black hat USA 2020
AUGUST 5-6, 2020
BRIEFINGS

Breaking Samsung's Root of Trust: Exploiting Samsung S10 S-Boot

Jeffxx

Jeff Chao (Jeffxx)

- Researcher at Trapa Security
- Ex-senior Researcher at TeamT5
- Member of HITCON CTF Team
- Member of Chroot
- Focus on Mobile and IoT Vulnerabilities

AGENDA

01 Demo

02 Related Work

03 Vulnerabilities in Secure Boot

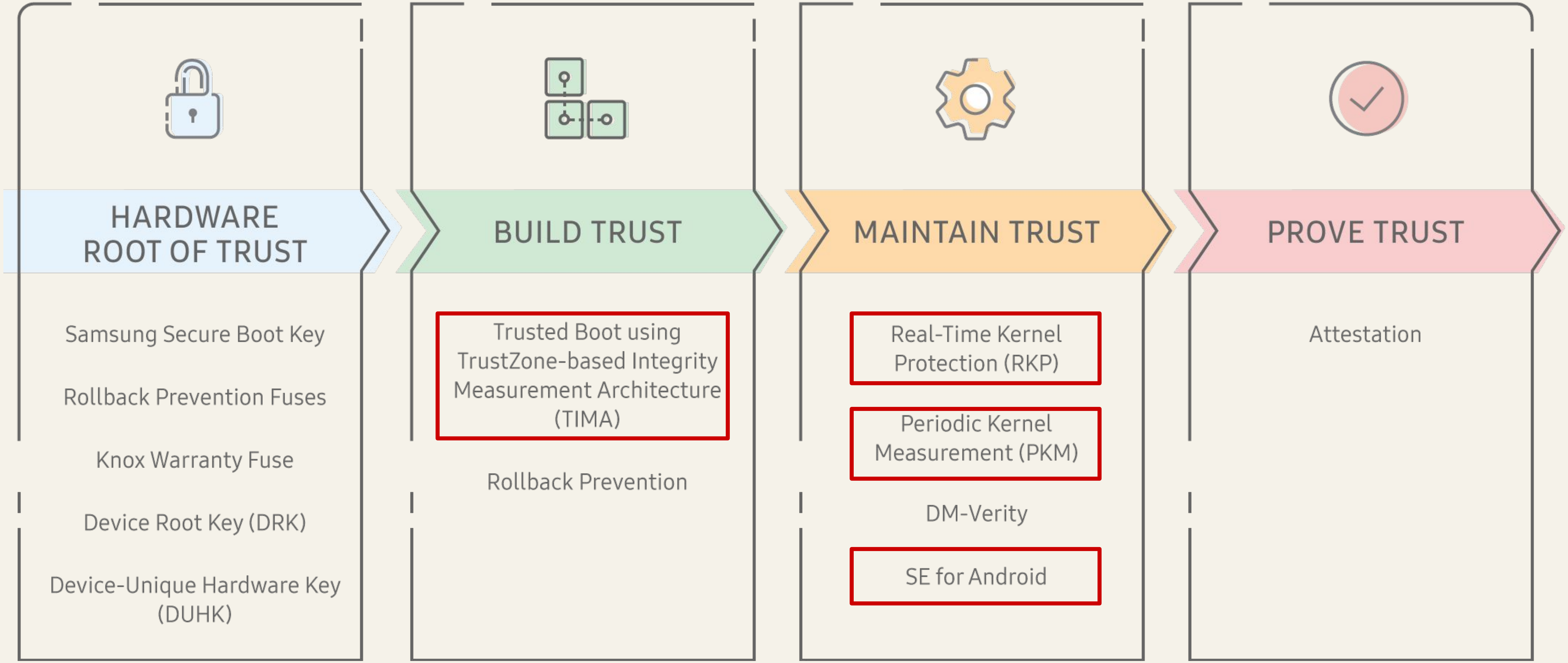
04 Samsung Security Framework - Knox

05 After Code Execution on S-boot

Nintendo

Samsung Security Framework Knox

Knox - Root of Trust



HARDWARE ROOT OF TRUST

Samsung Secure Boot Key

Rollback Prevention Fuses

Knox Warranty Fuse

Device Root Key (DRK)

Device-Unique Hardware Key (DUHK)

BUILD TRUST

Trusted Boot using TrustZone-based Integrity Measurement Architecture (TIMA)

Rollback Prevention

MAINTAIN TRUST

Real-Time Kernel Protection (RKP)

Periodic Kernel Measurement (PKM)

DM-Verity

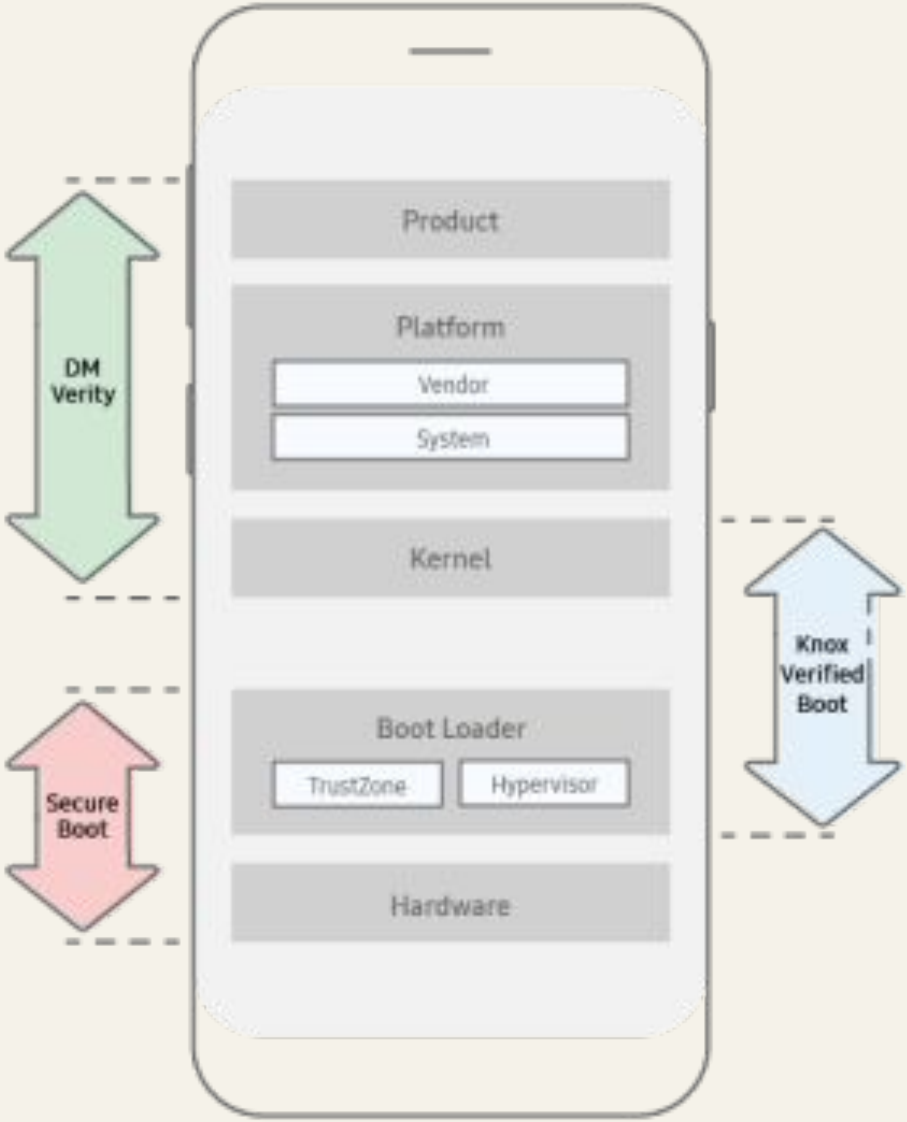
SE for Android

PROVE TRUST

Attestation

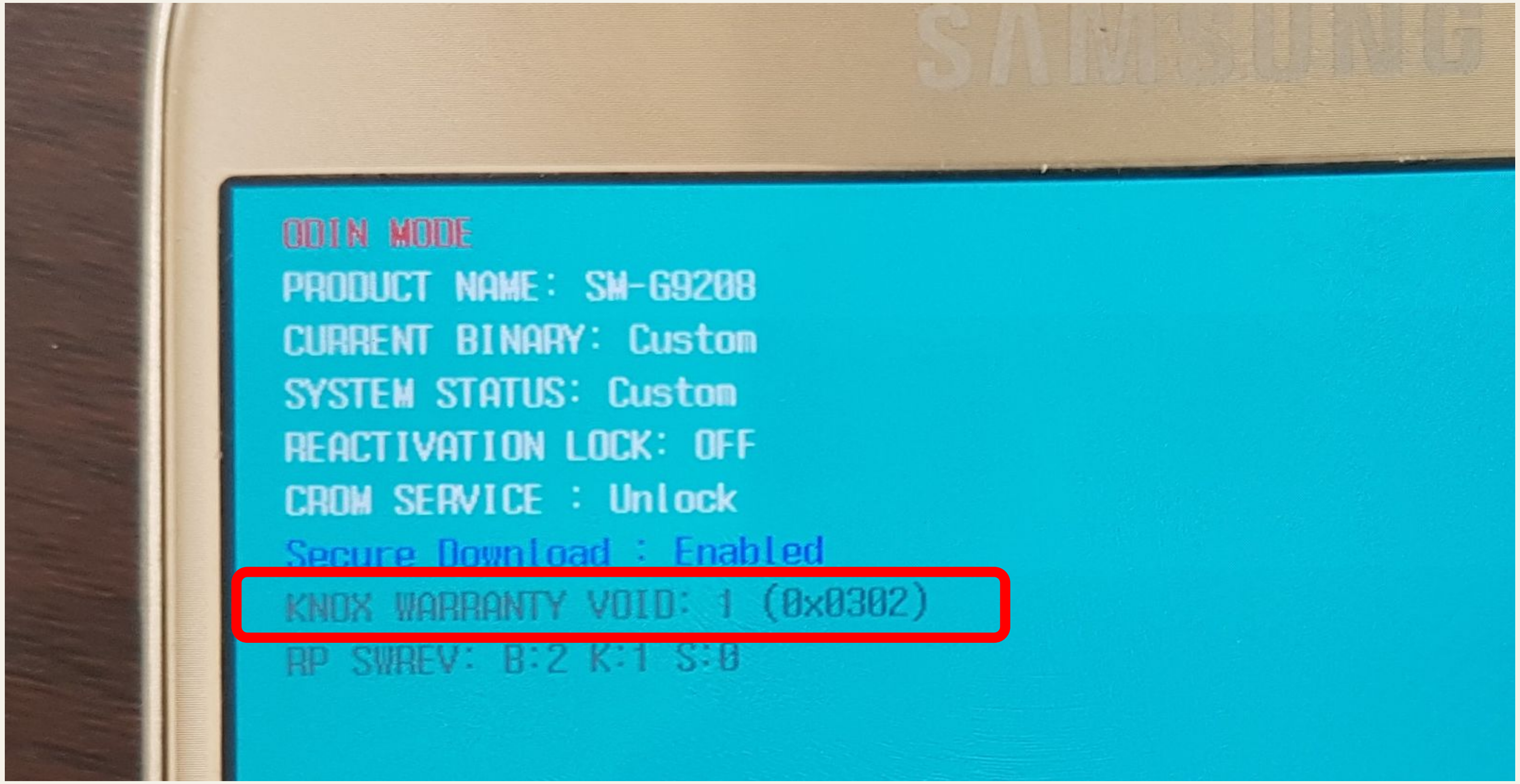
Knox – Trusted Boot

- Hardware PBL
  - Verify secure boot(S-Boot) & load
- S-Boot
  - Set handler for Monitor mode, drop privilege
  - Request EL3 to initial TEEOS
  - Verify & Load Hypervisor (uh.bin)
  - Verify & Load Kernel (boot.img)
- Kernel with DM-Verity
  - Verify system.img & mount
  - Verify vendor.img & mount



Knox bit (warranty bit)

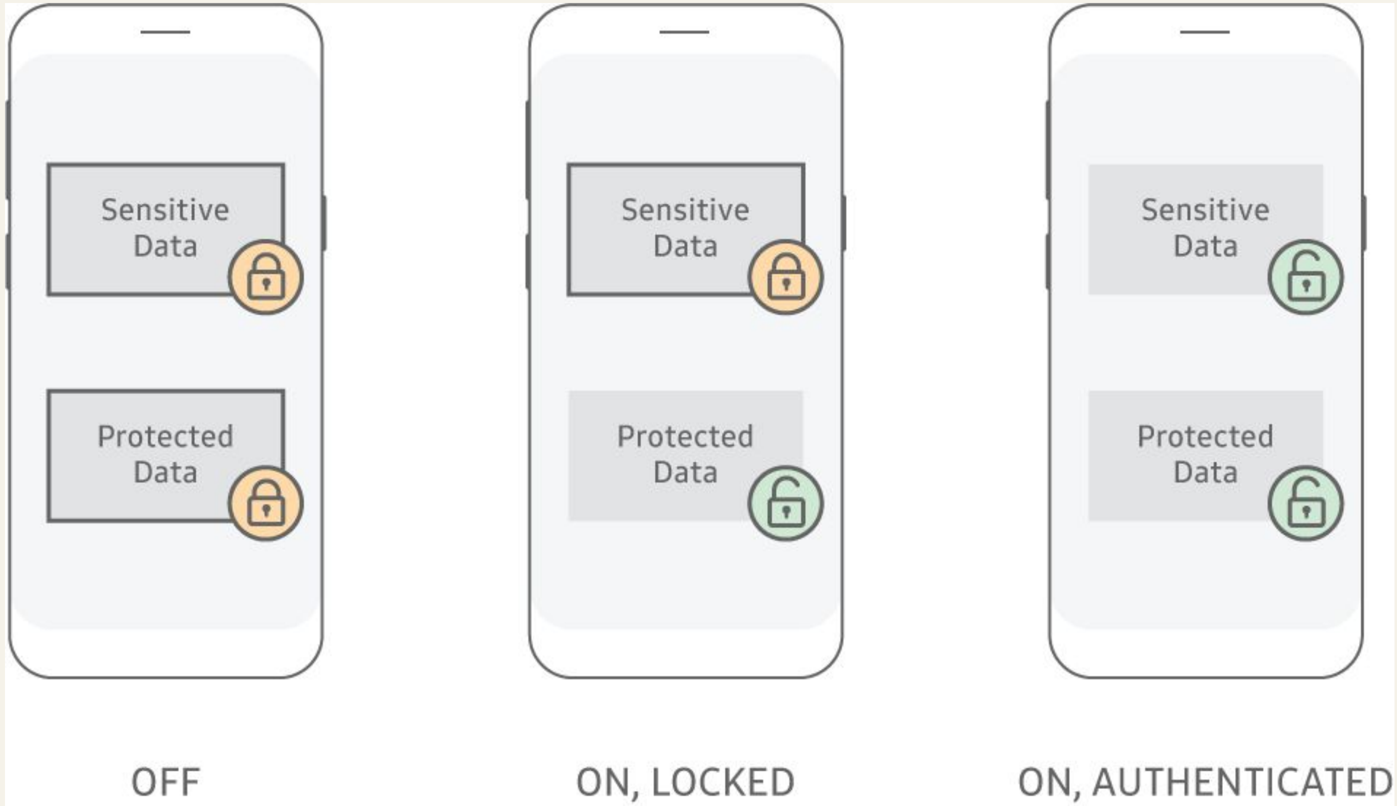
- One-time fuse, can’t restore
- Blow the fuse when trying to boot a custom image and prevent further booting



```
ODIN MODE
PRODUCT NAME: SM-G9208
CURRENT BINARY: Custom
SYSTEM STATUS: Custom
REACTIVATION LOCK: OFF
CROM SERVICE : Unlock
Secure Download : Enabled
KNOX WARRANTY VOID: 1 (0x0302)
RP SWREV: B:2 K:1 S:0
```

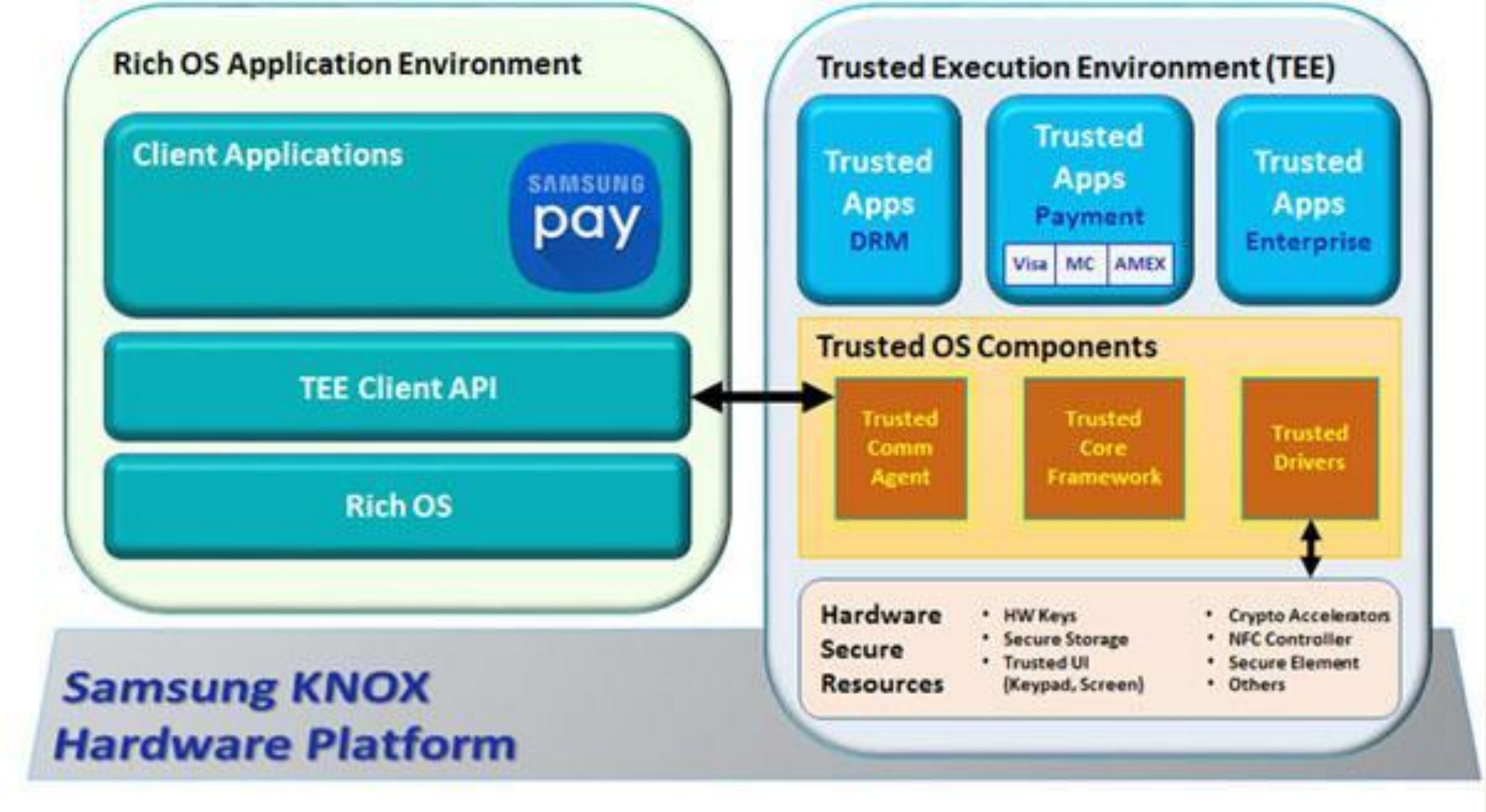
Sensitive Data Protection

- The storage (Sensitive Data) is encrypted when the device is locked
- Encrypted Keys are stored in trustzone

