pinauten
security research

Hello!

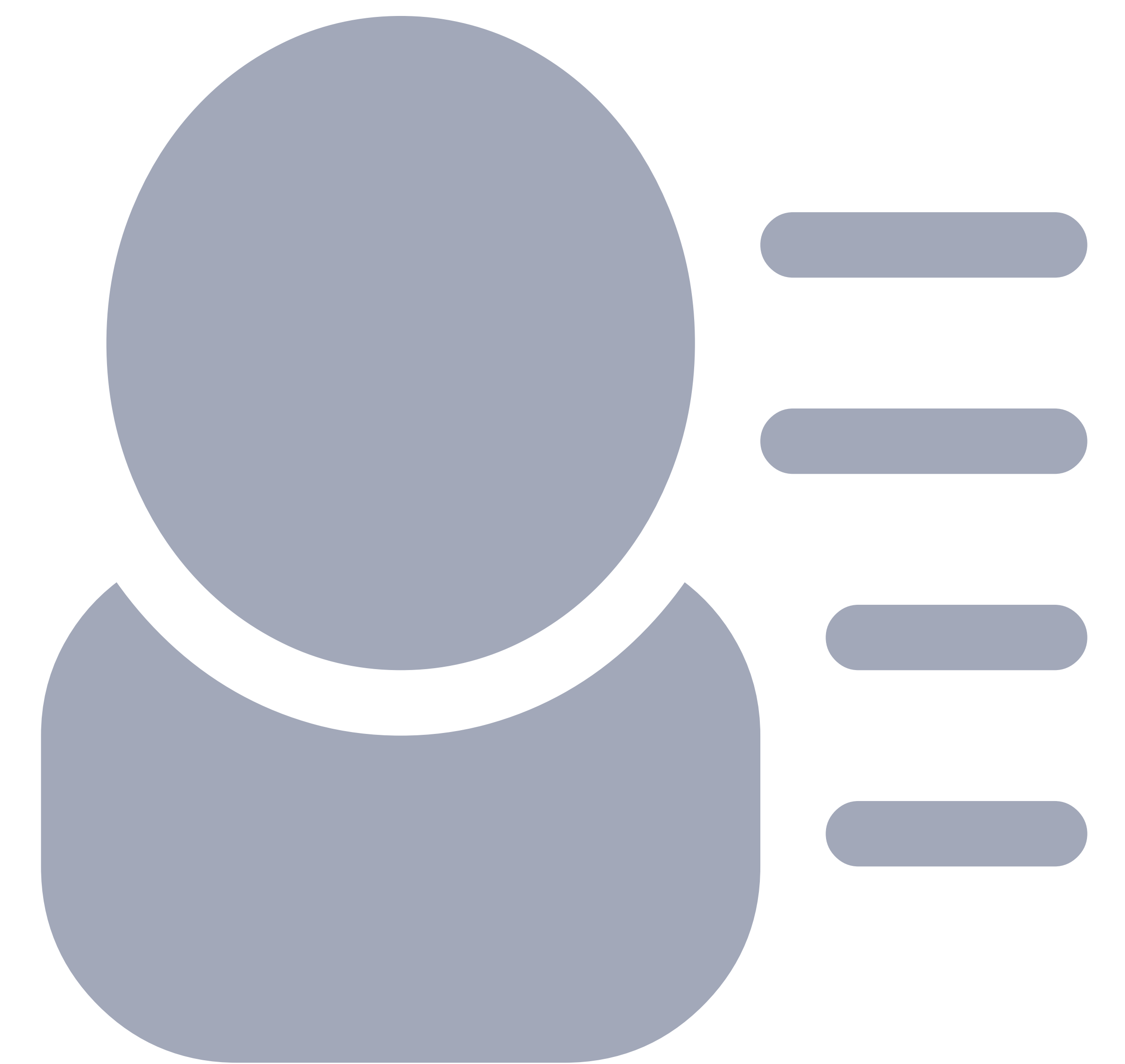
Linus Henze

Fugu15

www.pinauten.de

About me

- Linus Henze (@LinusHenze)
- CEO of Pinauten GmbH (an iOS and macOS security research company)
- CS student at Universität Koblenz
- Website: pinauten.de
- Exploits can be found on GitHub: github.com/LinusHenze and github.com/pinauten



About me

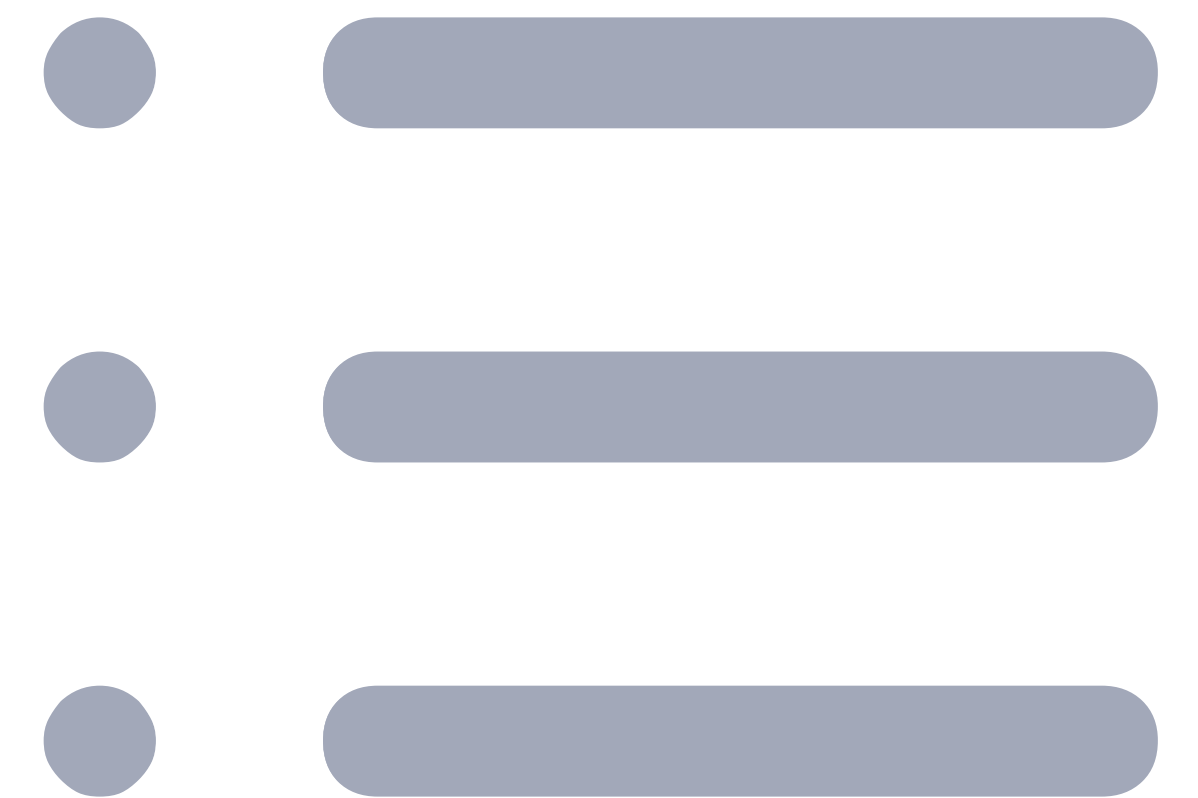
What is Fugu15?

- iOS 15 jailbreak up to 15.4.1 (+ some 15.5 betas)
    - Supports A12+ (support for other devices can be added)
- Code signature bypass via TrustCache injection
- No support for tweaks

Fugu15

Agenda

- fastPath (code signing bypass)
- oobPCI (arbitrary kernel r/w)
- badRecovery (CFI/PAC bypass)
- tlbFail (PPL bypass)
- Fugu15 Demo



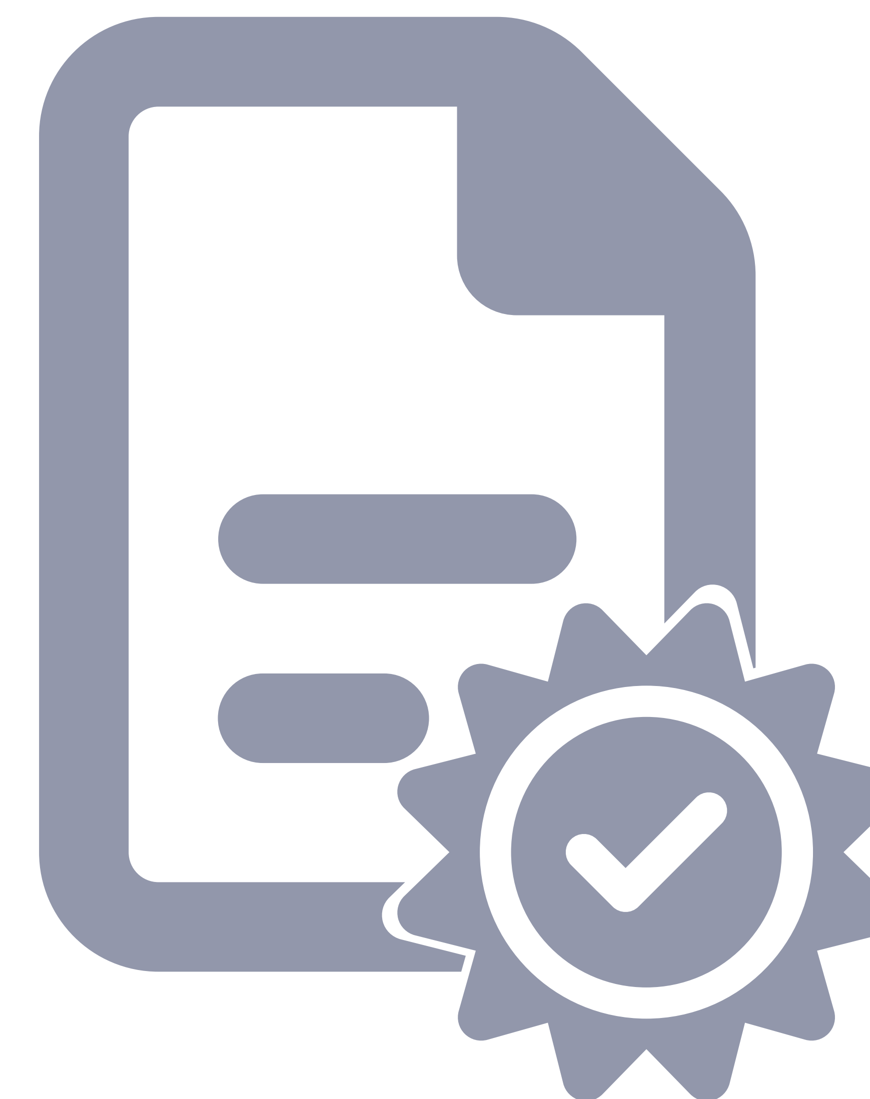
Agenda

fastPath

CoreTrust Certificate Validation Vulnerability (CVE-2022-26766)

Code Signing - Recap

- All executables on iOS must have a valid code signature
  - Enforced in the kernel by Apple Mobile File Integrity (AMFI)
- Two types of code signatures
  - Ad-Hoc: Self-Signed (preinstalled binaries, signature hash in TrustCache)
  - CMS: Signed using a certificate
- Code signatures can contain entitlements (a list of additional permissions)



Code Signing

CoreTrust

- AMFI delegates CMS validation to CT (via CTEvaluateAMFICodeSignatureCMS)
- CT ensures CMS blob is valid and the hash inside it matches the code signature hash
- Returns flags to AMFI to indicate certificate type (e.g. developer cert, AppStore cert, ...)
- CT returns success <u>even if the signer is not trusted</u> (not a vulnerability)

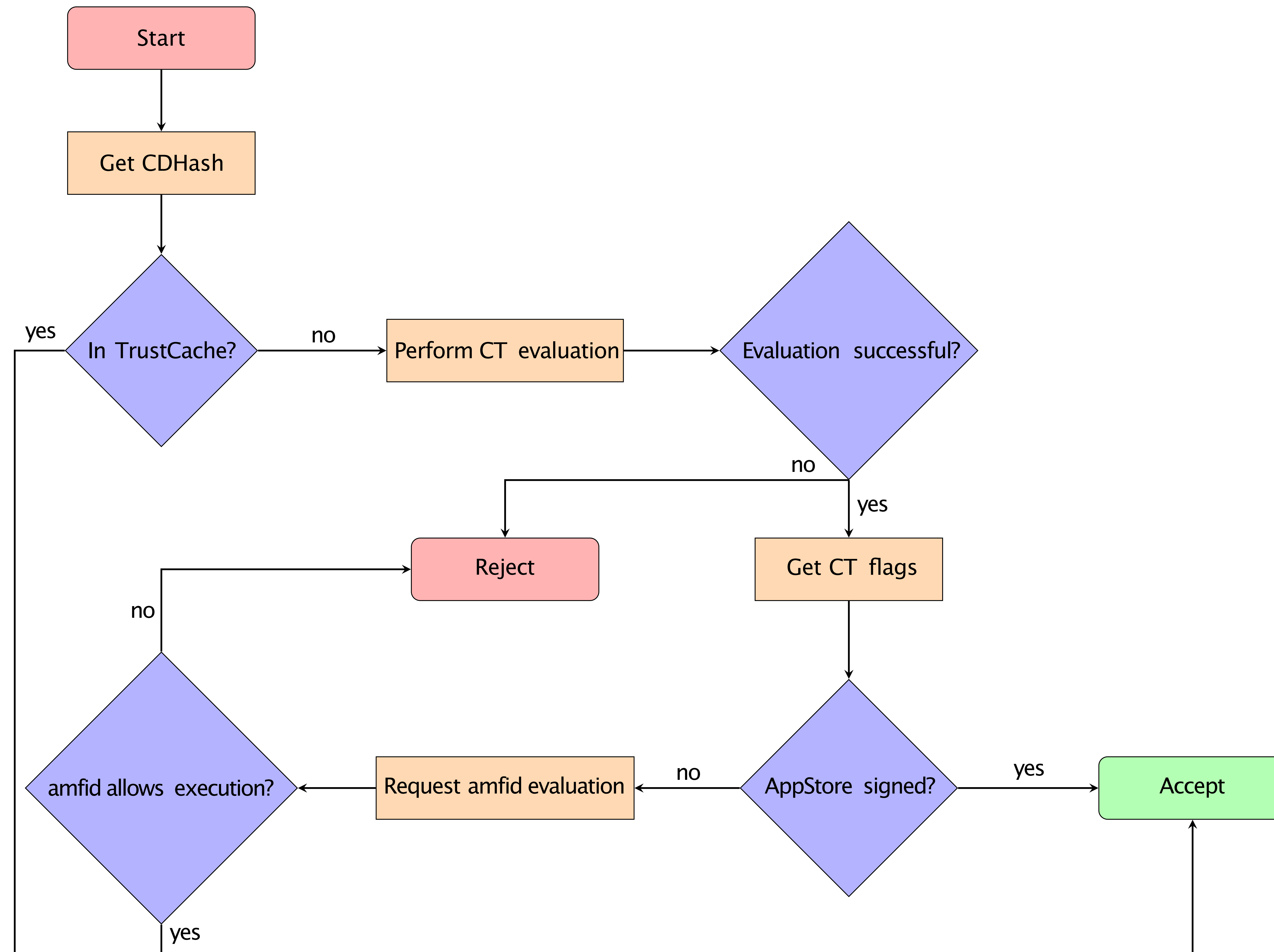
CoreTrust

Apple Mobile File Integrity

- Rejects code signature if CT validation fails
- If it succeeds, checks CT flags
  - Flags indicate binary was signed by an AppStore certificate -> Accept code signature
- Otherwise, amfid (a userspace daemon) will be asked to validate the signature

