Exploiting Trustzone on Android

Di Shen(@returnsme)
retme7@gmail.com

1 Introduction

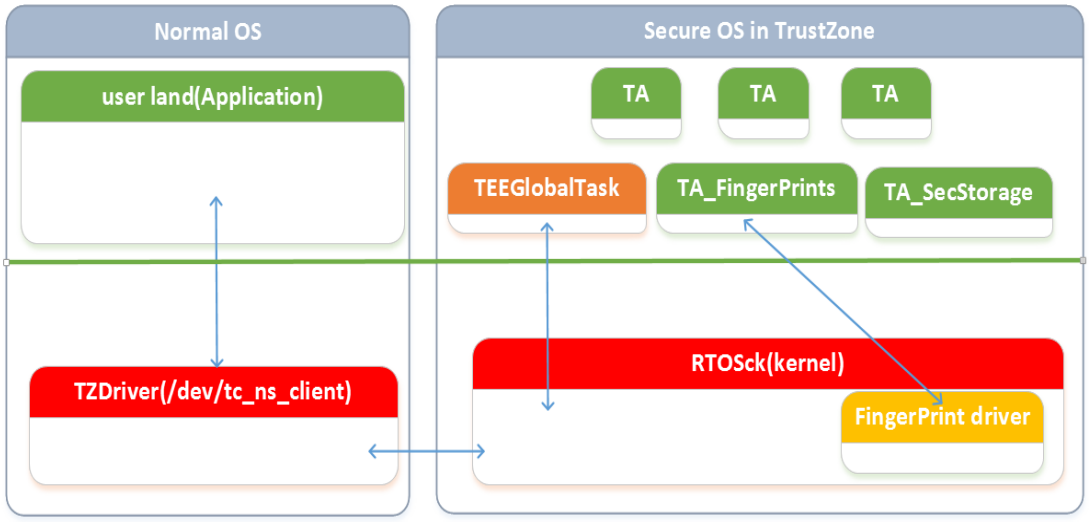
This paper tells a real story about exploiting TrustZone step by step. I target an implementation of Trusted Execution Environment(TEE) used by Huawei HiSilicon. Firstly I find a vulnerability to gain kernel-level privileges in normal world. Then I find another one for arbitrarily code execution in TEE. It's a proof-of-concept that any local application is able to execute shellcode in HiSilicon's TEE.

These vulnerabilities affected all Huawei devices with Huawei HiSilicon SoC chipset.

2 Background

"ARM® TrustZone® technology is a system-wide approach to security for a wide array of client and server computing platforms, including handsets, tablets, wearable devices and enterprise systems." Devices developed with TrustZone technology can support a full Trusted Execution Environment. TEE runs in a special CPU mode called "secure mode", so memory for secure mode and security functions can be hidden to "normal world". In this way, Android vendors can supply many secure features such as fingerprint scanning, DRM, kernel protection, secure boot and so on.You can find more information on TrustZone's website[1] and an exploration on genode.org[2].

Huawei HiSilicon's TEE is compliant with the recent Global Platform TEE specifications. But its implementation is totally undocumented. By reversing the firmware of its TEE, some log strings let me understand its' architecture:



TZDriver "/dev/tc_ns_client" is a kernel driver which provides an interface to user space clients for communication with TEE OS. Just like "/dev/qseecom" in QSEE.The only diffirent is every local Android application can access TZDriver on mate 7, qseecom only can be accessed by some system process.

TA is short for Trusted Application. TAs are elf files. Each of them provide a secure-related service, such as secure storage(TA_SecStorage), secure boot, keystone, fingerprint scanning(TA_FingerPrints) and so on.

“TEEGlobalTask” is the first TA of TEE’s user mode. It can load elf modules, dispatched call from outside to sub-service provided by other TAs.

“RTOSck” is the kernel of TEE, it’s a Real-Time OS developed by HUAWEI, being similar to uC/OS-II.A fingerprint driver also includes in RTOSck. The driver only can be accessed by TA_FingerPrints , it help TA read finger print image from sensor.

3. Vulnerability in /dev/tc_ns_client (CVE-2015-4421)

A Secure Monitor Call (SMC) is used to enter the Secure Monitor mode and perform a Secure Monitor kernel service call. This instruction can only be executed in privileged modes. So if user process want to send malformed SMC to secure world and exploit TEE, it must gain kernel-level privileges first.

“/dev/tc_ns_client” is a kernel driver which provides an ioctl interface to both user space clients and other kernel module. Clients use its “TCAPI” to communicate with secure world. The driver’s file permission is “rw-rw-rw-“ and its SE context is “u:object_r:device:s0”. Any client in user mode can access this driver.

