Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 16, 2015

A. Popov
M. Nystroem
Microsoft Corp.
D. Balfanz, Ed.
A. Langley
Google Inc.

October 13, 2014

Token Binding over HTTP
draft-balfanz-https-token-binding-00

Abstract

This document describes a collection of mechanisms that allow HTTP servers to cryptographically bind authentication tokens (such as cookies and OAuth tokens) to a TLS [RFC5246] connection.

We describe both _first-party_ as well as _federated_ scenarios. In a first-party scenario, an HTTP server issues a security token (such as a cookie) to a client, and expects the client to send the security token back to the server at a later time in order to authenticate. Binding the token to the TLS connection between client and server protects the security token from theft, and ensures that the security token can only be used by the client that it was issued to.

Federated token bindings, on the other hand, allow servers to cryptographically bind security tokens to a TLS [RFC5246] connection that the client has with a _different_ server than the one issuing the token.

This Internet-Draft is a companion document to The Token Binding Protocol [DraftPopov]

Status of This Memo

This Internet-Draft will expire on April 16, 2015.

Copyright Notice

Table of Contents

1. Introduction

The Token Binding Protocol [DraftPopov] defines a Token Binding ID for a TLS connection between a client and a server. The Token Binding ID of a TLS connection is related to a private key that the client proves possession of to the server, and is long-lived (i.e., subsequent TLS connections between the same client and server have the same Token Binding ID). When issuing a security token (e.g. an HTTP cookie or an OAuth token) to a client, the server can include the Token Binding ID in the token, thus cryptographically binding the

token to TLS connections between that particular client and server, and inoculating the token against theft by attackers.

While the Token Binding Protocol [DraftPopov] defines a message format for establishing a Token Binding ID, it doesn’t specify how this message is embedded in higher-level protocols. The purpose of this specification is to define how TokenBindingMessages are embedded in HTTP (both versions 1.1 [RFC2616] and 2 [I-D.ietf-httpbis-http2]). Note that TokenBindingMessages are only defined if the underlying transport uses TLS. This means that Token Binding over HTTP is only defined when the HTTP protocol is layered on top of TLS (commonly referred to as HTTPS).

HTTP clients establish a Token Binding ID with a server by including a special HTTP header in HTTP requests. The HTTP header value is a TokenBindingMessage.

TokenBindingMessages allow clients to establish multiple Token Binding IDs with the server, by including multiple TokenBinding structures in the TokenBindingMessage. By default, a client will establish a _provided_ Token Binding ID with the server, indicating a Token Binding ID that the client will persistently use with the server. Under certain conditions, the client can also include a _referred_ Token Binding ID in the TokenBindingMessage, indicating a Token Binding ID that the client is using with a _different_ server than the one that the TokenBindingMessage is sent to. This is useful in federation scenarios.

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 [RFC2119].

2. The Token-Binding Header

Once a client and server have negotiated the Token Binding Protocol with HTTP/1.1 or HTTP/2 (see The Token Binding Protocol [DraftPopov]), clients MUST include the following header in their HTTP requests:

```
Token-Binding: EncodedTokenBindingMessage
```

The EncodedTokenBindingMessage is a web-safe Base64-encoding of the TokenBindingMessage as defined in the TokenBindingProtocol [DraftPopov].

The TokenBindingMessage MUST contain a TokenBinding with TokenBindingType provided_token_binding, which MUST be signed with the Token Binding key used by the client for connections between itself and the server that the HTTP request is sent to (clients use different Token Binding keys for different servers). The Token Binding ID established by this TokenBinding is called a _Provided Token Binding ID_

In HTTP/2, the client SHOULD use Header Compression [I-D.ietf-httpbis-header-compression] to avoid the overhead of repeating the same header in subsequent HTTP requests.

3. Federation Use Cases

3.1. Introduction

For privacy reasons, clients use different private keys to establish Provided Token Binding IDs with different servers. As a result, a server cannot bind a security token (such as an OAuth token or an OpenID Connect identity token) to a TLS connection that the client has with a different server. This is, however, a common requirement in federation scenarios: For example, an Identity Provider may wish to issue an identity token to a client and cryptographically bind that token to the TLS connection between the client and a Relying Party.

