Architecture Security Advisory
ASA

Version: 1.0-r2

TLB channels, SLAM-like attacks, and transient translation of non-canonical addresses

Non-Confidential

Copyright © 2023 Arm Limited (or its affiliates).
All rights reserved.

Architecture Security Advisory

TLB channels, SLAM-like attacks, and transient translation of non-canonical addresses

Copyright © 2023 Arm Limited (or its affiliates). All rights reserved.

Arm Non-Confidential Document Licence ("Licence")

This Licence is a legal agreement between you and Arm Limited ("**Arm**") for the use of Arm's intellectual property (including, without limitation, any copyright) embodied in the document accompanying this Licence ("**Document**"). Arm licenses its intellectual property in the Document to you on condition that you agree to the terms of this Licence. By using or copying the Document you indicate that you agree to be bound by the terms of this Licence.

"**Subsidiary**" means any company the majority of whose voting shares is now or hereafter owner or controlled, directly or indirectly, by you. A company shall be a Subsidiary only for the period during which such control exists.

This Document is **NON-CONFIDENTIAL** and any use by you and your Subsidiaries ("Licensee") is subject to the terms of this Licence between you and Arm.

Subject to the terms and conditions of this Licence, Arm hereby grants to Licensee under the intellectual property in the Document owned or controlled by Arm, a non-exclusive, non-transferable, non-sub-licensable, royalty-free, worldwide licence to:

(i) use and copy the Document for the purpose of designing and having designed products that comply with the Document;

(ii) manufacture and have manufactured products which have been created under the licence granted in (i) above; and

(iii) sell, supply and distribute products which have been created under the licence granted in (i) above.

Licensee hereby agrees that the licences granted above shall not extend to any portion or function of a product that is not itself compliant with part of the Document.

Except as expressly licensed above, Licensee acquires no right, title or interest in any Arm technology or any intellectual property embodied therein.

THE DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. Arm may make changes to the Document at any time and without notice. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

NOTWITHSTANING ANYTHING TO THE CONTRARY CONTAINED IN THIS LICENCE, TO THE FULLEST EXTENT PETMITTED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT

MATTER OF THIS LICENCE (INCLUDING WITHOUT LIMITATION) (I) LICENSEE'S USE OF THE DOCUMENT; AND (II) THE IMPLEMENTATION OF THE DOCUMENT IN ANY PRODUCT CREATED BY LICENSEE UNDER THIS LICENCE). THE EXISTENCE OF MORE THAN ONE CLAIM OR SUIT WILL NOT ENLARGE OR EXTEND THE LIMIT. LICENSEE RELEASES ARM FROM ALL OBLIGATIONS, LIABILITY, CLAIMS OR DEMANDS IN EXCESS OF THIS LIMITATION.

This Licence shall remain in force until terminated by Licensee or by Arm. Without prejudice to any of its other rights, if Licensee is in breach of any of the terms and conditions of this Licence then Arm may terminate this Licence immediately upon giving written notice to Licensee. Licensee may terminate this Licence at any time. Upon termination of this Licence by Licensee or by Arm, Licensee shall stop using the Document and destroy all copies of the Document in its possession. Upon termination of this Licence, all terms shall survive except for the licence grants.

Any breach of this Licence by a Subsidiary shall entitle Arm to terminate this Licence as if you were the party in breach. Any termination of this Licence shall be effective in respect of all Subsidiaries. Any rights granted to any Subsidiary hereunder shall automatically terminate upon such Subsidiary ceasing to be a Subsidiary.

The Document consists solely of commercial items. Licensee shall be responsible for ensuring that any use, duplication or disclosure of the Document complies fully with any relevant export laws and regulations to assure that the Document or any portion thereof is not exported, directly or indirectly, in violation of such export laws.

This Licence may be translated into other languages for convenience, and Licensee agrees that if there is any conflict between the English version of this Licence and any translation, the terms of the English version of this Licence shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. No licence, express, implied or otherwise, is granted to Licensee under this Licence, to use the Arm trade marks in connection with the Document or any products based thereon. Visit Arm's website at **http://www.arm.com/company/policies/trademarks** for more information about Arm's trademarks.

