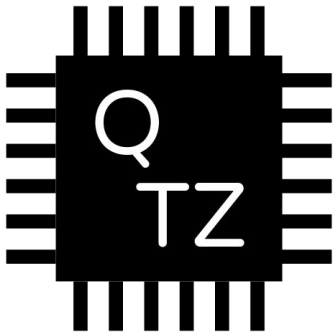
The road to Qualcomm TrustZone apps fuzzing



cp<r>

Slava Makkaveev

Motivation

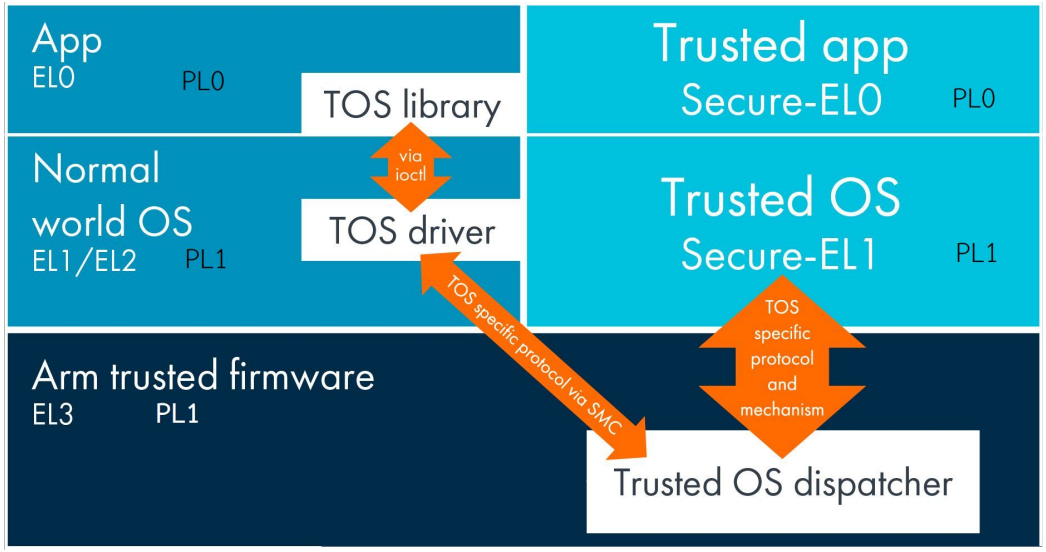
- TrustZone environment is the most protected part of a mobile device
  - keystore
  - storage security
  - biometric authentication
  - media rights management
  - mobile payment
  - ...

- There is very little public research in TrustZone apps area
- Building a fuzzing platform requires an engineering approach

ARM TrustZone Basics

ARM TrustZone Technology

- Two CPU (& SoC) modes: Normal + Secure
- Normal World does not have access to RAM & Cache of Secure World
- Normal World uses Security Monitor Call (SMC) instruction to communicate with Secure World



Trusted Execution Environment (TEE) for Mobile

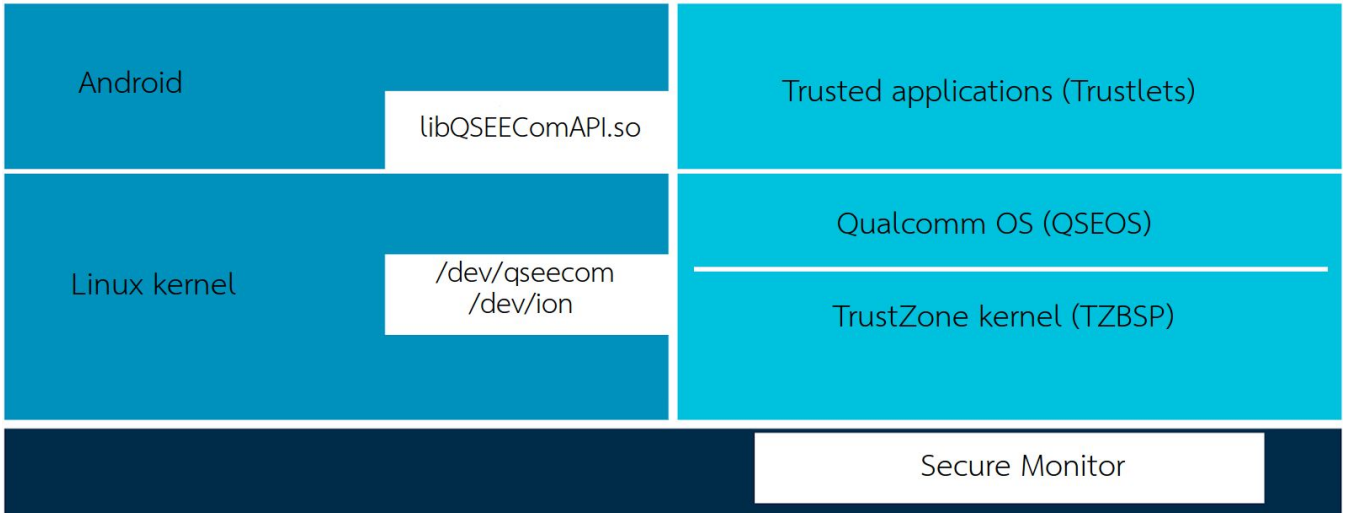
Qualcomm's
Secure Execution Environment
(QSEE)

