OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2017

B. Campbell
J. Bradley
Ping Identity
M. Jones
Microsoft

July 8, 2016

A Token Binding method for OAuth 2.0 Proof Key for Code Exchange draft-campbell-oauth-tbpkce-00

Abstract

This specification describes a Proof Key for Code Exchange (PKCE) [RFC7636] method utilizing Token Binding over HTTP [I-D.ietf-tokbind-https] to cryptographically bind the OAuth 2.0 [RFC6749] authorization code to a key pair on the client, which it proves possession of during the access token request with the authorization code.

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 9, 2017.

Copyright Notice

Table of Contents

1. Introduction

This specification minimally describes an OAuth 2.0 [RFC6749] PKCE [RFC7636] method based on the Token Binding Protocol [I-D.ietf-tokbind-protocol] and Token Binding over HTTP [I-D.ietf-tokbind-https]. The general details and motivations of PKCE are discussed in that document and this specification defines only the additional pieces needed for a Token Binding PKCE method.

1.1. Requirements Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Terminology

This specification uses the terms "authorization code", "authorization endpoint", "authorization server", "authorization request", "access token request", "client", and "token endpoint" defined by OAuth 2.0 [RFC6749], and the terms "Provided", "Token Binding" and "Token Binding ID" defined by Token Binding over HTTP [I-D.ietf-tokbind-https].

2. Code Challenge

As defined in Proof Key for Code Exchange [RFC7636], the client sends the code challenge as part of the OAuth 2.0 Authorization Request with the two additional parameters: "code_challenge" and "code_challenge_method".

For the Token Binding method of PKCE defined herein, "tb2" is used for the value of the "code_challenge_method" parameter.

The value of the "code_challenge" parameter is the base64url encoding (per Section 5 of [RFC4648] with all trailing pad (’=’) characters omitted and without the inclusion of any line breaks or whitespace) of the SHA-256 hash [RFC6234] of the Provided Token Binding ID that the client will use when calling the authorization server’s token endpoint. Note that, prior to making the authorization request, the client may need to establish a TLS connection between itself and the authorization server’s token endpoint in order to obtain the appropriate Token Binding ID.

When the authorization server issues the authorization code in the authorization response, it associates the code challenge and method values with the authorization code so it can be verified later when the code is presented in the access token request.

3. Code Verifier

Upon receipt of the authorization code, the client sends the access token request to the token endpoint. The Token Binding Protocol [I-D.ietf-tokbind-protocol] is negotiated on the TLS connection between the client and the authorization server and the "Sec-Token-Binding" header, as defined in Token Binding over HTTP [I-D.ietf-tokbind-https], is included in the access token request. The authorization server extracts the Provided Token Binding ID from the header value, hashes it with SHA-256, and compares it to the "code_challenge" value previously associated with the authorization code. If the values match, the token endpoint MUST continue processing as normal (as defined by OAuth 2.0 [RFC6749]). If the values do not match, an error response indicating "invalid_grant" MUST be returned.

The "Sec-Token-Binding" header contains sufficient information for verification of the authorization code and its association to the original authorization request. However, PKCE [RFC7636] requires that a "code_verifier" parameter be sent with the access token request, so the static value "provided" is used to meet that requirement and indicate that the Provided Token Binding ID is used for the verification.

4. Security Considerations

TBD

5. IANA Considerations

5.1. PKCE Code Challenge Method Registration

This specification requests registration of the following Code Challenge Method Parameter Name in the IANA "PKCE Code Challenge Methods" registry [IANA.OAuth.Parameters] established by [RFC7636].

5.1.1. Registry Contents

- Code Challenge Method Parameter Name: tb2
- Change controller: IESG
- Specification document(s): Section 2 of [[this specification]]

6. Normative References

[I-D.ietf-tokbind-https]
Popov, A., Nystrom, M., Balfanz, D., Langley, A., and J. Hodges, "Token Binding over HTTP", draft-ietf-tokbind-https-03 (work in progress), March 2016.

[I-D.ietf-tokbind-protocol]
Popov, A., Nystrom, M., Balfanz, D., Langley, A., and J. Hodges, "The Token Binding Protocol Version 1.0", draft-ietf-tokbind-protocol-06 (work in progress), May 2016.

[IANA.OAuth.Parameters]
IANA, "OAuth Parameters", <http://www.iana.org/assignments/oauth-parameters>.

[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>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <http://www.rfc-editor.org/info/rfc6234>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <http://www.rfc-editor.org/info/rfc6749>.

[RFC7636] Sakimura, N., Ed., Bradley, J., and N. Agarwal, "Proof Key for Code Exchange by OAuth Public Clients", RFC 7636, DOI 10.17487/RFC7636, September 2015, <http://www.rfc-editor.org/info/rfc7636>.

Appendix A. Acknowledgements

Dirk Balfanz, William Dennis (and others?) also provided input to this specification.

Appendix B. Document History

[[to be removed by the RFC Editor before publication as an RFC]]

draft-campbell-oauth-tbpkce-00

- Initial version.

Authors' Addresses

Brian Campbell
Ping Identity

Email: brian.d.campbell@gmail.com
URI: https://twitter.com/__b_c

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com
URI: http://www.thread-safe.com/

Michael B. Jones
Microsoft

Email: mbj@microsoft.com
URI: http://self-issued.info/

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 8, 2017

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

July 7, 2016

Token Binding over HTTP
draft-ietf-tokbind-https-05

Abstract

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

We describe both _first-party_ and _federated_ scenarios. In a first-party scenario, an HTTP server is able to cryptographically bind the security tokens it issues to a client, and which the client subsequently returns to the server, to the TLS connection between the client and server. Such bound security tokens are protected from misuse since the server can generally detect if they are replayed inappropriately, e.g., over other TLS connections.

Federated token bindings, on the other hand, allow servers to cryptographically bind security tokens to a TLS 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 [I-D.ietf-tokbind-protocol]

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 8, 2017.

Table of Contents

1. Introduction

The Token Binding Protocol [I-D.ietf-tokbind-protocol] 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 abuse (re-use, attempted impersonation, etc.) by attackers.

While the Token Binding Protocol [I-D.ietf-tokbind-protocol] defines a message format for establishing a Token Binding ID, it does not 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 [RFC7230] and 2 [RFC7540]). 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 field in HTTP requests. The HTTP header field value is a base64url-encoded 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 Sec-Token-Binding Header Field

Once a client and server have negotiated the Token Binding Protocol with HTTP/1.1 or HTTP/2 (see [I-D.ietf-tokbind-protocol] and [I-D.ietf-tokbind-negotiation]), clients MUST include the Sec-Token-Binding header field in their HTTP requests. The ABNF of the Sec-Token-Binding header field is (in [RFC7230] style, see also [RFC7231] Section 8.3):

```
Sec-Token-Binding = EncodedTokenBindingMessage
```

The header field name is "Sec-Token-Binding", and EncodedTokenBindingMessage is a base64url encoding (see [RFC4648] Section 5) of the TokenBindingMessage as defined in [I-D.ietf-tokbind-protocol].

For example:

```
Sec-Token-Binding: <base64url-encoded TokenBindingMessage>
```

The TokenBindingMessage MUST contain one TokenBinding structure with TokenBindingType of provided_token_binding, which MUST be signed with the Token Binding private 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, see Section 2.1 below). The Token Binding ID established by this TokenBinding is called a _Provided Token Binding ID_.

The TokenBindingMessage MAY also contain one TokenBinding structure with TokenBindingType of referred_token_binding, as specified in Section 4.3. In addition to the latter, or rather than the latter, the TokenBindingMessage MAY contain other TokenBinding structures. This is use case-specific, and such use cases are outside the scope of this specification.

In HTTP/2, the client SHOULD use Header Compression [RFC7541] to avoid the overhead of repeating the same header field in subsequent HTTP requests.

2.1. HTTPS Token Binding Key Pair Scoping

HTTPS is used in conjunction with various application protocols, and application contexts, in various ways. For example, general purpose Web browsing is one such HTTP-based application context. Within the latter context, HTTP cookies [RFC6265] are typically utilized for state management, including client authentication. A related, though distinct, example of other HTTP-based application contexts is where OAuth tokens [RFC6749] are utilized to manage authorization for

third-party application access to resources. The token scoping rules of these two examples can differ: the scoping rules for cookies are concisely specified in [RFC6265], whereas OAuth is a framework and defines various token types with various scopings, some of which are determined by the encompassing application.

