Workgroup: TEEP WG
Internet-Draft:
draft-ietf-teep-otrp-over-http-14
Published: 14 October 2022
Intended Status: Standards Track
Expires: 17 April 2023
Authors: D. Thaler
Microsoft

HTTP Transport for Trusted Execution Environment Provisioning: Agent Initiated Communication

Abstract

The Trusted Execution Environment Provisioning (TEEP) Protocol is used to manage code and configuration data in a Trusted Execution Environment (TEE). This document specifies the HTTP transport for TEEP communication where a Trusted Application Manager (TAM) service is used to manage code and data in TEEs on devices that can initiate communication to the TAM.

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 https://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 17 April 2023.

Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction

A Trusted Execution Environment (TEE) is an environment that enforces that any code within that environment cannot be tampered with, and that any data used by such code cannot be read or tampered with by any code outside that environment. The Trusted Execution Environment Provisioning (TEEP) protocol is designed to provision authorized code and configuration into TEEs.

To be secure against malware, a TEEP implementation (referred to as a TEEP "Agent" on the client side, and a "Trusted Application Manager (TAM)" on the server side) SHOULD themselves run inside a TEE, although a TAM running outside a TEE is also supported. However, the transport for TEEP, along with the underlying TCP/IP stack, does not necessarily 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 or QUIC [RFC9000]) that only

sees encrypted messages, to be kept out of the TEE. See section 6.2 of [I-D.ietf-teep-architecture] for a depiction of various implementation models.

The TEEP specification [I-D.ietf-teep-protocol] describes the behavior of TEEP Agents and TAMs, but does not specify the details of the transport. The purpose of this document is to provide such details. That is, a TEEP-over-HTTP (TEEP/HTTP) implementation delivers messages up to a TEEP implementation, and accepts messages from the TEEP implementation to be sent over a network. The TEEP-over-HTTP implementation can be implemented either outside a TEE (i.e., in a TEEP "Broker") or inside a TEE.

There are two topological scenarios in which TEEP could be deployed:

1. TAMs are reachable on the Internet, and Agents are on networks that might be behind a firewall or stateful NAT, so that communication must be initiated by an Agent. Thus, the Agent has an HTTP Client and the TAM has an HTTP Server.

2. Agents are reachable on the Internet, and TAMs are on networks that might be behind a firewall or stateful NAT, so that communication must be initiated by a TAM. Thus, the Agent has an HTTP Server and the TAM has an HTTP Client.

The remainder of this document focuses primarily on the first scenario as depicted in Figure 1, but some sections (Section 4 and Section 8) may apply to the second scenario as well. A more complete discussion of the second scenario may be handled by a separate document.

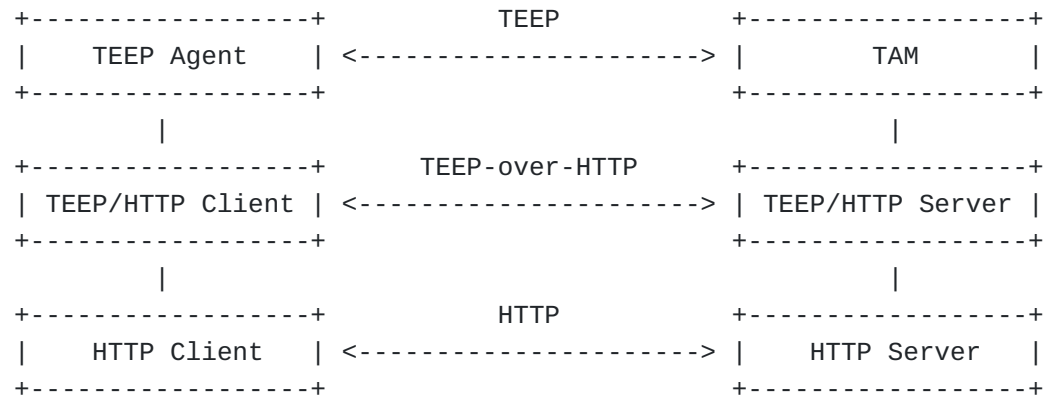

Figure 1: Agent Initiated Communication

This document specifies the middle layer (TEEP-over-HTTP), whereas the top layer (TEEP) is specified in [I-D.ietf-teep-protocol] and the bottom layer (HTTP) is specified in [RFC9110].

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, TEEP Broker, and Rich Execution Environment (REE).

3. TEEP Broker

Section 6 of the TEEP architecture [I-D.ietf-teep-architecture] defines a TEEP "Broker" as being a component on the device, but outside the TEE, that facilitates communication with a TAM. That document further explains that the protocol layer at which the TEEP broker operates may vary by implementation, and it depicts several exemplary models. An implementation is free to choose any of these models, although model A is the one we will use in our examples.