...

HiSilicon's
Trusted Core

Trustronic's
Kinibi

Qualcomm's TEE



- surfaceflinger
- cacaoserver
- audioserver
- cameraserver
- drmserver
- ric
- keystore
- mediadrmserver
- mediaserver
- media.codec
- ...

How can talk with a trustlet?

```
G8142:/ $ ls -la /dev/qseecom
crw-rw---- 1 system drmrpc 235,   0 1970-08-09 22:55 /dev/qseecom
```

We need a source...

- libQSEEComAPI.so
  - image: *vendor.img* | *system.img*
  - device: /system/vendor/lib
- /dev/qseecom android/kernel/msm/drivers/misc/qseecom.c
- QSEOS & TZBSP (ELF32 | ELF64)
  - image *bootloader.img* | *tz.img* | ...
  - device

```
root@shamu:/ # ls -la /dev/block/platform/msm_sdcc.1/by-name/
lrwxrwxrwx root root 1970-05-28 00:58 tz -> /dev/block/mmcblk0p10
lrwxrwxrwx root root 1970-05-28 00:58 tzBackup -> /dev/block/mmcblk0p16
```

- A trustlest (*.mdt & *.bXX)
  - image *vendor.img* | *modem.bin* | *non-hlos.bin* | ...
  - device */firmware/image* | */vendor/firmware* | */system/etc/firmware* | ...

From Android to Trustlet Flow

libQSEEComAPI.so

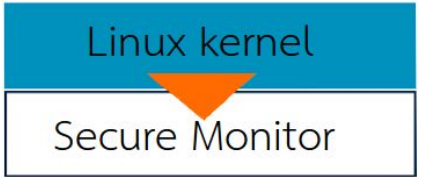
- android/platform/hardware/qcom/keymaster/QSEEComAPI.h

```
struct QSEECom_handle {
    unsigned char *ion_sbuffer;
};

int QSEECom_load_external_elf(struct QSEECom_handle **clnt_handle,
        const char *path, const char *fname);

int QSEECom_send_cmd(struct QSEECom_handle *handle,
        void *send_buf, uint32_t sbuf_len, void *rcv_buf, uint32_t rbuf_len);

int QSEECom_set_bandwidth(struct QSEECom_handle *handle, bool high);

int QSEECom_unload_external_elf(struct QSEECom_handle **handle);
```

- Requesting Linux kernel to load a trustlet

```
int __fastcall QSEECom_load_external_elf(int **a1, int a2, int a3)
{
  ...
  v9 = open("/dev/qseecom", 2);
  ...
  v19 = ioctl(v9, 0xC030970D, v46);
```

/dev/qseecom



- SMC instruction generates exception to enter the Security Monitor

```
static u32 smc(u32 cmd_addr) {
    int context_id;
    register u32 r0 asm("r0") = 1;
    register u32 r1 asm("r1") = (uintptr_t)&context_id;
    register u32 r2 asm("r2") = cmd_addr;
    do {
        asm volatile(

            "smc    #0\n"
            : "=r" (r0)
            : "r" (r0), "r" (r1), "r" (r2)
            : "r3");
    } while (r0 == SCM_INTERRUPTED);

    return r0;
}
```