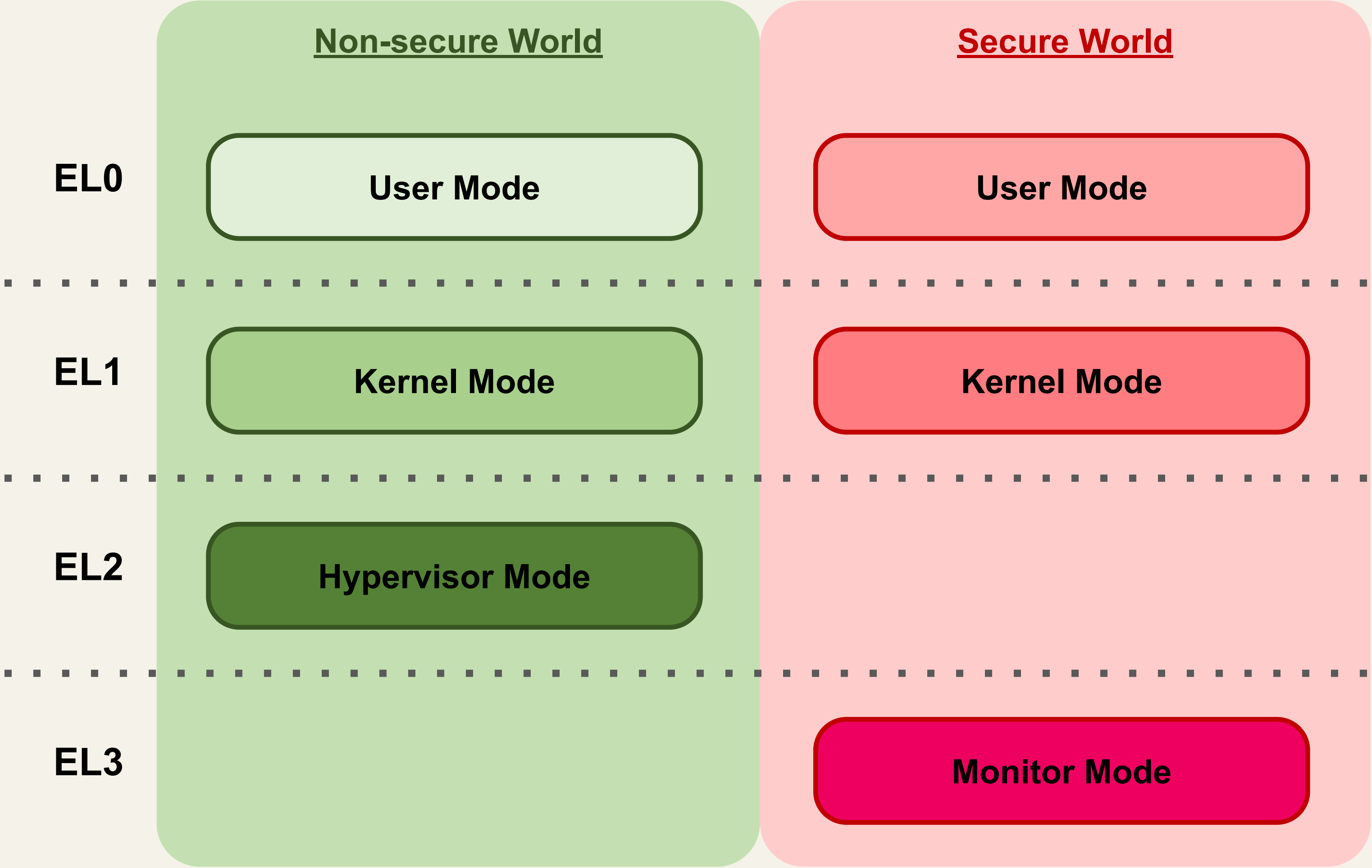


Sensitive Data Protection cont

- Some critical information can only be decrypted by trustlet



ARM Trustzone

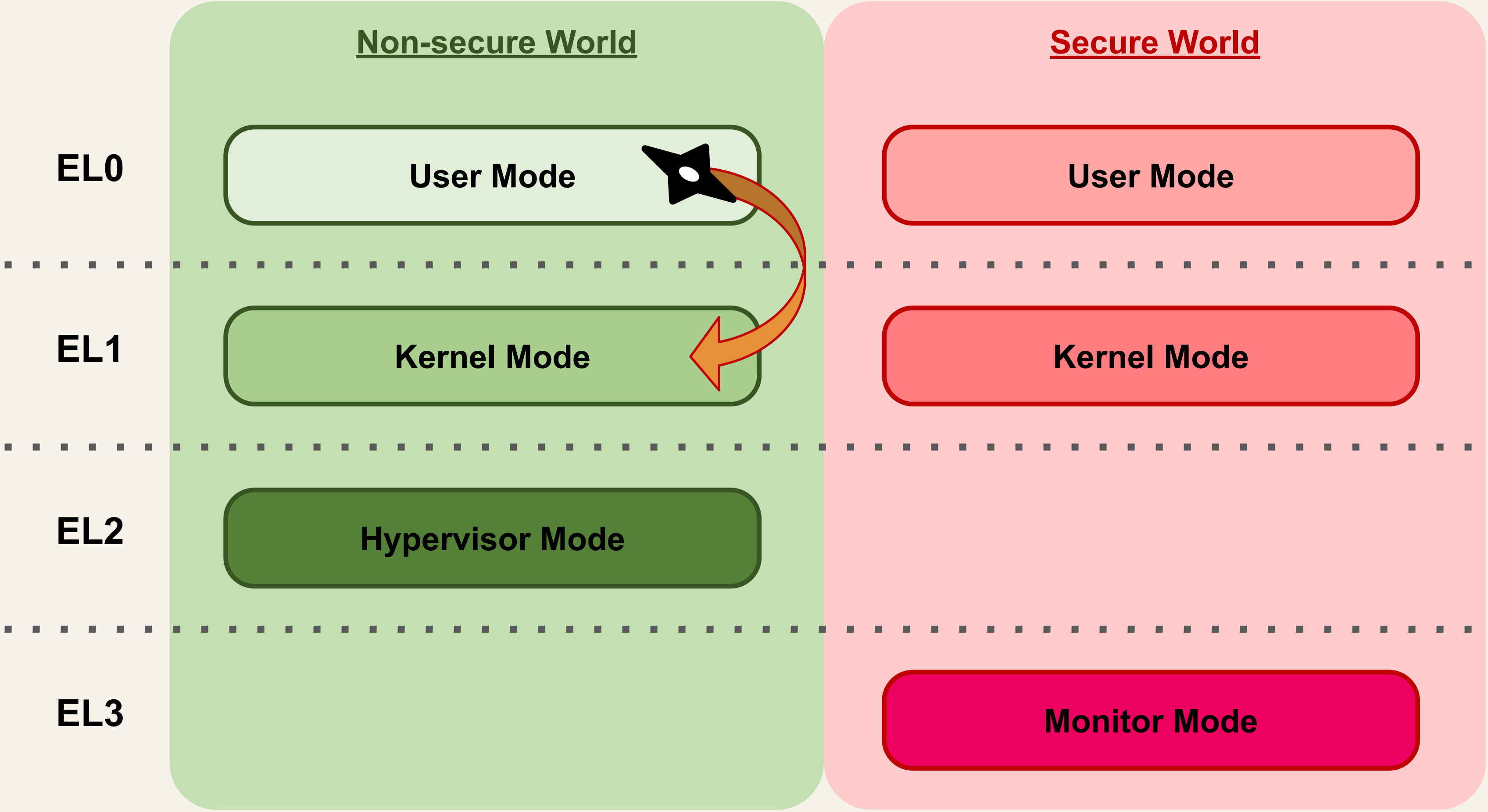


	Non-secure World	Secure World
EL0	User Mode	User Mode
EL1	Kernel Mode	Kernel Mode
EL2	Hypervisor Mode	
EL3		Monitor Mode

Related Work

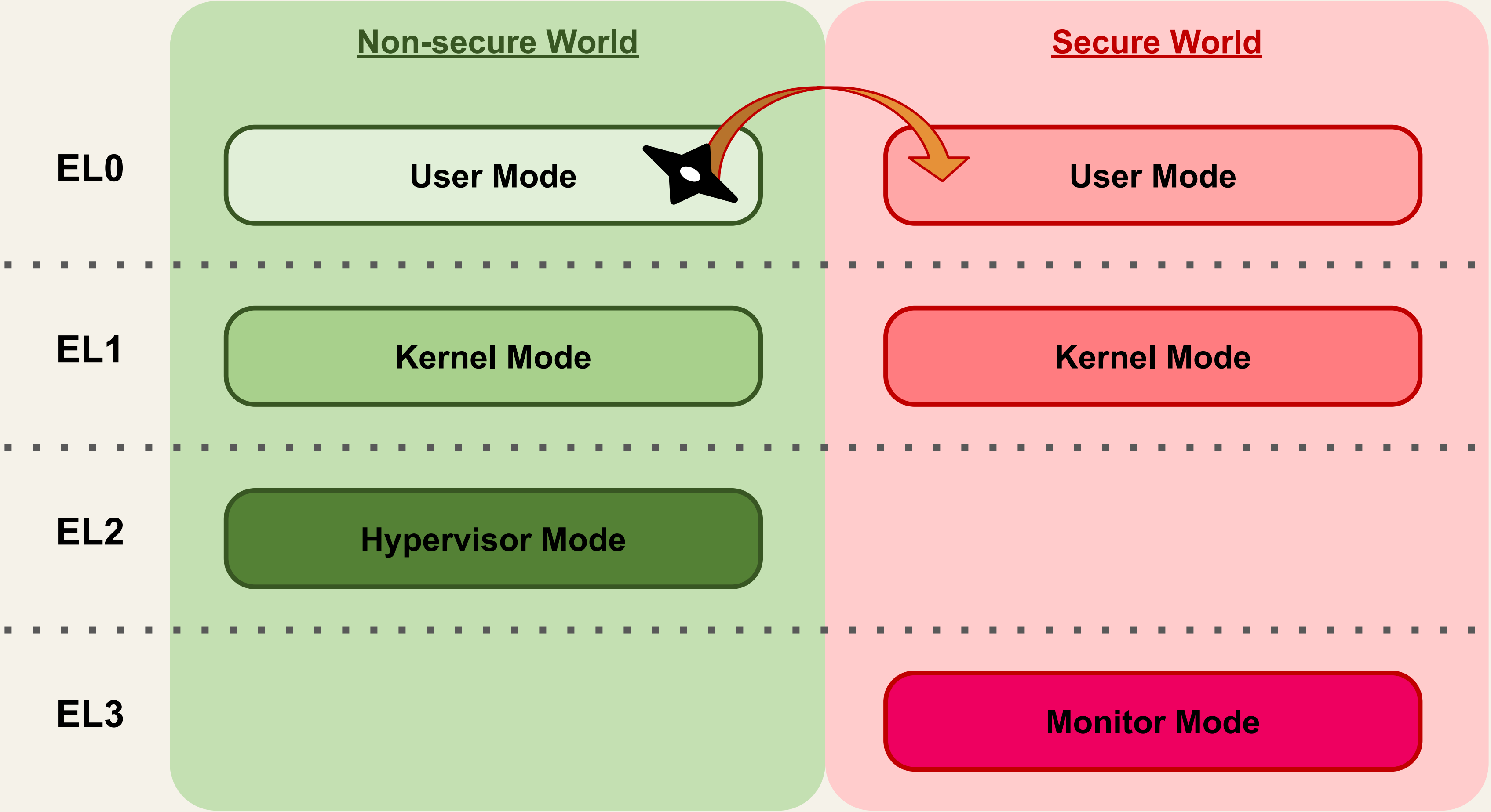
BH17 – Defeating Samsung KNOX with zero privilege by returnsme

- EL0 -> EL1 (kinibi)



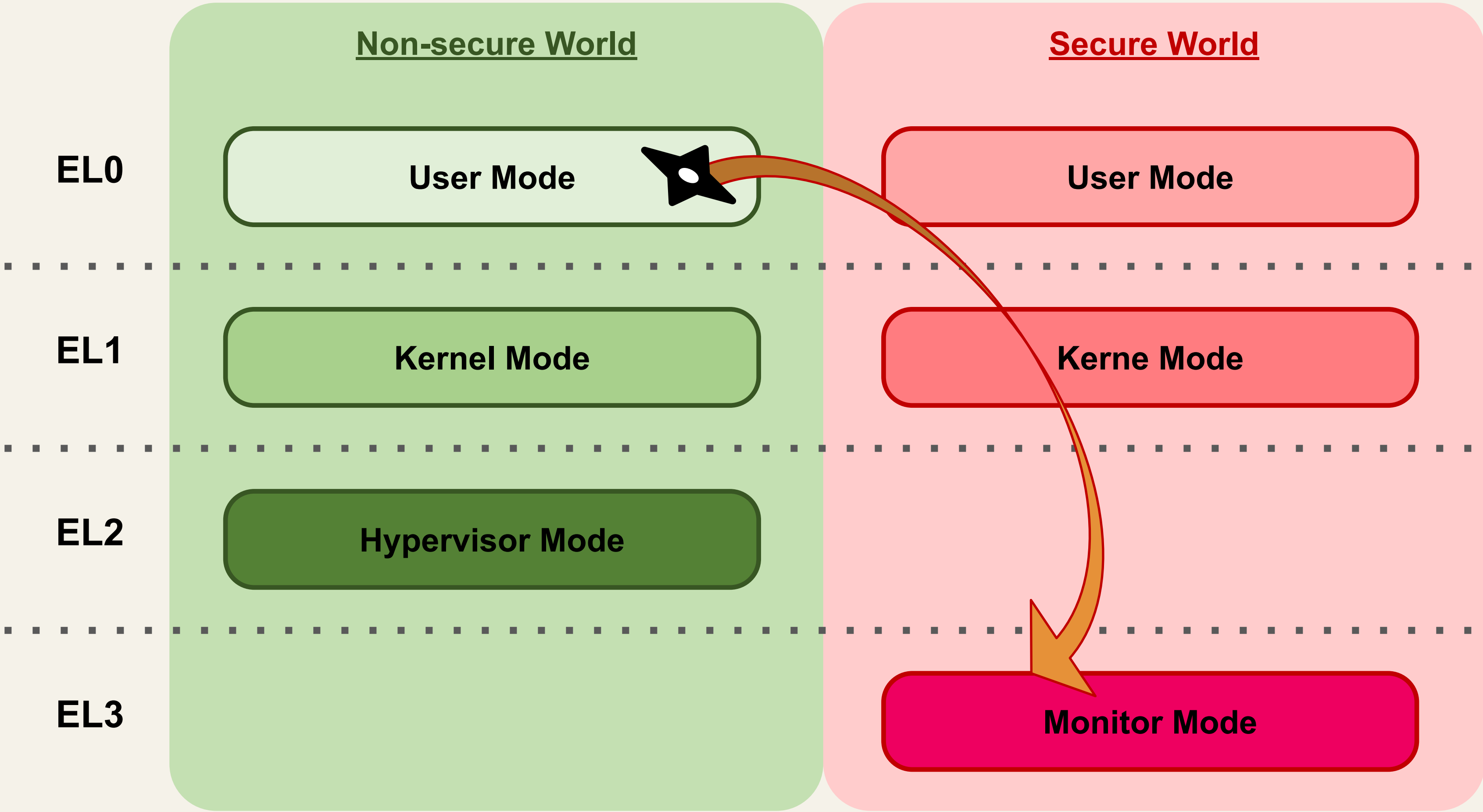
BH17 EU - How Samsung Secures Your Wallet by Tencent Lab

- EL0 -> Secure EL0 (kinibi)



BH19 – Breaking Samsung's Arm Trustzone

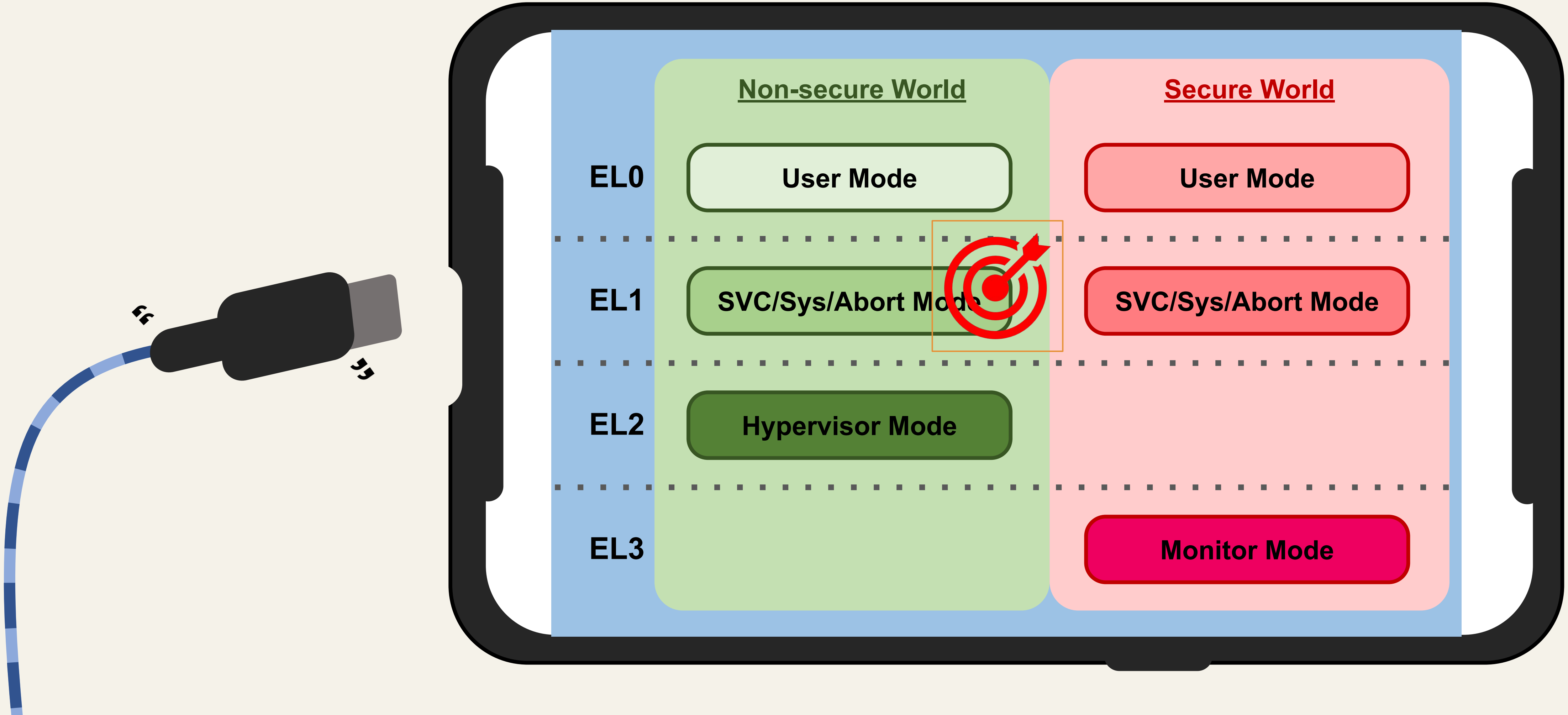
- EL0 -> Secure-EL3 (kinibi, S8 and before)



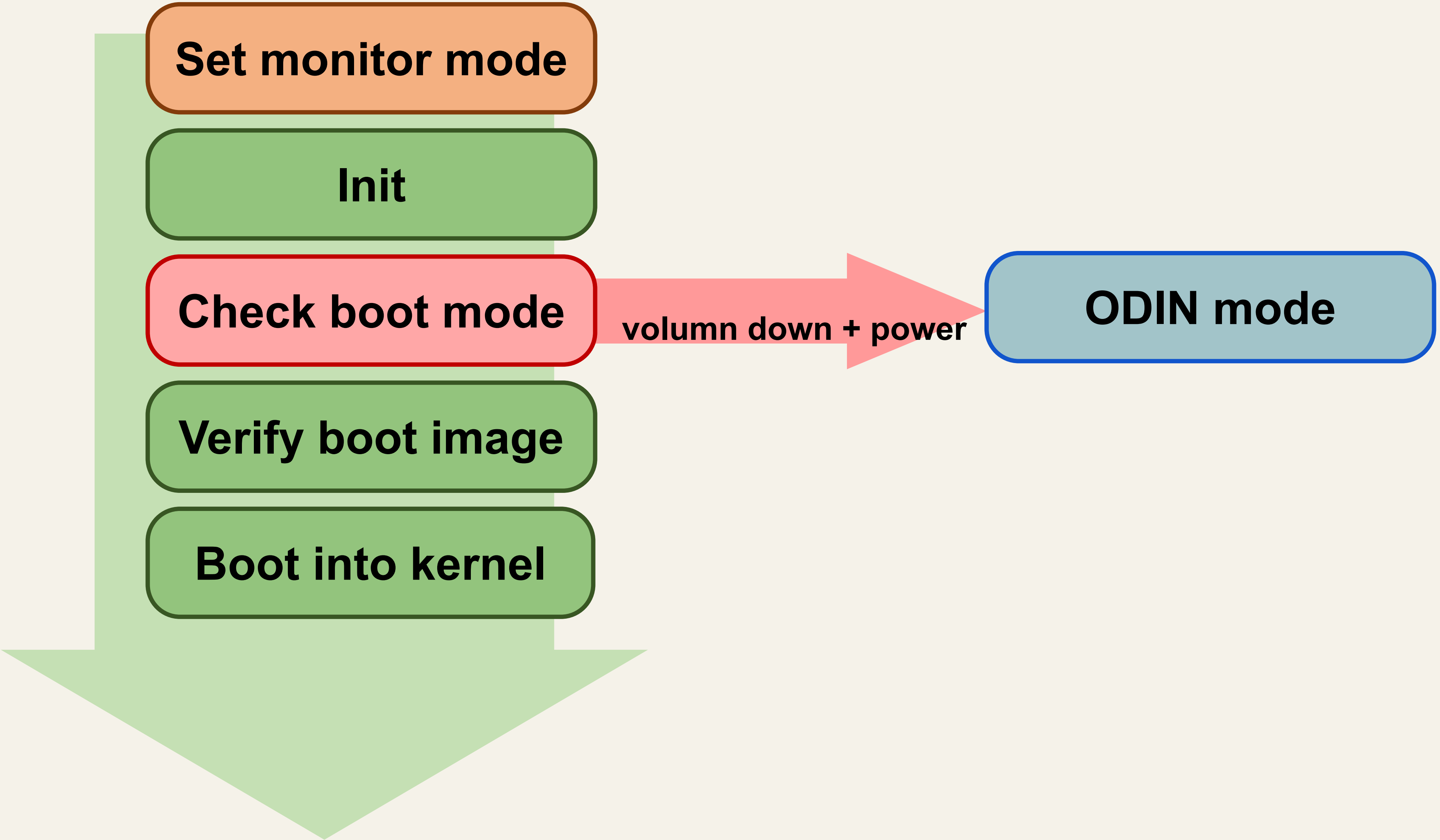
What if the device is turned off & we don’t know the passcode?

In this talk

- out-side the box(locked phone) -> Non-Secure EL1



S-Boot Boot Flow



Set monitor mode

Init

Check boot mode

volumn down + power

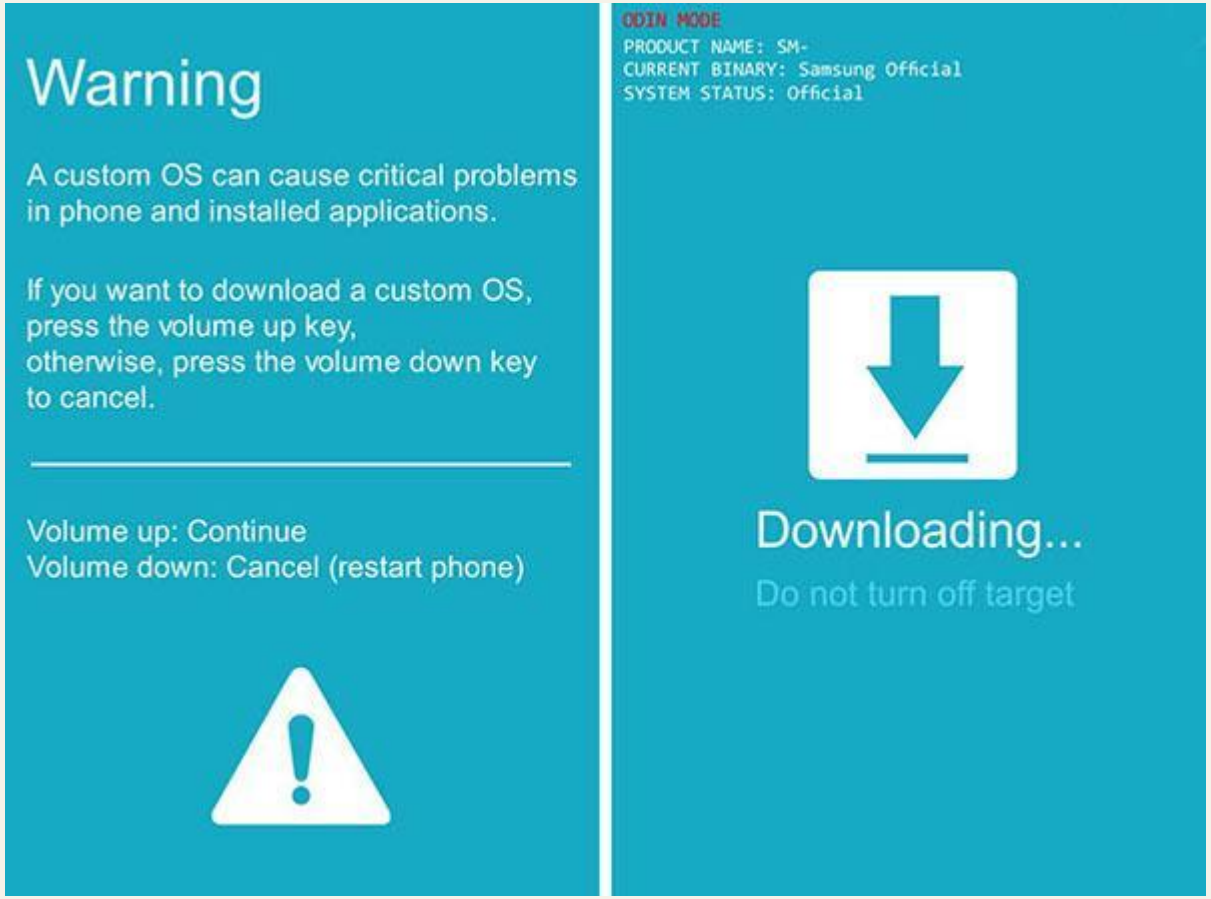
ODIN mode

Verify boot image

Boot into kernel

ODIN mode

- Flash **stock** firmware
- Rollback prevention



Warning

A custom OS can cause critical problems in phone and installed applications.

