Cache Speculation Side-channels Linux kernel mitigations

Date: 26 February 2018

Version : 1.5

Linux kernel mitigations

These slides are based on information available in the Arm Whitepaper
@ https://developer.arm.com/support/security-update

Variant	Exploit	Kernel Mitigation
1 Spectre	Bypassing software checking of untrusted values	Inhibit speculation of out-of-bounds-pointer at loads/pointer access Requires identification of (multiple) vulnerable locations Variant 1 mitigations are expected to be an ongoing process (cross-architecture)
2 Spectre	Speculation through training branch predictors	Invalidate branch predictors at key points in kernel corresponding to trust boundaries: Context switch, VM exit, interrupt
3 Meltdown	Speculative reads of inaccessible data	Protection via **Kernel Page Table Isolation** (**KPTI**) Switch entire page tables between userspace and kernel (two sets of page tables) on OS entry/exit
3a Meltdown	Speculative reads of system register values (Arm specific)	Variant 3 mitigation provides protection for vector base address. Investigating if additional software mitigations are necessary.

arm

Arm Linux Kernel mitigations update

Arm has developed Linux kernel mitigations for Variants 1/2/3

Arm64 Variant 1/2/3 patches are now queued for Linux-4.16.

Patches support Arm64 (Arm v8.x-A cores) and initial support for Arm32

Linaro are providing backports: **www.linaro.org/downloads/security/**

git.linaro.org/kernel/speculation-fixes-staging (currently 4.4 / 4.9 / 4.14)

Android backports to 3.18 / 4.4 / 4.9 / 4.14 available on https://android.googlesource.com/kernel/common.git

arm

Arm 'kpti' branch patch stack:

Arm has produced an initial patch stack:
https://git.kernel.org/pub/scm/linux/kernel/git/arm64/linux.git/log/?h=kpti

'kpti' branch is a staging integration which includes Variant 1/2/3 support (not just kpti)

Currently assembled on top of Linux-4.15

Full patch stack listed on next 3 slides

arm

[Variant */Spectre */Meltdown] arm64: **Add README describing the branch** security_fixes_20180208 kpti

README

[Variant 2/Spectre-v2] arm: KVM: Invalidate icache on guest exit for Cortex-A15
[Variant 2/Spectre-v2] arm: Invalidate icache on prefetch abort outside of us...
[Variant 2/Spectre-v2] arm: Add icache invalidation on switch_mm for Cortex-A15
[Variant 2/Spectre-v2] arm: KVM: Invalidate BTB on guest exit for Cortex-A12/A17
[Variant 2/Spectre-v2] arm: Invalidate BTB on prefetch abort outside of user ...
[Variant 2/Spectre-v2] arm: Add BTB invalidation on switch_mm for Cortex-A9, ...
[Variant 2/Spectre-v2] arm64: Kill PSCI_GET_VERSION as a variant-2 workaround arm64-spectre-meltdown-for-4.15-stable
[Variant 2/Spectre-v2] arm64: Add ARM_SMCCC_ARCH_WORKAROUND_1 BP hardening su...
[Variant 2/Spectre-v2] arm/arm64: smccc: Implement SMCCC v1.1 inline primitive
[Variant 2/Spectre-v2] arm/arm64: smccc: Make function identifiers an unsigne...
[Variant 2/Spectre-v2] firmware/psci: Expose SMCCC version through psci_ops
[Variant 2/Spectre-v2] firmware/psci: Expose PSCI conduit
[Variant 2/Spectre-v2] arm64: KVM: Add SMCCC_ARCH_WORKAROUND_1 fast handling
[Variant 2/Spectre-v2] arm64: KVM: Report SMCCC_ARCH_WORKAROUND_1 BP hardenin...
[Variant 2/Spectre-v2] arm/arm64: KVM: Turn kvm_psci_version into a static in...
[Variant 2/Spectre-v2] arm64: KVM: Make PSCI_VERSION a fast path
[Variant 2/Spectre-v2] arm/arm64: KVM: Advertise SMCCC v1.1
[Variant 2/Spectre-v2] arm/arm64: KVM: Implement PSCI 1.0 support
[Variant 2/Spectre-v2] arm/arm64: KVM: Add smccc accessors to PSCI code
[Variant 2/Spectre-v2] arm/arm64: KVM: Add PSCI_VERSION helper
[Variant 2/Spectre-v2] arm/arm64: KVM: Consolidate the PSCI include files
[Variant 2/Spectre-v2] arm64: KVM: Increment PC after handling an SMC trap
[Variant 2/Spectre-v2] arm64: Branch predictor hardening for Cavium ThunderX2
[Variant 2/Spectre-v2] arm64: Implement branch predictor hardening for Falkor
[Variant 2/Spectre-v2] arm64: Implement branch predictor hardening for affect...
[Variant 2/Spectre-v2] arm64: cputype: Add missing MIDR values for Cortex-A72...
[Variant 2/Spectre-v2] arm64: entry: Apply BP hardening for suspicious interr...
[Variant 2/Spectre-v2] arm64: entry: Apply BP hardening for high-priority syn...

