arm Research

SUIT Manifest Tutorial

Brendan Moran

17/02/2020

Agenda

- What SUIT is
- How SUIT works
- Complexity
- Generating Manifests with the SUIT tool

SUIT – Motivation

Assumptions:

- Networked devices must be kept up to date
- Cohesively updating devices via diverse update management systems is hard
- Practical IoT systems will be diverse
  - Multiple vendors
  - Multiple network protocols

Goals:

- Provide a unified update format to enable unified update management systems
- Define behavior of devices performing updates to ensure predictable results

SUIT Overview

- What SUIT is
  - A document format
  - A behavioral definition for devices performing updates
  - A set of options for different use cases
  - Security guidance
- What SUIT is not
  - A protocol
  - A security checklist
  - A standard that must be fully implemented

arm Research

How SUIT Processors Work

How SUIT Works – High level

- A simple model:
  - Some pre-defined variables (parameters)
  - Some tests that are performed against parameters
  - Some operations that are performed on firmware images, using those parameters
  - Several different stages of operation
- This is encoded into a byte code using CBOR.
- Some constraints:
  - No function calls
  - No recursion
  - Only predefined variables
  - No reverse branching – (Applying the same operation multiple times TBD)

How SUIT works – Powerful Results

- Most update, secure boot operations are composed of similar components
  - Copy an image from one place to another
  - Perform a transform on an image while copying it
  - Verify an image
  - Check system parameters
  - Jump to an image
- By reorganizing a few simple operations, this can enable:
  - Multi-image updates
  - Compressed/differential updates
  - Boot from external, encrypted storage
  - A/B in-place updates

How SUIT Works – Update procedure

SUIT manifests are processed in a specific order

This order is based on several rules

- Update rules
  - All dependency manifests must be fetched and validated before any payload is fetched
  - In some applications, payloads must be fetched and validated prior to installation
- Trusted boot rules
  - All dependencies and payloads must be validated prior to loading
  - All loaded images must be validated prior to execution

How SUIT Works – Sections

Because of the rules listed and their interaction with dependencies, updates are divided into the following sections:

- Dependency Resolution
- Image Acquisition
- Image Installation
- System Validation
- Image Load
- Image Invocation

Many of these sections use common information. To accommodate this, a further section is added:

- Common

How SUIT Works – Update Interpreter Operations

Some operations enable specific use cases. Few are mandatory.

Mandatory-to-implement operations

- Set parameters
- Verify device identity
- Verify image presence (correctness) or absence
- Move (Fetch) an Image or Document
- Activate

Optional operations

- Verify component properties
- Verify system properties
- Verify 3rd-party authorisation
- Process sub-behaviours
- Process dependencies
- Invoke an Image
- Wait for an event
- Additional operations can be added easily, without complicating the parser

Reference information

Some information must exist outside of the scope of commands so that it can be accessed outside the parser

- Structure version of the manifest
- Sequence number of the manifest

Some information must exist outside of the scope of commands to simplify parser design

- Dependencies
- Components

arm Research

Updater, Executor Workflows

Process to execute an update

Updating an application consists of:

- Selecting the latest valid, supported manifest
  - Choose the manifest with the latest sequence number and a supported structure version
  - Validate its authenticity (signature)
- Perform dependency resolution, if present
  - Execute Common
  - Execute Dependency Resolution
- Perform image fetch if present
  - Execute Common
  - Execute Image Fetch
- Perform image installation
  - Execute Common
  - Execute Image Installation

Process to execute trusted boot

Updating an application consists of:

- Selecting the latest valid, supported manifest
  - Choose the manifest with the latest sequence number and a supported structure version
  - Validate its authenticity (signature)
- Perform system validation, if present
  - Execute Common
  - Execute System Validation
- Perform image loading if present
  - Execute Common
  - Execute Image Loading
- Perform image invocation
  - Execute Common
  - Execute Image Invocation

arm Research

Parameters

Parameters

Some updates require additional configuration

- Values that reflect configuration are stored in settable parameters
- Some parameters can be overridden (for example, URIs)
- Some parameters are changed based on a decision (for example, A/B images)
- ACLs control write access to parameters

Parameter list

The default set of possible parameters is:

- Image Digest
- Image Size
- Vendor ID
- Class ID
- Device ID
- URI
- Encryption Info
- Compression Info
- Unpack Info
- Source Component
- Strict Order

Parameter extensions

Additional parameters can be registered as standard parameters

Application specific parameters can be used, but intermediate systems will not be able to interpret them

arm Research

Dependency Handling

Dependency processing

Each dependency may need to be handled at each section (except common)

It is up to each manifest to ensure that its dependencies are processed correctly

Dependency processing is represented with a ”process dependency” command

When Process Dependency is invoked, it:

- Loads the identified dependency
- Executes the common for the dependency
- Executes the current named section for the dependency

arm Research

Interpreter Complexity

Update Interpreter Complexity

Sample implementation: Bootloader adapted from mbed-bootloader

https://github.com/ARMmbed/suit-manifest-generator/tree/master/parser_examples/suitloader