In this section we describe mechanisms to achieve this. The common idea among these mechanisms is that a server (called the _Token Consumer_ in this document) gives the client permission to reveal the Provided Token Binding ID that is used between the client and itself, to another server (called the _Token Provider_ in this document). Also common across the mechanisms is how the Token Binding ID is revealed to the Token Provider: The client uses the Token Binding Protocol [DraftPopov], and includes a TokenBinding structure in the Token-Binding HTTP header defined above. What differs between the various mechanisms is _how_ the Token Consumer grants the permission to reveal the Token Binding ID to the Token Provider.

3.2. Overview

In a Federated Sign-On protocol, an Identity Provider issues an identity token to a client, which sends the identity token to a Relying Party to authenticate itself. Examples of this include OpenID Connect (where the identity token is called "ID Token") and SAML (where the identity token is a SAML assertion).

To better protect the security of the identity token, the Identity Provider may wish to bind the identity token to the TLS connection

between the client and the Relying Party, thus ensuring that only said client can use the identity token: The Relying Party will compare the Token Binding ID in the identity token with the Token Binding ID of the TLS connection between it an the client.

This is an example of a federation scenario, which more generally can be described as follows:

- o A Token Consumer causes the client to issue a token request to the Token Provider. The goal is for the client to obtain a token and then use it with the Token Consumer.

- o The client delivers the token request to the Token Provider.

- o The Token Provider issues the token. The token is issued for the specific Token Consumer who requested it (thus preventing malicious Token Consumers from using tokens with other Token Consumers). The token is, however, typically a bearer token, meaning that any client can use it with the Token Consumer, not just the client to which it was issued.

- o Therefore, in the previous step, the Token Provider may want to include the Token Binding ID of the TLS connection between the client and the Token Consumer in the token.

- o That Token Binding ID must therefore be communicated to the Token Provider along with the token request. Communicating a Token Binding ID involves proving possession of a private key and is described in the Token Binding Protocol [DraftPopov].

The client will perform this last operation (proving possession of a private key that corresponds to a Token Binding ID between the client and the Token Consumer while delivering the token request to the Token Provider) only if the Token Consumer permits the client to do so.

Below, we will enumerate a number of mechanisms available to Token Consumers to grant this permission.

3.3. HTTP Redirects

When a Token Consumer redirects the client to a Token Provider as a means to deliver the token request, it SHOULD include the following HTTP response header in its HTTP response:

```
Include-Referer-Token-Binding-ID: true
```

Including this response header signals to the client that it should reveal the Token Binding ID used between the client and the Token Consumer to the Token Provider. In the absence of this response header, the client will not disclose any information about the Token Binding used between the client and the Token Consumer to the Token Provider.

This header has only meaning if the HTTP status code is 302 or 301, and MUST be ignored by the client for any other status codes. If the client supports the Token Binding Protocol, and has negotiated the Token Binding Protocol with both the Token Consumer and the Token Provider, it already sends the following header to the Token Provider with each HTTP request (see above):

```
Token-Binding: EncodedTokenBindingMessage
```

The TokenBindingMessage SHOULD contain a TokenBinding with TokenBindingType referred_token_binding. If included, this TokenBinding MUST be signed with the Token Binding key used by the client for connections between itself and the Token Consumer (more specifically, the web origin that issued the Include-Referer-Token-Binding-ID response header). The Token Binding ID established by this TokenBinding is called a _Referred Token Binding ID_.

As described above, the TokenBindingMessage MUST additionally contain a Provided Token Binding ID, i.e., a TokenBinding structure with TokenBindingType provided_token_binding, which MUST be signed with the Token Binding key used by the client for connections between itself and the Token Privider (more specifically, the web origin that the token request sent to).

3.4. Cross-Origin Resource Sharing

When issuing an XML HTTP request across origins to a Token Provider, a Token Consumer can reveal its Token Binding ID through the withRefererTokenBindingID property of the XmlHttpRequest object. Example:

```
var xhr = new XMLHttpRequest();
xhr.withCredentials = true; // send cookies
xhr.withRefererTokenBindingID = true;
xhr.open(method, url, true);
```

The client SHOULD include the Token-Binding: header to the outgoing request as described above if:

- o the withRefererTokenBindingID property of the XmlHttpRequest object is set to true, and

- o the client has negotiated the Token Binding Protocol both with the web origin that issued the XmlHttpRequest, and the web origin to which the XmlHttpRequest is addressed.