If you want to download a custom OS, press the volume up key, otherwise, press the volume down key to cancel.

Volume up: Continue
Volume down: Cancel (restart phone)

```
ODIN MODE
PRODUCT NAME: SM-
CURRENT BINARY: Samsung Official
SYSTEM STATUS: Official
```

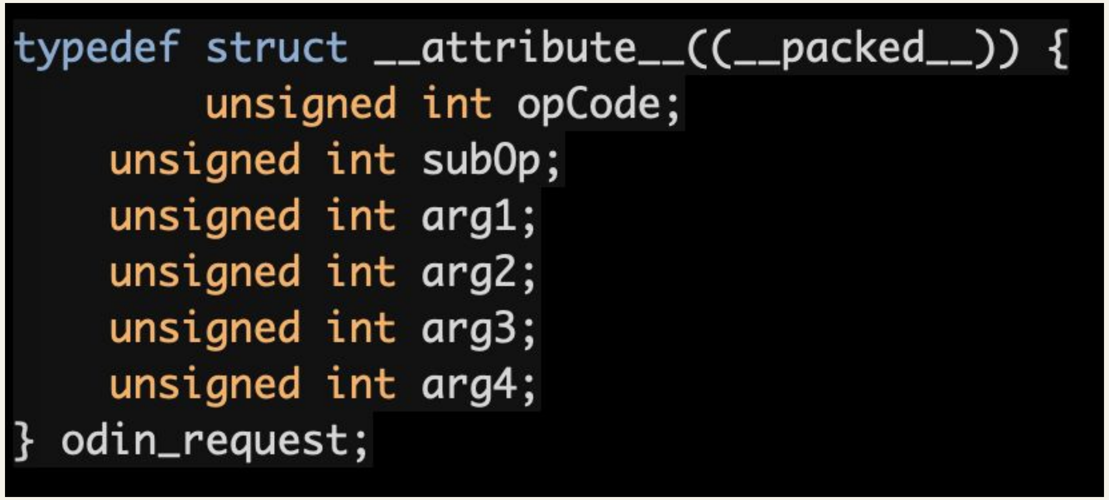
Downloading...

Do not turn off target

Vulnerability I

Odin Request

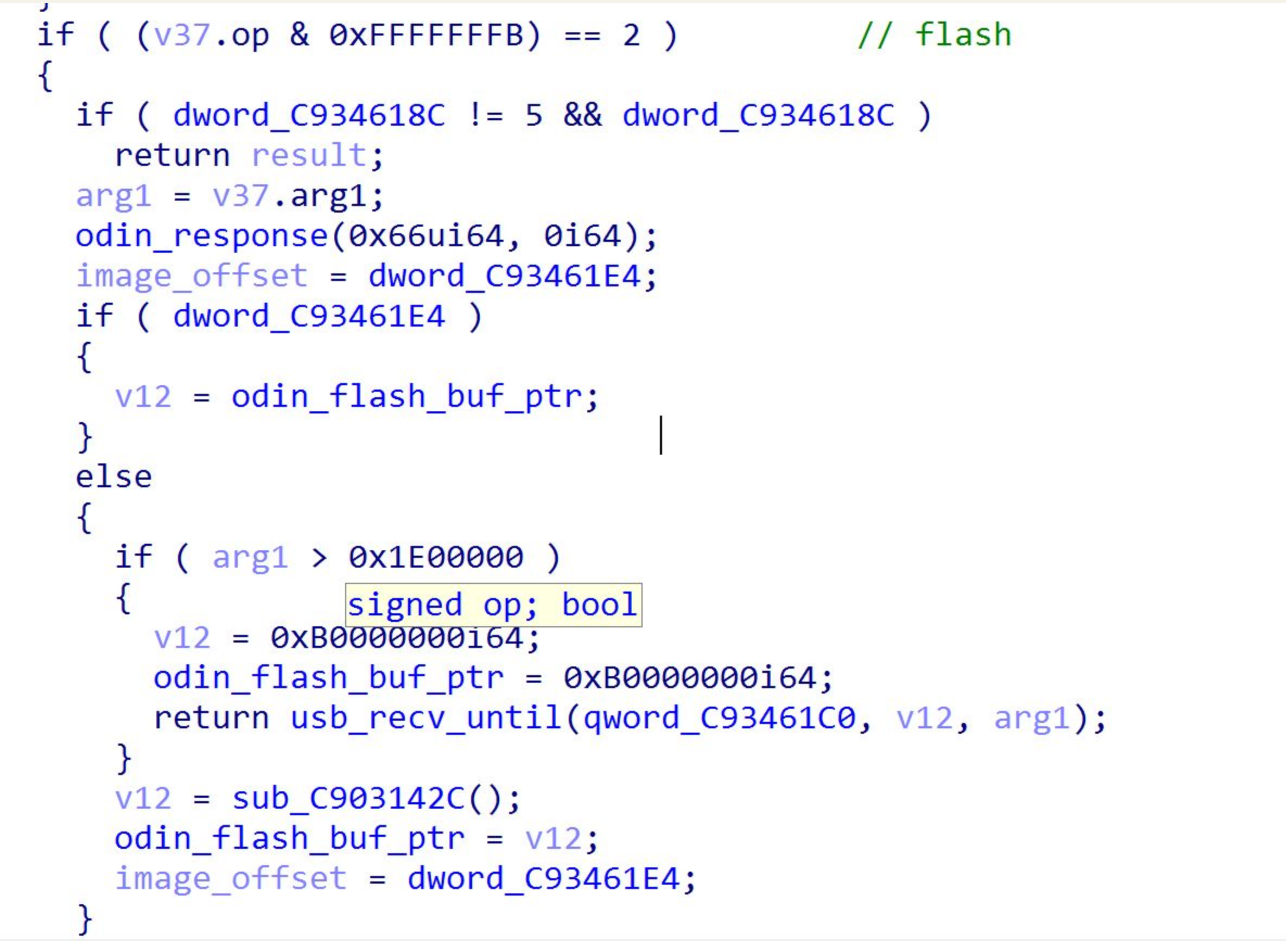
- opCode
  - 0x64 Odin mode initial & settings
  - 0x65 Flash PIT
  - 0x66 Flag image
- subOp
  - Depends on opCode
  - Maybe initialize, set, get …etc
- arg1 ~ arg4
  - assign size or some value



```
typedef struct __attribute__((__packed__)) {
        unsigned int opCode;
    unsigned int subOp;
    unsigned int arg1;
    unsigned int arg2;
    unsigned int arg3;
    unsigned int arg4;
} odin_request;
```

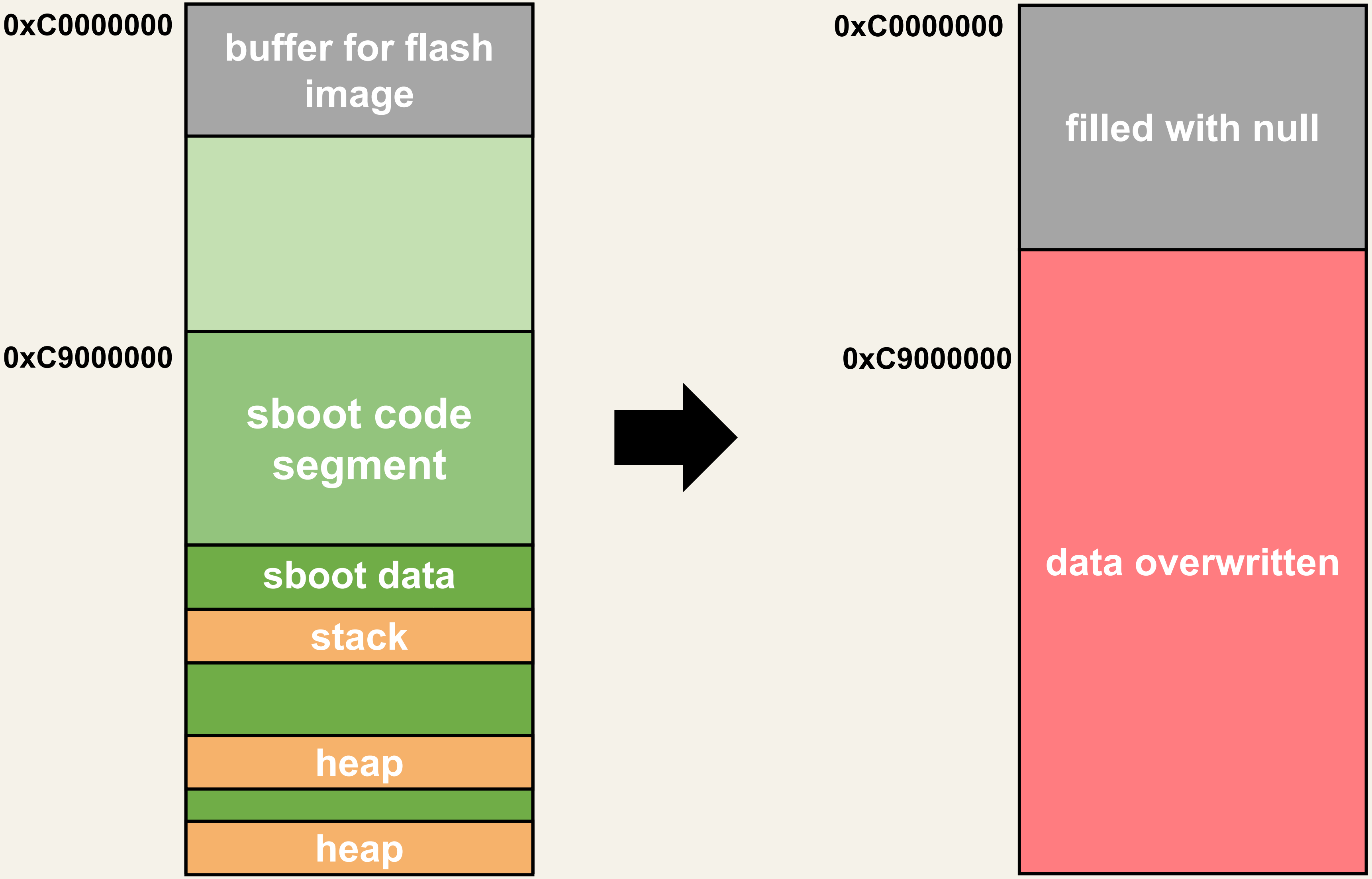
Odin Flash Image Command

- No check for provided size
- Integer overflow
  - Use 0xC0000000 if less then 0x1e00000
  - Otherwise use 0xB0000000
- Copy to buffer
  - S8 and before at 0xC0000000
  - S9 and later at 0x880000000



```
if ( (v37.op & 0xFFFFFFFB) == 2 )         // flash
{
  if ( dword_C934618C != 5 && dword_C934618C )
    return result;
  arg1 = v37.arg1;
  odin_response(0x66ui64, 0i64);
  image_offset = dword_C93461E4;
  if ( dword_C93461E4 )
  {
    v12 = odin_flash_buf_ptr;
  }
  else
  {
    if ( arg1 > 0x1E00000 )
    {
      v12 = 0xB0000000i64;
      odin_flash_buf_ptr = 0xB0000000i64;
      return usb_recv_until(qword_C93461C0, v12, arg1);
    }
    v12 = sub_C903142C();
    odin_flash_buf_ptr = v12;
    image_offset = dword_C93461E4;
  }
```

Overflow the physical memory

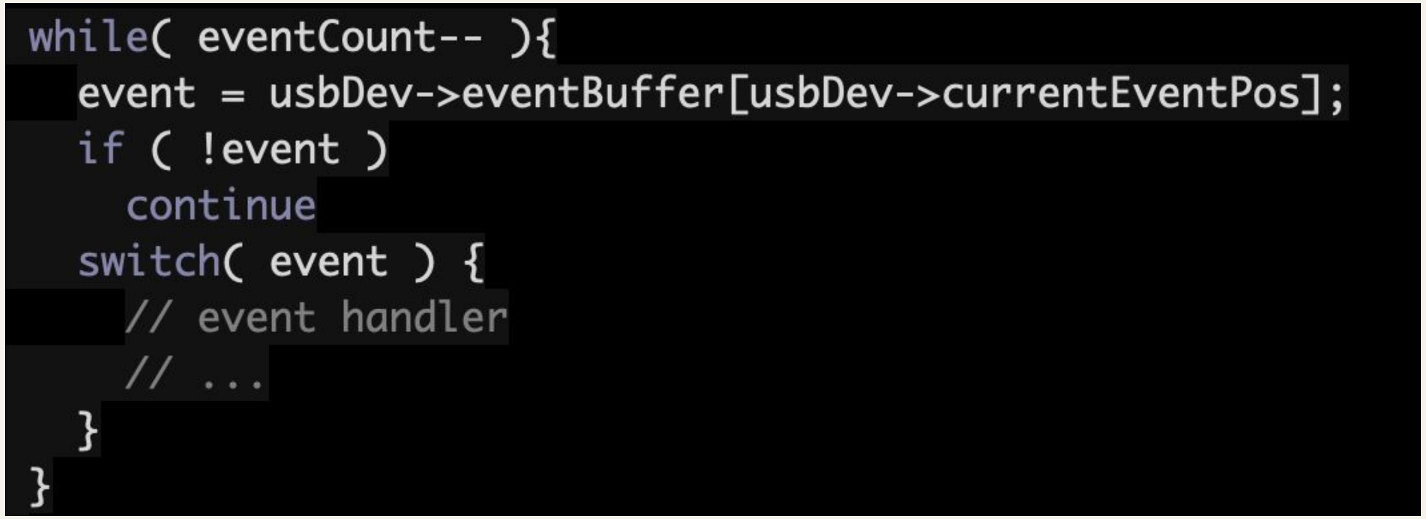


Bypass MMU

- S-Boot code segment at 0xC9000000 but read only
- USB devices have direct memory access
  - Ignores mmu control

Cache Incoherency

- While receiving data, the CPU keeps tracking the USB event
  - This code is cached

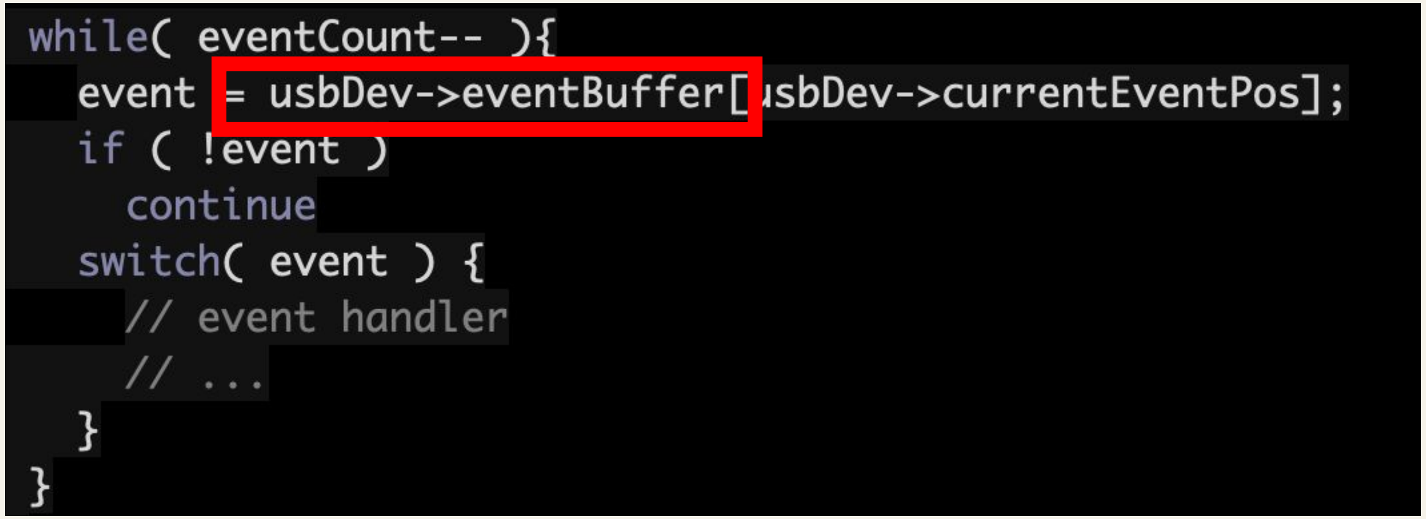


```
while( eventCount-- ){
  event = usbDev->eventBuffer[usbDev->currentEventPos];
  if ( !event )
    continue
  switch( event ) {
    // event handler
    // ...
  }
}
```

- Only the heap will not be cached

Code Execution

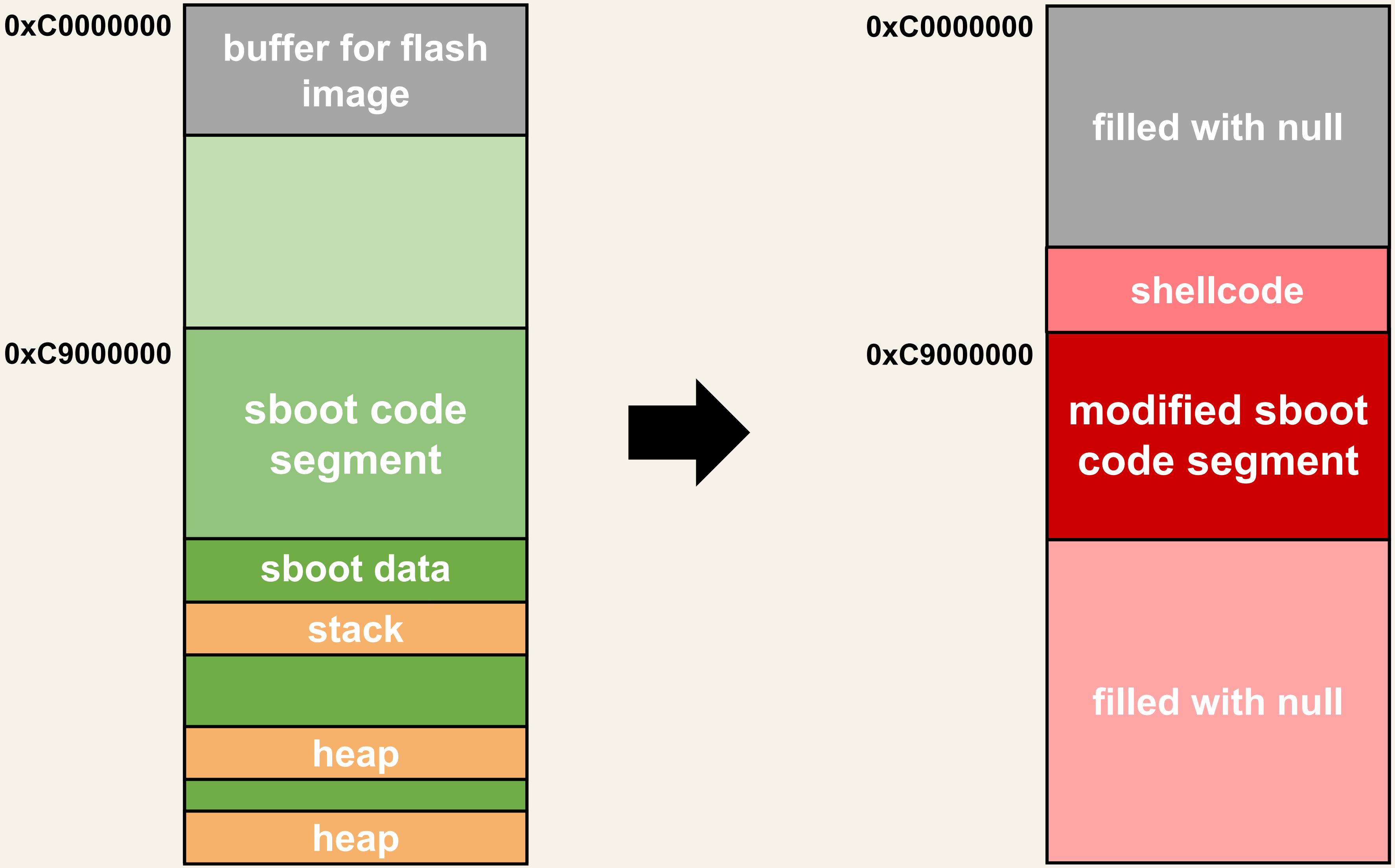
- The heap is not cached, the code accesses a pointer in the heap…
  - Trigger data-abort as soon as we overwrite heap data with NULL



```
while( eventCount-- ){
  event = usbDev->eventBuffer[usbDev->currentEventPos];
  if ( !event )
    continue
  switch( event ) {
    // event handler
    // ...
  }
}
```

- Overwrite the error handler code with jump sled
- Put shellcode in front of the code segment

