DisARMed: Memory Aliasing Attacks on Arm TrustZone

Jacqueline Henes, Matthew Bowden, Mihai Ordean, David Oswald

University of Birmingham, UK

What we did...

- Categorise Arm-A aliasing behavior
- RAM-based exploit targeting 64-bit ARM platforms
  - Leak TrustZone secure memory regions
  - Non-privileged userspace exploit chain

PoC code for above & Arm TFA patch available!



Context: Memory Aliases

Modifying DIMM SPD data results in 'doubling' or aliasing

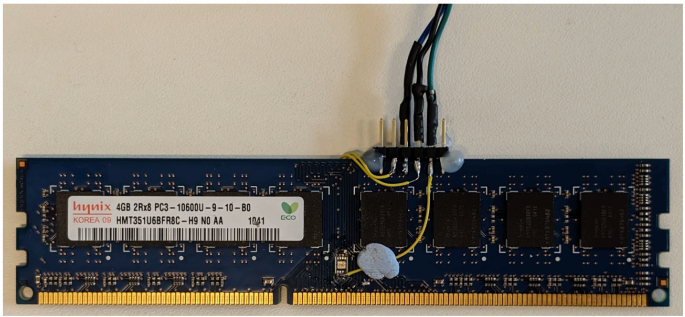


Figure 10. Closeup of the removed SPD chip from the DIMM in Figure 9. Note that some connections are made to the vias on the backside.

Source: De Meulemeester et al., BadRAM: Practical Memory Aliasing Attacks on Trusted Execution Environments

Context: Memory Aliases

Modifying DIMM SPD data results in 'doubling' or aliasing

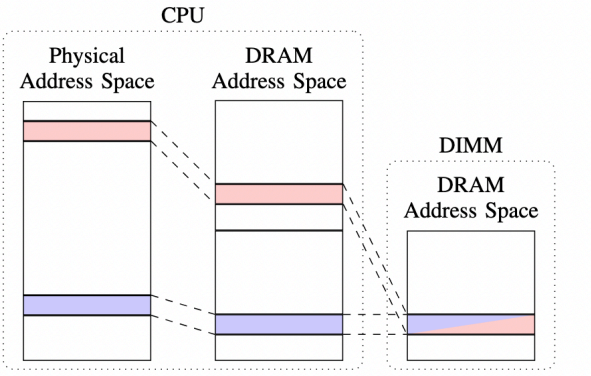


Figure 4. An incorrect configuration of the memory controller can result in unused address bits. Two addresses that only differ in the ghost bit alias to the same physical location inside the DRAM memory as this bit is ignored.

Source: De Meulemeester et al., BadRAM: Practical Memory Aliasing Attacks on Trusted Execution Environments

Motivation...

- Architectures
  - *AMD*: SEV-SNP exploit
  - *Intel*: Boottime alias checks
  - *RISCV*: PMP exploit
  - *Arm*: ?
- Do we always need to be in the kernel?

Target Platforms

Arm architecture with TrustZone support

$\Rightarrow$ Armv8-A 64-bit platforms

- Ampere Altra Developer Platform
- Layerscape LS1046A
- Arm Morello

Rationale: Server Usecases

- Fits attacker model
- Likely to use DIMM sockets
- Likely to have TrustZone support

How does Arm TrustZone work anyway?

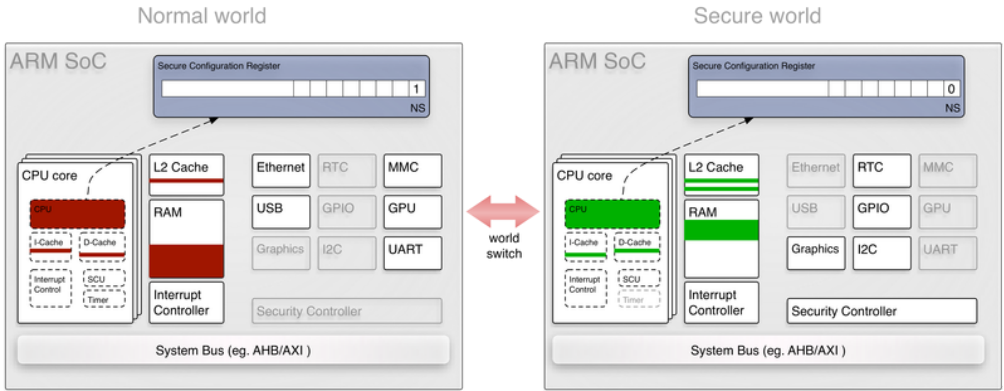
What Is TrustZone Technology?