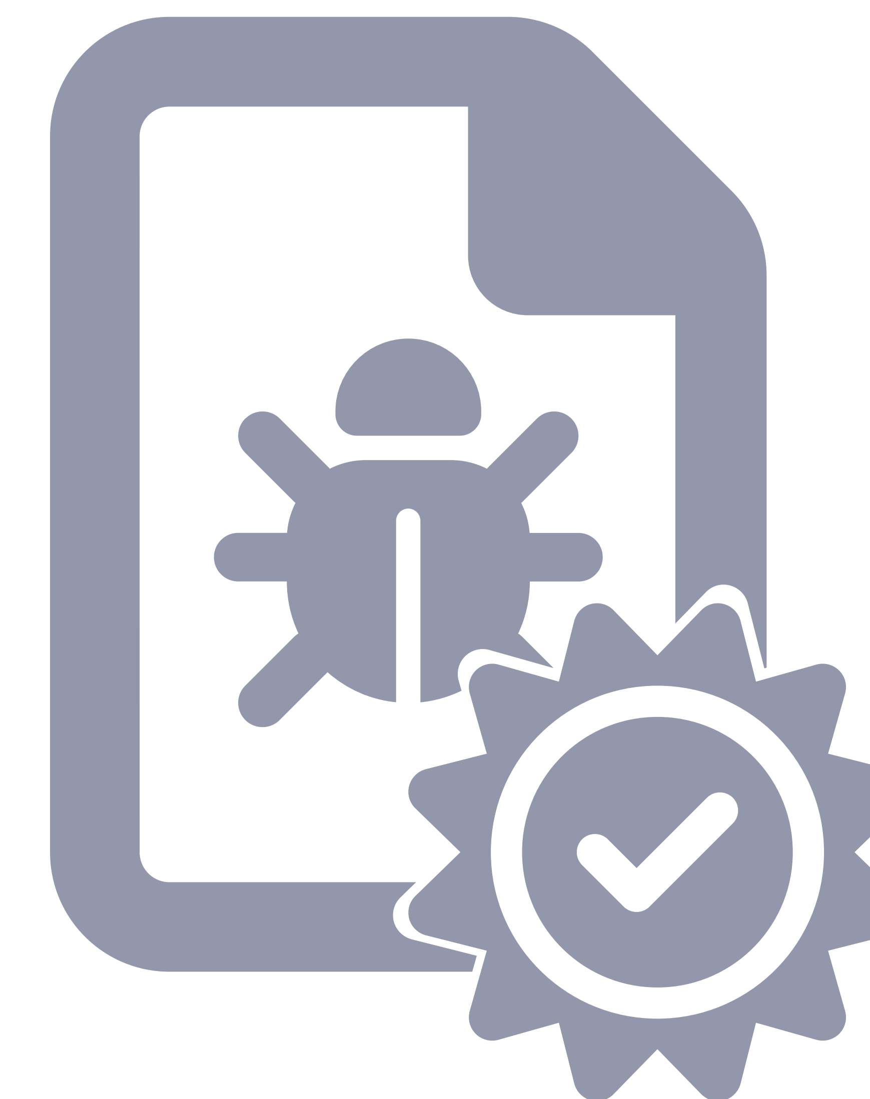
AMFI

AMFI Flow Chart (simplified)



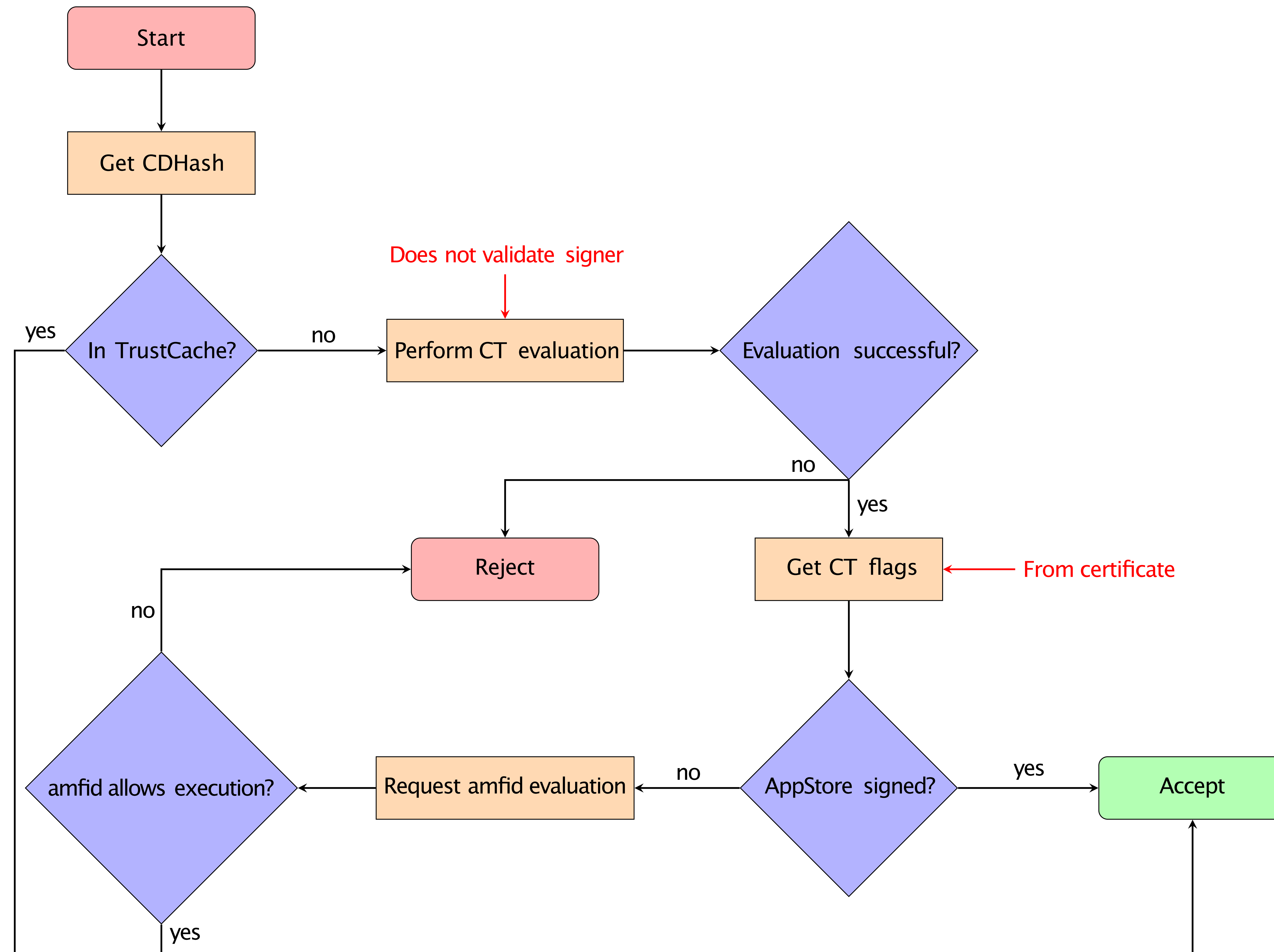
Vulnerability

- CT flags are taken directly from the certificate
  - If certain OIDs are present in the certificate, flags will be set
- Include AppStore OID in certificate
  - CT will return "signed using AppStore certificate" in the CT flags
  - AMFI performs no further validation when signed using an AppStore certificate
- Remember: CT does not verify the cert issuer



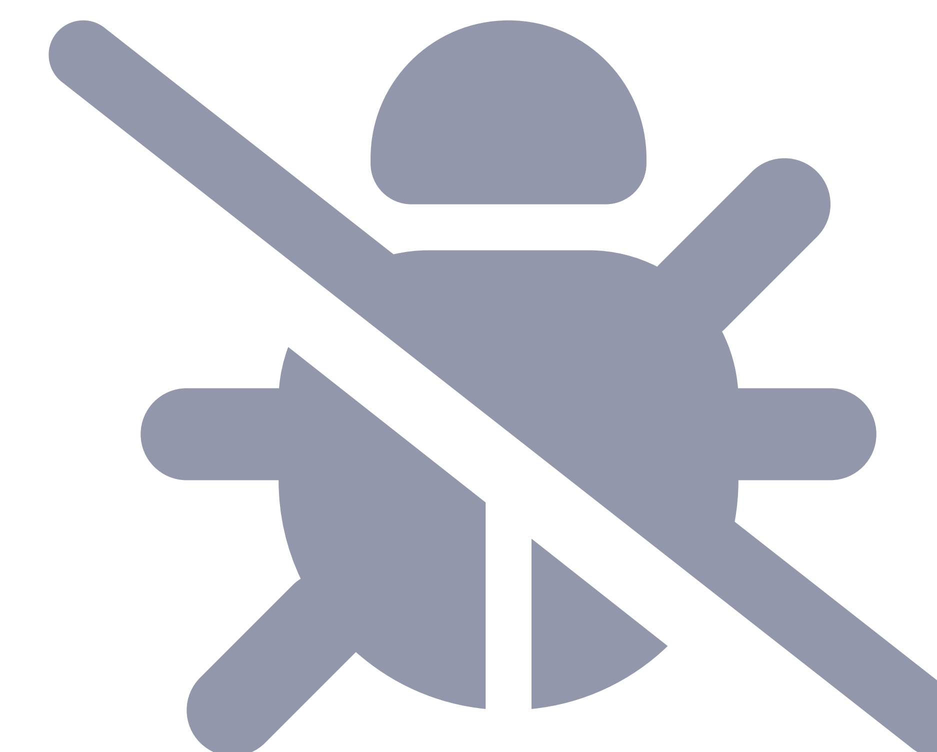
Vulnerability

AMFI Flow Chart (simplified)



Apple's fix

- CT now only returns flags if the certificate was issued by Apple
  - CT still returns success even if the issuer is untrusted but flags will remain 0 (macOS compatibility)
- This vulnerability was a regression, iOS 13 and below are not affected

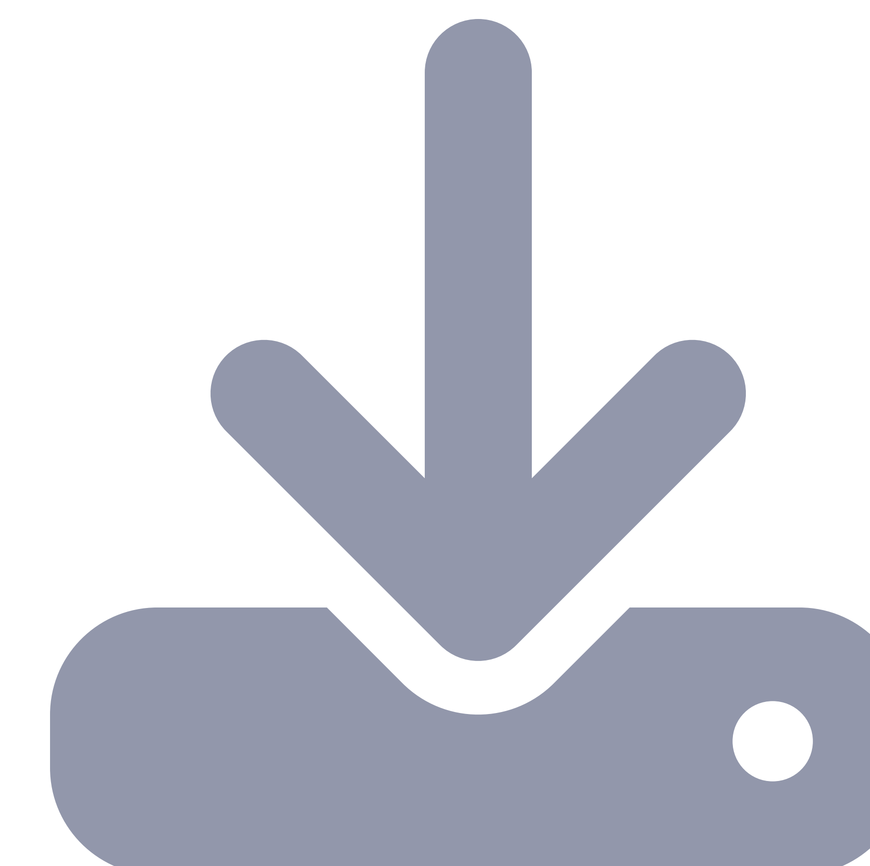


Fix

Installing a fastPath signed App

installd

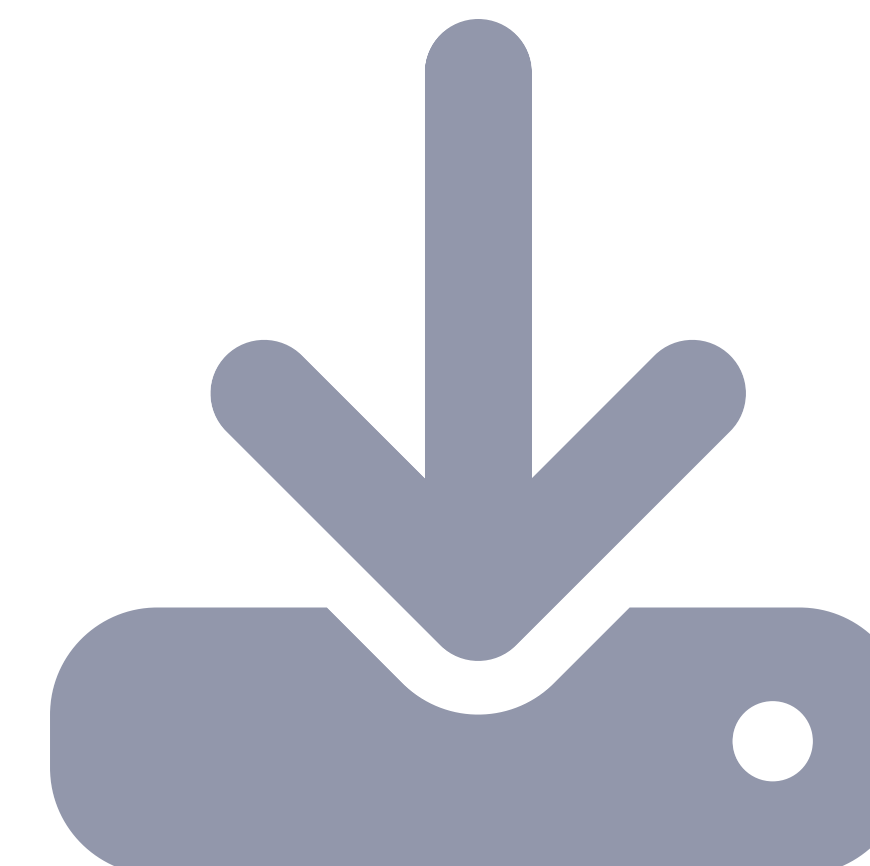
- Responsible for App installation
- Checks code signature of every App before installing it
- Not affected by the fastPath vulnerability
  - Another vulnerability is needed