The validity, construction and performance of this Licence shall be governed by English Law.

Copyright © 2023 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-21585 version 4.0

Web Address

http://www.arm.com

Contact

psirt@arm.com

Contents

Copyright © 2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1 Introduction

Arm has been informed of a new paper named “Leaky Address Masking: Exploiting Unmasked Spectre Gadgets with Noncanonical Address Translation” [1] presenting the SLAM attack that demonstrates how some architectural extensions such as Intel’s Linear Address Masking (LAM), AMD’s Upper Address Ignore (UAI), or Arm’s Top Byte Ignore (TBI) may augment the number of exploitable Spectre gadgets.

Transient translation of non-canonical addresses (TTNA), which has been demonstrated on some AMD systems [2], has a similar impact on security.

Arm considers that these techniques can increase the likelihood of exploiting existing vulnerabilities such as Spectre v2 or BHB by augmenting the number of exploitable gadgets. In the absence of such vulnerabilities, Arm accepts the security risk of SLAM and TTNA.

Arm systems already mitigate against Spectre v2 and BHB, and it is considered the software’s responsibility to protect itself against Spectre v1. Hence, no further action is required to protect against SLAM or TTNA attacks.

Copyright © 2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

2 Technical details

2.1 Cache exfiltration channels

Transient execution attacks such as Spectre or Meltdown often rely on a cache exfiltration channel due to its simplicity and high bandwidth. The most common scenario is Flush+Reload, where:

1. Multiple lines from an adversary-controlled buffer are flushed from the cache.
2. The secret is accessed under speculation and part of its value is masked, shifted[1], and used as an offset on the adversary-controlled buffer.
3. The buffer's cache lines are reloaded measuring their access time. The one causing a cache hit reveals information about the secret value.

This process can be repeated by shifting the part of the secret that is masked and exfiltrated until the whole secret is leaked.

Listing 1. Masked cache exfiltration gadget.

`buffer[(*secret & 0xff) * 4096]`	`ldr x0, [ secret ]`
	`and x0, x0, #0xff`
	`lsl x0, x0, 12`
	`ldr x1, [ buffer, x0 ]`

Meltdown attacks on vulnerable cores require a deep enough pipeline to be able to execute all those instructions during the same speculation window. This is easily satisfied by most out-of-order cores, but it is much more difficult on in-order cores.

Spectre attacks require the presence of such gadgets in a path reachable under speculation with adversary-controlled data. Finding such gadgets in the victim code with an adversary-controlled buffer is challenging but can be automated with binary or source code scanners.

An alternative to relax these requirements is to rely on a Prime+Probe cache channel:

1. The adversary computes eviction sets for every cache set in the system, and all cache sets are primed with adversary-controlled lines.

[1] In theory, shifting by 6 (or multiplying by 64) is enough to ensure that each different value lands into a different cache line. In practice, shifting by 12 (or multiplying by 4096) is used to land into different pages and prevent noise caused by the prefetcher.

Copyright © 2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

2. The secret is accessed under speculation and immediately[2] dereferenced as a valid address.

3. All the eviction sets are accessed to detect the cache set that has been modified by the victim access. This reveals the bits of the address of the secret-dependent load used as index, and thus leak part of the secret.

As before, this process can be repeated by increasing the secret pointer and exfiltrate different bits each time.

For Meltdown, the adversary now only needs to execute 2 instructions, which might make it possible even on in-order cores.

For Spectre, these gadgets are much more likely to be present and reachable in the victim program, and furthermore do not require a shared adversary-controlled buffer.

However, the main impediment to an exploit is that secrets are unlikely to be valid addresses unless the secret data are structured such that this is true. For non-structured data the secret-dependent load is doomed to fault with very high probability, and if that happens no cache refill should occur, and no information be transmitted.

Listing 2. Unmasked cache exfiltration gadget.

`**secret`	`ldr x0, [ secret ]`
	`ldr x1, [ x0 ]`

Note that in the Meltdown scenario[3], the adversary could also use a `ldr w0, [ secret ]` (or even `ldrh w0, [ secret ]`) instruction to ensure that the most-significant bits of X0 are set to zero and ensure that the addresses are always valid.

