HTTP
Internet-Draft
Updates: 7450 (if approved)
Intended status: Standards Track
Expires: April 21, 2016

M. Thomson
Mozilla
M. Bishop
Microsoft

October 19, 2015

Reactive Certificate-Based Client Authentication in HTTP/2
draft-thomson-http2-client-certs-00

Abstract

Some HTTP servers provide a subset of resources that require additional authentication to interact with. HTTP/1.1 servers rely on TLS renegotiation that is triggered by a request to a protected resource. HTTP/2 made this pattern impossible by forbidding the use of TLS renegotiation.

This document describes a how client authentication might be requested by a server as a result of receiving a request to a protected resource. This document updates RFC 7540 to allow TLS renegotiation in limited circumstances.

Table of Contents

1. Introduction

Many existing HTTP [RFC7230] servers have different authentication requirements for the different resources they serve. Of the bountiful authentication options available for authenticating HTTP requests, client certificates present a unique challenge for resource-specific authentication requirements because of the interaction with the underlying TLS RFC5246 [I-D.ietf-tls-tls13] layer.

For servers that wish to use client certificates to authenticate users, they might request client authentication during the TLS handshake. However, if not all users or resources need certificate-based authentication, a request for a certificate has the unfortunate consequence of triggering the client to seek a certificate. Such a request can result in a poor experience, particular when sent to a client that does not expect the request.

The TLS CertificateRequest can be used by servers to give clients hints about which certificate to offer. Servers that rely on certificate-based authentication might request different certificates for different resources. Such a server cannot use contextual information about the resource to construct an appropriate TLS CertificateRequest message during the initial handshake.

Consequently, client certificates are requested at connection establishment time only in cases where all clients are expected or required to have a single certificate that is used for all resources. Many other uses for client certificates are reactive, that is, certificates are requested in response to the client making a request.

CAVEAT: As of 2015-10-02, TLS 1.3 does not include the client authentication features this draft relies on. While these features have been agreed in the TLS working group, the exact design is still under revision. The basic functionality shouldn't change in a way that will affect this document, though some details such as field names are highly likely to change.

1.1. Reactive Certificate Authentication in HTTP/1.1

In HTTP/1.1, a server that relies on client authentication for a subset of users or resources does not request a certificate when the connection is established. Instead, it only requests a client certificate when a request is made to a resource that requires a certificate.

Figure 1 shows the server initiating a TLS-layer renegotiation in response to receiving an HTTP/1.1 request to a protected resource.

Figure 1: HTTP/1.1 Reactive Certificate Authentication with TLS 1.2

In this example, the server receives a request for a protected resource (at *1 on Figure 1). Upon performing an authorization check, the server determines that the request requires authentication

using a client certificate and that no such certificate has been provided.

The server initiates TLS renegotiation by sending a TLS HelloRequest (at *2). The client then initiates a TLS handshake. Note that some TLS messages are elided from the exchange for the sake of brevity.

The critical messages for this example are the server requesting a certificate with a TLS CertificateRequest (*3); this request might use information about the request or resource. The client then provides a certificate and proof of possession of the private key in Certificate and CertificateVerify messages (*4).

When the handshake completes, the server performs any authorization checks a second time. With the client certificate available, it then authorizes the request and provides a response (*5).

1.2. TLS 1.3 Client Authentication

TLS 1.3 [I-D.ietf-tls-tls13] introduces a new client authentication mechanism that allows for clients to authenticate after the handshake has been completed. For the purposes of authenticating an HTTP request, this is functionally equivalent to renegotiation. Figure 2 shows the simpler exchange this enables.

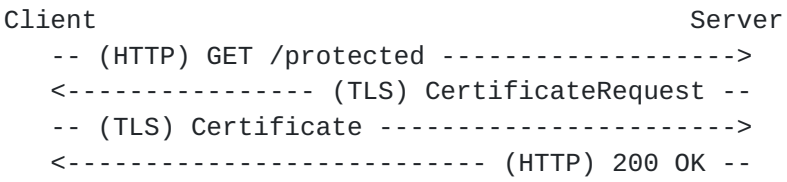

Figure 2: HTTP/1.1 Reactive Certificate Authentication with TLS 1.3

TLS 1.3 does not support renegotiation, instead supporting direct client authentication. In contrast to the TLS 1.2 example, in TLS 1.3, a server can simply request a certificate.

1.3. Reactive Client Authentication in HTTP/2

An important part of the HTTP/1.1 exchange is that the client is able to easily identify the request that caused the TLS renegotiation. The client is able to assume that the next unanswered request on the connection is responsible. The HTTP stack in the client is then able to direct the certificate request to the application or component that initiated that request. This ensures that the application has the right contextual information for processing the request.

In HTTP/2, a client can have multiple outstanding requests. Without some sort of correlation information, a client is unable to identify which request caused the server to request a certificate.

Thus, the minimum necessary mechanism to support reactive certificate authentication in HTTP/2 is an identifier that can be use to correlate an HTTP request with either a TLS renegotiation or CertificateRequest.