installd

installd

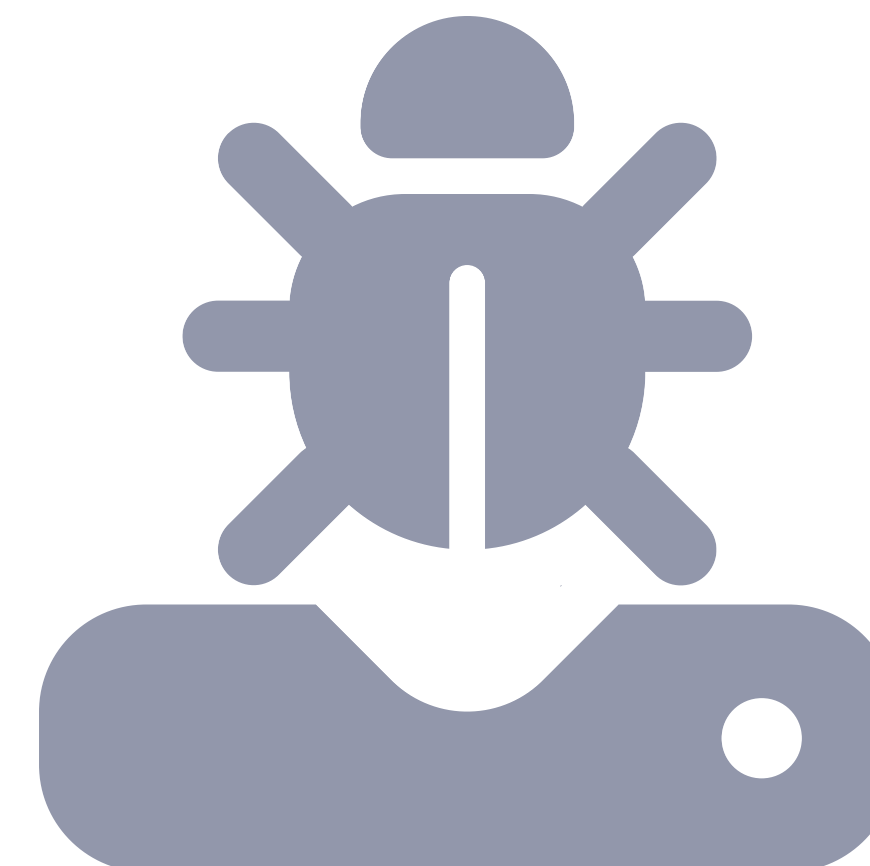
- Responsible for App installation
- Checks code signature of every App before installing it
- Not affected by the fastPath vulnerability
  - Another vulnerability is needed
    - Exploit CVE-2021-30773 again...
    - (Patch is incomplete)



installd

installHaxx (CVE-2021-30773)

- Part of Fugu14
  - Works out-of-the-box on iOS 15, with modifications even on non-PAC devices
- installd only verifies the code signature of the "best" slice in a FAT binary
  - Create a FAT binary containing a validly signed binary and our binary
  - Ensure installd verifies the validly signed binary but kernel executes our binary



installd

oobPCI

PCI DriverKit out-of-bounds memory access
(CVE-2022-26763)

"DriverKit provides a fully modernized replacement for IOKit to create device drivers"

- Apple

Source: https://developer.apple.com/system-extensions/

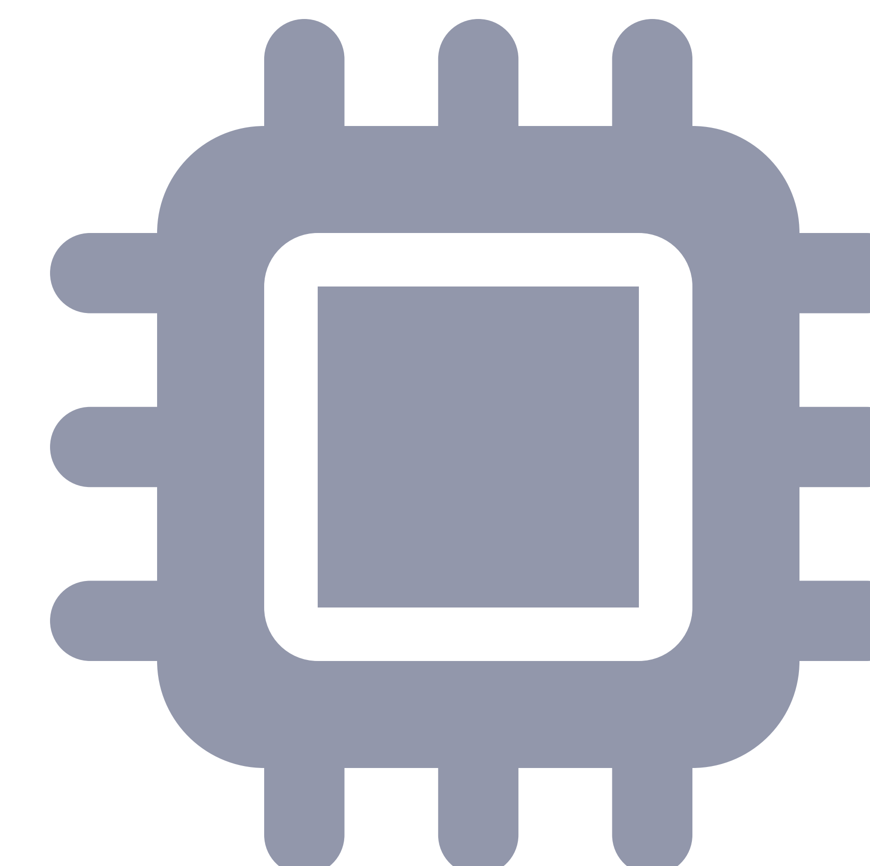
"System extensions and drivers built with DriverKit run in user space, where they can't compromise the security or stability of macOS"

\- Apple

Source: https://developer.apple.com/system-extensions/

DriverKit - Recap

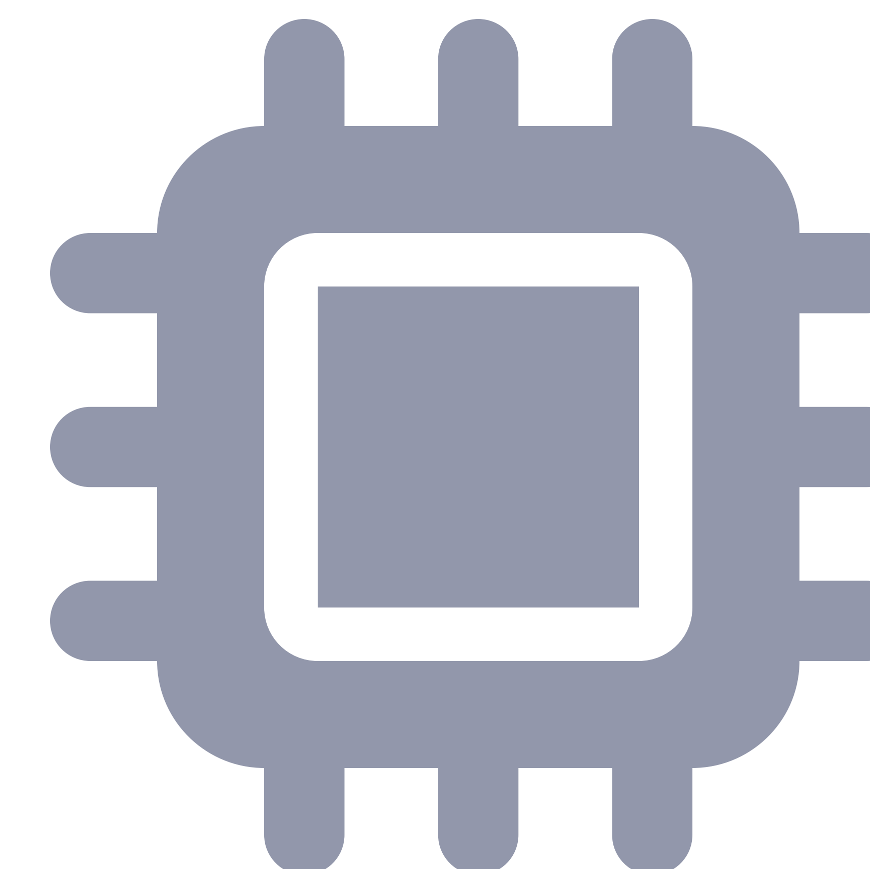
- Framework for running Drivers in userspace
- Allows low-level access to devices
- Exposes various methods to userspace that were never intended to be used by untrusted code...
  - But: Drivers must have DriverKit entitlements to use DriverKit



DriverKit

PCI DriverKit

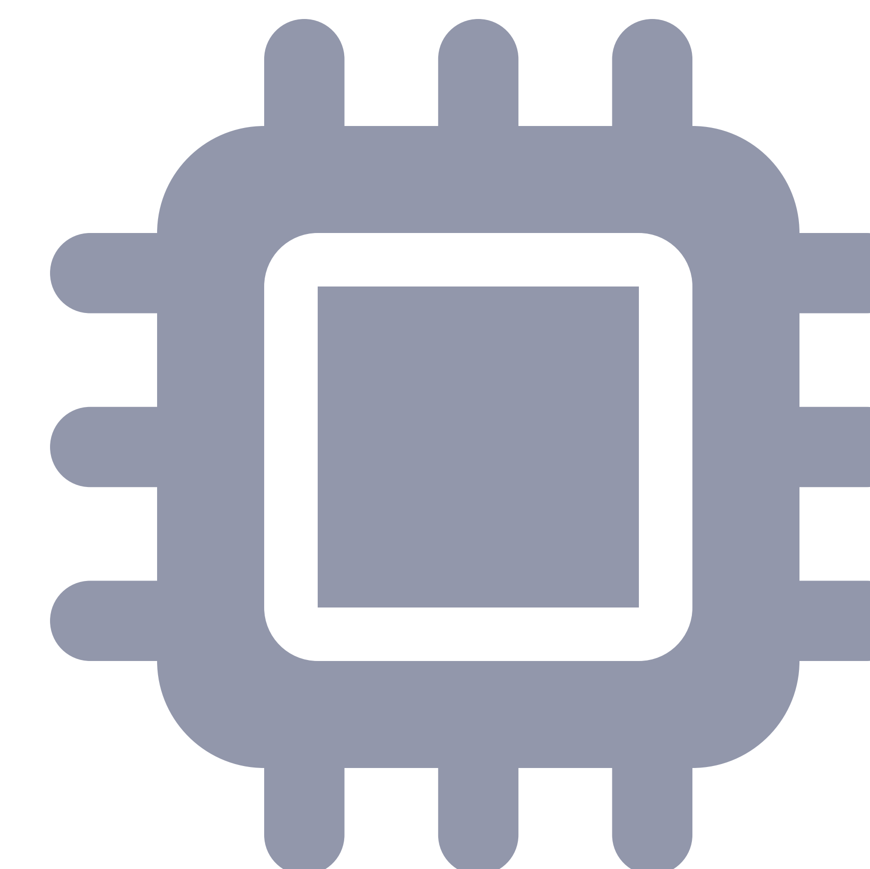
- Allows low-level access to PCI devices
  - Directly write to device memory
- Exposes various methods to read/write memory
  - Methods take a memory index, offset and data parameter



PCI DriverKit

PCI DriverKit

- Allows low-level access to PCI devices
  - Directly write to device memory
- Exposes various methods to read/write memory
  - Methods take a memory index, offset and data parameter



PCI DriverKit

Vulnerable Code (Example)

```
IOReturn IOPCIDevice::deviceMemoryWrite64(uint8_t  memoryIndex,
                                          uint64_t offset,
                                          uint64_t data)
{
    IOReturn result = kIOReturnUnsupported;

    IOMemoryMap* deviceMemoryMap = reserved->deviceMemoryMap[memoryIndex];
    if(deviceMemoryMap != NULL)
    {
        ml_io_write(deviceMemoryMap->getVirtualAddress() + offset, data, sizeof(uint64_t));
        result = kIOReturnSuccess;
    }
    else
    {
        DLOG("IOPCIDevice::deviceMemoryRead64: index %u could not get mapping\n", memoryIndex);
        return kIOReturnNoMemory;
    }

    return result;
}
```

Source: https://github.com/apple-oss-distributions/IOPCIFamily/blob/2c6dd94c1f4bbbe2300eca914c1da5fa6f66ca2e/IOPCIDevice.cpp#L1906

Vulnerable Code (Example)

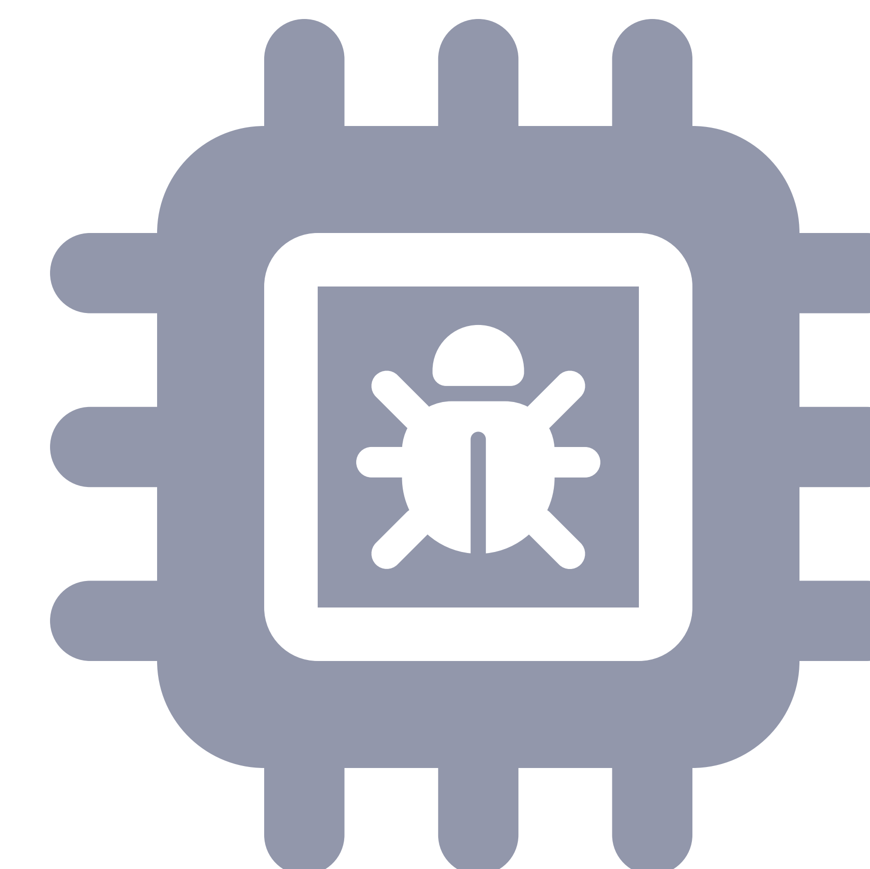
```
IOReturn IOPCIDevice::deviceMemoryWrite64(uint8_t  memoryIndex, // Checked by _MemoryAccess
                                          uint64_t offset, // Attacker controlled, unchecked
                                          uint64_t data)
{
    IOReturn result = kIOReturnUnsupported;

    IOMemoryMap* deviceMemoryMap = reserved->deviceMemoryMap[memoryIndex];
    if(deviceMemoryMap != NULL)
    {
        ml_io_write(deviceMemoryMap->getVirtualAddress() + offset, data, sizeof(uint64_t));
        result = kIOReturnSuccess;
    }
    else
    {
        DLOG("IOPCIDevice::deviceMemoryRead64: index %u could not get mapping\n", memoryIndex);
        return kIOReturnNoMemory;
    }

    return result;
}
```

Source: https://github.com/apple-oss-distributions/IOPCIFamily/blob/2c6dd94c1f4bbbe2300eca914c1da5fa6f66ca2e/IOPCIDevice.cpp#L1906

How can this vulnerability be exploited?