3.5. Negotiated Key Parameters

The Token Binding Protocol [DraftPopov] allows the server and client to negotiate a signature algorithm used in the TokenBindingMessage. It is possible that the Token Binding ID used between the client and the Token Consumer, and the Token Binding ID used between the client and Token Provider, use different signature algorithms. The client MUST use the signature algorithm negotiated with the Token Consumer in the referred_token_binding TokenBinding of the TokenBindingMessage, even if that signature algorithm is different from the one negotiated with the origin that the header is sent to.

Token Providers SHOULD support all the SignatureAndHashAlgorithms specified in the Token Binding Protocol [DraftPopov]. If a token provider does not support the SignatureAndHashAlgorithm specified in the referred_token_binding TokenBinding in the TokenBindingMessage, it MUST issue an unbound token.

4. Security Considerations

4.1. Security Token Replay

The goal of the Federated Token Binding mechanisms is to prevent attackers from exporting and replaying tokens used in protocols between the client and Token Consumer, thereby impersonating legitimate users and gaining access to protected resources. Bound tokens can still be replayed by malware present in the client. In order to export the token to another machine and successfully replay it, the attacker also needs to export the corresponding private key. The Token Binding private key is therefore a high-value asset and MUST be strongly protected, ideally by generating it in a hardware security module that prevents key export.

4.2. Privacy Considerations

The Token Binding protocol uses persistent, long-lived TLS Token Binding IDs. To protect privacy, TLS Token Binding IDs are never transmitted in clear text and can be reset by the user at any time, e.g. when clearing browser cookies. Unique Token Binding IDs MUST be generated for connections to different origins, so they cannot be used by cooperating servers to link user identities.

4.3. Triple Handshake Vulnerability in TLS

The Token Binding protocol relies on the tls_unique value to associate a TLS connection with a TLS Token Binding. The triple handshake attack [TRIPLE-HS] is a known TLS protocol vulnerability allowing the attacker to synchronize tls_unique values between TLS connections. The attacker can then successfully replay bound tokens. For this reason, the Token Binding protocol MUST NOT be negotiated unless the Extended Master Secret TLS extension [I-D.ietf-tls-session-hash] has also been negotiated.

5. References

5.1. Normative References

[DraftPopov]
Popov, A., "The Token Binding Protocol Version 1.0", 2014.

[I-D.ietf-httpbis-header-compression]
Peon, R. and H. Ruellan, "HPACK - Header Compression for HTTP/2", draft-ietf-httpbis-header-compression-09 (work in progress), July 2014.

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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May 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.

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, July 2010.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, July 2014.

5.2. Informative References

[I-D.ietf-httpbis-http2]
Belshe, M., Peon, R., and M. Thomson, "Hypertext Transfer Protocol version 2", draft-ietf-httpbis-http2-14 (work in progress), July 2014.

[I-D.ietf-tls-session-hash]
Bhargavan, K., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", draft-ietf-tls-session-hash-02 (work in progress), October 2014.

[TRIPLE-HS]
Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Pironti, A., and P. Strub, "Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS. IEEE Symposium on Security and Privacy", 2014.

Authors' Addresses

Andrei Popov
Microsoft Corp.
USA

Email: andreipo@microsoft.com

Magnus Nystroem
Microsoft Corp.
USA

Email: mnystrom@microsoft.com

Dirk Balfanz (editor)
Google Inc.
USA

Email: balfanz@google.com

Adam Langley
Google Inc.
USA

Email: agl@google.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 16, 2015

A. Popov, Ed.
M. Nystroem
Microsoft Corp.
D. Balfanz
A. Langley
Google Inc.

October 13, 2014

The Token Binding Protocol Version 1.0
draft-popov-token-binding-00

Abstract

This document specifies Version 1.0 of the Token Binding protocol. The Token Binding protocol allows client/server applications to create long-lived, uniquely identifiable TLS [RFC5246] bindings spanning multiple TLS sessions and connections. Applications are then enabled to cryptographically bind security tokens to the TLS layer, preventing token export and replay attacks. To protect privacy, the TLS Token Binding identifiers are only transmitted encrypted and can be reset by the user at any time.