- Handling of ioctl QSEECOM_IOCTL_LOAD_EXTERNAL_ELF_REQ

```
#define SCM_SVC_TZSCHEDULER                 0xFC

enum qseecom_qceos_cmd_id {
    QSEOS_CLIENT_SEND_DATA_COMMAND        = 0x06,
    QSEOS_LOAD_EXTERNAL_ELF_COMMAND       = 0x08,
    ...
};

struct qseecom_load_app_ireq cmd_buf;
cmd_buf.qsee_cmd_id = QSEOS_LOAD_EXTERNAL_ELF_COMMAND;
cmd_buf.phy_addr = pa; /* physical address of the ION BUF
                          which contains the trustlet blob*/
u32 svc_id = SCM_SVC_TZSCHEDULER;
u32 cmd_id = 1;

struct scm_command* scm_buf = kzalloc(..., GFP_KERNEL);
scm_buf->id = (svc_id << 10) | cmd_id; /* 0x3F001 */
memcpy(scm_buf->buf, &cmd_buf, cmd_len);

u32 cmd_addr = virt_to_phys(scm_buf);
smc(cmd_addr);
```

QSEOS & TZBSP

Nexus 6 (LMY48Y) tz.bin

- ELF32 ARM
- There is XPU unprotected code segment: 0xFE806000 - 0xFE80FFB0

Secure Channel Manager table

Beginning at 0xFE82B01C

- Id (0x3F001)
- Ptr to SCM function name (“tzbsp_exec_smc”)
- Type of call
- Ptr to handling function (0xFE808540)
- Number of arguments
- Size of each argument

```
LOAD:FE82B6AC        DCD 0x1801
LOAD:FE82B6B0        DCD aTzbspIsService ; "tzbsp_is_service_available"
LOAD:FE82B6B4        DCD 0xF
LOAD:FE82B6B8        DCD tzbsp_is_service_available+1
LOAD:FE82B6BC        DCD 1
LOAD:FE82B6C0        DCD 4
LOAD:FE82B6C4        DCD 0x3F001
LOAD:FE82B6C8        DCD aTzbspExecSmc    ; "tzbsp_exec_smc"
LOAD:FE82B6CC        DCD 0x2F
LOAD:FE82B6D0        DCD tzbsp_exec_smc+1
LOAD:FE82B6D4        DCD 1
LOAD:FE82B6D8        DCD 0x10
```

tzbsp_exec_smc (0xFE808540) → *qsee_load_and_auth_elf_image* (0xFE8575DE) if qsee_cmd_id == 0x08

Verifies and loads trustlet

Example. Send a command to a Trustlet

Load a trustlet

- Path (“firmware/image”)
- Name (“prov”)

Send a command

- Request buffer + size
- Response buffer + size

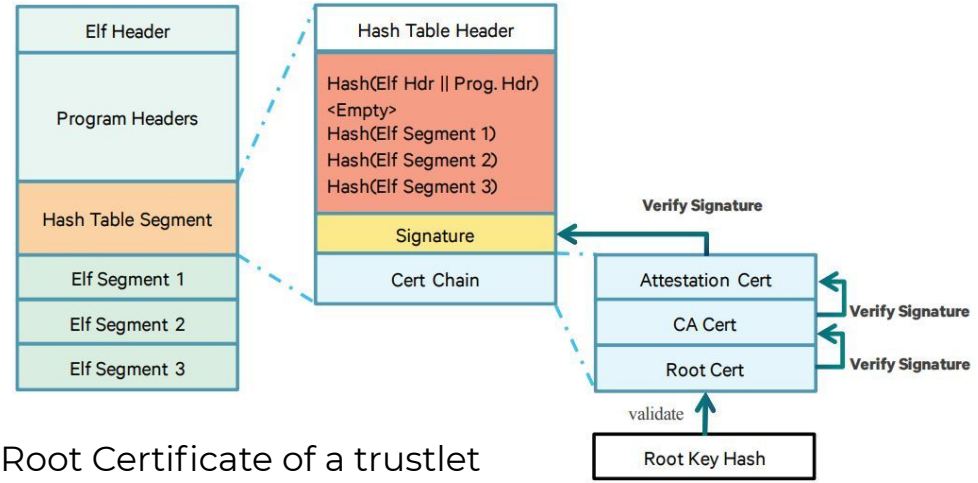
```
struct qcom_tl_handle {
    void *lib;
    struct QSEECom_handle *qseecom;
    int (*QSEECom_start_app)(struct QSEECom_handle ** handle,
        char* path, char* appname, uint32_t size);
    int (*QSEECom_send_cmd)(struct QSEECom_handle* handle,
        void *cbuf, uint32_t clen, void *rbuf, uint32_t rlen);
};

struct qcom_tl_handle* initialize_tl_handle() {
    struct qcom_tl_handle* handle = malloc(sizeof(struct qcom_tl_handle));
    memset(handle, 0, sizeof(struct qcom_tl_handle));

    handle->lib = dlopen("libQSEEComAPI.so", RTLD_NOW);
    *(void **)(&handle->QSEECom_start_app) = dlsym(handle->lib, "QSEECom_start_app");
    *(void **)(&handle->QSEECom_send_cmd) = dlsym(handle->lib, "QSEECom_send_cmd");

    return handle;
}

struct qcom_tl_handle* handle = initialize_tl_handle();

(*handle->QSEECom_start_app)((struct QSEECom_handle **)&handle->qseecom,
    "/firmware/image", "prov", 0x15000);

uint32_t req_size = 0x2000, resp_size = 0x2000;
uint8_t* req = handle->qseecom->ion_sbuffer;
uint8_t* resp = req + reqsize;

req[0] = 0x1400;
req[1] = 0xFFFFFFE2;

(*handle->QSEECom_send_cmd)(handle->qseecom, req, req_size, resp, resp_size);
```

