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

[WebOrigin]
Barth, A., "The Web Origin Concept", <http://tools.ietf.org/html/draft-abarth-origin-09>.

Authors' Addresses

Dirk Balfanz (editor)
Google Inc.
1600 Ampitheatre Parkway
Mountain View, CA
USA

Phone:
Email: balfanz@google.com

D K Smetters
Google Inc.

Mayank Upadhyay
Google Inc.

Adam Barth
Google Inc.

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 12, 2012

R. Seggelmann
M. Tuexen
Muenster Univ. of Appl. Sciences
M. Williams

July 11, 2011

Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension
draft-ietf-tls-dtls-heartbeat-02.txt

Abstract

This document describes the Heartbeat Extension for the Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) protocol.

The Heartbeat Extension provides a new protocol for TLS/DTLS allowing the usage of keep-alive functionality without performing a renegotiation and a basis for path maximum transmission unit (PMTU) discovery for DTLS.

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 January 12, 2012.

Table of Contents

1. Introduction

1.1. Overview

This document describes the Heartbeat Extension for the Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) protocols, as defined in [RFC5246] and [RFC4347] and their adoptions to specific transport protocol as described in [RFC3436], [RFC5238], and [RFC6083].

DTLS is designed to secure traffic running on top of unreliable transport protocols. Usually such protocols have no session management. The only mechanism available at the DTLS layer to figure out if a peer is still alive is performing a costly renegotiation. If the application uses unidirectional traffic there is no other way. Furthermore, DTLS needs to perform path maximum transmission unit (PMTU) discovery but has no specific message type to realize it without affecting user message transfer.

TLS is based on reliable protocols but there is not necessarily a feature available to keep the connection alive without continuous data transfer.

The Heartbeat Extension as described in this document overcomes these limitations. The user can use the new HeartbeatRequest message which has to be answered by the peer with a HeartbeartResponse immediately. To perform PMTU discovery, HeartbeatRequest messages containing padding can be used as probe packets as described in [RFC4821].

1.2. Conventions

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].

2. Heartbeat Hello Extension

The support of Heartbeats is indicated with Hello Extensions. A peer can not only indicate that its implementation supports Heartbeats, it can also choose whether it is willing to receive HeartbeatRequest messages and respond with HeartbeatResponse messages or only to send HeartbeatRequest messages. The former is indicated by using peer_allowed_to_send as the HeartbeatMode, the latter is indicated by using peer_not_allowed_to_send as the Heartbeat mode. This decision can be changed with every renegotiation. HeartbeatRequest messages MUST NOT be sent to a peer indicating peer_not_allowed_to_send. If an endpoint has indicated peer_not_allowed_to_send and receives a

HeartbeatRequest message SHOULD drop the message silently and MAY send an unexpected_message Alert message.

The format of the Heartbeat Hello Extension is defined by:

```
enum {
   peer_allowed_to_send(1),
   peer_not_allowed_to_send(2),
   (255)
} HeartbeatMode;

struct {
   HeartbeatMode mode;
} HeartbeatExtension;
```

Upon reception of an unknown mode, an error Alert message using illegal_parameter as its AlertDescription MUST be sent in response.

3. Heartbeat Protocol

The Heartbeat protocol is a new protocol on top of the Record Layer. The protocol itself consists of two message types: HeartbeatRequest and HeartbeatResponse.

```
enum {
   heartbeat_request(1),
   heartbeat_response(2),
   (255)
} HeartbeatMessageType;
```

Like the ChangeCipherSpec message, a HeartbeatRequest message can arrive at any time during the lifetime of a connection. Whenever a HeartbeatRequest message is received, it has to be answered with a corresponding HeartbeatResponse message immediately.

However, a HeartbeatRequest message SHOULD NOT be sent during handshakes. If a handshake is initiated while a HeartbeatRequest is still in flight, the sending peer MUST stop the retransmission timer for it. The receiving peer SHOULD discard it silently, if it arrives during or after the handshake. HeartbeatRequest messages from older epochs SHOULD be discarded.

There MUST NOT be more than one HeartbeatRequest message in flight at a time. A HeartbeatRequest message is considered to be in flight until the corresponding HeartbeatResponse message is received, or until the retransmit timer expires.

When using an unreliable transport protocol like DCCP or UDP, HeartbeatRequest messages MUST be retransmitted using the simple timeout and retransmission scheme DTLS uses for flights as described in Section 4.2.4 of [RFC4347]. In particular, after a number of retransmissions without receiving a corresponding HeartbeatResponse message having the expected payload the DTLS connection SHOULD be terminated. The threshold used for this SHOULD be the same as for DTLS handshake messages. Please note, that after the timer supervising a HeartbeatRequest messages expires, this message is no longer considered in flight. Therefore the HeartbeatRequest message is eligible for retransmission. The retransmission scheme in combination with the restriction that only one HeartbeatRequest is allowed to be in flight ensures that the congestion control is handled appropriately in case of the transport protocol not providing one, like in the case of DTLS over UDP.