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 April 16, 2015.

Table of Contents

1. Introduction

Web services generate various security tokens (e.g. HTTP cookies, OAuth tokens) for web applications to access protected resources. Any party in possession of such token gains access to the protected resource. Attackers export bearer tokens from the user's machine, present them to web services, and impersonate authenticated users. The idea of Token Binding is to prevent such attacks by cryptographically binding security tokens to the TLS layer.

A TLS Token Binding is established by the user agent generating a private-public key pair (possibly within a secure hardware module, such as TPM) per target server, and proving possession of the private key on every TLS connection to the target server. The proof of possession involves signing the tls_unique value [RFC5929] for the TLS connection with the private key. Such TLS Token Binding is identified by the corresponding public key. TLS Token Bindings are long-lived, i.e. they encompass multiple TLS connections and TLS sessions between a given client and server. To protect privacy, TLS

Token Binding identifiers are never transmitted in clear text and can be reset by the user at any time, e.g. when clearing browser cookies.

When issuing a security token to a client that supports TLS Token Binding, a server includes the client’s TLS Token Binding ID in the token. Later on, when a client presents a security token containing a TLS Token Binding ID, the server makes sure the ID in the token matches the ID of the TLS Token Binding established with the client. In the case of a mismatch, the server discards the token.

In order to successfully export and replay a bound security token, the attacker needs to also be able to export the client’s private key, which is hard to do in the case of the key generated in a secure hardware module.

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 [RFC2119].

2. Token Binding Protocol Overview

The client and server use ALPN protocol IDs [RFC7301] to negotiate the use of the Token Binding protocol, in addition to the actual application protocol such as HTTP/1.1 [RFC2616] or HTTP/2 [I-D.ietf-httpbis-http2]. ALPN IDs are also used to negotiate the parameters (signature algorithm, length) of the Token Binding key. This negotiation does not require TLS protocol changes or additional round-trips.

The "IANA Considerations" section of this document defines an initial set of ALPN protocol IDs that allow the use of the Token Binding protocol with HTTP/1.1 and HTTP/2. The initial set of supported key parameters includes ECDSA with NIST P256 curve and 2048-bit RSA. New ALPN protocol IDs can be defined in the future to support Token Binding usage with other application protocols and key parameters.

The Token Binding protocol consists of one message sent by the client to the server, proving possession of one or more client-generated asymmetric keys. This message is only sent if the client and server agree on the use of the Token Binding protocol and the key parameters. The Token Binding message is sent with the application protocol data in TLS application_data records.

A server receiving the Token Binding message verifies that the key parameters in the message match the Token Binding parameters negotiated via ALPN, and then validates the signatures contained in

the Token Binding message. If either of these checks fails, the server terminates the connection, otherwise the TLS Token Binding is successfully established with the ID contained in the Token Binding message.

When a server supporting the Token Binding protocol receives a bound token, the server compares the TLS Token Binding ID in the security token with the TLS Token Binding ID established with the client. If the bound token came from a TLS connection without a Token Binding, or if the IDs don’t match, the token is discarded.

This document describes the negotiation of the Token Binding protocol and key parameters, the format of the Token Binding protocol message, the process of establishing a TLS Token Binding, the format of the Token Binding ID, and the process of validating a security token. Token Binding over HTTP [HTTPSTB] explains how the Token Binding message is encapsulated within application protocol messages. [HTTPSTB] also describes Token Binding between multiple communicating parties: User Agent, Identity Provider and Relying Party.

3. Negotiating the Token Binding Protocol and Key Parameters

The Token Binding protocol is used within TLS connections, in combination with an application protocol such as HTTP/1.1 or HTTP/2. The "IANA Considerations" section of this document defines a set of ALPN protocol IDs that combine application protocol and token binding key parameters:

- "h2_tb_p256" indicates support for HTTP/2 with Token Binding using ECDSA key and NIST P256 curve;

- "h2_tb_rsa2048" indicates support for HTTP/2 with Token Binding using 2048-bit RSA key;

- "http/1.1_tb_p256" indicates support for HTTP/1.1 with Token Binding using ECDSA key and NIST P256 curve;

- "http/1.1_tb_rsa2048" indicates support for HTTP/1.1 with Token Binding using 2048-bit RSA key.