Qualcomm’s Trusted Application

Signed ELF

ELF extended by Hash Table Segment

- Hash table header
- SHA-256 hash of ELF header and program headers data block
- SHA-256 hashes of all program segments
- Hash block's signature
- Certificates chain



QSEOS validates Root Certificate of a trustlet

A trustlet's files

Split signed ELF32 | ELF64

.mdt

.b00

.b01

.b02 - bXX

Nexus 6

- isdbtmm
- keymaster
- playready
- prov
- sampleapp
- securemm
- tqs
- widevine

Samsung S7

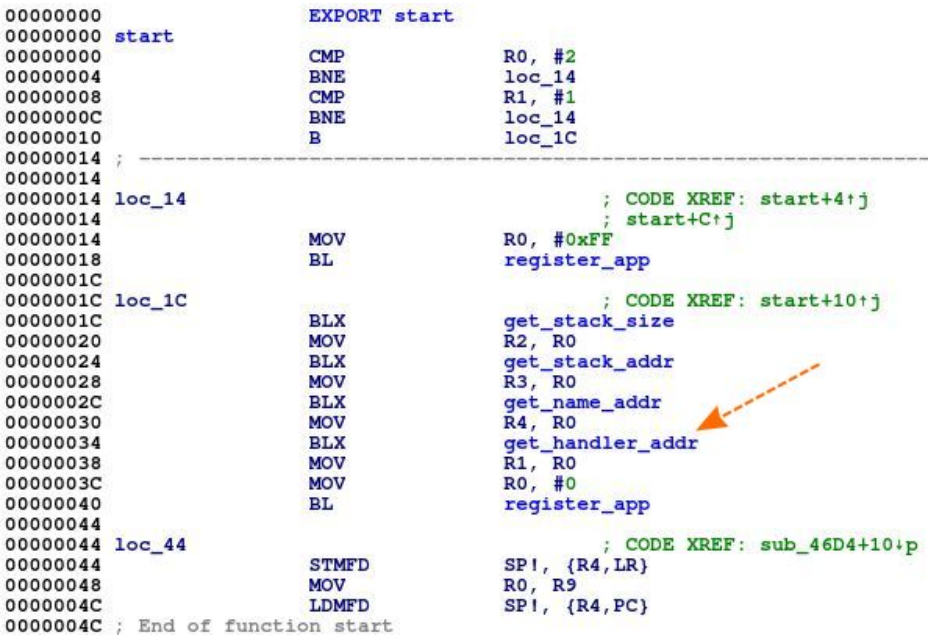
- authhat
- authnr
- bdwlan30
- cmnlib
- cppf
- dhsecapp
- dmverity
- dxprdy
- esecomm
- fingerpr
- gptest
- isdbtmm
- lksecapp
- mldap
- mst
- sdtvrmp
- securefp
- securemm
- sem
- skeymast
- skm
- skmm_ta
- skpm
- smplap32
- softsim
- sshdcpap
- tima_atn
- tima_key
- tima_lkm
- tima_pkm
- tz_ccm
- tz_iccc
- tz_otp
- widevine
- wsm

Trustlet entry point

Start function

- Registers app via -0x100 syscall
  - Stack address + size
  - Address of name
  - Handler function

prov trustlet (Moto G4)



```
00000000                EXPORT start
00000000 start
00000000                CMP             R0, #2
00000004                BNE             loc_14
00000008                CMP             R1, #1
0000000C                BNE             loc_14
00000010                B               loc_1C
00000014 ; ---------------------------------------------------------------------
00000014
00000014 loc_14                                 ; CODE XREF: start+4↑j
00000014                                        ; start+C↑j
00000014                MOV             R0, #0xFF
00000018                BL              register_app
0000001C
0000001C loc_1C                                 ; CODE XREF: start+10↑j
0000001C                BLX             get_stack_size
00000020                MOV             R2, R0
00000024                BLX             get_stack_addr
00000028                MOV             R3, R0
0000002C                BLX             get_name_addr
00000030                MOV             R4, R0
00000034                BLX             get_handler_addr
00000038                MOV             R1, R0
0000003C                MOV             R0, #0
00000040                BL              register_app
00000044
00000044 loc_44                                 ; CODE XREF: sub_46D4+10↓p
00000044                STMFD           SP!, {R4,LR}
00000048                MOV             R0, R9
0000004C                LDMFD           SP!, {R4,PC}
0000004C ; End of function start
```

