Attack Android Through Mutiple Hiden Interface

CanSecWest 2018
Vancouver

Hanxiang Wen
Jiashui Wang

蚂蚁金服 ANT FINANCIAL | 支付宝 ALIPAY

About us

Hanxiang Wen

- ID: arnow117
- Security researcher at Ant-financial Light-Year Security Lab
- Focus on Android vulnerability research and exploit development

Jiashui Wang

- ID: Quhe
- Teamleader at Ant-financial Light-Year Security Lab
- Focus on mobile security and vulnerability hunting

About Light-Year security lab

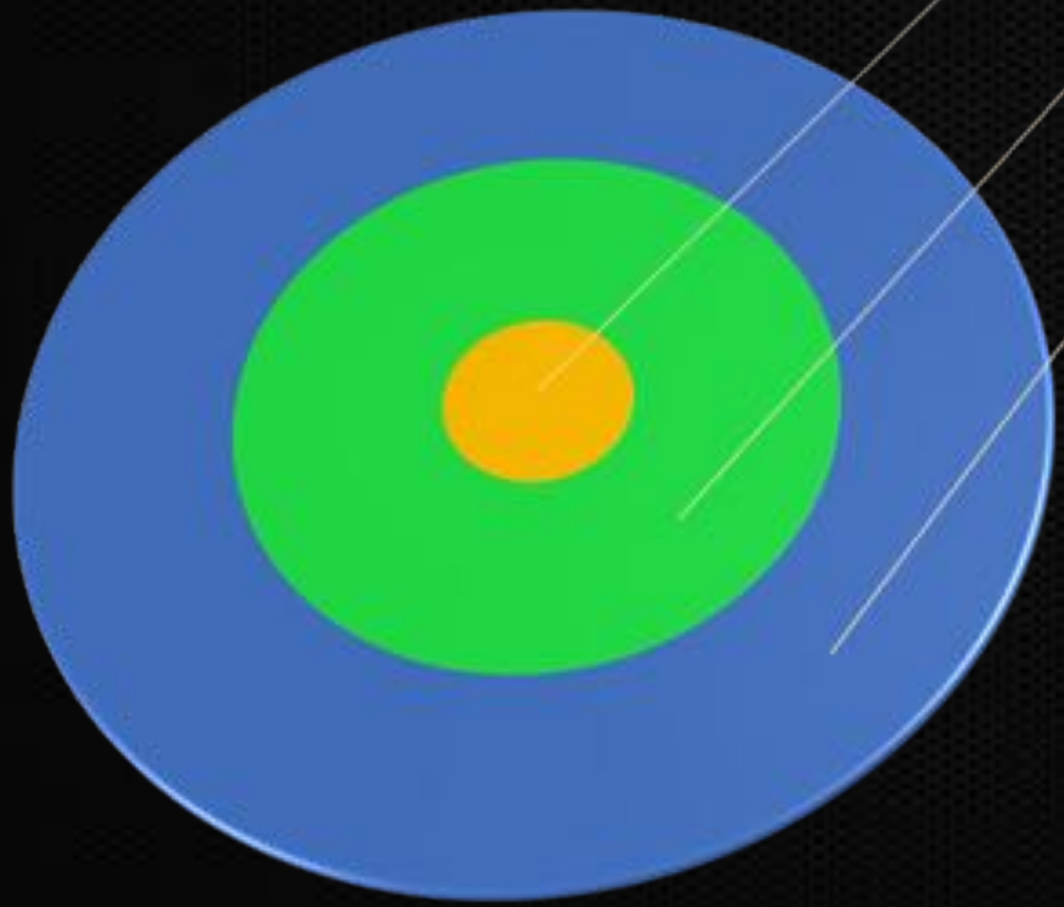
- Security research team in the field of financial payment security composed of senior security experts.
- Protect Alipay and Ant Financial-related products.
- Explore the security issues of external vendors, merchants, and eco-partners.

Agenda

- Start from FOTA
  - FOTA mechanism
  - FOTA Service Vulnerability
- LocalSocket to Remote Socket

- BufferQueue in Android graphic system
  - Fake GraphicBufferProducer/Consumer
- From GraphicBuffer to NativeHandle
  - Defects under Android HAL