The Token Binding key pair scoping for those key pairs generated in the context of the first-party and federation use cases defined in this specification (below), and to be used for binding HTTP cookies MUST be at the granularity of "effective top-level domain (public suffix) + 1" (eTLD+1), i.e., at the same granularity at which cookies can be set (see [RFC6265]). Key pairs used to bind other application tokens, such as OAuth tokens, SHOULD adhere to the above eTLD+1 scoping requirement for those tokens being employed in first-party or federation scenarios as described below, e.g., OAuth refresh tokens or Open ID Connect "ID Tokens". See also Section 6.1, below.

Scoping rules for other HTTP-based application contexts are outside the scope of this specification.

3. First-party Use Cases

In a first-party use case, an HTTP server issues a security token such as a cookie (or similar) to a client, and expects the client to return the security token at a later time, e.g., in order to authenticate. Binding the security token to the TLS connection between client and server protects the security token from misuse since the server can detect if the security token is replayed inappropriately, e.g., over other TLS connections.

See [I-D.ietf-tokbind-protocol] Section 6 for general guidance regarding binding of security tokens and their subsequent validation.

4. Federation Use Cases

4.1. 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) signals to the client that it should 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 [I-D.ietf-tokbind-protocol], and includes a TokenBinding structure in the Sec-Token-Binding HTTP header field defined above. What differs between the various mechanisms is _how_ the Token Consumer signals to the client that it should reveal the Token Binding ID to the Token Provider. Below we specify one such mechanism, which is suitable for redirect-based interactions between Token Consumers and Token Providers.

4.2. 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 and the client.

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

- 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.

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

- 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.

- Therefore, in the previous step, the Token Provider may want to include in the token the Token-Binding public key that the client

uses when communicating with the Token Consumer, thus _binding_ the token to client's Token-Binding keypair. The client proves possession of the private key when communicating with the Token Consumer through the Token Binding Protocol [I-D.ietf-tokbind-protocol], and reveals the corresponding public key of this keypair as part of the Token Binding ID. Comparing the public key from the token with the public key from the Token Binding ID allows the Token Consumer to verify that the token was sent to it by the legitimate client.

- To allow the Token Provider to include the Token-Binding public key in the token, the Token Binding ID (between client and Token Consumer) 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 [I-D.ietf-tokbind-protocol].

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 requests the client to do so.

Below, we specify how Token Consumers can signal this request in redirect-based federation protocols. Note that this assumes that the federated sign-on flow starts at the Token Consumer, or at the very least include a redirect from Token Consumer to Token Provider. It is outside the scope of this document to specify similar mechanisms for flows that do not include such redirects.

4.3. HTTP Redirects

When a Token Consumer redirects the client to a Token Provider as a means to deliver the token request, it SHOULD include a Include-Referred-Token-Binding-ID HTTP response header field in its HTTP response. The ABNF of the Include-Referred-Token-Binding-ID header is (in [RFC7230] style, see also [RFC7231] Section 8.3):

```
Include-Referred-Token-Binding-ID = "true"
```

Where the header field name is "Include-Referred-Token-Binding-ID", and the field-value of "true" is case-insensitive. For example:

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

Including this response header field signals to the client that it should reveal, to the Token Provider, the Token Binding ID used between itself and the Token Consumer. In the absence of this

response header field, the client will not disclose any information about the Token Binding used between the client and the Token Consumer to the Token Provider.

As illustrated in Section 4.5, when a client receives this header field, it should take the TokenBindingID of the provided TokenBinding from the referrer and create a referred TokenBinding with it to include in the TokenBindingMessage on the redirect request. In other words, the Token Binding message in the redirect request to the Token Provider now includes one provided binding and one referred binding, the latter constructed from the binding between the client and the Token Consumer. Note that that the referred token binding is sent only on the request resulting from the redirect and not on any subsequent requests to the Token Provider

If the Include-Referred-Token-Binding-ID header field is received in response to a request that did not include the Token-Binding header field, the client MUST ignore the Include-Referred-Token-Binding-ID header field.

This header field has only meaning if the HTTP status code is 301, 302, 303, 307 or 308, 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 Sec-Token-Binding header field to the Token Provider with each HTTP request (see above).

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-Referred-Token-Binding-ID response header field). 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 Provider (more specifically, the web origin that the token request is being sent to).

If for some deployment-specific reason the initial Token Provider ("TP1") needs to redirect the client to another Token Provider ("TP2"), rather than directly back to the Token Consumer, it can be accomodated using the header fields defined in this specification in the following fashion ("the redirect-chain approach"):

Initially, the client is redirected to TP1 by the Token Consumer ("TC"), as described above. Upon receiving the client's request, containing a TokenBindingMessage which contains both provided and referred TokenBindings (for TP1 and TC, respectively), TP1 responds to the client with a redirect response containing the Include-Referred-Token-Binding-ID header field and directing the client to send a request to TP2. This causes the client to follow the same pattern and send a request containing a TokenBindingMessage which contains both provided and referred TokenBindings (for TP2 and TP1, respectively) to TP2. Note that this pattern can continue to further Token Providers. In this case, TP2 issues a security token, bound to the client's TokenBinding with TP1, and sends a redirect response to the client pointing to TP1. TP1 in turn constructs a security token for the Token Consumer, bound to the TC's referred TokenBinding which had been conveyed earlier, and sends a redirect response pointing to the TC, containing the bound security token, to the client.

The above is intended as only a non-normative example. Details are specific to deployment contexts. Other approaches are possible, but are outside the scope of this specification.

4.4. Negotiated Key Parameters

The TLS Extension for Token Binding Protocol Negotiation [I-D.ietf-tokbind-negotiation] allows the server and client to negotiate the parameters (signature algorithm, length) of the Token Binding key. 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 key parameters. The client MUST use the key parameters negotiated with the Token Consumer in the referred_token_binding TokenBinding of the TokenBindingMessage, even if those key parameters are different from the ones negotiated with the origin that the header field is sent to.

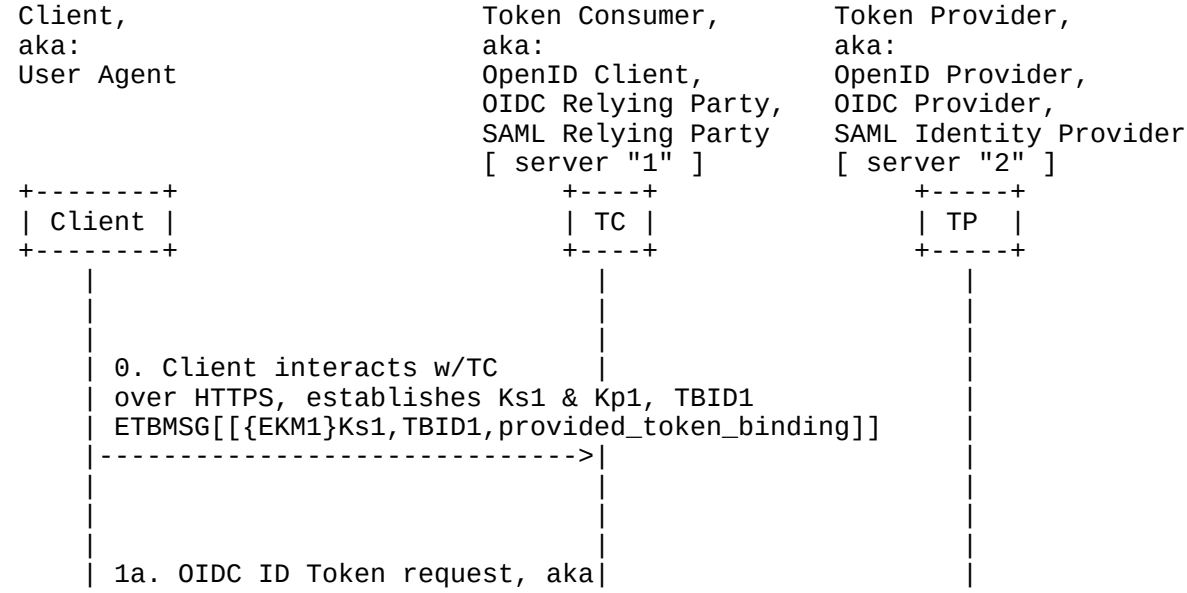
Token Providers SHOULD support all the Token Binding key parameters specified in the [I-D.ietf-tokbind-protocol]. If a token provider does not support the key parameters specified in the referred_token_binding TokenBinding in the TokenBindingMessage, it MUST issue an unbound token.