Trustlet initialization

Handler function

- Trustlet initialization
- Linkage with *cmnlib*
- Listening to commands

cmnlib trusted app

- Common code library
- One instance for all trustlets

tzpr25 trustlet (Samsung S5)

```
void __fastcall handler(void (__fastcall *cmnlib_text_seg_arg1),
{                                       int cmnlib_data_seg_arg2)
  cmnlib_text_seg = cmnlib_text_seg_arg1;
  import_table = (data_seg + 0xA5B0);
  data_seg[0xA5B2] = cmnlib_data_seg_arg2;
  data_seg[0xA5B3] = get_data_seg_addr();
  exec_init_array();
  if ( cmnlib_text_seg )
  {
    v6 = data_seg + 0xA71B1;
    v7 = data_seg + 0xA91C9;
    v8 = 0x20000;
    cmnlib_text_seg(data_seg + 0xA5B0, &v6, data_seg[0xA5B3], data_seg[0xA5B2]);
  }
  if ( *import_table )
  {
    v5 = **import_table;
    data_seg[0xA5B1]= v5;
    *v5 = sub_2746E;
    v5[1] = sub_27440;
    data_seg[0xA71AC] = sub_2740A;
    data_seg[0xA71AD] = sub_273DA;
    data_seg[0xA71AE] = data_seg + 1;
    data_seg[0xA71AF] = data_seg[0xA5B2];
    data_seg[0xA71B0] = data_seg[0xA5B3];
  }
  init_trustlet();
  while ( 1 )
  {
    FFFFFFE2_syscall((data_seg + 0xA5B4), 4);
    handle_cmd();
  }
}
```

Linking with *cmnlib*

Waiting for commands

Trustlet command handler

Command handler function

- Processing commands from the Normal World
  - Request buffer + size
  - Response buffer + size

```
int  __fastcall cmd_handler(unsigned __int8 *in, unsigned int in_size, unsigned __int8 *out, unsigned int out_size)
{
  cmd_id = *(_DWORD *) in;
  qsee_log(5, "\"%s: cmdreq %08x cmd_addr %08x\"", "tz_app_cmd_handler", cmd_id, cmd_id);
  result = cmd_id - 0x70001;
  switch ( cmd_id )
  {
    case 0x70001:
      v6 = get_int32(in + 4);
      v7 = Prov_GetRandom((int)(out + 4), v6);
      result = set_int32(v7, (int)(out + 0x7D4));
      break;
    case 0x70002:
      v8 = get_int32(in + 4);
      v9 = Prov_GetInfo(v8);
      result = set_int32(v9, (int)(out + 4));
      break;
    case 0x70003:
      qsee_log(5, "\"%s: KEY IMPORT %x\"", "tz_app_cmd_handler", cmd_id);
```

Trustlet memory

secapp-region of physical memory

```
root@shamu:/ # dmesg | grep qsee
QSEECOM: qseecom_probe: disk-encrypt-pipe-pair=0x2
QSEECOM: qseecom_probe: Device does not support PFEQSEECOM: qseecom_probe: hlos-ce-hw-instance=0x1
QSEECOM: qseecom_probe: qsee-ce-hw-instance=0x0QSEECOM: qseecom_probe: secure app region addr=0xd600000 size=0x500000
```

- Trustlet does not have access to memory region of any trustlet other than *cmnlib*
- Communication with TZBSP through syscalls (*SVC 1400* | *SVC 14F9*)

Trustlet's data segment region

- Trustlet's heap and stack regions are part of the trustlet's data segment region

Our goal is to execute a trusted app in the Normal World

Let’s just run a trustlet ELF on Android

Smoke test

1. Dump a trustlet’s data and *cmnlib*’s data segments from the *secapp-region*
2. Implement a “trustlet loader” as Android program which
   a. *mmap* dumped blobs and relevant .text segments to virtual memory at the same address as in the *secapp-region* (memory near 0xd600000 is accessible)
   b. Points *R9* to the allocated trustlet’s data
   c. Calls the *cmd_handler* function