Overflow the physical memory

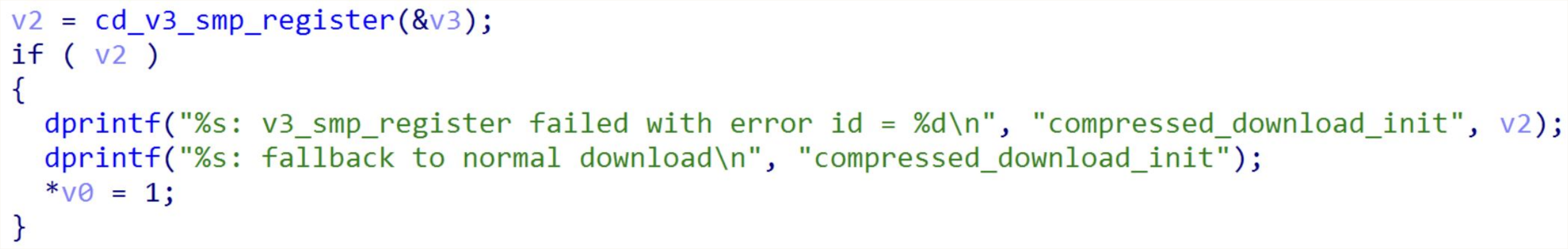


But

- S9 and later are not exploitable
- The default buffer is changed to 0x880000000
- Spent half a year trying to exploit S10

Potential Exploit Path on S10

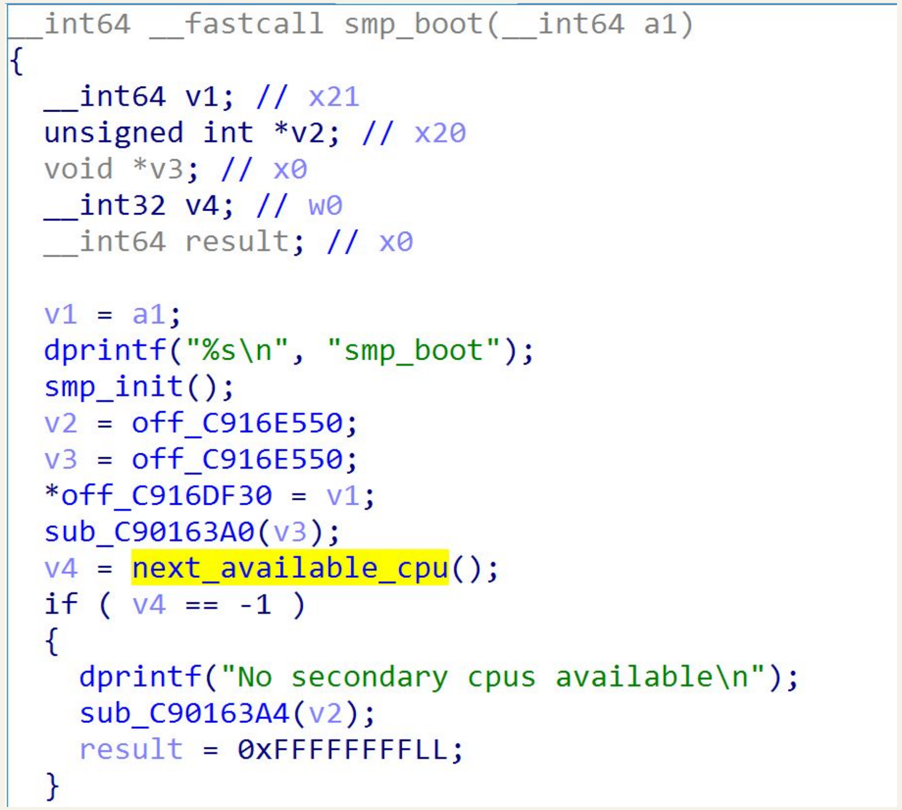
- In S9 and later, ODIN has parallel & compressed download mode
  - It will boot up another 2 cpu, and set the image buffer to 0x880000000
  - Fallback to normal download if boot cpu failure
    - Buffer change back to 0xC0000000



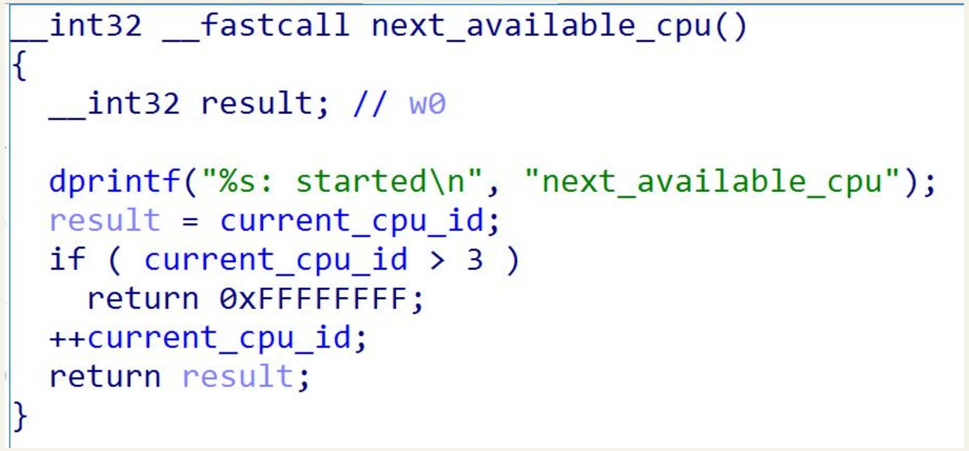
```
v2 = cd_v3_smp_register(&v3);
if ( v2 )
{
  dprintf("%s: v3_smp_register failed with error id = %d\n", "compressed_download_init", v2);
  dprintf("%s: fallback to normal download\n", "compressed_download_init");
  *v0 = 1;
}
```

Potential Exploit Path on S10

- Make CPU boot fails



```
__int64 __fastcall smp_boot(__int64 a1)
{
  __int64 v1; // x21
  unsigned int *v2; // x20
  void *v3; // x0
  __int32 v4; // w0
  __int64 result; // x0

  v1 = a1;
  dprintf("%s\n", "smp_boot");
  smp_init();
  v2 = off_C916E550;
  v3 = off_C916E550;
  *off_C916DF30 = v1;
  sub_C90163A0(v3);
  v4 = next_available_cpu();
  if ( v4 == -1 )
  {
    dprintf("No secondary cpus available\n");
    sub_C90163A4(v2);
    result = 0xFFFFFFFFLL;
  }
```



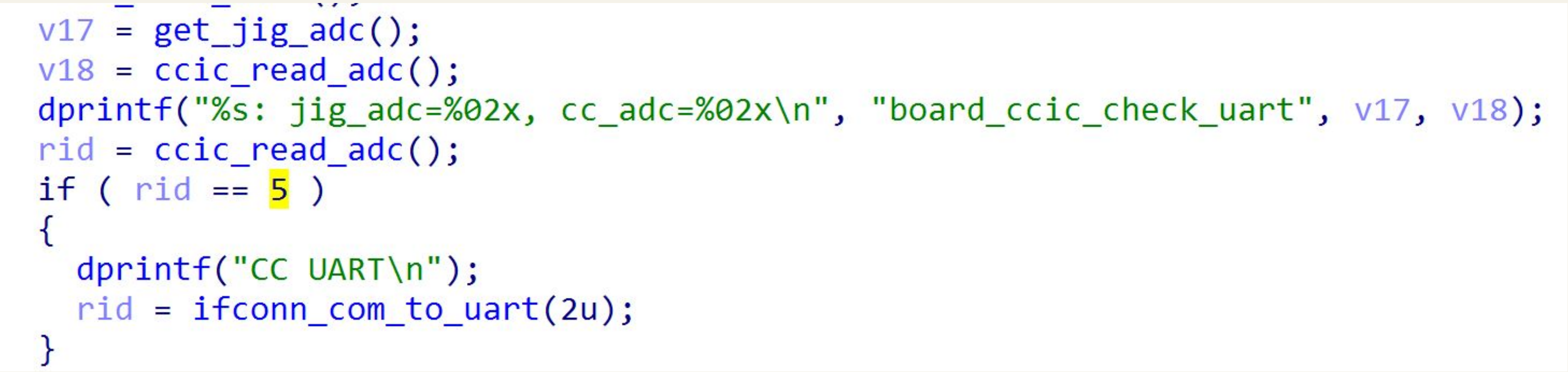
```
__int32 __fastcall next_available_cpu()
{
  __int32 result; // w0

  dprintf("%s: started\n", "next_available_cpu");
  result = current_cpu_id;
  if ( current_cpu_id > 3 )
    return 0xFFFFFFFF;
  ++current_cpu_id;
  return result;
}
```

Potential Exploit Path on S10

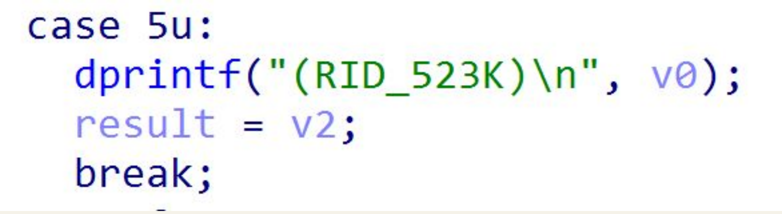
- Uart mode
  - Cmd – smp_test
    - Test Boot up a cpu core and shutdown immediately
    - But count of booted cores will not decrease
  - Cmd – download
    - Enter Odin mode

Potential Exploit Path on S10

- Enter Uart Mode
  - We need a debug cable to make S-Boot detect RID_523K

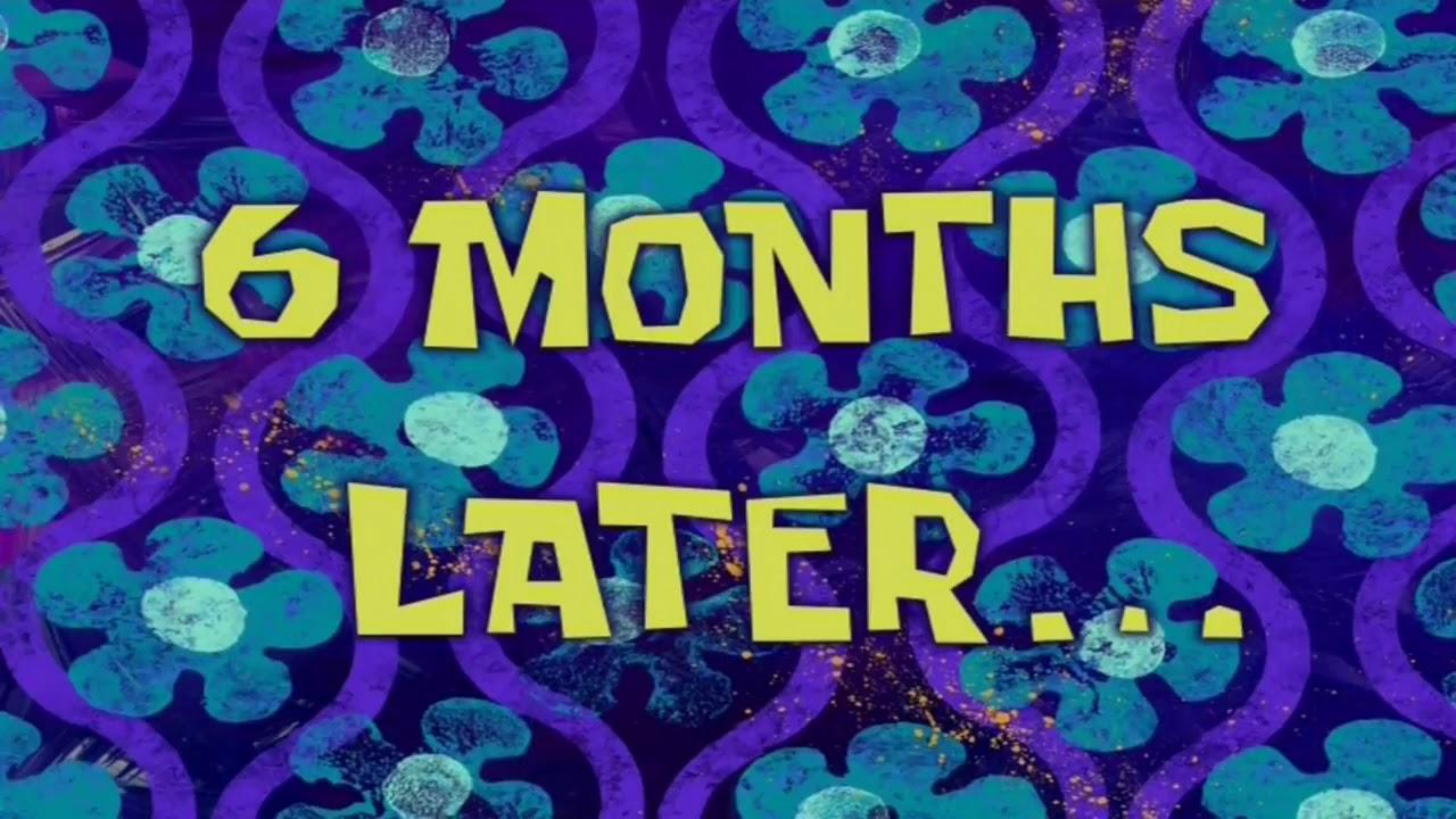


```
v17 = get_jig_adc();
v18 = ccic_read_adc();
dprintf("%s: jig_adc=%02x, cc_adc=%02x\n", "board_ccic_check_uart", v17, v18);
rid = ccic_read_adc();
if ( rid == 5 )
{
  dprintf("CC UART\n");
  rid = ifconn_com_to_uart(2u);
}
```

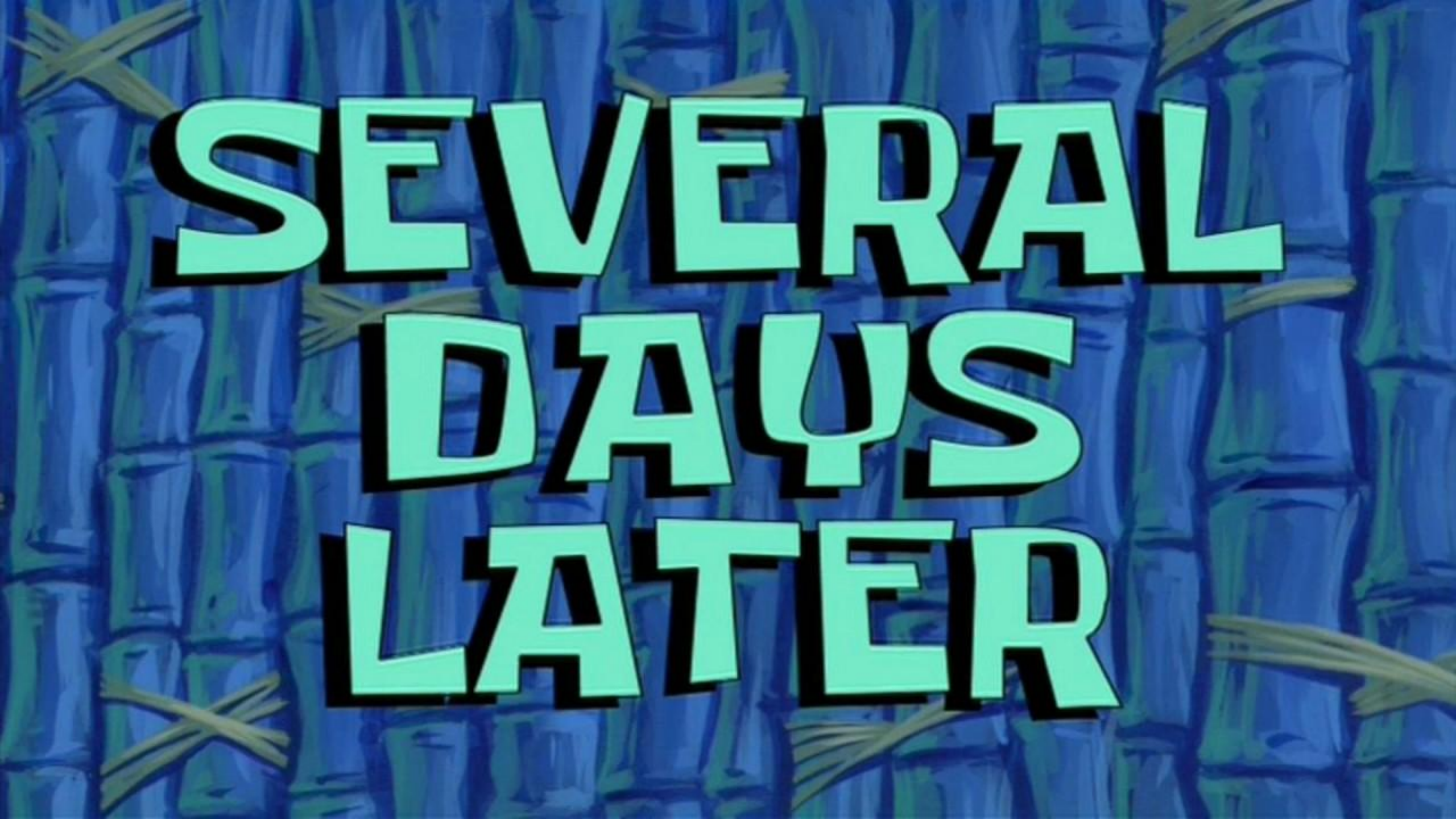


```
case 5u:
  dprintf("(RID_523K)\n", v0);
  result = v2;
  break;
```

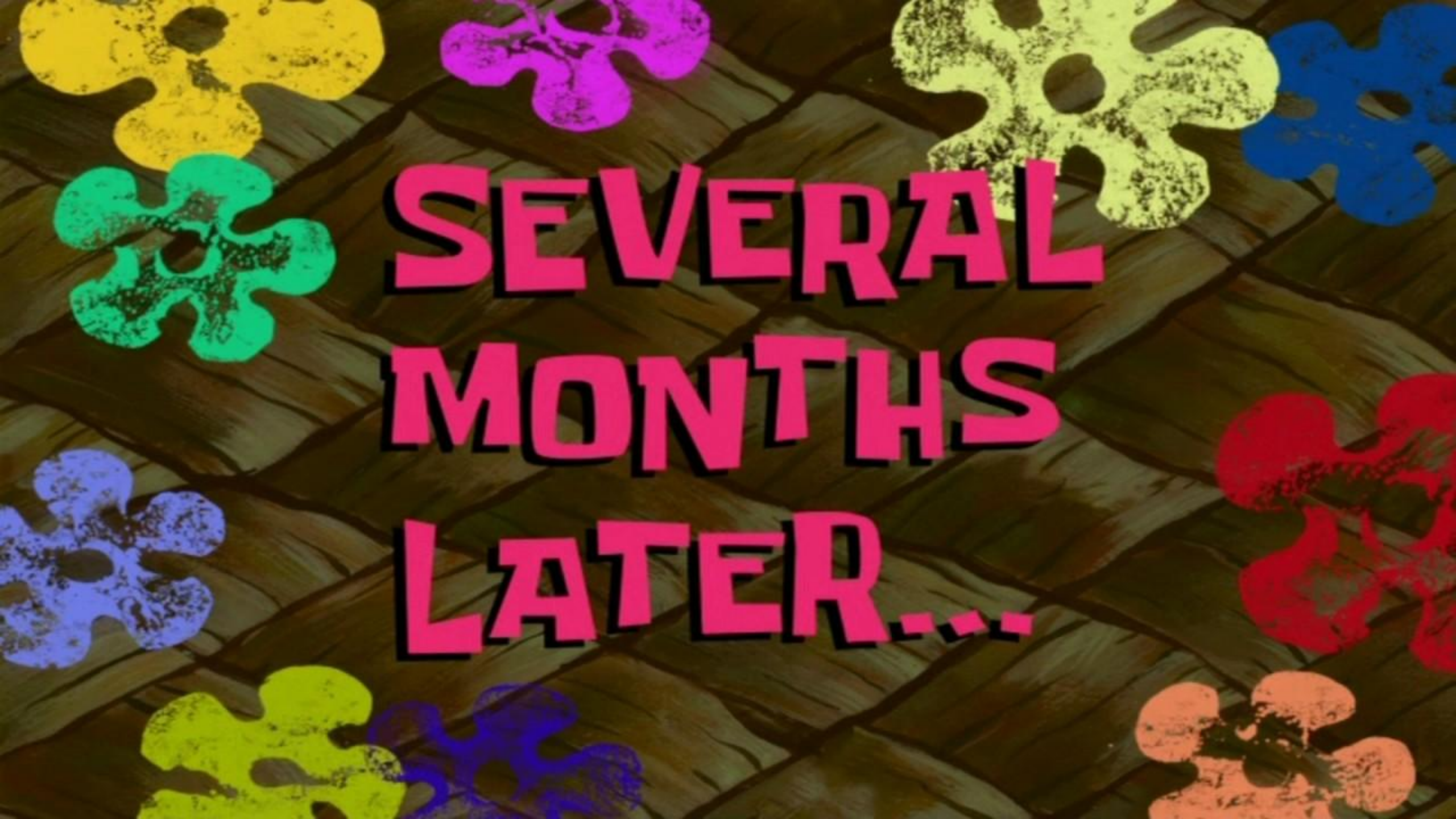
  - Tried TypeC VDM mode, accessory mode, pull-down pull-up resistor
  - All failed


6 MONTHS
LATER...

We reported the bug on Aug 2019

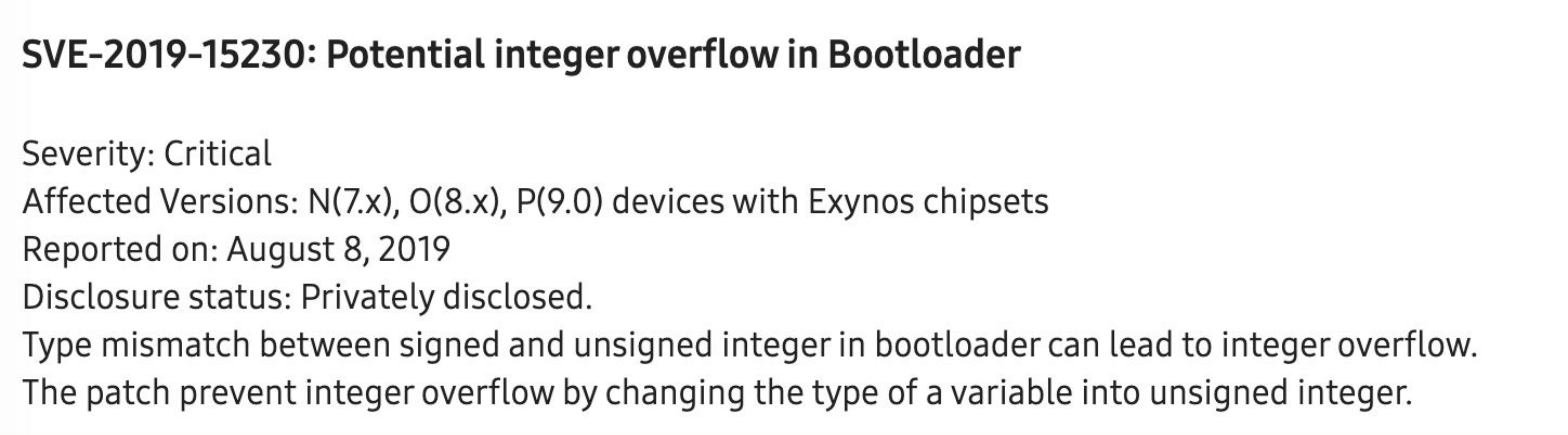

SEVERAL
DAYS
LATER

Result: Duplicated


SEVERAL
MONTHS
LATER...