Exploit Strategy #1

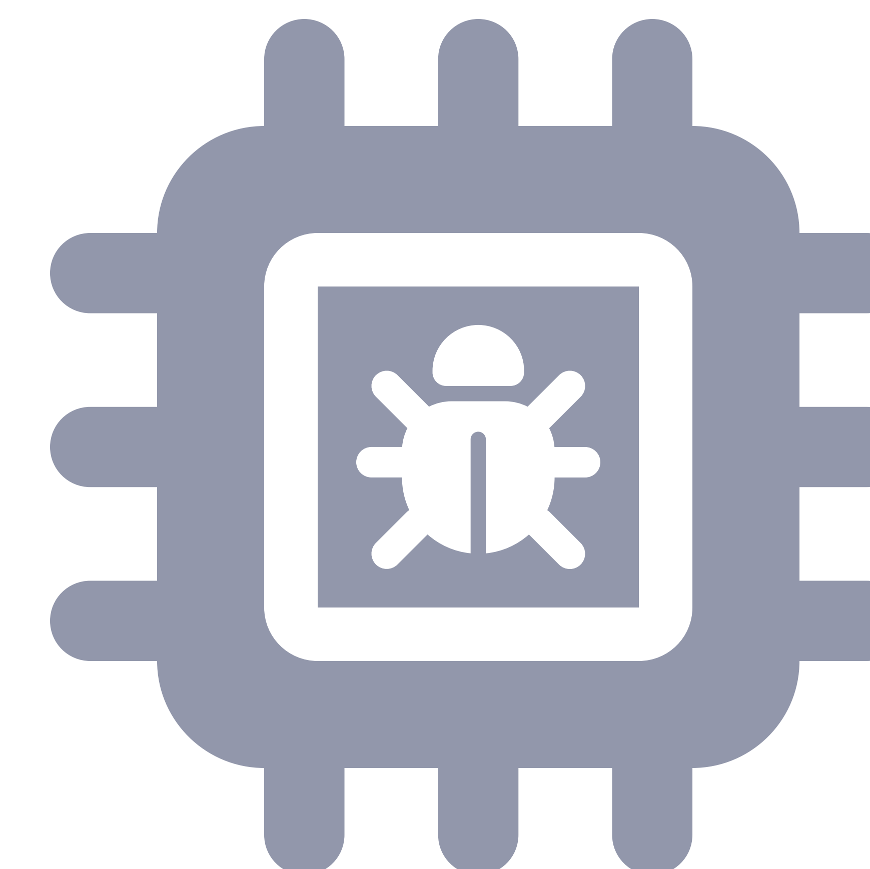
- Spray kernel memory and use the out-of-bounds access to leak/modify kernel pointers
- Problem:
  - PCI Device Memory is part of the kernel_data_map
  - kernel_data_map contains no pointers...



Strategy #1

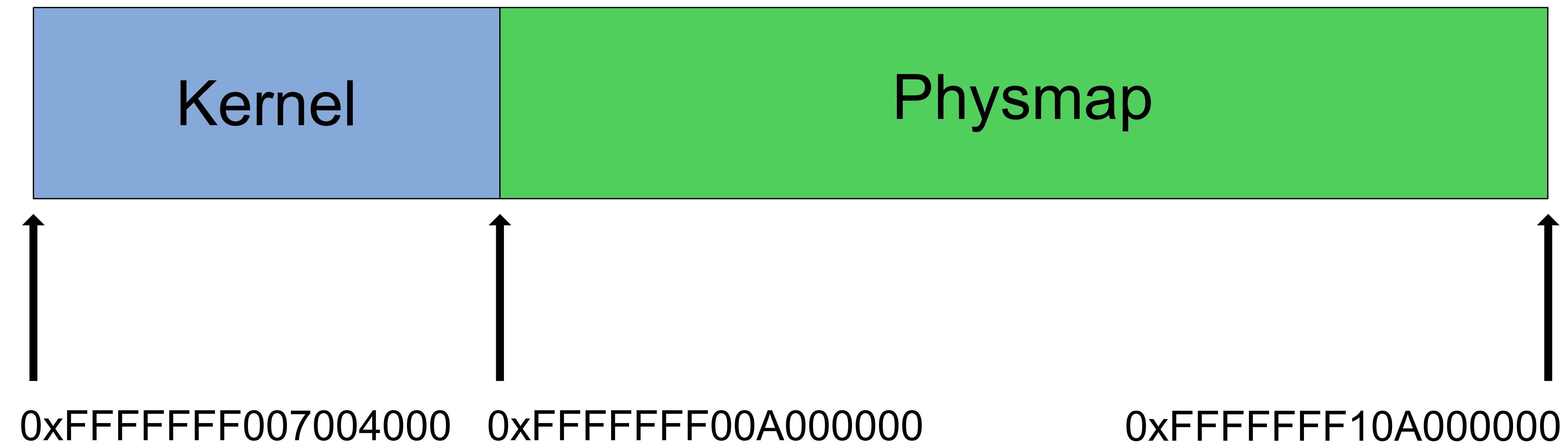
Exploit Strategy #2

- PCI Memory address is deterministic (+/- 128 MB)
- All RAM is mapped as part of the (contiguous) physmap
- Physmap is located at a random offset after the kernel (max 1GB)
- Kernel load address randomization is max 1GB
- 1GB + 1GB = 2GB randomization, modern iPhones have at least 4GB RAM



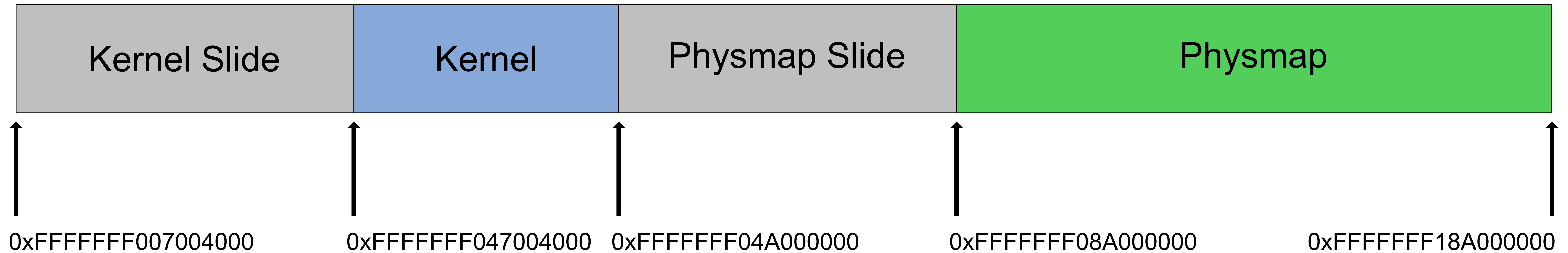
Strategy #2

Memory Layout - No Randomization



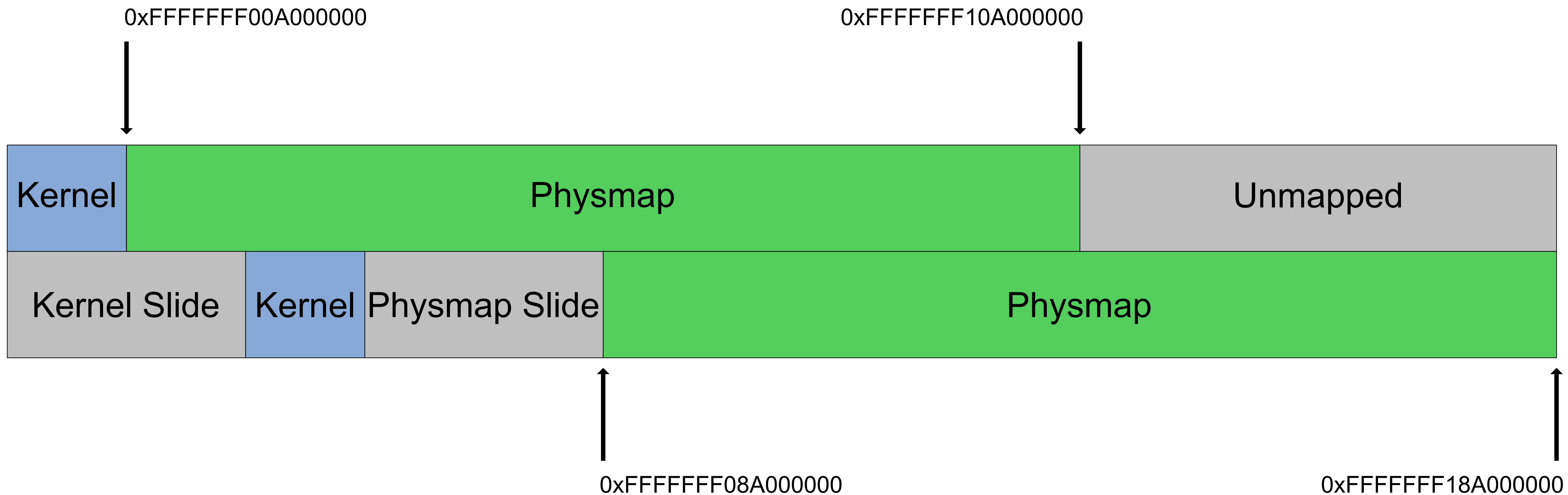
*Not to scale, addresses may differ

Memory Layout - Maximum Randomization



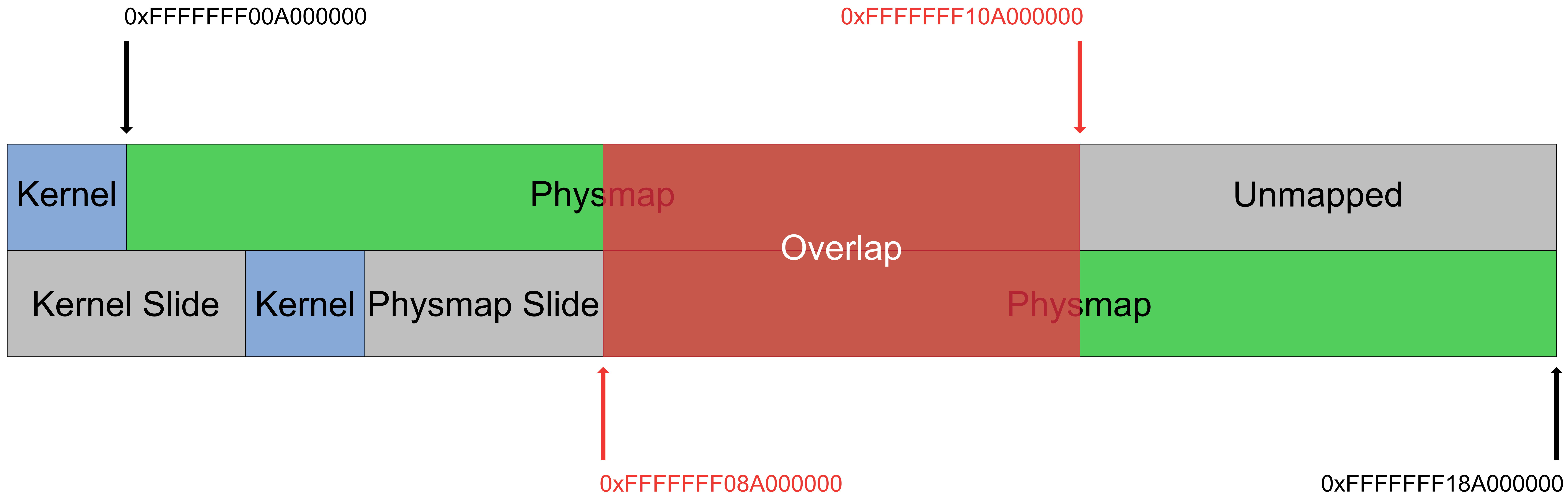
*Not to scale, addresses may differ

Memory Layout - Min/Max Randomization



*Not to scale, addresses may differ

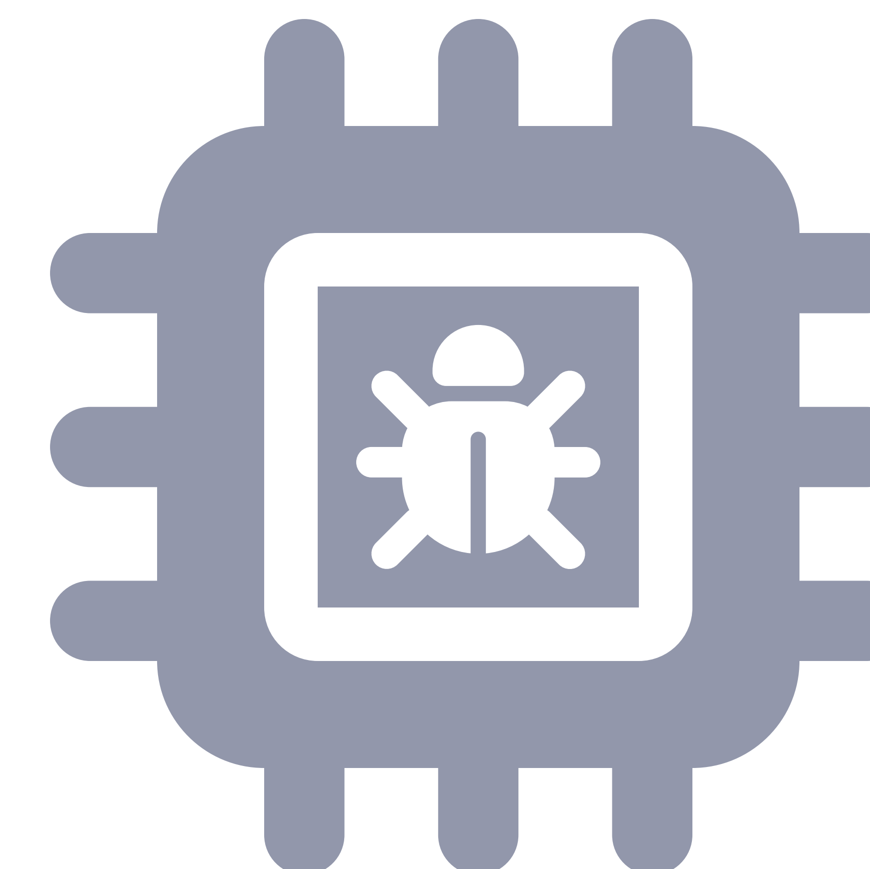
Memory Layout - Overlap



*Not to scale, addresses may differ

Exploit Strategy #2

- Guess offset to physmap, find the boot-args region in the physmap (it's at the start)
- Scan the initial page tables (directly after boot-args) to determine exact offset of boot-args in physmap
- Calculate offset from PCI Memory to start of physmap