The client advertises support of the Token Binding protocol by sending some of these IDs in the ALPN extension in the ClientHello. Application protocol IDs without Token Binding, such as "http/1.1" and "h2", can also be included for compatibility with the servers that do not support the Token Binding protocol.

The server indicates support of the Token Binding protocol by sending one of the above IDs in the ALPN extension in the ServerHello. The

server implements the protocol selection logic as described in section 3.2 "Protocol Selection" of [RFC7301], taking into account the application protocols and key parameters supported by the server.

4. Token Binding Protocol Message

The Token Binding message is sent by the client and proves possession of one or more private keys held by the client. This message MUST be sent if the client and server successfully negotiated the use of the Token Binding protocol via ALPN, and MUST NOT be sent otherwise. This message MUST be sent in the client's first application protocol message. This message MAY also be sent in subsequent application protocol messages, proving possession of other keys by the same client, to facilitate token binding between more than two communicating parties. Token Binding over HTTP [HTTPSTB] specifies the encapsulation of the Token Binding message in the application protocol messages, and the scenarios involving more than two communicating parties. The Token Binding message format is defined using TLS specification language, and reuses existing TLS structures and IANA registrations where possible:

```
enum {
    sha256(4), (255)
} HashAlgorithm;

enum {
    rsa(1), ecdsap256(3), (255)
} SignatureAlgorithm;

struct {
    HashAlgorithm hash;
    SignatureAlgorithm signature;
} SignatureAndHashAlgorithm;

struct {
    opaque modulus<1..2^16-1>;
    opaque publicexponent<1..2^8-1>;
} RSAPublicKey;

enum {
    secp256r1 (23), (0xFFFF)
} NamedCurve;

struct {
    opaque point <1..2^8-1>;
} ECPoint;
```

```
struct {
    NamedCurve namedcurve;
    ECPoint point;      // Uncompressed format
} ECDSAParams;

enum {
    provided_token_binding(0), referred_token_binding(1), (255)
} TokenBindingType;

struct {
    TokenBindingType tokenbinding_type;
    SignatureAndHashAlgorithm algorithm;
    select (algorithm.signature) {
        case rsa: RSAPublicKey rsapubkey;
        case ecdsa: ECDSAParams ecdsaparams;
    }
} TokenBindingID;

enum {
    (255)           // No initial ExtensionType registrations
} ExtensionType;

struct {
    ExtensionType extension_type;
    opaque extension_data<0..2^16-1>;
} Extension;

struct {
    TokenBindingID tokenbindingid;
    opaque signature<0..2^16-1>;// Signature over hashed ("token binding", tls_u
nique)
    Extension extensions<0..2^16-1>;
} TokenBinding;

struct {
    TokenBinding tokenbindings<0..2^16-1>;
} TokenBindingMessage;
```

The Token Binding message consists of a series of TokenBinding structures containing the TokenBindingID, a signature over the hash of the NUL-terminated, ASCII label ("token binding") and the tls_unique, optionally followed by Extension structures. An implementation MUST ignore any unknown extensions. Initially, no extension types are defined. At least one TokenBinding MUST be included in the Token Binding message. The signature algorithm and key length used in the TokenBinding MUST match the parameters negotiated via ALPN. The client SHOULD generate and store Token Binding keys in a secure manner that prevents key export. In order

to prevent cooperating servers from linking user identities, different keys SHOULD be used by the client for connections to different servers, according to the token scoping rules of the application protocol.

5. Establishing a TLS Token Binding

Triple handshake vulnerability in the TLS protocol affects the security of the Token Binding protocol, as described in the "Security Considerations" section below. Therefore, the server MUST NOT negotiate the use of the Token Binding protocol unless the server also negotiates Extended Master Secret TLS extension [I-D.ietf-tls-session-hash].