Problem

- TZBSP syscalls are not recognised by Linux Kernel

Let’s try to patch the trustlet

Inject a handler for a new command id

- Get base address of the trustlet
- Read/write data from/to *secapp-region*
- Invoke a syscall

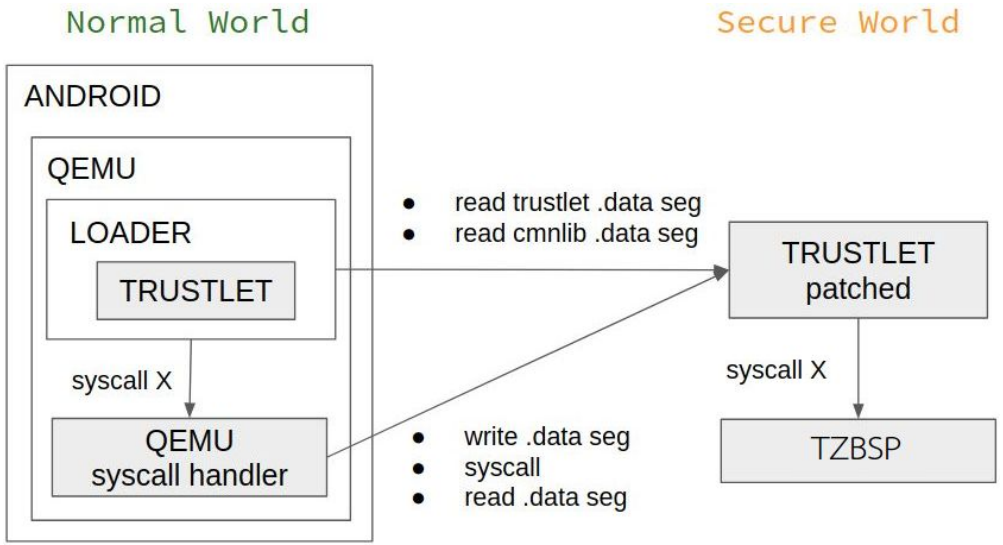
```
main:
    push {r0, r1, r2, r3, r4, r5, r6, lr}

    mov r5, r0
    mov r6, r2

    ldr r1, [r5]
    cmp r1, #0x99
    bne empty

    ldr r1, [r5, #4]
    add r1, r1, #2
    mov r3, pc
    ldrb r1, [r1, r3]
    add pc, r1

    .byte   2
    .byte   4
    .byte   6
    .byte   8

    b call
    b get_base
    b read
    b write

call:
    ldr r0, [r5, #8]
    mov r1, r5
    add r1, r1, #0xc

    bl syscall
    str r0, [r6]

    b empty
```

```
get_base:
    mov r0, r9
    str r0, [r6]

    ldr r1, cmnlib
    ldr r0, [r0, r1]
    str r0, [r6, #4]

    b empty

read:
    ldr r0, [r5, #8]
    mov r1, r6
    ldr r2, [r5, #0xc]
    bl copy

    b empty

write:
    mov r0, r5
    add r0, r0, #0x10
    ldr r1, [r5, #8]
    ldr r2, [r5, #0xc]
    bl copy

    b empty

empty:
    pop {r0, r1, r2, r3, r4, r5, r6, pc}
```

Emulation Scheme

Using QEMU as channel between Worlds

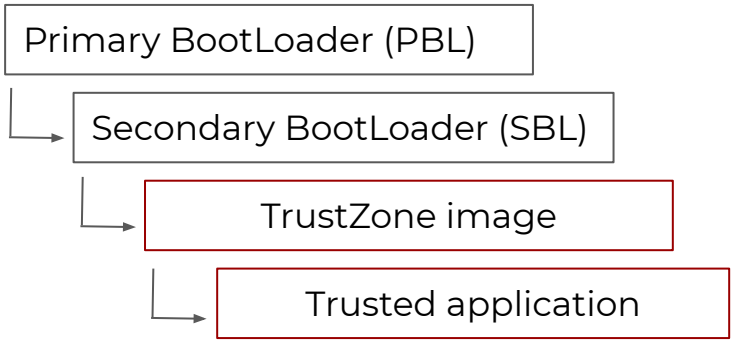
```
void cpu_loop(CPUARMState *env) {
...
for(;;) {
  ...
  switch(trapnr) {
  case EXCP_BKPT:
     ...
     if (n == ARM_NR_qsee_1400 || n == ARM_NR_qsee_14F9) {
         if (qsee_proxy_write_data_seg() < 0) {
             goto error;
         }
         env->regs[0] = qsee_proxy_syscall(
             env->regs[0], env->regs[4], env->regs[5],
             env->regs[6], env->regs[7], env->regs[8],
             env->regs[9], n);
         if (qsee_proxy_read_data_seg() < 0) {
             goto error;
         }
```