Clients use TC_NS_ClientParam struct to send a buffer pointer to driver. Then driver request a SMC to TEE, and copy returned value to pointer in TC_NS_ClientParam.

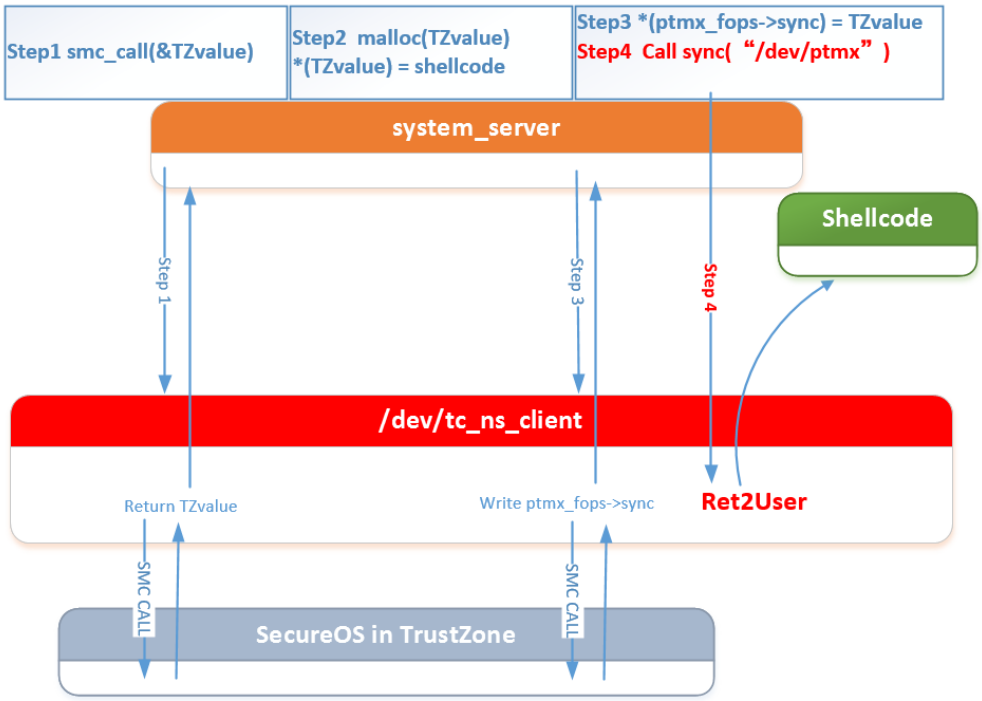
```
typedef union {
struct {
    unsigned int buffer; //ptr of buffer
    unsigned int offset; //size of buffer
    unsigned int size_addr;
} memref;
struct {
    unsigned int a_addr; //ptr of a 4-bytes buffer
    unsigned int b_addr; //ptr of a 4-bytes buffer
    } value;
} TC_NS_ClientParam;
```

The driver made a mistake in bounds-checking. The pseudo-code is as follow:

```
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(....);
    }
    ....
}
```

What if we send a kernel pointer from user mode? The driver will copy results directly without using copy_to_user. So we have an opportunity to write a given value to kernel space. If the value returned by TEE is under 0xc0000000,we may gain root privileges by "ret2user" trick as follow.



I choose TEE OS's internal time as a given returned value. The global service in TEE provide a time query interface with cmd_id = "GLOBAL_CMD_ID_TEE_TIME".

Here is the pseudo-code in "main_task".

```
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
    v1->params[0].value.b = 1000 * v3; //millisecond,as the given
value
    return result;
}
```

Millisecond of TEE will be written to ptmx->fops pointer and “ret2user” exploit can be triggered.

4. Vulnerability in TEE OS (CVE-2015-4422)

When executing SMC instruction, a physical address pointed to TC_NS_SMC_CMD structure will be sent to TEE. A malformed TC_NS_SMC_CMD give me a chance to write one byte to almost any physical address.

```
typedef struct tag_TC_NS_SMC_CMD{
    unsigned int uuid_phys; //physical addr of uuid
    unsigned int cmd_id;
    unsigned int dev_file_id;
    unsigned int context_id;
    unsigned int agent_id;
    unsigned int operation_phys;//physical addr of kernel buffer
    unsigned int login_method;
    unsigned int login_data;
    unsigned int err_origin;
    bool started;
} TC_NS_SMC_CMD;
```