Passing information from an REE component to a TEE component is typically spoken of as being passed "in" to the TEE, and information passed in the opposite direction is spoken of as being passed "out". In the protocol layering sense, information is typically spoken of as being passed "up" or "down" the stack. Since the layer at which information is passed in/out may vary by implementation, we will generally use "up" and "down" in this document.

3.1. Use of Abstract APIs

This document refers to various APIs between a TEEP implementation and a TEEP/HTTP 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.

Some TEE architectures (e.g., SGX) may support API calls both into and out of a TEE. In other TEE architectures, there may be no calls out from a TEE, but merely data returned from calls into a TEE. This document attempts to be agnostic as to the concrete API architecture for Broker/Agent communication. Since in model A, the Broker/Agent communication is done at the layer between the TEEP and TEEP/HTTP implementations, and there may be some architectures that do not support calls out of the TEE (which would be downcalls from TEEP in model A), we will refer to passing information up to the TEEP implementation as API calls, but will simply refer to "passing data"

back down from a TEEP implementation. A concrete API might pass data back via an API downcall or via data returned from an API upcall.

This document will also refer to passing "no" data back out of a TEEP implementation. In a concrete API, this might be implemented by not making any downcall, or by returning 0 bytes from an upcall, for example.

4. Use of HTTP as a Transport

This document uses HTTP [RFC9110] as a transport. For the motivation behind the HTTP recommendations in this document, see the discussion of HTTP as a transport in [RFC9205].

Redirects MUST NOT be automatically followed. Cookies are not used.

