arm

PSA Firmware Framework - M

Secure Functions

Part of the PSA FF-M v1.1 update

Andrew Thoelke, ATG

June 2020

Secure Functions

Step 2 of the roadmap to *PSA Firmware Framework - M* v1.1

- Context
- Analysis
- Proposal
- Appendix – the roadmap

Context

Today we have two programming models for developing and running security services

IPC model (PSA FF-M v1.0 and TF-M)

- Services are deployed in Secure Partitions (SP)
- Each SP is programmed like a single-threaded C program with a non-returning entry-point
- The SP thread polls for service messages and other events, and responds to them
- The SP has its own execution context and stack
- The SP determines which signal to process next

Library model (TF-M)

- Services are functions
- The functions are invoked by the framework within the secure processing environment
- Each service function handles requests from a corresponding client-side function
- The framework is in control of the execution context and sequence in which the service handlers run

The programming model applies to the whole system. TF-M is built to either run all the services in the Library model, or all services in the IPC model.

In PSA FF-M v1.1, we want to enable simple frameworks, similar to the Library model for small systems.

Analysis

- The IPC model design provides flexibility for the RoT Service developer
- However, this design places constraints on the framework implementation:
  - The framework must allocate and manage a thread stack and execution context for each SP
  - The SP may have to run in a different processor state, resulting in more context switches
  - The SP entry point and main processing loop are ‘boilerplate’ code for simple SPs
- For many RoT Services, the flexibility is not used and the necessary costs to the developer and implementation provide no benefits
- However, using the simpler Library model is often too simple to use as an alternative:
  - The choice of model affects the entire system, not just a single SP
  - It does not have built-in support for connection/session-based APIs
  - Library model does not allow a service to identify different clients
  - Library model doesn't scale easily to concurrent or isolated services
  - Library model doesn’t provide some valuable mitigations against errors in client parameter processing
- Library model uses a completely different API in the client and service to the IPC model

Proposal – Secure Function model

***Note:** this is an initial proposal, and open for review, feedback and update*

- The Secure Function model (SFN model) is an alternative SP programming model
- The SFN model looks like a hybrid between the IPC model and the Library model
  - Secure services are implemented as Secure Functions (SFN) that are called by the framework
  - SFNs are invoked by a client call to `psa_connect()`, `psa_call()` and `psa_close()`
  - SFNs are provided with a client identity
  - SFNs access client parameters indirectly using APIs
- The framework invokes the SFNs directly to process a request
  - The framework provides the execution context for the SFN
  - There is no SP entry-point and signal handling loop
- Execution within and between SPs is the same as the IPC model:
  - An SP using the SFN model is single-threaded, so SFNs within a single SP are run sequentially
  - The framework is permitted to run SFNs from different SPs concurrently
- The SFN model API is compatible with the IPC model API, not the Library model API

Proposal – SP definition details

***Note:** this is an initial proposal, and open for review, feedback and update*

- The SP manifest file must define a new attribute `model`, to be either `IPC` or `SFN`
  - If it is `SFN`, then SP is using the SFN model and the following changes apply to the SP:

- The `entry_point` attribute is replaced with an optional `entry_init` attribute
  - If present, this identifies a function that is used to initialise the SP
- The `stack_size` and `heap_size` attributes become **hints** to the framework
- There are no service signals, and the service signal names are not defined
- Each RoT service defined in the manifest has a Secure Function with the prototype:

```
psa_status_t sfn_name(const psa_msg_t* msg);
```

  - where *name* is the lowercase version of the service's `name` attribute
- IRQs and the doorbell still use SP signals
  - An SFN can use `psa_wait()` to check or block for a specific interrupt or doorbell signal

Proposal – writing SFNs

Note: *this is an initial proposal, and open for review, feedback and update*

- SFNs will still receive *connect*, *disconnect* and *request messages*, in the same way that these were delivered to an SP using the IPC model[1]
- A SFN processes the delivered message using the `psa_read()`, `psa_write()`, `psa_skip()`, and `psa_set_rhandle()` functions
- The return value from the SFN is used as the reply status for the message
- A SFN cannot use the `psa_get()` or `psa_reply()` functions, as this functionality is performed by the framework
- A SFN can use `psa_wait()` to wait for IRQ signals that are defined in the manifest, or the Secure Partition doorbell signal
- The remaining PSA-FF-M APIs work in the same way as in the IPC model

[1] Step 6 in the roadmap will provide *stateless services*, without *connection* and *disconnection messages*

Implementation – Framework impact

- Conceptually, for a single service named *SERVICE* in an SP, the framework behaves *as if* it was the following IPC model entry point:

```
void sp_main(void)
{
    psa_msg_t msg;

    for (;;)
    {
        psa_wait(SERVICE_SIGNAL, PSA_BLOCK);
        if (psa_get(SERVICE_SIGNAL, &msg) == PSA_SUCCESS)
            psa_reply(msg.handle, sfn_service(&msg));
    }
}
```

- (Note that in the SFN model, `SERVICE_SIGNAL` would not be defined)
- In practice, the framework can choose to implement this very differently. For example, by running `sfn_service()` on the SPM execution stack

Next steps

- Continue with the detailed development of the steps in the roadmap
  - Finalising the proposal for step 3. Memory mapped client parameters
  - Draft proposal for step 4+5. Interrupt handling
  - Draft proposal for step 6. Stateless services
- Compile all of the proposals into a PSA FF-M v1.1-alpha update specification

- Please provide feedback on this proposal, or the roadmap in the TF-M mailing list, or to arm.psa-feedback@arm.com

arm

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

arm

Appendix

PSA FF-M v1.1 Roadmap

PSA FF-M v1.1 Roadmap

- This is a roadmap proposal
  - We haven't worked out the details of all of the steps
  - Or even if we need them all, or if we need some others

1. Default handles (proposed)
   - Special build-time handle values that allow clients to request one-shot services without making an explicit connection. Services still receive a *connection message* for this implicit connection.
2. Secure Functions (proposed)
   - This introduces the SFN model as a per-SP option. Services are functions called by the framework, and use the IPC model APIs to read and write request parameters
3. ~~Direct client memory access~~ Memory mapped client parameters (draft)
   - This optional API introduces the ability for a service to directly read and write the client parameter memory. This will not work on all implementations, but is necessary for efficiency in simple systems.

Roadmap – continued

4. First Level Interrupt Handling
   - This adds a deprivileged, low-latency, interrupt handling capability to SPs that are using the IPC model. FLIH functions cannot use normal SP APIs, but can signal the SP for later in-thread processing.
5. Second Level Interrupt Handling
   - This adds a non-concurrent interrupt handling capability to SPs that are using the SFN model. An SLIH functions can run if no Secure Function is running in the SP.
6. Stateless services
   - This attribute indicates that a service does not maintain any per-connection state. The framework will not deliver *connection* or *disconnection messages*, and connections are automatically accepted.
7. Miscellaneous
   - Ensure alignment of functionality between SFN model and IPC model.