

TEEP WG
Internet-Draft
Intended status: Informational
Expires: January 9, 2020

D. Thaler
Microsoft
July 08, 2019

HTTP Transport for the Open Trust Protocol (OTrP)
draft-ietf-teep-otrp-over-http-01

Abstract

This document specifies the HTTP transport for the Open Trust Protocol (OTrP), which is used to manage code and configuration data in a Trusted Execution Environment (TEE). An implementation of this document can run outside of any TEE, but interacts with an OTrP implementation that runs inside a TEE.

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, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

- [1. Introduction](#) [2](#)
- [2. Terminology](#) [3](#)
- [3. Use of Abstract APIs](#) [3](#)
- [4. Use of HTTP as a Transport](#) [3](#)
- [5. TEEP Broker Behavior](#) [4](#)
 - 5.1. Receiving a request to install a new Trusted Application [4](#)
 - [5.1.1. Session Creation](#) [5](#)
 - [5.2. Getting a message buffer back from an TEEP Agent](#) [5](#)
 - [5.3. Receiving an HTTP response](#) [6](#)
 - [5.4. Handling checks for policy changes](#) [6](#)
 - [5.5. Error handling](#) [7](#)
- [6. TAM Broker Behavior](#) [7](#)
 - [6.1. Receiving an HTTP POST request](#) [7](#)
 - [6.2. Getting an empty buffer back from the TAM](#) [7](#)
 - [6.3. Getting a message buffer from the TAM](#) [7](#)
 - [6.4. Error handling](#) [7](#)
- [7. Sample message flow](#) [7](#)
- [8. Security Considerations](#) [9](#)
- [9. IANA Considerations](#) [10](#)
- [10. References](#) [10](#)
 - [10.1. Normative References](#) [10](#)
 - [10.2. Informative References](#) [10](#)
- Author's Address [11](#)

[1. Introduction](#)

Trusted Execution Environments (TEEs), including Intel SGX, ARM TrustZone, Secure Elements, and others, enforce that only authorized code can execute within the TEE, and any memory used by such code is protected against tampering or disclosure outside the TEE. The Open Trust Protocol (OTrP) is designed to provision authorized code and configuration into TEEs.

To be secure against malware, an OTrP implementation (referred to as an OTrP "Agent" on the client side, and a "Trusted Application Manager (TAM)" on the server side) must themselves run inside a TEE. However, the transport for OTrP, along with typical networking stacks, need not run inside a TEE. This split allows the set of highly trusted code to be kept as small as possible, including allowing code (e.g., TCP/IP) that only sees encrypted messages to be kept out of the TEE.

The OTrP specification [[I-D.ietf-teep-opentrustprotocol](#)] describes the behavior of TEEP Agents and TAMs, but does not specify the details of the transport, an implementation of which is referred to as a "Broker". The purpose of this document is to provide such

details. That is, the HTTP transport for OTrP is implemented in a Broker (typically outside a TEE) that delivers messages up to an OTrP implementation, and accepts messages from the OTrP implementation to be sent over a network.

[2.](#) Terminology

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 [BCP 14](#) [[RFC2119](#)] [[RFC8174](#)] when, and only when, they appear in all capitals, as shown here.

This document also uses various terms defined in [[I-D.ietf-teep-architecture](#)], including Trusted Execution Environment (TEE), Trusted Application (TA), Trusted Application Manager (TAM), TEEP Agent, and TEEP Broker.

[3.](#) Use of Abstract APIs

This document refers to various APIs between a Broker and an OTrP implementation in the abstract, meaning the literal syntax and programming language are not specified, so that various concrete APIs can be designed (outside of the IETF) that are compliant.

It is common in some TEE architectures (e.g., SGX) to refer to calls into a Trusted Application (TA) as "ECALLs" (or enclave-calls), and calls out from a Trusted Application (TA) as "OCALLs" (or out-calls).

In other TEE architectures, there may be no OCALLs, but merely data returned from calls into a TA. This document attempts to be agnostic as to the concrete API architecture. As such, abstract APIs used in this document will refer to calls into a TA as API calls, and will simply refer to "passing data" back out of the TA. A concrete API might pass data back via an OCALL or via data returned from an API call.

