Open Source Software

Arm Trusted Firmware-A exposure to speculative processor vulnerabilities

Date: 11 June 2018

Version: 1.6

Vulnerabilities summary

- Spectre & Meltdown attacks
  - Spectre – Variant 1 - CVE-2017-5753
  - Spectre – Variant 2 - CVE-2017-5715
  - Meltdown – Variant 3 - CVE-2017-5754
  - Variant 4 – CVE-2018-3639
- Arm official response & Whitepaper:
  - https://developer.arm.com/support/arm-security-updates
- Arm Trusted Firmware-A Official Advisories
  - https://github.com/ARM-software/arm-trusted-firmware/wiki/ARM-Trusted-Firmware-Security-Advisory-TFV-6
  - https://github.com/ARM-software/arm-trusted-firmware/wiki/Trusted-Firmware-A-Security-Advisory-TFV-7
- Arm Firmware Interface document:
  - Firmware interfaces for mitigating cache speculation vulnerabilities

Spectre – Variant 1

```
1 struct array {
2         unsigned long length;
3         unsigned char data[];
4 };
5 struct array *arr1 = ...; /* small array */
6 struct array *arr2 = ...; /*array of size 0x400 */
7 unsigned long untrusted_offset_from_user = ...;
8 if (untrusted_offset_from_user < arr1->length) {
9         unsigned char value;
10        value =arr1->data[untrusted_offset_from_user];
11        unsigned long index2 =((value&1)*0x100)+0x200;
12        if (index2 < arr2->length) {
13                unsigned char value2 = arr2->data[index2];
14        }
15 }
```

From Arm whitepaper: "*One value speculatively loaded is then used to construct an address for a second load or indirect branch that can also be performed speculatively, that second load or indirect branch can leave an indication of the value loaded by the first speculative load in a way that could be read using a timing analysis of the cache by code that would otherwise not be able to read that value.*"

Arm Trusted Firmware-A Status as of May 2018:

- Manual inspection of upstream codebase showed **no occurrence** of this vulnerable pattern so far
- No workarounds currently developed nor planned as of May 2018

arm

Spectre – Variant 2 on Armv8 Cortex cores

CPU-specific vulnerability

- General recommendation is to invalidate the Branch Predictor (BP) as early as possible on entry into the secure world, before any branch instruction is executed
- There are various different implementation defined ways to achieve that goal

Arm Trusted Firmware-A available workarounds: PR 1214 / PR 1228 / PR 1240

Workaround description for affected Armv8 Cortex cores (AArch64):

- **Cortex-A57 & Cortex-A72**: MMU disable/enable (toggling) on EL3 entry to invalidate the BP
  - If MMU toggling unfeasible at lower ELs, any SMC call into the EL3 Firmware will cause BP invalidation
- **Cortex-A73 & Cortex-A75**: on EL3 entry, temporarily dropping into AArch32 S-EL1 and executing `BPIALL` instruction
  - MMU toggling is not effective at invalidating the BP on A73/A75

Cortex-A32, Cortex-A53, Cortex-A55 and Cortex-A76 are **NOT** affected by this vulnerability

arm

Spectre – Variant 2 on Armv8 Cortex cores (2)

Lower ELs software dependencies on Arm Trusted Firmware-A changes:

- If other ELs software (e.g.: Rich OS) wants to leverage EL3 Firmware (by issuing an SMC) to perform Branch Predictor invalidation, then a TF update, including the workarounds here described, **must** be deployed first in order for the BP invalidation to be effective

- If instead other privileged software directly implements the workaround to invalidate the Branch Predictor at its Exception Level (for example, by performing “MMU disable/enable” itself), then there is no such dependency on Firmware changes to be deployed

Spectre – Variant 2 on Armv7 Cortex cores

On Armv7 AArch32 affected Cortex cores:

- **Cortex-A8, Cortex-A9, Cortex-A12, Cortex-A17**
  - `BPIALL` instruction should be executed as early as possible on entry into the secure world to invalidate the Branch Predictor
    - For **Cortex-A8** also set `ACTLR[6]` to 1 during early processor initialization
- **Cortex-A15**
  - `BPIALL` instruction is not effective at invalidating the Branch Predictor
  - Set instead `ACTLR[0]` to 1 during early processor initialisation and then execute `ICIALLU` instruction to invalidate the Branch Predictor

Cortex-A5 and Cortex-A7 are **NOT** affected by this vulnerability

Since monitor and secure-SVC layers are tightly integrated on AArch32 systems, usually as part of a Trusted OS, recommendation is to seek for a workaround by Trusted OS vendors

An example implementation of the workarounds for Cortex-A9, Cortex-A15 and Cortex-A17 in the minimal AArch32 Secure Payload (SP_MIN) is provided as a reference in Arm Trusted Firmware-A Pull Request 1228

arm

Spectre – Variant 2 – SMCCC v1.1 support

Pull Request 1240 implements support for the SMCCC v1.1 specification, as defined in the document Firmware interfaces for mitigating cache speculation vulnerabilities

It must be used in conjunction with the Pull Request 1214

Summary of the changes:

- Added support for SMCCC_VERSION and SMCCC_ARCH_FEATURES
- Added support for the optimized SMCCC_ARCH_WORKAROUND_1 for use by normal world AArch64 privileged software (more efficient than an arbitrary SMC, like PSCI_VERSION)
- Added SMCCC_VERSION discovery via PSCI_FEATURES
- Optimized `BPIALL` workaround
- Other SMCs (e.g.: PSCI_VERSION) will also trigger Branch Predictor invalidation, but less efficiently
- Workaround for Variant2 (CVE-2017-5715) is disabled on upstream unaffected platforms

Spectre – Variant 2 – Considerations

Workarounds are enabled by default on vulnerable Armv8 CPUs (A57, A72, A73, A75)

Platforms can choose to disabled them at compile time

Pull Request 1240 disables the workarounds for unaffected upstream platforms

Workarounds have been currently measured servicing both PSCI_VERSION and SMCCC_ARCH_WORKAROUND_1 SMCs on a Juno-r1 (A57) target, with the following results:

Test	Time (ns)*
PSCI_VERSION baseline (without PRs in Advisory TFV-6)	515
PSCI_VERSION baseline (with PRs in Advisory TFV-6)	527
PSCI_VERSION with “MMU disable/enable”	930
SMCCC_ARCH_WORKAROUND_1 with “MMU disable/enable”	386
PSCI_VERSION with “BPIALL at AArch32 Secure-EL1”	1276
SMCCC_ARCH_WORKAROUND_1 with “BPIALL at AArch32 Secure-EL1”	770

*: Including the time spent in the caller implementing SMCCC from AArch64. For the SMCCC_ARCH_WORKAROUND_1 cases the caller uses SMCCC v1.1

arm

Spectre – Variant 2 – Cortex cores summary

Cortex cores summary of Arm Trusted Firmware-A workarounds for Variant 2

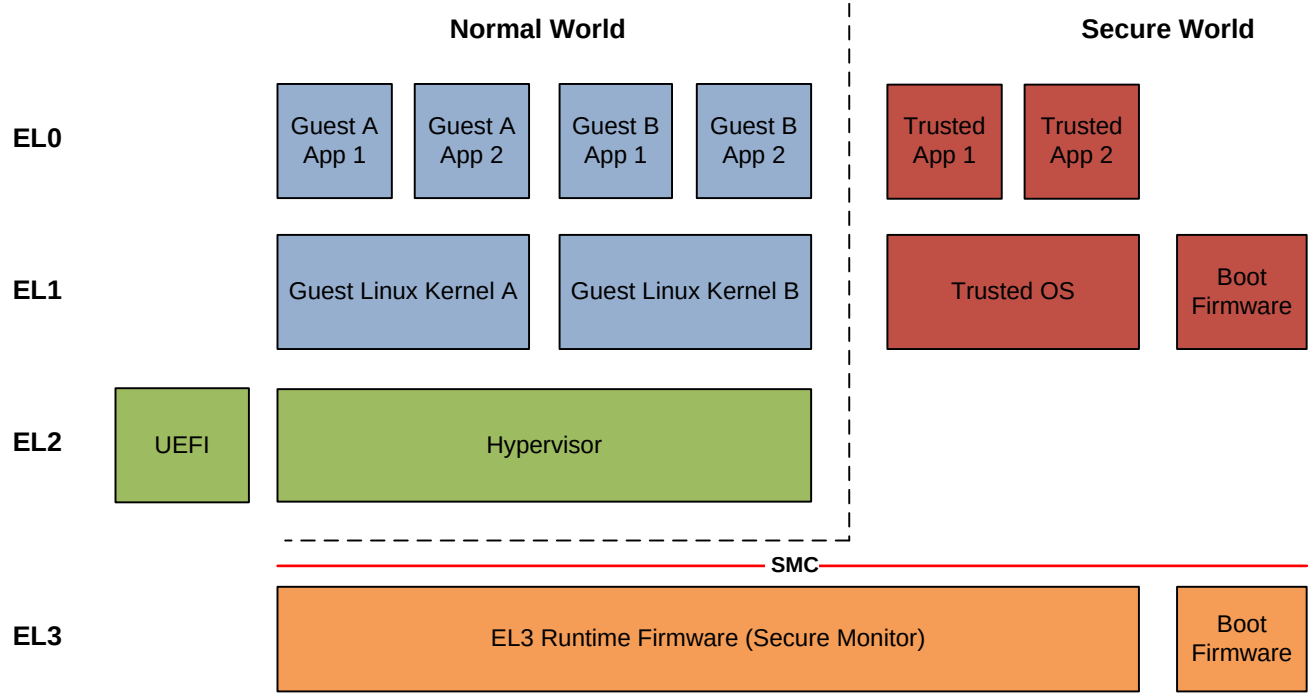
	Cortex Cores	AArch64 Workarounds	AArch32 Workarounds	Where
Armv8	Cortex-A57 Cortex-A72	MMU Disable/Enable	NOT AVAILABLE*	Trusted Firmware PRs:1214/1240
	Cortex-A73 Cortex-A75	BPIALL into AArch32 S-EL1	NOT AVAILABLE**	Trusted Firmware PRs:1214/1240
	Cortex-A35 Cortex-A53 Cortex-A55 Cortex-A76	NOT AFFECTED	NOT AFFECTED	N/A
	Cortex-A32	N/A	NOT AFFECTED	N/A
Armv7	Cortex-A8	N/A	BPIALL + ACTRL[6] to 1	Trusted OS vendors
	Cortex-A9 Cortex-A12 Cortex-A17	N/A	BPIALL instruction	Trusted OS vendors / Trusted Firmware Example PR 1228
	Cortex-A15	N/A	ICIALLU + ACTRL[0] to 1	Trusted OS vendors Trusted Firmware Example PR 1228
	Cortex-A5 Cortex-A7	N/A	NOT AFFECTED	N/A

