Attacking your “Trusted Core”
Exploiting TrustZone on Android

Di Shen (@returnsme)

BlackHat USA 15
Agenda

- **Background**
  - About Huawei Ascend Mate 7
  - TEE architecture of Huawei Hisilicon
  - Attack Surface
- **Vulnerability in Normal World**
  - technical details
  - gain root privilege
- **Vulnerabilities in Secure World (TEE)**
  - technical details
  - read fingerprint image from sensor / bypass sec features
- **Conclusion**
Who am I

- Security researcher from Qihoo 360
- Mainly focus on Android
- Always like console games and manga/anime
Background
Huawei Ascend Mate 7

- HiSilicon Kirin 925 SoC chipset
- HiSilicon implemented its own TEE kernel (Trusted Core)
- The world’s first Android smartphone with touch fingerprint sensor, featuring FPC1020
- 1 million units sold by Huawei in the first month
Fingerprint: protected by SecureOS
TEE architecture of Huawei
- accepting malformed ioctl command may allow installed application to execute arbitrary code in Linux Kernel.

- mistake in input structure bound-check may lead to an arbitrary code execution vulnerability in TEE

- system call bugs may allow a malicious TA to escalate privilege
Attack “TrustedCore”

- Installed application
  - Attack TZdriver
    - Send ioctl command
      - Attack TA
        - execute SMC instructions
          - Attack TEE kernel
            - system call
Vulnerability in Normal World
TZDriver: /dev/tc_ns_client

• Accessible to any installed applications

• provide communication APIs between NW and SW

• provide an ioctl interface to both user space clients and other kernel module

  - for user clients, use copy_to_user/copy_from_user to copy input/output param buffer

  - for kernel modules, use memcpy directly
typedef struct {
    unsigned char uuid[16];
    unsigned int session_id;
    unsigned int cmd_id;
    TC_NS_ClientReturn return;
    TC_NS_ClientLogin login;
    unsigned int paramTypes; //type of input param
    TC_NS_ClientParam params[4]; //address or value of input
    bool started;
} TC_NS_ClientContext;
typedef union {
    struct {
        unsigned int buffer; //ptr of buffer
        unsigned int offset; //size of buffer
        unsigned int size_addr;
    } memref;
    struct {
        unsigned int a_addr; //ptr of a 4-bytes buffer
        unsigned int b_addr; //ptr of a 4-bytes buffer
    } value;
} TC_NS_ClientParam;

What if user client send a kernel pointer to driver?
static int TC_NS_SMC_Call(TC_NS_ClientContext *client_context, TC_NS_DEV_File *dev_file, bool is_global){
    ....
    // build a TC_NS_SMC_CMD struct
    ....
    // execute SMC instruction
    TC_NS_SMC(smc_cmd_phys);
    // copy result from smc_cmd.operation_phys to callers' buffer(client_param.value)
    if(client_operation->params[0].value.a > 0xbfffffff){
        //(driver think caller is from kernel space)
        *(u32 *)client_param->value.a_addr = operation->params[i].value.a;
    }
    else{
        //(driver think caller is from user space
        copy_to_user(....);
    }
    if(client_operation->params[0].value.b > 0xbfffffff){
        *(u32 *)client_param->value.b_addr = operation->params[i].value.b;
    }
    else{
        copy_to_user(....);
    }
    ....

Step 1: `smc_call(&TZvalue)`

Step 2: `malloc(TZvalue)`
   *(TZvalue) = shellcode

Step 3: Offset = ptmx_fops->sync - buff
   *(ptmx_fops->sync) = TZvalue

Step 4: Call sync(“/dev/ptmx”)
How to find a stable “TZValue”

• Extract TEE image from firmware. Using HuaweiUpdateExtractor.exe

• TEEOS.img is not encrypted. Drag into IDA.

• Find a interface provided by TA will return a stable “TZvalue”
Time querying interface in TEEGlobalTask

```c
int get_sys_time()
{
    int result; // r0@1
    tag_TC_NS_Operation *v1; // r3@1
    unsigned int v2; // [sp+0h] [bp-10h]@1
    int v3; // [sp+4h] [bp-Ch]@1
    get_time((int)&v2);
    result = 0;
    v1 = dword_5E2E0->operation_phys;
    v1->params[0].value.a = v2; //second from startup
    v1->params[0].value.b = 1000 * v3; //millisecond
    return result;
}
```
Vulnerabilities in Secure World
Send malformed request to TA

- now I can execute SMC instruction by TZDriver ret2user exploit

- SMC param: a pointer to structure TC_NS_SMC_CMD

```c
typedef struct tag_TC_NS_SMC_CMD{
    unsigned int uuid_phys;
    unsigned int cmd_id;
    unsigned int dev_file_id;
    unsigned int context_id;
    unsigned int agent_id;
    unsigned int operation_phys;
    unsigned int login_method;
    unsigned int login_data;
    unsigned int err_origin;
    bool started;
} TC_NS_SMC_CMD;
```
**Time querying interface in TEEGlobalTask**

```c
int get_sys_time()
{
    int result; // r0@1
    tag_TC_NS_Operation *v1; // r3@1
    unsigned int v2; // [sp+0h] [bp-10h]@1
    int v3; // [sp+4h] [bp-Ch]@1

    get_time((int)&v2);
    result = 0;
    operation_phys = dword_5E2E0->operation_phys;
    *(int*)(operation_phys+4) = v2;
    *(int*)(operation_phys+8) = 1000 * v3;
    return result;
}
```