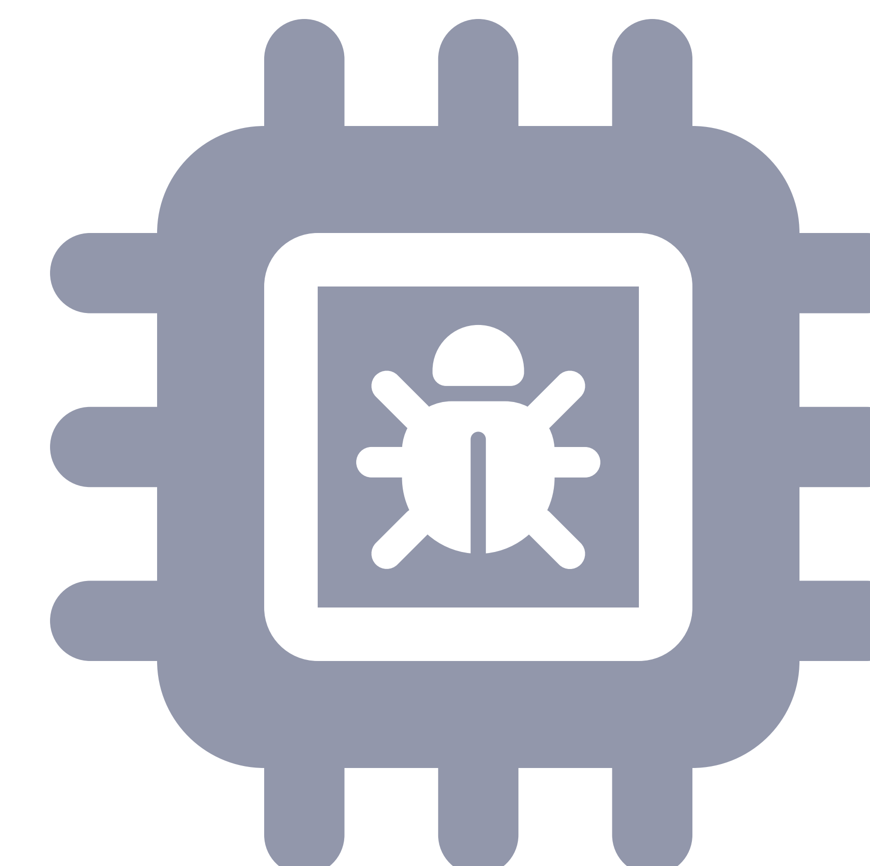


Strategy #2

pinauten security research

Exploit Strategy #2

- Physmap is located at a L2 boundary, calculate low 25 bits of the PCI Memory region via offset to physmap
  - low25 = 0x2000000 - (physmapStartOff & 0x1FFFFFF)
- Scan entire RAM to find IOMemoryMap object corresponding to the PCI Memory (using the low 25 bits calculated previously) and use it to determine PCI Memory start address



Strategy #2

badRecovery

CFI/PAC bypass via thread fault handlers (CVE-2022-26765)

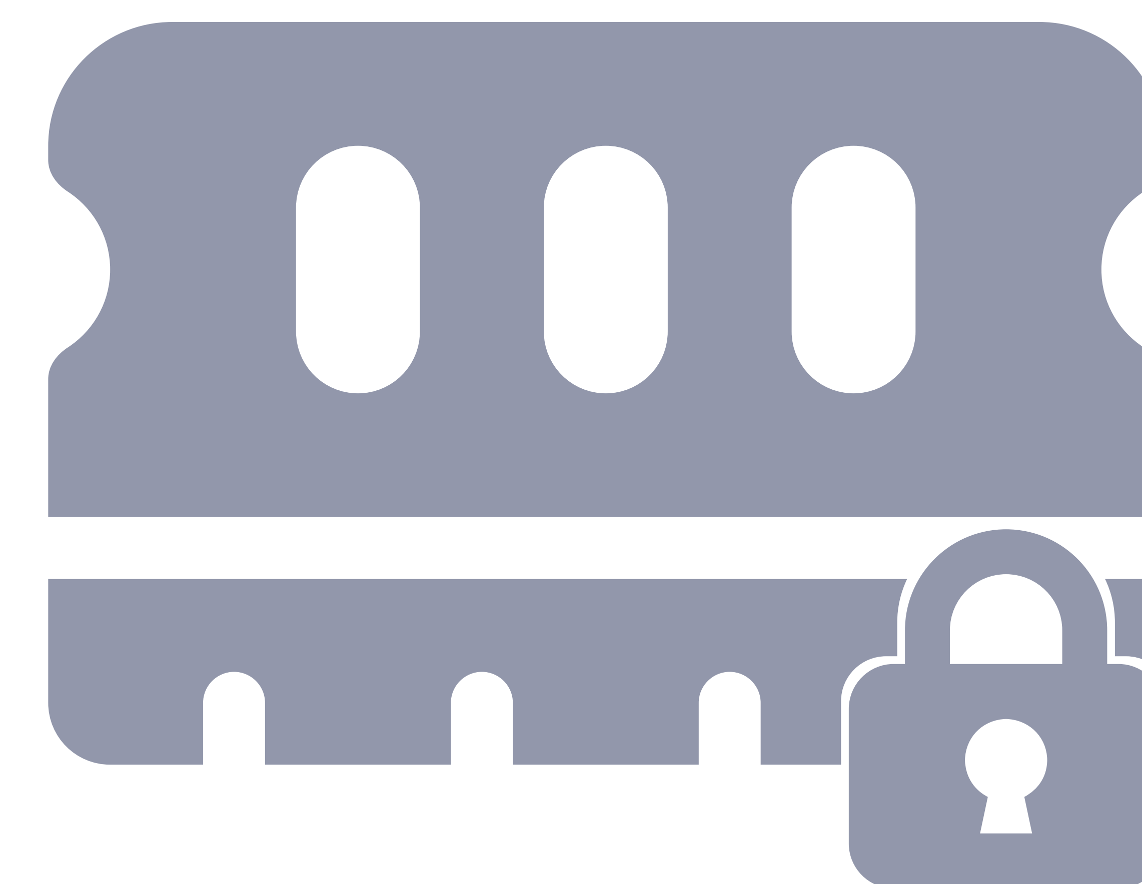
PAC - Recap

- Pointer Authentication Codes, cryptographic signature for pointers
- Prevents modification of critical pointers/data
  - e.g. return addresses on the stack
- Also provides Control Flow Integrity (CFI)

PAC

Thread Fault Handlers

- Mechanism to handle expected faults during data accesses
- Allows storing a pointer to a fault handler in the thread struct (PAC signed)
  - Kernel will jump to that handler when a data access fault occurs in the kernel



Fault Handlers

Example - Copyin

```
/*
 * int _copyin_atomic64(const char *src, uint32_t *dst)
 */
LEXT(_copyin_atomic64)
   ARM64_STACK_PROLOG // pacibsp
   PUSH_FRAME
   SET_RECOVERY_HANDLER copyio_error

   ldr    x8, [x0]
   str    x8, [x1]
   mov    x0, #0

   CLEAR_RECOVERY_HANDLER
   POP_FRAME
   ARM64_STACK_EPILOG // retab
```

Source: https://github.com/apple-oss-distributions/xnu/blob/a325d9c4a84054e40bbe985afedcb50ab80993ea/osfmk/arm64/machine_routines_asm.s#L530

Vulnerable Code

```
/*
 * hw_lck_ticket_t
 * hw_lck_ticket_reserve_orig_allow_invalid(hw_lck_ticket_t *lck)
 */
LEXT(hw_lck_ticket_reserve_orig_allow_invalid)
    SET_RECOVERY_HANDLER 9f, label_in_adr_range=1

    mov     x8, x0
    mov     w9, #HW_LCK_TICKET_LOCK_INCREMENT
1:
    ldr     w0, [x8]
2:
    tbz     w0, #HW_LCK_TICKET_LOCK_VALID_BIT, 9f /* lock valid ? */

    add     w11, w0, w9
    mov     w12, w0
    casa    w0, w11, [x8]
    cmp     w12, w0
    b.ne    2b

    CLEAR_RECOVERY_HANDLER
    ret

9: /* invalid */
    CLEAR_RECOVERY_HANDLER
    mov     w0, #0
    ret
```

Source: https://github.com/apple-oss-distributions/xnu/blob/a325d9c4a84054e40bbe985afedcb50ab80993ea/osfmk/arm64/machine_routines_asm.s#L758

Vulnerable Code

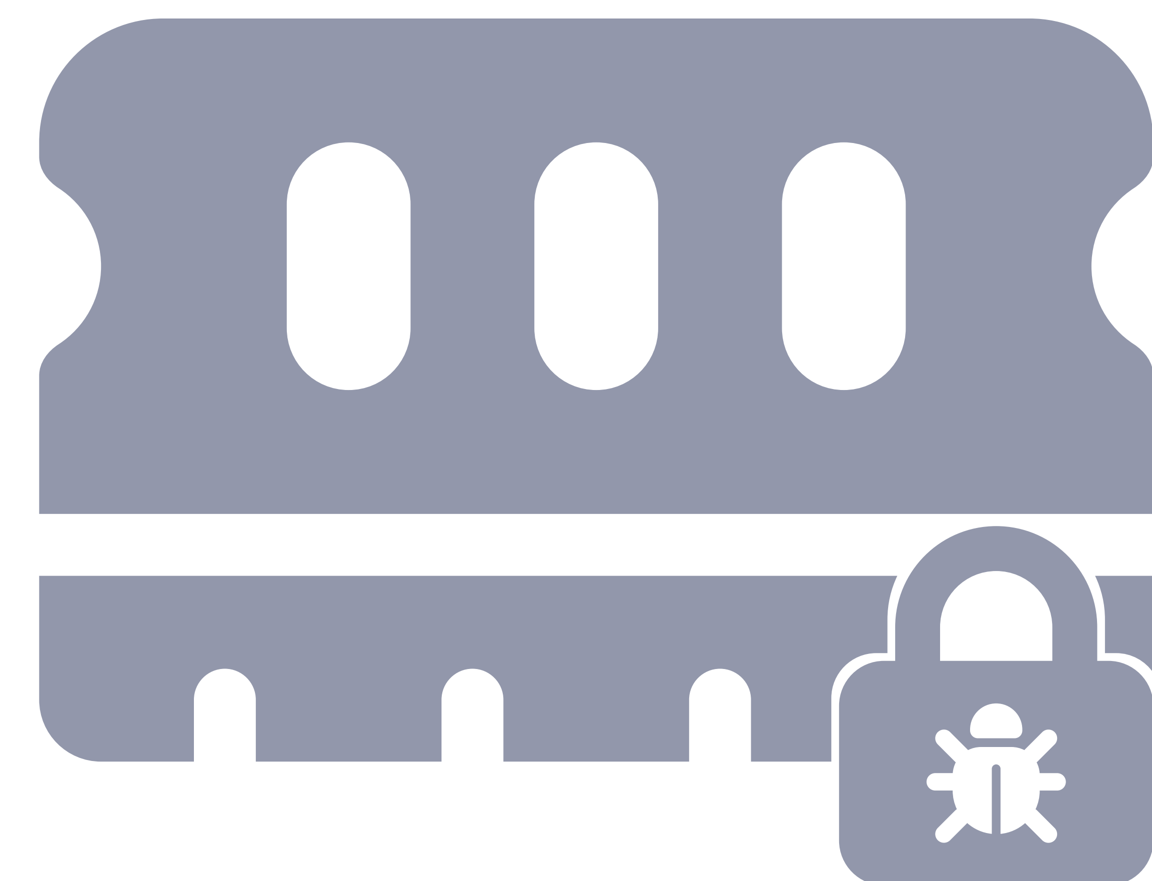
```
/*
 * hw_lck_ticket_t
 * hw_lck_ticket_reserve_orig_allow_invalid(hw_lck_ticket_t *lck)
 */
LEXT(hw_lck_ticket_reserve_orig_allow_invalid)
    SET_RECOVERY_HANDLER 9f, label_in_adr_range=1

    // Some code

9: // Failure - Recovery Handler
    CLEAR_RECOVERY_HANDLER
    mov     w0, #0
    ret
```

