Transport Layer Security (TLS) Cached Information Extension
<draft-ietf-tls-cached-info-04.txt>

Status of this Memo

Copyright and License Notice

Abstract

This document defines a Transport Layer Security (TLS) extension for cached information. This extension allows the TLS client to inform a server of cached information from previous TLS sessions, allowing the server to omit sending cached static information to the client during the TLS handshake protocol exchange.

Table of Contents

1. Introduction

TLS handshakes often include fairly static information such as server certificate and a list of trusted Certification Authorities (CAs). Static information such as a server certificate can be of considerable size. This is the case in particular if the server certificate is bundled with a complete certificate path, including all intermediary certificates up to the trust anchor public key.

Significant benefits can be achieved in low bandwidth and high latency networks, in particular if the communication channel also has a relatively high rate of transmission errors, if a known and previously cached server certificate path can be omitted from the TLS handshake.

This specification defines the Cached Information TLS extension, which may be used by a client and a server to exclude transmission of known cached parameters from the TLS handshake.

1.1. Terminology

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].

2. Cached Information Extension

A new extension type (cached_information(TBD)) is defined and used in both the client hello and server hello messages. The extension type is specified as follows.

```
enum {
     cached_information(TBD), (65535)
} ExtensionType;
```

The "extension_data" field of this extension, when included in the client hello, SHALL contain "CachedInformation" according to the following structure:

```
enum {
     certificate_chain(1), trusted_cas(2), (255)
} CachedInformationType;

struct {
     CachedInformationType type;
     opaque digest_value<0..8>;
} CachedObject;

struct {
     CachedObject cached_info<1..2^16-1>;
} CachedInformation;
```

The digest_value of a CachedObject MUST either be empty (0 bytes) or contain a 64 bit FNV digest (8 bytes) as specified in Annex A.

When CachedInformationType identifies certificate_chain, then digest_value MUST include a digest calculated over the certificate_list element of a server side Certificate message.

When CachedInformationType identifies trusted_cas, then digest_value MUST include a digest calculated over the certificate_authorities element of a server side CertificateRequest message.

Other specifications MAY define more CachedInformationType types.

4. Extension Exchange

4.1. Reconnaissance

A client MAY include an empty cached_information extension (with empty extension_data field) in its (extended) client hello to query whether the server supports cached information.

A server indicates that it supports cached information in handshakes according to section 4.2. by including a cached_information extension in its (extended) server hello.

4.2. Cached Information

Clients MAY specify cached information from previous handshakes by including a "cached_information" extension in the (extended) client hello, which contains at least one cached object (CachedObject) for each present object type (CachedInformationType), as specified in section 2. Clients MAY need the ability to cache different values depending on other information in the Client Hello that modify what values the server uses, in particular the Server Name Indication [RFC4366] value. Clients sending a non-empty cached_information extension MUST provide a 64 bit (8 byte) digest_value for each cached object.

Servers that receive an extended client hello containing a "cached_information" extension, MAY indicate that they support caching of information objects by including an cached_information extension in their (extended) server hello.

A cached_information extension provided in the server hello has the following semantics:

- An empty cached_information extension indicates that the server supports information caching but provides no information about what information types it supports.

- A non-empty cached information extension indicates that the server supports only those CachedInformationType types that are identified by each present CachedObject.

- A CachedObject with an empty digest_value indicates that the server supports caching of the specified object type (CachedInformationType), but does not specify any digest values it will accept.

- A present non-empty digest_value indicates that the server will honor caching of objects of the specified type that matches the

present digest value.

5. Data Substitution

Following a successful exchange of "cached_information" extensions, the server may substitute data objects in the handshake exchange with a matching digest_value representing a matching object type. received from the client in its client hello.

The handshake protocol will proceed using the cached data as if it was provided in the handshake protocol. The Finished message will however be calculated over the actual data exchanged in the handshake protocol. That is, the Finished message will be calculated over the digest values of cached information objects and not over the cached objects that were omitted from transmission.

Each CachedInformationType MUST specify how actual data is replaced by a digest in a way that does not violate the defined syntax of existing handshake messages. the data exchange syntax for certificate_chain(1) and trusted_cas(2) are provided below.