The server MUST terminate the connection if the use of the Token Binding protocol has been successfully negotiated via ALPN within the TLS handshake, but the client’s first application message does not contain the Token Binding message. The server MUST terminate the connection if the use of the Token Binding protocol was not negotiated, but the client sends the Token Binding message.

If the Token Binding type is "provided_token_binding", the server MUST verify that the signature algorithm (including elliptic curve in the case of ECDSA) and key length in the Token Binding message match those negotiated via ALPN. In the case of a mismatch, the server MUST terminate the connection. As described in [HTTPSTB], Token Bindings of type "referred_token_binding" may have different key parameters than those negotiated via ALPN.

If the Token Binding message does not contain at least one TokenBinding structure, or the signature contained in a TokenBinding structure is invalid, the server MUST terminate the connection. Otherwise, the TLS Token Binding is successfully established and its ID can be provided to the application for security token validation.

6. TLS Token Binding ID Format

The ID of the TLS Token Binding established as a result of Token Binding message processing is a binary representation of the following structure:

```
struct {
    TokenBindingType tokenbinding_type;
    SignatureAndHashAlgorithm algorithm;
    select (algorithm.signature) {
        case rsa: RSAPublicKey rsapubkey;
        case ecdsa: ECDSAParams ecdsaparams;
    }
} TokenBindingID;
```

TokenBindingID includes the type of the token binding and the key parameters negotiated via ALPN. This document defines two token binding types: provided_token_binding used to establish a Token Binding when connecting to a server, and referred_token_binding used when requesting tokens to be presented to a different server. Token Binding over HTTP [HTTPSTB] describes Token Binding between multiple communicating parties: User Agent, Identity Provider and Relying Party. TLS Token Binding ID can be obtained from the TokenBinding structure described in the "Token Binding Protocol Message" section of this document by discarding the signature and extensions. TLS Token Binding ID will be available at the application layer and used by the server to generate and verify bound tokens.

7. Security Token Validation

Security tokens can be bound to the TLS layer either by embedding the Token Binding ID in the token, or by maintaining a database mapping tokens to Token Binding IDs. The specific method of generating bound security tokens is application-defined and beyond the scope of this document.

Upon receipt of a security token, the server attempts to retrieve TLS Token Binding ID information from the token and from the TLS connection with the client. Application-provided policy determines whether to honor non-bound (bearer) tokens. If the token is bound and a TLS Token Binding has not been established for the client connection, the server MUST discard the token. If the TLS Token Binding ID for the token does not match the TLS Token Binding ID established for the client connection, the server MUST discard the token.

8. IANA Considerations

This document establishes a registry for Token Binding type identifiers entitled "Token Binding Types" under the "Token Binding Protocol" heading.

Entries in this registry require the following fields:

- o Value: The octet value that identifies the Token Binding type (0-255).
- o Description: The description of the Token Binding type.
- o Specification: A reference to a specification that defines the Token Binding type.

This registry operates under the "Expert Review" policy as defined in [RFC5226]. The designated expert is advised to encourage the inclusion of a reference to a permanent and readily available specification that enables the creation of interoperable implementations using the identified Token Binding type.

An initial set of registrations for this registry follows:

Value: 0

Description: provided_token_binding

Specification: this document

Value: 1

Description: referred_token_binding

Specification: this document

This document establishes a registry for Token Binding extensions entitled "Token Binding Extensions" under the "Token Binding Protocol" heading.

Entries in this registry require the following fields:

- o Value: The octet value that identifies the Token Binding extension (0-255).
- o Description: The description of the Token Binding extension.

- o Specification: A reference to a specification that defines the Token Binding extension.

This registry operates under the "Expert Review" policy as defined in [RFC5226]. The designated expert is advised to encourage the inclusion of a reference to a permanent and readily available specification that enables the creation of interoperable implementations using the identified Token Binding extension. This document creates no initial registrations in the "Token Binding Extensions" registry.