This document will also refer to passing "no" data back out of a TA. In an OCALL-based architecture, this might be implemented by not making any such call. In a return-based architecture, this might be implemented by returning 0 bytes.

4. Use of HTTP as a Transport

This document uses HTTP [[I-D.ietf-httpbis-semantic](#)s] as a transport. When not called out explicitly in this document, all implementation recommendations in [[I-D.ietf-httpbis-bcp56bis](#)] apply to use of HTTP by OTrP.

Thaler

Expires January 9, 2020

[Page 3]

Internet-Draft

OTrP HTTP Transport

July 2019

Redirects MAY be automatically followed, and no additional request headers beyond those specified by HTTP need be modified or removed upon a following such a redirect.

Content is not intended to be treated as active by browsers and so HTTP responses with content SHOULD have the following headers as explained in Section 4.12 of [[I-D.ietf-httpbis-bcp56bis](#)] (replacing the content type with the relevant OTrP content type per the OTrP specification):

```
Content-Type: <content type>
Cache-Control: no-store
X-Content-Type-Options: nosniff
Content-Security-Policy: default-src 'none'
Referrer-Policy: no-referrer
```

Only the POST method is specified for TAM resources exposed over HTTP. A URI of such a resource is referred to as a "TAM URI". A TAM URI can be any HTTP(S) URI. The URI to use is configured in an TEEP Agent via an out-of-band mechanism, as discussed in the next section.

When HTTPS is used, TLS certificates MUST be checked according to [[RFC2818](#)].

5. TEEP Broker Behavior

5.1. Receiving a request to install a new Trusted Application

When the TEEP Broker receives a notification (e.g., from an

application installer) that an application has a dependency on a given Trusted Application (TA) being available in a given type of TEE, the notification will contain the following:

- A unique identifier of the TA
- Optionally, any metadata to pass to the TEEP Agent. This might include a TAM URI provided in the application manifest, for example.
- Optionally, any requirements that may affect the choice of TEE, if multiple are available to the TEEP Broker.

When such a notification is received, the TEEP Broker first identifies in an implementation-dependent way which TEE (if any) is most appropriate based on the constraints expressed. If there is only one TEE, the choice is obvious. Otherwise, the choice might be based on factors such as capabilities of available TEE(s) compared with TEE requirements in the notification.

The TEEP Broker then informs the TEEP Agent in that TEE by invoking an appropriate "RequestTA" API that identifies the TA needed and any other associated metadata. The TEEP Broker need not know whether the TEE already has such a TA installed or whether it is up to date.

The TEEP Agent will either (a) pass no data back, (b) pass back a TAM URI to connect to, or (c) pass back a message buffer and TAM URI to send it to. The TAM URI passed back may or may not be the same as the TAM URI, if any, provided by the broker, depending on the TEEP Agent's configuration. If they differ, the TEEP Broker MUST use the TAM URI passed back.

[5.1.1.](#) Session Creation

If no data is passed back, the TEEP Broker simply informs its client (e.g., the application installer) of success.

If the TEEP Agent passes back a TAM URI with no message buffer, the TEEP Broker attempts to create session state, then sends an HTTP(S) POST to the TAM URI with an Accept header and an empty body. The HTTP request is then associated with the TEEP Broker's session state.

If the TEEP Agent instead passes back a TAM URI with a message buffer, the TEEP Broker attempts to create session state and handles the message buffer as specified in [Section 5.2](#).

Session state consists of:

- Any context (e.g., a handle) that identifies the API session with the TEEP Agent.
- Any context that identifies an HTTP request, if one is outstanding. Initially, none exists.

[5.2](#). Getting a message buffer back from an TEEP Agent

When a message buffer (and TAM URI) is passed to a TEEP Broker from an TEEP Agent, the TEEP Broker MUST do the following, using the TEEP Broker's session state associated with its API call to the TEEP Agent.

The TEEP Broker sends an HTTP POST request to the TAM URI with Accept and Content-Type headers with the OTrP media type in use, and a body containing the OTrP message buffer provided by the TEEP Agent. The HTTP request is then associated with the TEEP Broker's session state.