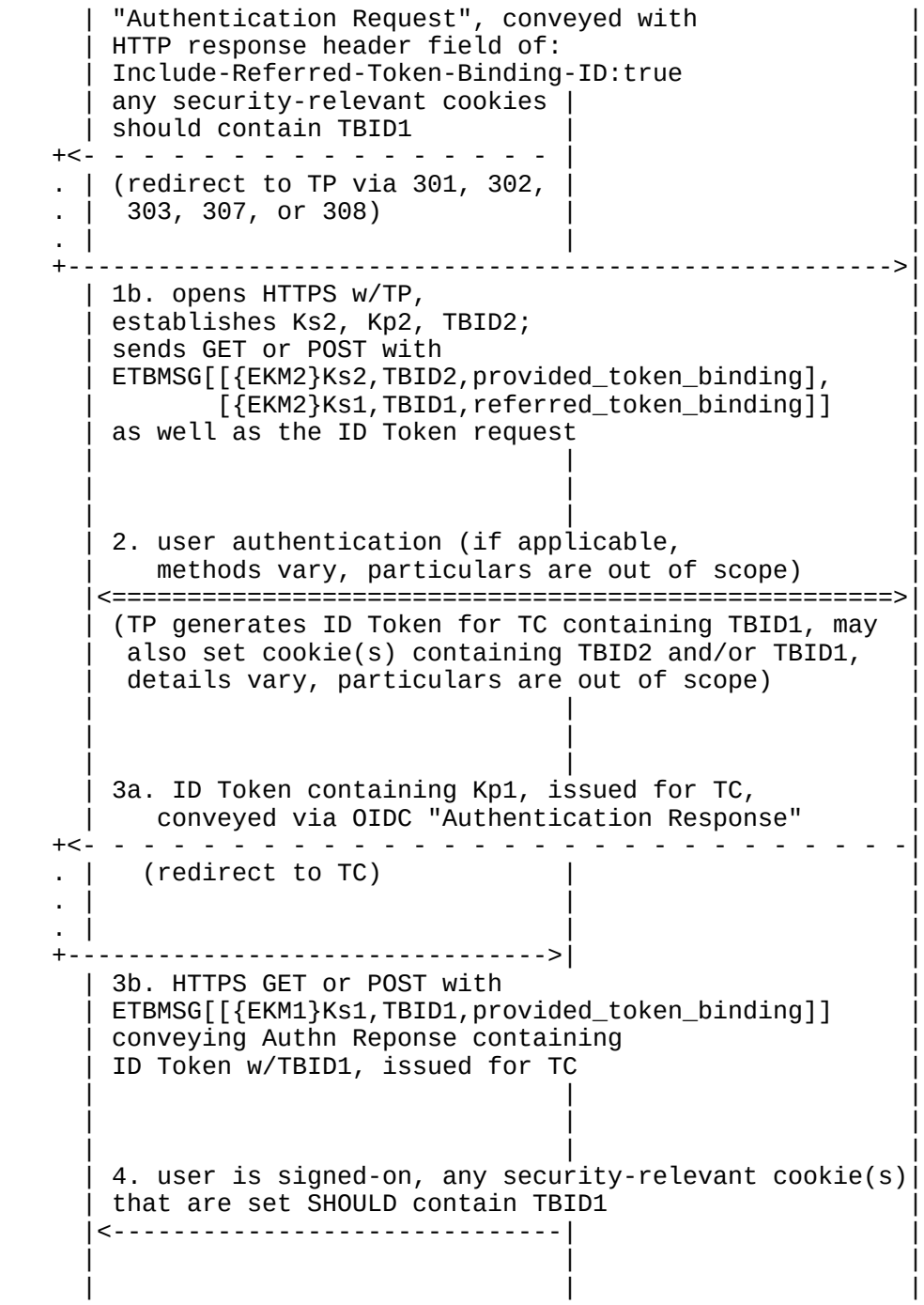
4.5. Federation Example

The diagram below shows a typical HTTP Redirect-based Web Browser SSO Profile (no artifact, no callbacks), featuring binding of, e.g., a TLS Token Binding ID into an OpenID Connect "ID Token".

Legend:

EKM:	TLS Exported Keying Material [RFC5705]
{EKMn}Ksm:	EKM for server "n", signed by private key of TBID "m", where "n" must represent server receiving the ETBMSG, if a conveyed TB's type is provided_token_binding, then m = n, else if TB's type is referred_token_binding, then m != n. E.g., see step 1b in diagram below.
ETBMSG:	"Sec-Token-Binding" HTTP header field conveying an EncodedTokenBindingMessage, in turn conveying TokenBinding (TB)struct(s), e.g.: ETBMSG[[TB]] or ETBMSG[[TB1],[TB2]]
ID Token:	the "ID Token" in OIDC, it is the semantic equivalent of a SAML "authentication assertion". "ID Token w/TBIDn" denotes a "token bound" ID Token containing TBIDn.
Ks & Kp:	private (aka secret) key, and public key, respectively, of client-side Token Binding key pair
OIDC:	Open ID Connect
TB:	TokenBinding struct containing signed EKM, TBID, and TB type, e.g.: [{EKM1}Ks1,TBID1,provided_token_binding]
TBIDn:	Token Binding ID for client and server n's token-bound TLS association. TBIDn contains Kpn.




"Authentication Request", conveyed with
HTTP response header field of:
Include-Referred-Token-Binding-ID:true
any security-relevant cookies
should contain TBID1
(redirect to TP via 301, 302,
303, 307, or 308)
1b. opens HTTPS w/TP,
establishes Ks2, Kp2, TBID2;
sends GET or POST with
ETBMSG[[{EKM2}Ks2,TBID2,provided_token_binding],
[{EKM2}Ks1,TBID1,referred_token_binding]]
as well as the ID Token request
2. user authentication (if applicable,
methods vary, particulars are out of scope)
(TP generates ID Token for TC containing TBID1, may
also set cookie(s) containing TBID2 and/or TBID1,
details vary, particulars are out of scope)
3a. ID Token containing Kp1, issued for TC,
conveyed via OIDC "Authentication Response"
(redirect to TC)
3b. HTTPS GET or POST with
ETBMSG[[{EKM1}Ks1,TBID1,provided_token_binding]]
conveying Authn Reponse containing
ID Token w/TBID1, issued for TC
4. user is signed-on, any security-relevant cookie(s)
that are set SHOULD contain TBID1

5. Security Considerations

5.1. 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.

5.2. Triple Handshake Vulnerability in TLS 1.2 and Older TLS Versions

The Token Binding protocol relies on the exported key material (EKM) value [RFC5705] to associate a TLS connection with a TLS Token Binding. The triple handshake attack [TRIPLE-HS] is a known vulnerability in TLS 1.2 and older TLS versions, allowing the attacker to synchronize keying material between TLS connections. The attacker can then successfully replay bound tokens. For this reason, the Token Binding protocol MUST NOT be negotiated with these TLS versions, unless the Extended Master Secret [RFC7627] and Renegotiation Indication [RFC5746] TLS extensions have also been negotiated.

5.3. Sensitivity of the Sec-Token-Binding Header

The purpose of the Token Binding protocol is to convince the server that the client that initiated the TLS connection controls a certain key pair. For the server to correctly draw this conclusion after processing the Sec-Token-Binding header field, certain secrecy and integrity requirements must be met.

For example, the client’s private Token Binding key must be kept secret by the client. If the private key is not secret, then another actor in the system could create a valid Token Binding header field, impersonating the client. This can render the main purpose of the protocol - to bind bearer tokens to certain clients - moot: Consider, for example, an attacker who obtained (perhaps through a network intrusion) an authentication cookie that a client uses with a certain server. Consider further that the server bound that cookie to the client’s Token Binding ID precisely to thwart misuse of the cookie. If the attacker were to come into possession of the client’s private key, he could then establish a TLS connection with the server and craft a Sec-Token-Binding header field that matches the binding

present in the cookie, thus successfully authenticating as the client, and gaining access to the client's data at the server. The Token Binding protocol, in this case, did not successfully bind the cookie to the client.

Likewise, we need integrity protection of the Sec-Token-Binding header field: A client should not be tricked into sending a Sec-Token-Binding header field to a server that contains Token Binding messages about key pairs that the client does not control. Consider an attacker A that somehow has knowledge of the exported keying material (EKM) for a TLS connection between a client C and a server S. (While that is somewhat unlikely, it is also not entirely out of the question, since the client might not treat the EKM as a secret - after all, a pre-image-resistant hash function has been applied to the TLS master secret, making it impossible for someone knowing the EKM to recover the TLS master secret. Such considerations might lead some clients to not treat the EKM as a secret.) Such an attacker A could craft a Sec-Token-Binding header field with A's key pair over C's EKM. If the attacker could now trick C to send such a header field to S, it would appear to S as if C controls a certain key pair when in fact it does not (the attacker A controls the key pair).

