

TRUSTZONE安全攻防

申迪, QIHOO360

个人介绍

- * 申迪 @retme / retme7@gmail.com
- * 来自奇虎360的安全研究员
- * 从事Android相关的安全研究工作
- * 爱好主机游戏、动漫



TRUSTZONE简介

科普时间 XD

什么是TrustZone?

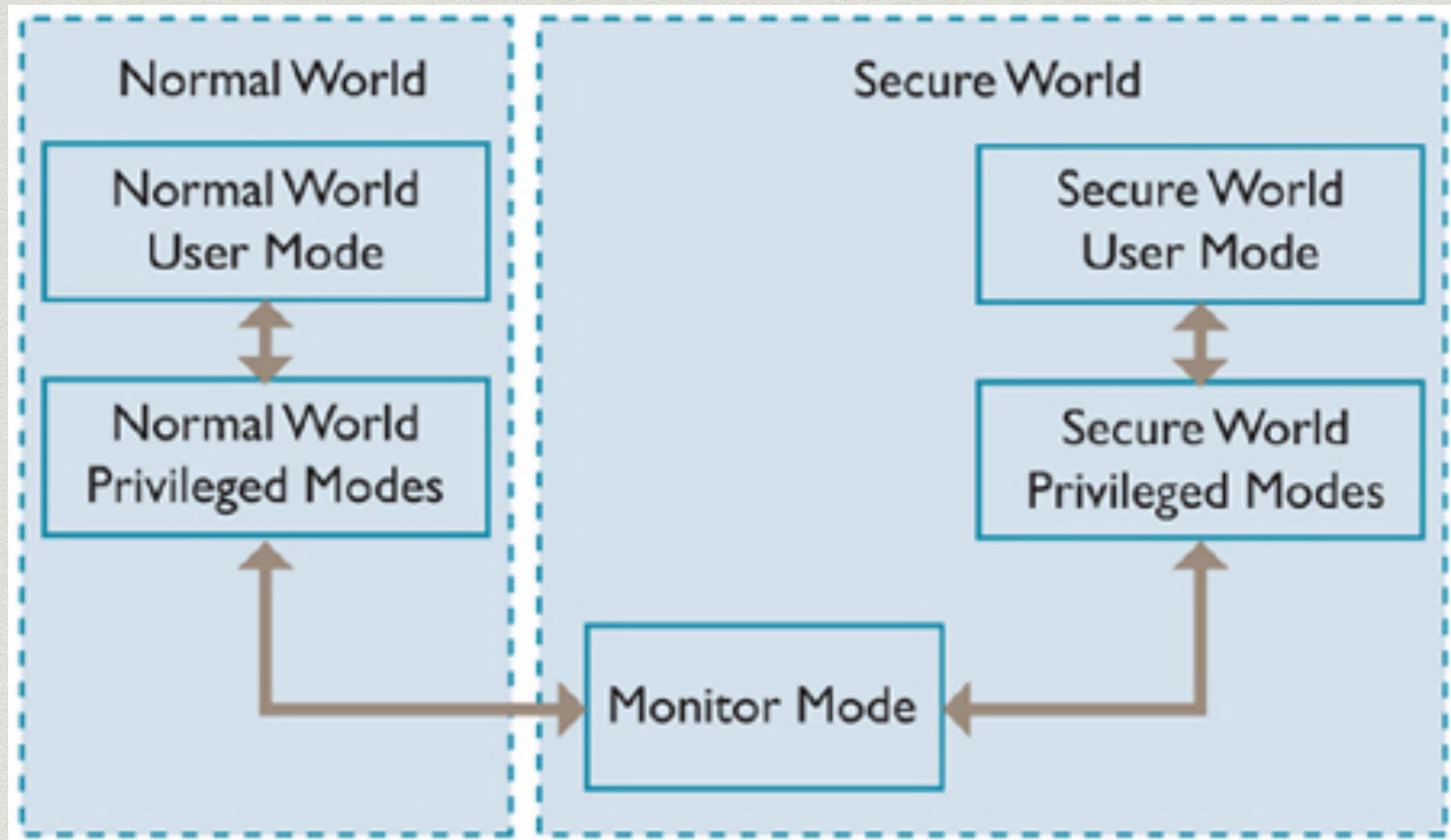
- * ARM处理器提供一个特殊的“安全模式”
- * 受信执行环境(TEE)运行于安全模式下
- * 将安全相关的逻辑放置于TEE执行，以避免遭受常规系统下的攻击
- * 相关应用：身份认证、DRM、BYOD、安全引导、内核安全监控.....