TrustZone technology within Cortex-A based application processors is commonly used to run trusted boot and trusted OS to create a Trusted Execution Environment (TEE). Typical use cases include the protection of authentication mechanisms, cryptography, mobile device management, payment, key material, and digital rights management (DRM).

https://www.arm.com/technologies/trustzone-for-cortex-a

Execution states split into *secure* and *non-secure* 'worlds'

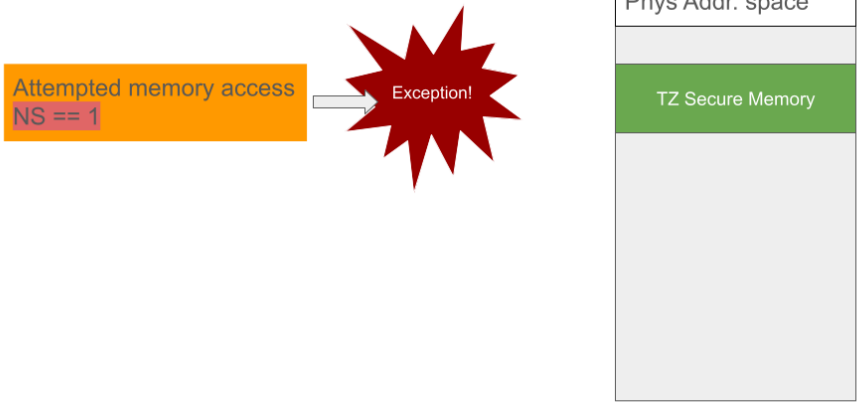
- Secure world backed by secure memory region
- MMU governs access based on Non-secure bit state in CPU



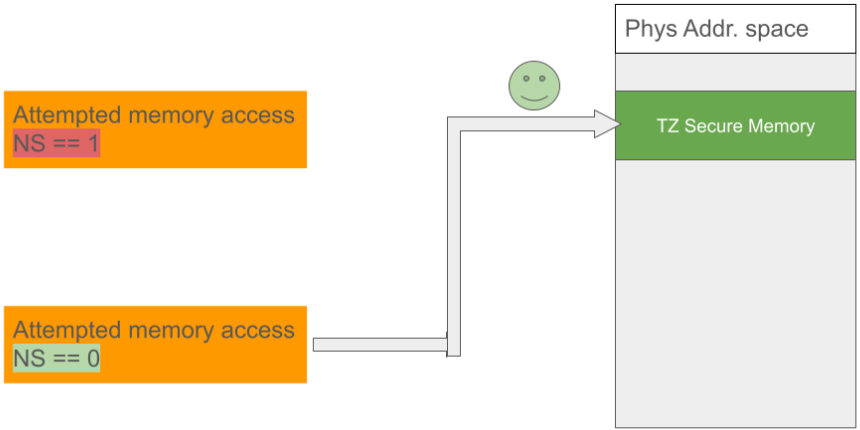
https://genode.org/documentation/articles/trustzone

Hardware-enforced isolation of two worlds
that can run on a shared processor with shared underlying resources