This document creates the following registrations for the identification of the Token Binding protocol in the "Application Layer Protocol Negotiation (ALPN) Protocol IDs" registry originally created in [RFC7301]:

Protocol: HTTP/2 with Token Binding using ECDSA key and NIST P256 curve

Identification Sequence: 0x68 0x32 0x5f 0x74 0x62 0x5f 0x70 0x32 0x35 0x36 ("h2_tb_p256")

Specification: this document

Protocol: HTTP/2 with Token Binding using 2048-bit RSA key

Identification Sequence: 0x68 0x32 0x5f 0x74 0x62 0x5f 0x72 0x73 0x61 0x32 0x30 0x34 0x38 ("h2_tb_rsa2048")

Specification: this document

Protocol: HTTP/1.1 with Token Binding using ECDSA key and NIST P256 curve

Identification Sequence: 0x68 0x74 0x74 0x70 0x2f 0x31 0x2e 0x31 0x5f 0x74 0x62 0x5f 0x70 0x32 0x35 0x36 ("http/1.1_tb_p256")

Specification: this document

Protocol: HTTP/1.1 with Token Binding using 2048-bit RSA key

Identification Sequence: 0x68 0x74 0x74 0x70 0x2f 0x31 0x2e 0x31 0x5f 0x74 0x62 0x5f 0x72 0x73 0x61 0x32 0x30 0x34 0x38 ("http/1.1_tb_rsa2048")

Specification: this document

This document uses "TLS SignatureAlgorithm" and "TLS HashAlgorithm" registries originally created in [RFC5246], and "TLS NamedCurve" registry originally created in [RFC4492]. This document creates no new registrations in these registries.

9. Security Considerations

9.1. Security Token Replay

The goal of the Token Binding protocol is to prevent attackers from exporting and replaying security tokens, thereby impersonating legitimate users and gaining access to protected resources. Bound tokens can still be replayed by the malware present in the User Agent. In order to export the token to another machine and successfully replay it, the attacker also needs to export the corresponding private key. Token Binding private keys are therefore high-value assets and SHOULD be strongly protected, ideally by generating them in a hardware security module that prevents key export.

9.2. Downgrade Attacks

The Token Binding protocol is only used when negotiated via ALPN within the TLS handshake. TLS prevents active attackers from modifying the messages of the TLS handshake, therefore it is not possible for the attacker to remove or modify the ALPN IDs used to negotiate the Token Binding protocol and key parameters. The signature algorithm and key length used in the TokenBinding of type "provided_token_binding" MUST match the parameters negotiated via ALPN.

9.3. Privacy Considerations

The Token Binding protocol uses persistent, long-lived TLS Token Binding IDs. To protect privacy, TLS Token Binding IDs are never transmitted in clear text and can be reset by the user at any time, e.g. when clearing browser cookies. In order to prevent cooperating servers from linking user identities, different keys SHOULD be used by the client for connections to different servers, according to the token scoping rules of the application protocol.

9.4. Triple Handshake Vulnerability in TLS

The Token Binding protocol relies on the tls_unique value to associate a TLS connection with a TLS Token Binding. The triple handshake attack [TRIPLE-HS] is a known TLS protocol vulnerability allowing the attacker to synchronize tls_unique values between TLS connections. The attacker can then successfully replay bound tokens.

For this reason, the Token Binding protocol MUST NOT be negotiated unless the Extended Master Secret TLS extension [I-D.ietf-tls-session-hash] has also been negotiated.

10. References

10.1. Normative References

[HTTPSTB] Balfanz, D., Langley, A., Popov, A., and M. Nystroem, "Token Binding over HTTP", 2014.

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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May 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.

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, July 2010.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, July 2014.

10.2. Informative References

[I-D.ietf-httpbis-http2]
Belshe, M., Peon, R., and M. Thomson, "Hypertext Transfer Protocol version 2", draft-ietf-httpbis-http2-14 (work in progress), July 2014.

[I-D.ietf-tls-session-hash]
Bhargavan, K., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", draft-ietf-tls-session-hash-02 (work in progress), October 2014.

[TRIPLE-HS]
Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Pironti, A., and P. Strub, "Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS. IEEE Symposium on Security and Privacy", 2014.

Authors' Addresses

Andrei Popov (editor)
Microsoft Corp.
USA

Email: andreipo@microsoft.com

Magnus Nystroem
Microsoft Corp.
USA

Email: mnystrom@microsoft.com

Dirk Balfanz
Google Inc.
USA

Email: balfanz@google.com

Adam Langley
Google Inc.
USA

Email: agl@google.com