In a Spectre scenario, the likelihood of a compiler generating a gadget where a 32-bit integer is treated as a pointer should be low. However, with speculation, it is possible that such a dereference occurs under a mis-predicted, and potentially adversary-induced, path.

2.2 Non-canonical addresses, TBI, and speculation

In Arm, a translation regime that supports two VA ranges maintains two sets of translation tables:

- The lower VA range pointed by TTBR0_ELx, going from 0x0 to $2^{(64\text{-}T0SZ)} - 1$.

- The upper VA range pointed by TTBR1_ELx, going from $2^{(64\text{-}T1SZ)}$ to 0xFFFFFFFFFFFFFFFF.

[2] It is also possible to execute further instructions to better encode the value into the cache, as with Flush+reload.
[3] It is very unlikely that a compiler generates such gadgets to be abused in a Spectre attack.

Copyright © 2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Any address in between is considered a non-canonical address. Accesses to non-canonical addresses are invalid and must cause a translation fault.

The TTBR to use is selected by the VA bit[55], and canonical addresses must have all the upper bits of the address equal to VA bit[55].

Arm's Top Byte Ignore (TBI), or analogous mechanisms in other architectures, relax the faulting behavior of non-canonical addresses and permit software to store metadata in the unused bits of an address. In practice, with TBI, if bit[55] of an address is 1 (or 0) then bits[63:56] will be treated as if they were 1 (or 0) when accessing the memory subsystem, regardless of their actual value.

Going back to the previous exfiltration channel (Listing 2. Unmasked cache exfiltration gadget.), TBI increases the probability of an arbitrary value being a canonical address, especially with large VA ranges. Although the probability of random data being a mapped address can still be quite low. On most Arm systems with 48-bit addresses, bits[54:48] must be equal to bit[55]. Arm systems implementing FEAT_LVA can use 52-bit addresses, but bits [54:52] still need to be equal to bit[55].

Similarly, some implementations from other architectures [2] are known to speculatively ignore canonicality checks and perform transient non-canonical loads and stores using only the lower address bits.

2.3 TLB exfiltration channel

2.3.1 Address translation

The MMU translates VAs into PAs by performing a translation table walk. The translation table is organized in multiple levels based on the system's VA range and page granularity, every non-leaf level stores a pointer to the next translation table, and the final level stores the physical output address.

In practice, the VA is split into multiple chunks, and each one (starting with the most-significant bits) is used as an index in the current translation table. For example, in a 48-bit VA space with 4KB granules we have 4 translation levels:

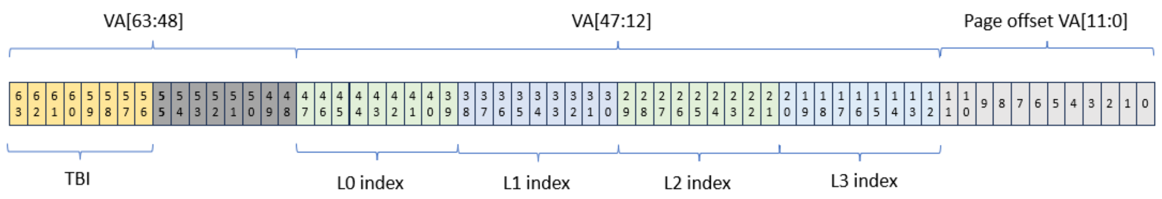

Figure 1. Layout of a virtual address.

Each Translation Table Entry (TTE) is 8 bytes, and the translation table walk process is briefly described in Listing 3.

Copyright © 2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Listing 3. Translation Table walk pseudocode.

```
L1table = *(TTBRx + L0index)
L2table = *(L1table + L1index)
L3table = *(L2table + L2index)
OutputAddress = *(L3table + L3index) + PageOffset
```

This means that for every explicit memory access in a program there can be up to 4 implicit memory accesses that depend on part of the virtual address. Each implicit access has its own cache footprint.