Patch Note

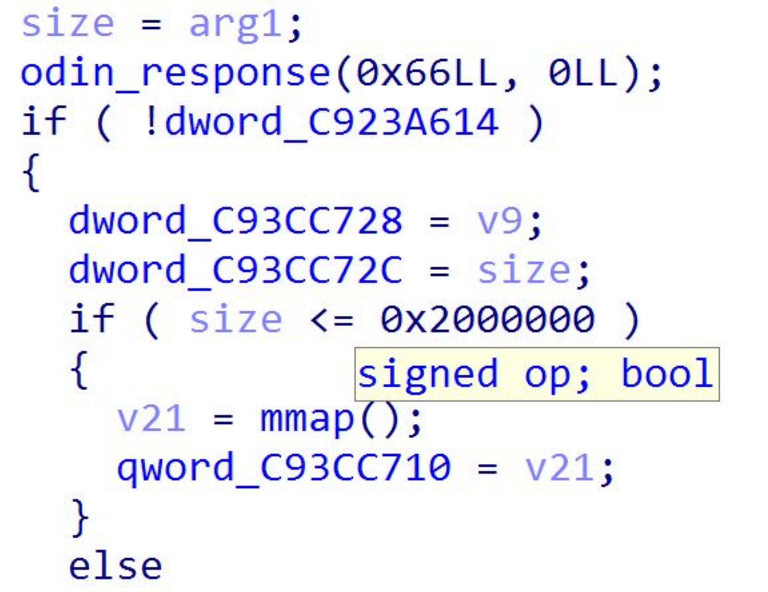
- Samsung Security Update - October 2019
  - SVE-2019-15230 Potential Integer overflow in Bootloader



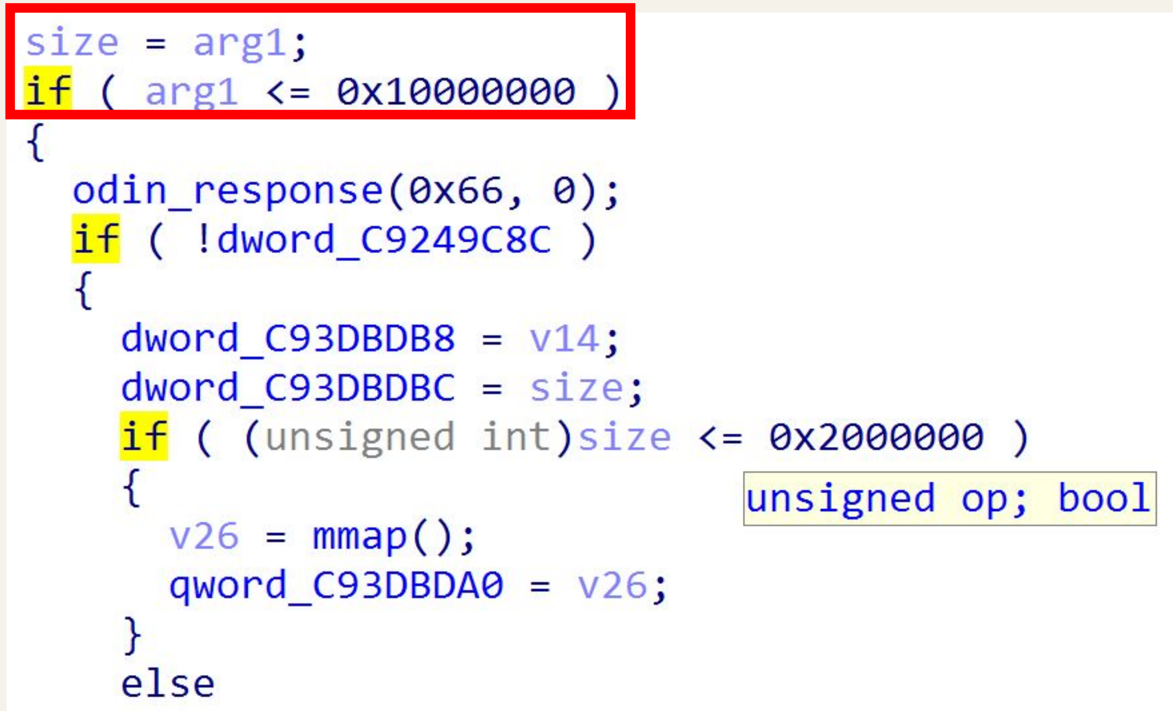
SVE-2019-15230: Potential integer overflow in Bootloader

Severity: Critical
Affected Versions: N(7.x), O(8.x), P(9.0) devices with Exynos chipsets
Reported on: August 8, 2019
Disclosure status: Privately disclosed.
Type mismatch between signed and unsigned integer in bootloader can lead to integer overflow.
The patch prevent integer overflow by changing the type of a variable into unsigned integer.

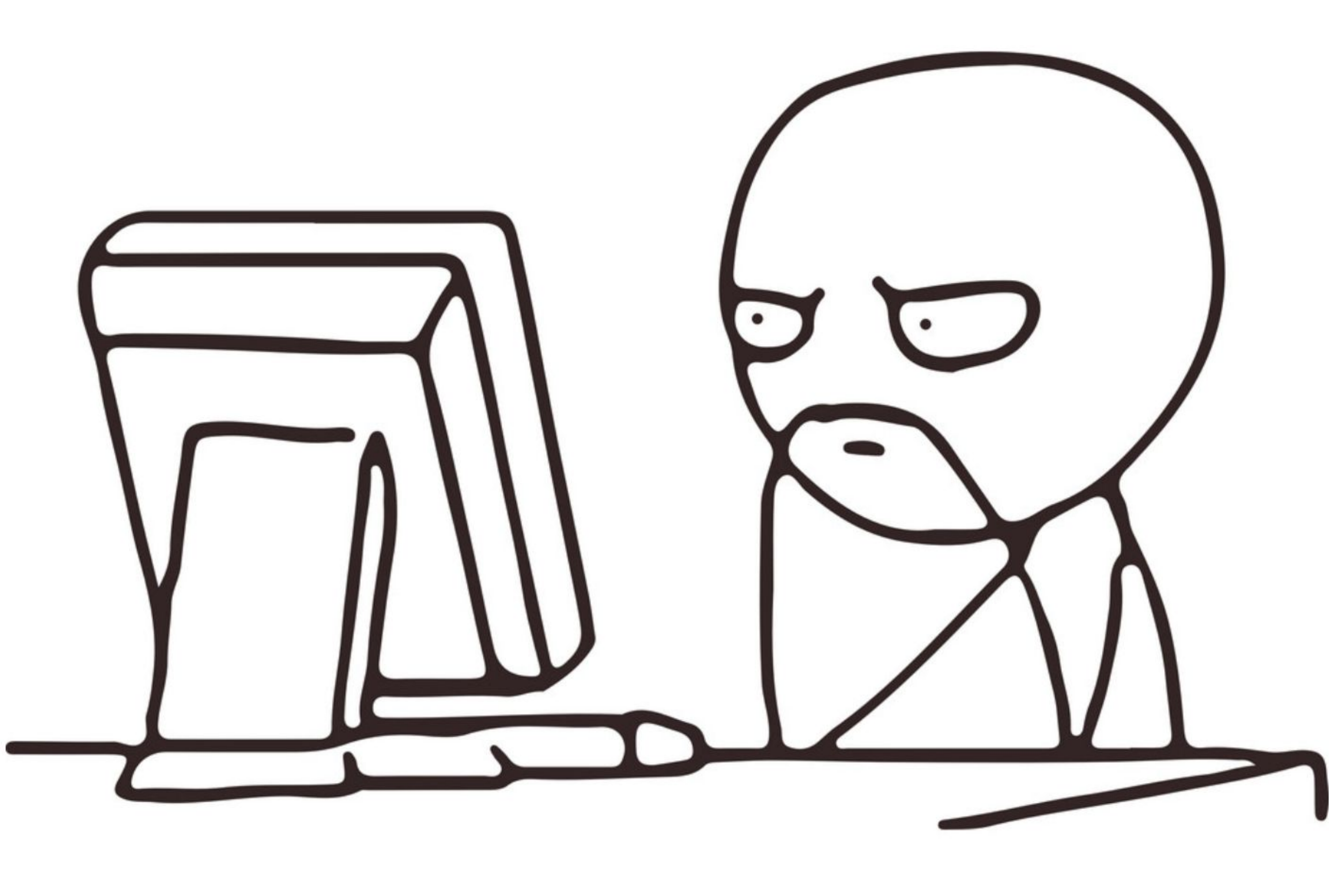
The Patch

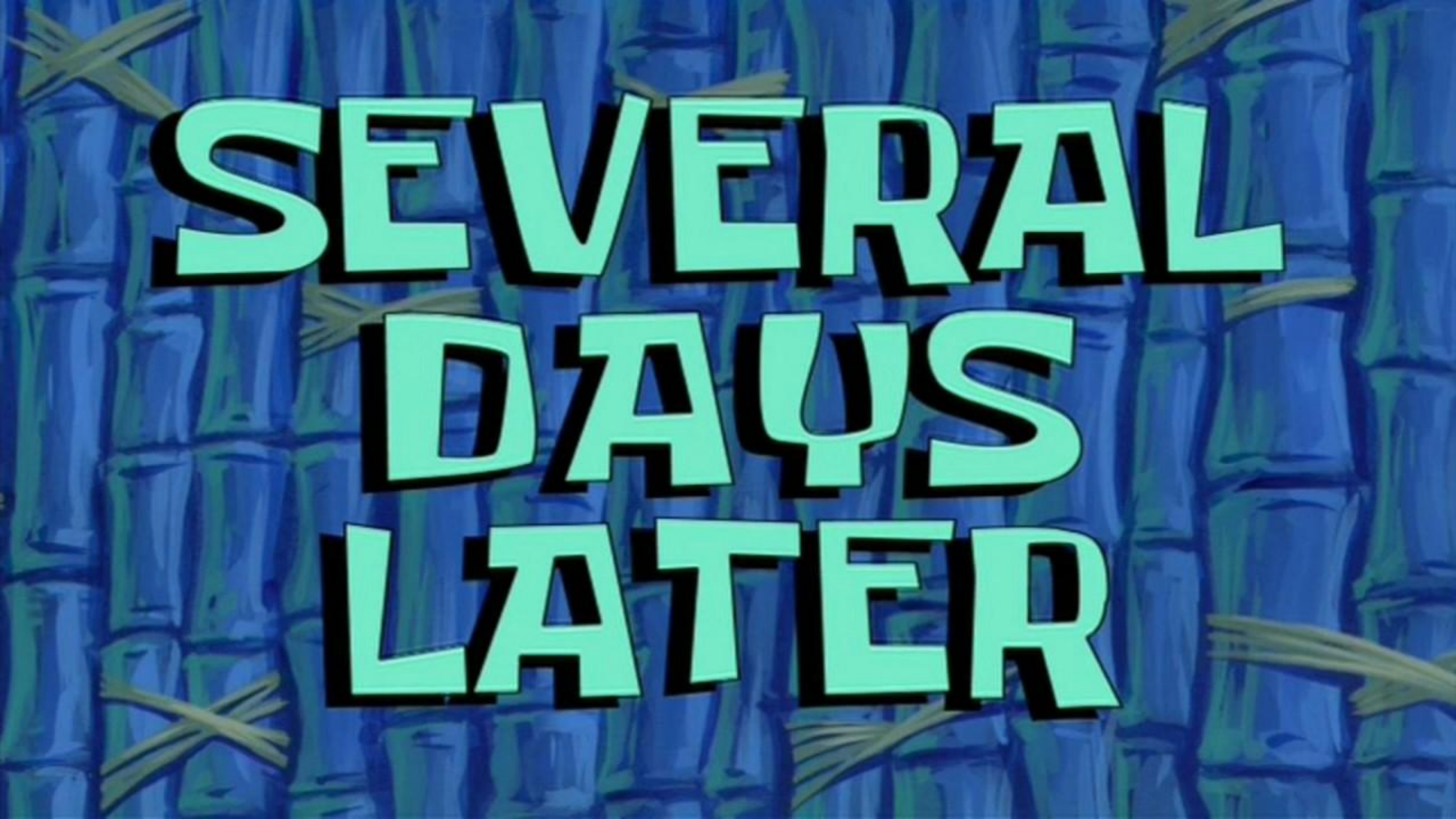


```
size = arg1;
odin_response(0x66LL, 0LL);
if ( !dword_C923A614 )
{
  dword_C93CC728 = v9;
  dword_C93CC72C = size;
  if ( size <= 0x2000000 )
  {
    v21 = mmap();
    qword_C93CC710 = v21;
  }
  else
```



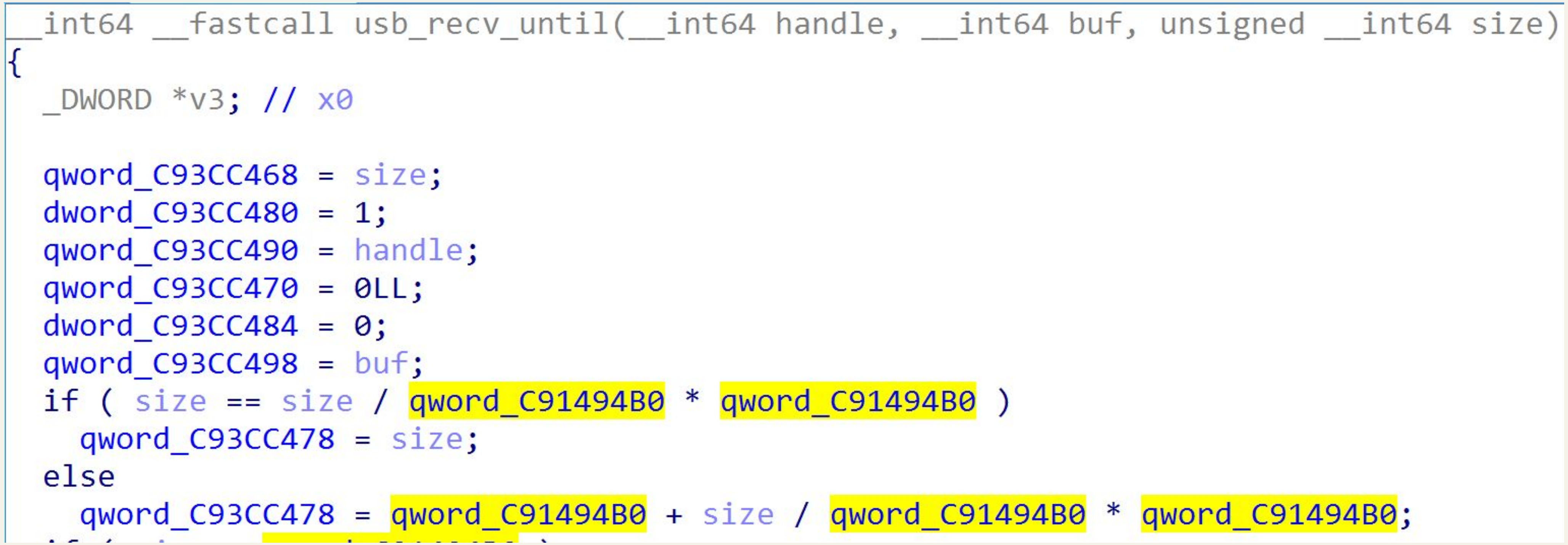
```
size = arg1;
if ( arg1 <= 0x10000000 )
{
  odin_response(0x66, 0);
  if ( !dword_C9249C8C )
  {
    dword_C93DBDB8 = v14;
    dword_C93DBDBC = size;
    if ( (unsigned int)size <= 0x2000000 )
    {
      v26 = mmap();
      qword_C93DBDA0 = v26;
    }
    else
```




SEVERAL
DAYS
LATER

Vulnerability II

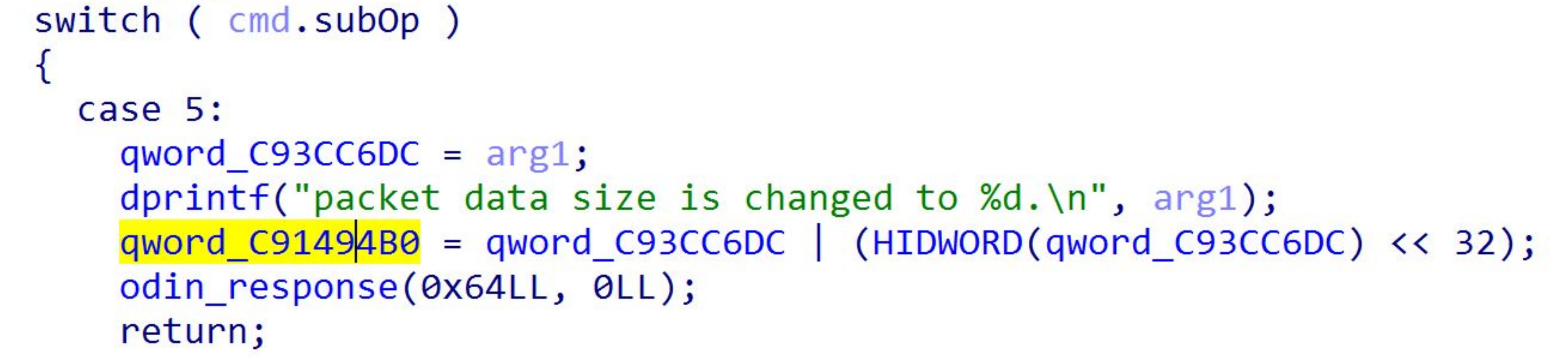
Aligned Size?



```
__int64 __fastcall usb_recv_until(__int64 handle, __int64 buf, unsigned __int64 size)
{
  _DWORD *v3; // x0

  qword_C93CC468 = size;
  dword_C93CC480 = 1;
  qword_C93CC490 = handle;
  qword_C93CC470 = 0LL;
  dword_C93CC484 = 0;
  qword_C93CC498 = buf;
  if ( size == size / qword_C91494B0 * qword_C91494B0 )
    qword_C93CC478 = size;
  else
    qword_C93CC478 = qword_C91494B0 + size / qword_C91494B0 * qword_C91494B0;
```

Odin - packet data size

- We can set packet data size with opCode 0x64, subOp 0x05

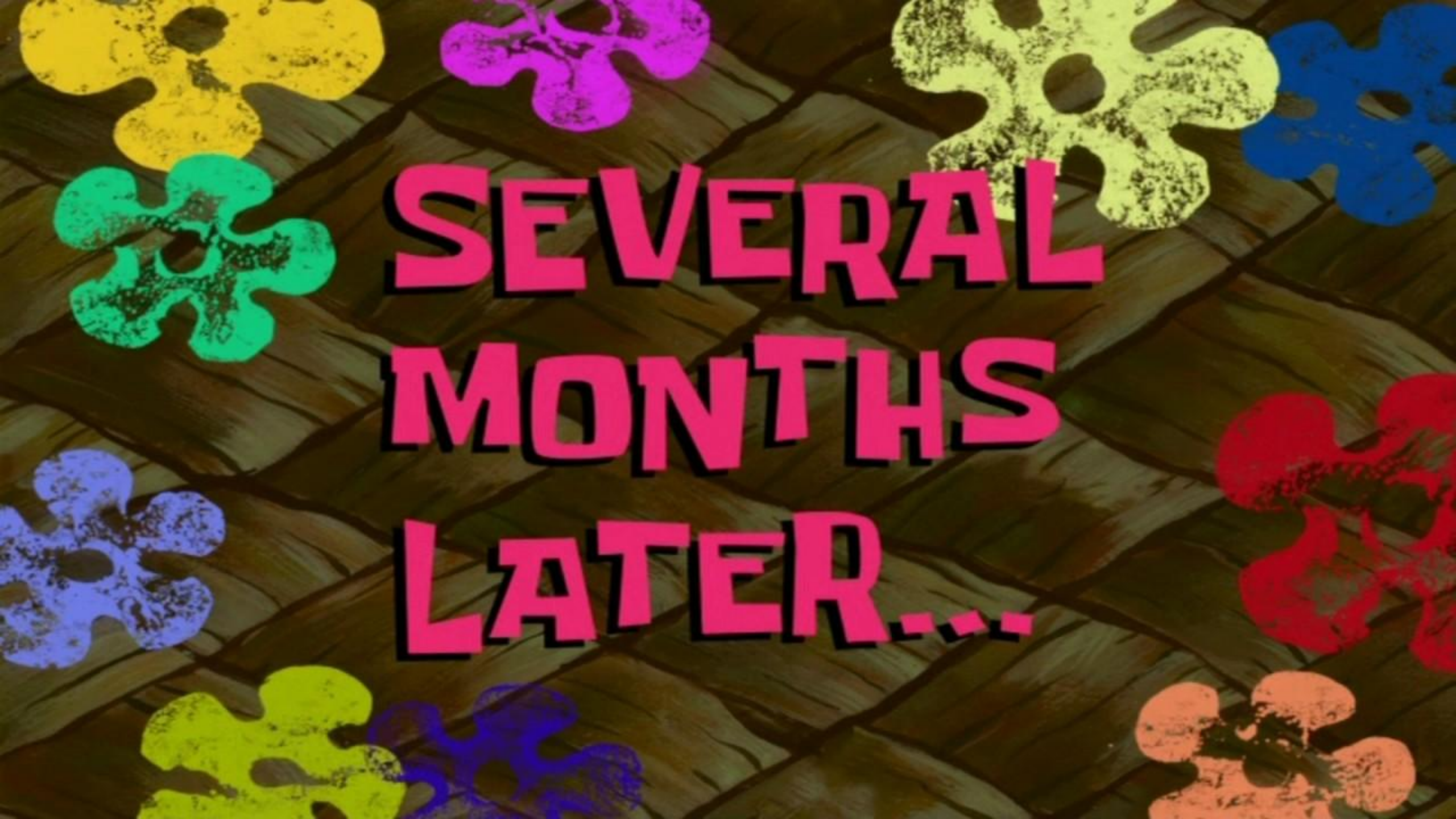


```
switch ( cmd.subOp )
{
  case 5:
    qword_C93CC6DC = arg1;
    dprintf("packet data size is changed to %d.\n", arg1);
    qword_C91494B0 = qword_C93CC6DC | (HIDWORD(qword_C93CC6DC) << 32);
    odin_response(0x64LL, 0LL);
    return;
```

Exploit

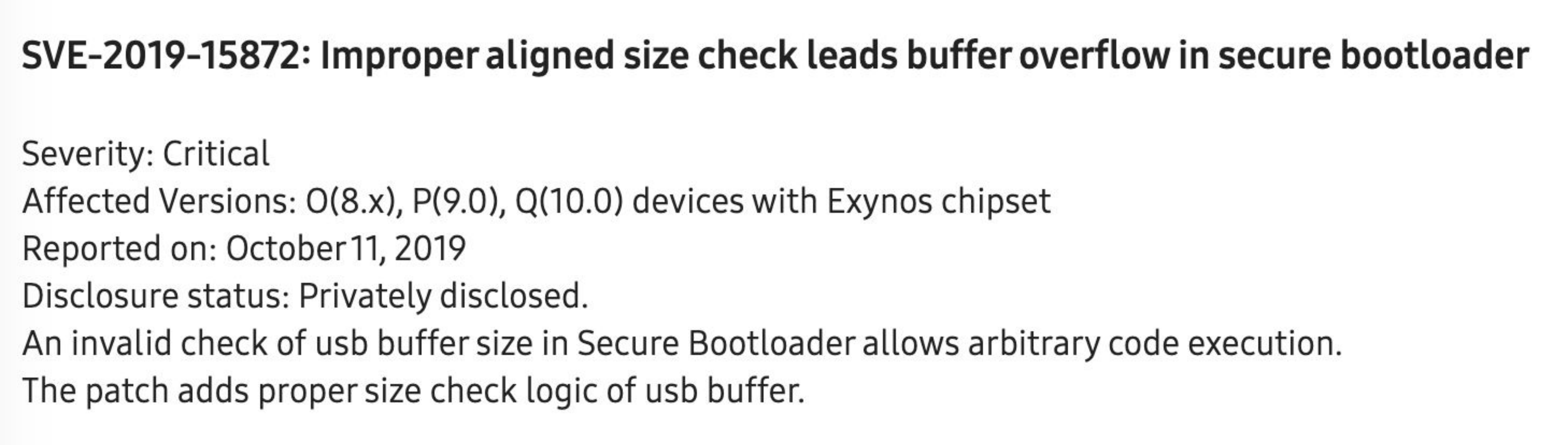
- Bypass the check
- The usb receive size can be larger than 0x10000000 again
- Achieve code execution in the same way as the previous vulnerability

I reported the bug immediately


SEVERAL
MONTHS
LATER...

