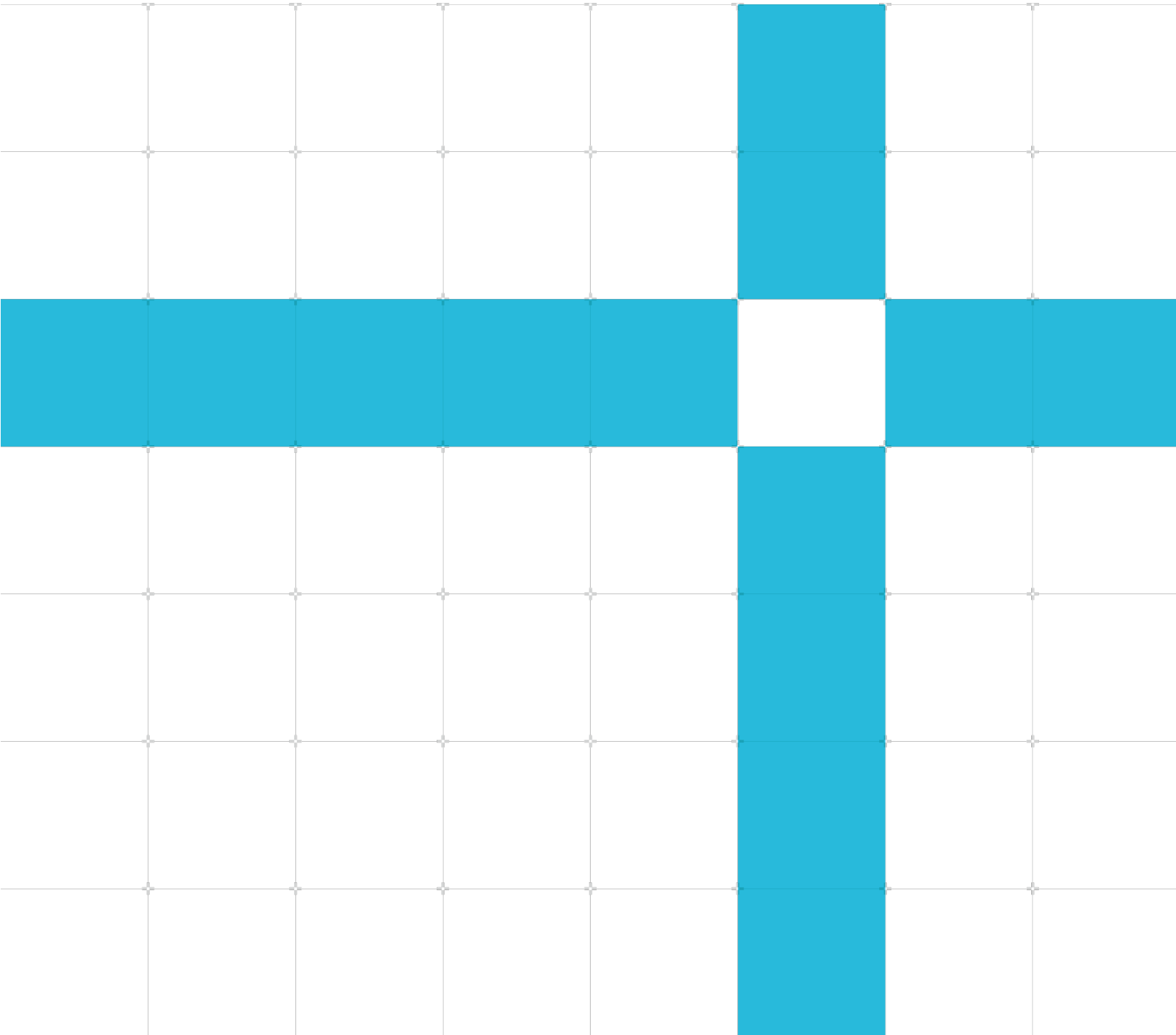
arm

Armv8-M Architecture

Stack sealing and why it is needed in TrustZone for Armv8-M

Non-Confidential

Issue 1.0
Document ID: 102446



Armv8-M Architecture

Stack sealing and why it is needed in TrustZone for Armv8-M

Release information

Document history

Issue	Date	Confidentiality	Change
1.0	28th April 2021	Non-Confidential	First release

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Web Address

www.arm.com

Progressive terminology commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used terms that can be offensive. Arm strives to lead the industry and create change.

Contents

1 Introduction

1.1 Abstract

This document explains what stack-sealing is - an operation that is applicable to Secure firmware running on Armv8-M processors with TrustZone Security Extension enabled. It also covers how it helps dealing with security threats from untrusted software.

1.2 Intended audience

This document is written for software developers writing Secure software for Armv8-M processors.

1.3 Conventions

The following subsections describe conventions used in Arm documents.

1.3.1 Glossary

The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm Glossary for more information: **https://developer.arm.com/glossary**.

1.4 Additional reading

This document contains information that is specific to this product. See the following documents for other relevant information:

Table 1-1 Arm publications

Document name	Document ID	Licensee only
Armv8-M Architecture Reference Manual	DDI0553	No
Secure software guidelines for ARMv8-M	100720	No
Arm Security Advisory Notice: Armv8-M Secure Stack Sealing	NA	No

2 Stack sealing overview

Stack sealing is a software method that ensures that a fault exception is always triggered if an underflow attack happens in a Secure stack. Stack sealing is applicable to Armv8.0-M and Armv8.1-M-based processor systems using the TrustZone Security Extension. Stack sealing is only needed for Secure software. There is no need to apply stack sealing in software that runs on the Non-secure side.

