PSA Firmware Framework - M

Roadmap to v1.1

Introducing a Secure Function model

Andrew Thoelke, ATG
May 2020

Roadmap to *PSA Firmware Framework - M* v1.1

Content

- Purpose
- Context
- Analysis
- Proposal
- Roadmap
- Discussion

Purpose

- The original scope for v1.1 of the PSA Firmware Framework – M was (approximately):
  - Important enhancements to the existing “IPC model” framework.
  - Include an architectural definition for the TF-M “library model” framework.
- Technical challenge:
  - TF-M library model looks like an entirely different architecture and API to PSA-FF-M v1.0.
- Working with the TF-M team, we think that we have a way to [mostly] address both objectives without splitting the architecture
  - The aim is to have a architecture (and implementations) that can scale better
- This roadmap provides the context for each of the steps that we propose along the way
- NOTE: this roadmap is only an outline, we haven’t worked out details for all the steps

Context

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

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

- Services are deployed in Secure Partitions (SP)
- Each SP is programmed like a C program
- The SP thread polls for service messages and other events, and responds to them
- The communication API presents session-based connections to secure services
- Clients make structured, synchronous requests
- The framework provides a secure client identity
  - Enables delegated resource ownership
- Services use the same API to connect and make requests of other secure services

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
- Each request is a singleton (no connections)
- Clients make structured, synchronous requests
- There is only one client (the non-secure domain)
- Services use direct function calls to make requests of other services.

Analysis – two architectures

- The IPC model is good for flexible and complex systems:
  - Service developers manage execution within each Secure Partition
  - The one-thread-per-partition execution model is easy to analyse when integrating multiple SPs
  - The API design requires that request data is copied between the client and service, mitigating common service implementation vulnerabilities
- The Library model is good for simple systems:
  - Easy to describe, and leads to simple implementations for systems implementing level 1 isolation
  - Service functions must complete execution before another can start
  - Direct access to client memory is assumed in the API, reducing the overhead of copying data
- BUT: system and product requirements are not binary
  - There is a spectrum of system complexity and product security needs
  - For systems that fall in between these two points, which framework design should be used?

Analysis – scaling and flexibility

Real systems often lie in between ‘full’ IPC model and library model

IPC model does not scale down efficiently

- Simple one-shot secure operations require a connection
- Simple services require boilerplate code in the SP to handle signals and dispatch requests to their respective service handlers
- The framework has to manage an execution context for each SP, and switch between them to process requests

Library model does not scale up safely

- Adding more isolation domains (level 2+) breaks the simplifying assumptions
  - Services must be isolated from dispatcher
  - Client identity is required
  - Inter-service calls must go through framework
  - Services need a different execution stack
- Concurrent service execution requires additional execution contexts and synchronization for use of shared-data
- Direct client memory access requires that **every** service needs review to mitigate errors

Proposal

- There are already ideas that tackle some of the challenges:
  - Evolution of the Library model API for service functions, removing the client memory addresses and requiring the use of framework APIs to read and write parameter data
  - The Default Handles proposal (TF-M Forum 30th April) to optimize the client for one-shot services
  - The *Multi-threaded single-scheduler model proposal* discussed on the mailing list (here and here) and in the TF-M Forum on 2nd April.

- These all make more sense if viewed as part of a larger roadmap that aims to address the main challenges
- The roadmap proposed here:
  - Introduces changes that together provide an API for implementing a framework that has the simplicity of the Library model, but which is part of the same overall architecture as the IPC model
  - Adds options for service developers that provide the ability to simplify the implementation of both client and service code, which are all useful within the IPC model
  - Aims to unify the approach to interrupt handling between the programming models

Proposal – Secure Function model

- The *Secure Function model* (SFN model) is alternative programming model, for code within a Secure Partition
- 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 invoked by the framework
  - Secure Functions are invoked by a client call to psa_call()
  - Secure Functions are provided with a client identity, to enable separation of per-client resources
  - Secure Functions access client parameters indirectly, using APIs to read and write the parameter data
- The SFN model API is not compatible with the Library model API
- If the system is simple enough the framework implementation can be optimized
  - It might look very much like the TF-M library model design
- The SFN model permits multiple SPs, and higher levels of isolation
  - But these require a more complex framework implementation

Roadmap

- At the stage, 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
   - This introduces the SFN model as a per-SP option. Services are functions called by the framework, and use the IPC model APIs (or something very similar) to read and write request parameters
3. Direct client memory access
   - 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.

Discussion items

- The items on the roadmap have many open issues that will require further discussion
  - Expect that this will happen as part of defining the details for each step

- Default handles are not a universal replacement for SIDs
  - Limited resource, and an integrator’s challenge (but great for small systems and important services)
- How important is Direct client memory access?
- Use cases for the interrupt handling in Secure Partitions
  - Do we need support for mixed models such as FLIH + signal/wait in SFN model partitions?
- Should the SFN model API be the same as the v1.0 IPC model API?
  - Using psa_msg_t objects, and message handles for psa_read() etc.
  - Or a similar API that permits implementation optimization such as the removal of unused message fields, or the simplification of message references.

- Does this roadmap fail to address any important use cases?

Next steps

- Continue with the detailed development of the steps in the roadmap
  - Detail of the architecture changes for step 1. Default handles
  - Full write-up of step 2. Secure Functions

- Please provide feedback on this 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