If A has a pre-existing relationship with S (perhaps has an account on S), it now appears to the server S as if A is connecting to it, even though it is really C. (If the server S does not simply use Token Binding keys to identify clients, but also uses bound authentication cookies, then A would also have to trick C into sending one of A's cookies to S, which it can do through a variety of means - inserting cookies through Javascript APIs, setting cookies through related-domain attacks, etc.) In other words, A tricked C into logging into A's account on S. This could lead to a loss of privacy for C, since A presumably has some other way to also access the account, and can thus indirectly observe A's behavior (for example, if S has a feature that lets account holders see their activity history on S).

Therefore, we need to protect the integrity of the Sec-Token-Binding header field. One origin should not be able to set the Sec-Token-Binding header field (through a DOM API or otherwise) that the User Agent uses with another origin. Employing the "Sec-" header field prefix helps to meet this requirement by denoting the header field name to be a "forbidden header name", see [fetch-spec].

5.4. Securing Federated Sign-On Protocols

As explained above, in a federated sign-in scenario a client will prove possession of two different key pairs to a Token Provider: One key pair is the "provided" Token Binding key pair (which the client

normally uses with the Token Provider), and the other is the "referred" Token Binding key pair (which the client normally uses with the Token Consumer). The Token Provider is expected to issue a token that is bound to the referred Token Binding key.

Both proofs (that of the provided Token Binding key and that of the referred Token Binding key) are necessary. To show this, consider the following scenario:

- The client has an authentication token with the Token Provider that is bound to the client's Token Binding key.
- The client wants to establish a secure (i.e., free of men-in-the-middle) authenticated session with the Token Consumer, but has not done so yet (in other words, we are about to run the federated sign-on protocol).
- A man-in-the-middle is allowed to intercept the connection between client and Token Consumer or between Client and Token Provider (or both).

The goal is to detect the presence of the man-in-the-middle in these scenarios.

First, consider a man-in-the-middle between the client and the Token Provider. Recall that we assume that the client possesses a bound authentication token (e.g., cookie) for the Token Provider. The man-in-the-middle can intercept and modify any message sent by the client to the Token Provider, and any message sent by the Token Provider to the client. (This means, among other things, that the man-in-the-middle controls the Javascript running at the client in the origin of the Token Provider.) It is not, however, in possession of the client's Token Binding key. Therefore, it can either choose to replace the Token Binding key in requests from the client to the Token Provider, and create a Sec-Token-Binding header field that matches the TLS connection between the man-in-the-middle and the Token Provider; or it can choose to leave the Sec-Token-Binding header field unchanged. If it chooses the latter, the signature in the Token Binding message (created by the original client on the exported keying material (EKM) for the connection between client and man-in-the-middle) will not match the EKM between man-in-the-middle and the Token Provider. If it chooses the former (and creates its own signature, with its own Token Binding key, over the EKM for the connection between man-in-the-middle and Token Provider), then the Token Binding message will match the connection between man-in-the-middle and Token Provider, but the Token Binding key in the message will not match the Token Binding key that the client's authentication token is bound to. Either way, the man-in-the-middle is detected by

the Token Provider, but only if the proof of key possession of the provided Token Binding key is required in the protocol (as we do above).

Next, consider the presence of a man-in-the-middle between client and Token Consumer. That man-in-the-middle can intercept and modify any message sent by the client to the Token Consumer, and any message sent by the Token Consumer to the client. The Token Consumer is the party that redirects the client to the Token Provider. In this case, the man-in-the-middle controls the redirect URL, and can tamper with any redirect URL issued by the Token Consumer (as well as with any Javascript running in the origin of the Token Consumer). The goal of the man-in-the-middle is to trick the Token Issuer to issue a token bound to _its_ Token Binding key, not to the Token Binding key of the legitimate client. To thwart this goal of the man-in-the-middle, the client's referred Token Binding key must be communicated to the Token Producer in a manner that can not be affected by the man-in-the-middle (who, as we recall, can modify redirect URLs and Javascript at the client). Including the referred Token Binding message in the Sec-Token-Binding header field (as opposed to, say, including the referred Token Binding key in an application-level message as part of the redirect URL) is one way to assure that the man-in-the-middle between client and Token Consumer cannot affect the communication of the referred Token Binding key to the Token Provider.

Therefore, the Sec-Token-Binding header field in the federated sign-on use case contains both, a proof of possession of the provided Token Binding key, as well as a proof of possession of the referred Token Binding key.

6. Privacy Considerations

6.1. Scoping of Token Binding Keys

Clients use different Token Binding key pairs for different servers, so as to not allow Token Binding to become a tracking tool across different servers. However, the scoping of the Token Binding key pairs to servers varies according to the scoping rules of the application protocol ([I-D.ietf-tokbind-protocol] section 4.1).

In the case of HTTP cookies, servers may use Token Binding to secure their cookies. These cookies can be attached to any sub-domain of effective top-level domains, and clients therefore should use the same Token Binding key across such subdomains. This will ensure that any server capable of receiving the cookie will see the same Token Binding ID from the client, and thus be able to verify the token binding of the cookie. See Section 2.1, above.

6.2. Life Time of Token Binding Keys

Token Binding keys do not have an expiration time. This means that they can potentially be used by a server to track a user across an extended period of time (similar to a long-lived cookie). HTTPS clients such as web user agents should therefore provide a user interface for discarding Token Binding keys (similar to the affordances provided to delete cookies).

If a user agent provides modes such as private browsing mode in which the user is promised that browsing state such as cookies are discarded after the session is over, the user agent should also discard Token Binding keys from such modes after the session is over. Generally speaking, users should be given the same level of control over life time of Token Binding keys as they have over cookies or other potential tracking mechanisms.

7. IANA Considerations

Below are the Internet Assigned Numbers Authority (IANA) Permanent Message Header Field registration information per [RFC3864].

Header field name:	Sec-Token-Binding
Applicable protocol:	HTTP
Status:	standard
Author/Change controller:	IETF
Specification document(s):	this one

Header field name:	Include-Referred-Token-Binding-ID
Applicable protocol:	HTTP
Status:	standard
Author/Change controller:	IETF
Specification document(s):	this one

[[TODO: possibly add further considerations wrt the behavior of the above header fields, per <https://tools.ietf.org/html/rfc7231#section-8.3>]]

8. Acknowledgements

This document incorporates comments and suggestions offered by Eric Rescorla, Gabriel Montenegro, Martin Thomson, Vinod Anupam, Anthony Nadalin, Michael Jones, Bill Cox, Nick Harper, Brian Campbell and others.

9. References

9.1. Normative References

[fetch-spec]
WhatWG, "Fetch", Living Standard ,
<https://fetch.spec.whatwg.org/>.

[I-D.ietf-tokbind-negotiation]
Popov, A., Nystrom, M., Balfanz, D., and A. Langley,
"Transport Layer Security (TLS) Extension for Token
Binding Protocol Negotiation", draft-ietf-tokbind-
negotiation-03 (work in progress), July 2016.

[I-D.ietf-tokbind-protocol]
Popov, A., Nystrom, M., Balfanz, D., Langley, A., and J.
Hodges, "The Token Binding Protocol Version 1.0", draft-
ietf-tokbind-protocol-07 (work in progress), July 2016.

[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>.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration
Procedures for Message Header Fields", BCP 90, RFC 3864,
DOI 10.17487/RFC3864, September 2004,
<http://www.rfc-editor.org/info/rfc3864>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data
Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006,
<http://www.rfc-editor.org/info/rfc4648>.

[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>.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport
Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705,
March 2010, <http://www.rfc-editor.org/info/rfc5705>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265,
DOI 10.17487/RFC6265, April 2011,
<http://www.rfc-editor.org/info/rfc6265>.

[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>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7541] Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015, <http://www.rfc-editor.org/info/rfc7541>.

9.2. Informative References

[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>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <http://www.rfc-editor.org/info/rfc6749>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <http://www.rfc-editor.org/info/rfc6750>.

[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>.

[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

Jeff Hodges
Paypal
USA

Email: Jeff.Hodges@paypal.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 8, 2017

A. Popov, Ed.
M. Nystroem
Microsoft Corp.
D. Balfanz
A. Langley
Google Inc.
July 7, 2016

Transport Layer Security (TLS) Extension for Token Binding Protocol Negotiation
draft-ietf-tokbind-negotiation-03

Abstract

This document specifies a Transport Layer Security (TLS) [RFC5246] extension for the negotiation of Token Binding protocol [I-D.ietf-tokbind-protocol] version and key parameters.