Variant 2

arm support for v7-A cores using inline code sequences

arm64 support for v8-A cores using SMCCC

arm

[Variant 2/Spectre-v2] arm64: KVM: Use per-CPU vector when BP hardening is en...
[Variant 2/Spectre-v2] arm64: Move BP hardening to check_and_switch_context
[Variant 2/Spectre-v2] arm64: Add skeleton to harden the branch predictor aga...
[Variant 2/Spectre-v2] arm64: Move post_ttbr_update_workaround to C code
[Variant 2/Spectre-v2] drivers/firmware: Expose psci_get_version through psci...
[Variant 2/Spectre-v2] arm64: cpufeature: Pass capability structure to ->enab...
[Variant 2/Spectre-v2] arm64: Run enable method for errata work arounds on la...
[Variant 2/Spectre-v2] arm64: cpufeature: __this_cpu_has_cap() shouldn't stop...

Variant 2 – arm64

[Variant 1/Spectre-v1] arm64: futex: Mask __user pointers prior to dereference
[Variant 1/Spectre-v1] arm64: uaccess: Mask __user pointers for __arch_{clear...
[Variant 1/Spectre-v1] arm64: uaccess: Don't bother eliding access_ok checks ...
[Variant 1/Spectre-v1] arm64: uaccess: Prevent speculative use of the current...
[Variant 1/Spectre-v1] arm64: entry: Ensure branch through syscall table is b...
[Variant 1/Spectre-v1] arm64: Use pointer masking to limit uaccess speculation
[Variant 1/Spectre-v1] arm64: Make USER_DS an inclusive limit
[Variant 1/Spectre-v1] arm64: Implement array_index_mask_nospec()
[Variant 1/Spectre-v1] arm64: barrier: Add CSDB macros to control data-value ...
[Variant 1/Spectre-v1] array_index_nospec: Sanitize speculative array de-refe...
[Variant 1/Spectre-v1] Documentation: Document array_index_nospec

Variant 1 – arm64
Use data speculation barrier

[Variant 3/Meltdown] perf: arm_spe: Fail device probe when arm64_kernel_unmap...
[Variant 3/Meltdown] arm64: idmap: Use "awx" flags for .idmap.text .pushsecti...
[Variant 3/Meltdown] arm64: entry: Reword comment about post_ttbr_update_work...
[Variant 3/Meltdown] arm64: Force KPTI to be disabled on Cavium ThunderX
[Variant 3/Meltdown] arm64: kpti: Add ->enable callback to remap swapper usin...
[Variant 3/Meltdown] arm64: mm: Permit transitioning from Global to Non-Globa...
[Variant 3/Meltdown] arm64: kpti: Make use of nG dependent on arm64_kernel_un...
[Variant 3/Meltdown] arm64: Turn on KPTI only on CPUs that need it