Patch Note

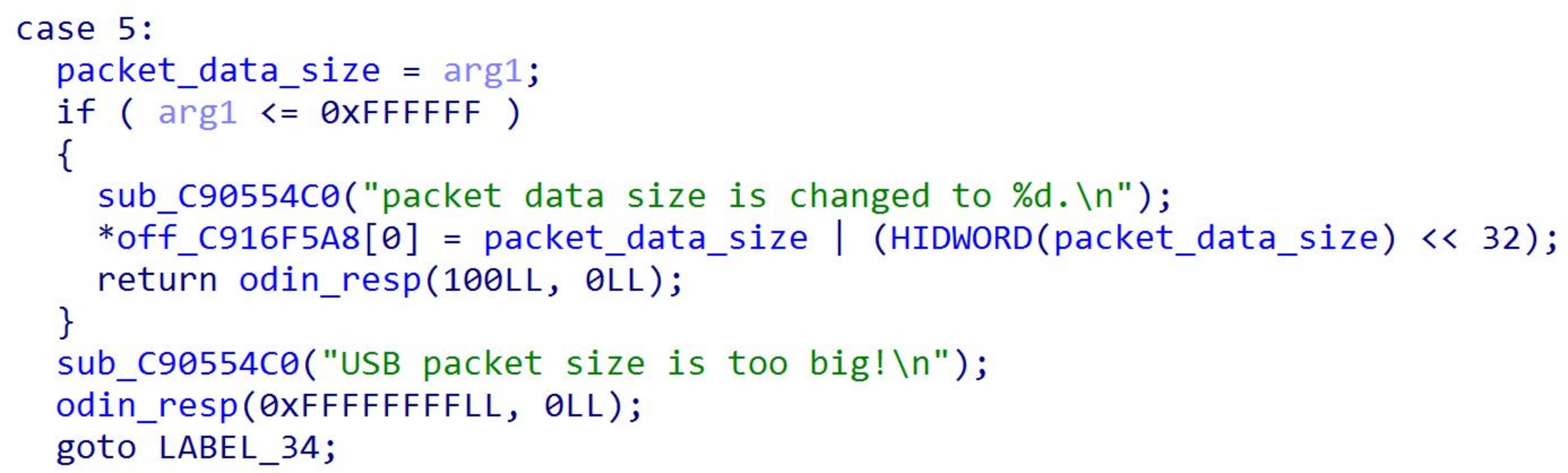
- Samsung Security Update - Jan 2020



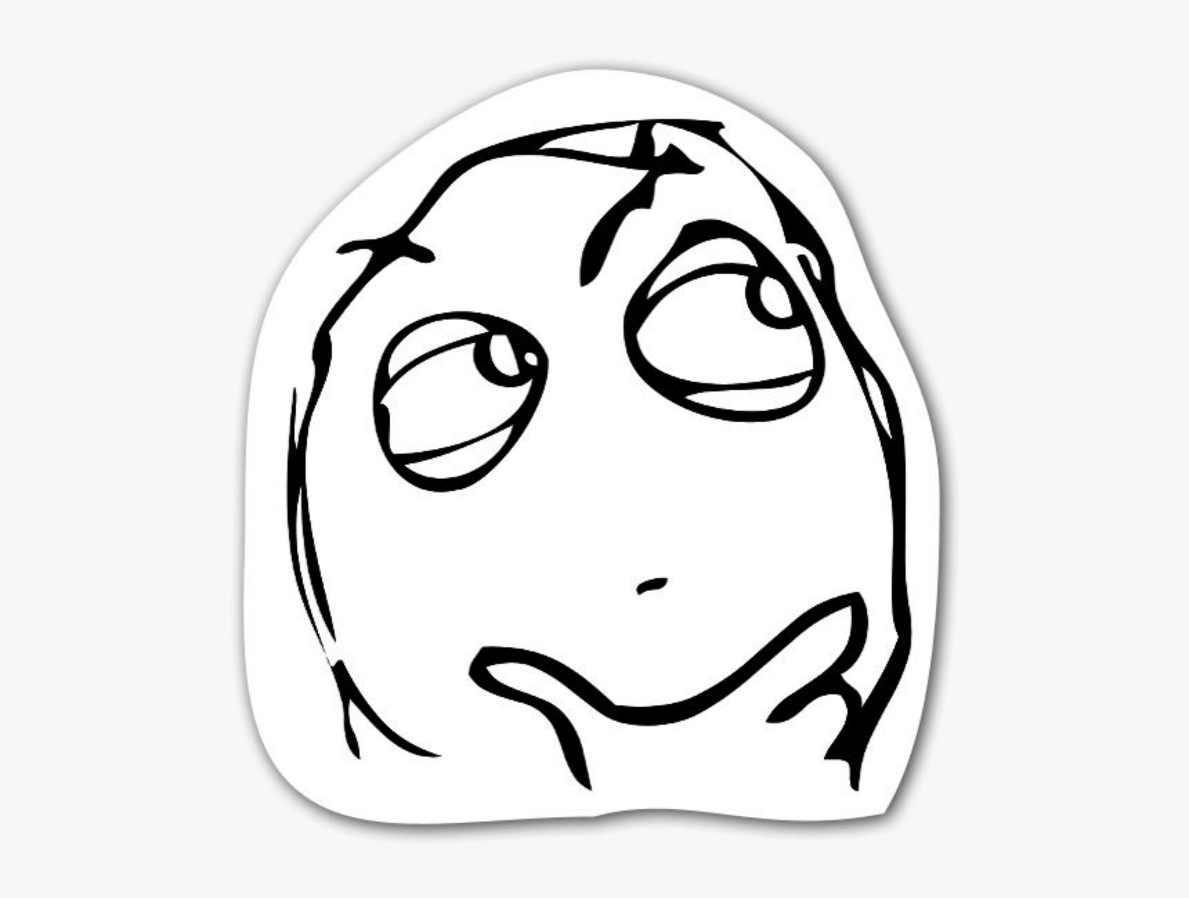
SVE-2019-15872: Improper aligned size check leads buffer overflow in secure bootloader

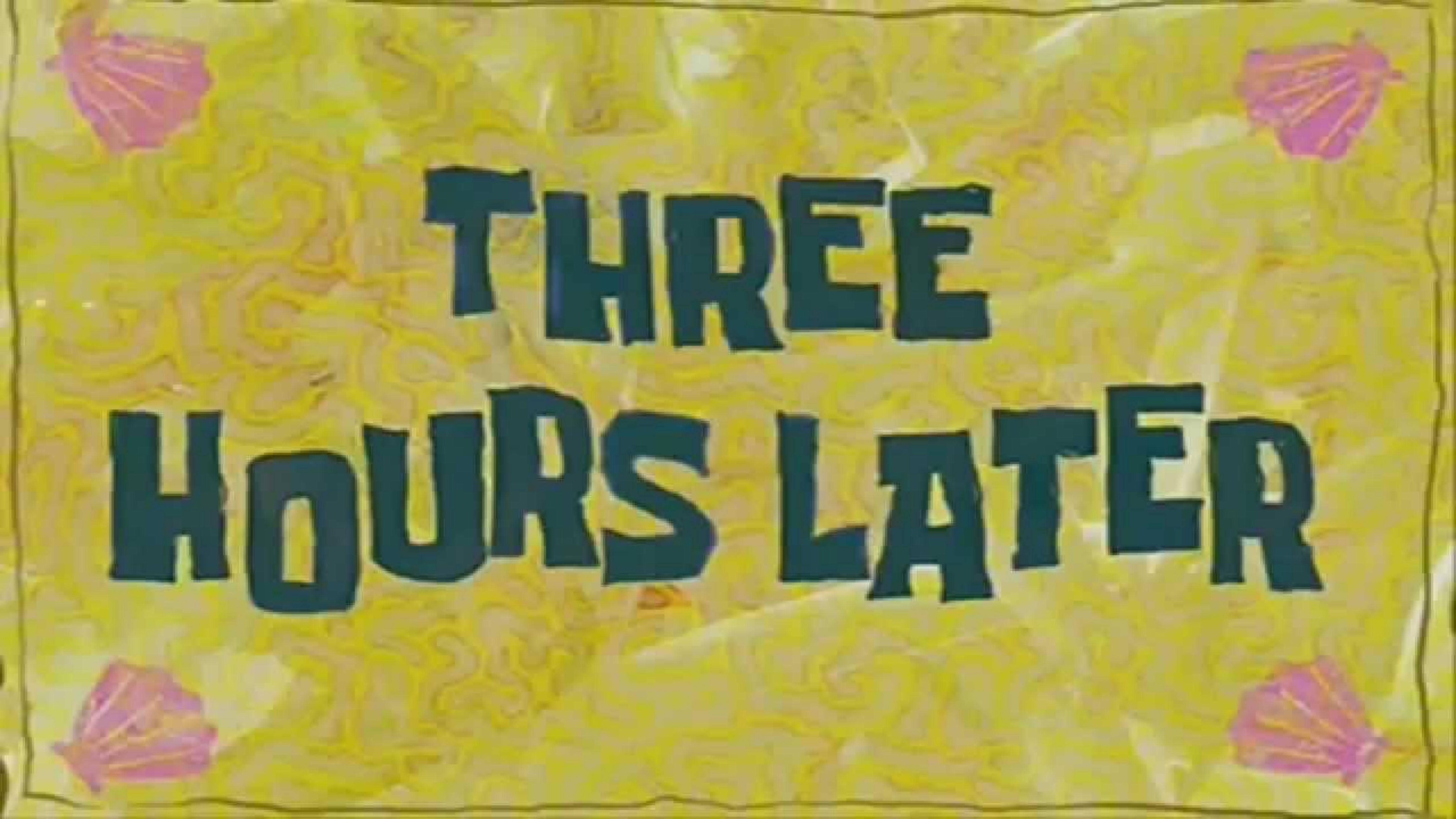
Severity: Critical
Affected Versions: O(8.x), P(9.0), Q(10.0) devices with Exynos chipset
Reported on: October 11, 2019
Disclosure status: Privately disclosed.
An invalid check of usb buffer size in Secure Bootloader allows arbitrary code execution.
The patch adds proper size check logic of usb buffer.

The Patch



```
case 5:
  packet_data_size = arg1;
  if ( arg1 <= 0xFFFFFF )
  {
    sub_C90554C0("packet data size is changed to %d.\n");
    *off_C916F5A8[0] = packet_data_size | (HIDWORD(packet_data_size) << 32);
    return odin_resp(100LL, 0LL);
  }
  sub_C90554C0("USB packet size is too big!\n");
  odin_resp(0xFFFFFFFFLL, 0LL);
  goto LABEL_34;
```



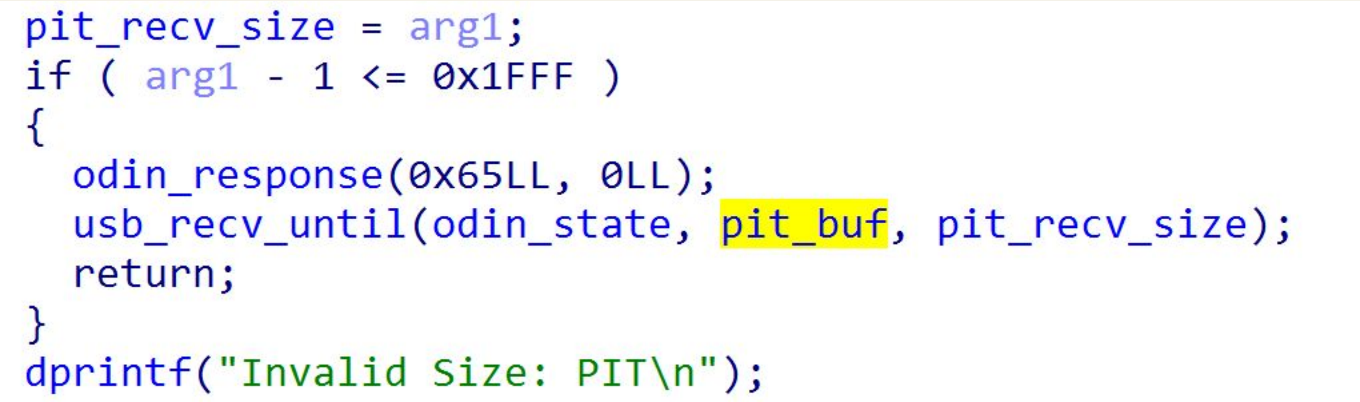

THREE
HOURS LATER

Vulnerability III

ODIN – PIT flash command

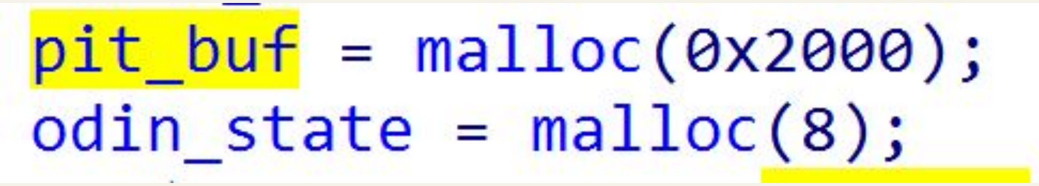
- opCode = 0x65

- PIT is very small, odin store it to heap buffer



```
pit_recv_size = arg1;
if ( arg1 - 1 <= 0x1FFF )
{
  odin_response(0x65LL, 0LL);
  usb_recv_until(odin_state, pit_buf, pit_recv_size);
  return;
}
dprintf("Invalid Size: PIT\n");
```

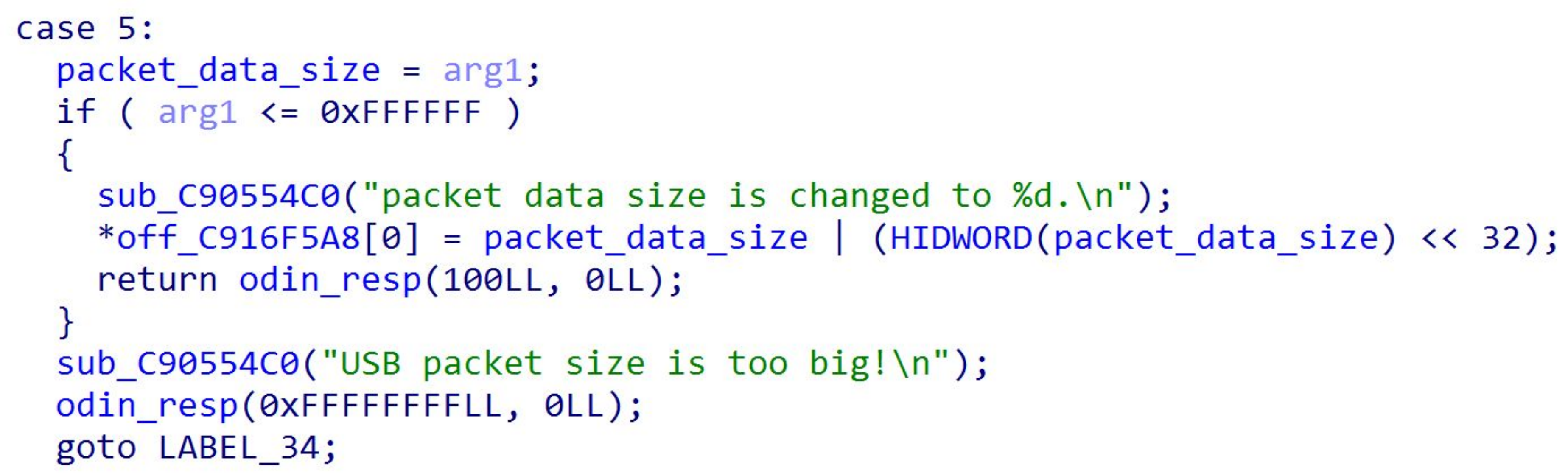
- With the size 0x2000



```
pit_buf = malloc(0x2000);
odin_state = malloc(8);
```

The patch of vulnerability II

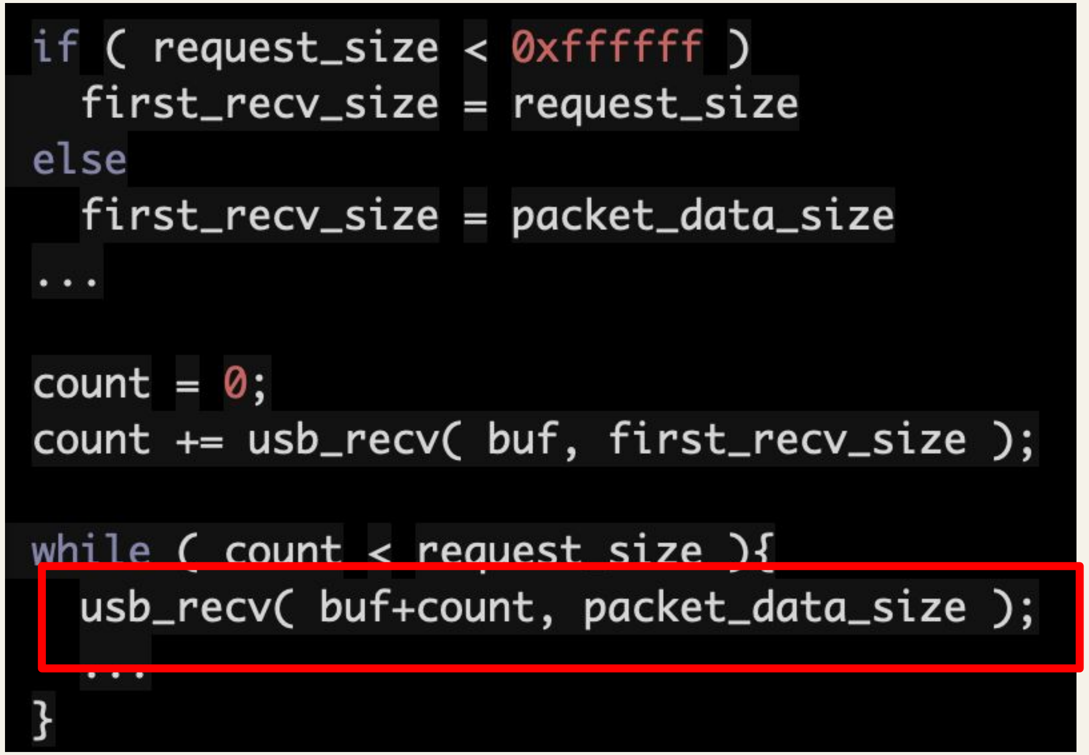
- Size of packet data can be upto 0xFFFFFF
    - > 0x2000 => heap overflow



```
case 5:
  packet_data_size = arg1;
  if ( arg1 <= 0xFFFFFF )
  {
    sub_C90554C0("packet data size is changed to %d.\n");
    *off_C916F5A8[0] = packet_data_size | (HIDWORD(packet_data_size) << 32);
    return odin_resp(100LL, 0LL);
  }
  sub_C90554C0("USB packet size is too big!\n");
  odin_resp(0xFFFFFFFFLL, 0LL);
  goto LABEL_34;
```