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 8, 2017.

Copyright Notice

Table of Contents

1. Introduction

In order to use the Token Binding protocol [I-D.ietf-tokbind-protocol], the client and server need to agree on the Token Binding protocol version and the parameters (signature algorithm, length) of the Token Binding key. This document specifies a new TLS extension to accomplish this negotiation without introducing additional network round-trips.

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 Negotiation Client Hello Extension

The client uses the "token_binding" TLS extension to indicate the highest supported Token Binding protocol version and key parameters.

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

The "extension_data" field of this extension contains a "TokenBindingParameters" value.

```
struct {
    uint8 major;
    uint8 minor;
} ProtocolVersion;

enum {
    (255)
} TokenBindingKeyParameters;

struct {
    ProtocolVersion token_binding_version;
    TokenBindingKeyParameters key_parameters_list<1..2^8-1>
} TokenBindingParameters;
```

"token_binding_version" indicates the version of the Token Binding protocol the client wishes to use during this connection. This SHOULD be the latest (highest valued) version supported by the client. [I-D.ietf-tokbind-protocol] describes version {1, 0} of the protocol. Prototype implementations of Token Binding drafts can indicate support of a specific draft version, e.g. {0, 1} or {0, 2}.

"key_parameters_list" contains the list of identifiers of the Token Binding key parameters supported by the client, in descending order of preference. [I-D.ietf-tokbind-protocol] defines an initial set of identifiers for Token Binding key parameters.

3. Token Binding Negotiation Server Hello Extension

The server uses the "token_binding" TLS extension to indicate support for the Token Binding protocol and to select the protocol version and key parameters.

The server that supports Token Binding and receives a client hello message containing the "token_binding" extension, will include the "token_binding" extension in the server hello if all of the following conditions are satisfied:

1. The server supports the Token Binding protocol version offered by the client or a lower version.

2. The server finds acceptable Token Binding key parameters on the client’s list.

3. The server is also negotiating Extended Master Secret [RFC7627] and Renegotiation Indication [RFC5746] TLS extensions. This requirement only applies when TLS 1.2 or an older TLS version is used (see security considerations section below for more details).

The server will ignore any key parameters that it does not recognize. The "extension_data" field of the "token_binding" extension is structured the same as described above for the client "extension_data".

"token_binding_version" contains the lower of the Token Binding protocol version offered by the client in the "token_binding" extension and the highest version supported by the server.

"key_parameters_list" contains exactly one Token Binding key parameters identifier selected by the server from the client's list.

4. Negotiating Token Binding Protocol Version and Key Parameters

It is expected that a server will have a list of Token Binding key parameters identifiers that it supports, in preference order. The server MUST only select an identifier that the client offered. The server SHOULD select the most highly preferred key parameters identifier it supports which is also advertised by the client. In the event that the server supports none of the key parameters that the client advertises, then the server MUST NOT include "token_binding" extension in the server hello.

The client receiving the "token_binding" extension MUST terminate the handshake with a fatal "unsupported_extension" alert if any of the following conditions are true:

1. The client did not include the "token_binding" extension in the client hello.

2. "token_binding_version" is higher than the Token Binding protocol version advertised by the client.

3. "key_parameters_list" includes more than one Token Binding key parameters identifier.

4. "key_parameters_list" includes an identifier that was not advertised by the client.

5. TLS 1.2 or an older TLS version is used, but Extended Master Secret [RFC7627] and Renegotiation Indication [RFC5746] TLS extensions are not negotiated (see security considerations section below for more details).

If the "token_binding" extension is included in the server hello and the client supports the Token Binding protocol version selected by the server, it means that the version and key parameters have been negotiated between the client and the server and SHALL be definitive

for the TLS connection. In this case, the client MUST use the negotiated key parameters in the "provided_token_binding" as described in [I-D.ietf-tokbind-protocol].

If the client does not support the Token Binding protocol version selected by the server, then the connection proceeds without Token Binding.

Please note that the Token Binding protocol version and key parameters are negotiated for each TLS connection, which means that the client and server include their "token_binding" extensions both in the full TLS handshake that establishes a new TLS session and in the subsequent abbreviated TLS handshakes that resume the TLS session.

5. IANA Considerations

This document defines a new TLS extension "token_binding", which needs to be added to the IANA "Transport Layer Security (TLS) Extensions" registry.

This document uses "Token Binding Key Parameters" registry originally created in [I-D.ietf-tokbind-protocol]. This document creates no new registrations in this registry.

6. Security Considerations

6.1. Downgrade Attacks

The Token Binding protocol version and key parameters are negotiated via "token_binding" extension 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 "token_binding" extension. The signature algorithm and key length used in the TokenBinding of type "provided_token_binding" MUST match the parameters negotiated via "token_binding" extension.

6.2. Triple Handshake Vulnerability in TLS 1.2 and Older TLS Versions

The Token Binding protocol relies on the TLS Exporters [RFC5705] to associate a TLS connection with a Token Binding. The triple handshake attack [TRIPLE-HS] is a known TLS protocol vulnerability allowing the attacker to synchronize exported keying material between TLS connections. The attacker can then successfully replay bound tokens. For this reason, the Token Binding protocol MUST NOT be negotiated with these TLS versions, unless the Extended Master Secret [RFC7627] and Renegotiation Indication [RFC5746] TLS extensions have also been negotiated.

7. Acknowledgements

This document incorporates comments and suggestions offered by Eric Rescorla, Gabriel Montenegro, Martin Thomson, Vinod Anupam, Anthony Nadalin, Michael Jones, Bill Cox, Nick Harper, Brian Campbell and others.

8. References

8.1. Normative References

[I-D.ietf-tokbind-protocol]
Popov, A., Nystrom, M., Balfanz, D., Langley, A., and J. Hodges, "The Token Binding Protocol Version 1.0", draft-ietf-tokbind-protocol-06 (work in progress), May 2016.

[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>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[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>.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <http://www.rfc-editor.org/info/rfc5705>.

[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>.

[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>.

8.2. Informative References

[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

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2017

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

July 8, 2016

The Token Binding Protocol Version 1.0
draft-ietf-tokbind-protocol-08

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 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 January 9, 2017.

Table of Contents

1. Introduction

Servers generate various security tokens (e.g. HTTP cookies, OAuth tokens) for 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 the servers, and impersonate authenticated users. The idea of Token Binding is to prevent such attacks by cryptographically binding security tokens to the TLS layer.

A 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 exported keying material [RFC5705] for the TLS connection with the private key. The corresponding public key is included in the Token Binding identifier structure (described in the "Token Binding ID Format" section of this document). Token Bindings are long-lived, i.e. they encompass multiple TLS connections and TLS sessions between a given client and server. To protect privacy, 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.

When issuing a security token to a client that supports Token Binding, a server includes the client's Token Binding ID in the token. Later on, when a client presents a security token containing a Token Binding ID, the server makes sure the ID in the token matches the ID of the 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 the Token Binding Negotiation TLS Extension [I-D.ietf-tokbind-negotiation] to negotiate the Token Binding protocol version and the parameters (signature algorithm, length) of the Token Binding key. This negotiation does not require additional round-trips.

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 [I-D.ietf-tokbind-negotiation], and then validates the signatures contained in the Token Binding message. If either of these checks fails, the server terminates the connection, otherwise the 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 Token Binding ID in the security token with the 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 defines the format of the Token Binding protocol message, the process of establishing a Token Binding, the format of the Token Binding ID, and the process of validating a security token. Token Binding Negotiation TLS Extension [I-D.ietf-tokbind-negotiation] describes the negotiation of the Token Binding protocol and key parameters. Token Binding over HTTP [I-D.ietf-tokbind-https] explains how the Token Binding message is encapsulated within HTTP/1.1 [RFC7230] or HTTP/2 [RFC7540] messages. [I-D.ietf-tokbind-https] also describes Token Binding between multiple communicating parties: User Agent, Identity Provider and Relying Party.

3. Token Binding Protocol Message

The Token Binding message is sent by the client to prove 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 [I-D.ietf-tokbind-negotiation], 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 additional private keys held by the same client, which can be used to facilitate token binding between more than two communicating parties. For example, Token Binding over HTTP [I-D.ietf-tokbind-https] specifies an encapsulation of the Token Binding message in HTTP application protocol messages, as well as scenarios involving more than two communicating parties.