*: No workaround implemented when in AArch32

**: No upstream AArch32 EL3 support in Arm Trusted Firmware-A

arm

Variant 3 – Meltdown



Meltdown attack is only exploitable between Exception Levels within the same translation regime

EL0 App → reading EL1 Rich OS Memory

S-EL0 Trusted Apps → reading S-EL1 Trusted OS memory

It cannot be used to access Secure Memory from non-secure world

Not applicable to Arm Trusted Firmware-A

Secure Payloads (e.g.: Trusted OS) should provide mitigations on vulnerable CPUs to protect themselves from exploited Secure-EL0 applications.

arm

Variant 4 – CVE-2018-3639

Variant 4 derives a Spectre type attack from CPUs stores & loads re-ordering feature

Affected Arm Cortex cores are:

- Cortex-A57, Cortex-A72, Cortex-A73, Cortex-A75 and Cortex-A76

As of May 2018, Arm is not aware of any Variant 4 exploit against Trusted Firmware-A

- It is likely to be very difficult to achieve a Variant 4 exploit against current standard configurations of TF-A, due to the limited interfaces into the secure world with attacker-controlled inputs
- However, this is becoming increasingly difficult to guarantee with the introduction of complex new firmware interfaces, for example SDEI. Additionally, TF-A lacks visibility of all vendor-supplied interfaces
- Therefore, TF-A takes a conservative approach by mitigating Variant 4 wherever possible during secure world execution

A general firmware mitigation is available on all impacted Arm cores

- Trusted Firmware-A can set a CPU implementation specific control bit to prevent the re-ordering of stores and loads, effectively preventing Variant 4 exploits to be applicable
- The CPU-specific mitigation code is enabled by default on affected cores, but it can be disabled at compile time for platforms which are unaffected or where the risk is deemed to be low enough

arm

Variant 4 – Static Mitigation

There are two possible CPU-specific approaches in Trusted Firmware-A for applying the mitigation: Static or Dynamic mitigation

1. Static mitigation (Pull Request 1392)

- This is the recommended approach for Cortex-A57, Cortex-A72, Cortex-A73, Cortex-A75
- Trusted Firmware-A enables the mitigation during EL3 initialisation
- **There is no CPU-specific mechanism provided to disable the mitigation at runtime from lower ELs, so the mitigation will be permanently applied to the entire software stack**
- TF-A CPU-specific implementation:
  - **Cortex-A57** & **Cortex-A72**: by setting bit 55 of `CPUACTLR_EL1`(`S3_1_C15_C2_0`)
  - **Cortex-A73**: by setting bit 3 of `S3_0_C15_C0_0` (not documented in the TRM)
  - **Cortex-A75**: by setting bit 35 (reserved in TRM) of `CPUACTLR_EL1`(`S3_0_C15_C1_0`)

Variant 4 – Dynamic Mitigation

2. Dynamic mitigation (investigation ongoing)

- This approach is currently being developed for Arm Cortex-A76 core (Pull Request 1397)
- Trusted Firmware-A enables the mitigation during EL3 initialisation
  - **For Cortex-A76**: by setting bit 16 (reserved in TRM) of `CPUACTLR2_EL1`(`S3_0_C15_C1_1`)
- **TF-A will additionally implement a newly defined SMCCC_ARCH_WORKAROUND_2 to allow callers at lower ELs to dynamically enable or disable mitigation for Variant 4 in their execution context**
- SMCCC_ARCH_WORKAROUND_2 will be discoverable through SMCCC_ARCH_FEATURES
- TF-A will enable the mitigation on every entry into EL3 and it will restore the mitigation state of the lower exception level on exit from EL3
- So, even without any mitigation code in other software components, this approach will effectively permanently mitigate the entire software stack, since the default mitigation state for firmware-managed execution contexts is enabled
- On systems where more software will execute with mitigation disabled, this approach may result in better system performance than the static one for some use-cases. However, for other systems or use-cases, this performance saving may be outweighed by the additional overhead of SMCCC_ARCH_WORKAROUND_2 calls and TF-A exception handling

arm

The Arm trademarks featured in this presentation are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.

www.arm.com/company/policies/trademarks

The information provided in this document is based on Arm's present understanding of the relevant vulnerabilities and mitigations, and is subject to change as additional information is revealed and as additional analyses are performed.