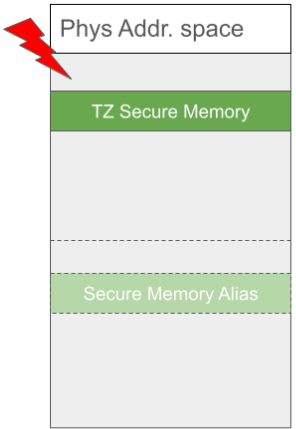
TrustZone in practice



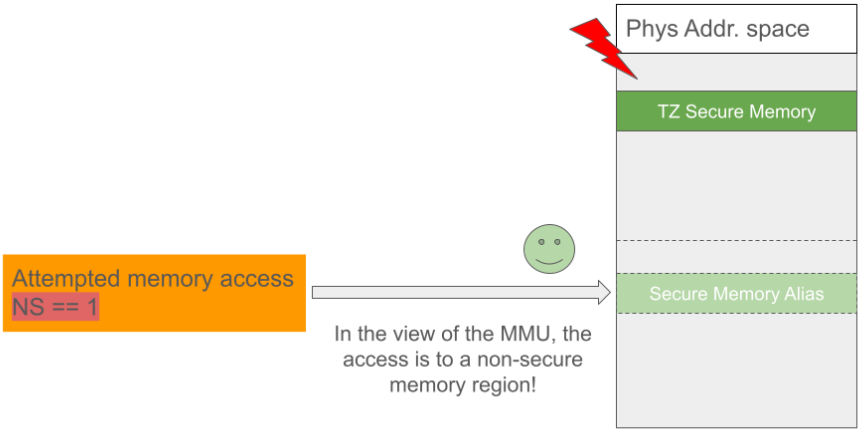
TrustZone in practice



TrustZone Attack!



TrustZone Attack!



One Step Further...

Assuming root/kernel access, this is already a violation of TrustZone security

One Step Further...

Assuming root/kernel access, this is already a violation of TrustZone security

Can we do this without root?

Userspace Attack: Building Blocks

Goal: Control memory allocation to alias of our own processes page tables

- Edit our own page table to point at arbitrary memory locations
- I.E. Arbitrary read/write to all of memory

Leak & control kernel memory, other processes, and of course TZ Secure Memory

Userspace Attack: Complications

Being subject to the kernel's memory allocation processes introduces 3 headaches:

- Lack of allocation control
- Lack of allocation knowledge
- Copy-on-Write semantics

Userspace Attack: Complications

Being subject to the kernel's memory allocation processes introduces 3 headaches:

- Lack of allocation control
- Lack of allocation knowledge
- Copy-on-Write semantics

Typical solution is to use **memory massaging** to influence allocator

Userspace Attack: Complications

Memory Massaging

Typical memory massaging often relies on exhausting memory to force the allocator to give specific physical pages

Memory Massaging

Typical memory massaging often relies on exhausting memory to force the allocator to give specific physical pages

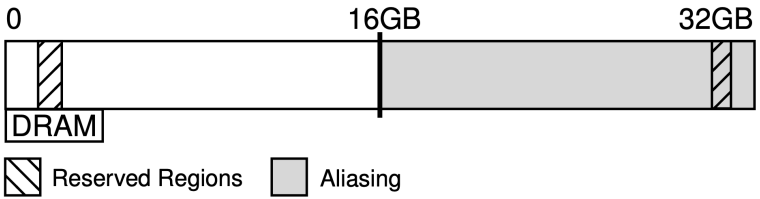


Figure 3: Contents of DRAM, with regions reserved for kernel use

Memory Massaging

Typical memory massaging often relies on exhausting memory to force the allocator to give specific physical pages

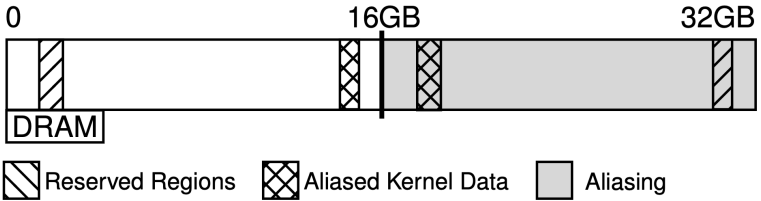


Figure 4: Contents of DRAM with aliasing; kernel regions are now aliased into the *middle* of DRAM