Source: https://github.com/apple-oss-distributions/xnu/blob/a325d9c4a84054e40bbe985afedcb50ab80993ea/osfmk/arm64/machine_routines_asm.s#L758

pinauten
security research

Strategy

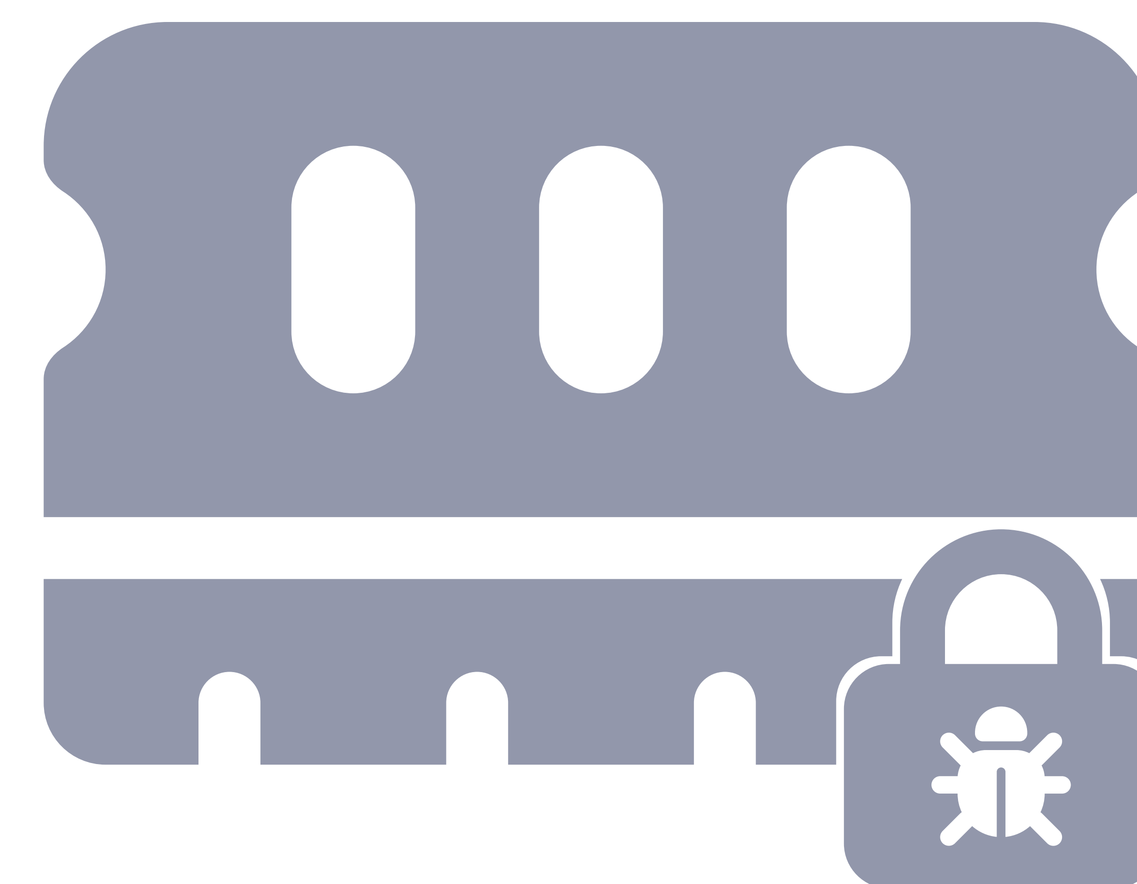
- Call hw_lck_ticket_reserve_orig_allow_invalid and steal the signed fault handler
  - e.g. via mach_port_mod_refs
- Restore the fault handler and perform a syscall
- Ensure a data abort happens shortly after entering the kernel while most registers are still user-controlled (especially lr)



Strategy

Strategy

- Kernel only accesses userspace register state before trashing registers
  - Change register state pointer to force data abort
- Problem: Kernel must be able to save sp, otherwise a panic will occur
  - Solution: Ensure the memory is mapped up to the sp register, unmapped afterwards



Strategy

pinauten
security research

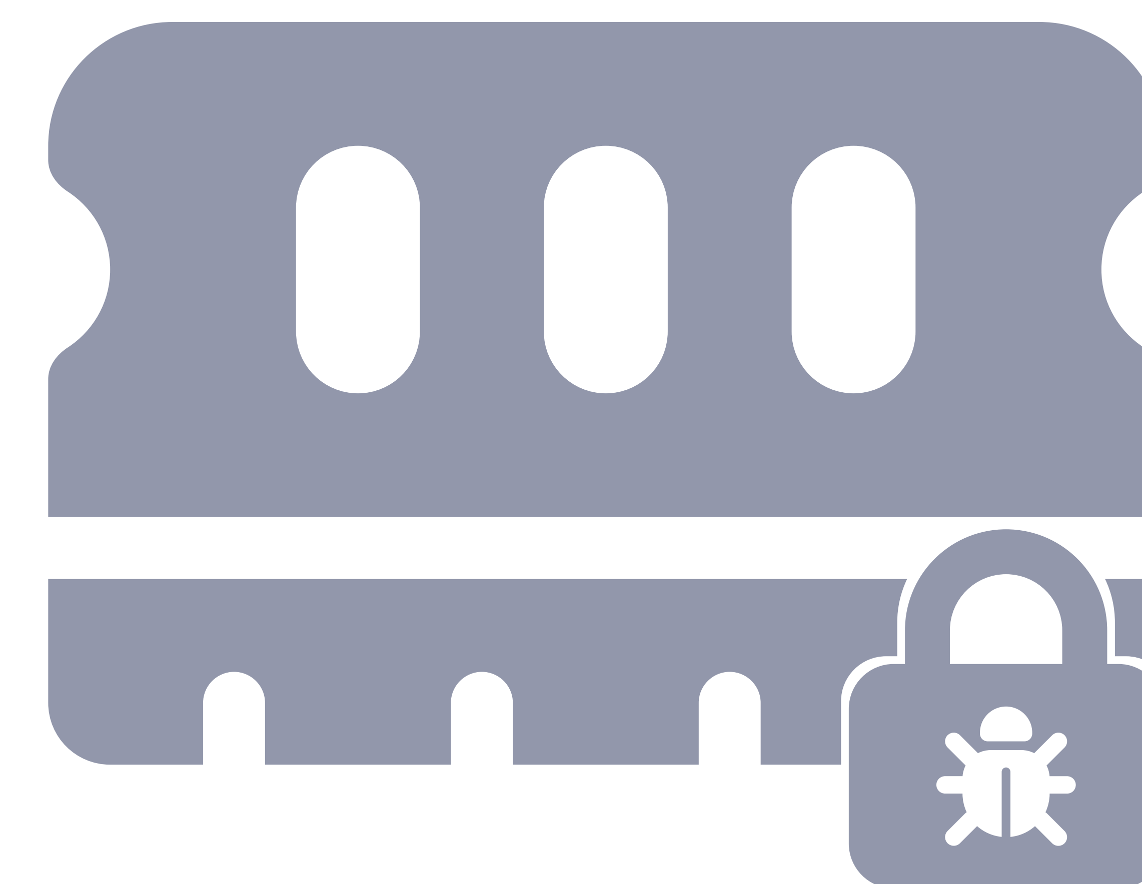
Stack Check

```
if (exceptionFrame->sp >= thread->kstackptr)
    goto istackTest;
if (exceptionFrame->sp  > (thread->kstackptr - KERNEL_STACK_SIZE))
    goto validStack;

istackTest:
if (exceptionFrame->sp >= thread->cpuData->interruptStack)
    panic("Stack corrupt!");
if (exceptionFrame->sp <= (thread->cpuData->interruptStack - INTSTACK_SIZE_NUM))
    panic("Stack corrupt!");

validStack:
// Handle exception normally
```

Pseudo-C Code, original source: https://github.com/apple-oss-distributions/xnu/blob/a325d9c4a84054e40bbe985afedcb50ab80993ea/osfmk/arm64/locore.s#L145

Bypass Stack Check

- Replace thread kernel stack pointer and cpu stack pointer
  - This will not actually change the CPU's interrupt stack
- Ensure thread kernel stack pointer and cpu stack pointer overlap



Stack Check

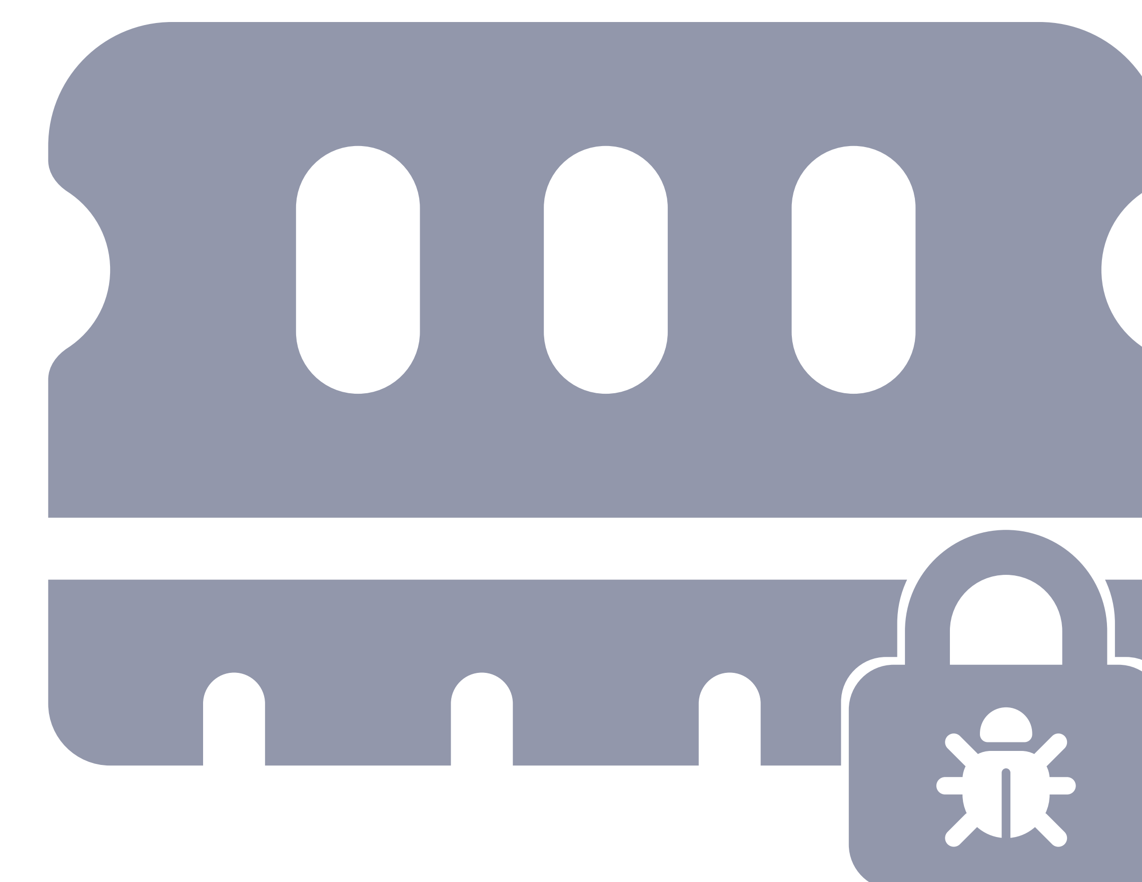
Recovery Handler

```
CLEAR_RECOVERY_HANDLER // str x10, [x16, TH_RECOVER]
mov    w0, #0
ret
```

Source: https://github.com/apple-oss-distributions/xnu/blob/a325d9c4a84054e40bbe985afedcb50ab80993ea/osfmk/arm64/machine_routines_asm.s#L790

Recovery Handler

- Invoked with all register controlled (except x0, x1 and sp)
- Performs a store of x10 to x16 + offset and clears x0
- Finally an unauthenticated return is done



Recovery Handler

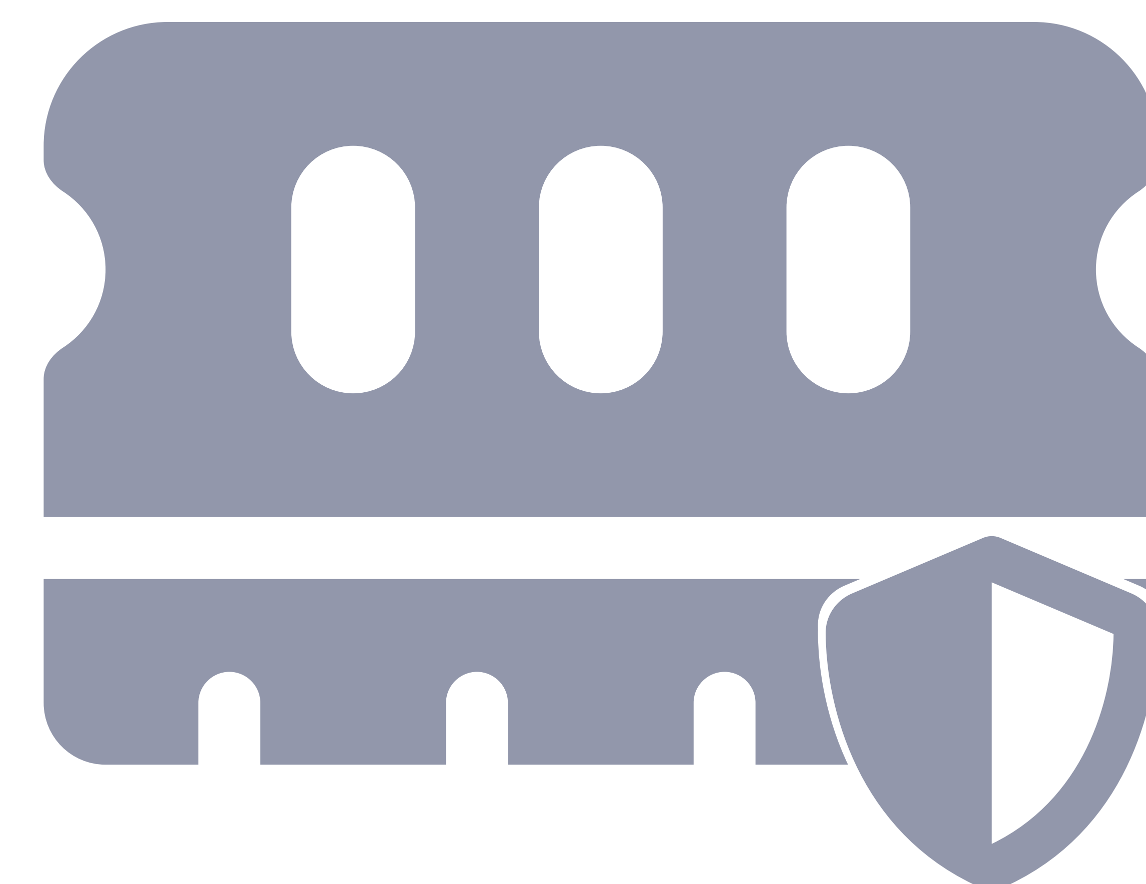
pinauten
security research

tlbFail

PPL bypass via improper TLB flush
(CVE-2022-26764)