受信执行环境(TEE)

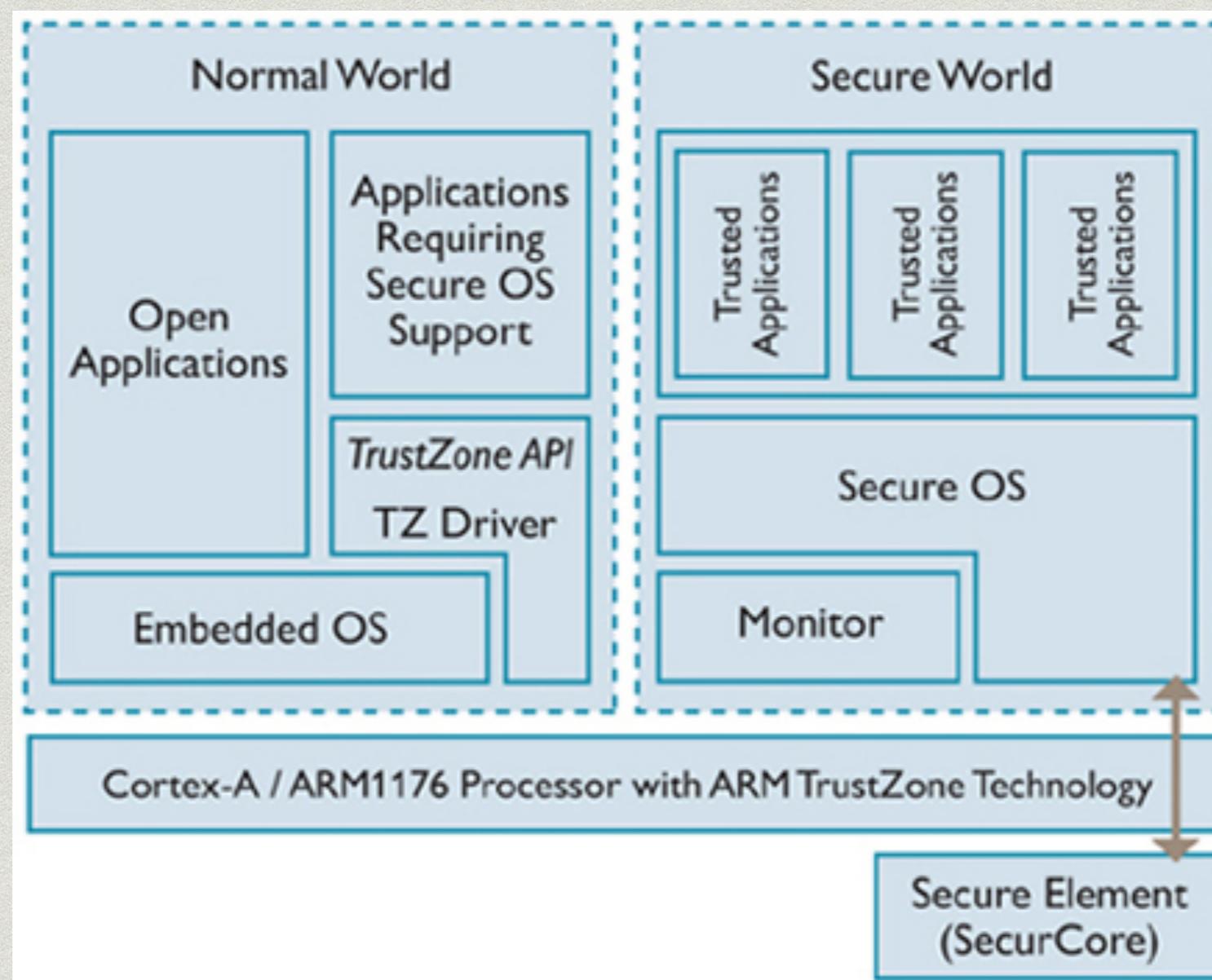
- * 额外运行一个操作系统(TEE OS)
- * 内部可运行一些可信程序(TA)
- * TA运行于用户模式下，OS代码运行于特权模式
- * 标准化：GlobalPlatform TEE API
- * 开源：OP-TEE Trusted OS