As there is no bound-checking in “TEEGlobalTask”, we can modify any physical memory except the memory used by TEE kernel. For example, let’s review the pseudo-code of get_sys_time() in “TEEGlobalTask”:

```
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 is a physical address in Secure World
    v1 = dword_5E2E0->operation_phys;

    //operation_phys+4 and operation_phys+8  will be modified
    *(int*)(operation_phys + 4) = v2;
    *(int*)(operation_phys + 8)= 1000 * v3;
    return result;
}
```

“Operation_phys+4” and “operation_phys + 8” will be covered by internal time of TEE. If current internal time of TEE is “0x0000AABB,0xCCDDEEFF”, covered value in memory must be “BB AA 00 00,FF EE DD CC” in little endian system. The “BB” is the last byte of second and cycle from 0x00 to 0xFF.

With this vulnerability we can write a specified byte to any physical address at a right second.

5 Code execution in TEE OS

Based on the above information, a local application can modify physical memory only accessed by TEE. In this section let's talk about how to achieve a code execution under TEEGlobalTask context.

With reversing its firmware and some black-box testing on Hisilicon's TEE,I find some good news and bad news. The good news is that there's few vulnerability mitigation features in TEE, no ASLR, no DEP, no "r-x" .text section, no stack canaries…almost nothing but a traditional RTOS inside. The bad news is we don't know patching where may change the original code flow and jump to our shellcode. In other words we need to know the base of "TEEGlobalTask".

Another interface provided by "TEEGlobalTask" has an interesting function named "ALLOC_EXCEPTION_MEM".
It's allows that normal world provide a physical address to "RTOSck", the kernel of TEE. RTOSck may write some crash information to this given physical address when task in TEE crashed. This physical address can be accessed by normal world, of course.

So I request SMC with an invalid physical address and make it crash. Here's a example of RTOSck's crash information:

```
DCD 0x2EF7D7A8          ; [g_crash_task_info]
DCD 0
DCD 0x100C0
DCD 0x1000              ; stack size
DCD 0x2E1FEF50          ; stack_top
DCD 0
DCD 0x47454554          ; TaskName
DCD 0x61626F6C
DCD 0x7361546C
DCD 0x6B
DCD 0x55667788          ; END_FLAG
DCD 0x11223344
DCD 0x2E1FEF50          ; [g_crash_task_STACK_info] stack top
DCD 0x2E1FFF50          ; stack bottom
DCD 0x2EF7D7A8          ; current stack pointer
DCD 0xFF2827A8
DCD 0xCBC
DCD 0
DCD 0x55667788          ; END_FLAG
DCD 0x11223344
DCD 0x60000110          ; [register_info] CPSR R0~R12 LR PC
DCD 0
DCD 0x2E1FFF1C
DCD 0x2E15E38C
DCD 0x2E15E2D0
DCD 0x3FE79400
DCD 0x2E15E37C
DCD 0x2E1FFF1B
DCD 0x2E1FFF1C
DCD 0x2E15E360
DCD 0x2E1FFF1B
DCD 0x2E1FFF1C
DCD 0x11111111
DCD 0x2E1FE140          ; SP
DCD 0x2EF00BBC          ; LR
DCD 0x2E103050          ; PC
```

This info shows that $PC of crash is 0x2E103050.So I suspect the base of "TEEGlobalTask" is 0x2E100000. Now I can patch some code to "TEEGlobalTask", make it jump to my own shellcode in normal world. Then trigger patched code execution from outside, all done.

6 Get fingerprint image from sensor

Only “TA_Fingerprint” can read fingerprint image from sensor by calling syscall “__FPC_readImage”.
Unfortunately the code execution exploit is under “TEE_GlobalTask” context, fingerprints reading request will be denied because of security check made by RTOSck.

So I need to find a vulnerability on RTOSck to bypass the security check.

```
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;
}
}
```

This is a syscall in RTOSck without any input check, I can overwrite memory in TEE kernel to patch the check code.

Now the code execution exploit is able to get fingerprints image by calling “__FPC_readImage”.

7 Conclusion

With these exploits, a local application is able to get fingerprint images or other encrypted data, disable signature verification of modem image and TA, load any module to TEE and modify the efuse data.

Even though TrustZone is designed for solving security problems, an implementation with security issues is still exploitable.

References

[1] http://www.arm.com/products/processors/technologies/trustzone/index.php
[2] http://genode.org/documentation/articles/trustzone

Responsible Disclosure

These vulnerabilities were disclosed to Huawei PSIRT in March 2015,a fix was provided by Huawei in May 2015.CVE IDs were assigned as CVE-2015-4421 and CVE-2015-4422.