CVE ID: CVE-2015-4422
arbitrary physical memory overwriting

- no security checking on operation_phys

- if second = 0xAABBCCDD, every time we can write 4 byte “DD,CC,BB,AA” at operation_phys + 4

- The “DD” is the last byte of second and cycle from 0x00 to 0xFF.

- Write a byte you want at a right second — arbitrary physical address overwriting
Code execution in TEE

• Main idea
  - patch text code of TEEGlobalTask, call TEE function and return to my shellcode

• Good news:
  - few mitigation in RTOSck, the kernel of TEE
  - No ASLR, XN or “unwritable Text code”.

• Bad news:
  - I don’t know where to patch without base address of TEEGlobalTask

• Don’t give up:
  - try to find a backdoor which may leak some address by reverse engineering :)

Main idea
patch text code of TEEGlobalTask, call TEE function and return to my shellcode

Good news:
- few mitigation in RTOSck, the kernel of TEE
- No ASLR, XN or “unwritable Text code”.

Bad news:
- I don’t know where to patch without base address of TEEGlobalTask

Don’t give up:
- try to find a backdoor which may leak some address by reverse engineering :)
• send an invalid operation _phys from Normal world.

• RTOSck may write register value to shared memory when task crashed.

• estimate base of “TEEGlobalTask” by crashed $pc

• PC = 0x2E103050  base = 0x2E100000
Patch 4 bytes

Before patch:
```
alloc_exception_mem
    STMFD    SP!, {R3-R5,LR}
    LDR     R3, =(dword_2E15CFC0 - 0x2E104828)
    LDR     R3, [PC,R3] ; dword_2E15CFC0
    LDR     R3, [R3,#0x10]
    LDR     R3, [R3,#0x14]
    LDR     R5, [R3,#4]
    LDR     R4, [R3,#8]
    MOV     R0, R5 ; int
    MOV     R4 ; int
    BL  map_memory
    MOV     R0, R5
    MOV     R4
    LDMFD   SP!, {R3-R5,LR}
    BX  syscall_f084

; End of function alloc_exception_mem
```

After patch:
```
syscall_f084
    STMFD    SP!, {LR}
    SVC 0x0
    LDMFD   SP!, {LR}
    BX  LR

; End of function syscall_f084
```

; void __cdecl patch_syscall(int, int)
• alloc buffer for shellcode via kmalloc

• Normal world : send request to TEE
  - cmd = GLOBAL_CMD_ID_ALLOC_EXCEPTION_MEM
  - with param (0,shellcode_physical_addr)

• TEE call syscall_f084(0,kernel_pool_phy)
What we can do with a TEE exploit

• Modify physical memory of Linux Kernel
  - e.g. patch “avc_has_perm" to bypass SELinux for Android

• Modify memory of TEE
  - disable hash checking for Modem image
  - disable TA signature checking in TEE and load unsigned TA from normal world

• Call TEE API
  - read encrypted data from sec-storage
  - read fingerprint image from sensor
  - read/write efuse data

• Install a rootkit
  - hook Linux kernel
  - hook TEE API
Read fingerprint from sensor

- "__FPC_readImage" is a syscall in TEE kernel(RTOSck)
  - Provided by FPC1020 driver
  - Only can be used by TA_Fingerprint task
  - Unfortunately my code execution exploit is under "TEE_GlobalTask" context. :( 

- Patch TEE kernel to bypass this restriction.
  - Need another vulnerability to modify TEE kernel memory.

```
warning: map secure section to ns
PAGE: no page reference found
warning: map secure section to ns
do not support TA  TaskPID is [16],  acName is [TEEGlobalTask]
readImage error = [-5]
```

<- TEE error log
Overwriting TEE kernel

```c
signed int __fastcall sys_call_overwrite(int a1, int a2) {
    signed int v2; // r3@2
    int v4; // [sp+0h] [bp-14h]@1 int v5; // [sp+4h] [bp-10h]@1 v5 = a1;
    v4 = a2;
    if ( *(DWORD *)a1 == 0x13579BDF ) {
        // write (*(int*)(arg1 + 0x18C) + 7) >> 3 to arg2
        *(WORD *)v4 = (unsigned int)(*(DWORD *)(v5 + 0x18C) + 7) >> 3;
        v2 = 0;
    }
    return v2;
}
```

```
*(uint16*)r1 = (*(int*)(r0 + 0x18C) + 7) >> 3
```
Read fingerprint image from sensor

DEMO!
Thank you

retme7@gmail.com

@returnsme