IETF 112 SUIT TEEP Hackathon

- Date November 2 Tuesday – November 5 Friday
  Jointly with SUIT

- Participants:

  Akira Tsukamoto, AIST  
  Kuniyasu Suzuki, TRASO/AIST  
  Kohei Isobe, SECOM  
  Ken Takayama, SECOM  
  Masashi Kikuchi, TRASIO  
  Takahiko Nagata, TRASIO

  Brendan Moran, ARM  
  Hannes Tschofenig, ARM  
  Dave Thaler, Microsoft
Background and objective

Background

● TEEP requires SUIT, RATS and COSE
● Started implementing to support SUIT manifests in TEEP Update messages in TEEP protocol around IETF 111

● Would like to complete how to handle SUIT manifests on TEEP at IETF112
● After IETF112, start implement to support COSE and EAT in RATS on TEEP

Objective

● Finalizing formats of SUIT manifests in the draft after trying the implementation
● Start looking in to COSE on TEEP messages, SUIT manifests already handling COSE, so use it as much as possible in the implementation
What got done

Implementations

- TAM: tamproto: Distribute the TC binary integrated in the SUIT manifest in TEEP Update
- libcsuit: Supporting three examples of suit manifests
- TEEP-device: Update TEEP message formats from draft-05 to 06
  - Adding handling COSE on SUIT manifests
  - Adding handling 'integrated-payload' (not complete)

Feedback on drafts

- TEEP: When to include token or not in Update message
  - https://github.com/ietf-teep/teep-protocol/issues/166
- TEEP: Proposing cipher-suites to use cddl description in SUIT for TEEP
- TEEP: Revising the suit manifest for deleting TC by using 'unlink'
  - https://github.com/ietf-teep/teep-protocol/pull/169
- TEEP: Adding three examples of SUIT manifest formats
  - https://github.com/ietf-teep/teep-protocol/pull/161
- SUIT: One line patch for defining ES256 used in SUIT examples
  - https://github.com/suit-wg/manifest-spec/pull/50
Three SUIT manifest formats, Example 1 (1/2)

Having one SUIT Manifest pointing to a URI of a Trusted Component Binary

- **Pros:**
  - The Trusted Component Developer can ensure that the intact Trusted Component Binary is downloaded by TEEP Devices
  - The TAM does not have to deliver Update message containing Trusted Component Binary which may have a large size

- **Cons:**
  - The Trusted Component Developer must host the Trusted Component Binary server
  - The TEEP Device must fetch the Trusted Component Binary in another connection after receiving an Update message
Three SUIT manifest formats, Example 1 (2/2)

```
<table>
<thead>
<tr>
<th>TAM</th>
<th>TEEP Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update</td>
<td></td>
</tr>
</tbody>
</table>

```

```
TEGR Message(
TEEP-TYPE-update,
   options:
      manifest-list: [
        += suit-manifest "tc-uuid.suit" (TC Developer) =>
        SUIT_Envelope({
          manifest: {
            install: {
              set-parameter: {
                uri: "https://tc.org/tc-uuid.ta"
              },
              fetch
            }
          }
        })
      ]
    ]
  ]
)
```

and then,

```
<table>
<thead>
<tr>
<th>TEEP Agent</th>
<th>TC Developer</th>
</tr>
</thead>
</table>

```

```
fetch "https://tc.org/tc-uuid.ta"'
```

```
+-------- tc-uuid.ta +--------+
| 48 65 6C 6C 6F 2C 20 ... |
+--------------------------+
```
Having a SUIT Manifest include the Trusted Component Binary

● **Pros:**
  ○ The TEEP Device can obtain the Trusted Component Binary and its SUIT manifest together in one Update message
  ○ The Trusted Component Developer does not have to host a server to deliver the Trusted Component Binary directly to TEEP Devices

● **Cons:**
  ○ The TAM must host the Trusted Component Binary itself, rather than delegating such storage to the Trusted Component Developer
  ○ The TAM must deliver Trusted Component Binaries with integrated SUIT manifest in Update messages, which may result increasing the Update message size
Three SUIT manifest formats, Example 2 (2/2)
Three SUIT manifest formats, Example 3

Supplying Personalization Data for the installed Trusted Component Binary
Format examples expanding in the draft

- Indispensable to keep compatibilities of implementation among different vendors
- Require breakdowns of binary representations when debugging between different TAM and TEEP devices
- It is good CBOR tutorial
D.1.1. CBOR Diagnostic Notation

```
query-request = /
[
  1: / type: TEEP-TYPE-query-request = 1 (uint (0..23)) /
  / options: /
  /
  20: 0xa0a1a2a3a4a5a6a7a8a9a9aabacadaeaf, 
    / token = 20 (mapkey): 
    h'a0a1a2a3a4a5a6a7a8a9a9aabacadaeaf' (bstr . size (8..64)), 
    generated by TAM / 
  1: [ 1 ], / supported-cipher-suites = 1 (mapkey): 
    TEEP-AES-CCM-16-64-128-HMAC256--256-X25519-EdDSA =,
    [ 1 ], (array of .within uint .size 4) /
  3: [ 0 ], / version = 3 (mapkey): 
    [ 0 ], (array of .within uint .size 4) /
], /
3 / data-item-requested: /
  attestation | trusted-components = 3 (.within uint .size 8) /
]
```

D.1.2. CBOR Binary Representation

```
83
  01
  A4
  14
  4F
  A0A1A2A3A4A5A6A7A8A9A9AABACADAeAF
  01
  81
  01
  03
  81
  00
  04
  43
  010203
  03
```

# array(3)
# unsigned(1) uint (0..23)
# map(4)
# unsigned(20) uint (0..23)
# bytes(16) (8..64)
# A0A1A2A3A4A5A6A7A8A9A9AABACADAeAF
# unsigned(1) uint (0..23)
# array(1)
# unsigned(1) within uint .size 4
# unsigned(3) uint (0..23)
# array(1)
# unsigned(0) within uint .size 4
# unsigned(4) uint (0..23)
# bytes(3)
# "x01"x02"x03"
# unsigned(3) .within uint .size 8
SUIT message format examples

E.2. Example 2: SUIT Manifest including the Trusted Component Binary

```c
// SUIT_Envelope_Tagged / 107 { 
// suit-authentication-wrapper / 2: << [ 
// << [ 
// suit-digest-algorithm-id: / -16 / cose-alg-sha256 / 
// suit-digest-bytes: / h' C8363BDF3DCF68F0234A9DD320C2FEA72DE68F46AAE7CE700AFF: 
// ]] >> / COSE_Sign1_Tagged / 18 { 
// protected: / << [ 
// algorithm-id / 1: -7 / ES256 / 
// ]] >> ] 
// unprotected: / }} 
// payload: / null, 
// signature: / h'E0D2973A7B7185BBDA108458FB68E6AF65C0C' 
// ]} >> ]
// suit-integrated-payload / "to": h' 48656C6C66F2C2053656367 
// suit-manifest / 3: << [ 
// suit-manifest-version / 1: 1, 
// suit-manifest-sequence-number / 2: 3, 
// suit-common / 3: << [ 
// suit-components / 2: [ 
// h'544545502D4455628Y696365', 
// "TEEP-Devic" 
// h'53656E75726566653', 
// "SecureFS" 
// h'8DB2573A926D4754935332DC29997F74', 
// "tc-uuid" 
// h'7461' 
// "ta" 
```

---

# tag(107) / SUIT_Envelope_Tagged /
# map(3) /
# unsigned(2) / suit-authentication-wrapper /
# bytes(115) /
# array(2) /
# bytes(36) /
# array(2) /
# negative(15) / -16 = cose-alg-sha256 /
# bytes(32) /
# bytes(74) /
# tag(18) / COSE_Sign1_Tagged /
# array(4) /
# bytes(3) /
# map(1) /
# unsigned(1) / algorithm-id /
# negative(6) / -7 = ES256 /
# map(0) /
# primitive(22) / null /
# bytes(64) /
---

E0D2973A7B7185BBDA108458FB68E6AF65C0D31F2283E784129A5D4229F0E8B11F8947D3E!
Summary

- Moving from purifying TEEP message itself, to defining and handling SUIT and COSE in TEEP message formats
- Implementation progressed for supporting SUIT manifests
- Draft updated related on SUIT and COSE from the hackathon
- Explaining three formats of SUIT manifests
- How do the examples looks of CBOR in TEEP
- After 112, many remining on COSE and EAT on TEEP protocol

A part of this hackathon presentation is based on results obtained from a project, JPNP16007, commissioned by the New Energy and Industrial Technology Development Organization (NEDO).