Armv8-M Security Extensions Threat Model

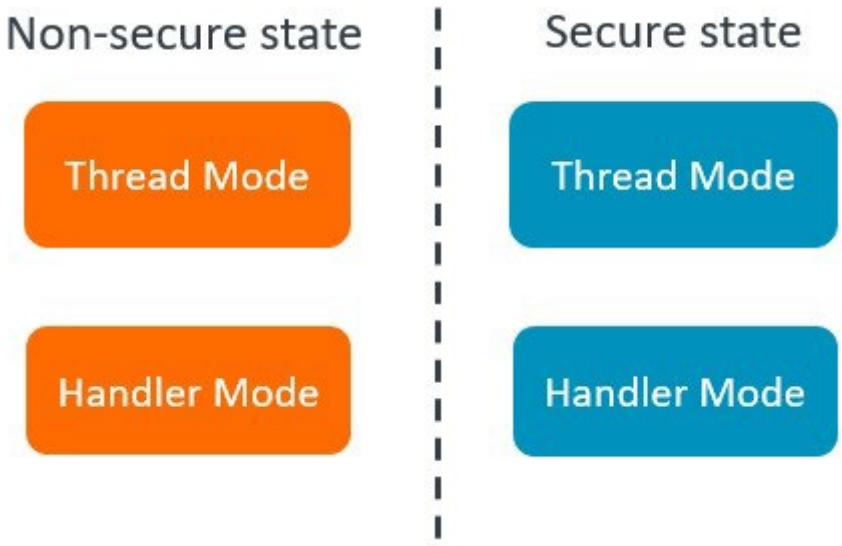
Uma Maheswari Ramalingam, Principal Engineer, Arm
Thomas Grocutt, Lead Architect and Fellow, Arm
Francois Botman, Senior Principal Engineer, Arm

This white paper describes the Armv8-M Security Extension Threat model. It includes:

- Quick overview
- Use cases
- Assumptions and Requirements
- Adversarial model
- Threat model

Quick Overview

The Armv8-M architecture introduced the Security Extension provides higher security assurances. It divides the system into two security states which are orthogonal to the existing thread and handler mode separation. The following figure shows the four possible states:



Various architectural features are provided that prevent code executing in the Non-secure state from being able to access resources (including memory, register values, interrupt priorities, etc.,) belonging to Secure states. However, the Secure state has access to Non-secure resources. You can directly call functions in a different security state and function arguments can be passed in registers. These inter domain function calls provide an efficient method of transitioning between states.

Similarly, exceptions can cause direct transitions to the appropriate destination handler mode. Because the hardware handles many of the security operations required (e.g. protecting the register state when an exception is taken from secure background code to a non-secure exception handler) the integrity of the system is not compromised by enabling direction transition between security states.

With an Armv8-M based processor with security extensions, you must consider other system components that can generate memory transactions such as DMA, Display controller etc. If the component can generate Secure transactions, then ensure that this does not provide

a mechanism for Non-secure software to control transactions in Secure memory space. For example, Secure memory must not be accessed by a non-secure DMA controller.

Use Cases

Microcontrollers have a very diverse set of use cases, ranging from simple bare-metal systems to complex RTOS-based environments. As such, there are many ways that you can use the extra states provided by the Armv8-M Security extensions. Some of the possible use cases are:

- Monolithic stacks and bug isolation:
    - Only privileged thread mode is used in both Secure and Non-secure states.
    - Vendor-specific application code can be run in Non-secure state.
    - Certified stack (for example, Bluetooth) can be run in Secure state.
    - Bug isolation: if Vendor A supplies a microcontroller with some built-in libraries to vendor B and Vendor B adds additional software specific to the end application on top of it. In such situations, if vendor A's code is isolated in the secure domain, bugs in vendor B's code (e.g. writing beyond the end of an array) in Non-secure state cannot corrupt data associated with vendor A's libraries. This can prevent bugs inadvertently manifesting as symptoms in other pieces of software, and reduce the support burden to vendor A.
- Safety-critical systems:
    - Being able to protect a section of software from accidental corruption is very useful in safety critical systems such as automotive and medical devices. In these use cases you can isolate the core safety critical code in Secure state from the bulk of the other code (for example: GUI code) executing in Non-secure state. In this way code issues in Non-secure state do not pose a safety risk to the end user. This way also provides isolation in the chain of liability.
- Platform Security Architecture - Firmware framework:
    - Secure services that manage critical features like remote firmware update, security identity, crypto services etc.,
- Code protection and software confidentiality
    - You can run multiple secure unprivileged software libraries from several mutually distrustful vendors in Secure state within the same processor.
    - A secure privileged library manager can use secure MPU to make separation across different secure libraries.
    - You can use function calls for both Non-secure inter library calls and Secure inter library calls.

Assumptions

The threat model for the Armv8-M Security Extensions assumes that the following threats have been mitigated by other parts of the system by software:

- The asset under attack (see A1 to A4 in the assets section below) does not contain any security vulnerabilities, including but not limited to:
    - Buffer overflows
    - Use after free bugs
    - Secret data being revealed in address access patterns, leading to cache or other microarchitecture side channels
    - Secret data being revealed by changes in the execution time of the asset
    - Missing speculation barriers
    - Not properly sealing the secure stacks
- System and memory-level attacks have been mitigated, for example RowHammer, CLKSCREW
- Debug permissions have been restricted appropriately for the asset being protected.