The Token Binding message format is defined using TLS Presentation Language (see Section 4 of [RFC5246]):

```
enum {
    rsa2048_pkcs1.5(0), rsa2048_pss(1), ecdsap256(2), (255)
} TokenBindingKeyParameters;

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

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

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

struct {
    TokenBindingKeyParameters key_parameters;
    select (key_parameters) {
        case rsa2048_pkcs1.5:
        case rsa2048_pss:
            RSAPublicKey rsapubkey;
        case ecdsap256:
            ECPoint point;
    }
} TokenBindingID;

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

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

struct {
    TokenBindingType tokenbinding_type;
    TokenBindingID tokenbindingid;
    opaque signature<0..2^16-1>;  /* Signature over the exported
                                     keying material (EKM) value */
    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, each containing the type of the token binding, the TokenBindingID, a signature over an exported keying material (EKM) value, optionally followed by Extension structures.

This document defines two Token Binding types:

- provided_token_binding - used to establish a Token Binding when connecting to a server.
- referred_token_binding - used when requesting tokens to be presented to a different server.

Token Binding over HTTP [I-D.ietf-tokbind-https] describes a use case for referred_token_binding where Token Bindings are established between multiple communicating parties: User Agent, Identity Provider and Relying Party. User Agent sends referred_token_binding to the Identity Provider in order to prove possession of the Token Binding key it uses with the Relying Party. The Identity Provider can then bind the token it is supplying (for presentation to the Relying Party) to the Token Binding ID contained in the referred_token_binding. Such bound token enjoys the protections discussed below in Section 8 "Security Considerations".

This document establishes an IANA registry for Token Binding extensions in Section 7.2 "Token Binding Types Registry". An implementation MUST ignore any unknown Token Binding types.

When an rsa2048_pkcs1.5 or rsa2048_pss key is used, TokenBinding.signature contains the signature generated using, respectively, the RSASSA-PKCS1-v1_5 or RSASSA-PSS signature scheme defined in [RFC3447]. RSAPublicKey.modulus and RSAPublicKey.publicexponent contain the length-prefixed modulus and exponent of the RSA public key represented in big-endian format.

When an ecdsap256 key is used, TokenBinding.signature contains a pair of 32-byte integers, R followed by S, generated using Curve P-256 as defined in [ANSI.X9-62.2005] and [FIPS.186-4.2013]. R and S are encoded in big-endian format, preserving any leading zero bytes. ECPoint.point contains the X coordinate followed by the Y coordinate. The X and Y coordinates are unsigned 32-byte integers encoded in big-endian format, preserving any leading zero bytes. Future specifications may define Token Binding keys using other elliptic curves with their corresponding signature and point formats.

The EKM is obtained using the Keying Material Exporters for TLS defined in [RFC5705], by supplying the following input values:

- Label: The ASCII string "EXPORTER-Token-Binding" with no terminating NUL.
- Context value: NULL (no application context supplied).
- Length: 32 bytes.

4. Establishing a Token Binding

4.1. Client Processing Rules

The client MUST include at least one TokenBinding structure in the Token Binding message. The key parameters used in the provided_token_binding MUST match those negotiated with the server via [I-D.ietf-tokbind-negotiation].

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.

When the client needs to send a referred_token_binding to the Identity Provider, the client SHALL construct the referred TokenBinding structure in the following manner:

- Set TokenBinding.tokenbinding_type to referred_token_binding.
- Set TokenBinding.tokenbindingid to the Token Binding ID used with the Relying Party.
- Set TokenBinding.signature to the result of signing the EKM value of the TLS connection to the Identity Provider, using the Token Binding key established with the Relying Party and the signature algorithm indicated by the associated key parameters. Note that these key parameters may differ from the key parameters negotiated with the Identity Provider.

Conveying referred Token Bindings in this fashion allows the Identity Provider to verify that the client controls the Token Binding key used with the Relying Party.

4.2. Server Processing Rules

The triple handshake vulnerability in TLS 1.2 and older TLS versions affects the security of the Token Binding protocol, as described in Section 8 "Security Considerations". Therefore, the server MUST NOT negotiate the use of the Token Binding protocol with these TLS

versions, unless the server also negotiates Extended Master Secret [RFC7627] and Renegotiation Indication [RFC5746] TLS extensions.

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 [I-D.ietf-tokbind-negotiation]. In the case of a mismatch, the server MUST terminate the connection. Token Bindings of type "referred_token_binding" may use different key parameters than those negotiated with this client.

If the Token Binding message does not contain at least one TokenBinding structure, or if a signature contained in any TokenBinding structure is invalid, the server MUST terminate the connection.

Servers MUST ignore any unknown extensions. Initially, no extension types are defined (see Section 7.3 "Token Binding Extensions Registry"). One of the possible uses of extensions envisioned at the time of this writing is attestation: cryptographic proof that allows the server to verify that the Token Binding key is hardware-bound. The definitions of such Token Binding protocol extensions are outside the scope of this specification.

If all checks defined above have passed successfully, the Token Binding between this client and server is established. The Token Binding ID(s) conveyed in the Token Binding Message can be provided to the server-side application. The application may then use the Token Binding IDs for bound security token creation and validation, see Section 6.

5. Token Binding ID Format

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