Section 2 describes how the existing TLS 1.3 fields and a new HTTP/2 frame described in Section 4 can be used to correlate a request with a TLS CertificateRequest. Section 3 describes how the same can be done in TLS 1.2 using TLS renegotiation and a new TLS "application_context_id" extension. Finally, Section 5 describes how an HTTP/2 client can announce support for this feature so that a server might use these capabilities.

1.4. Terminology

RFC 2119 [RFC2119] defines the terms "MUST", "MUST NOT", "SHOULD" and "MAY".

2. HTTP/2 Request Correlation in TLS 1.3

An HTTP/2 request from a client that has signaled support for reactive certificate authentication (see Section 5) might cause a server to request client authentication. In TLS 1.3 a server does this by sending a new TLS 1.3 CertificateRequest.

The server MUST first send a WAITING_FOR_AUTH frame (see Section 4) on the stream which triggered the request for client credentials. The certificate_request_id (name TBD) field of the TLS CertificateRequest is populated by the server with the same value in the WAITING_FOR_AUTH frame. Subsequent WAITING_FOR_AUTH frames with the same request identifier MAY be sent on other streams while the server is awaiting client authentication with the same parameters. This allows a client to correlate the TLS CertificateRequest with one or more outstanding requests.

A server MAY send multiple concurrent TLS CertificateRequest messages. If a server requires that a client provide multiple certificates before authorizing a single request, it MUST send WAITING_FOR_AUTH frames with different request identifiers before sending subsequent TLS CertificateRequest messages.

3. HTTP/2 Request Correlation in TLS 1.2

An HTTP/2 server that uses TLS 1.2 initiates client authentication by sending a an HTTP/2 WAITING_FOR_AUTH frame followed by a TLS HelloRequest. This triggers a TLS renegotiation.

An HTTP/2 client that receives a TLS HelloRequest message MUST initiate a TLS handshake, including an empty "application_context_id" extension. If the client has not indicated support for renegotiation (see Section 5), the client MUST send a fatal TLS "no_renegotiation" alert.

The server populates the "application_context_id" extension with the same value it previously sent in a WAITING_FOR_AUTH frame.

Absence of an "application_context_id" extension or an empty value from the server MUST be treated as a fatal error; endpoints MAY send a fatal TLS "no_renegotiation" alert.

As with the TLS 1.3 solution, a server MAY request multiple client certificates, either for different requests or for the same request. If multiple requests are waiting for authentication and require different certificates, the server SHOULD immediately send the WAITING_FOR_AUTH frames with unique values. Only one TLS renegotiation can be in progress at a time, though a new HelloRequest can be emitted once the renegotiation has completed.

A server MAY treat all certificates presented in the same connection as cumulative, remembering multiple certificates as they are presented. Note that the authentication information collected from the client will need to be checked after each TLS renegotiation completes, since most TLS stacks only report the presence of the client certificate presented during the last TLS handshake.

3.1. The TLS application_context_id Hello Extension

The "application_context_id" TLS Hello Extension is used to carry an identifier from an application context in the TLS handshake. This is used to identify the application context that caused the TLS handshake to be initiated. The semantics of the field depend on application protocol, and could further depend on application protocol state.

Either client or server can populate this field. A client can provide an empty value to indicate that it does not know the application context, but would like the server to provide a value. A server can provide an empty value in response to a non-empty value only.

In HTTP/2 clients always provide an empty "application_context_id" value, and servers always provide a value that will appear in a subsequent WAITING_FOR_AUTH frame.

```
enum {
    ...
    application_context_id(EXTENSION-TBD),
    (65535)
} ExtensionType;

struct {
    opaque id<0..255>;
} ApplicationContextId;
```

Figure 3: The application_context_id Extension Format

3.2. Permitting TLS Renegotiation in HTTP/2

The prohibition from Section 9.2.1 of [RFC7540] against TLS renegotiation is removed, provided that the requirements of this section are adhered to.

TLS renegotiation MUST NOT be used to circumvent the other restrictions on TLS use from Section 9.2 of [RFC7540]. Furthermore, TLS renegotiation MUST negotiate the same ALPN [RFC7301] identifier (that is, "h2"). An endpoint MAY treat failure to comply with these requirements as a connection error (Section 5.4.1 of [RFC7540]) of type INADEQUATE_SECURITY.

Note: A client need not offer cipher suites that might otherwise be offered for compatibility reasons when renegotiating. In particular, cipher suites on the black list from Appendix A of [RFC7540] can be removed from the handshake.

In addition to the requirements from [RFC7540], endpoints that renegotiate MUST implement the TLS extended master secret extension [RFC7627] and the TLS renegotiation indication extension [RFC5746]. These extensions MUST be negotiated and used to prevent serious attacks on TLS renegotiation. If an endpoint receives a TLS ClientHello or ServerHello that does not include these extensions, it MUST respond with a fatal TLS "no_renegotiation" alert.