If an Armv8-M based system does not use TrustZone security technology, then stack sealing is not needed. Stack Sealing is also not required for Cortex-A processor systems.

Normally, TrustZone protects Secure stacks in the following ways:

- Non-secure software cannot access the Secure stack pointers and Secure stack limit registers.
- Non-secure software cannot access Secure stack memories, if the Secure stacks are correctly placed in Secure memories.
- Non-secure software cannot trigger an execution of Secure stack push/pop instructions.

However, there are two cases where a Non-secure software operation can trigger Secure stack operations:

- When a Non-secure interrupt handler is completed and returns to the Secure world (that is, exception return)
- When a Non-secure API that is called by Secure software is completed and returns to the Secure world, that is, a function return.

When a stack has not been automatically protected by a hardware-generated stack frame, for example an exception stack frame, or a function return stack frame, a malicious Non-secure software agent can use an invalid exception return, or an invalid function return, to trigger a stack underflow attack. This might occur, for example, when a stack is empty. The stack sealing operation ensures that this type of attack is detected.

3 Normal operations of Non-secure function call and interrupt handling

Before we describe the detail of stack sealing operation, let's look at an overview of the stack operations that are involved during security state transitions:

- In normal Non-secure function calls, and
- In normal Non-secure interrupt handling during Secure code execution

When an Armv8-M processor executes Secure software, and if the Secure software calls a Non-secure function using the `BLXNS` instruction, then two words of data are pushed to the currently selected Secure stack. The data that is pushed to the stack includes:

- The return address
- The Interrupt Program Status Register (IPSR), which is part of the Program Status Register (PSR).

Before starting the execution of the Non-secure function which is called, the processor will also change the Link Register (LR) to a Function Return value (FNC_RETURN), which has a value of `0xFEFFFFFF` or `0xFEFFFFFE`. This operation masks the real Secure program address that called the Non-secure API. If the processor is in Handler mode, the value of the IPSR is switched to a value of 1 for masking Secure interrupt information. 3.1 shows the stack operations involved when Secure software calls a Non-secure function.

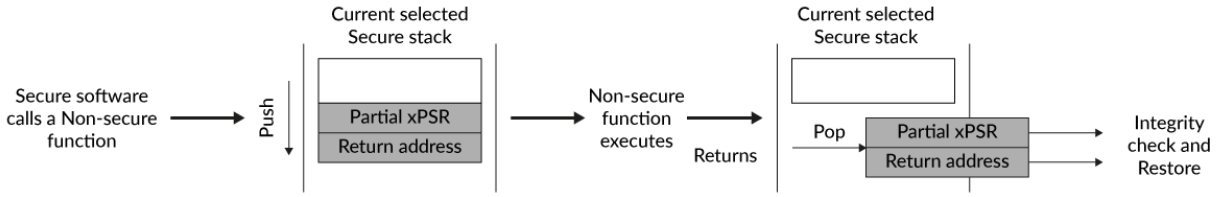


Figure 3.1: Stack operations when Secure software calls a Non-secure function

When the Non-secure function is completed, the FNC_RETURN value that was previously placed into LR by the processor hardware is loaded into PC using a normal function return instruction, for example, `BX LR`. This triggers the function return operation, which reads the function return stack from the currently selected Secure stack. The function return operation:

- Carries out an integrity check of the IPSR. This ensures that the processor mode is consistence with the stacked partial PSR.
- Restores the return address to the Program Counter and the partial PSR into IPSR.

The Secure software can then continue operations.

Similarly, when a Non-secure interrupt occurred during Secure software execution, the contents of the register bank are pushed to the currently selected Secure stack as an exception stack frame. The last word of this stack frame has an integrity signature, `0xFEFA125A`/ `0xFEFA125B`, which will be used for integrity check later. Figure 3.2 shows the stack operations involved when a Non-secure exception is trigger during the execution of Secure software.

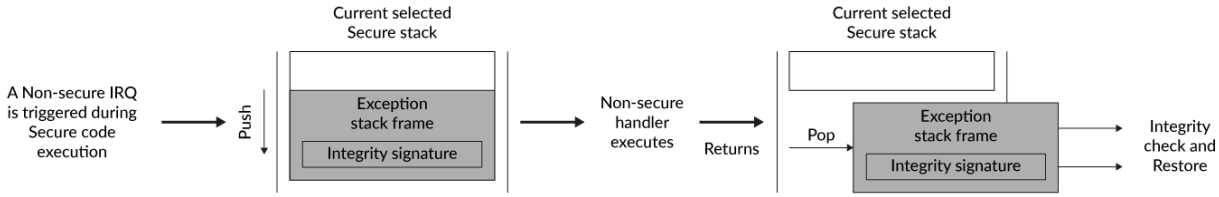


Figure 3.2: Stack operations when a Non-secure interrupt occurred during the execution of Secure software

Before switching from Secure state to Non-secure state to execute the Non-secure interrupt handler, the processor erases the contents of the register bank to prevent Secure information leakage. In addition, the Link Register (LR) is switched to an Exception Return (EXC_RETURN) value, which is `0xFFFFFFxx`.