Variant 3 – arm64
KPTI

arm

[Variant 3/Meltdown] arm64: cputype: Add MIDR values for Cavium ThunderX2 CPUs
[Variant 3/Meltdown] arm64: kpti: Fix the interaction between ASID switching ...
[Variant 3/Meltdown] arm64: mm: Introduce TTBR_ASID_MASK for getting at the A...
[Variant 3/Meltdown] arm64: capabilities: Handle duplicate entries for a capa...
[Variant 3/Meltdown] arm64: Take into account ID_AA64PFR0_EL1.CSV3
[Variant 3/Meltdown] arm64: Kconfig: Reword UNMAP_KERNEL_AT_EL0 kconfig entry
[Variant 3/Meltdown] arm64: Kconfig: Add CONFIG_UNMAP_KERNEL_AT_EL0
[Variant 3/Meltdown] arm64: use RET instruction for exiting the trampoline
[Variant 3/Meltdown] arm64: kaslr: Put kernel vectors address in separate dat...
[Variant 3/Meltdown] arm64: entry: Add fake CPU feature for unmapping the ker...
[Variant 3/Meltdown] arm64: tls: Avoid unconditional zeroing of tpidrro_el0 f...
[Variant 3/Meltdown] arm64: cpu_errata: Add Kryo to Falkor 1003 errata
[Variant 3/Meltdown] arm64: erratum: Work around Falkor erratum #E1003 in tra...
[Variant 3/Meltdown] arm64: entry: Hook up entry trampoline to exception vectors
[Variant 3/Meltdown] arm64: entry: Explicitly pass exception level to kernel_...
[Variant 3/Meltdown] arm64: mm: Map entry trampoline into trampoline and kern...
[Variant 3/Meltdown] arm64: entry: Add exception trampoline page for exceptio...
[Variant 3/Meltdown] arm64: mm: Invalidate both kernel and user ASIDs when pe...
[Variant 3/Meltdown] arm64: mm: Add arm64_kernel_unmapped_at_el0 helper
[Variant 3/Meltdown] arm64: mm: Allocate ASIDs in pairs
[Variant 3/Meltdown] arm64: mm: Fix and re-enable ARM64_SW_TTBR0_PAN
[Variant 3/Meltdown] arm64: mm: Rename post_ttbr0_update_workaround
[Variant 3/Meltdown] arm64: mm: Remove pre_ttbr0_update_workaround for Falkor...
[Variant 3/Meltdown] arm64: mm: Move ASID from TTBR0 to TTBR1
[Variant 3/Meltdown] arm64: mm: Temporarily disable ARM64_SW_TTBR0_PAN
[Variant 3/Meltdown] arm64: mm: Use non-global mappings for kernel space

Linux 4.15

Variant 3 – arm64

KPTI

arm

Overview of Linux kernel changes

Arm Cortex licensees should take changes from released kernels (from 4.16 on), or backports from Linaro

- Changes are in core and generic code and do not need to be further modified*
- Linaro providing backports to LTS (Long Term Support) kernels 4.4 / 4.9 / 4.14.
  See https://www.linaro.org/downloads/security/ for further details.

**Arm has been working with Arm architecture licensees (those companies who design their own CPUs that implement the Arm architecture. These CPUs do not use the Cortex® brand in their model names. Small additional patches may be required, contact your silicon partner (e.g. use of core-specific Branch predictor invalidate, etc)*

It should not be necessary for silicon partners or OEMs using Arm designed Cortex®-A CPUs to cherry-pick, or otherwise modify the patches

For Linux versions 4.16 and later, the 'kpti' branch may only contain small incremental changes (if anything)

arm

Variant 1: Kernel implementation detail

Aligned with Linux-4.16 mainline implementation for the new **array_index_mask_nospec()** macro

arm64 patches provide:

- An arm64 version of **array_index_mask_nospec()** to suppress any compiler optimisations that could introduce unwanted speculative paths
- Masking of the syscall number to restrict speculation through the syscall table
- Masking of __user pointers prior to deference in uaccess routines

Vulnerable sites in kernel need to be identified: active discussions on LKML about how to identify locations in the kernel. This is an architecture-agnostic issue.

Fixes for Variant1 in the kernel are expected to be an ongoing effort, perhaps many months.
Vendors should expect to issue further kernel updates.

Variant 2: Speculation through training branch predictors

Vulnerable points are ‘trust boundaries’ where level of trust changes:

- Entry to kernel: switch between processes, VM entry/exit
- Also affects userspace: JIT to app, etc.