The TLS renegotiation handshake MUST include the "application_context_id" extension when used with HTTP/2.

A server MUST present the same certificate during TLS renegotiation it used during the initial handshake. Clients MUST verify that the server certificate does not change. Clients MUST verify that the

server certificate has not changed; a different certificate MUST be treated as a fatal error and MAY cause a fatal "handshake_failure" alert to be sent.

Once the HTTP/2 connection preface has been received from a peer, an endpoint SHOULD treat the receipt of a TLS ClientHello or ServerHello without an "application_context_id" extension as a fatal error and SHOULD send a fatal TLS "no_renegotiation" alert.

4. Indicating Stream Dependency on Certificate Authentication

The WAITING_FOR_AUTH frame (0xFRAME-TBD) is sent by servers to indicate that processing of a request is blocked pending authentication outside of the HTTP channel. The frame includes a request identifier which can be used to correlate the stream with challenges for authentication received at other layers, such as TLS.

The WAITING_FOR_AUTH frame contains between 1 and 255 octets, which is the authentication request identifier. A client that receives a WAITING_FOR_AUTH of any other length MUST treat this as a stream error of type PROTOCOL_ERROR. Frames with identical request identifiers refer to the same TLS CertificateRequest.

The WAITING_FOR_AUTH frame MUST NOT be sent by clients. A WAITING_FOR_AUTH frame received by a server SHOULD be rejected with a stream error of type PROTOCOL_ERROR.

The server MUST NOT send a WAITING_FOR_AUTH frame on stream zero, a server-initiated stream or a stream that does not have an outstanding request. In other words, a server can only send in the "open" or "half-closed (remote)" stream states.

A client that receives a WAITING_FOR_AUTH frame on a stream which is not in a valid state ("open" or "half-closed (local)" for clients) SHOULD treat this as a connection error of type PROTOCOL_ERROR.

5. Indicating Support for Reactive Certificate Authentication

Clients that support reactive certificate authentication indicate this using the HTTP/2 "SETTINGS_REACTIVE_AUTH" (0xSETTING-TBD) setting.

The initial value for the "SETTINGS_REACTIVE_AUTH" setting is 0, indicating that the client does not support reactive client authentication. A client sets the "SETTINGS_REACTIVE_AUTH" setting to a value of 1 to indicate support for reactive certificate authentication as defined in this document. Any value other than 0

or 1 MUST be treated as a connection error (Section 5.4.1 of [RFC7540]) of type PROTOCOL_ERROR.

6. Security Considerations

The TLS extended master secret extension [RFC7627] and the TLS renegotiation indication extension [RFC5746] MUST be used to mitigate several known attacks on TLS renegotiation.

Adding correlation between requests and TLS-layer authentication addresses the primary functional concerns with mid-session client authentication. However, implementations need to be aware of the potential for confusion about the state of a connection.

The presence or absence of a validated client certificate can change during the processing of a request, potentially multiple times. A server that uses reactive certificate authentication needs to be prepared to reevaluate the authorization state of a request as the set of certificates changes.

7. IANA Considerations

The TLS "application_context_id" extension is registered in Section 7.1. The HTTP/2 "SETTINGS_REACTIVE_AUTH" setting is registered in Section 7.2. The HTTP/2 "WAITING_FOR_AUTH" frame type is registered in Section 7.3.

7.1. TLS application_context_id Extension

The "application_context_id" TLS extension is registered in the "ExtensionType Values" registry established by [RFC5246].

Value: EXTENSION-TBD

Extension name: application_context_id

Reference: This document.

7.2. HTTP/2 SETTINGS_REACTIVE_AUTH Setting

The SETTINGS_REACTIVE_AUTH setting is registered in the "HTTP/2 Settings" registry established in [RFC7540].

Name: SETTINGS_REACTIVE_AUTH

Code: 0xSETTING-TBD

Initial Value: 0

Specification: This document.

7.3. HTTP/2 WAITING_FOR_AUTH Frame

The WAITING_FOR_AUTH frame type is registered in the "HTTP/2 Frame Types" registry established in [RFC7540].

Frame Type: WAITING_FOR_AUTH

Code: 0xFRAME-TBD

Specification: This document.

8. Acknowledgements

Eric Rescorla pointed out several failings in an earlier revision.

9. Normative References

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-08 (work in progress), August 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5746] Rescorla, E., Ray, M., Dispensa, S., and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, DOI 10.17487/RFC5746, February 2010, <http://www.rfc-editor.org/info/rfc5746>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <http://www.rfc-editor.org/info/rfc7301>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <http://www.rfc-editor.org/info/rfc7540>.

[RFC7627] Bhargavan, K., Ed., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", RFC 7627, DOI 10.17487/RFC7627, September 2015, <http://www.rfc-editor.org/info/rfc7627>.

Authors' Addresses

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

Mike Bishop
Microsoft

Email: michael.bishop@microsoft.com