PPL - Recap

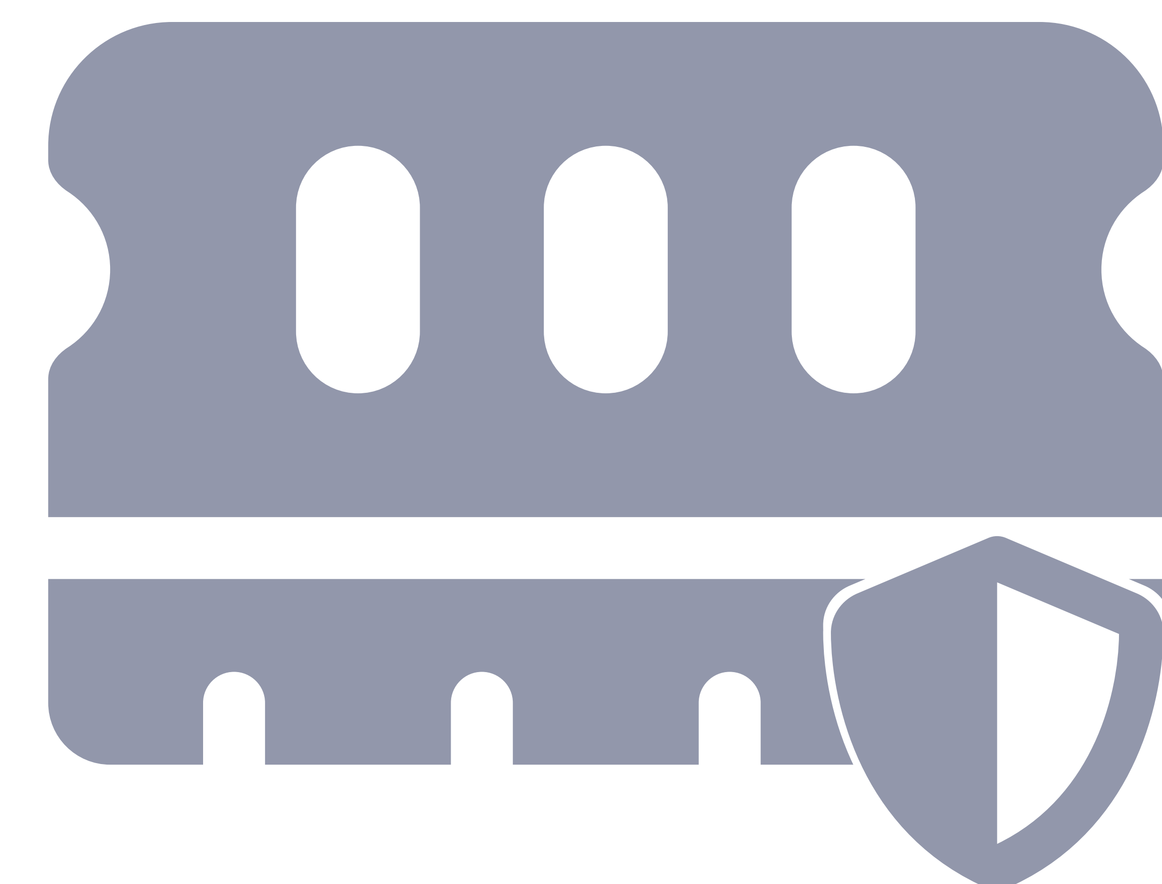
- Page Protection Layer
- Protects signed userspace code and some kernel data from being modified (even by the kernel)
  - Essentially the last line of defense
- Higher privileged than the kernel itself



PPL

Page Tables

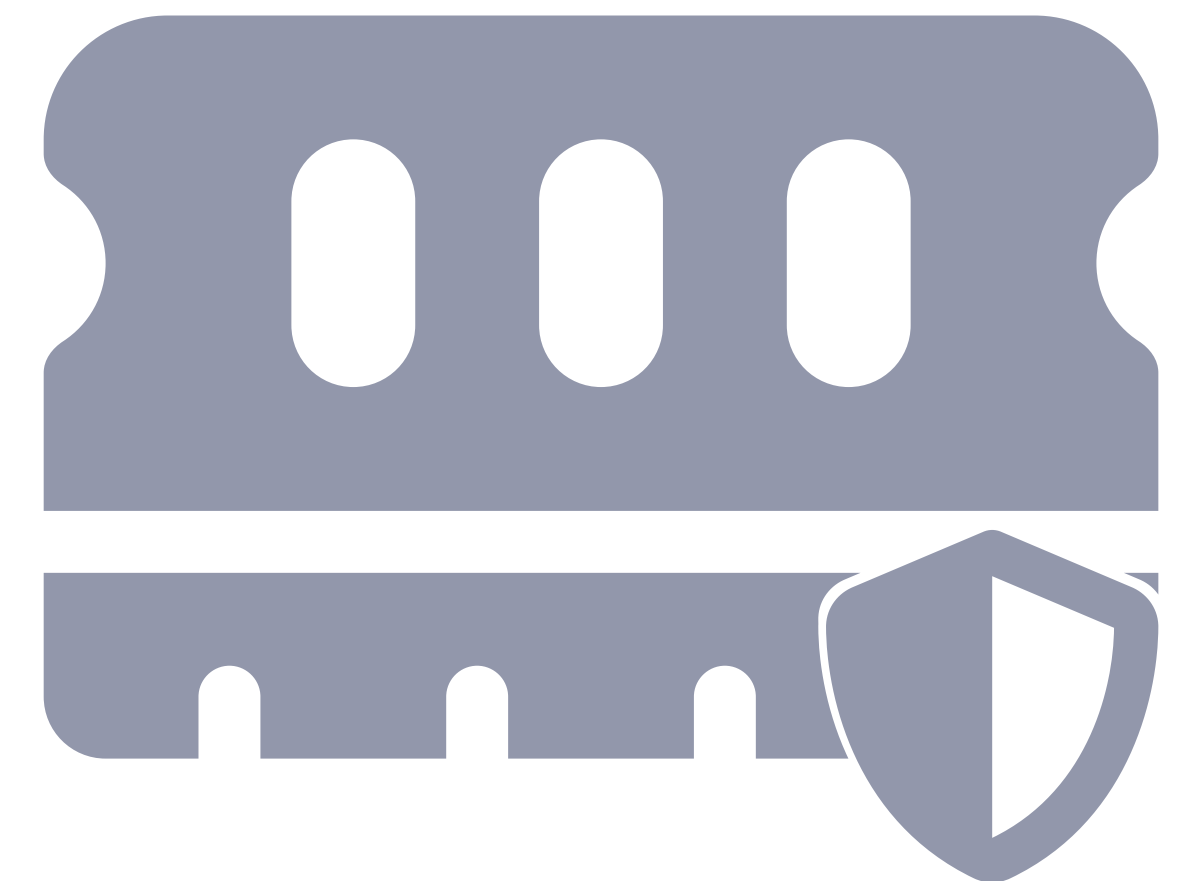
- PPL also manages all page tables
  - To prevent the kernel from mapping PPL-protected data
- Exports some functions to the regular kernel for (un)mapping



PPL

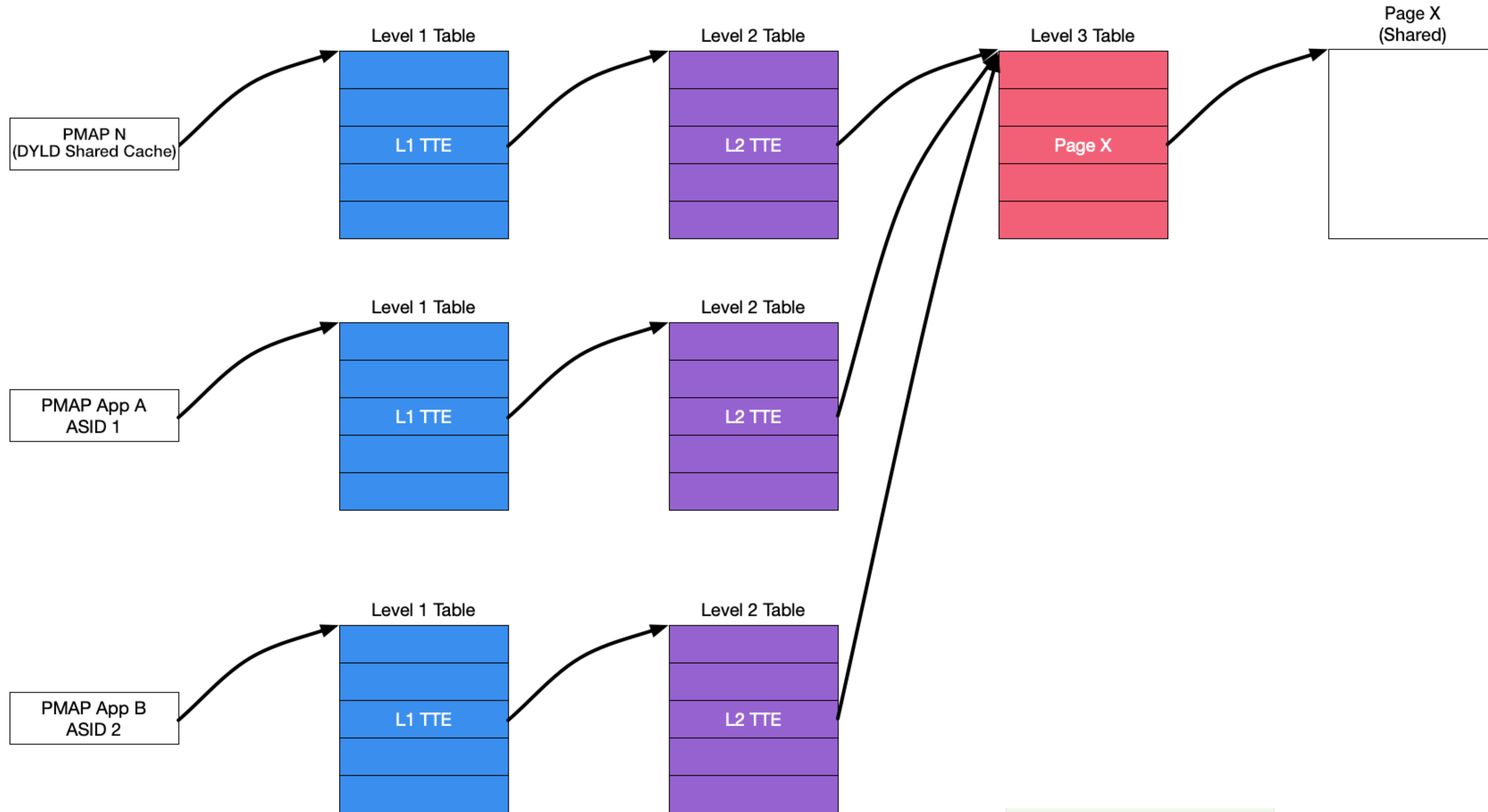
Nested Page Tables

- Dyld shared cache is mapped into every process and very large
- Page tables for the shared cache are reused across multiple processes (nesting)
  - Saves quite a lot of memory



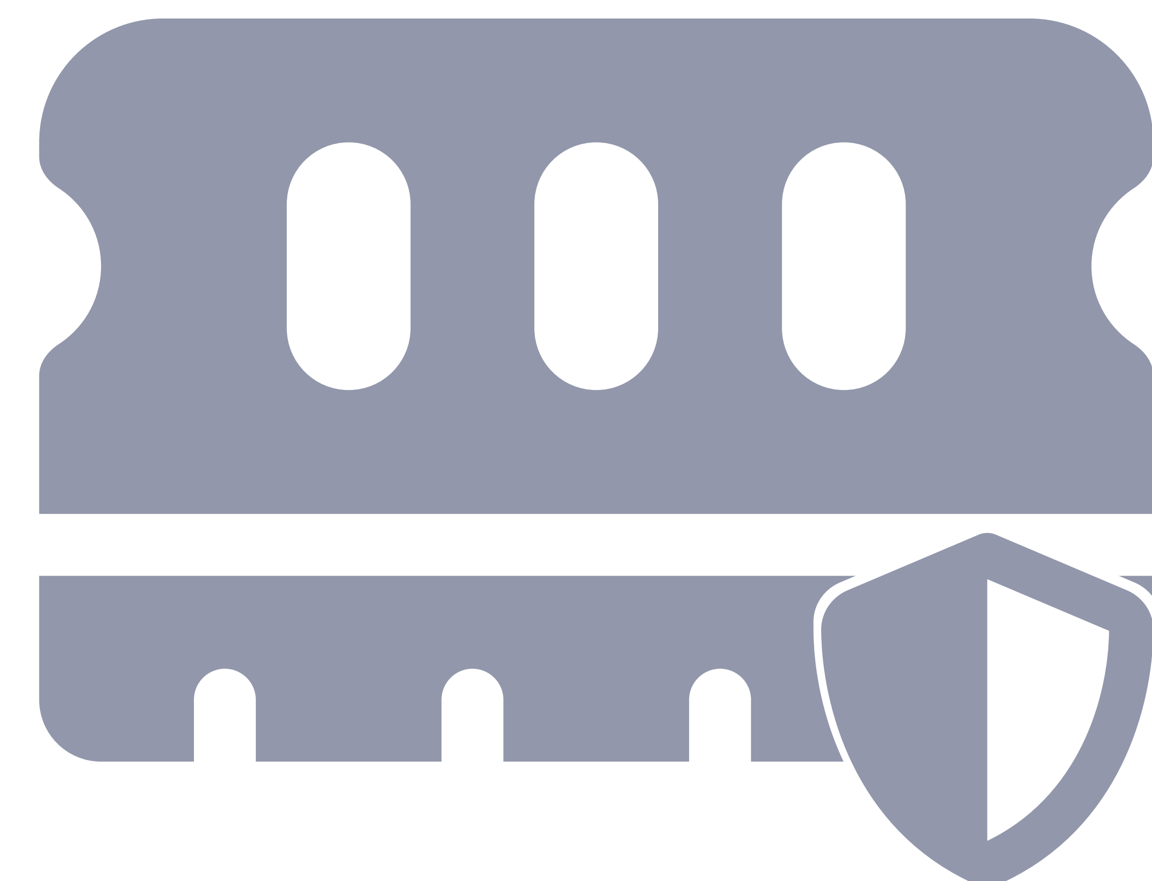
PPL

DYLD Shared Cache Example



TLB

- Translation Lookaside Buffer
- Caches virtual-to-physical address translations
- TLB must be flushed when changing page table entries
- TLB entries may have an Address Space ID (ASID) to limit flushes