Memory Massaging

Typical memory massaging often relies on exhausting memory to force the allocator to give specific physical pages

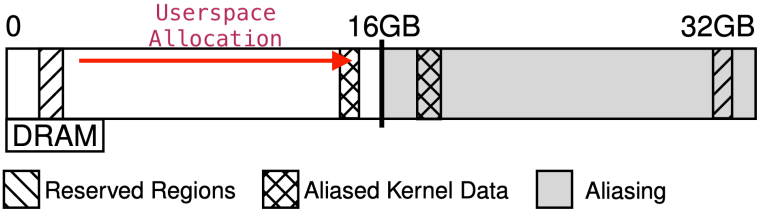


Figure 4: Contents of DRAM with aliasing; kernel regions are now aliased into the *middle* of DRAM

Memory Massaging

Typical memory massaging often relies on exhausting memory to force the allocator to give specific physical pages

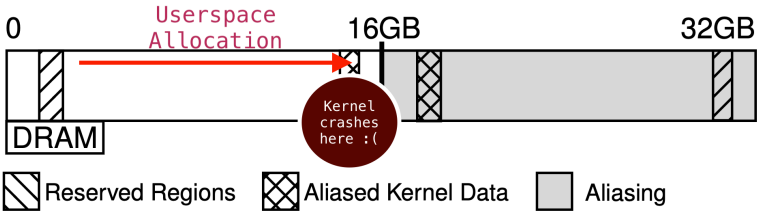


Figure 4: Contents of DRAM with aliasing; kernel regions are now aliased into the *middle* of DRAM

Memory Massaging

Typical memory massaging often relies on exhausting memory to force the allocator to give specific physical pages

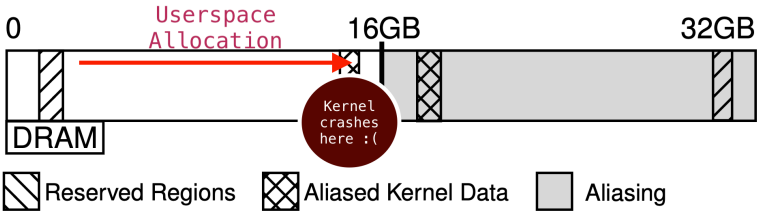


Figure 4: Contents of DRAM with aliasing; kernel regions are now aliased into the *middle* of DRAM

Need a way to 'jump' over middle of DRAM

Memory Massaging: Huge Pages

Hugepages: Linux feature to provide larger pagesizes (2MB & 1GB)

- HugeTLB reserves regions at boot for hugepage use

Hugepages: Linux feature to provide larger pagesizes (2MB & 1GB)

- HugeTLB reserves regions at boot for hugepage use

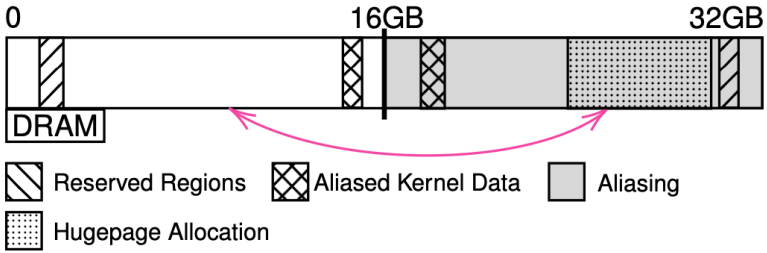


Figure 5: Contents of DRAM with `hugepages` allocated; they reside towards the top of DRAM.

