Manifest

draft-moran-suit-manifest-03

Usability & Threat Model

- Criteria in draft-moran-suit-architecture, Appendix A
- The manifest format is designed to fulfill certain usability criteria
- The manifest format is to mitigate threats against firmware update solutions

MANIFEST ENCODING

Manifest Encoding

- Initially specified in ASN.1/DER. Used CMS-based security wrapper.
  - Not well received based on mailing list feedback.
- Changed to CBOR/COSE. Described in CDDL.
- Is everyone happy now?

Design Decision

MANIFEST ATTRIBUTES

Use of CBOR maps

- Integers are used for indices
- Positive integers reserved for standard values
- Negative integers reserved for customization

Manifest CDDL

```
Manifest = [
    manifestVersion : uint,
    text : {* int => tstr } / nil,
    nonce : bstr,
    timestamp : uint,
    conditions: [ * condition ],
    directives: [ * directive ] / nil,
    aliases: [ * ResourceReference ] / nil,
    dependencies: [ * ResourceReference ] / nil,
    extensions: { * int => bstr } / nil,
    payloadInfo: ? PayloadInfo
]
```

Version number of the manifest format

Manifest CDDL

```
Manifest = [
    manifestVersion : uint,
    text : {* int => tstr } / nil,
    nonce : bstr,
    timestamp : uint,
    conditions: [ * condition ],
    directives: [ * directive ] / nil,
    aliases: [ * ResourceReference ] / nil,
    dependencies: [ * ResourceReference ] / nil,
    extensions: { * int => bstr } / nil,
    payloadInfo: ? PayloadInfo
]
```

Optional, textual description of the Update.

Manifest CDDL

```
Manifest = [
    manifestVersion : uint,
    text : {* int => tstr } / nil,
    nonce : bstr,
    timestamp : uint,
    conditions: [ * condition ],
    directives: [ * directive ] / nil,
    aliases: [ * ResourceReference ] / nil,
    dependencies: [ * ResourceReference ] / nil,
    extensions: { * int => bstr } / nil,
    payloadInfo: ? PayloadInfo
]
```

Random value to ensure that every manifest is unique.

Manifest CDDL

```
Manifest = [
    manifestVersion : uint,
    text : {* int => tstr } / nil,
    nonce : bstr,
    timestamp : uint,
    conditions: [ * condition ],
    directives: [ * directive ] / nil,
    aliases: [ * ResourceReference ] / nil,
    dependencies: [ * ResourceReference ] / nil,
    extensions: { * int => bstr } / nil,
    payloadInfo: ? PayloadInfo
]
```

Monotonic sequence number implemented as a UTC timestamp.

Used for rollback protection.

Manifest CDDL

1. Vendor ID
2. Class ID
3. Device ID
4. Best Before

```
= [
    manifestVersion : uint,
    text : {* int => tstr } / nil,
    nonce : bstr,
    timestamp : uint,
    conditions: [ * condition ],
    directives: [ * directive ] / nil,
    aliases: [ * ResourceReference ] / nil,
    dependencies: [ * ResourceReference ] / nil,
    extensions: { * int => bstr } / nil,
    oadInfo: ? PayloadInfo
```

Used to construct IF … THEN … Rules

1. Apply Immediately
2. Apply After

Manifest CDDL

```
Manifest = [
    manifestVersion : uint,
    text : {* int => tstr } / nil,
    nonce : bstr,
    timestamp : uint,
    conditions: [ * condition ],
    directives: [ * directive ] / nil,
    aliases: [ * ResourceReference ] / nil,
    dependencies: [ * ResourceReference ] / nil,
    extensions: { * int => bstr } / nil,
    payloadInfo: ? PayloadInfo
]
```

Used to refer to alternative locations of the firmware image

Manifest CDDL

```
Manifest = [
    manifestVersion : uint,
    text : {* int => tstr } / nil,
    nonce : bstr,
    timestamp : uint,
    conditions: [ * condition ],
    directives: [ * directive ] / nil,
    aliases: [ * ResourceReference ] / nil,
    dependencies: [ * ResourceReference ] / nil,
    extensions: { * int => bstr } / nil,
    payloadInfo: ? PayloadInfo
]
```

To express the requirement that more than one image has to be installed on a device

Payload CDDL

```
PayloadInfo = [
    format = [
        type: int,
        ? parameters : bstr
    ],
    size: uint,
    storageIdentifier: bstr,
    uris: [*[
        rank: int,
        uri: tstr
    ]] / nil,
    digestAlgorithm = [
        type : int,
        ? parameters: bstr
    ] / nil,
    digests = {* int => bstr} / nil,
    payload = COSE_Encrypt / bstr / nil
]
```

Format of the binary

Payload CDDL

```
PayloadInfo = [
    format = [
        type: int,
        ? parameters : bstr
    ],
    size: uint,
    storageIdentifier: bstr,
    uris: [*[
        rank: int,
        uri: tstr
    ]] / nil,
    digestAlgorithm = [
        type : int,
        ? parameters: bstr
    ] / nil,
    digests = {* int => bstr} / nil,
    payload = COSE_Encrypt / bstr / nil
]
```

Size of the firmware image in bytes

Payload CDDL

```
PayloadInfo = [
    format = [
        type: int,
        ? parameters : bstr
    ],
    size: uint,
    storageIdentifier: bstr,
    uris: [*[
        rank: int,
        uri: tstr
    ]] / nil,
    digestAlgorithm = [
        type : int,
        ? parameters: bstr
    ] / nil,
    digests = {* int => bstr} / nil,
    payload = COSE_Encrypt / bstr / nil
]
```

Indicates where the image should be placed on the device

Useful when device contains multiple MCUs and requires multiple firmware images.

Payload CDDL

```
PayloadInfo = [
    format = [
        type: int,
        ? parameters : bstr
    ],
    size: uint,
    storageIdentifier: bstr,
    uris: [*[
        rank: int,
        uri: tstr
    ]] / nil,
    digestAlgorithm = [
        type : int,
        ? parameters: bstr
    ] / nil,
    digests = {* int => bstr} / nil,
    payload = COSE_Encrypt / bstr / nil
]
```

A set of ranked references for where to find the payload.

Payload CDDL

```
PayloadInfo = [
    format = [
        type: int,
        ? parameters : bstr
    ],
    size: uint,
    storageIdentifier: bstr,s
    uris: [*[
        rank: int,
        uri: tstr
    ]] / nil,
    digestAlgorithm = [
        type : int,
        ? parameters: bstr
    ] / nil,
    digests = {* int => bstr} / nil,
    payload = COSE_Encrypt / bstr / nil
]
```

Fingerprint computed over the firmware image using the indicated algorithm.

Payload CDDL

```
PayloadInfo = [
    format = [
        type: int,
        ? parameters : bstr
    ],
    size: uint,
    storageIdentifier: bstr,
    uris: [*[
        rank: int,
        uri: tstr
    ]] / nil,
    digestAlgorithm = [
        type : int,
        ? parameters: bstr
    ] / nil,
    digests = {* int => bstr} / nil,
    payload = COSE_Encrypt / bstr / nil
]
```

Attached firmware image

Proposals for changes

- Current aliases:

```
aliases: [ * ResourceReference ] / nil,
```

- Proposed aliases:

```
aliases: [*[digest: bstr,
      uris: [
                rank: int,
                uri: tstr
       ]]] / nil,
```