[5.3](#). Receiving an HTTP response

When an HTTP response is received in response to a request associated with a given session state, the TEEP Broker MUST do the following.

If the HTTP response body is empty, the TEEP Broker's task is complete, and it can delete its session state, and its task is done.

If instead the HTTP response body is not empty, the TEEP Broker calls a "ProcessOTrPMessage" API (Section 6.2 of [\[I-D.ietf-teep-opentrustprotocol\]](#)) to pass the response body to the TEEP Agent associated with the session. The TEEP Agent will then pass no data back, or pass back a message buffer.

If no data is passed back, the TEEP Broker's task is complete, and it

can delete its session state, and inform its client (e.g., the application installer) of success.

If instead the TEEP Agent passes back a message buffer, the TEEP Broker handles the message buffer as specified in [Section 5.2](#).

[5.4](#). Handling checks for policy changes

An implementation MUST provide a way to periodically check for OTrP policy changes. This can be done in any implementation-specific manner, such as:

A) The TEEP Broker might call into the TEEP Agent at an interval previously specified by the TEEP Agent. This approach requires that the TEEP Broker be capable of running a periodic timer.

B) The TEEP Broker might be informed when an existing TA is invoked, and call into the TEEP Agent if more time has passed than was previously specified by the TEEP Agent. This approach allows the device to go to sleep for a potentially long period of time.

C) The TEEP Broker might be informed when any attestation attempt determines that the device is out of compliance, and call into the TEEP Agent to remediate.

The TEEP Broker informs the TEEP Agent by invoking an appropriate "RequestPolicyCheck" API. The TEEP Agent will either (a) pass no data back, (b) pass back a TAM URI to connect to, or (c) pass back a message buffer and TAM URI to send it to. Processing then continues as specified in [Section 5.1.1](#).

[5.5](#). Error handling

If any local error occurs where the TEEP Broker cannot get a message buffer (empty or not) back from the TEEP Agent, the TEEP Broker deletes its session state, and informs its client (e.g., the application installer) of a failure.

If any HTTP request results in an HTTP error response or a lower

layer error (e.g., network unreachable), the TEEP Broker calls the TEEP Agent's "ProcessError" API, and then deletes its session state and informs its client of a failure.

[6.](#) TAM Broker Behavior

[6.1.](#) Receiving an HTTP POST request

When an HTTP POST request is received with an empty body, the TAM Broker invokes the TAM's "ProcessConnect" API. The TAM will then pass back a (possibly empty) message buffer.

When an HTTP POST request is received with a non-empty body, the TAM Broker calls the TAM's "ProcessOTrPMessage" API to pass it the request body. The TAM will then pass back a (possibly empty) message buffer.

[6.2.](#) Getting an empty buffer back from the TAM

If the TAM passes back an empty buffer, the TAM Broker sends a successful (2xx) response with no body.

[6.3.](#) Getting a message buffer from the TAM

If the TAM passes back a non-empty buffer, the TAM Broker generates a successful (2xx) response with a Content-Type header with the OTrP media type in use, and with the message buffer as the body.

[6.4.](#) Error handling

If any error occurs where the TAM Broker cannot get a message buffer (empty or not) back from the TAM, the TAM Broker generates an appropriate HTTP error response.

[7.](#) Sample message flow

The following shows a sample OTrP message flow that uses application/otrp+json as the Content-Type.

1. An application installer determines (e.g., from an app manifest)

that the application has a dependency on TA "X", and passes this notification to the TEEP Broker. The TEEP Broker picks an TEEP Agent (e.g., the only one available) based on this notification.