When using a reliable transport protocol like SCTP or TCP, HeartbeatRequest messages only need to be sent once. The transport layer will handle retransmissions. If no corresponding HeartbeatResponse message has been received after a user configured amount of time, the DTLS/TLS connection SHOULD be terminated.

4. Heartbeat Request and Response Messages

The Heartbeat protocol messages consist of their type and an arbitrary payload and padding.

```
struct {
   HeartbeatMessageType type;
   uint16 payload_length;
   opaque payload[HeartbeatMessage.payload_length];
   opaque padding[padding_length];
} HeartbeatMessage;
```

The length of a HeartbeatMessage in total MUST NOT exceed 2^14 or max_fragment_length when negotiated as defined in [RFC6066].

type: The message type, either heartbeat_request or heartbeat_response.

payload_length: The length of the payload.

payload: The payload consists of arbitrary content.

padding: The padding is additional arbitrary content which MUST be ignored by the receiver. The padding_length is TLSPlaintext.length - payload_length - 3 with TLS and DTLSPlaintext.length - payload_length - 3 with DTLS.

When a HeartbeatRequest message is received, a corresponding HeartbeatResponse message MUST be sent carrying an exact copy of the payload of the HeartbeatRequest. The padding of the received HeartbeatRequest message MUST be ignored. It MUST NOT be included in the HeartbeatResponse message, i.e. the padding field of the HeartbeatResponse message MUST have a length of zero.

If a received HeartbeatResponse message does not contain the expected payload the message MUST be discarded silently. If it does contain the expected payload the retransmission timer MUST be stopped.

If payload_length is either shorter than expected and thus indicates padding in a HeartbeatResponse or exceeds the actual message length in any message type, an error Alert message using illegal_parameter as its AlertDescription MUST be sent in response.

5. Use Cases

5.1. Path MTU Discovery

DTLS performs path MTU discovery as described in Section 4.1.1.1 of [RFC4347]. A detailed description how to perform path MTU discovery is given in [RFC4821]. The necessary probe packets are the HeartbeatRequest messages.

This method using HeartbeatRequest messages for DTLS is similar to the one for the Stream Control Transmission Protocol (SCTP) using the padding chunk (PAD-chunk) defined in [RFC4820].

5.2. Liveliness check

Sending HeartbeatRequest messages allows the sender to make sure that it can reach the peer and the peer is alive. Even in case of TLS/TCP this allows this check at a much higher rate than the TCP keepalive feature would allow.

Besides making sure that the peer is still reachable, sending HeartbeatRequest messages refreshes the NAT state of all involved NATs.

HeartbeatRequest messages SHOULD only be sent after an idle period that is at least multiple round trip times long.

6. IANA Considerations

[NOTE to RFC-Editor:

"RFCXXXX" is to be replaced by the RFC number you assign this document.

]

IANA needs to assign the heartbeat content type (value TBD) from the TLS ContentType Registry as specified in [RFC5246]. The reference should be RFCXXXX.

IANA needs to maintain a new registry for Heartbeat Message Types. The message types are numbers in the range from 0 to 255 (decimal). Initially IANA needs to assign the heartbeat_request (suggested value 1) and the heartbeat_response (suggested value 2) message type. The values 0 and 255 should be reserved. This registry uses the Specification Required policy as described in [RFC5226]. The reference should be RFCXXXX.

IANA needs to assign the heartbeat extension type (value TBD) from the TLS Extension Type Registry as specified in [RFC5246]. The reference should be RFCXXXX.

IANA needs to maintain a new registry for Heartbeat Modes. The modes are numbers in the range from 0 to 255 (decimal). Initially IANA needs to assign the peer_allowed_to_send (suggested value 1) and the peer_not_allowed_to_send (suggested value 2) modes. The values 0 and 255 should be reserved. This registry uses the Specification Required policy as described in [RFC5226]. The reference should be RFCXXXX.

7. Security Considerations

This document does not add any additional security considerations in addition to the ones given in [RFC4347] and [RFC5246].

8. Acknowledgments

The authors wish to thank Pasi Eronen, Adam Langley, Tom Petch, Eric Rescorla, Peter Saint-Andre, and Juho Vaehae-Herttua for their invaluable comments.

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.

[RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

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

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

9.2. Informative References

[RFC3436] Jungmaier, A., Rescorla, E., and M. Tuexen, "Transport Layer Security over Stream Control Transmission Protocol", RFC 3436, December 2002.

[RFC4820] Tuexen, M., Stewart, R., and P. Lei, "Padding Chunk and Parameter for the Stream Control Transmission Protocol (SCTP)", RFC 4820, March 2007.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.