```
struct {
    TokenBindingKeyParameters key_parameters;
    select (key_parameters) {
        case rsa2048_pkcs1.5:
        case rsa2048_pss:
            RSAPublicKey rsapubkey;
        case ecdsap256:
            ECPoint point;
    }
} TokenBindingID;
```

TokenBindingID contains the key parameters negotiated via [I-D.ietf-tokbind-negotiation]. Token Binding ID can be obtained from the TokenBinding structure described in the "Token Binding Protocol Message" section of this document by discarding the Token Binding type, signature and extensions. Token Binding protocol implementations SHOULD make Token Binding IDs available to the application as opaque byte sequences. E.g. server applications will use Token Binding IDs when generating and verifying bound tokens.

6. Bound Security Token Creation and 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. Note that applicable security considerations are outlined in Section 8.

Either or both clients and servers MAY create bound security tokens. For example, HTTPS servers employing Token Binding for securing their HTTP cookies will bind the cookies. In the case of a server-initiated challenge-response protocol employing Token Binding and TLS, the client can, for example, incorporate the Token Binding ID within the signed object it returns, thus binding the object.

Upon receipt of a security token, the server attempts to retrieve 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 Token Binding has not been established for the client connection, the server MUST discard the token. If the Token Binding

ID for the token does not match the Token Binding ID established for the client connection, the server MUST discard the token.

7. IANA Considerations

This section establishes three IANA registries: "Token Binding Key Parameters", "Token Binding Types" and "Token Binding Extensions". It also registers a new TLS exporter label in the TLS Exporter Label Registry.

7.1. Token Binding Key Parameters Registry

This document establishes a registry for identifiers of Token Binding key parameters entitled "Token Binding Key Parameters" under the "Token Binding Protocol" heading.

Entries in this registry require the following fields:

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

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 set of Token Binding key parameters.

An initial set of registrations for this registry follows:

Value: 0

Description: rsa2048_pkcs1.5

Specification: this document

Value: 1

Description: rsa2048_pss

Specification: this document

Value: 2

Description: ecdsap256

Specification: this document

7.2. Token Binding Types Registry

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:

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

7.3. Token Binding Extensions Registry

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:

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

7.4. Registration of Token Binding TLS Exporter Label

This document adds a registration for the "EXPORTER-Token-Binding" value in the TLS Exporter Label Registry to correspond to this specification.

8. Security Considerations

8.1. 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.

The manner in which a token is bound to the TLS layer is application-defined and beyond the scope of this document. However, the resulting bound token needs to be integrity-protected, so that an attacker cannot remove the binding or substitute a Token Binding ID of their choice without detection.

8.2. Downgrade Attacks

The Token Binding protocol is only used when negotiated via [I-D.ietf-tokbind-negotiation] 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 Token Binding Negotiation TLS Extension 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 [I-D.ietf-tokbind-negotiation].

8.3. Privacy Considerations

The Token Binding protocol uses persistent, long-lived Token Binding IDs. To protect privacy, 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. Some applications offer a special privacy mode where they don't store or use tokens supplied by the server, e.g. "in private" browsing. When operating in this special privacy mode, applications SHOULD use newly generated Token Binding keys and delete them when exiting this mode, or else SHOULD NOT negotiate Token Binding at all.

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.

A server can use tokens and Token Binding IDs to track clients. Client applications that automatically limit the lifetime of tokens to maintain user privacy SHOULD apply the same validity time limits to Token Binding keys.

8.4. Token Binding Key Sharing Between Applications

Existing systems provide a variety of platform-specific mechanisms for certain applications to share tokens, e.g. to enable single sign-on scenarios. For these scenarios to keep working with bound tokens, the applications that are allowed to share tokens will need to also share Token Binding keys. Care must be taken to restrict the sharing of Token Binding keys to the same group(s) of applications that share the same tokens.

8.5. Triple Handshake Vulnerability in TLS 1.2 and Older TLS Versions

The Token Binding protocol relies on the exported keying material (EKM) to associate a TLS connection with a Token Binding. The triple handshake attack [TRIPLE-HS] is a known vulnerability in TLS 1.2 and older TLS versions, allowing the attacker to synchronize keying material between TLS connections. The attacker can then successfully replay bound tokens. For this reason, the Token Binding protocol MUST NOT be negotiated with these TLS versions, unless the Extended

Master Secret [RFC7627] and Renegotiation Indication [RFC5746] TLS extensions have also been negotiated.

9. Acknowledgements

This document incorporates comments and suggestions offered by Eric Rescorla, Gabriel Montenegro, Martin Thomson, Vinod Anupam, Anthony Nadalin, Michael Jones, Bill Cox, Nick Harper, Brian Campbell and others.

10. References

10.1. Normative References

[ANSI.X9-62.2005]
American National Standards Institute, "Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)", ANSI X9.62, 2005.

[FIPS.186-4.2013]
National Institute of Standards and Technology, "Digital Signature Standard (DSS)", FIPS 186-4, 2013.

[I-D.ietf-tokbind-https]
Popov, A., Nystrom, M., Balfanz, D., Langley, A., and J. Hodges, "Token Binding over HTTP", draft-ietf-tokbind-https-05 (work in progress), July 2016.

[I-D.ietf-tokbind-negotiation]
Popov, A., Nystrom, M., Balfanz, D., and A. Langley, "Transport Layer Security (TLS) Extension for Token Binding Protocol Negotiation", draft-ietf-tokbind-negotiation-03 (work in progress), July 2016.

[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>.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, DOI 10.17487/RFC3447, February 2003, <http://www.rfc-editor.org/info/rfc3447>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[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>.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <http://www.rfc-editor.org/info/rfc5705>.

[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>.

[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>.

10.2. Informative References

[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

Jeff Hodges
Paypal
USA

Email: Jeff.Hodges@paypal.com

OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 5, 2017

M. Jones
Microsoft
J. Bradley
B. Campbell
Ping Identity
July 4, 2016

OAuth 2.0 Token Binding
draft-jones-oauth-token-binding-00

Abstract

This specification enables OAuth 2.0 implementations to apply Token Binding to Access Tokens and Refresh Tokens. This cryptographically binds these tokens to the TLS connections over which they are intended to be used. This use of Token Binding protects these tokens from man-in-the-middle and token export and replay attacks.

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, 2017.

Table of Contents

1. Introduction

This specification enables OAuth 2.0 [RFC6749] implementations to apply Token Binding The Token Binding Protocol Version 1.0 [I-D.ietf-tokbind-protocol] Token Binding over HTTP [I-D.ietf-tokbind-https] to Access Tokens and Refresh Tokens. This cryptographically binds these tokens to the TLS connections over which they are intended to be used. This use of Token Binding protects these tokens from man-in-the-middle and token export and replay attacks.

1.1. Requirements Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Terminology

This specification uses the terms "Access Token", "Authorization Code", "Authorization Endpoint", "Authorization Grant", "Authorization Server", "Client", "Client Authentication", "Client Identifier", "Client Secret", "Grant Type", "Protected Resource", "Redirection URI", "Refresh Token", "Resource Owner", "Resource Server", "Response Type", and "Token Endpoint" defined by OAuth 2.0 [RFC6749], the terms "Claim", "Claim Name", "Claim Value", and "JSON Web Token (JWT)" defined by JSON Web Token (JWT) [JWT], the term "User Agent" defined by RFC 7230 [RFC7230], and the terms "Provided", "Referred", "Token Binding" and "Token Binding ID" defined by Token Binding over HTTP [I-D.ietf-tokbind-https].

2. Token Binding for Refresh Tokens

Token Binding of refresh tokens is a straightforward first-party scenario, applying term "first-party" as used in Token Binding over HTTP [I-D.ietf-tokbind-https]. It cryptographically binds the refresh token to the TLS connection between the client and the token endpoint. This case is straightforward because the refresh token is both retrieved by the client from the token endpoint and sent by the client to the token endpoint. Unlike the federated scenarios described in Section 3 (Federation Use Cases) of Token Binding over HTTP [I-D.ietf-tokbind-https] and the access token case described in the next section, only a single TLS connection is involved in the refresh token case.

Token Binding a refresh token requires that the authorization server do two things. First, when refresh token is sent to the client, the authorization server needs to remember the Provided Token Binding ID and remember its association with the issued refresh token. Second, when a token request containing a refresh token is received at the token endpoint, the authorization server needs to verify that the Provided Token Binding ID for the request matches the remembered Token Binding ID associated with the refresh token. If the Token Binding IDs do not match, the authorization server should return an error in response to the request.

The means by which the authorization server remembers the association between the refresh token and the Token Binding ID is an

implementation detail that beyond the scope of this specification. Some authorization servers will choose to store the Token Binding ID (or a cryptographic hash of it, such a SHA-256 hash [SHS]) in the refresh token itself, thus reducing the amount of state to be kept by the server. Other authorization servers will add the Token Binding ID value (or a hash of it) to an internal data structure also containing other information about the refresh token, such as grant type information. These choices make no difference to the client, since the refresh token is opaque to it.

3. Token Binding for Access Tokens

Token Binding for access tokens cryptographically binds the access token to the TLS connection between the client and the resource server. Token Binding is applied to access tokens in a similar manner to that described in Section 3 (Federation Use Cases) of Token Binding over HTTP [I-D.ietf-tokbind-https]. It is also builds upon the mechanisms for Token Binding of ID Tokens defined in OpenID Connect Token Bound Authentication 1.0 [OpenID.TokenBinding].

In the OpenID Connect [OpenID.Core] use case, HTTP redirects are used to pass information between the identity provider and the relying party; this HTTP redirect makes the Token Binding ID of the relying party available to the identity provider as the Referred Token Binding ID, information about which is then added to the ID Token. No such redirect occurs between the authorization server and the resource server in the access token case; therefore, information about the Token Binding ID for the TLS connection between the client and the resource server needs to be explicitly communicated by the client to the authorization server to achieve Token Binding of the access token. This information is passed to the authorization server using this request parameter:

resource_tbh
: Base64url encoding of the SHA-256 hash [SHS] of the Token Binding ID for the TLS connection between the client and the resource server.

Note that to obtain this Token Binding ID, the client needs to establish a TLS connection between itself and the resource server prior to making the authorization request so that the Provided Token Binding ID for the TLS connection to the resource server can be obtained. The means by which the client retrieves this Token Binding ID from the underlying Token Binding API is implementation and operating system specific. An alternative, if supported, is for the client to generate a Token Binding key to use for the resource server, use the Token Binding ID for that key, and then later use

that key when the TLS connection to the resource server is established.

The authorization server MUST ignore the "resource_tbh" parameter if it does not support Token Binding for the access token.

3.1. Initial Access Tokens

Upon receiving the hash of the Token Binding ID in an authorization request containing the "resource_tbh" (resource token binding hash) authorization request parameter, the authorization server then records it in the issued access token. Alternatively, in some implementations, the resource's Token Binding ID hash might be communicated to the resource server by other means, such as by introspecting [RFC7662] the access token.

3.2. Refreshed Access Tokens

Access tokens obtained from refresh requests can also be token bound. In this case, the hash of the Token Binding ID of the TLS connection between the client and the resource server is sent to the authorization server at the token endpoint using the "resource_tbh" (resource token binding hash) token request parameter; its syntax is exactly the same as the corresponding authorization request parameter. The authorization server then records it in the issued access token or communicates it to the resource server by other means, just as in the previous case.

3.3. Resource Server Token Binding Validation

Upon receiving a token bound access token, the resource server validates the binding by computing a SHA-256 hash of the Provided Token Binding ID and comparing it to the token binding hash value for the access token. If these values do not match, the resource access attempt MUST be rejected with an error.

3.4. Representing Token Binding in JWT Access Tokens

If the access token is represented as a JWT, the token binding information SHOULD be represented in the same way that it is in token bound OpenID Connect ID Tokens [OpenID.TokenBinding]. That specification defines the new JWT Confirmation Method RFC 7800 [RFC7800] member "tbh" (token binding hash) to represent the SHA-256 hash of a Token Binding ID in an ID Token. The value of the "tbh" member is the base64url encoding of the SHA-256 hash of the Token Binding ID.

The following example demonstrates the JWT Claims Set of an access token containing the base64url encoding of the SHA-256 hash of a Token Binding ID as the value of the "tbh" (token binding hash) element in the "cnf" (confirmation) claim:

```
{
 "iss": "https://server.example.com",
 "aud": "https://resource.example.com",
 "iat": 1467324320,
 "exp": 1467324920,
 "cnf":{
   "tbh": "n0jI3trBK6_Gp2qiLOf48ZEZTjpBnhm-QOyzJxhBeAk"
  }
}
```

4. Phasing in Token Binding and Preventing Downgrade Attacks

Many OAuth implementations will be deployed in situations in which not all participants support Token Binding. Any of combination of the client, the authorization server, the resource server, and the User Agent may not yet support Token Binding, in which case it will not work end-to-end.

It is a context-dependent deployment choice whether to allow interactions to proceed in which Token Binding is not supported or whether to treat Token Binding failures at any step as fatal errors. Particularly in dynamic deployment environments in which End Users have choices of clients, authorization servers, resource servers, and/or User Agents, it is RECOMMENDED that authorizations using one or more components that do not implement Token Binding be allowed to successfully proceed. This enables different components to be upgraded to supporting Token Binding at different times, providing a smooth transition path for phasing in Token Binding. However, when Token Binding has been performed, any Token Binding key mismatches MUST be treated as fatal errors.

If all the participants in an authorization interaction support Token Binding and yet one or more of them does not use it, this is likely evidence of a downgrade attack. In this case, the authorization SHOULD be aborted with an error. For instance, if the resource server knows that the authorization server and the User Agent both support Token Binding and yet the access token received does not contain Token Binding information, this is almost certainly a sign of an attack.

The authorization server and client can determine whether the other supports Token Binding using the metadata values defined in the next section. They can determine whether the User Agent supports Token

Binding by whether it negotiated Token Binding for the TLS connection. At present, there is no defined mechanism for determining whether the resource server supports Token Binding or not. However, it always safe to proceed as if it does; at worst, the resource server simply won't verify the Token Binding.

5. Token Binding Metadata

5.1. Token Binding Client Metadata

Clients supporting Token Binding that also support the OAuth 2.0 Dynamic Client Registration Protocol [RFC7591] use these metadata values to register their support for Token Binding of Access Tokens and Refresh Tokens:

client_access_token_token_binding_supported
: OPTIONAL. Boolean value specifying whether the Client supports Token Binding of Access Tokens. If omitted, the default value is "false".

client_refresh_token_token_binding_supported
: OPTIONAL. Boolean value specifying whether the Client supports Token Binding of Refresh Tokens. If omitted, the default value is "false".

5.2. Token Binding Authorization Server Metadata

Authorization Servers supporting Token Binding that also support OAuth 2.0 Discovery Metadata [OAuth.Discovery] use this metadata values to register their support for Token Binding of Access Tokens and Refresh Tokens:

as_access_token_token_binding_supported
: OPTIONAL. Boolean value specifying whether the Authorization Server supports Token Binding of Access Tokens. If omitted, the default value is "false".

as_refresh_token_token_binding_supported
: OPTIONAL. Boolean value specifying whether the Authorization Server supports Token Binding of Refresh Tokens. If omitted, the default value is "false".

6. Security Considerations

If a refresh request is received by the authorization server containing a "resource_tbh" (resource token binding hash) value requesting a token bound access token and the refresh token in the request is not itself token bound, then it is not clear that token

binding the access token adds significant value. This situation should be considered an open issue for discussion by the working group.

7. IANA Considerations

7.1. OAuth Parameters Registration

This specification registers the following parameter in the IANA "OAuth Parameters" registry [IANA.OAuth.Parameters] established by RFC 6749 [RFC6749]:

7.1.1. Registry Contents

- Parameter name: "resource_tbh"
- Parameter usage location: Authorization Request, Token Request
- Change controller: IESG
- Specification document(s): Section 3 of this document
- Related information: None

7.2. OAuth Dynamic Client Registration Metadata Registration

This specification registers the following client metadata definitions in the IANA "OAuth Dynamic Client Registration Metadata" registry [IANA.OAuth.Parameters] established by [RFC7591]:

7.2.1. Registry Contents

- Client Metadata Name: "client_access_token_token_binding_supported"
- Client Metadata Description: Boolean value specifying whether the Client supports Token Binding of Access Tokens
- Change Controller: IESG
- Specification Document(s): Section 5.1 of [[this specification]]

- Client Metadata Name: "client_refresh_token_token_binding_supported"
- Client Metadata Description: Boolean value specifying whether the Client supports Token Binding of Refresh Tokens
- Change Controller: IESG
- Specification Document(s): Section 5.1 of [[this specification]]

7.3. OAuth Authorization Server Discovery Metadata Registration

This specification registers the following discovery metadata definitions in the IANA "OAuth Authorization Server Discovery Metadata" registry established by [OAuth.Discovery]:

7.3.1. Registry Contents

- Discovery Metadata Name: "as_access_token_token_binding_supported"
- Discovery Metadata Description: Boolean value specifying whether the Authorization Server supports Token Binding of Access Tokens
- Change Controller: IESG
- Specification Document(s): Section 5.2 of [[this specification]]

- Discovery Metadata Name: "as_refresh_token_token_binding_supported"
- Discovery Metadata Description: Boolean value specifying whether the Authorization Server supports Token Binding of Refresh Tokens
- Change Controller: IESG
- Specification Document(s): Section 5.2 of [[this specification]]

8. References

8.1. Normative References

[I-D.ietf-tokbind-https]
Popov, A., Nystrom, M., Balfanz, D., Langley, A., and J. Hodges, "Token Binding over HTTP", draft-ietf-tokbind-https-03 (work in progress), March 2016.

[I-D.ietf-tokbind-protocol]
Popov, A., Nystrom, M., Balfanz, D., Langley, A., and J. Hodges, "The Token Binding Protocol Version 1.0", draft-ietf-tokbind-protocol-06 (work in progress), May 2016.

[IANA.OAuth.Parameters]
IANA, "OAuth Parameters", <http://www.iana.org/assignments/oauth-parameters>.

[JWT] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <http://tools.ietf.org/html/rfc7519>.

[OpenID.TokenBinding]
Jones, M., Bradley, J., and B. Campbell, "OpenID Connect Token Bound Authentication 1.0", July 2016, <http://self-issued.info/docs/openid-connect-token-bound-authentication-1_0.html>.

[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>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <http://www.rfc-editor.org/info/rfc6749>.

[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>.

[RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <http://www.rfc-editor.org/info/rfc7662>.

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <http://www.rfc-editor.org/info/rfc7800>.

[SHS] National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS PUB 180-4, March 2012, <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>.

8.2. Informative References

[OAuth.Discovery] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Discovery", draft-ietf-oauth-discovery-02 (work in progress), March 2016, <http://tools.ietf.org/html/draft-ietf-oauth-discovery-02>.

[OpenID.Core] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", November 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.

[RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <http://www.rfc-editor.org/info/rfc7591>.

Appendix A. Acknowledgements

The authors would like to thank the following people for their contributions to the specification: Dirk Balfanz, William Denniss, Andrei Popov, and Nat Sakimura.

Appendix B. Open Issues

- Some token binding implementations apparently provide APIs that enable native applications to provide Referred Token Bindings, just as the federation support in the HTTPS Token Binding spec does. Can we count on these APIs being supported on all platforms, and if so, does this enable us to somehow do without the "resource_tbh" parameter by mandating that the client send both a Provided and a Referred Token Binding to the authorization server? If this isn't the case, is "resource_tbh" actually secure or does this open a cross-channel validation hole? This area probably needs more attention from both the Token Binding and OAuth working groups.
- How should we support crypto agility for the hash function?

Appendix C. Document History

[[to be removed by the RFC Editor before publication as an RFC]]

-00

- Created the initial version.

Authors' Addresses

Michael B. Jones
Microsoft

Email: mbj@microsoft.com
URI: http://self-issued.info/

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com
URI: http://www.thread-safe.com/

Brian Campbell
Ping Identity

Email: brian.d.campbell@gmail.com
URI: https://twitter.com/__b_c