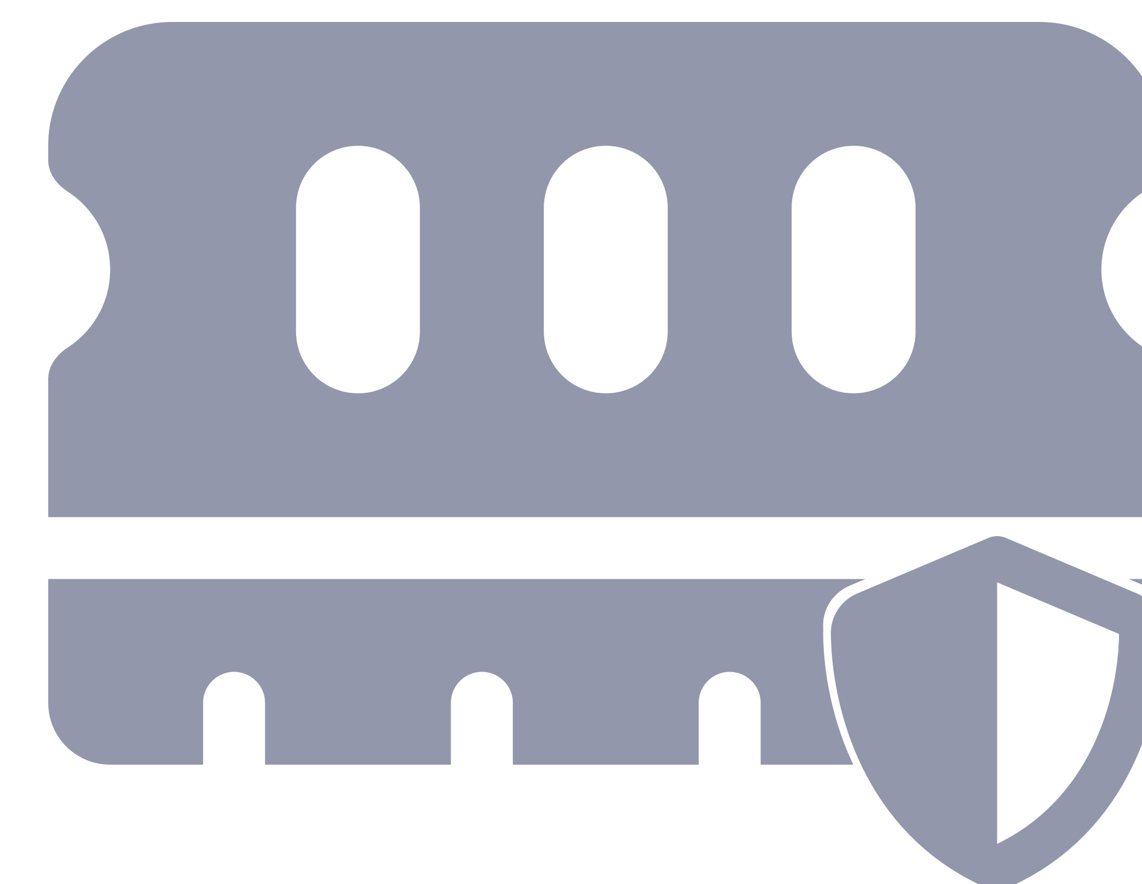


flush_mmu_tlb_region_asid_async

```
// ...
if (npages > ARM64_FULL_TLB_FLUSH_THRESHOLD) {
 boolean_t       flush_all = FALSE;
 if ((asid == 0) || (pmap->type == PMAP_TYPE_NESTED)) {
  flush_all = TRUE;
 }

 if (flush_all) {
  flush_mmu_tlb_async();
 } else {
  flush_mmu_tlb_asid_async((uint64_t)asid << TLBI_ASID_SHIFT);
 }

 return;
}

vm_offset_t end = tlbi_asid(asid) | tlbi_addr(va + length);
va = tlbi_asid(asid) | tlbi_addr(va);
if (pmap->type == PMAP_TYPE_NESTED) {
 flush_mmu_tlb_allentries_async(va, end, pmap_page_size, last_level_only);
} else {
 flush_mmu_tlb_entries_async(va, end, pmap_page_size, last_level_only);
}
```

PPL TLB Flush

- PPL uses the page table ASID to flush the TLB
  - Except for nested page tables
  - Entries with different ASID's are unaffected
- What would happen if a nested entry is deleted through a non-nested page table?



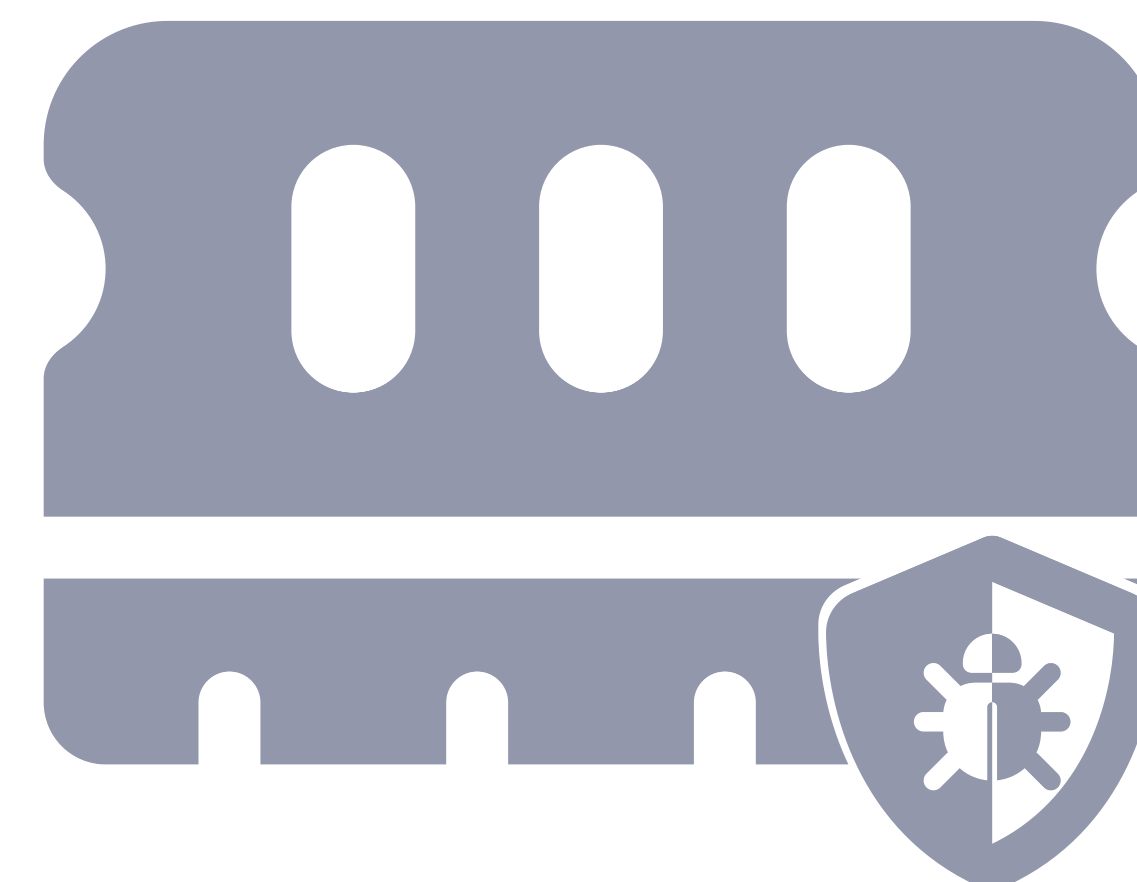
PPL

tlbFail

tlbFail - Page X deleted through pmap B

Exploitation

- Process A still has access to Page X through the TLB
  - But: Page X has a refcount of zero
- Tell PPL that it now owns Page X and force it to reuse the page as Level 3 translation table
- Use the stale TLB entry to map Page X by creating a new translation table entry

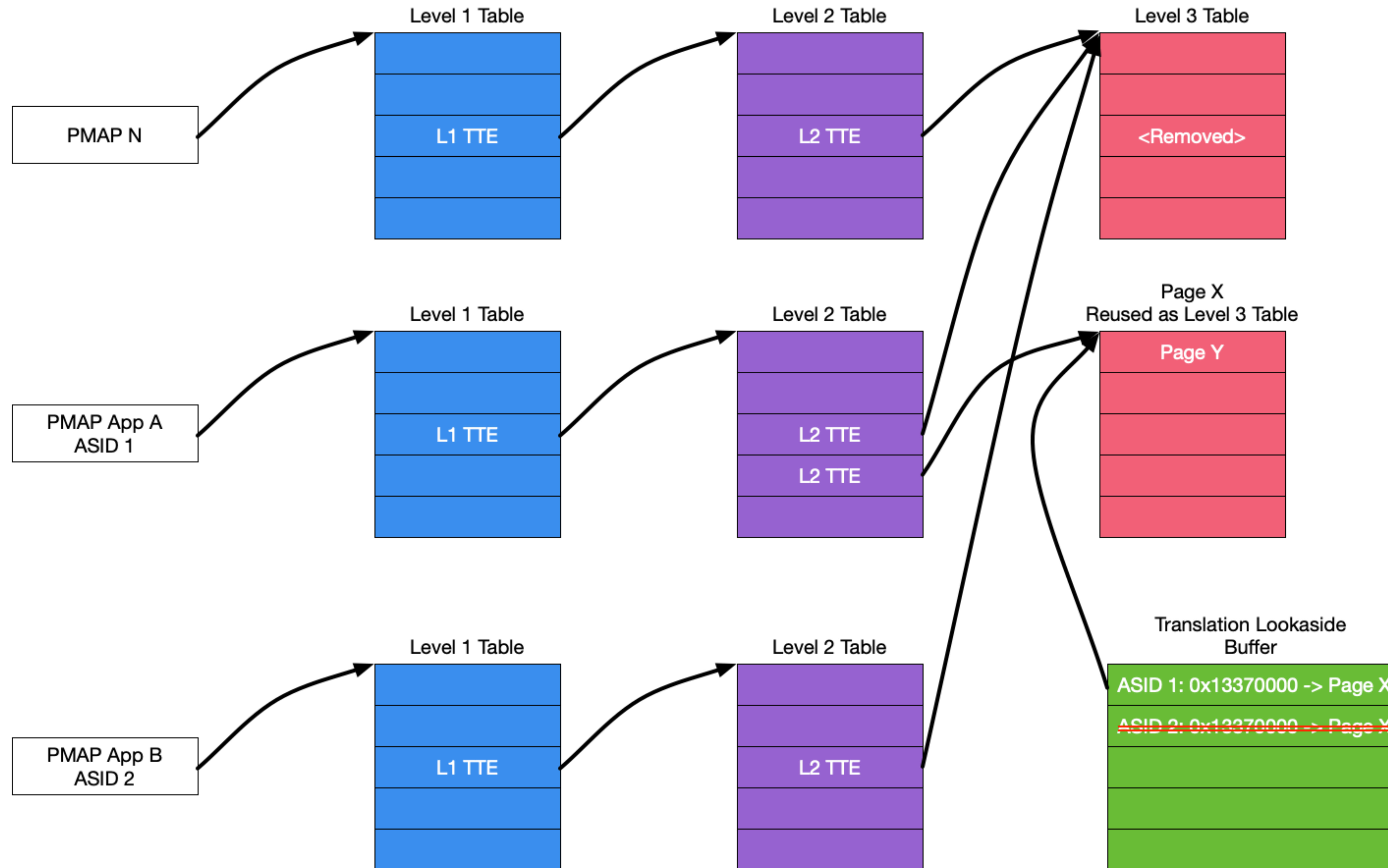


PPL

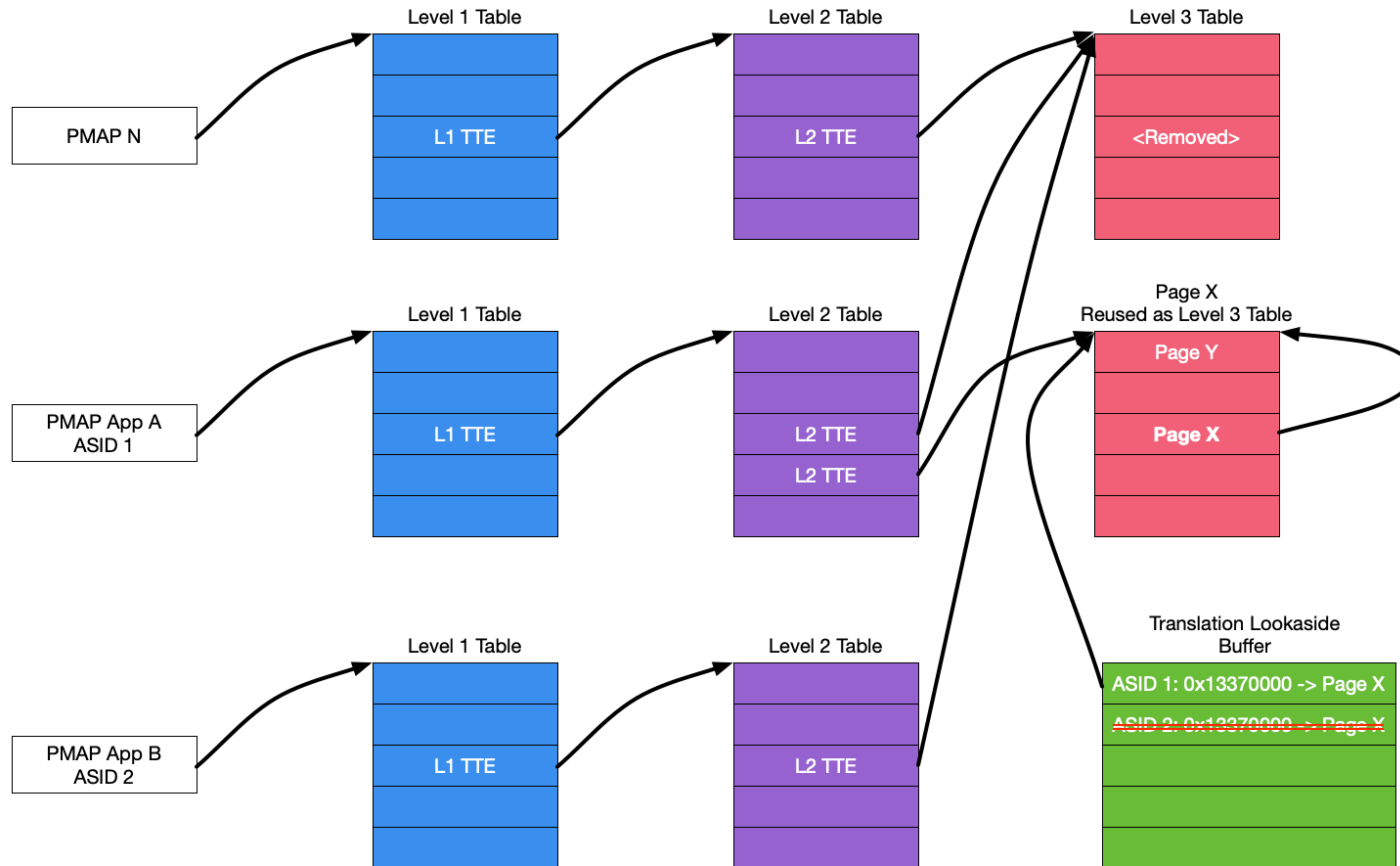
pinauten
security research

tlbFail - PPL now owns Page X

tlbFail - PPL reuses Page X



tlbFail - Page X mapped to itself



Demo

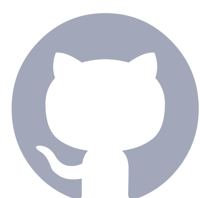
pinauten
security research

Any Questions?

Linus Henze

@LinusHenze

www.pinauten.de


github.com/pinauten/Fugu15

Contact