Network Working Group
Internet-Draft
Expires: May 12, 2013

D. Balfanz
R. Hamilton
Google Inc

November 8, 2012

Transport Layer Security (TLS) Channel IDs
draft-balfanz-tls-channelid-00

Abstract

This document describes a Transport Layer Security (TLS) extension for identifying client machines at the TLS layer without using bearer tokens.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

Many applications on the Internet use _bearer tokens_ to authenticate clients to servers. The most prominent example is the HTTP-based World Wide Web, which overwhelmingly uses HTTP cookies to authenticate client requests. Other examples include OpenID or SAML assertions, and OAuth tokens. All these have in common that the _bearer_ of the HTTP cookie or authentication token is granted access to a protected resource, regardless of the channel over which the token is presented, or who presented it.

As a result, an adversary that manages to steal a bearer token from a client can impersonate that client to services that require the token.

This document describes a light-weight mechanism for establishing a _cryptographic channel_ between client and server. A server can choose to bind authentication tokens to this channel, thus rendering the theft of authentication tokens fruitless - tokens must be sent over the channel to which they are bound (i.e., by the client to which they were issued) or else they will be ignored.

This document does not prescribe _how_ authentication tokens are bound to the underlying channel. Rather, it prescribes how a client can establish a long-lived channel with a server. Such a channel persists across HTTP requests, TLS connections, and even multiple TLS sessions, as long as the same client communicates with the same server.

The basic idea is that the client proves, during the TLS handshake, possession of a private key. The corresponding public key becomes the "Channel ID" that identifies this TLS connection. Clients should re-use the same private/public key pair across subsequent TLS connections to the same server, thus creating TLS connections that share the same Channel ID.

Using private/public key pairs to define a channel (as opposed to, say, an HTTP session cookie) has several advantages: One, the credential establishing the channel (the private key) is never sent from client to server, thus removing it from the reach of eavesdroppers in the network. Two, clients can choose to implement cryptographic operations in a secure hardware module, which further removes the private key from the reach of eavesdroppers residing on the client itself.

2. Why not client certificates

TLS already supports a means of identifying clients without using bearer tokens: client certificates. However, a number of problems with using client certificates motivated the development of an alternative.

Most importantly, it’s not acceptable for a client identifier to be transmitted in the clear and client certificates in TLS are sent unencrypted. Although we could also define a change to the TLS state machine to move the client certificates under encryption, such changes eliminate most of the benefits of reusing something that’s already defined.

TLS client certificates are also defined to be part of the session state. This turns session resumption secrets into equivalent barer tokens; completely defeating our objectives.

Client-certificates typically identify a user, while we seek to identify machines. Since they are not, conceptually, mutually exclusive and as only a single client certificate can be provided in TLS, we don’t want to consume that single slot and eliminate the possibility of also using existing client certificates.

Client certificates are implemented in TLS as X.509 certificates and we don’t wish to require servers to parse arbitrary ASN.1. ASN.1 is a complex encoding that has been the source of several security vulnerabilities in the past and typical TLS servers can currently avoid doing ASN.1 parsing.

X.509 certificates always include a signature, which would be a self-signature in this case. Calculating and transmitting the self-signature is a waste of computation and network traffic in our use. Although we could define a null signature algorithm with an empty signature, such deviations from X.509 eliminate many of the benefits of reusing something that is already implemented.

Finally, client certificates trigger significant server-side processing by default and often need to be stored in their entirety for the duration of the connection. Since this design is intended to be widely used, it allows servers to retain only a cryptographic hash of the client’s public key after the handshake completes.

3. Requirements Notation

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].

4. Channel ID Extension

A new extension type ("channel_id(TBD)") is defined and MAY be included by the client in its "ClientHello" message. If, and only if, the server sees this extension in the "ClientHello", it MAY choose to echo the extension in its "ServerHello". In both cases, the "extension_data" field MUST be empty.