2. The TEEP Broker calls the TEEP Agent's "RequestTA" API, passing TA Needed = X.
3. The TEEP Agent finds that no such TA is already installed, but that it can be obtained from a given TAM. The TEEP Agent passes the TAM URI (e.g., "https://example.com/tam") to the TEEP Broker. (If the TEEP Agent already had a cached TAM certificate that it trusts, it could skip to step 9 instead and generate a GetDeviceStateResponse.)
4. The TEEP Broker sends an HTTP POST request to the TAM URI:

```
POST /tam HTTP/1.1
Host: example.com
Accept: application/otrp+json
Content-Length: 0
User-Agent: Foo/1.0
```

5. The TAM Broker receives the HTTP POST request, and calls the TAM's "ProcessConnect" API.
6. The TAM generates an OTrP message (typically GetDeviceStateRequest is the first message) and passes it to the TAM Broker.
7. The TAM Broker sends an HTTP successful response with the OTrP message in the body:

```
HTTP/1.1 200 OK
Content-Type: application/otrp+json
Content-Length: [length of OTrP message here]
Server: Bar/2.2
Cache-Control: no-store
X-Content-Type-Options: nosniff
Content-Security-Policy: default-src 'none'
Referrer-Policy: no-referrer

[OTrP message here]
```

8. The TEEP Broker gets the HTTP response, extracts the OTrP message and calls the TEEP Agent's "ProcessOTrPMessage" API to pass it the message.
9. The TEEP Agent processes the OTrP message, and generates an OTrP response (e.g., GetDeviceStateResponse) which it passes back to the TEEP Broker.
10. The TEEP Broker gets the OTrP message buffer and sends an HTTP POST request to the TAM URI, with the OTrP message in the body:

```
POST /tam HTTP/1.1
Host: example.com
Accept: application/otrp+json
Content-Type: application/otrp+json
Content-Length: [length of OTrP message here]
User-Agent: Foo/1.0
```

```
[OTrP message here]
```

11. The TAM Broker receives the HTTP POST request, and calls the TAM's "ProcessOTrPMessage" API.
12. Steps 6-11 are then repeated until the TAM passes no data back to the TAM Broker in step 6.
13. The TAM Broker sends an HTTP successful response with no body:

```
HTTP/1.1 204 No Content
Server: Bar/2.2
```

14. The TEEP Broker deletes its session state.

8. Security Considerations

Although OTrP is protected end-to-end inside of HTTP, there is still value in using HTTPS for transport, since HTTPS can provide additional protections as discussed in Section 6 of [\[I-D.ietf-httpbis-bcp56bis\]](#). As such, Broker implementations MUST support HTTPS. The choice of HTTP vs HTTPS at runtime is up to policy, where an administrator configures the TAM URI to be used, but it is expected that real deployments will always use HTTPS TAM URIs.

9. IANA Considerations

This document has no actions for IANA.

10. References

10.1. Normative References

[I-D.ietf-httpbis-semantic]

Fielding, R., Nottingham, M., and J. Reschke, "HTTP Semantics", [draft-ietf-httpbis-semantic-05](#) (work in progress), July 2019.

[I-D.ietf-teep-opentrustprotocol]

Pei, M., Atyeo, A., Cook, N., Yoo, M., and H. Tschofenig, "The Open Trust Protocol (OTrP)", [draft-ietf-teep-opentrustprotocol-03](#) (work in progress), May 2019.

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

[RFC2818] Rescorla, E., "HTTP Over TLS", [RFC 2818](#), DOI 10.17487/RFC2818, May 2000, <<https://www.rfc-editor.org/info/rfc2818>>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in [RFC 2119](#) Key Words", [BCP 14](#), [RFC 8174](#), DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

10.2. Informative References

[I-D.ietf-httpbis-bcp56bis]

Nottingham, M., "Building Protocols with HTTP", [draft-ietf-httpbis-bcp56bis-08](#) (work in progress), November 2018.

[I-D.ietf-teep-architecture]

Pei, M., Tschofenig, H., Wheeler, D., Atyeo, A., and D. Liu, "Trusted Execution Environment Provisioning (TEEP) Architecture", [draft-ietf-teep-architecture-02](#) (work in progress), March 2019.

Thaler

Expires January 9, 2020

[Page 10]

Internet-Draft

OTrP HTTP Transport

July 2019

Author's Address

Dave Thaler
Microsoft

EMail: dthaler@microsoft.com

Thaler

Expires January 9, 2020

[Page 11]