Pseudo code - receive data

- This is a pseudocode representation of the receive operation



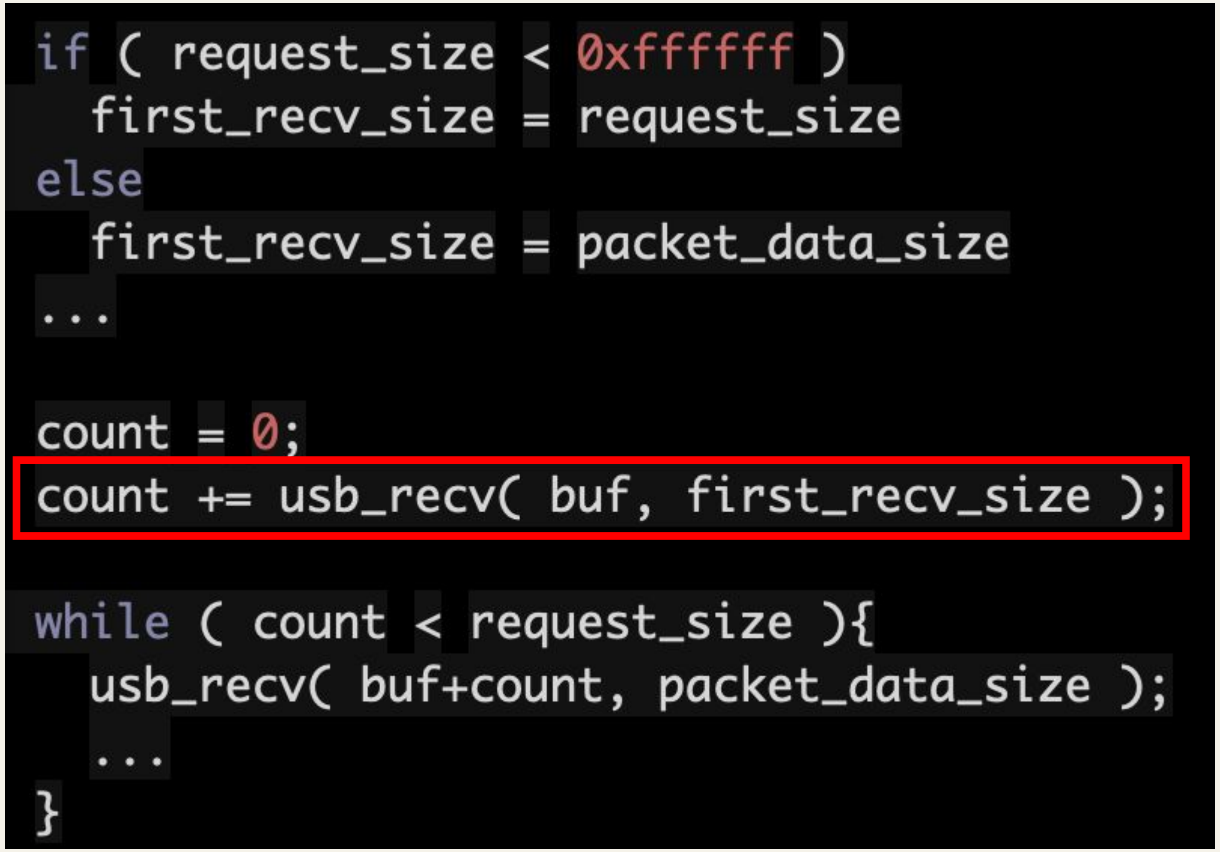
```
if ( request_size < 0xffffff )
  first_recv_size = request_size
else
  first_recv_size = packet_data_size
...

count = 0;
count += usb_recv( buf, first_recv_size );

while ( count < request_size ){
  usb_recv( buf+count, packet_data_size );
  ...
}
```

- In our test, the usb_recv function will receive until the passed size is reached
    - Even if we send data with a huge interval

We thought this was un-exploitable, so I stuck to vulnerability I

How About Interrupting the USB

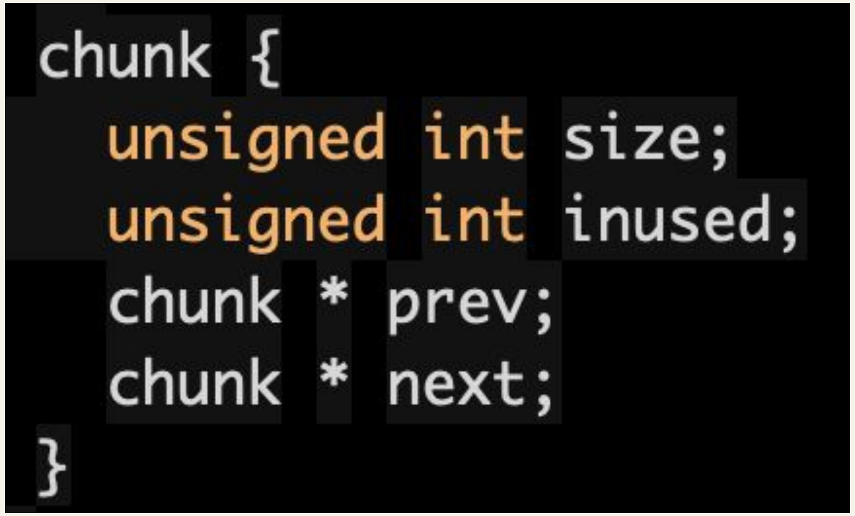
- Remove and Re-insert the USB cable
- the usb_recv returns with insufficient size



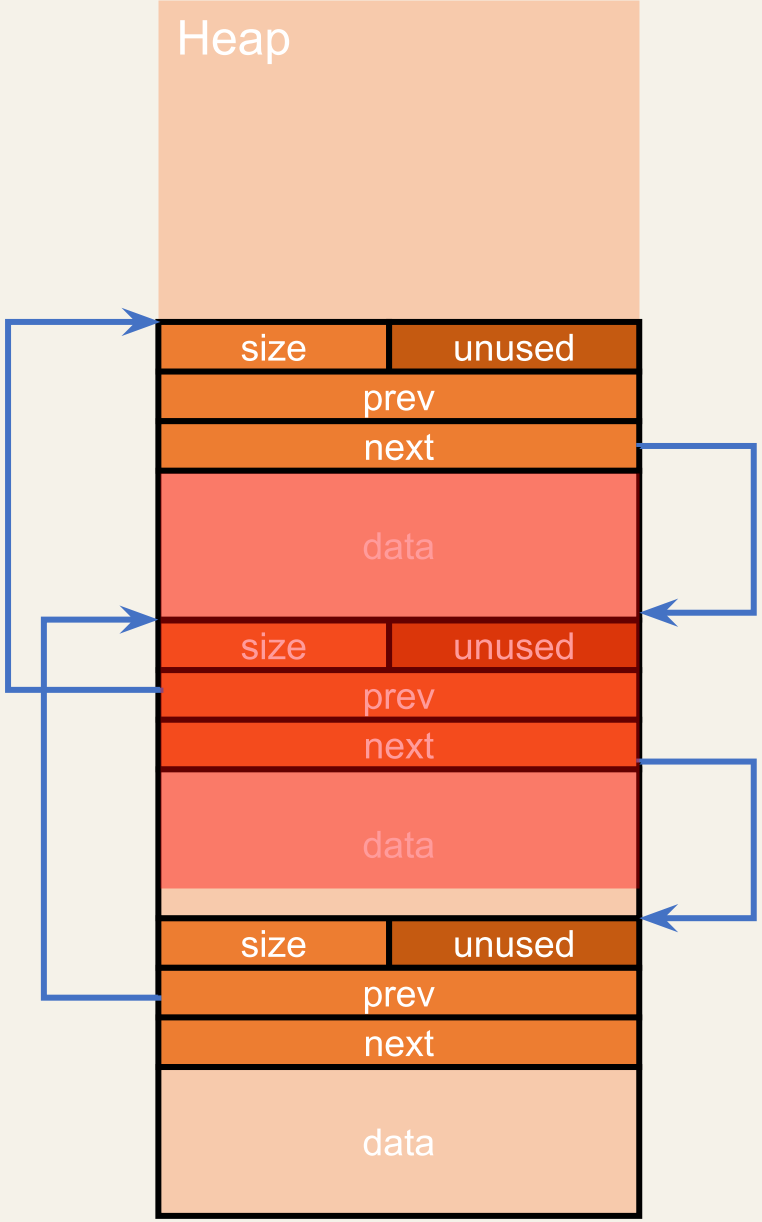
```
if ( request_size < 0xffffff )
   first_recv_size = request_size
else
   first_recv_size = packet_data_size
...

count = 0;
count += usb_recv( buf, first_recv_size );

while ( count < request_size ){
   usb_recv( buf+count, packet_data_size );
   ...
}
```

Heap overflow

- We can overwrite the metadata of heap chunk
- House of Spirit

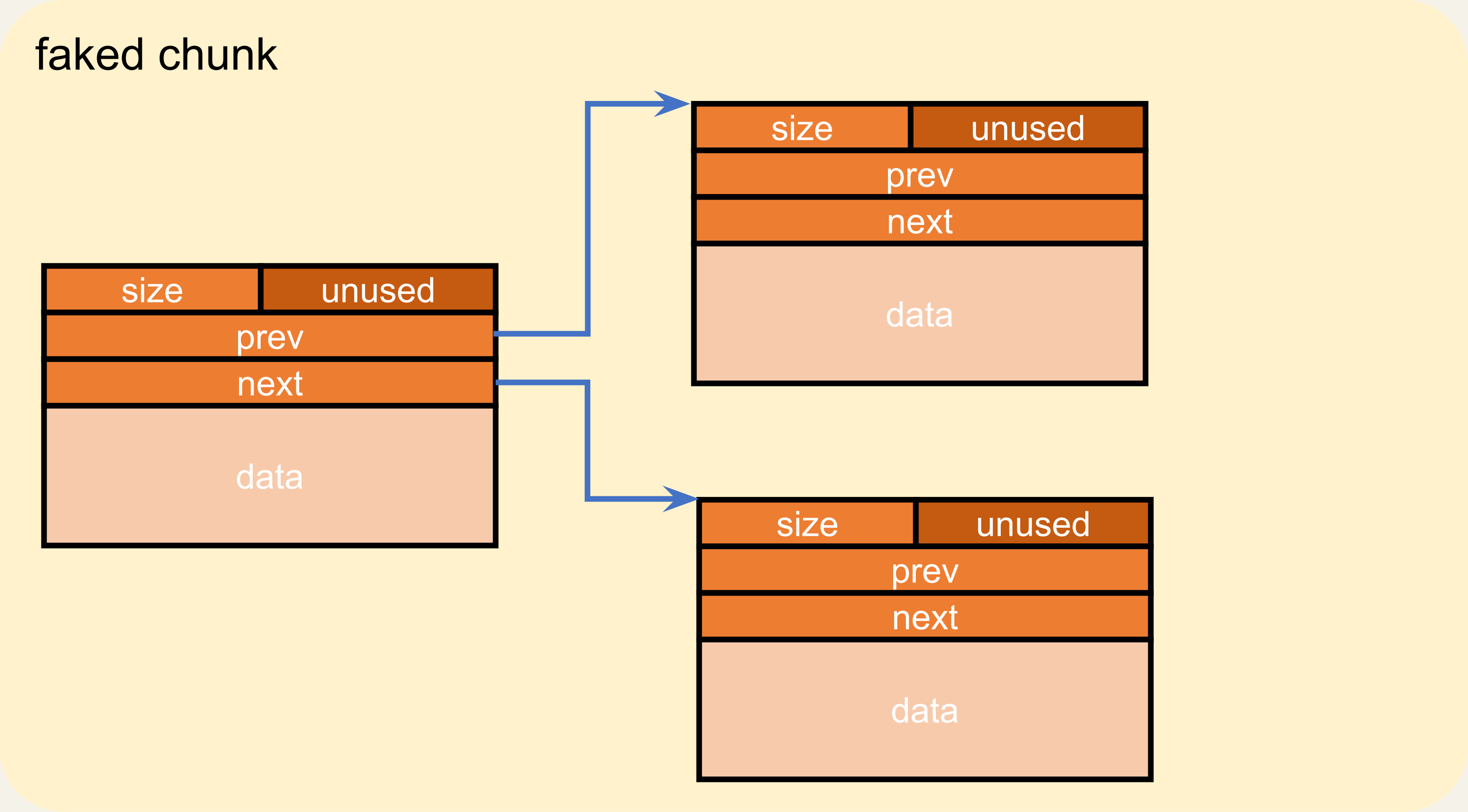


```
chunk {
  unsigned int size;
  unsigned int inused;
  chunk * prev;
  chunk * next;
}
```



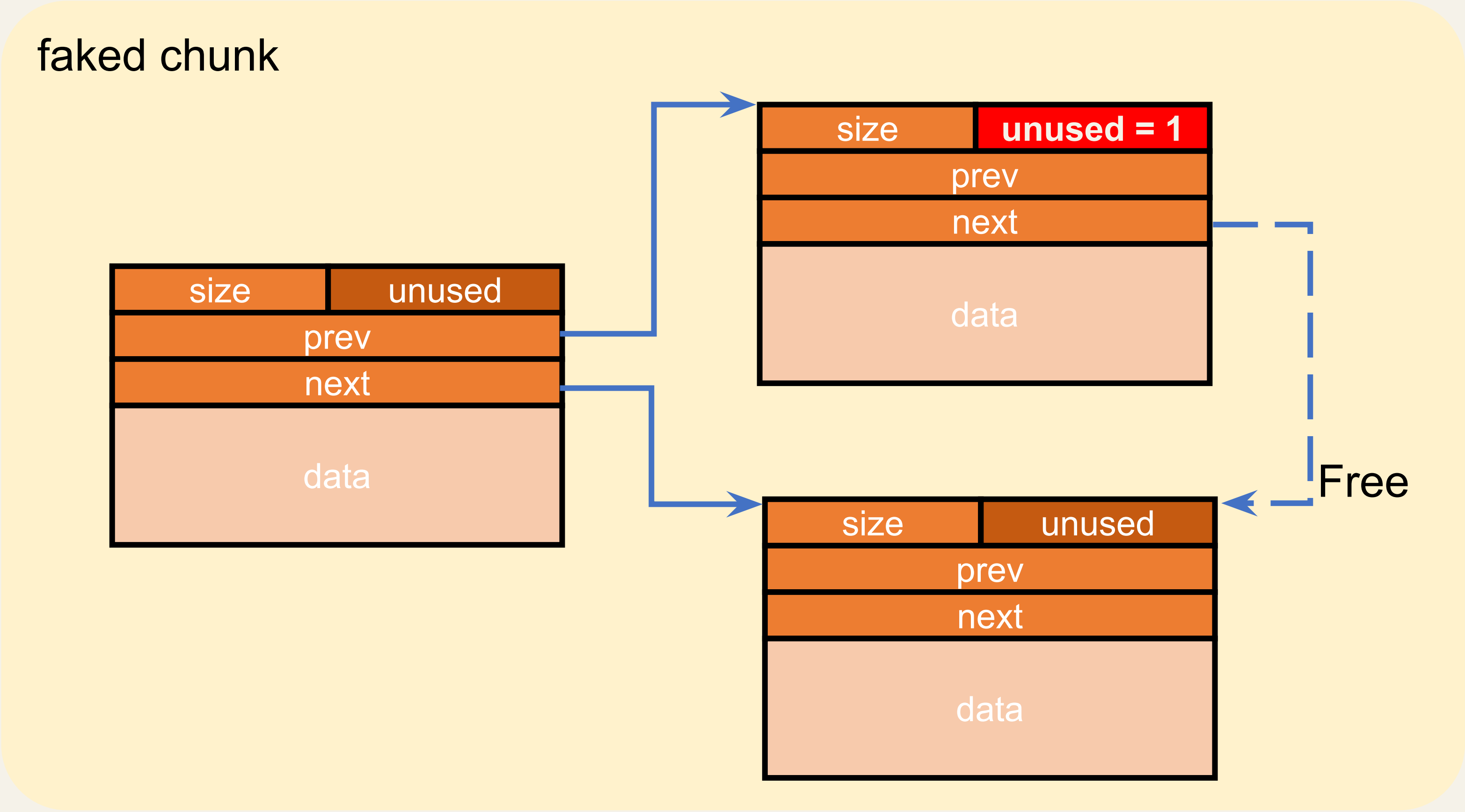
Fake Chunk

No check for double linked list



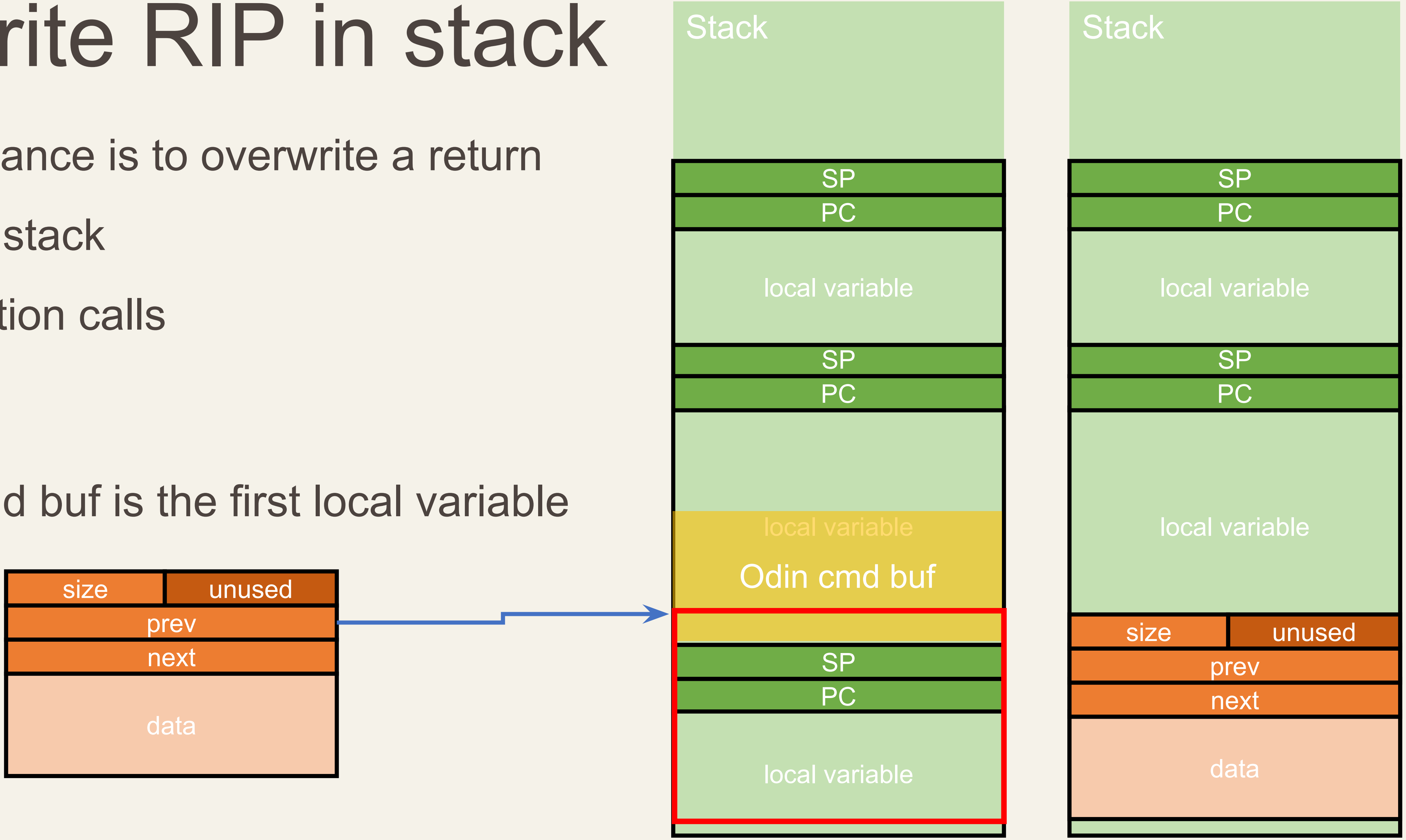
Limited Overwrite Data

- *prev + 4 = 1
- It aarch64, integer 64 bit
    - Code at 0xC9000000
    - We can not point to
        - Got
        - Function pointer



Overwrite RIP in stack

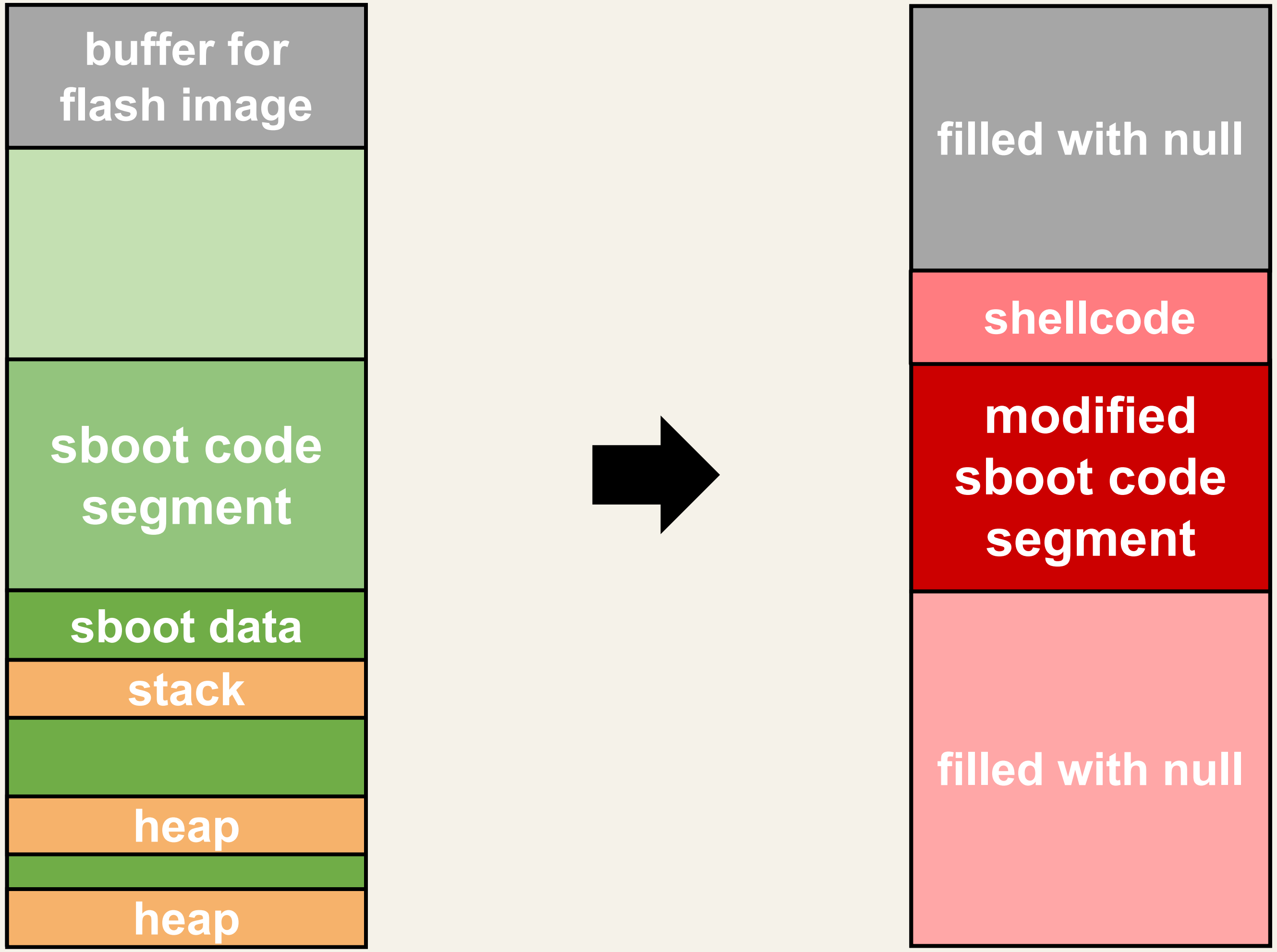
- The only chance is to overwrite a return address on stack
- Only 3 function calls
- Fortunately
  - Odin cmd buf is the first local variable