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

A new handshake message type ("encrypted_extensions(TBD)") is defined. If the server included a "channel_id" extension in its "ServerHello" message, the client MUST verify that the selected cipher suite is sufficiently strong. If the cipher suite provides < 80-bits of security, the client MUST abort the handshake with a fatal "illegal_parameter" alert. Otherwise, the client MUST send an "EncryptedExtensions" message after its "ChangeCipherSpec" and before its "Finished" message.

```
enum {
  encrypted_extensions(TBD), (65535)
} HandshakeType;
```

Therefore a full handshake with "EncryptedExtensions" has the following flow (contrast with section 7.3 of RFC 5246 [RFC5246]):

```
Client                                                     Server

ClientHello (ChannelID extension)   -------->
                                                      ServerHello
                                            (ChannelID extension)
                                                     Certificate*
                                               ServerKeyExchange*
                                              CertificateRequest*
                              <--------           ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
EncryptedExtensions
Finished                      -------->
                                               [ChangeCipherSpec]
                              <--------                  Finished
Application Data              <------->          Application Data
```

An abbreviated handshake with "EncryptedExtensions" has the following flow:

```
Client                                                 Server

ClientHello (ChannelID extension)    -------->
                                                  ServerHello
                                        (ChannelID extension)
                                           [ChangeCipherSpec]
                              <--------              Finished
[ChangeCipherSpec]
EncryptedExtensions
Finished                      -------->
Application Data              <------->    Application Data
```

The "EncryptedExtensions" message contains a series of "Extension" structures (see section 7.4.1.4 of RFC 5246 [RFC5246]

If the server included a "channel_id" extension in its "ServerHello" message, the client MUST include an "Extension" with "extension_type" equal to "channel_id(TBD)". The "extension_data" of which has the following format:

```
struct {
  opaque x[32];
  opaque y[32];
  opaque r[32];
  opaque s[32];
} ChannelIDExtension;
```

The contents of each of "x", "y", "r" and "s" is a 32-byte, big-endian number. The "x" and "y" fields contain the affine coordinates of a P-256 [DSS] curve point. The "r" and "s" fields contain an ECDSA [DSS] signature by the corresponding private key of "TLS Channel ID signature\x00" followed by the handshake hash(es) prior to the "EncryptedExtensions" message.

Unlike many other TLS extensions, this extension does not establish properties of the session, only of the connection. When session resumption or session tickets [RFC5077] are used, the previous contents of this extension are irrelevant and only the values in the new handshake messages are considered.

5. Security Considerations

There are four classes of attackers against which we consider our security guarantees: passive network attackers, active network attackers, active network attackers with misissued certificates and attackers in possession of the legitimate server’s private key.

First, we wish to guarantee that we don’t disclose the Channel ID to passive or active network attackers. We do this by sending a constant-length Channel ID under encryption. However, since the Channel ID may be transmitted before the server’s Finished message is received, it’s possible that the server isn’t in possession of the certificate that it presented. In this situation, an active attacker could cause a Channel ID to be transmitted under a random key in a cipher suite of their choosing. Therefore we limit the permissible cipher suites to those where decrypting the message is infeasible.

Even with this limit, an active attacker can cause the Channel ID to be transmitted in a non-forward-secure manner. Subsequent disclosure of the server’s private key would allow previously recorded Channel IDs to be decrypted.

Second, we wish to guarantee that none of the first three attackers can terminate/hijack a TLS connection and impersonate a Channel ID from that connection when connecting to the legitimate server. We assume that TLS provides sufficient security to prevent a connection from being hijacked once established by these attackers. An active attacker with a misissued certificate can successfully terminate the TLS connection and decrypt the Channel ID. However, as the signature covers the handshake hashes, and therefore the server’s certificate, it wouldn’t be accepted by the true server.

Against an attacker with the legitimate server’s private key we can provide the second guarantee only if the legitimate server uses a forward-secret cipher suite, otherwise the attacker can hijack the connection.

6. Privacy Considerations

The TLS layer does its part in protecting user privacy by transmitting the Channel ID public key under encryption. Higher levels of the stack must ensure that the same Channel ID is not used with different servers in such a way as to provide a linkable identifier. For example, a user-agent must use different Channel IDs for communicating with different servers. User-agents must also ensure that Channel ID state can be reset by the user in the same way as other identifiers, i.e. cookies.

However, there are some security concerns that could result in the disclosure of a client’s Channel ID to a network attacker. This is covered in the Security Considerations section.

7. IANA Considerations

This document requires IANA to update its registry of TLS extensions to assign an entry referred to here as "channel_id".

This document also requires IANA to update its registry of TLS handshake types to assign an entry referred to here as "encrypted_extensions".

8. References

8.1. 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.

[DSS] National Institute of Standards and Technology, "FIPS 186-3: Digital Signature Standard".

8.2. Informative References

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

Appendix A. Acknowledgements

The following individuals contributed to this specification:

Dirk Balfanz, Wan-Teh Chang, Ryan Hamilton, Adam Langley, and Mayank Upadhyay.

Authors’ Addresses

Dirk Balfanz
Google Inc

Email: balfanz@google.com

Ryan Hamilton
Google Inc

Email: rch@google.com