Kernel mitigation relies on invalidation of Branch Predictor at the above trust boundaries

How this is done is subject to Implementation Defined sequences
(full detail here https://developer.arm.com/support/security-update)

- 64-bit uses new SMC call. **This requires support in firmware:**
  https://developer.arm.com/-/media/developer/pdf/ARM_DEN_0070A_Firmware_interfaces_for_mitigating_CVE-2017-5715.pdf
- 32-bit Arm Cortex Armv7-A use in-line BP-invalidate sequences (e.g. ARMv7-A ‘BPIALL’)

arm

Variant 3: KPTI

Background: ‘Kaiser’ is a 2017 proposal to improve KASLR (Kernel Address Space Layout Randomization)

- Address-space layout randomization (ASLR) is a well-known technique to make exploits harder by placing various objects at random, rather than fixed, addresses. Linux has long had ASLR for user-space programs, but Kees Cook [Google security] would like to see it applied to the kernel itself as well
- Update: KASLR enabled for Android Common Kernel 4.4 / 4.9 / 4.14

For more details on Kaiser see Austria paper: **https://gruss.cc/files/kaiser.pdf**

Kaiser was public earlier in 2017 – it proposes **Kernel Page Table Isolation**

Basically **Kaiser == KPTI** and this is the complete Variant 3 mitigation

Variant 3: KPTI

The KPTI patches in the stack protect against Variant 3
(The only Arm designed core affected is Cortex-A75, but Variant 3 may affect other Arm Architecture licensee implementations)

KPTI is implemented for arm64 and is made the default in the Linux kernel as:

- Underlying KPTI support is compiled in by default (use of TTBR0/1 etc.)
- KPTI additionally hardens kernel KASLR (address randomization) which improves security
  (see Kaiser paper for details)
- **Arm64 is particularly efficient at KPTI** with two TTBR registers & ASID & VMID
  First benchmarking indicates minimal performance overhead
- It also provides Variant 3a protection for vector base register

Applying KPTI is not essential for Arm-designed cores, except for Cortex-A75 r0/r1/r2
However, Arm recommends KPTI is enabled by default if at least one CPU in the system is affected.

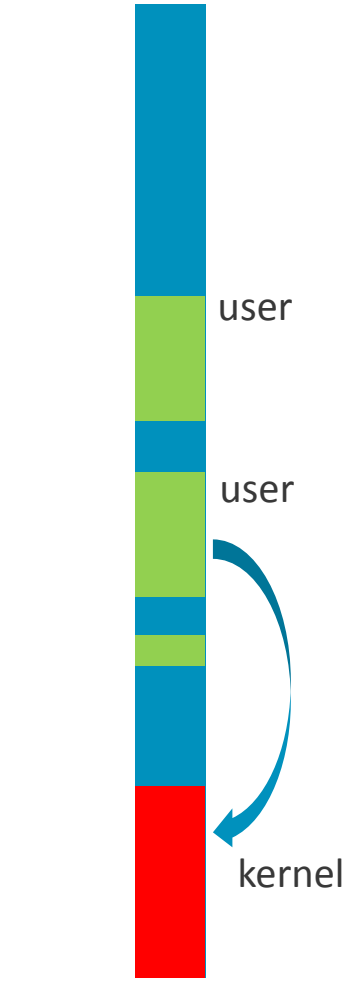
arm

Without KPTI

unmapped

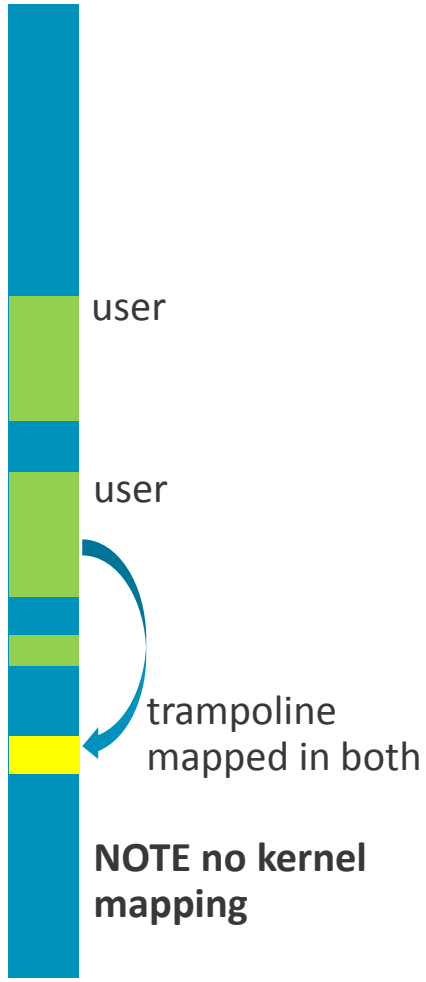
user mapping

kernel mapping

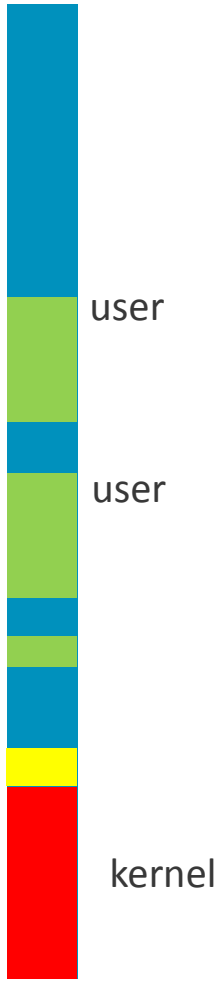


user & kernel mode

With KPTI (highly simplified)



user mode



kernel mode

arm

Variant 3a: System register read

This is not considered to be a security risk, as system registers do not hold ‘interesting’ information.

This exploit could obtain the vector base address (VBAR_EL1, VBAR_EL2...) – probably the most interesting value

When KPTI is enabled, VBAR_EL1 has a fixed value, and doesn't reflect the location of the kernel. This mitigates VBAR access on Cortex-A57 and Cortex-A72 arm64 kernels.

Cortex-A15 remains susceptible (no KPTI on arm32). However, this is not considered to be a security risk because arm32 does not support KASLR, and with a fixed kernel there are various ways to discover kernel addresses in addition to looking in VBAR registers – e.g. obtain the kernel image, etc.

Kernel configuration to enable mitigations

Variant 1	**Applied by default:** New Arm speculation barrier sequence used where needed. **64-bit:** Infrastructure support for arm64 speculation barrier & uaccess mitigations
Variant 2	**Applied by default:** **64-bit:** New SMCCC_ARCH_WORKAROUND_1 call must be implemented by firmware **32-bit: Applied by default:** uses 32-bit architectural BPIALL (most Arm v7-A cores) Can be disabled if **CONFIG_EXPERT=Y** and **CONFIG_HARDEN_BRANCH_PREDICTOR** is not set
Variant 3 (64-bit only)	**Applied by default:** KPTI is enabled for cores that need it, or when KASLR is enabled as it improves kernel KASLR and mitigates Variant 3a Can be disabled if **CONFIG_EXPERT=Y** and **CONFIG_UNMAP_KERNEL_AT_EL0** is not set Kernel command-line option 'kpti=off' allows KPTI to be turned off (no reason to do this)

Summary: no special CONFIG options required to enable mitigations
Arm Linux kernel adopts a 'secure by default' policy

arm

Arm 32-bit mitigations

Arm is committed to support of 32-bit Arm Linux

Variant 1: Generic mitigations exist from mainline 4.16 Variant 1 support.

Fixes for Variant1 in the kernel are expected to be an ongoing effort, perhaps many months.
Vendors should expect to issue further kernel updates

Variant 2:
For v7-A cores: Kernel mitigations are published and use direct BP-Invalidate sequences integrated into appropriate locations
For v8.x-A cores: Patches for Arm32 kernel running on Armv8-A Cortex cores are still in development. Solution will involve SMCC and will require a firmware update in addition to kernel patches

Variant 3: The only Arm Cortex core affected is Cortex-A75, no other Cortex-A cores are impacted

Arm expects 64-bit kernels (which include mitigations) to be used on Cortex-A75 systems, and no Arm32 KPTI support is necessary for other Arm Cortex cores.

There is no KASLR on Arm 32-bit Linux, so KPTI is not relevant.

Kernel patch status summary

	arm64	Arm32
Variant 1	arm64 speculation barrier Queued for 4.16 Expect further updates	Generic mitigations available Expect further updates
Variant 2	Uses SMCCC Queued for 4.16	V7-A cores: Uses BP-Invalidate code sequences - patches posted In community/maintainer review v8-A cores: In development
Variant 3	Queued for 4.16	No KPTI required

arm

Summary

Arm has produced a patchset for Linux Kernel mitigations for arm64 and arm32

- Firstly, partners should investigate specific CPU susceptibility & device / application risk
- Mitigations for Variant 1/2/3 are now available for arm64 and are queued for 4.16
- For Variant 2, a firmware update will be required to support SMCCC v1.1
- Backports available from Linaro & Google
- Opensource process means further updates may be required
  (particularly for Variant 1 where generic C code kernel mitigations will evolve over time)

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.