The server MUST NOT provide more than one digest value as substitution for the cached data.

5.1. Data Substitution Syntax for certificate_chain

When a digest for an object of type trusted_cas is provided in the client hello, the server MAY substitute the cached data with a matching digest value received from the client by expanding the Certificate handshake message as follows.

Original handshake message syntax defined in RFC 5246 [RFC5246]:

```
opaque ASN.1Cert<1..2^24-1>;

struct {
    ASN.1Cert certificate_list<0..2^24-1>;
} Certificate;
```

Substitution syntax is defined by expanding the definition of the opaque ASN.1Cert structure:

```
struct {
    opaque digest_value<0..8>;
} ASN.1Cert
```

5.2. Data Substitution Syntax for trusted_cas

When a digest for an object of type trusted_cas is provided in the client hello, the server MAY substitute the cached data with a matching digest value received from the client by expanding the CertificateRequest handshake message as follows.

Original handshake message syntax defined in RFC 5246 [RFC5246]:

```
opaque DistinguishedName<1..2^16-1>;

struct {
    ClientCertificateType certificate_types<1..2^8-1>;
    SignatureAndHashAlgorithm
      supported_signature_algorithms<2^16-1>;
    DistinguishedName certificate_authorities<0..2^16-1>;
} CertificateRequest
```

The substitution syntax is defined by expanding the definition of the opaque DistinguishedName structure:

```
struct {
    opaque digest_value<0..8>;
} DistinguishedName
```

5. Security Considerations

The digest algorithm used in this specification is required to have reasonable random properties in order to provide reasonably unique identifiers. There is no requirement that this digest algorithm must have strong collision resistance. A non unique digest may at most lead to a failed TLS handshake followed by a new attempt without the cached information extension. There are no identified security threats that require the selected digest algorithm to have strong collision resistance.

6. IANA Considerations

1) Create an entry, cached_information(TBD), in the existing registry for ExtensionType (defined in RFC 5246 [RFC5246]).

2) Establish a registry for TLS CachedInformationType values. The first entries in the registry are certificate_chain(1) and trusted_cas(2). TLS CachedInformationType values in the inclusive range 0-63 (decimal) are assigned via RFC 5226 [RFC5226] Standards Action. Values from the inclusive range 64-223 (decimal) are assigned via RFC 5226 Specification Required. Values from the inclusive range 224-255 (decimal) are reserved for RFC 5226 Private Use.

7. Normative References

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

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

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

[RFC4366] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, April 2006

NOTE: RFC 4366 will be updated by RFC4366bis, currently in IESG process.

Annex A - 64 bit FNV Digest

FNV-1 digest algorithm is a non-cryptographic hash function created by Glenn Fowler, Landon Curt Noll, and Phong Vo. The FNV digest algorithms and sample FNV source code have been released into the public domain.

The FNV-1 digest is generated as follows:

```
digest = FNV_offset_basis
for each octet_of_data to be digested
  digest = digest * FNV_prime
  digest = digest XOR octet_of_data
return digest
```

In the above pseudocode, all variables are unsigned integers. All variables, except for octet_of_data, have the same number of bits as the FNV digest (64 Bits). The variable, octet_of_data, is an 8 bit unsigned integer. Specifically for a 64 bit FNV-1 digest the following applies:

- o All variables, except for octet_of_data, are 64-bit unsigned integers.

- o The variable, octet_of_data, is an 8 bit unsigned integer.

- o The FNV_offset_basis is the 64-bit FNV offset basis value: 14695981039346656037.

- o The FNV_prime is the 64-bit FNV prime value: 1099511628211.

- o The multiply function (indicated by the '*' symbol) returns the lower 64-bits of the product.

- o The XOR is an 8-bit operation that modifies only the lower 8-bits of the digest value.

- o The digest value returned is an 64-bit unsigned integer.

Authors' Addresses

Stefan Santesson

3xA Security AB
Bjornstorp 744
247 98 Genarp
Sweden

EMail: sts@aaa-sec.com