TrustZone硬件架构



图片引自<HTTP://WWW.ARM.COM/PRODUCTS/PROCESSORS/TECHNOLOGIES/TRUSTZONE/INDEX.PHP>

TrustZone软件架构



- * Secure World几乎可访问Normal world所有资源,反之则不能

进入安全模式

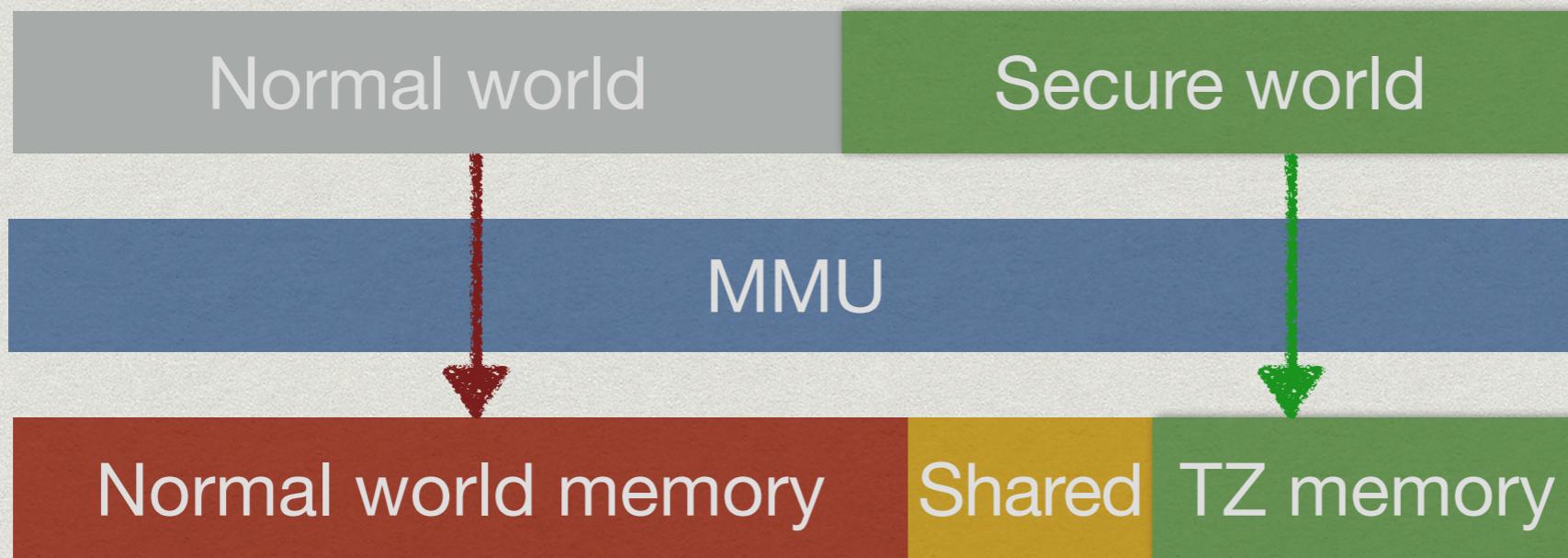
- * Secure Monitor Call(SMC) 是一个特权指令.
- * 硬件异常(IRQ,FIQ,etc.....)

Secure Monitor Call

- * 普通OS下的应用向内核发起请求
- * 内核使用特权指令Secure Monitor Call (SMC)，向安全OS发起调用请求
- * 安全OS处理请求，将结果返回给普通模式指定的缓冲区（物理地址）
- * 所以攻击安全OS需要首先获得普通OS中的内核代码执行权 @_@

TrustZone的内存管理

- * TEE OS有独立的页表、页目录
- * 有自己的物理内存区，Normal World不可访问
- * 可以访问Normal World的物理内存



更多信息可参考 “BUILDING A SECURE SYSTEM USING TRUSTZONE® TECHNOLOGY”

为什么研究TrustZone

- * TZ已经12岁了...现在被频频提及
- * 移动支付和移动安全的兴起
- * 芯片厂商各自实现，可能存在~~问题~~
- * 能够拿到比手机内核更高的权限，很有趣

缩略词

- * TZ: TrustZone
- * TEE: Trusted Execution Environment
- * TA: Trusted Application
- * NW/SW : Normal/Secure World
- * SMC: Secure Monitor Call

“Trust” Zone

 thomas lim 转推了

 **Dan Rosenberg** @djrbliss · 3月11日

@jduck @i0n1c Don't be ridiculous. TrustZone is 100% secure. It even has "trust" right there in the name!