To speed up this process, the Translation Lookaside Buffer (TLB) caches recent translations, to ensure that a walk is only done upon a TLB miss. Hitting or missing the TLB causes a measurable timing difference.

TLBs are set-associative structures like other caches, and they are tagged by VA and ASID/VMID, so that translations from different contexts can be stored simultaneously without having to issue TLB invalidations.

2.3.2 Exfiltration channel

Although caches still are the most common exfiltration channel, TLBs can sometimes present advantages. We describe an Evict+Reload scenario:

1. The adversary needs a TLB eviction set to evict all the translations in the TLB, which normally will also serve for L1D evictions.
2. The secret is accessed under speculation and dereferenced.
3. The adversary measures the access time to a set of addresses.

The reload phase is possible because the userspace adversary and kernel victim share the same translations if they use the same ASID. While E0PD1 prevents EL0 from performing TTBR1 translations, there is no E1PD0 counterpart to prevent EL1 from translating TTBR0 addresses and thus causing TLB refills.

Privileged Access Never (PAN) prevents EL1 from accessing EL0 locations, but a translation is still required to obtain the permissions and evaluate whether the privileged access should be restricted. PAN prevents cache refills to EL0 accessible locations, but it does not prevent the translation table walk or the corresponding TLB refills.

Note that even with TBI or TTNA the secret must still be a valid (i.e., mapped) EL0 address in order to cause a measurable TLB refill. To ensure that, the adversary can map arbitrarily many addresses in their own address space. The only limitation in the worst case is that the secret bit[55] is 0.

Bit[55] being 0 is not a limitation for structured data formats with predictable 0 locations. For example, the top bit of any ASCII byte is 0. This means that bit[55] is 0 for any bytealigned 64-bit chunk of an ASCII string encoded as a VA. Secrets encoded like this will be interpreted as EL0 addresses.

A naïve reload buffer requires impractical memory overhead, but by leveraging some prior knowledge about the secret, the entropy can be reduced such that memory overhead is manageable. For this approach, the adversary uses a sliding window over the secret to leak different parts at each iteration.

The reload buffer of VAs is initialized with known bits of the secret data in corresponding bits of all the reload buffer's VAs, with the rest of these VAs taking all possible values in the remaining bits. This results in a non-contiguous reload buffer in all cases except when the unknown part of the secret is in the least significant bits of the VA. It is guaranteed by brute force that this reload buffer contains a VA with the correct secret values. In vulnerable devices and configurations, an ordinary Evict+Reload after the speculative execution of the gadget in Listing 2. Unmasked cache exfiltration gadget. will reveal which one is correct.

At each subsequent iteration the reload buffer is remapped such that the newly recovered bits are shifted equally for all candidate VAs. If the known part of the secret is the least significant bits, then shifting is to the right, which results in a non-contiguous buffer. If the most significant bits of the secret are known, then shifting is to the left, which produces a contiguous reload buffer at each iteration. Forward and reverse shifting can be combined to leak secret data of arbitrary length.

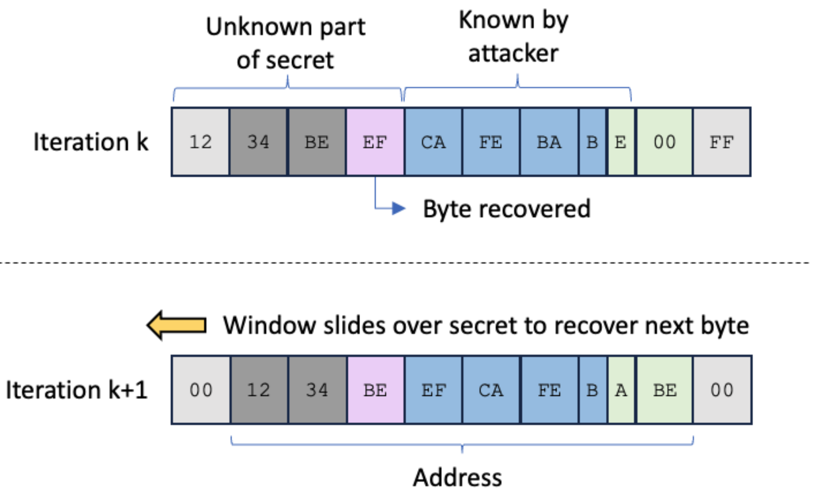

Figure 2. Sliding to the left over a secret to leak different bytes on each iteration.

