TEEP WG
Internet-Draft
Intended status: Informational
Expires: October 2, 2020

D. Thaler
Microsoft

March 31, 2020

HTTP Transport for Trusted Execution Environment Provisioning: Agent-to-TAM Communication

draft-ietf-teep-otrp-over-http-05

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 TEEs in devices that can initiate communication to the TAM. An implementation of this document can (if desired) run outside of any TEE, but interacts with a TEEP 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 October 2, 2020.

Table of Contents

1. Introduction

Trusted Execution Environments (TEEs), including environments based on 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 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) must themselves run inside a TEE. 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) that only sees encrypted messages, to be kept out of the TEE.

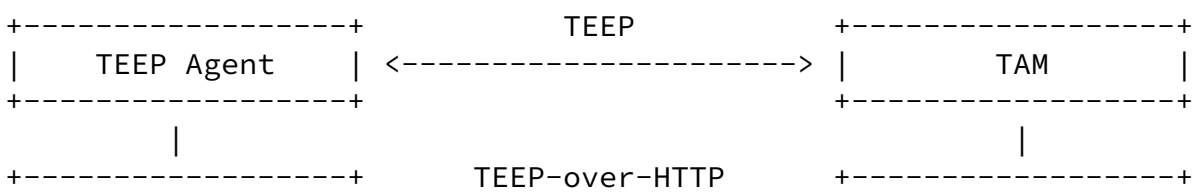
The TEEP specification [I-D.ietf-teep-protocol] (like its predecessors [I-D.ietf-teep-opentrustprotocol] and [GP-OTrP]) describes the behavior of TEEP Agents and TAMs, but does not specify the details of the transport. The purpose of this document is to provide such details. That is, a TEEP-over-HTTP (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, 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, 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 fuller discussion of the second scenario may be handled by a separate document.



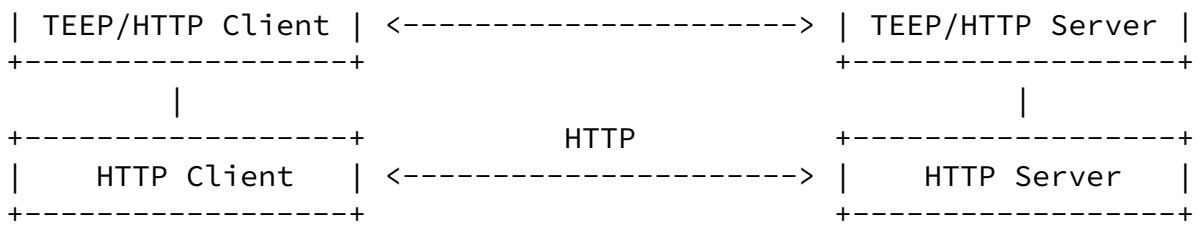


Figure 1: Agent-to-TAM Communication

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, and Rich Execution Environment (REE).

3. TEEP Broker Models

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. As depicted in Figure 2, there are multiple ways in which this can be implemented, with more or fewer layers being inside the TEE. For example, in model A, the model with the smallest TEE footprint, only the TEEP implementation is inside the TEE, whereas the TEEP/HTTP implementation is in the TEEP Broker outside the TEE.