After Code Execution in S-boot

Boot the phone

- We smashed the stack & heap
- Hard to recover
- Call the boot functions one by one



Skip Trustzone related call

- We only have EL1 privilege
- Some smc call to trustzone can not call twice
- Skip the smc call and set the related parameter

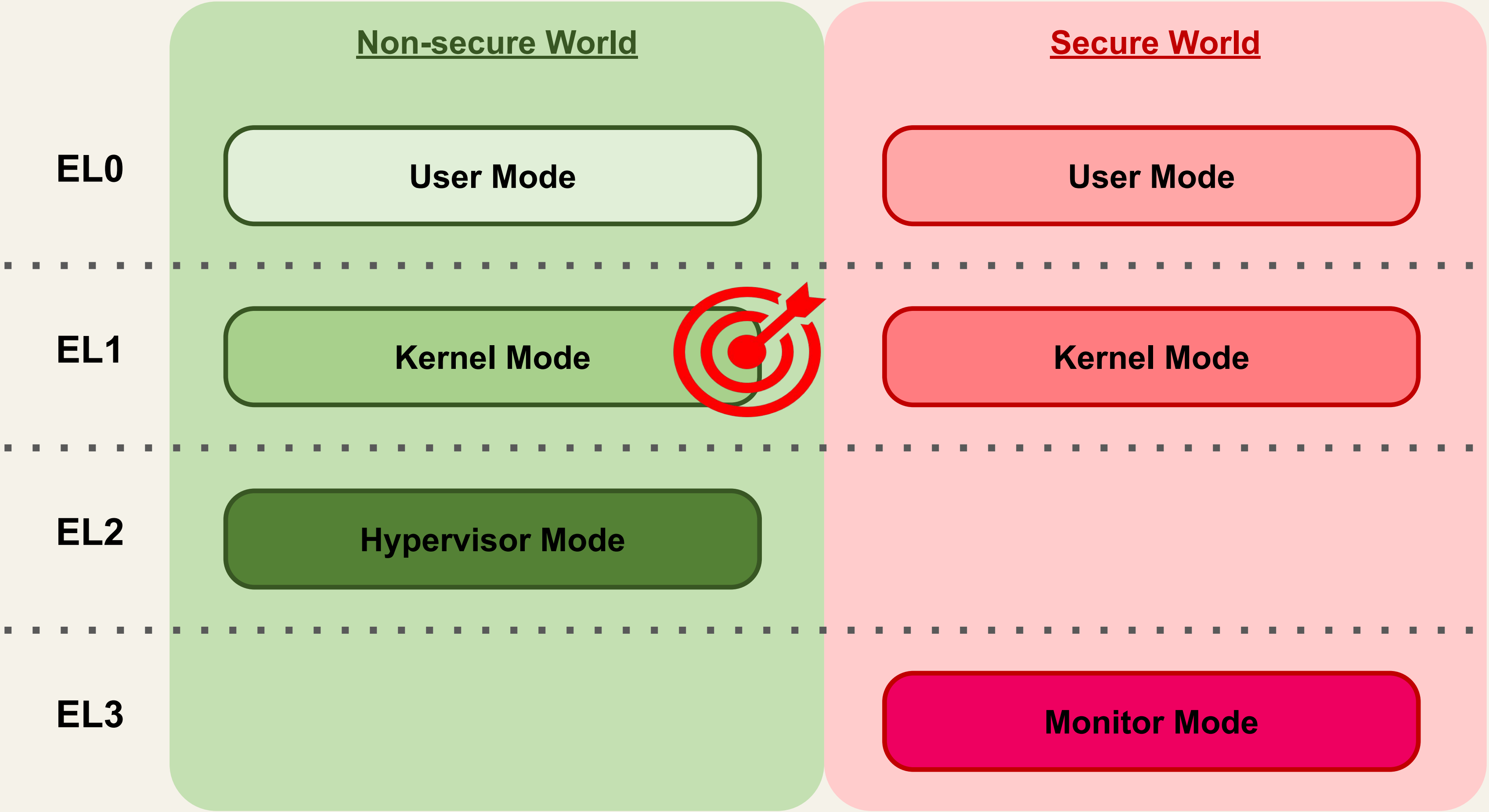
Load Custom Kernel

- After loading kernel to memory (the function cmd_load_kernel)
- Replace the image with custom one
- Booting the kernel (call the function cmd_boot)

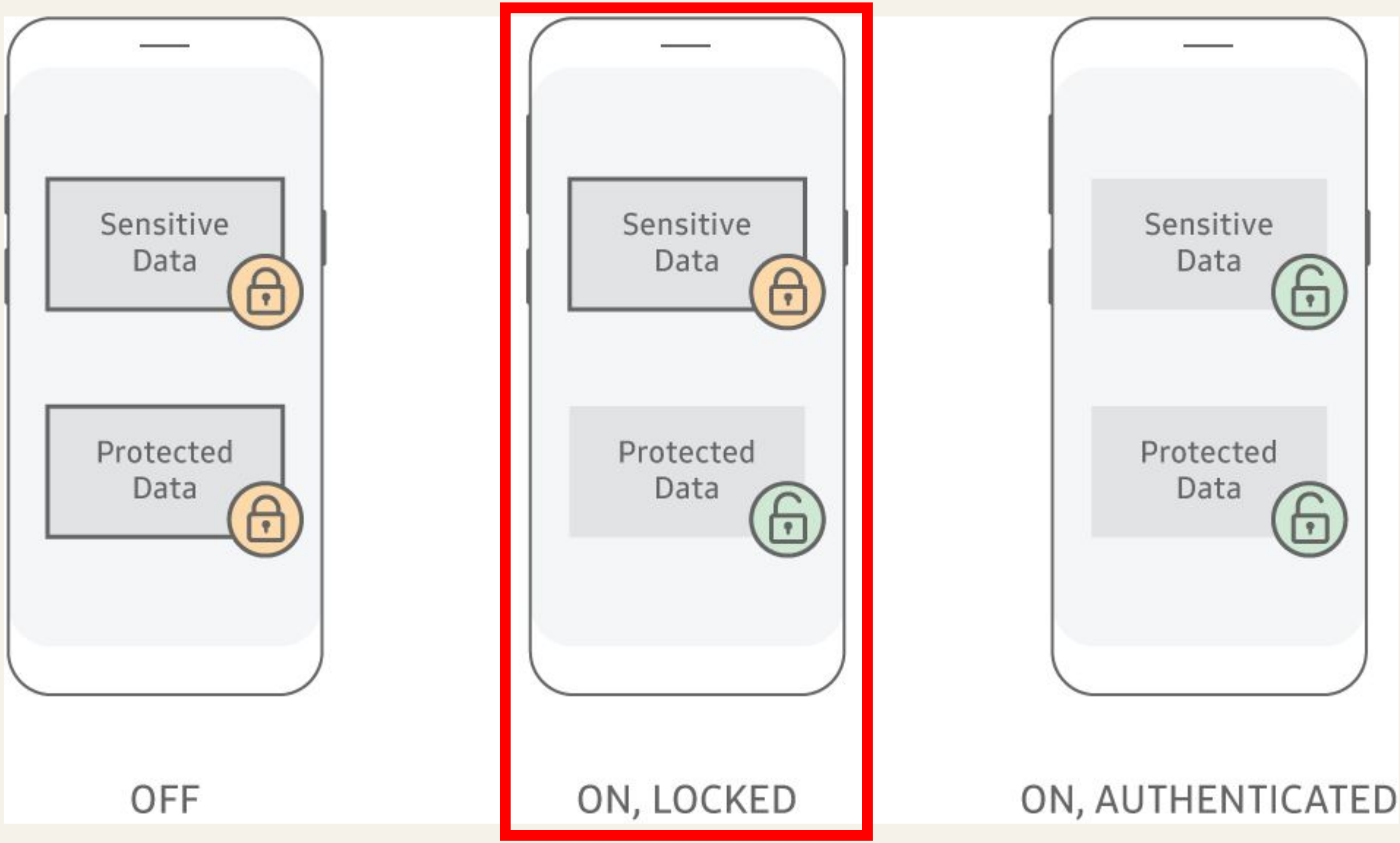
Exploit

- Set the size of packet data to a big number
- Send Odin PIT flash command
- Send payload after Interrupt the usb_recv(), leads to heap overflow
- Send Another Odin command to trigger malloc & free the buffer
- Overwrite RIP on stack, jump to shellcode
    - Re-init heap and stack
    - Continue booting
    - Before boot into kernel, replace the boot image

We got el1 in normal world



But the phone is still locked

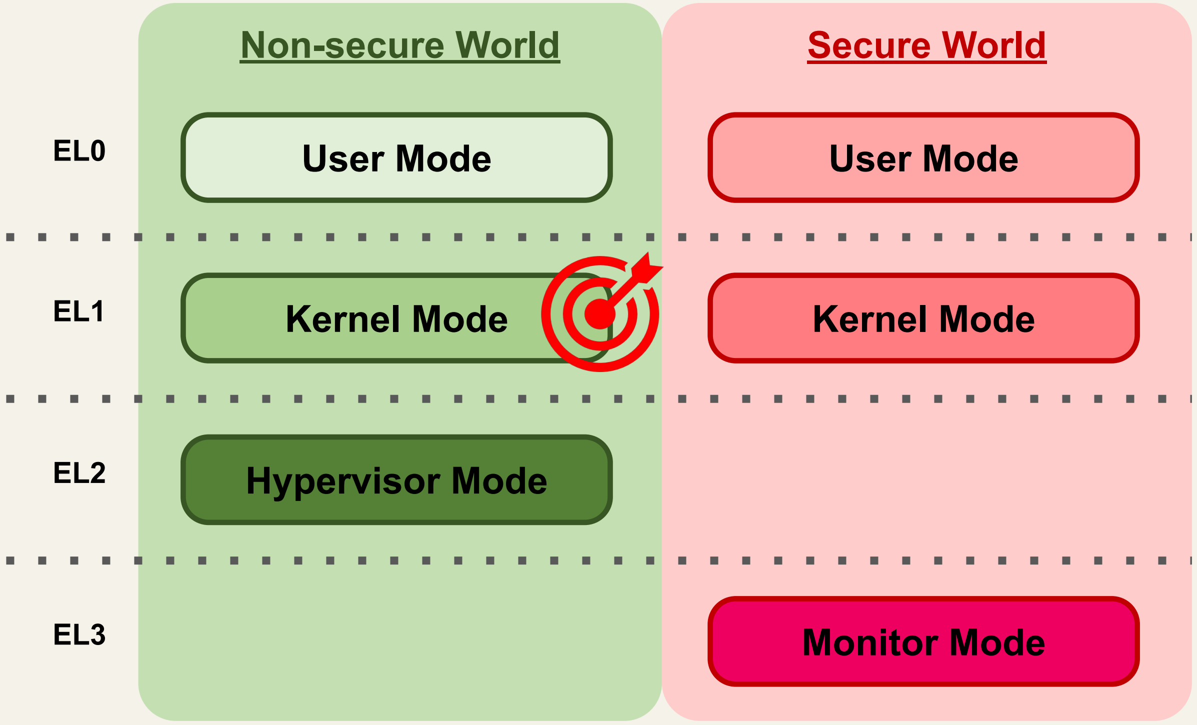


Can not read sensitive data

- Storage is still encrypted if we didn’t provide the screen passcode
  - Encryption key can only be decrypted in the gatekeeper trustlet
- Some data in trustlet can not be reached

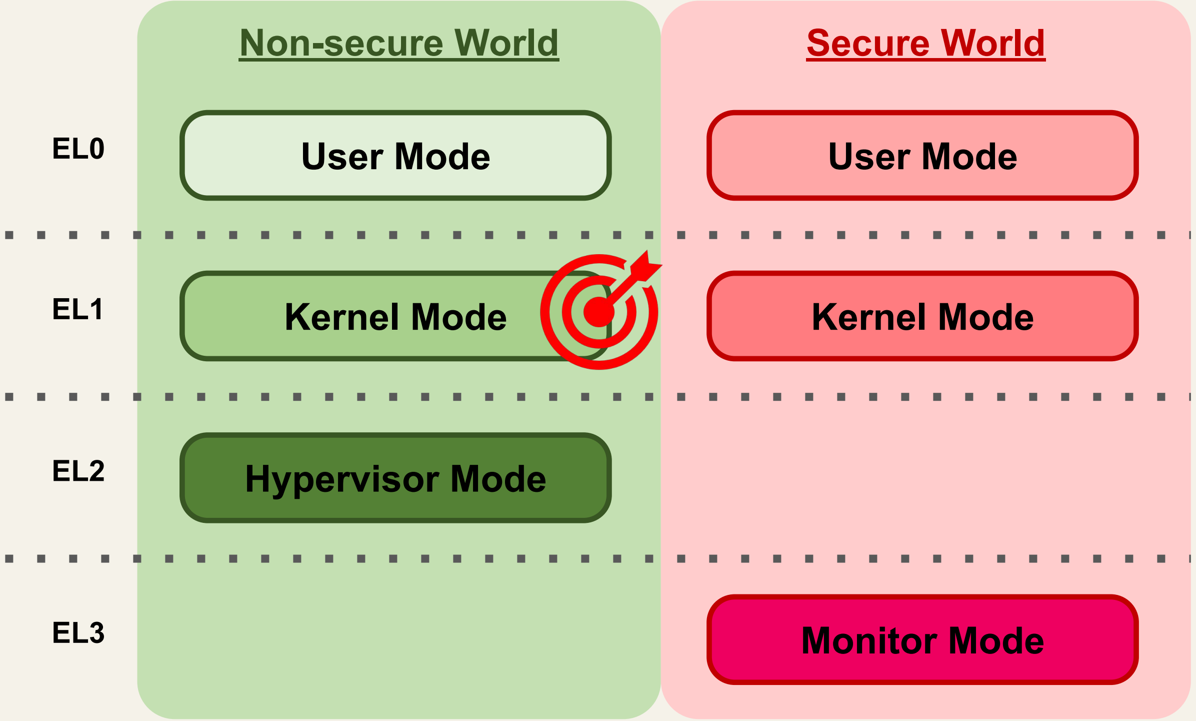
Man in the Non-secure EL1

- Wait for the user to unlock the phone
- Hijack / Sniff everything between non-secure world and secure world



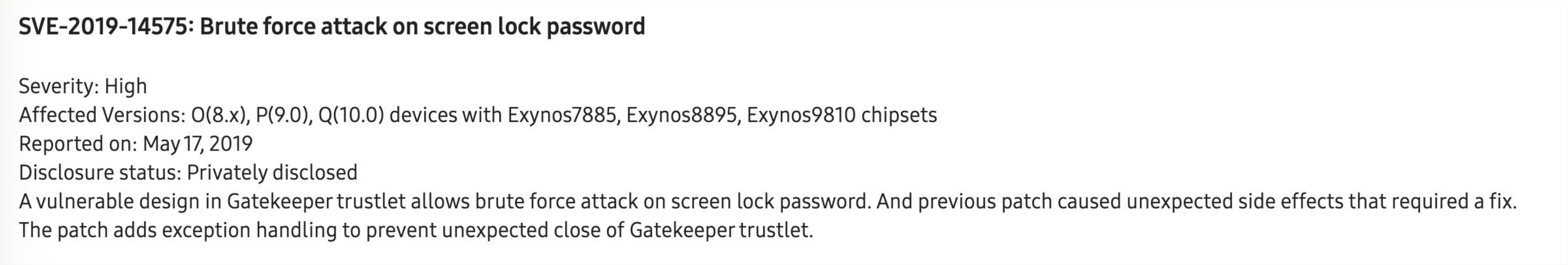
Exposed Attacking surface

- Attacking secure world trustlet
  - Gatekeeper trustlet
  - Samsung Pay trustlet
  - Keystore trustlet
  - …
- Many vulnerabilities in the past



Attack the gatekeeper trustlet to decrypt storage

- SVE-2019-14575

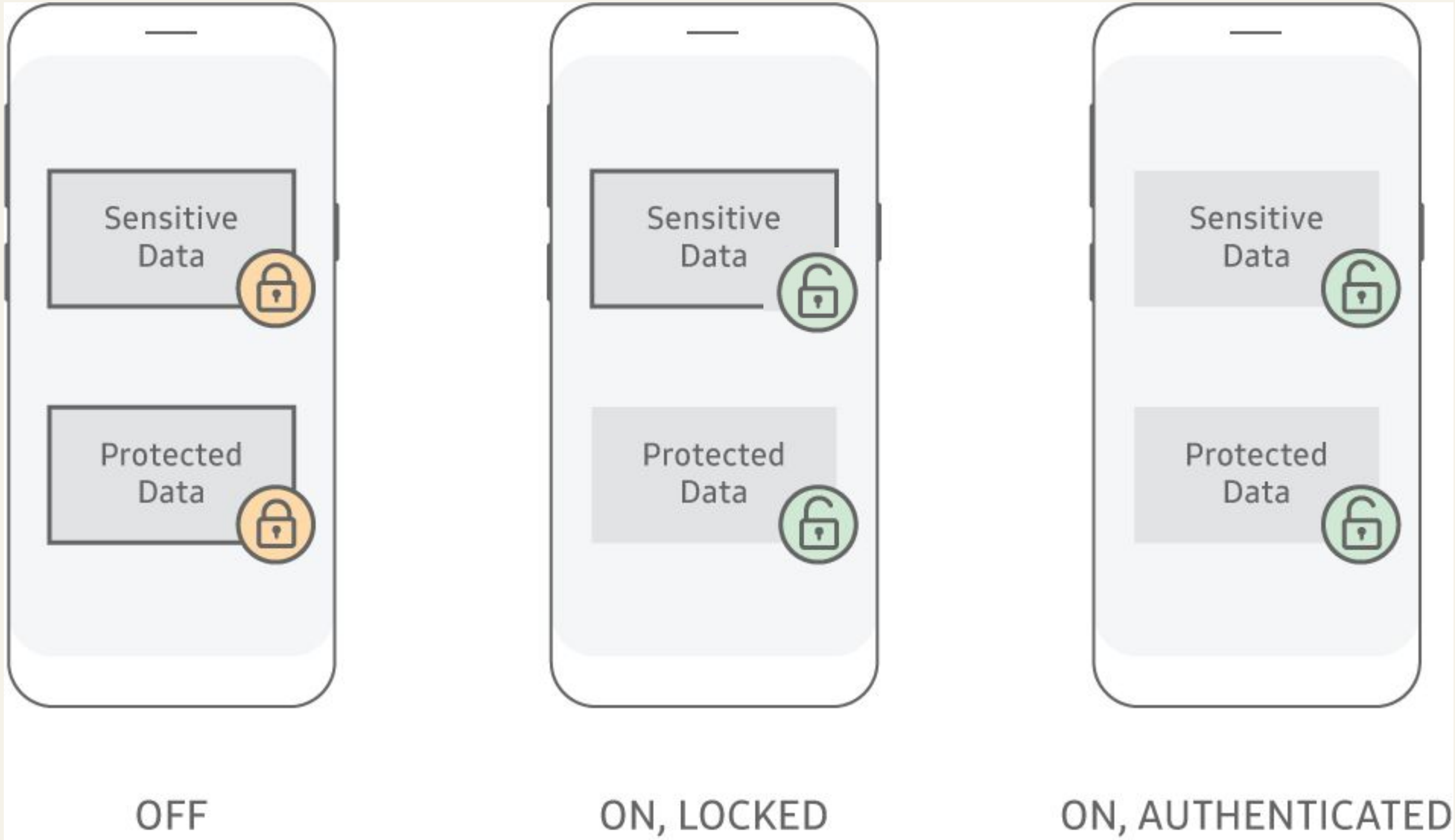


SVE-2019-14575: Brute force attack on screen lock password

Severity: High
Affected Versions: O(8.x), P(9.0), Q(10.0) devices with Exynos7885, Exynos8895, Exynos9810 chipsets
Reported on: May 17, 2019
Disclosure status: Privately disclosed
A vulnerable design in Gatekeeper trustlet allows brute force attack on screen lock password. And previous patch caused unexpected side effects that required a fix.
The patch adds exception handling to prevent unexpected close of Gatekeeper trustlet.

- With this vulnerability, we can try all the possible pattern codes in a few hours.

Sensitive Data unlocked



Conclusion

- Even if the data is stored in secure world, it doesn’t mean it’s 100% secure
- But it’s made exploiting complex, multiple actions are needed to retrieve the data
    - Landing - RCE / Local USB Exploit / Social Engineering
    - Privilege escalation to non-secure EL1
    - Vulnerabilities in trustlet to get into secure-world EL0
    - Privilege escalation from secure-world EL0 to secure-world EL1 or EL3
- Without all of this, especially the points in red, the data in the phone is still safe

Disclosure Timeline

- 2019-10-02 Report Vulnerability I
- 2019-10-08 Informed Vulnerability I duplicated
- 2019-10-11 Report Vulnerability II
- 2020-01-06 Samsung Patched, SVE-2019-15872
- 2020-01-21 Report Vulnerability III
- 2020-05-06 Samsung Patched, SVE-2020-16712

THANK YOU!

Jeffxx

jeffxx@trapa.tw

TEAMT5

Persistent Cyber Threat Hunters