Loading a patched trusted app in the Secure World

Break Qualcomm's Chain of Trust

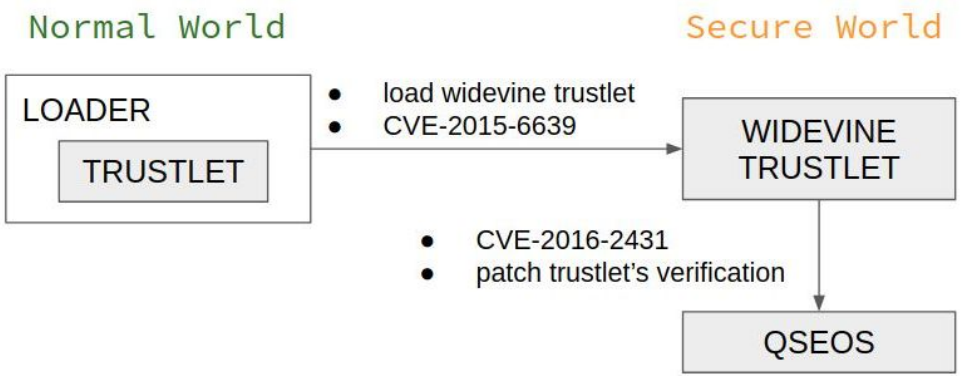
Secure boot



An exploit should be used to break the chain

Well-described 1-day exploits *CVE-2015-6639* + *CVE-2016-2431* allow

- Write to QSEOS data segments
- Write to XPU unprotected code segment 0xFE806000 - 0xFE80FFB0

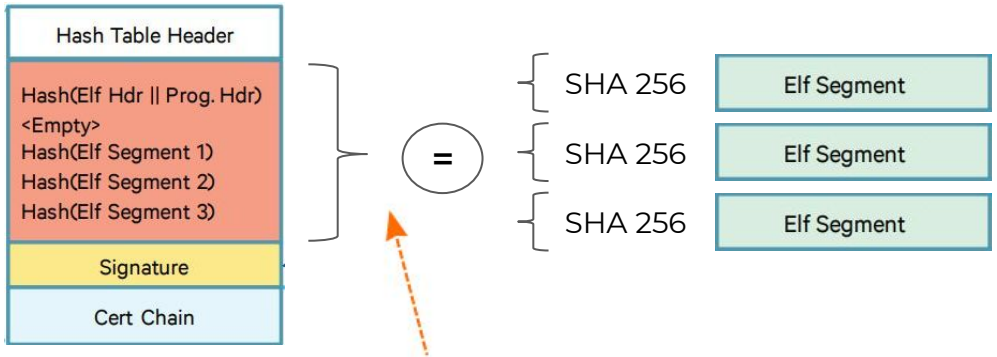


Break trustlet's verification algorithm

QSEOS *qsee_load_and_auth_elf_image* function

1) calls tzbsp_pil_init_image function
- parses Hash Table Segment
- validates Hash Block's signature

2) calls tzbsp_pil_auth_reset function
- calculates actual SHA-256 segments' hashes
- validates actual hashes against Hash Block



Overwrite Hash Block after verification

Break trustlet's verification algorithm

tzbsp_pil_init_image function

```
if ( elf_buff )
{
  if ( tzbsp_subsys_is_supported(id) )
  {
    tzbsp_pil_unlock_area(id);
    if ( !tzbsp_is_ns_range(elf_buff, 0x34) && id != 7 )
    {
      tzbsp_log(3, "(%u)", 0x1F);
      return 0xFFFFFFEE;
    }
    tzbsp_dcache_inval_region(elf_buff, 0x34);
    base_info = (int *)(0x3C * id - 0x17D3E80);// 0xFE82C324
    tzbsp_mutex_lock((unsigned int *)(0x3C * id - 0x17D3E50));
    tzbsp_clean_pil_priv(base_info);
    base_info[0xB] = id;
    if ( tzbsp_pil_is_elf(elf_buff) )
    {
      if ( !tzbsp_pil_populate_elf_info(id, elf_buff, base_info) )
      {
        if ( tzbsp_pil_verify_sig(id, (int)base_info) >= 0 )
        {
          if ( !tzbsp_ssd_parse_md((_DWORD *)(0x3C * id - 0x17D3E80),
                  elf_buff, 0x3C * id - 0x17D3E48) )
          {
            *(_DWORD *)(0x3C * id - 0x17D3E4C) = 2;// 0xFE82C358
```