 3

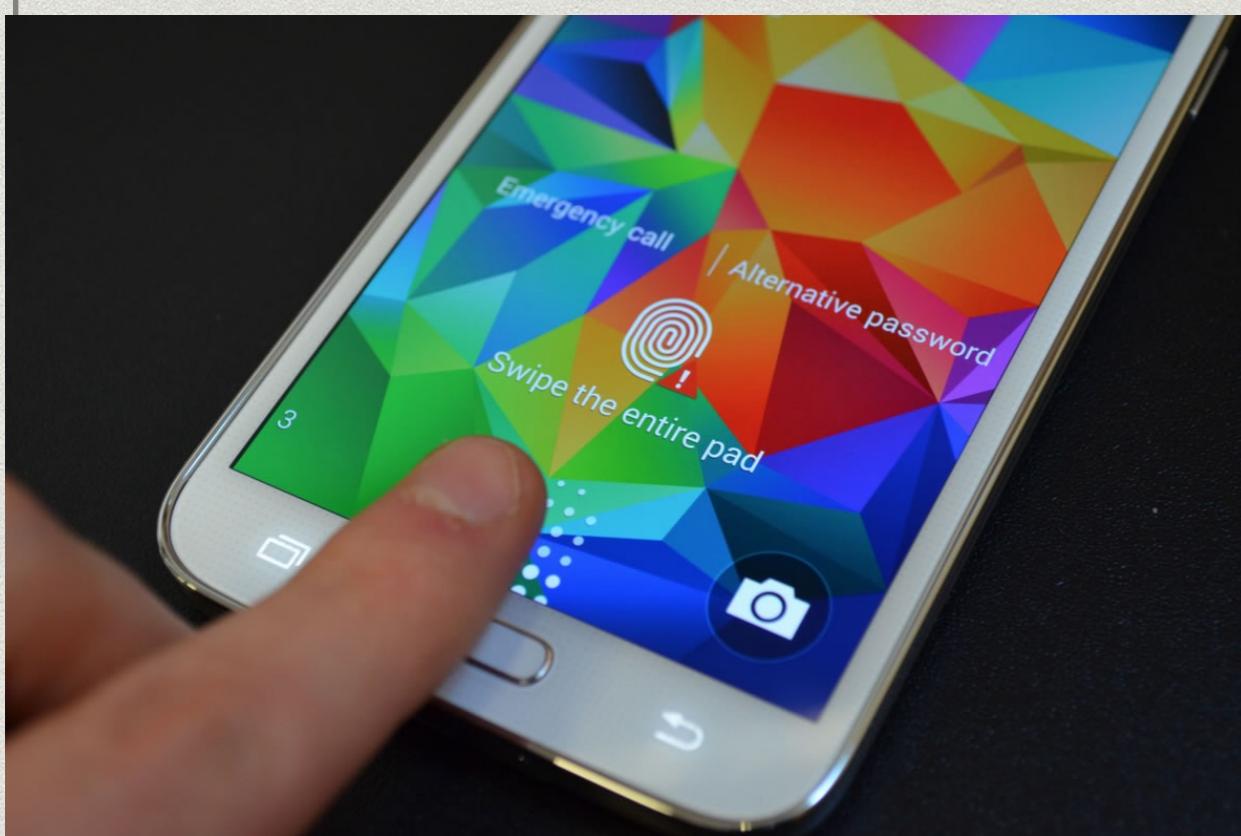
 7

...

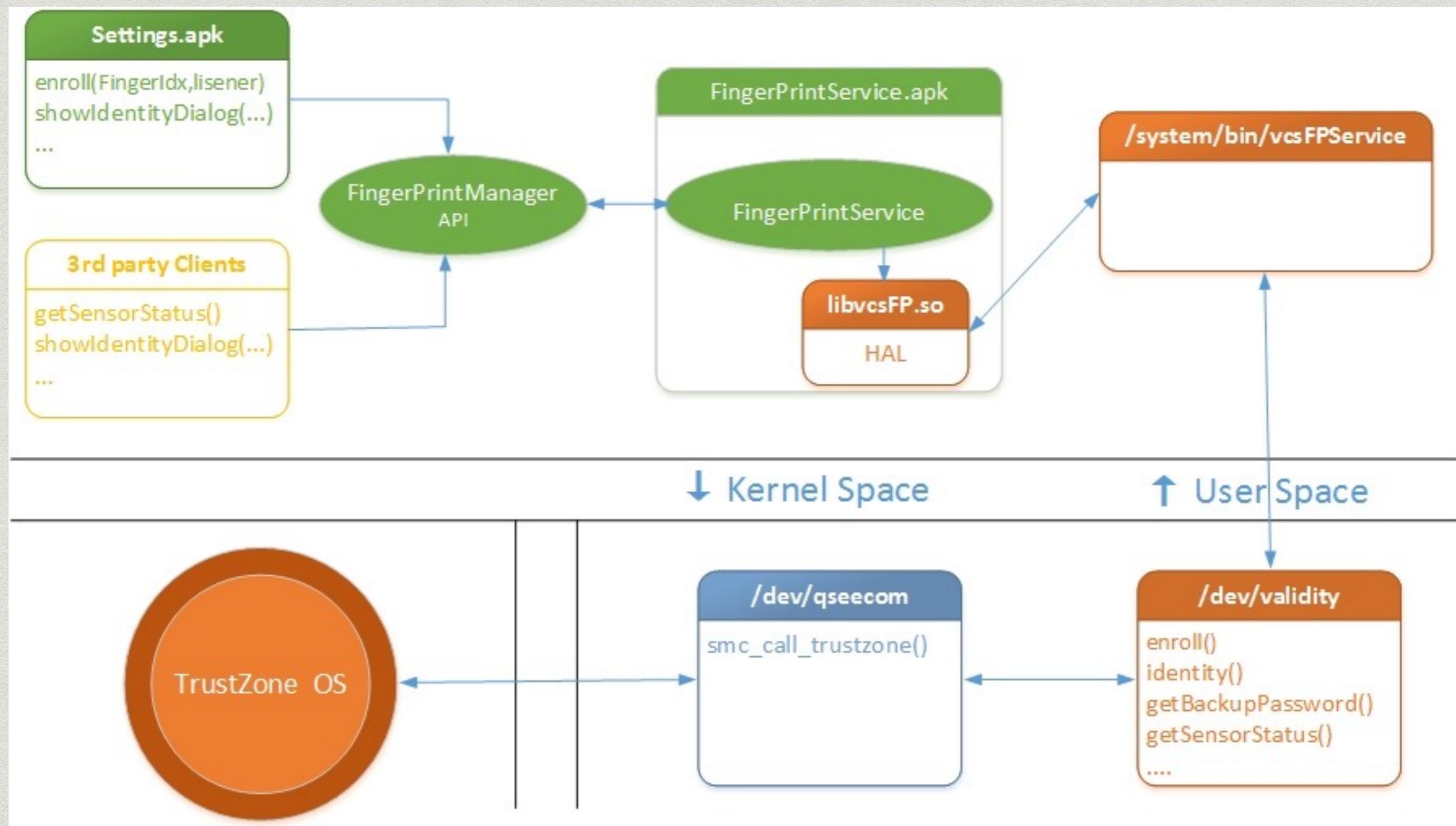
查看对话

一些已有的相关研究

- * "Reflections on trusting TrustZone" - Dan Rosenberg, 2014
- * "An Infestation of Dragons: Exploring Vulnerabilities in the ARM TrustZone Architecture" - Josh "m0nk" Thomas, Charles Holmes, Nathan Keltner, Atredis Partners, 2014
- * "Qualcomm TrustZone Integer Signedness bug" - Frédéric Basse, 2015



指纹识别调用流程示意



攻击面

- * FingerPrintService 是否对调用者权限进行了校验？
- * 内核中负责发起SMC调用的驱动程序是否校验了调用者权限？接收用户传递参数时是否存在安全漏洞？
- * TrustZone中的受信程序(TA)接收输入参数时是否存在安全漏洞？

可能存在一些漏洞

- * 内核驱动中可能存在一些安全漏洞，也许帮助我们获取内核的root权限？
- * 利用TrustZone技术运行的SW中，是否存在一些漏洞，能够让我们NW中侵入SW？
- * 在TEE OS中执行代码，获取想要的东西，禁用不想要的功能，就是一次完美利用！

为什么研究Mate7?

- * Kirin 925 处理器
- * “TrustedCore”
- * TEE 内核 : RTOSck
- * 没有任何文档可参考
- * 根据fingerprints.com,是市场上第一个搭载指纹系统的安卓手机
- * 热门机型，官方推特(@HuaweiDevice)宣布发行首月销量破百万



寻找内核漏洞
先看看源码？

下载内核源码

- * 内核源码是“不可信的朋友”
- * 一份很老旧的代码，其中有很多漏洞，但大都已经修复
- * 从源码中可以理解用户层与/dev/tc_ns_client进行ioctl通信的格式
- * 一些有意思的注释，其中的信息也许永远无法从IDA里面获得

/dev/tc_ns_client

- * 执行SMC指令的驱动程序
- * 任何应用程序都可与之通信，权限666，无SE policy上的访问限制
- * 让我们看看接受用户的参数是否安全。 . .



/dev/tc_ns_client

- * 类似于高通方案中的/dev/qseecom，负责提供API与TEE OS通信
- * 调用者传入的参数地址将会复制到内核缓冲区中，再传给TEE OS，返回时亦然。
- * **同时接受**用户态和内核态的调用请求，只是处理逻辑略有不同
 - * 对于用户态调用，传入的内存要使用copy_to/from_user中转
 - * 对于内核态调用，内存直接使用memcpy进行中转

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;
```

从用户态传入一个内核地址，会怎样？

驱动处理参数传递伪代码

```
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 > 0xffffffff){
        //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 > 0xffffffff){
        *(u32 *)client_param->value.b_addr = operation->params[i].value.b;
    }
    else{
        copy_to_user(...);
    }
    ...
}
```

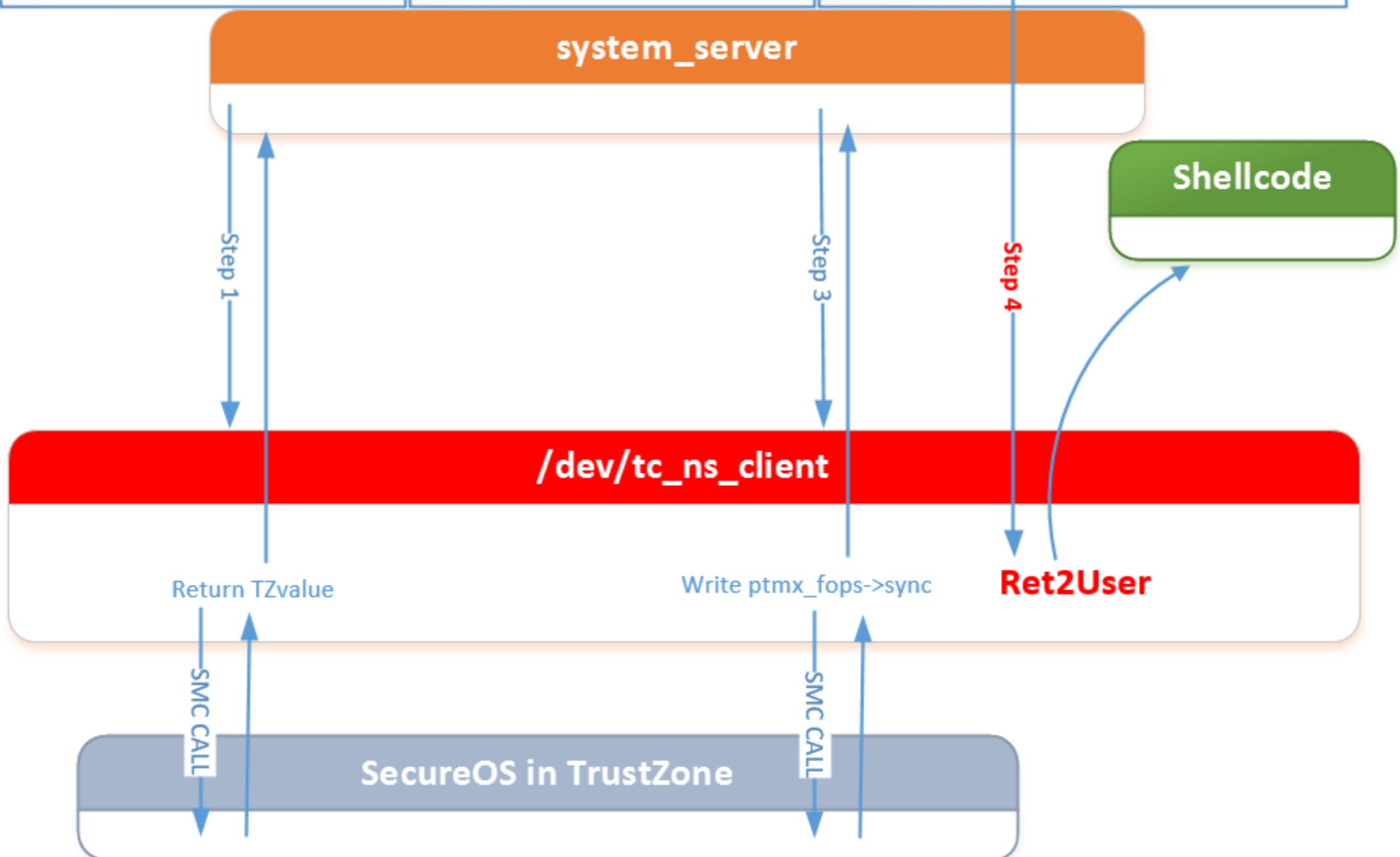
任意地址写漏洞

- * 草率地用“传入地址是否大于0xbfffffff”判断调用者是来自内核还是来自用户态
- * 用户态传入一些内核地址，可以在SMC调用返回时，由驱动程序把这个值写向传入的**任意内核地址**
- * 如果写回的值是一个相对稳定的值，则可用于提权利用

任意地址写固定值的利用思路

- * 进行一次正常调用，获得TEE中返回的固定值 value。并确保 `min_mmap_addr < value < 0xffffffff`
- * 向驱动传入`/dev/ptmx` 的`&fops.sync`，内核会将 value 写入指针表
- * 将 value 看做一个虚拟地址，在 value 处 map 一段内存并部署 shellcode，功能是完成权限提升
- * 调用 `sync(/dev/ptmx)` 触发 ret2user

Step1 smc_call(&TZvalue)	Step2 malloc(TZvalue) *(TZvalue) = shellcode	Step3 Offset = ptmx_fops->sync - buff *(ptmx_fops->sync) = TZvalue Step4 Call sync(“/dev/ptmx”)
-------------------------------------	---	--



距离root只差最后一步...

- * TEE 是否会返回一个这样的固定值?
- * 需要分析TEE 的提供的接口， 找到一个能返回固定值得方法

分析TEE OS

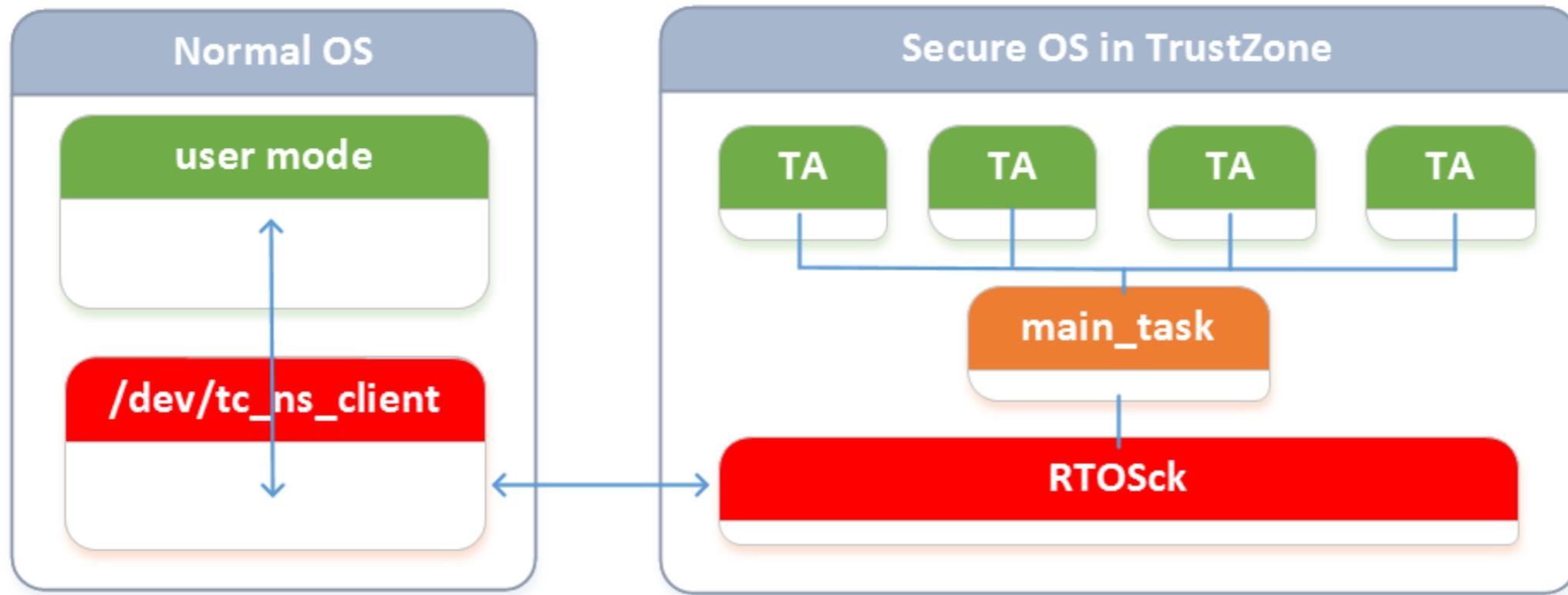
首先, 打开IDA....

提取TEEOS.img

- * 从恢复固件中提取，使用HuaweiUpdateExtractor
分解出TEEOS.img
- * 或从/dev/block/platform/ff1fe000.dwmmc0/by-name中提取
- * TEEOS.img中有很多明文字符串，应该没有加密
- * 那么直接拖进IDA

```
    {
        sub_E0059724();
        if ( get_current_status() == 0xFFFF3024 ) // target_dead
        {
            GetSystemTime(v32, v7);
            print_log("%s %s%d.%d %s: killing crash task.\n", "[error]", "[TEE_TIME]", v36, v33, "main");
            v26 = kill_crash_task();
            if ( v26 )
            {
                v27 = v26;
                GetSystemTime(v32, v25);
                print_log(
                    "%s %s%d.%d %s: Failed to kill crash task. ret_kill_crash=0x%x\n",
                    "[error]",
                    "[TEE_TIME]",
                    v36,
                    v33,
                    "main",
                    v27);
            }
        }
    }
```

- * 很多日志帮助我们理清函数功能和调用关系



* 根据逆向推测TEEOS的结构

TEEOS.img中的文件布局

- * 0x1400开始存储了各个模块的数据
- * RTOSck:实时操作系统，类似于μC/OS-II.负责处理系统调用，也包含指纹识别驱动。运行于Secure world的特权层.
- * main_task:主服务，作用是加载其他自服务(TA)，接收来自于Normal world的调用请求，进行派发。也提供一些基础功能，比如时间查询、崩溃回调注册。
- * 多个可信应用(TA):ELF文件，在需要时被主服务加载起来提供服务

TEE OS中提供的主要服务

- * TEE_SERVICE_GLOBAL (in main_task)
- * TEE_SERVICE_STORAGE
- * TEE_SERVICE_CRYPTO
- * TEE_SERVICE_EFUSE
- * TEE_SERVICE_HDCP
- * TEE_SERVICE_KEYMASTER
- * TEE_SERVICE_SECBOOT

main_task中的时间查询服务

- * cmd_id = GLOBAL_CMD_ID_TEE_TIME
- * 可以查询TEE OS自启动之后的运行秒数与毫秒数
- * 这就是我们想要的返回固定数值的接口！
- * 内核提权的最后一块拼图.

main_task中的时间查询服务

```
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;
    v1->params[0].value.b = 1000 * v3;
    return result;
}
```

提升至内核权限(普通模式)

- * 秒和毫秒两个4字节可写入ptmx->fops指针
- * 秒数0x000000xx < mmap_min_addr=4096，不能作为mmap的虚拟地址
- * 毫秒是0x00xxxxxx，可以使用
- * 按照之前的思路 (ret2user) 取得root

绕过 PXN / 代码只读？

- * 这个漏洞不仅仅是“任意地址写固定值”
- * 是否可以转化成任意地址写任意值？
- * 当然可以！ 虽然利用起来不如ret2user 稳定高效
- * 然而mate 7并没有支持pxn 0.0

寻找TEE OS中的漏洞 TEE内的安全性如何呢？

调用TEE的功能

- * 获取root之后，已经可以在内核模式下发起SMC，直接和TEE通信
- * 先来看看通信传递的结构体

```
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;
```

回过头来再看看时间查询接口？

```
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;
}
```

- * TC_NS_SMC_CMD被RTOSck映射到0x5e2e0

任意物理地址写入漏洞

- * 对operation_phys输入没有做任何校验
- * 若秒数是0xAABBCCDD，则每次可对operation_phys + 4开始的位置写入“DD CC BB AA”四个字节
- * “DD”是秒的最后4字节，在00 ~ FF之间循环
- * 在恰当的秒写入如想要的一个字节 —— 任意物理地址写入任意内存漏洞

利用TEE OS中的漏洞 有好消息也有坏消息

好消息是RTOSck...

- * 没有ASLR
- * 没有NX
- * 没有代码段只读保护
- * 没有stack canaries
- * 恩，啥都没有

坏消息是...

- * 不清楚模块基地址
- * 不清楚其内存布局
- * 读取不到实时日志，它们似乎只向UART输出的
- * 没有调试手段，开发exploit困难
- * 一个彻底地“未文档黑洞”，啥也不知道

逆向RTOSck收集信息

- * main_task不能访问所有物理内存，只能访问被RTOSck映射到页表上的内容
- * RTOSck存在**地址信息泄露**。它将在task崩溃时向用户指定的一块物理内存中写入崩溃信息。
- * RTOSck总是将崩溃时系统内的一级页表写入物理内存0x3FE79400，此处是Normal OS可访问的一段内存区

从崩溃信息获取模块基址

- * 传入一个不存在的 operation_phys造成崩溃
- * 从崩溃信息中找到崩溃时的PC，计算出 main_task的地址
- * 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
```

patch4字节代码

```
alloc_exception_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]
    MOU        R0, R5 ; int
    MOU        R1, R4 ; int
    BL         map_memory
    MOU        R0, R5
    MOU        R1, R4
    LDMFD      SP!, {R3-R5,LR}
    B          syscall_f084
; End of function alloc_exception_mem
```

patch前

```
syscall_f084 ; C
    STMFD      SP!, {LR}
    SUC        0xF084
    LDMFD      SP!, {LR}
    BX         LR
; End of function syscall_f084
```

patch后

```
; void __cdecl patch_syscall(int, int) ; C
patch_syscall
    STMFD      SP!, {LR}
    BLX        R1
    LDMFD      SP!, {LR}
    BX         LR
```

触发漏洞

- * 在内核中使用kmalloc申请一段内存，在内存中部署 shellcode
- * 计算出内存的物理地址 kernel_pool_phy
- * 向TEE发起调用: cmd = GLOBAL_CMD_ID_ALLOC_EXCEPTION_MEM
- * 传参(0,kernel_pool_phy)
- * TEE调用 syscall_f084(0,kernel_pool_phy)
- * 触发BLX R1; R1= kernel_pool_phy

EXPLOIT开发举例

我们能做什么呢？

禁用SE for Android

- * patch代码永远是最简单粗暴的禁用
- * 如果在在Normal world中patch内核代码有点麻烦，需要改变页面只读属性
- * 但在TEE OS中修改物理内存没有限制
- * 直接patch `avc_has_perm`放行一切SELinux规则查询

patch TEE内部的代码

- * 禁用对于Modem分区的哈希校验
- * 禁用TEE中对于TA的签名校验，从Normal OS中加载TA到SecOS

API HOOK

- * Hook一些TA中的关键TEE API
- * 拦截指纹相关API

存取加密数据

- * 数据都存储在/sec_storage_data下， 可通过文件API存取
- * 对efuse进行读写

还可以...安装一个rootkit

- * TEE可以用来监控Normal OS，也许我们也可以？
- * 不要担心隐藏问题，并没有人会在TEE里装杀毒软件 XD
- * 有人要写一个TA模块用来杀毒么？ =.=

总结

- * 都是“时间”惹的祸：内核提权和TEE提权都依赖 main_task中的时间查询接口
- * TEE中虽然漏洞缓解措施几乎都没，但难度在于摸清其内部实现（内存布局、进程、接口等等）和 exploit调试困难（仅有crash的PC已知）
- * 安全的设计也需要安全的实现，请严格校验调用者传入的参数：)

[**https://github.com/retme7/mate7_TZ_exploit**](https://github.com/retme7/mate7_TZ_exploit)

致谢

- * HUAWEI PSIRT Team
- * 360手机卫士团队
- * GongGuang (@oldfresher)

QUESTIONS?

RETME7@GMAIL.COM

[@RETME](https://twitter.com/RETME)

THANK YOU :)