[RFC5238] Phelan, T., "Datagram Transport Layer Security (DTLS) over the Datagram Congestion Control Protocol (DCCP)", RFC 5238, May 2008.

[RFC6083] Tuexen, M., Seggelmann, R., and E. Rescorla, "Datagram Transport Layer Security (DTLS) for Stream Control Transmission Protocol (SCTP)", RFC 6083, January 2011.

Authors' Addresses

Robin Seggelmann
Muenster University of Applied Sciences
Stegerwaldstr. 39
48565 Steinfurt
DE

Email: seggelmann@fh-muenster.de

Michael Tuexen
Muenster University of Applied Sciences
Stegerwaldstr. 39
48565 Steinfurt
DE

Email: tuexen@fh-muenster.de

Michael Williams

Email: michael.glenn.williams@gmail.com

IETF
Internet-Draft
Intended status: Standards Track
Expires: January 5, 2012

P. Wouters
Xelerance
J. Gilmore

S. Weiler
SPARTA, Inc.
July 4, 2011

TLS Extension for out-of-band public key validation
draft-wouters-tls-oob-pubkey-00

Abstract

This document specifies a new TLS extension as well as modified TLS client and TLS server behaviour when public keys are authenticated out-of-band to the current TLS connection. It is a companion document for RFC 5246, "The Transport Layer Security (TLS) Protocol Version 1.2". The new extension specified is "oob_pubkey_list" which can be used when the TLS client is already in possession of a validated public key of the TLS server before it starts the TLS handshake.

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 January 5, 2012.

Copyright Notice

Table of Contents

1. Introduction

1.1. Motivation

Traditionally, TLS server public keys are obtained in PKIX containers in-band using the TLS connection and validated using trust anchors based on a [PKIX] certification authority (CA). This method can add a complicated trust relationship that is difficult to validate. Examples of such complexity can be seen in [Defeating-SSL].

Alternative methods are available that allow a TLS client to obtain the TLS server public key:

- The TLS server public key is obtained from a [PKIX] certificate chain from an [LDAP] server
- The TLS server public key is obtained from a DNSSEC secured RRset using [DANE]
- The TLS server public key is provisioned by the operating system and updated via software updates
- A TLS client has connected to the TLS server before and has cached the TLS server certificate chain or TLS server public key for re-use

[RFC5246] does not provide a mechanism for a TLS client to tell the TLS server it is already in possession of the authenticated public key. Therefore, a TLS server must always send a list of trusted CA keys and its EE certificate containing its public key, even when the TLS client does not require or desire that data for authentication.

[RFC6066] allows suppression of the certificate trust anchor chain, but not suppression of the PKIX EE certificate container. These certificate chains are large opague blocks of data containing much more then the public key of the TLS server. Since the TLS client might only be able to validate the PKIX SubjectPublicKeyInfo via an out-of-band method, it has to explicitly forget any additional information received that was sent by the server that it could not validate. Furthermore, information that comes in via these certificate chains could contain contradicting or additional information that the TLS client cannot validate or trust, such as an expiry date that conflicts with information obtained from DNS or LDAP. This document specifies a method to suppress sending this additional information.

1.2. Applicability

The Transport Layer Security (TLS) Protocol Version 1.2 is specified in RFC 5246 [RFC5246]. RFC 5246 also provides a framework for extensions to TLS as well as considerations for designing such extensions. RFC 6066 [RFC6066] defines several new TLS extensions. This document extends the specifications of those RFCs with one new TLS extension to facilitate suppressing unneeded [PKIX] information from being sent during the TLS handshake when this information is not required to authenticate the TLS server.

1.3. Terminology

Most security-related terms in this document are to be understood in the sense defined in [SECTERMS]; such terms include, but are not limited to, "attack", "authentication", "authorization", "certification authority", "certification path", "certificate", "credential", "identity", "self-signed certificate", "trust", "trust anchor", "trust chain", "validate", and "verify".

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].

1.4. Specific Extension Covered

The extension described here describes a method for the TLS client to instruct the TLS server to omit sending the PKIX End Entity certificate and trusted root CA certificates.

The extension type is defined in this document are:

```
enum {
  oob_pubkey_list([TBD]) (65535)
} ExtensionType;
```

Specifically, the extension described in this document allows a TLS client to indicate to TLS servers that it already has a trusted copy of one or more of the TLS server’s public keys. Since the TLS server can have multiple public keys for a TLS connection, the TLS client sends the TLS server a set of public keys or hashes of public keys. The TLS server responds with sending the oob_pubkey_list back containing the preferred public key or hash thereof, selected from the client’s proposed list.

TLS clients and TLS servers may use the extension described in this document. The extension is designed to be backwards compatible,

meaning that TLS clients that support the extension can talk to TLS servers that do not support the extension, and vice versa.