The patch

Inject a code into map_region function (0xFE8066E8) that patches *base_info* object with a new Hash Block

```
main:
    push {r0, r1, r2, r3, r4, r5, lr}

    ldr r4, base_info

    mov r0, pc
    add r0, r0, #0x38

    ldr r1, [r4, #0x14]
    cmp r1, #0
    beq empty

    ldr r3, [r4, #0x34]
    cmp r3, #1
    bne second

    first:
        mov r2, #0x20
        bl copy

        b empty
```

```
second:
    cmp r3, #2
    bne empty

    mov r2, #0x20

    add r1, r1, r2

    ldr r4, [r4, #0x18]
    sub r4, r4, #0x40

    mov r3, #0
    loop1:
        add r0, r0, r2
        add r1, r1, r2
        bl copy

        add r3, r3, r2
        mov r5, r4
        sub r5, r3, r5

        blt loop1

empty:
pop {r0, r1, r2, r3, r4, r5, pc}

base_info:  .word 0xfe82c324
```

Putting it all together

What do we have?

- Vulnerable *widevine* trustlet in the Secure World
  - Breaks trustlet's verification by QSEOS
- Our patched trustlet in the Secure World
  - Provides dumps + Executes syscalls
- The trustlet loader over QEMU on Android
  - Executes the original trustlet code using the dumped data

What do we need?

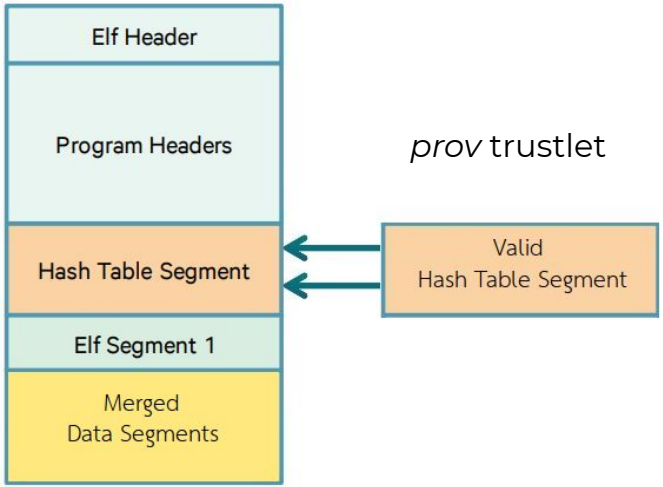
- Compile AFL fuzzer + QEMU for Android
  - libiconv 1.14
  - libffi 3.2.1
  - gettext 0.19.7
  - glib 2.48.1
  - AFL 2.52b
  - QEMU 2.10.0
- Fuzz the trustlet loader on Nexus 6 device

Executing a modern trusted app on Nexus device

What's new? Let's adapt

- Incompatible certificate chain
- There are several data segments, but QSEOS expects to see only one
- Base address of a trustlet's code segment is zero

Bypass QSEOS validation



prov trustlet

Scan .data segment for xrefs

```
begin = 0xe4000
end = 0xf06e0

minv = 0xd000
maxv = 0x1106e0

curr = begin
while curr < end:
    MakeDword(curr)

    value = Dword(curr)
    if value >= minv and value <= maxv:
        print'0x%x: 0x%x' % (curr, value)

    curr += 4
```

esecomm trustlet
(Samsung S7 Edge)

Fuzzing of trusted apps

Several discovered vulnerabilities

32 bytes to crash biometric authentication on Samsung phones

- Request buffer `21 00 00 00 8E 01 00 00 0A 35 00 00 00 00 00 00`
- Response buffer `00 00 00 00 00 00 00 00 00 DD 02 00 00 00 00 00`

Samsung

- *tzpr25* PlayReady digital rights management
- *sec_store* Storage security
- *esecomm* Secure payment transactions
- *authnr* Biometric authentication

Motorola

- *prov* Secure key storage

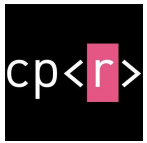
LG

- *dxhdcp2* Discretix digital content protection

Qualcomm

- *kmota* Secure key storage

Thank you!



slavam@checkpoint.com

@_cpresearch_
research.checkpoint.com