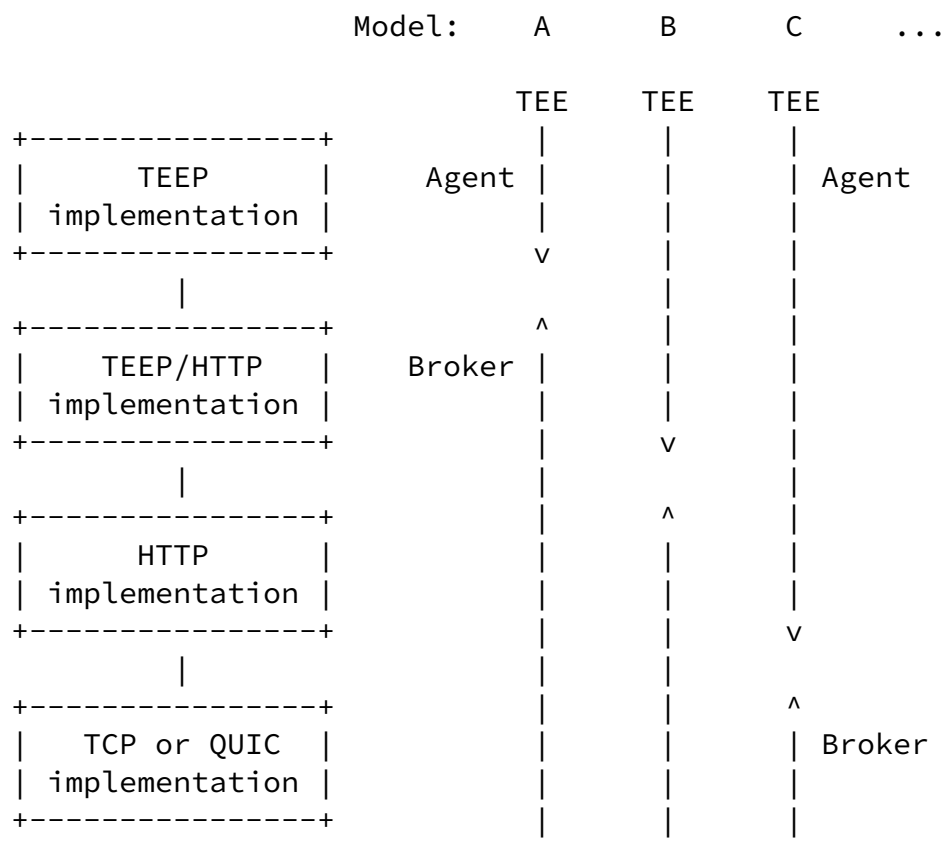

Model: A B C ...
TEE TEE TEE
TEEP implementation
Agent
Agent
TEEP/HTTP implementation
Broker
HTTP implementation
TCP or QUIC implementation
Broker

Figure 2: TEEP Broker Models

In other models, additional layers are moved into the TEE, increasing the TEE footprint, with the Broker either containing or calling the topmost protocol layer outside of the TEE. 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 informaton 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 [I-D.ietf-httpbis-semantics] 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 TEEP.

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 TEEP content type per the TEEP 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 a 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/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 app 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 implementation. 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 implementation 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 implementation 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 implementation'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 implementation 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 and an

empty body. The HTTP request is then associated with the TEEP/HTTP

Client's session state.

If the TEEP implementation 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 5.2.

Session state consists of:

- Any context (e.g., a handle) that identifies the API session with the TEEP implementation.

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

5.2. Getting a message buffer back from a TEEP implementation

When a TEEP implementation passes a message buffer (and TAM URI) 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 implementation.

The TEEP/HTTP Client sends an HTTP POST request to the TAM URI with Accept and Content-Type headers with the TEEP media type in use, and a body containing the TEEP message buffer provided by the TEEP implementation. The HTTP request is then associated with the TEEP/HTTP Client'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/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 "ProcessTeepMessage" API as mentioned in Section 6.2.1 of [I-D.ietf-teep-architecture]) the response body up to the TEEP implementation associated with the session. The TEEP implementation 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 implementation passes back a message buffer, the TEEP/HTTP Client 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 TEEP policy changes. This can be done in any implementation-specific manner, such as:

A) The TEEP/HTTP Client might call up to the TEEP implementation at an interval previously specified by the TEEP implementation. 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 implementation if more time has passed than was previously specified by the TEEP implementation. 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 implementation to remediate.

The TEEP/HTTP Client informs the TEEP implementation by invoking an appropriate "RequestPolicyCheck" API. The TEEP implementation 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/HTTP Client cannot get a message buffer (empty or not) back from the TEEP implementation, the TEEP/HTTP Client deletes its session state, and informs its caller (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 implementation'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

When an HTTP POST request is received with an empty body, the TEEP/HTTP Server 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 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 TEEP implementation

If the TEEP implementation passes back an empty buffer, the TEEP/HTTP Server sends a successful (2xx) response with no body.

6.3. Getting a message buffer from the TEEP implementation

If the TEEP implementation passes back a non-empty buffer, the TEEP/HTTP Server generates a successful (2xx) response with a Content-Type header with the appropriate media type in use, 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 TEEP implementation, the TEEP/HTTP Server generates an appropriate HTTP 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 app 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 implementation's "RequestTA" API, passing TA Needed = X.

3. The TEEP implementation 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/HTTP Client. (If the TEEP implementation already

had a cached TAM certificate that it trusts, it could skip to step 9 instead and generate a QueryResponse.)

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
```

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

6. The TEEP implementation 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
Cache-Control: no-store
X-Content-Type-Options: nosniff
```

```
Content-Security-Policy: default-src 'none'
Referrer-Policy: no-referrer

[TEEP message here]
```

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

9. The TEEP implementation 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 implementation.

12. Steps 6-11 are then repeated until the TEEP implementation 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

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 Section 6 of [I-D.ietf-httpbis-bcp56bis]. As such, TEEP/HTTP 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-semantics]
Fielding, R., Nottingham, M., and J. Reschke, "HTTP Semantics", draft-ietf-httpbis-semantics-07 (work in progress), March 2020.

[I-D.ietf-teep-protocol]
Tschofenig, H., Pei, M., Wheeler, D., and D. Thaler, "Trusted Execution Environment Provisioning (TEEP) Protocol", draft-ietf-teep-protocol-01 (work in progress), March 2020.

[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

[GP-OTrP] Global Platform, "TEE Management Framework: Open Trust Protocol (OTrP) Profile Version 1.0", Global Platform GPD_SPE_123, May 2019, <https://globalplatform.org/specs-library/tee-management-framework-open-trust-protocol/>.

[I-D.ietf-httpbis-bcp56bis] Nottingham, M., "Building Protocols with HTTP", draft-ietf-httpbis-bcp56bis-09 (work in progress), November 2019.

[I-D.ietf-teep-architecture] Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", draft-ietf-teep-architecture-07 (work in progress), March 2020.

[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.

Author's Address

Dave Thaler
Microsoft

EMail: dthaler@microsoft.com