Note that any messages associated with this extension that are sent during the TLS handshake MUST be included in the hash calculations involved in "Finished" messages.

Note also that the extension defined in this document is relevant only when a session is initiated. A client that requests session resumption does not in general know whether the server will accept this request, and therefore it SHOULD send the same extension as it would send if it were not attempting resumption. When a client includes the defined extension type in an extended ClientHello while requesting session resumption, the server MUST, when agreeing to resume the older session, ignore the extension and send a ServerHello that does not contain the extension. In this case, the functionality of this extension negotiated during the original session initiation is applied to the resumed session. If the resumption request is denied, the use of the extension is negotiated as normal.

2. TLS extension specifying Out-of-band public key retrieval

In order to indicate which server public keys they trust, clients MAY include an extension of type "oob_pubkey_list" in the (extended) ClientHello. The "extension_data" field of this extension SHALL contain "PublicKeyList" where:

```
struct {
    IdentifierType identifier_type;
    select (identifier_type) {
        case key_raw_pubkey: subjectPublicKeyInfo;
        case key_sha256_hash: SHA256Hash;
    } identifier;
} PublicKey;

enum {
    key_raw_pubkey(0), key_sha1_hash(1) (255)
} IdentifierType;

opaque subjectPublicKeyInfo<1..2^16-1>;

opaque SHA256Hash<32>;

 struct {
    PublicKey public_key_list<1..2^16-1>
 } PublicKeyList;
```

In each entry in the list, the client can either include the full public key, in the form of a subjectPublicKeyInfo (see RFC 2528 [RFC2528] and RFC 5480 [RFC5480]) or a SHA-256 hash of the subjectPublicKeyInfo. The PublicKeyList MUST contain at least one PublicKey entry.

The TLS server MAY respond with an extension of type "oob_pubkey_list" in the (extended) server hello. The "extension_data" field of this extension SHALL contain "PublicKeyList" containing exactly one of the "PublicKey" identifiers used in the received client hello message. It SHALL use the same IdentifierType as the TLS client used to send the identifier to the TLS server. It MUST NOT send and empty PublicKeyList.

If the TLS server responds with an extension of type "oob_pubkey_list", it SHOULD omit sending a "Server Certificate" message.

If the TLS server does not respond with an extension of type "oob_pubkey_list", the TLS client MUST assume the extension is not supported. The TLS client MAY fall back to using in-band PKIX validation. If the TLS client cannot fallback to PKIX authentication, it MUST abort the TLS handshake.

3. Security Considerations

The TLS extension defined here lets a TLS client attempt to supress the sending of server certificate as well as the certification chain for that certificate.

A client using this extension needs to be confident in the authenticity of the public key it is using. Since those public keys were obtained out-of-band (hence the name of the extension), the authentication must also be out-of-band.

Depending on exactly how the public keys were obtained, it may be appropriate to use authentication mechanisms tied to the public key transport. For example, if public keys were obtained using [DANE] it is appropriate to use DNSSEC to authenticate the public keys.

4. IANA Considerations

We request that IANA assign a TLS Extension Type value for oob_pubkey_list

5. Contributors

The following individuals made important contributions to this document: Paul Hoffman.

6. Acknowledgements

This document is based on material from RFC 6066 for which the author is Donald Eastlake 3rd. Contributions to that document also include Joseph Salowey, Alexey Melnikov, Peter Saint-Andre, and Adrian Farrel.

7. References

7.1. Normative References

[PKIX] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

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

[RFC2528] Housley, R. and T. Polk, "Internet X.509 Public Key Infrastructure Representation of Key Exchange Algorithm (KEA) Keys in Internet X.509 Public Key Infrastructure Certificates", RFC 2528, March 1999.

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

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, March 2009.

[SECTERMS] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.

7.2. Informative References

[DANE] Hoffman, P. and J. Schlyter, "Using Secure DNS to Associate Certificates with Domain Names For TLS", draft-ietf-dane-protocol-08 (work in progress), July 2011.

[Defeating-SSL] Marlinspike, M., "New Tricks for Defeating SSL in Practice", February 2009, <http://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf>.

[LDAP] Sermersheim, J., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006.

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

Authors' Addresses

Paul Wouters
Xelerance
4130 Ramsayville Road
Ottawa, On K1G 3N4
Canada

Phone: +1-647-722-5653
Email: paul@xelerance.com
URI: https://www.xelerance.com/

John Gilmore
PO Box 170608
San Francisco, California 94117
USA

Phone: +1 415 221 6524
Email: gnu@toad.com
URI: https://www.toad.com/

Sam Weiler
SPARTA, Inc.
7110 Samuel Morse Drive
Columbia, Maryland 21046
US

Email: weiler@tislabs.com