At the end of the interrupt service routine, the EXC_RETURN value is loaded into the Program Counter using a normal return instruction. This load operation triggers an exception return operation, which. includes an integrity check of the integrity signature in the stack frame, and then restoring the contents of the registers so that the interrupted Secure program can resume.

4 Illegal return operations

4.1 Simple examples

Because the value in the Link Register can be modified by the Non-secure software during the execution of the Non-secure function or the Non-secure interrupt handler, the Non-secure software can trigger illegal return operations. Here are two examples:

- Example 1: The processor switched from Secure to Non-secure using a Non-secure function call, but the Non-secure software attempt to return to the Secure world using an Exception Return (instead of a Function Return).
- Example 2: The processor switched from Secure to Non-secure due to a Non-secure interrupt, but the Non-secure software attempt to return to the Secure world using a Function Return (instead of an Exception Return).

Figure 4.1 shows the sequences in examples 1 and 2.

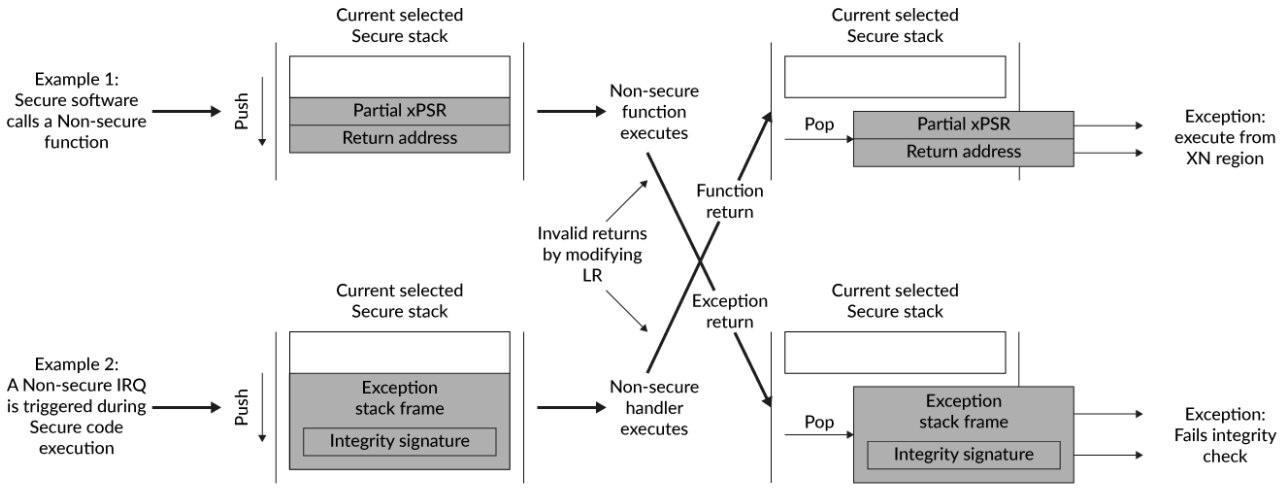


Figure 4.1: Simple illegal return operations where EXC_RETURN and FNC_RETURN values are interchanged.

In the first example, during the exception return operation the stacked return address in the function return stack frame is compared against the integrity signature value. This integrity check will fail and trigger an exception.

In the second example, the Secure software is interrupted by a Non-secure interrupt, and the Non-secure interrupt service routine carries out a return using FNC_RETURN. This activity will also trigger an exception because the value of the integrity signature is not an executable address. Based on the architecture definition, the address with the same value as FNC_RETURN always has the eXecute Never (XN) attribute.

In both examples, a fault exception in the Secure world is triggered. This is a good outcome, because invalid returns can be detected, and the Secure software can do something about it.

4.2 Complex examples

The situation gets much more complex when we consider that there are two Secure stack pointers in an Armv8-M processor: The Secure Main stack pointer (MSP_S), and the Secure Process stack pointer (PSP_S). The selection of the stack pointer is based on two factors as follows (Figure 4.2):

1. The current processor mode (either Thread or Handler) – If the processor is in Secure Handler mode, MSP_S is selected.
2. The current value of CONTROL_S.SPSEL – If the processor is in Thread mode and CONTROL_S.SPSEL is 0, then MSP_S is selected. If the processor is in Thread mode and CONTROL_S.SPSEL is 1, then PSP_S is selected.