Support for a subset of operations

Size: 2312 (Excludes: Libc, Drivers, crypto code)

arm Research

Manifest Generator

Suit-manifest-generator

- Takes a descriptive input:
  - Structure version number
  - Sequence number of the manifest
  - List of components and their parameters
    - Vendor & Class Identity
    - Component ID for installation
    - URI for fetching the image
    - Digest & size of image – or a local copy of the file
    - Loading information for run-from-RAM
    - Encryption, compression, unpacking information
- Outputs a manifest in one of three formats:
  - CBOR
  - JSON
  - CBOR diagnostic notation

Signing

- Signing is a separate operation
- Takes a specified key and adds a signature to an existing manifest

arm Research

Examples

Boot-only input

```
{
    "components" : [
        {
            "install-id" : ["00"],
            "bootable" : true,
            "install-digest": {
                "algorithm-id": "sha256",
                "digest-bytes": "00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210"
            },
            "vendor-id" : "fa6b4a53-d5ad-5fdf-be9d-e663e4d41ffe",
            "class-id" : "1492af14-2569-5e48-bf42-9b2d51f2ab45",
            "install-size" : 34768
        }
    ],
    "manifest-version": 1,
    "manifest-sequence-number": 1
}
```

Boot-only output

```
{
    / authentication-wrapper / 2:h'81d28443a10126a0582482025820730 54c8
cc42e3e76c974ad0bed685d88b0b99df40fbaf72f58cd0b97dcd0328558400e9ddaba5
d5c4eaa58ffe22e6f22ff1adf35c80ffdc76ba07b6318c2bf73738165445372657c0d5
f7cd91841506d552089284899dbee507c5fafb5ff7c5a242c',
    / manifest / 3:h'a50101020103585aa202448181410004585086015 0fa6b4a5
3d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab4514a20b820
2582000112233445566778899aabbccddeeff0123456789abcdeffedcba98765432100
c1987d00a438203f60c438217f6'
}
```

Boot-only output – Authentication-wrapper

```
/ authentication-wrapper / 2: [
        18([
                / protected / h'a10126' / {
                    / alg / 1:-7 / ES256 /,
                } /,
                / unprotected / {
                },
                / payload / h'8202582073054c8cc42e3e76c974ad0bed685d88
b0b99df40fbaf72f58cd0b97dcd03285' / [
                    / algorithm-id / 2 / sha256 /,
                    / digest-bytes /
h'73054c8cc42e3e76c974ad0bed685d88b0b99df40fbaf72f58cd0b97dcd03285'
                ] /,
                / signature / h'0e9ddaba5d5c4eaa58ffe22e6f22ff1adf35c8
0ffdc76ba07b6318c2bf73738165445372657c0d5f7cd91841506d552089284899dbee
507c5fafb5ff7c5a242c'
            ])
    ] /,
```

Boot-only output – Authentication-wrapper

```
/ manifest / 3: <bstr> / {
        / manifest-version / 1:1,
        / manifest-sequence-number / 2:1,
        / common / 3:h'a2024481814100045850860150fa6b4a53d5ad5fdfbe9de
663e4d41ffe02501492af1425695e48bf429b2d51f2ab4514a20b82025820001122334
45566778899aabbccddeeff0123456789abcdeffedcba98765432100c1987d0',
        / validate / 10:h'8203f6' / [
            / condition-image-match / 3,F6 / nil /
        ] /,
        / run / 12:h'8217f6' / [
            / directive-run / 23,None
        ] /,
    } /,
```

Boot-only output – Authentication-wrapper

```
/ common / 3: <bstr> / {
    / components / 2:h'81814100' / [
        [h'00']
    ] /,
    / common-sequence / 4: <bstr> / [
        / condition-vendor-identifier / 1, fa6b4a53-d5ad-5fdf-be9d-e663e4d41ffe,
        / condition-class-identifier / 2, 1492af14-2569-5e48-bf42-9b2d51f2ab45,
        / directive-override-parameters / 20,{
            / image-digest / 11:[
                / algorithm-id / 2 / sha256 /,
                / digest-bytes / h'00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210'
            ],
            / image-size / 12:34768,
        }
    ] /,
} /,
```

arm Research

Modifying the Manifest Generator

Code Structure

The manifest generator is divided into three parts

- The manifest representation in manifest.py
- The manifest compiler in compile.py
- The manifest signer in sign.py

Modifying the manifest generator

- Modify manifest.py to add support for new elements
- Modify compile.py to convert information from input file to new elements

arm Research

Modifying the Example Parser

Code Structure

- Some basic CBOR operations are used:
  - Get uint64_t
  - Get int64_t
  - Get **as** uint64_t
  - Get reference
    - Get integer part as uint64_t
    - Get pointer immediately following integer part
- One advanced operation
  - Handle Pairs
    - Associates integer keys with types and handlers
    - For maps, consumes 1 key and 1 value
    - For arrays consumes even elements as keys and odd elements as values
- Code in 2 primary files
  - Suit-parser.c: generic operations
  - Suit-bootloader.c: platform and bootloader specialisations

Modifying the suit-parser-example

- Write a handler for the key, including any parsing of arguments
- Add a key/value pair to the containing structure

arm Research

Hackathon ideas

Hackathon ideas

- Validate part of the draft by implementing it
  - Implement decryption or decompression of payloads
  - Evaluate dependency or multi-component handling
  - Prototype conditional precursor images for differential updates
- Add update support to the demo application
- Evaluate SUIT in a different application
  - Delivering configuration
  - Launching applications securely
  - TEEP

arm
Research

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
תודה

arm
Research