Hugepages: Linux feature to provide larger pagesizes (2MB & 1GB)

- HugeTLB reserves regions at boot for hugepage use

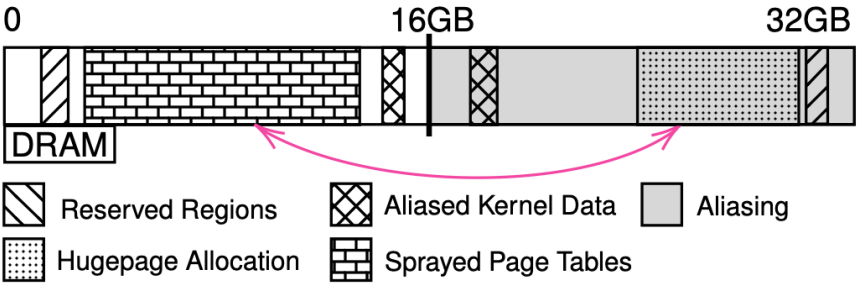


Figure 6: Contents of DRAM with `hugepages` and spraying; aliases now exist between sprayed page tables and the hugepage allocation.

Userspace Attack

1. Allocate hugepages (`mmap` w/ `MAP_HUGETLB`), write virtual address to each allocation

Userspace Attack

1. Allocate hugepages (`mmap` w/ `MAP_HUGETLB`), write virtual address to each allocation
2. Spray process page tables into memory (`mmap` w/ `MAP_SHARED`)

Userspace Attack

1. Allocate hugepages (`mmap` w/ `MAP_HUGETLB`), write virtual address to each allocation
2. Spray process page tables into memory (`mmap` w/ `MAP_SHARED`)
3. Alias Detection: Scan hugepages allocations for changes

Userspace Attack

1. Allocate hugepages (`mmap` w/ `MAP_HUGETLB`), write virtual address to each allocation
2. Spray process page tables into memory (`mmap` w/ `MAP_SHARED`)
3. Alias Detection: Scan hugepages allocations for changes
4. Modify page tables: remap memory as you like!

Evaluation

Out of our 3 eval platforms

- TrustZone attack successful on two
  - Leak key material from OP-TEE

Evaluation

Out of our 3 eval platforms

- TrustZone attack successful on two
  - Leak key material from OP-TEE
- Linux attack successful on all three
  - Used for `modprobe_path` exploit against Linux kernel

Countermeasures

- Detecting Aliasing

[1] https://batteringram.eu/
https://wiretap.fail/

Countermeasures

- Detecting Aliasing
  - Scan at boot/firmware/kernel bringup (our mitigation)
  - Won't detect dynamic aliasing (interposers!)[1]

[1] https://batteringram.eu/
https://wiretap.fail/

Countermeasures

- Detecting Aliasing
  - Scan at boot/firmware/kernel bringup (our mitigation)
  - Won't detect dynamic aliasing (interposers!)[1]
- Hardening page allocation
  - `CONFIG_SHUFFLE_PAGE_ALLOCATOR`
- Physical KASLR: Makes it harder, not impossible

[1] https://batteringram.eu/
https://wiretap.fail/

Think twice before trusting your DRAM!

Thank you!

Think twice before trusting your DRAM!
Thank you!

More on aliasing: USENIX'26 Download More RAM: Dismantling Windows Operating System Defences with Mischievous Memory

More bits..

- More stability
- Other platforms?
  - Laptop/portable PCs
  - Mobile
- Food for thought: other memory massaging techniques
- Arm Disclosure: 'TF-A only runs from ROM or on-chip trusted SRAM' 

Context: BadRAM Attack Primitive

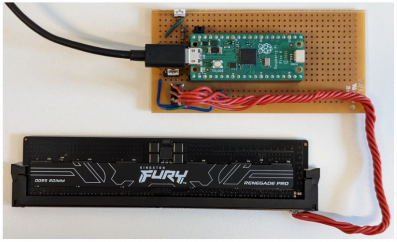


Figure 2. Raspberry Pi Pico setup to unlock and modify DDR4 and DDR5 SPDs.

SPD data encodes memory timings, serial, no. of banks and rows

- Connect Pi Pico to exposed interface
- Edit SPD data: +1 row address bit → +1 DRAM Address bit → 'doubles' capacity