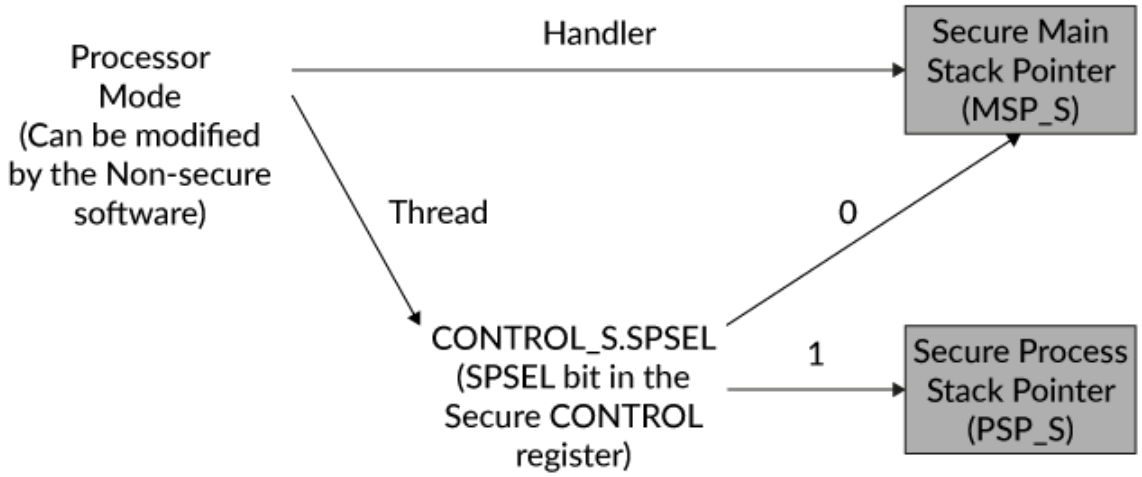


Figure 4.2: Secure stack pointer selection.

Although the value of CONTROL_S.SPSEL is accessible only by Secure privileged software, the processor mode can be changed by the Non-secure software during the execution of the Non-secure function or Non-secure interrupt handler. Therefore, the Non-secure software can trigger a return to the Secure world using an incorrect Secure stack pointer. In addition, as explained in section 4.1, the EXC_RETURN or FNC_RETURN value in the LR can also be modified by the Non-secure software. As a result, it is possible for the Non-secure software to force the processor to return to Secure state via an invalid execution path which use an incorrect stack pointer.

When an incorrect stack pointer is used, the memory location that the stack pointer points to might contain unpredictable, unknown, or other data. For an invalid function return that uses the wrong stack pointer, the unpredictable/unknown/other data could be used as a return address. This type of illegal return might end up with an exception because:

- The value that is used as return address might represent an address that is either illegal, non-executable, or the address is not in Secure address range.
- The integrity check for partial PSR might fail.
- The content of the memory that the incorrect return address points to does not have valid program code, resulting in other fault exception events.

However, there is still a chance that the invalid return is not detected immediately.

If an invalid exception return is carried out using an incorrect Secure stack pointer, the exception return operation is likely to trigger a fault exception. This is because the memory that the stack pointer points to is unlikely to have an integrity signature. But there is still a small chance that a data value matching an integrity signature is present in the memory. This means that a fault exception is not triggered immediately.

4.3 How stack sealing can help

To ensure that the illegal returns mentioned in section 4.2 trigger fault exception immediately, the alternate Secure stack (the Secure stack that is not selected when switching from Secure state to Non-secure state) should be sealed using stack sealing data. In the case of an illegal function return using an incorrect stack pointer, the stack sealing value is used as return address. This value points to a non-executable address location and triggers a fault exception immediately. The sequence is illustrated in Figure 4.3.

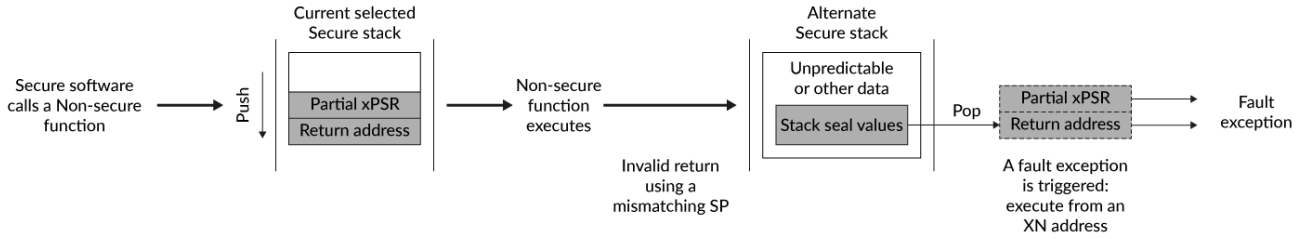


Figure 4.3:Illegal function return always trigger faults.

In the case of an invalid exception return using an incorrect stack pointer, the stack sealing value is compared against the integrity signature. This comparison will always mismatch and trigger a fault exception. The sequence is illustrated in Figure 4.4.

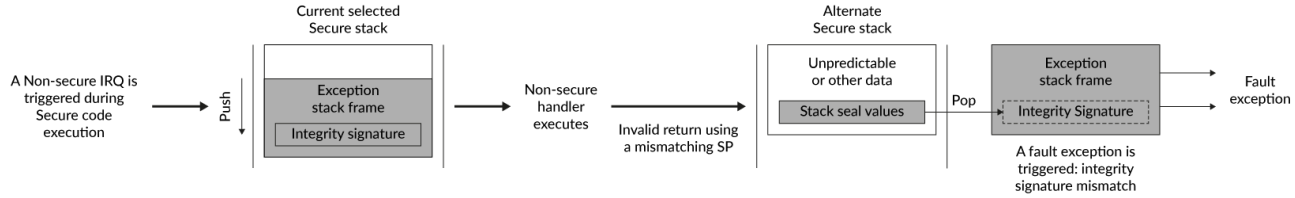


Figure 4.4:Illegal exception return always trigger faults.

4.4 Stack sealing value

The recommended value for a stack sealing operation is specified in the Armv8-M Architecture Reference Manual as `0xFEF5EDA5`. This value is an eXecute Never (XN) address, and also does not match the integrity signature.

To seal a stack, we need to push two words into the stack. Although we only need one word to detect an invalid return, we must push two words to keep the stack doubleword-aligned. Double word stack alignment is a requirement of the Procedure Call Standard for the Arm Architecture (AAPCS).

The stack sealing operation is shown in Figure 4.5.

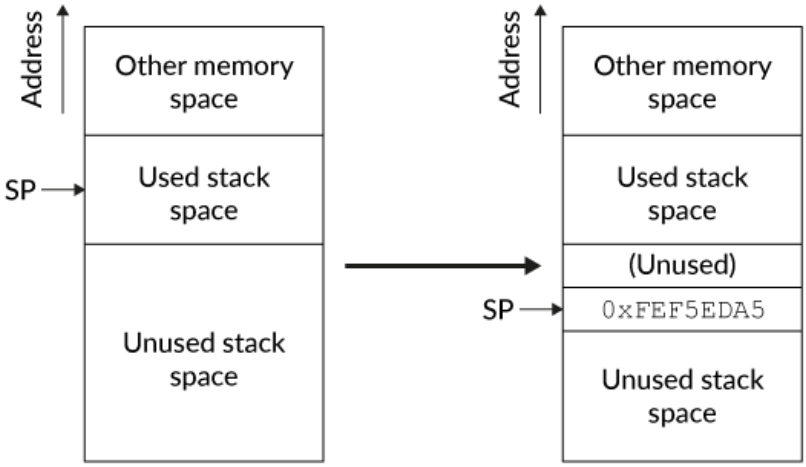


Figure 4.5:Sealing a stack

The Secure stack that is not currently selected must be sealed so that a return using the wrong stack must trigger a fault exception. For example:

- When Secure privileged software creates a new Process Stack for a new Secure thread.
- We also must seal the stacks when deprivileging a Secure interrupt handler. This process usually involves creating a new Process Stack for the unprivileged handler code, which must be sealed. In addition, when switching from the Secure privileged state to Secure unprivileged state, the common practice is to trigger Secure SVC exception, which then create a fake exception return stack in the Secure process stack so that the processor can return from an SVC handler to thread.

  When dealing with this process, the top of the fake stack frame for the return process should contain the stack seal value. In addition, the Secure main stack, which contains an exception stack frame generated from the SVC exception entry, must also be sealed before the exception return is carried out. This is because this stack frame does not have an integrity signature.
- The Secure stack or stacks must also be sealed before the initial switch to the Non-secure world.
  Note:
  - If a `BLXNS` instruction is used for the switching, then the current stack does not need sealing because the stack would have a function return stack.
  - If the SPSEL bit in the Secure CONTROL register is being kept as 0, then the Secure Process Stack do not need to be seal as it will never be used.

5 Examples

In this section of the paper, we provide a few examples of attacks and how stack sealing detects the attack is explained.

5.1 Example 1 – Underflow attack to the Secure Main Stack

You can see the first example in Figure 5.1:

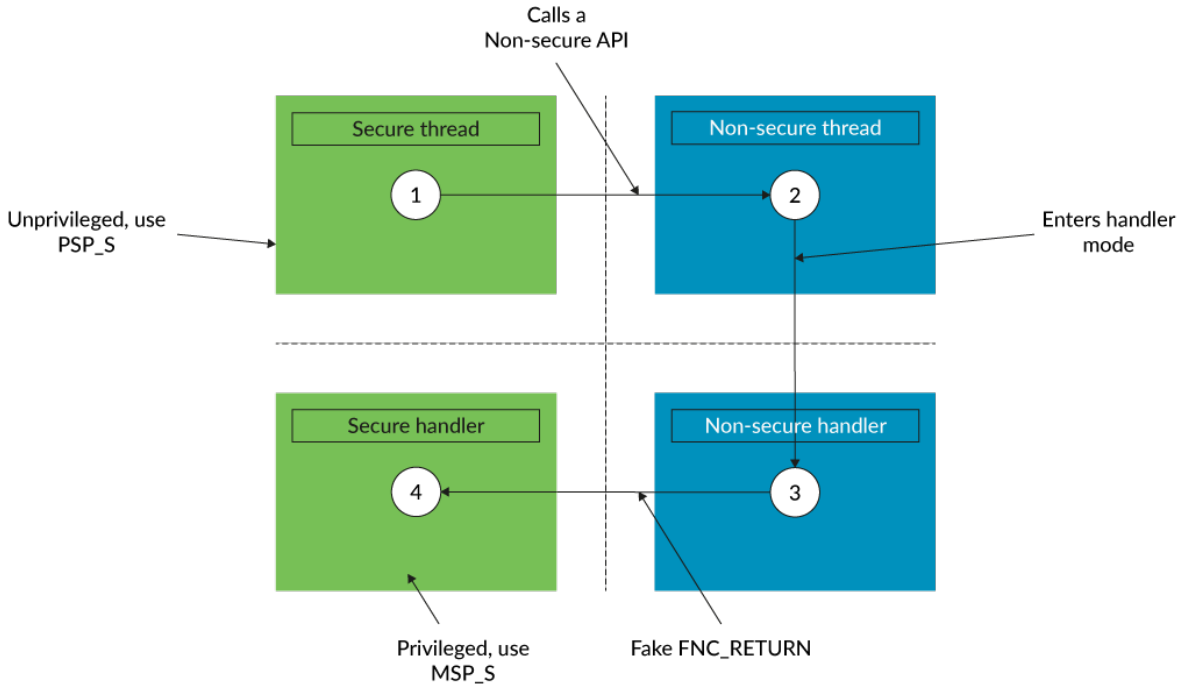


Figure 5.1: Attack example 1: Non-secure software uses a fake function return to attack the Secure main stack

Here are the steps that occur this attack example:

1. The Secure unprivileged software that runs on the processor calls a Non-secure API.
2. Non-secure code triggers a Non-secure exception
3. The Non-secure handler uses a fake FNC_RETURN to return to the Secure world.
4. Secure software executes code from an unintended address location.

Because the processor is now in handler mode, the Secure Main stack pointer is used for the function return, which might be empty or contain other data.

If the Secure Main Stack is sealed, the stack sealing value is interpreted as a return address. This interpretation triggers a fault exception because it attempts to execute from a non-executable region.

5.2 Example 2 – Underflow attack to the Secure process stack

In the second example, which you can see in Figure 5.2, the program flow goes the other way round:

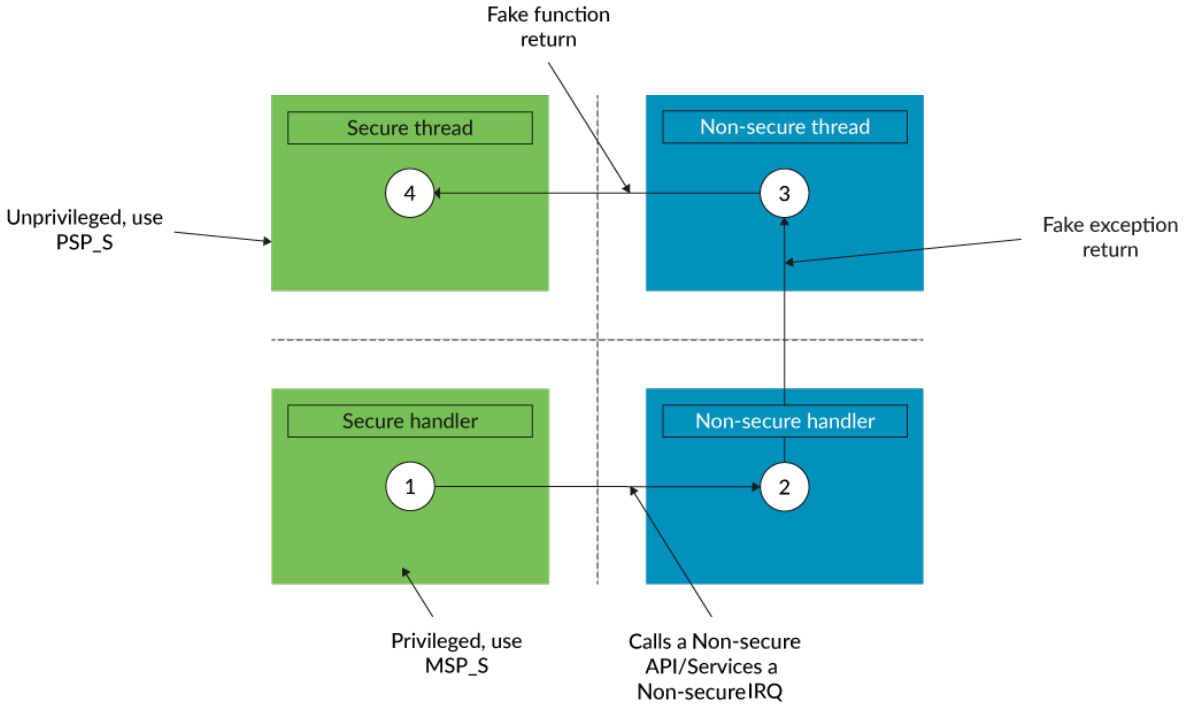


Figure 5.2: Attack example 2: Non-secure software uses a fake function return to attack the Secure process stack

Here are the steps for this attack example:

1. The Secure handler code, which uses the Secure Main Stack, switches to Non-secure handler mode because the code called a Non-secure function, or because the code is interrupted by a Non-secure exception.
2. Within the Non-secure code, a fake exception return is used to switch the processor into Non-secure thread mode.
3. A fake function return is used to switch to the Secure world.
4. Secure software executes code from an unintended address location.

If the SPSEL bit in the Secure CONTROL register is 1 (that is, CONTROL_S.SPSEL), the Secure Process Stack is used for the return operation. If the Secure process stack was empty, the memory contents at memory[PSP_S] and memory[PSP_S+4] could be UNPREDICTABLE, or contain data from other part of Secure software.

At step 4, the processor treats memory[PSP_S] like the return address, and memory[PSP_S+4] like the stacked partial PSR. If the lowest 9 bits of the value in memory[PSP_S+4] are zero, the integrity

check for IPSR passes. If the value in memory[PSP_S] points to a valid Secure address, the processor could jump into that address.

If the Process Stack is sealed, then the stack seal value is used as a return address and triggers a fault exception because it points to a non-executable address. To eliminate timing window where it is possible to launch this attack, the Process Stack should be sealed before the PSP_S is set to its address and CONTROL_S.SPSEL is set to 1.

5.3 Example 3 – Deprivileging a Secure interrupt handler

The third attack example, which is shown in Figure 5.3, has the following sequent of events:

1. A low priority Secure exception is triggered. It does not matter whether the processor is in Secure or Non-secure world in this step.
2. The processor starts the execution of the Secure exception handler.
3. This Secure interrupt handler is deprivileged so that some parts of the handler can be executed in an unprivileged level. To execute a part of the interrupt service in an unprivileged level, the Secure interrupt handler in step 2 triggers an SVC handler (step 3), which creates a fake exception stack frame, which is then used for an exception return that switch the processor into Secure unprivileged thread in step 4.
4. The processor executes a part of the Secure handler in Secure thread mode.
5. The unprivileged Secure handler calls a Non-secure function, which is then interrupted by a high priority Non-secure exception, or
6. The Secure unprivileged handler is interrupted by a high priority Non-secure exception. With either path, the processor reached Non-secure Handler mode. This Non-secure handler uses a fake exception return to switch the processor to Secure state. Because the processor is in the handler mode, the return to Secure world used the Secure MSP.
7. Secure software executes code from an unintended address location.

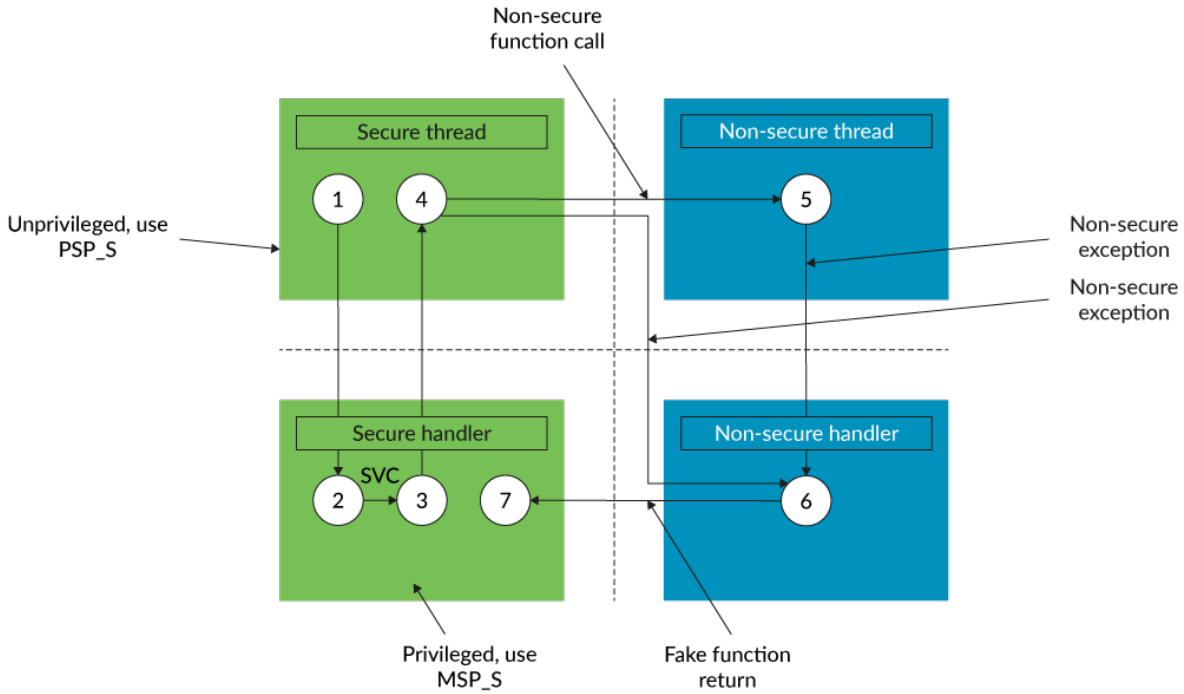


Figure 5.3: Attack example 3: Attack to the Secure Main Stack following a handler deprivileging operation

To ensure that the attack showing in Figure 5.3 is detected, the Secure main stack need to be sealed before the exception return in the Secure SVC handler. By doing so, the attack showing in Figure 5.3 triggers a fault exception. This is because the value of the stack seal value points to a non-executable address.

When Secure handler deprivileged itself to execute a part of the handler with unprivileged state, there is an additional attack possibility: After the exception return from Secure SVC to Secure thread in Step 4, the Secure process stack for the unprivileged handler code could be empty. If a high priority Non-secure exception take place, and the Non-secure exception handler uses a fake FNC_RETURN to return to the Secure world, an underflow could take place on the Secure main stack.

To minimize the security risk, the fake exception stack frame created for switching from step 3 to step 4 needs to contain the stack sealing value at the top of the fake stack frame (Figure 5.4). Because CONTROL_S.SPSEL is zero during the SVC handler execution, the bottom of the fake stack frame is not required to be sealed.

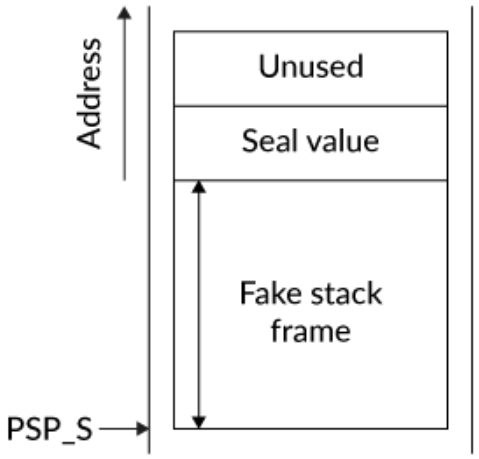


Figure 5.4: Recommended use of the stack sealing value on top of the fake stack frame for the Secure SVC handler to return to Secure thread.

5.4 Example 4 – Corrupting EXC_RETURN in a Non-secure handler

In addition to stack sealing, other mechanisms help to detect illegal returns, including:

- The integrity signature in the stack frame
- The integrity check in the interrupt Program Status Register

Figure 5.5 shows an example where an illegal return is detected due to these mechanisms. The event

1. During the execution of a Secure software, a Secure interrupt takes place.
2. Secure handler starts, with an exception stack frame in the Secure process stack.
3. During the execution of the Secure handler (2), either a higher priority Non-secure exception (for example, an interrupt) takes place and entered the Non-secure interrupt handler, or the Secure handler calls a Non-secure function.
4. The Non-secure handler attempts to return to the Secure thread mode instead of Secure handler by corrupting the EXC_RETURN value in the Link Register (LR).

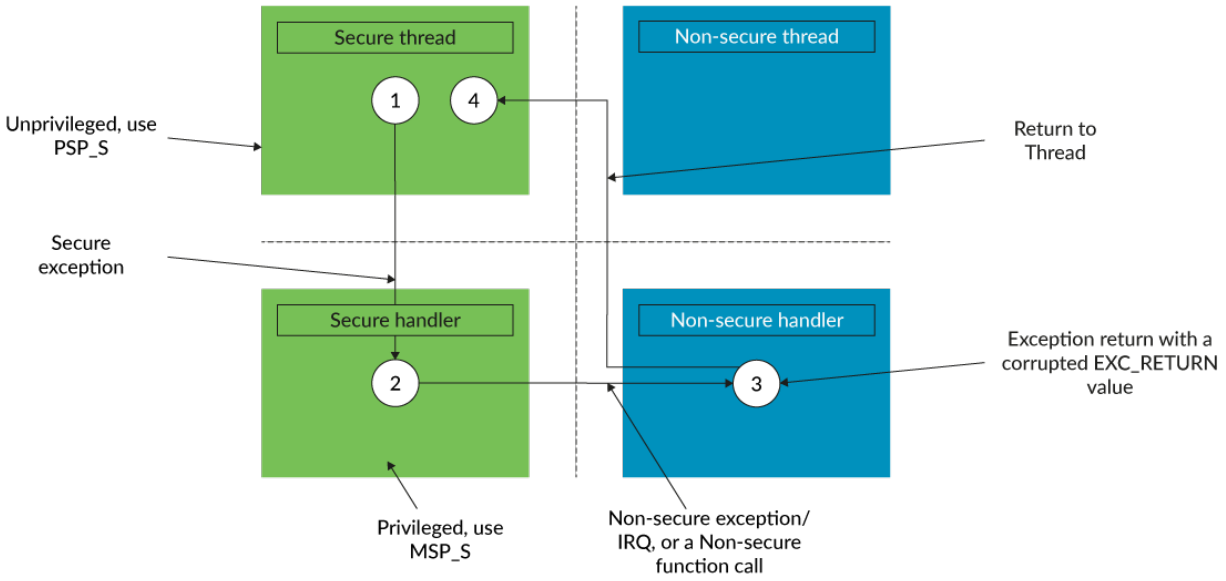


Figure 5.5: Attack example 4: Attack to the Secure main stack by attempting to return to incorrect processor mode

This attack scenario always triggers fault exception even without stack sealing operation. The explanation is as follows:

- When the processor switches from Step 1 to Step 2, the SPSEL bit in the Secure CONTROL register is cleared to zero so that the Secure main stack is used. The exception stack frame being pushed to the Secure process stack does not have the integrity signature and is not sealed.
- When the processor switches from Step 2 to Step 3:
    - If the transition is a Non-secure function call, then a function return stack frame is pushed to the Secure main stack.
    - If the transition is a Non-secure transition, then an exception stack frame is pushed to the Secure main stack, which has an integrity signature.
- After reaching the Non-secure code, a Non-secure software can trigger an invalid exception return. When the processor switches from Step 3 to Step 4, the Secure main stack is used because the CONTROL_S.SPSEL bit is 0.
    - If the transition from Step 2 to Step 3 was a Non-secure function call, a fault is raised. This is because the return address in the function return stack frame will not match the integrity signature expected.
    - If the transition from Step 2 to Step 3 was a Non-secure exception, although the stack frame has an integrity signature, the exception return fails an integrity check for IPSR. This is because the EXC_RETURN value indicates return-to-thread, but the restored IPSR value from the stack frame is not 0. The processor is running an exception handler when it was interrupted.

- If the Secure software uses the Secure Memory Protection Unit (MPU) for stack protection, the illegal return operation also triggers a Secure MemManagment fault exception. This is because the processor is returning to Secure unprivileged thread. However, the stack memory used for the unstacking is the main stack, which is marked as only accessible to privileged code.

6 Summary

The stack sealing operation, together with various security checking mechanisms implemented in the Armv8-M processors, ensures that illegal returns to the Secure world using exception return and function return are detected and always triggers fault exceptions. The fault exception handler can then deal with the issue. This means that stack sealing reduces the security risk.