For example, Figure 2 illustrates two iterations of the sliding window approach. The colors represent the part of a VA pointing to the page (blue), the page offset (green), portions of the address not used during translation (dark gray), bytes of secret data outside the current range of bytes used to form the address (light gray), and a secret byte to exfiltrate (purple). At iteration k (top of Figure 2), the reload buffer will contain 256 pages with addresses ending with 0xCAFEBABE, and the byte 0xEF is recovered. On iteration k+1 the adversary will remap the reload buffer such that it contains addresses ending with 0xEFCAFEBA, and 0xBE will be recovered, and so on.

Copyright © 2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3 Are Arm cores affected?

The answer will depend on the specific system and its configuration. Regardless, in the absence of other vulnerabilities, like Spectre-v2 and Spectre-BHB, Arm considers the risk of SLAM low.

An Arm system with FEAT_LVA using 52-bit addresses and TBI still requires bits[55:52] of a secret to be 0b0000 in order to be dereferenced and leaked through the TLB channel. It is possible that future extensions add support for 56-bit addresses or further relax the canonicality checks to reuse those bits (see PAC or MTE, for instance). This could increase the attack surface for TLB exfiltration channels.

As for Transient Translation of Non-Canonical Addresses (TTNA), although not forbidden by the architecture, most Arm implementations do not exhibit this behavior.

One of the main requirements for the TLB exfiltration channel is that both ends (kernel and userspace) share the same TLB entries. This is only possible on translation regimes supporting two VA ranges where both parties use the same ASID. On Linux with KPTI[4] [5] enabled, a pair of consecutives ASIDs is allocated for each process, and on every entry/exit to/from the kernel the corresponding one is set in TTBR1_EL1.ASID[6]. Note that even with different translation regimes or different ASIDs, there exists some residual leakage due to TLB contention, but that significantly hinders the probing side of the channel.

More importantly, in contrast to other systems[7], the Arm Linux kernel enables Spectre-BHB mitigations by default. Therefore, it is not possible for userspace to easily influence the kernel's branch predictions to branch into an exploitable gadget. Which effectively mitigates the risk of these attacks.

[4] Kernel Page Table Isolation is the software-based mitigation against Meltdown.
[5] FEAT_E0PD1 does not affect the TLB channel as privileged EL1 translations and accesses to user space are not limited, and neither are EL0 accesses.
[6] TCR_EL1.A1 is always set to 1.
[7] For example, Intel considered disabling untrusted eBPF sufficient to prevent Spectre-BHB exploits, but for full protection it is required to manually re-enable retpoline.

Copyright © 2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4 Mitigations

Arm considers that the described techniques only increase the attack surface of existing vulnerabilities such as Spectre v2 or BHB by augmenting the number of exploitable gadgets. In the absence of such a vulnerability, the security risk of SLAM or TTNA is low and can be accepted, and thus Arm does not require any further actions from software.

Arm systems already mitigate against Spectre v2 and BHB attacks [3]. The risk of Spectre v1 attacks, which could also benefit from SLAM or TTNA gadgets, is accepted by the architecture and it is the software's responsibility to protect itself against them.

On affected systems without KPTI where the risk of TLB exfiltration channels is not acceptable, it is possible to implement a two-ASID approach like KPTI does for Meltdown, but without the need of un-mapping translation tables, since CSV3 and E0PD1 must be already present.

Copyright © 2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5 References

1. “Leaky Address Masking: Exploiting Unmasked Spectre Gadgets with Noncanonical Address Translation”. Mathé Hertogh, Sander Wiebing, and Cristiano Giuffrida from the VUSec group at VU Amsterdam. 2023. https://vusec.net/projects/slam

2. “Transient execution of non-canonical accesses”. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-1010.html

3. “Spectre-BHB: Speculative Target Reuse Attacks” https://developer.arm.com/documentation/102898/latest/

Copyright © 2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential