CVE-2017-13253 :Android Drm服务 堆溢出漏洞

miy1z1ki / 2019-09-01 10:22:00 / 浏览数 3373 安全技术 移动安全 顶(0) 踩(0)

0x00 前言

跟着[1]调试CVE-2017-13253，
[1]使用的调试环境为nexus，HAL的实现没有分开，我的调试环境为pixel，HAL的实现分开了，我在这里整理成一篇文章，记录踩过的坑。
CVE-2017-13253 为Android Drm服务中的堆溢出漏洞。

0x01 理论

1 DRM

Android Drm 属于 Android Native 多媒体框架中的一部分。在播放受DRM保护的内容 如Google
Play电影中的影片时，会使用DRM服务器。该服务器会采用安全方式对加密的数据进行解密，因此可以访问证书和密钥存储以及其他敏感组件。但由于供应商依赖关系，DR
DRM 的架构如下图所示：

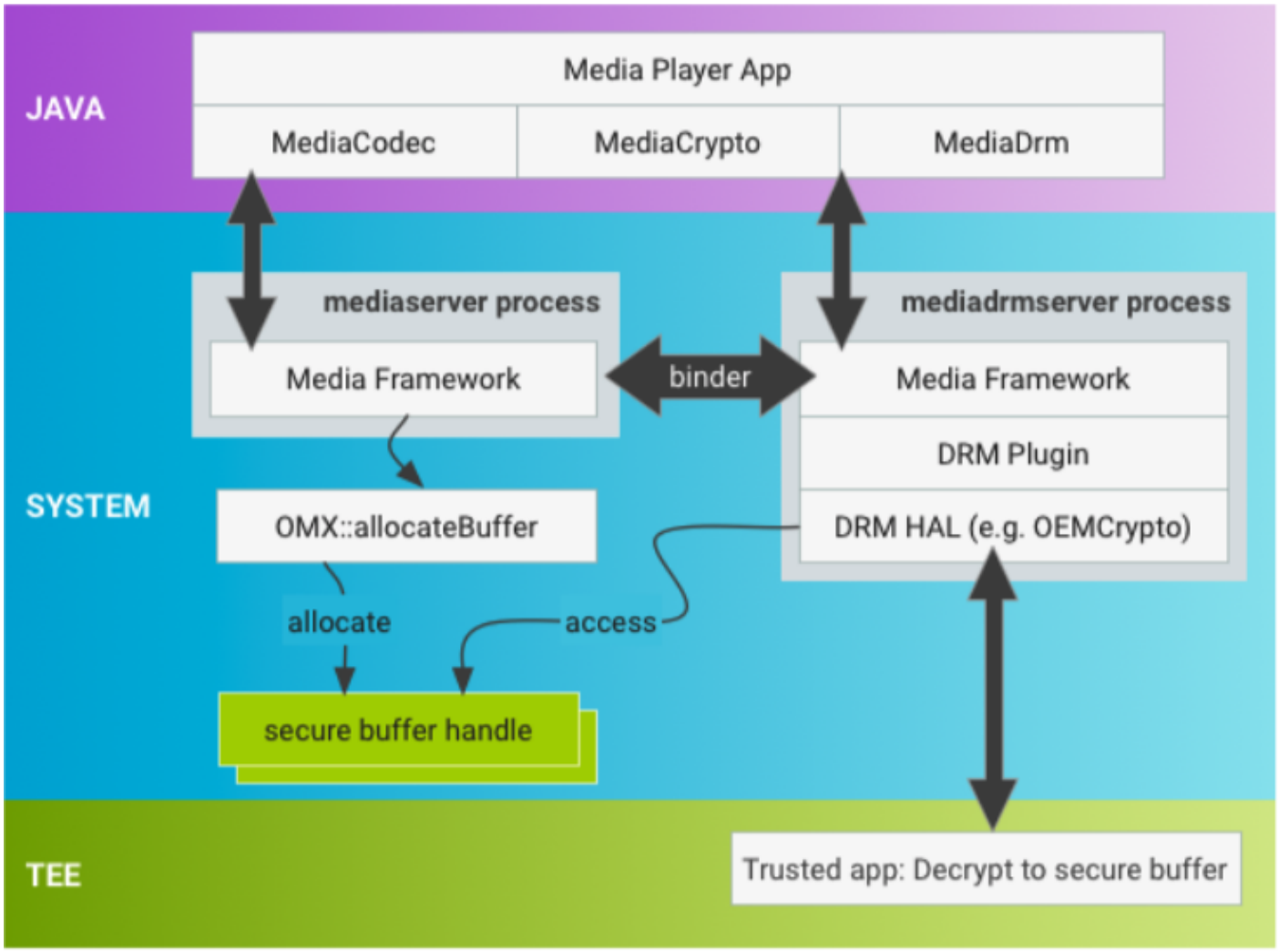


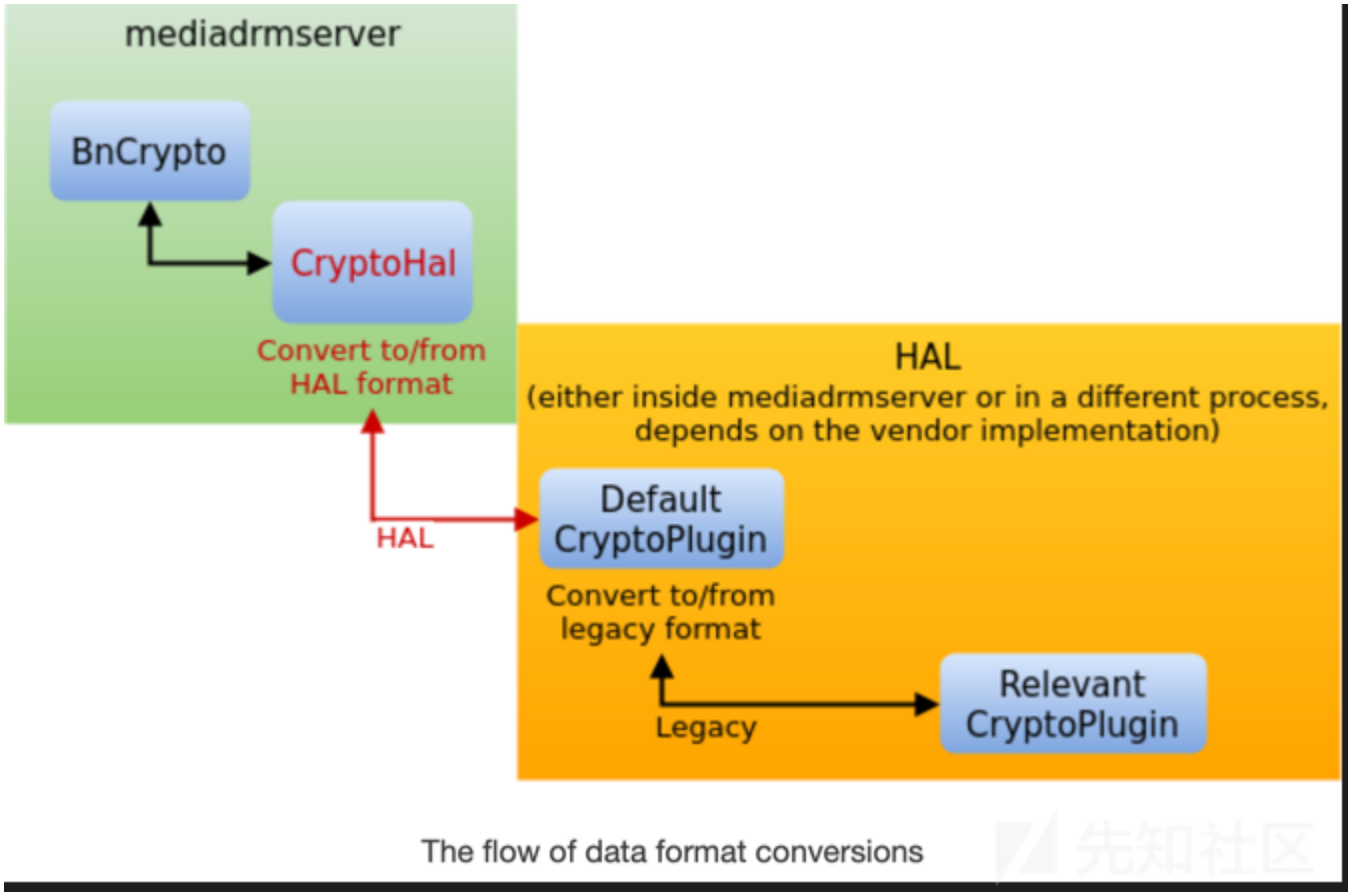
图 3. Android 7.0 及更高版本中的 mediaserver 中的缓冲区分配。

2 Crypto Plugins

Crypto Plugins
为DRM方案之一，在Android术语中，每个DRM方案的处理程序称为插件。供应商负责提供这些插件，但供应商可以使用AOSP中一些有用的代码。例如，AOSP包含Clear
DRM方案插件的完整开源实现。
java层 所涉及到的参数变量

Fields	
public byte[]	iv A 16-byte initialization vector
public byte[]	key A 16-byte key id
public int	mode The type of encryption that has been applied, see MediaCodec.CRYPTO_MODE_UNENCRYPTED, MediaCodec.CRYPTO_MODE_AES_CTR and MediaCodec.CRYPTO_MODE_AES_CBC
public int[]	numBytesOfClearData The number of leading unencrypted bytes in each subSample.
public int[]	numBytesOfEncryptedData The number of trailing encrypted bytes in each subSample.
public int	numSubSamples The number of subSamples that make up the buffer's contents.

https://developer.android.com/reference/android/media/MediaCodec.CryptoInfo.html#numSubSamples
Binder层的解释 这里引用原文的图



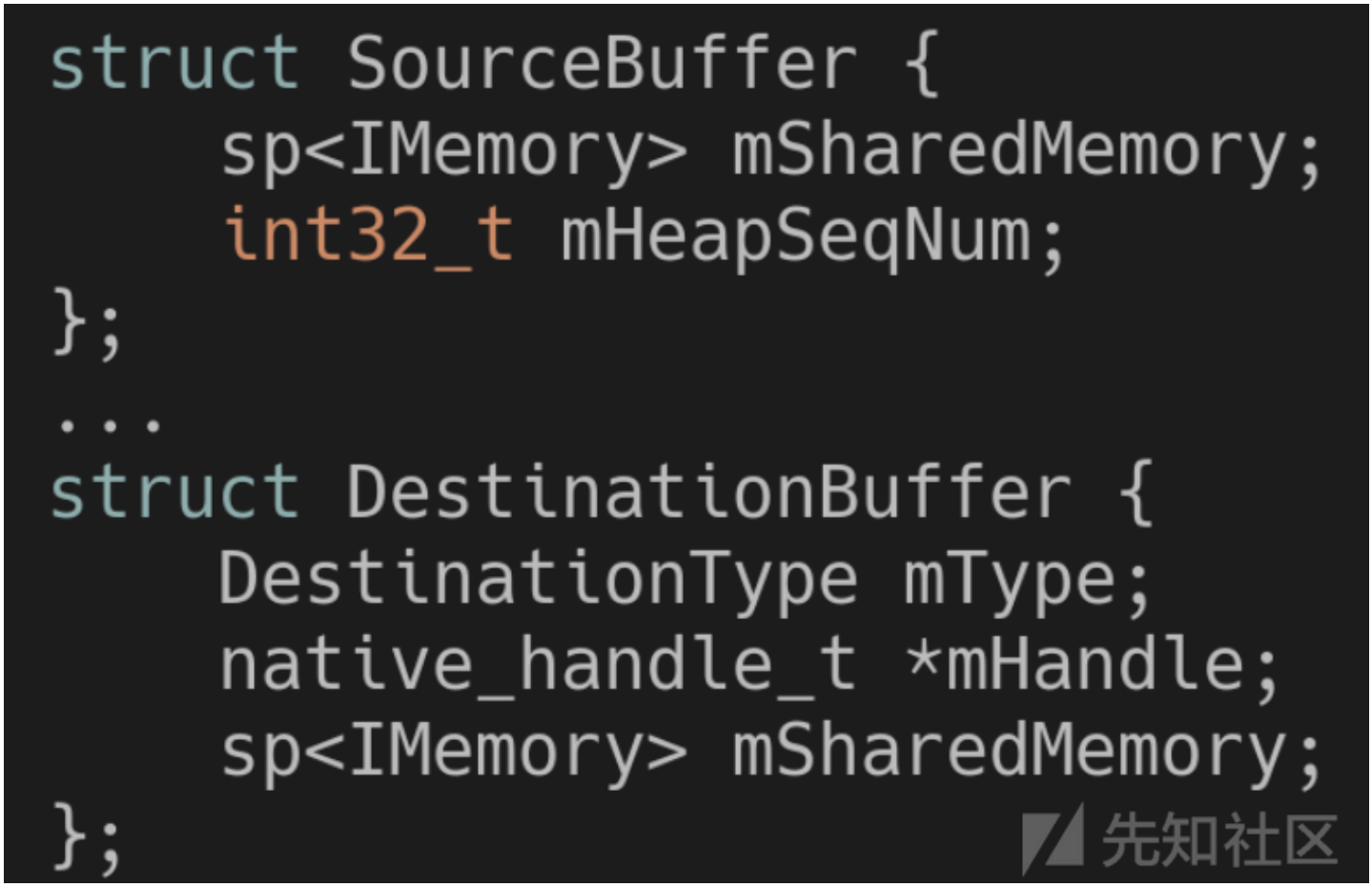
MediaDrmserver
提供了一个到Crypto对象。名字叫做ICrypto，现在改名叫CryptoHal。此接口的一般用途是允许非特权应用解密DRM数据，这需要更高的解密权限，例如访问TEE。
ICrypto里有很多种方法，毫无疑问最重要的方法是解密：

3 Binder's C++库

ICrypto应该是一个接口类，Binder’s
C++库里提供了许多依赖于Binder的C++代码的抽象。可以调用C++类的远程实例方法。使用此机制的每个对象在预定义的结构中实现几个类：

```
*  ■■■■■■■■■■Binder■■■■■■■■■■“I”■■■
* ”■■■“■■■■■■■■■■■■■■■■■■■■”Bp“■■■
* ”■■■■“■■■■■■■■■■■■■■■■■■■■”Bn“■■■
```

最终 在使用对象时，几乎总是使用接口类型。这里引用原文的图



这里相关的结构成员是mHeapSeqNum和两个mSharedMemory成员(DestinationBuffer的其余部分是在目标未存储为共享内存的情况下，与此漏洞无关的情况)。这里使用
mHeapSeqNum是这样的内存的标识符，之前使用ICrypto的方法（称为setHeap）共享。两个mSharedMemory成员仅表示堆内缓冲区的偏移和大小。这意味着尽管mHe
值得注意的是，参数结构的某些部分有点奇怪。
mSharedMemory是一个IMemory，它实际上连接到自己的堆，并且应该在其中表示一个缓冲区，但是这个堆被忽略，偏移量和大小用于mHeapSeqNum堆。源结构中也存
Treble的主要重新架构师的一部分而制作

4 源码

source.mSharedMemory 是什么含义呢 ，我猜是图里面的这个 shared
memory？仅表示堆内缓冲区的偏移和大小，实际上连接到自己的堆，应该用于表示一个缓冲区，但是这个堆被忽略，偏移量和大小用于mHeapSeqNum堆。
在ICrypto interface 的“服务端”代码，用于验证共享内存缓冲区的subsample。这段代码检查
subsample Encrypt Clear data都加起来 sum<= SIZE_MAX
subsampleSizes == totalSize
totalSize <= source.mSharedMemory->size()
Offset <= source.mSharedMemory->size() - totalSize

```
size_t sumSubsampleSizes = 0;
bool overflow = false;
for (int32_t i = 0; i < numSubSamples; ++i) {
    CryptoPlugin::SubSample &ss = subSamples[i];
    if (sumSubsampleSizes <= SIZE_MAX - ss.mNumBytesOfEncryptedData) {
        sumSubsampleSizes += ss.mNumBytesOfEncryptedData;
    } else {
        overflow = true;
    }
    if (sumSubsampleSizes <= SIZE_MAX - ss.mNumBytesOfClearData) {
        sumSubsampleSizes += ss.mNumBytesOfClearData;
    } else {
        overflow = true;
    }
}

if (overflow || sumSubsampleSizes != totalSize) {
    result = -EINVAL;
} else if (totalSize > source.mSharedMemory->size()) {
    result = -EINVAL;
} else if ((size_t)offset > source.mSharedMemory->size() - totalSize) {
    result = -EINVAL;
} else {
    result = decrypt(key, iv, mode, pattern, source, offset,
            subSamples, numSubSamples, destination, &errorDetailMsg);
}
```


http://androidxref.com/8.0.0_r4/xref/frameworks/av/drm/libmediadrm/ICrypto.cpp#370

接着进入 decrypt，对数据流进行序列化并继续进行验证

```
if (source.offset + offset + source.size > sourceBase->getSize()) {
    _hidl_cb(Status::ERROR_DRM_CANNOT_HANDLE, 0, "invalid buffer size");
    return Void();
}

uint8_t *base = static_cast<uint8_t *>
        (static_cast<void *>(sourceBase->getPointer()));
void *srcPtr = static_cast<void *>(base + source.offset + offset);

...
    if (destBuffer.offset + destBuffer.size > destBase->getSize()) {
        _hidl_cb(Status::ERROR_DRM_CANNOT_HANDLE, 0, "invalid buffer size");
        return Void();
    }
    destPtr = static_cast<void *>(base + destination.nonsecureMemory.offset);
...
ssize_t result = mLegacyPlugin->decrypt(secure, keyId.data(), iv.data(),
        legacyMode, legacyPattern, srcPtr, legacySubSamples,
        subSamples.size(), destPtr, &detailMessage);
```


http://androidxref.com/8.0.0_r4/xref/hardware/interfaces/drm/1.0/default/CryptoPlugin.cpp#111

sourceBase 指向 mSharedBuffer上的 source

'''

111 if (source.offset + offset + source.size > sourceBase->getSize()) {
112 _hidl_cb(Status::ERROR_DRM_CANNOT_HANDLE, 0, "invalid buffer size");
113 return Void();
114 }

'''

此处的本意是拷贝 的size = offset+所有的 subSamples 。

```
uint8_t *base = static_cast<uint8_t *>(static_cast<void *>(sourceBase->getPointer()));
Const SHaredBuufer& destBuffer = destination.nonsecureMemory;
Sp<IMemory> destBase = mSharedBufferMap[destBuffer.bufferId]
■■
source.offset + offset + source.size > sourceBase->getSize()
source.offset ■source ■heap■■■■■■
destBuffer.offset+destBuffer.size>destBase->getSize()
```

第一处检查是offsets和buffer size没有超出堆的size。SourceBase是堆，而source现在是source.mSharedMemory。
此处的本意是检查偏移+需要拷贝的数据是否有超出这段sharedMemory的size？？

另一处检查类似，但是是在destBuffer上执行。destBuffer 是 destination.mSharedMemory、destBase “same heap as”sourceBase。也就是说 destBuffer 和 sourceBuffer 都在同一段SharedMemory上。

最终，每个buffer简化为一个指向内存的指针，偏移现在是指针的一部分。
最后一处代码

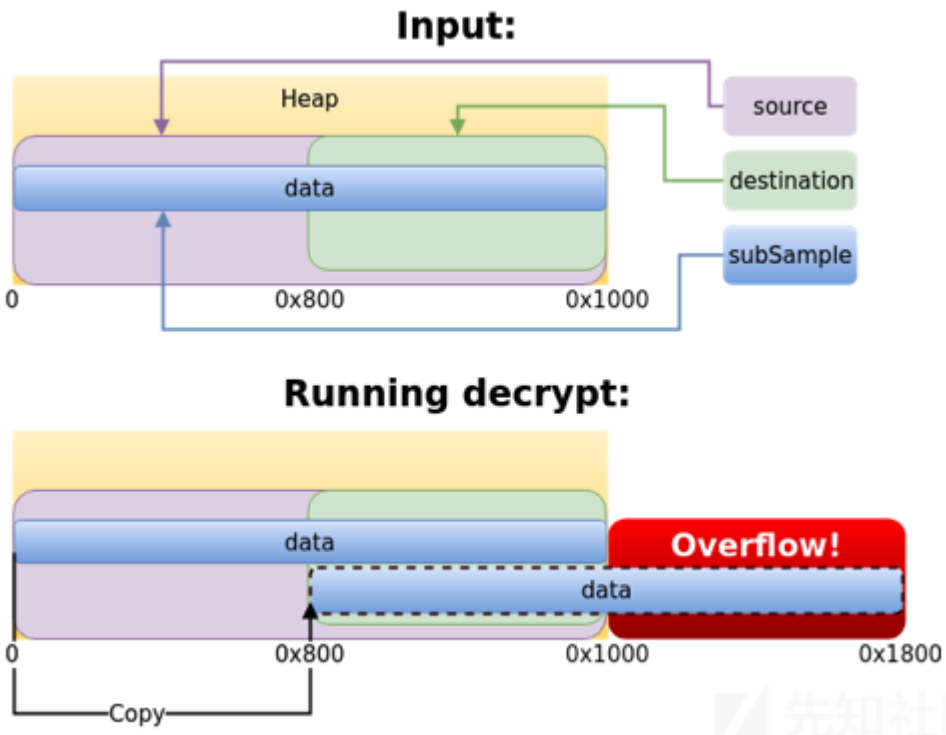
```
if (mode == kMode_Unencrypted) {
    size_t offset = 0;
    for (size_t i = 0; i < numSubSamples; ++i) {
        const SubSample& subSample = subSamples[i];

        if (subSample.mNumBytesOfEncryptedData != 0) {
            errorDetailMsg->setTo(
                    "Encrypted subsamples found in allegedly unencrypted "
                    "data.");
            return android::ERROR_DRM_DECRYPT;
        }

        if (subSample.mNumBytesOfClearData != 0) {
            memcpy(reinterpret_cast<uint8_t*>(dstPtr) + offset,
                   reinterpret_cast<const uint8_t*>(srcPtr) + offset,
                   subSample.mNumBytesOfClearData);
            offset += subSample.mNumBytesOfClearData;
        }
    }
    return static_cast<ssize_t>(offset);
```


http://androidxref.com/8.0.0_r4/xref/frameworks/av/drm/mediadrm/plugins/clearkey/CryptoPlugin.cpp#45
当 满足mode == kMode_Unencrypted时，
会执行到
'''
memcpy(reinterpret_cast<uint8_t>(dstPtr) + offset,
reinterpret_cast<const uint8_t>(strPtr) + offset,
subSample.mNumBytesOfClearData);
'''

4 漏洞处

漏洞原因：没有检查 要复制的数据+目标缓冲区的位置是否超过 堆的 size，属于检查不完整。

只有一个简单的检查 源缓冲区 ，BnCrypto的第三次检查了这一点，下一处的检查 考虑了源缓冲区+offset。唯一检查和目标缓冲区有关的是第二处的检查，但是太简单不足以阻止这种问题。 第二处的检查，确认目标缓冲区在堆内，且没有超出缓冲区的边界。



0x02 动态调试

动态调试[6][7][8]

在poc作者的github[9]里说到运行的结果应该是

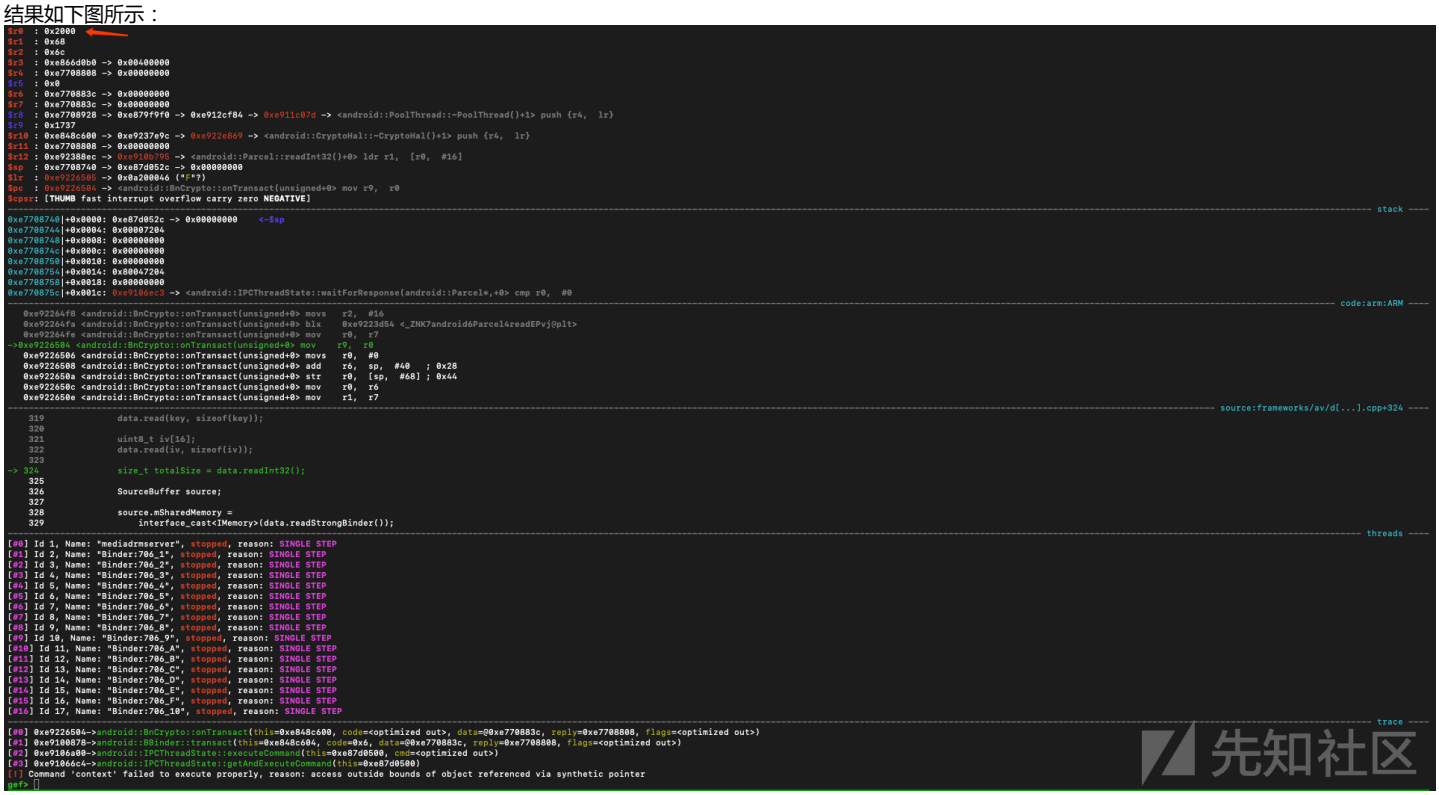
1. 如果在2018年3月之后的Android版本 decrypt 会返回bad_value（-22）。
2. 如果没有crash（overwritten data 是可写的）decrypt return 他copy的数据的量。
3. 如果供应商将HAL实现为单独的进程?（例如Pixel 2），则解密应该返回UNKNOWN_ERROR（-32）。
4. 如果供应商在同一过程中实施HAL?（例如Nexus 5X），则解密应返回0。

HAL的实现会影响是否造成Crash。此处没有返回UNKNOWN_ERROR

本漏洞涉及三处函数，分别是
android::BnCrypto::onTransact 序列化+验证
Android::CryptoHal::decrypt 序列化+验证
Clearkeydrm::CryptoPlugin::decrypt memcpy

前面两次进入BnCrypto::onTransact函数是调用createPlugin和setHeap，第三次调用decrypt
对android::BnCrypto::onTransact 、android::CryptoHal::decrypt的调试部分可参考[1]调试。

结果如下图所示：



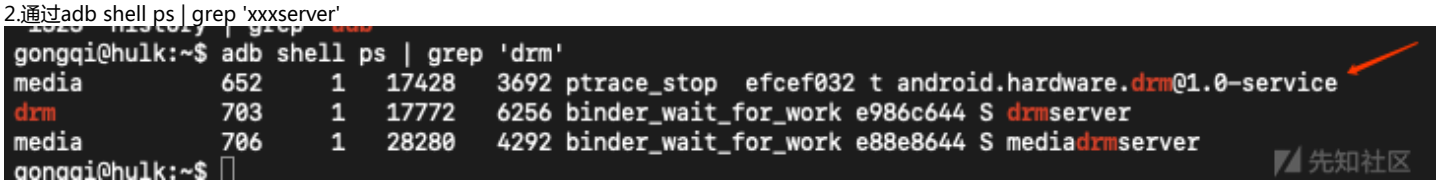
从反序列化种获取total值。这里能看到是0x2000

在对Clearkeydrm::CryptoPlugin::decrypt 实际调试时，供应商在实现的时候，对HAL分开实现。所以此处为多进程调试。

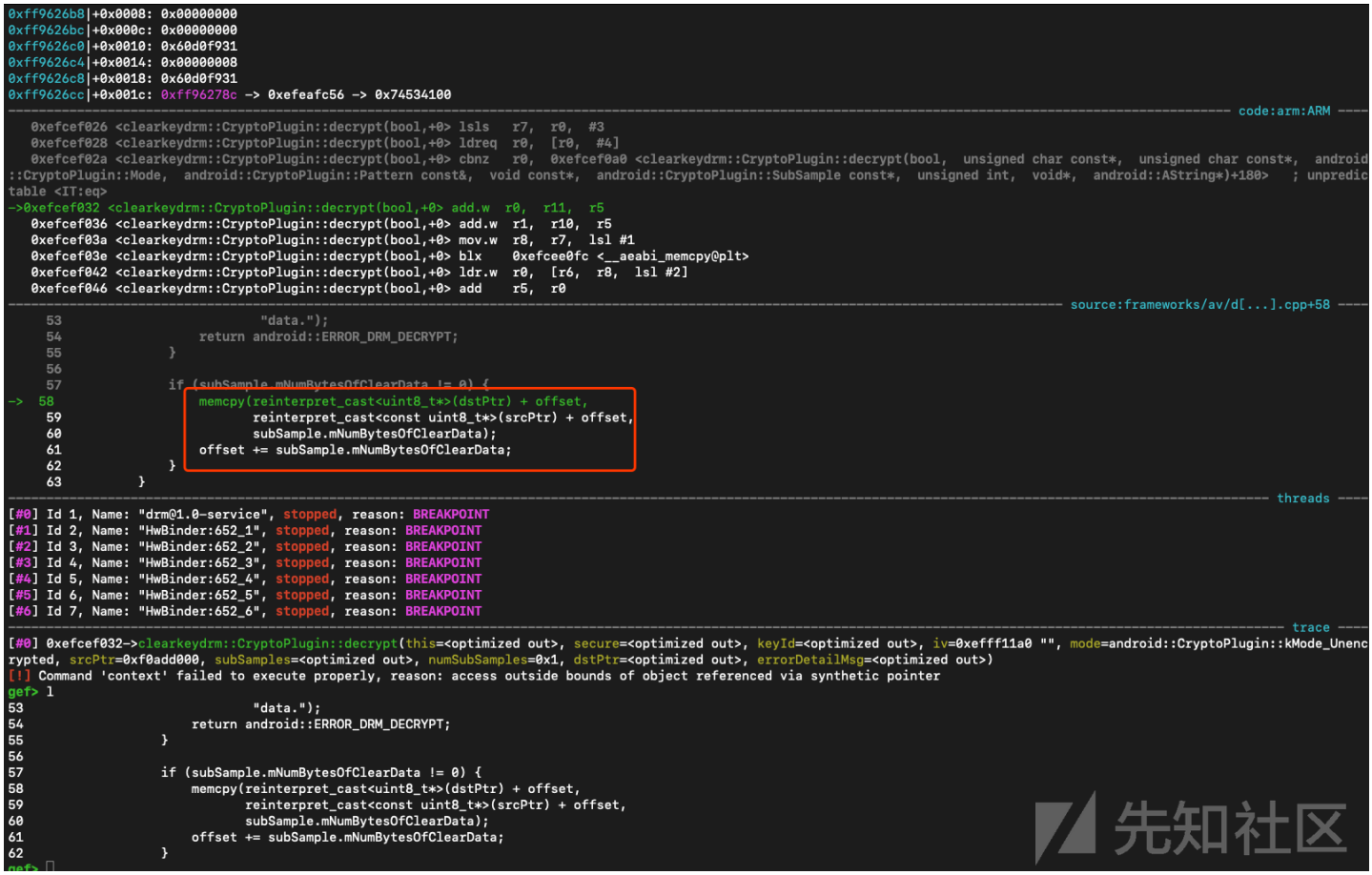
此处有两种方法找到binder的server端，如图所示

1.通过Android.mk or Android.bp文件 找到local_module

2.通过adb shell ps | grep 'xxxserver'



最后 找到漏洞点



0x03 结论与补丁

1

受影响的流程取决于供应商如何实现。如果供应商未将HAL分成不同的进程，则mediadrmserver受影响。如果供应商讲HAL分开，那么使用默认加密插件的Crypto插件的每

2

对这个漏洞的补丁很简单，增加对要传递数据检查的完整性， dest->size()
目标缓冲区堆的size，检查目标缓冲区的偏移+要拷贝的数据是否超出目标缓冲区堆的总和。

0x04 参考

[1] https://bbs.pediy.com/thread-225398.htm
[2]http://wangkuiwu.github.io/page2/ Android Binder机制
[3]https://www.anquanke.com/vul/id/1124887 漏洞简介
[4]https://source.android.com/devices/drm DRM，安卓版权管理框架
[5]https://developer.android.com/reference/android/media/MediaDrm.html MediaDrm
[6]https://source.android.com/setup/build/building 编译流程
[7]https://developers.google.com/android/drivers

[8]https://source.android.com/setup/start/build-numbers#source-code-tags-and-builds
[9]https://github.com/tamirzb/CVE-2017-13253

上一篇：渗透测试：从XSLT注入到Gets... 下一篇：渗透测试：从XSLT注入到Gets...
1. 0 条回复
   - 动动手指，沙发就是你的了！