Content is not intended to be treated as active by browsers and so HTTP responses with content SHOULD have the following header fields as explained in Section 4.13 of [RFC9205] (using the TEEP media type defined in [I-D.ietf-teep-protocol]):

```
Content-Type: application/teep+cbor
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. Since POST responses without explicit freshness information are uncacheable (see Section 9.3.3 of [RFC9110]), no Cache-Control header is needed.

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 a TEEP Agent via an out-of-band mechanism, as discussed in the next section.

It is strongly RECOMMENDED that implementations use HTTPS. Although TEEP 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 Sections 4.4.2 and 6 of [RFC9205].

However, there may be constrained nodes where code space is an issue. [RFC7925] provides TLS profiles that can be used in many constrained nodes, but in rare cases the most constrained nodes might need to use HTTP without a TLS stack, relying on the end-to-end security provided by the TEEP protocol. See Sections 4.4.2 and 6 of [RFC9205] for more discussion of additional security considerations that apply in this case.

When HTTPS is used, clients MUST use the procedures detailed in Section 4.3.4 of [RFC9110] to verify the authenticity of the server.

See [BCP195] for additional TLS recommendations and [RFC7925] for TLS recommendations related to IoT devices.

5. TEEP/HTTP Client Behavior

5.1. Receiving a request to install a new Trusted Application

In some environments, an application installer can determine (e.g., from an application manifest) that the application being installed or updated has a dependency on a given Trusted Application (TA) being available in a given type of TEE. In such a case, it will notify a TEEP Broker, where the notification will contain the following:

* A unique identifier of the TA

* Optionally, any metadata to provide 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 a TEEP Broker receives such a notification, it 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. Once the TEEP Broker picks a TEE, it passes the notification to the TEEP/HTTP Client for that TEE.

The TEEP/HTTP Client 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/HTTP Client 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 TEEP/HTTP Client, depending on the TEEP Agent's configuration. If they differ, the TEEP/HTTP Client MUST use the TAM URI passed back.

5.1.1. Session Creation

If no data is passed back, the TEEP/HTTP Client simply informs its caller (e.g., the application installer) of success.

If the TEEP Agent passes back a TAM URI with no message buffer, the TEEP/HTTP Client attempts to create session state, then sends an HTTP(S) POST to the TAM URI with an Accept header field with the TEEP media type specified in [I-D.ietf-teep-protocol], and an empty body. The HTTP request is then associated with the TEEP/HTTP Client's session state.

If the TEEP Agent instead passes back a TAM URI with a message buffer, the TEEP/HTTP Client attempts to create session state and handles the message buffer as specified in Section 5.3.

Session state consists of:

* Any context (e.g., a handle) that the TEEP Agent wishes to be provided back to it in any later conceptual API calls into it related to this session.

* Any context that identifies an HTTP request, if one is outstanding. Initially, none exists.

5.2. Receiving a notification that a Trusted Application is no longer needed

In some environments, an application installer can determine (e.g., from an application manifest) that a given Trusted Application is no longer needed, such as if the application that previously depended on the TA is uninstalled or updated in a way that removes the dependency. In such a case, it will notify a TEEP Broker, where the notification will contain the following:

* A unique identifier of the TA

* Optionally, any metadata to provide to the TEEP Agent. This might include a TAM URI provided in the original application manifest, for example.

* Optionally, any requirements that may affect the choice of TEE, if multiple are available to the TEEP Broker.

When a TEEP Broker receives such a notification, it first identifies in an implementation-dependent way which TEE (if any) is believed to contain the TA that is no longer needed, similar to the process in Section 5.1. Once the TEEP Broker picks a TEE, it passes the notification to the TEEP/HTTP Client for that TEE.

The TEEP/HTTP Client then informs the TEEP Agent in that TEE by invoking an appropriate "UnrequestTA" API that identifies the unneeded TA. The TEEP/HTTP Client need not know whether the TEE actually has the TA installed.

Finally, the TEEP Agent responds to the TEEP/HTTP Client as in Section 5.1. Specifically, 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 TEEP/HTTP Client, depending on the TEEP Agent's configuration. If they differ, the TEEP/HTTP Client MUST use the TAM URI passed back.

Processing then continues as in Section 5.1.1.

5.3. Getting a TAM URI and message buffer back from a TEEP Agent

When a TEEP Agent passes a TAM URI and optionally a message buffer to a TEEP/HTTP Client, the TEEP/HTTP Client MUST do the following, using the TEEP/HTTP Client's session state associated with its API call to the TEEP Agent.

The TEEP/HTTP Client sends an HTTP POST request to the TAM URI with Accept and Content-Type header fields with the TEEP media type, and a body containing the TEEP message buffer (if any) provided by the TEEP Agent. The HTTP request is then associated with the TEEP/HTTP Client's session state.

5.4. Receiving an HTTP response

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

If the HTTP response body is empty, the TEEP/HTTP Client'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/HTTP Client passes (e.g., using the "ProcessTeepMessage" API as mentioned in Section 6.2.1 of [I-D.ietf-teep-architecture]) the response body up to the TEEP Agent associated with the session. The TEEP Agent will then either pass no data back, or pass back a message buffer.

If no data is passed back, the TEEP/HTTP Client's task is complete, and it can delete its session state, and inform its caller (e.g., the application installer) of success.

If instead the TEEP Agent passes back a message buffer, the TEEP/HTTP Client handles the message buffer as specified in Section 5.3.

5.5. Handling checks for policy changes

An implementation MUST provide a way to periodically check for TAM policy changes, such as a Trusted Application needing to be deleted from a TEE because it is no longer permitted, or needing to be

updated to a later version. This can be done in any implementation-specific manner, such as any of the following or a combination thereof:

A) The TEEP/HTTP Client might call up to the TEEP Agent at an interval previously specified by the TEEP Agent. This approach requires that the TEEP/HTTP Client be capable of running a periodic timer.

B) The TEEP/HTTP Client might be informed when an existing TA is invoked, and call up to 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/HTTP Client might be informed when any attestation attempt determines that the device is out of compliance, and call up to the TEEP Agent to remediate.

The TEEP/HTTP Client 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.

The TEEP Agent might need to talk to multiple TAMs, however, as shown in Figure 1 of [I-D.ietf-teep-architecture]. To accomplish this, the TEEP/HTTP Client keeps invoking the "RequestPolicyCheck" API until the TEEP Agent passes no data back, so that the TEEP Agent can return each TAM URI in response to a separate API call.

5.6. Error handling

If any local error occurs where the TEEP/HTTP Client cannot get a message buffer (empty or not) back from the TEEP Agent, the TEEP/HTTP Client deletes its session state, and informs its caller (if any, 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/HTTP Client calls the TEEP Agent's "ProcessError" API, and then deletes its session state and informs its caller of a failure.

6. TEEP/HTTP Server Behavior

6.1. Receiving an HTTP POST request

If the TAM does not receive the appropriate Content-Type header field value, the TAM SHOULD fail the request, returning a 415 Unsupported Media Type response. Similarly, if an appropriate Accept header field is not present, the TAM SHOULD fail the request with an

appropriate error response. (This is for consistency with common implementation practice to allow the HTTP server to choose a default error response, since in some implementations the choice is done at the HTTP layer rather than the layer at which TEEP-over-HTTP would be implemented.) Otherwise, processing continues as follows.

When an HTTP POST request is received with an empty body, this indicates a request for a new TEEP session, and the TEEP/HTTP Server invokes the TAM's "ProcessConnect" API. The TAM will then pass back a message buffer.

When an HTTP POST request is received with a non-empty body, this indicates a message on an existing TEEP session, and the TEEP/HTTP Server passes the request body to the TAM (e.g., using the "ProcessTeepMessage" API mentioned in [I-D.ietf-teep-architecture]). 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 TEEP/HTTP Server sends a successful (2xx) response with no body. It SHOULD be status 204 (No Content).

6.3. Getting a message buffer from the TAM

If the TAM passes back a non-empty buffer, the TEEP/HTTP Server generates a successful (2xx) response with a Content-Type header field with the TEEP media type, and with the message buffer as the body.

6.4. Error handling

If any error occurs where the TEEP/HTTP Server cannot get a message buffer (empty or not) back from the TAM, the TEEP/HTTP Server generates an appropriate HTTP 5xx error response.

7. Sample message flow

The following shows a sample TEEP message flow that uses application/teep+cbor as the Content-Type.

1. An application installer determines (e.g., from an application manifest) that the application has a dependency on TA "X", and passes this notification to the TEEP Broker. The TEEP Broker picks a TEE (e.g., the only one available) based on this notification, and passes the information to the TEEP/HTTP Cient for that TEE.

2. The TEEP/HTTP Client 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 back the TAM URI (e.g., "https://example.com/tam") to the TEEP/HTTP Client.

4. The TEEP/HTTP Client sends an HTTP POST request to the TAM URI:

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

   where the TEEP/HTTP Client fills in an implementation-specific value in the User-Agent header field.

5. On the TAM side, the TEEP/HTTP Server receives the HTTP POST request, and calls the TAM's "ProcessConnect" API.

6. The TAM generates a TEEP message (where typically QueryRequest is the first message) and passes it to the TEEP/HTTP Server.

7. The TEEP/HTTP Server sends an HTTP successful response with the TEEP message in the body:

```
HTTP/1.1 200 OK
Content-Type: application/teep+cbor
Content-Length: [length of TEEP message here]
Server: Bar/2.2
X-Content-Type-Options: nosniff
Content-Security-Policy: default-src 'none'
Referrer-Policy: no-referrer

[TEEP message here]
```

   where the TEEP/HTTP Server fills in an implementation-specific value in the Server header field.

8. Back on the TEEP Agent side, the TEEP/HTTP Client gets the HTTP response, extracts the TEEP message and passes it up to the TEEP Agent.

9. The TEEP Agent processes the TEEP message, and generates a TEEP response (e.g., QueryResponse) which it passes back to the TEEP/HTTP Client.

10. The TEEP/HTTP Client gets the TEEP message buffer and sends an HTTP POST request to the TAM URI, with the TEEP message in the body:

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

[TEEP message here]
```

11. The TEEP/HTTP Server receives the HTTP POST request, and passes the payload up to the TAM.

12. Steps 6-11 are then repeated until the TAM passes no data back to the TEEP/HTTP Server in step 6.

13. The TEEP/HTTP Server sends an HTTP successful response with no body:

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

14. The TEEP/HTTP Client deletes its session state.

8. Security Considerations

Section 4 discussed security recommendations for HTTPS transport of TEEP messages. See Section 6 of [RFC9205] for additional discussion of HTTP(S) security considerations. See section 9 of [I-D.ietf-teep-architecture] for security considerations specific to the use of TEEP.

9. IANA Considerations

This document has no actions for IANA.

10. References

10.1. Normative References

[BCP195] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <https://www.rfc-editor.org/info/rfc7525>.

[I-D.ietf-teep-protocol] Tschofenig, H., Pei, M., Wheeler, D. M., Thaler, D., and A. Tsukamoto, "Trusted Execution Environment Provisioning (TEEP) Protocol", Work in Progress, Internet-Draft, draft-ietf-teep-protocol-10, 28

July 2022, <https://www.ietf.org/archive/id/draft-ietf-teep-protocol-10.txt>.

[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>.

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <https://www.rfc-editor.org/info/rfc7925>.

[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>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

10.2. Informative References

[I-D.ietf-teep-architecture] Pei, M., Tschofenig, H., Thaler, D., and D. M. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", Work in Progress, Internet-Draft, draft-ietf-teep-architecture-18, 11 July 2022, <https://www.ietf.org/archive/id/draft-ietf-teep-architecture-18.txt>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[RFC9205] Nottingham, M., "Building Protocols with HTTP", BCP 56, RFC 9205, DOI 10.17487/RFC9205, June 2022, <https://www.rfc-editor.org/info/rfc9205>.

Author's Address

Dave Thaler
Microsoft

Email: dthaler@microsoft.com