Requirements

The end user system based on Armv8-M Security extension contains correct configuration settings given as per *Armv8-M Security Extension User Guide*

Assets

Based on the use cases described above, you must determine which threats apply to Secure state and investigate whether it is feasible to provide mitigations against those threats. To enumerate the threats, determine which Assets that may be handled by the system, and which Adversaries can attempt to compromise them. Any system that implements Secure state can have the following Assets:

Asset name	Description
A1_System_Code_Data	Secure privileged system code and data (excluding floating-point data)
A2_System_FP_Data	Secure privileged system floating-point data
A3_App_Code_Data	Secure unprivileged application code and data (excluding floating-point data)
A4_App_FP_Data	Secure unprivileged application floating-point data

Adversarial model

Adversarial models describe the adversaries that the Armv8-M Architecture Security Extension is intended to defend against, grouped into classes. The models are derived from the market requirements and then mapped to one or more of the adversarial models (AM) described below:

AM 0. The adversary is only capable of accessing data that does not require either physical access to a system containing an implementation of the architecture, nor the ability to run software on it. This adversary intercepts or provides data or requests to the target system through a network or other remote connection.

For instance, the adversary can:

- Read any input and output to the target through external devices
- Provide, forge, replay, or modify such inputs and outputs
- Perform timings on the observable operations done by the target machine, either for normal operation or as a response to arranged inputs (such as timing attacks on web servers)

AM 1. The adversary can mount attacks from software running in the security state and privilege level specified below on a target device implementing the feature.

- AM 1.1: Attempted exploitation of asset by an attacker running software in the Non-secure state.
- AM 1.2: Attempted exploitation of asset by an attacker running software in the Non-secure state and the Secure unprivileged state.

AM 2. The adversary can mount passive hardware attacks that do not require physical breaching of the chips. The types of physical access are:

AM 2.1 : Access to Debug Port (via DAP)

AM 2.2 : This type of adversary, as well as the following ones, has access to a system. In particular such adversarial models include access to signals, communication buses or ports and the memory bus outside the SoC, for instance through insertion of malicious hardware including chip interposers. The malicious hardware serves to eavesdrop transactions, communications, signals. AM2.2 also includes other simple, reversible modifications to the system. The following are some examples:

i. Side channel analysis that requires measurement devices (including any leakage source such as electromagnetic emissions, power consumption, photonics emission, acoustic channels, and others),
ii. Plugging malicious hardware in the system without its modification,
iii. Gaining access to the internals of the target system and interposing the SoC or external memory for the purposes of reading, blocking, replaying, and injecting transactions, and
iv. Replacing or adding chips on the motherboard.

AM 3. The adversary can mount active hardware attacks and fault injection attacks that do not require physical breaching of the chips. The adversary has the same type of access as in model AM 2, where the malicious hardware serves not only to eavesdrop transactions, communications, signals but also to block, inject, corrupt, or replay them.

The modifications to the system described in model AM 2 apply to this AM as well, but they may be irreversible.

AM 4. The adversary performs invasive SoC attacks.

Threats that can be mounted by this type include, for instance:

- Chip decapsulation through laser or chemical etching, followed by microphotography to reverse engineer the chip.
- Use a focused ion beam microscope to perform gate-level modification.

Threat Model

The threat model is constructed with the principles of Confidentiality, Integrity, and Availability of the information.
The following table contains the list of threats applicable considering assets and adversary models mentioned in above sections.

Adversary Model/Asset	A1_System_Code_Data	A2_System_FP_Data	A3_App_Code_Data	A4_App_FP_Data
AM 0	Confidentiality, Integrity, and Availability*	Confidentiality, Integrity, and Availability* If FPCCR.TS is set to one, otherwise out of scope	Confidentiality, Integrity, and Availability*	Confidentiality, Integrity, and Availability* If FPCCR.TS is set to one, otherwise out of scope
AM 1.1	Confidentiality, Integrity, and Availability*	Confidentiality, Integrity, and Availability* If FPCCR.TS is set to one, otherwise out of scope	Confidentiality, and Integrity	Confidentiality, and Integrity If FPCCR.TS is set to one, otherwise out of scope
AM 1.2	Confidentiality, Integrity, and Availability* from Armv8.1-M onwards, otherwise out of scope	Confidentiality, Integrity, and Availability* from Armv8.1-M onwards and if FPCCR.TS is set, otherwise out of scope	N/A	N/A
AM 2.1	Confidentiality, Integrity	Confidentiality, and Integrity If FPCCR.TS is set to one, otherwise out of scope	Confidentiality, Integrity	Confidentiality, and Integrity If FPCCR.TS is set to

				one, otherwise out of scope
AM 2.2	Out of Scope	Out of Scope	Out of Scope	Out of Scope
AM 3	Out of Scope	Out of Scope	Out of Scope	Out of Scope
AM 4	Out of Scope	Out of Scope	Out of Scope	Out of Scope

* Depending on the software architecture, ensuring the availability of system code/data may require boosting the execution priority of system assets to be protected.

Tip: Implementations of SecureCore CPUs such as SC300 or Cortex-M35p can protect against additional adversary models not covered by Armv8-M Security Extensions.

Conclusions

The additional states and state transition mechanisms provided by the Armv8-M Security Extensions enable a wide variety of different use case mappings allowing complex systems to be built with a high degree of protection. The overall security of a platform is only as high as the weakest component in the system. As a result ,even with a very secure processor architecture the security measures taken in the software have a significant effect on the overall security. For example, some API styles are inherently insecure (e.g. strcpy() is vulnerable to buffer overflow attacks). Therefore, a vendor wishing to secure a firmware stack must look at every aspect of the system, including the firmware stack itself. Because this process can be very costly and have a significant impact on the time to market, no one level of security fits all use-cases. Instead, the level of security required must be balanced against the cost of a security breach. This is especially true of embedded environments where there is a wide range of use cases, and cost and time to market are often critical factors. The ability of the Armv8-M Security Extensions to divide up a software stack into different security compartments means that the level of security analysis can be tailored to different levels of criticality required for the various parts of the software stack.