Way of data transmission



Kernel

ioctl

Framework

binder

Application

socket

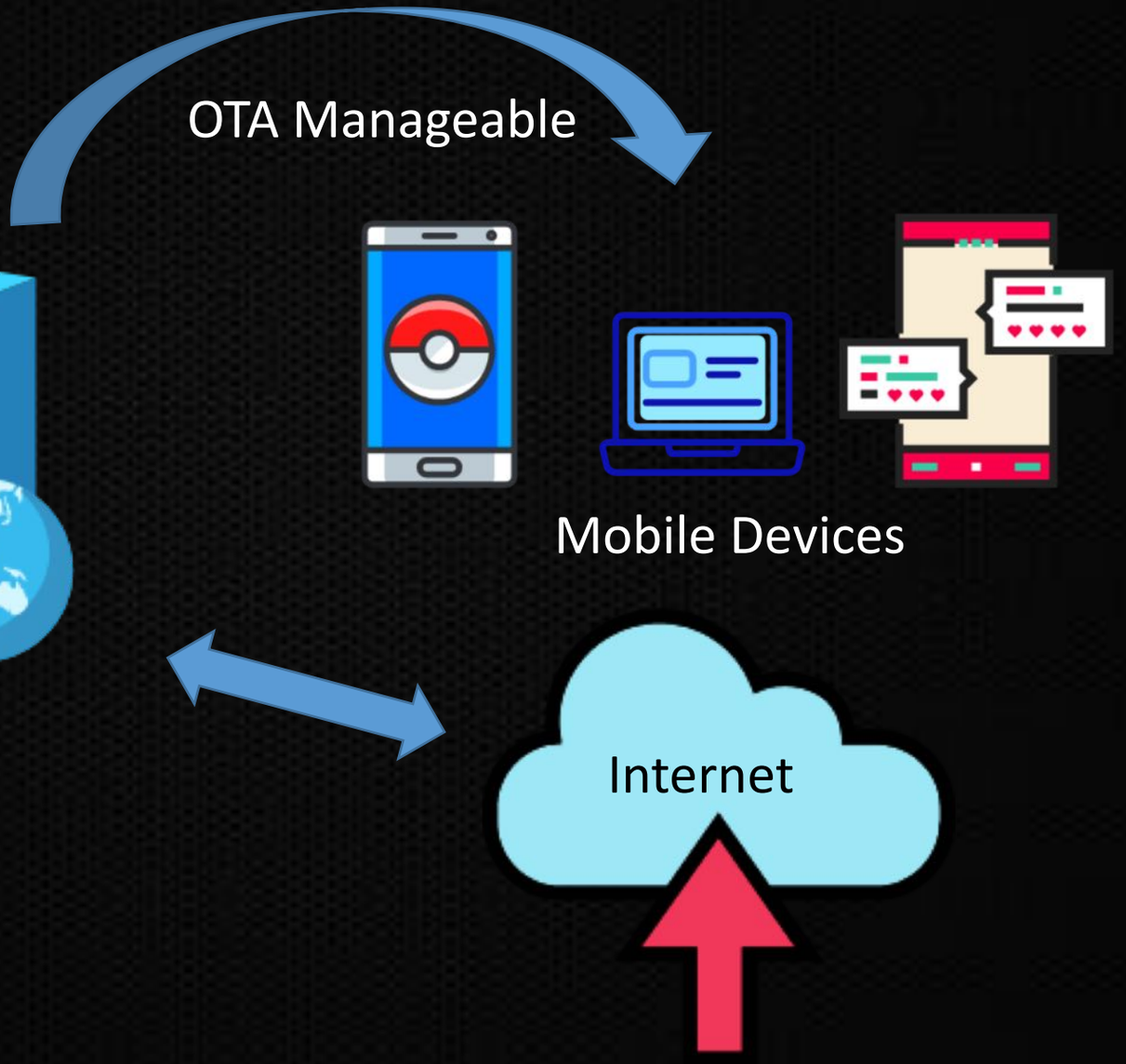
Background

System vs. Platform vs. Untrusted Apps

- main types of apps: System apps, Platform apps and Untrusted apps
- System Apps typically installed by Android system (e.g. PackageManager)
- Vendor can add system apps

System Apps	Platform Apps	Untrusted Apps
Installed when system is initialised	APK which uses android platform signature	Installed from app store, or from an APK
Can request system-only permissions	System-only permissions unavailable	System-only permissions unavailable
Can obtain permissions without user input	Set of non-system permissions can be requested from the user	Set of non-system permissions can be requested from the user

FOTA solutions



FOTA Service

- Firmware Over-the-Air (FOTA) updates

- Definition：

  A Mobile Software Management (MSM) technology in which the operating firmware of a mobile device is wirelessly upgraded and updated by its manufacturer.

FOTA in mobile

- FOTA facilitates the following:
  - Allow phone to repair bugs in new units
  - Allow phone to install new software updates, features and services remotely
- Essential service of Smart-phone
- Require at least system privilege

Acknowledgement


FOTA更新说明公告

日前，巴斯光年安全实验室向我们反馈了FOTA应用存在的权限漏洞，恶意攻击者可通过该bug获取手机System权限。对该漏洞给用户和合作伙伴可能造成的影响，我们深感不安，并第一时间发布修复版本，提交巴斯光年安全实验室和Google安全团队检测，目前已通过Google安全团队检测，请相关合作伙伴及用户及时更新。

个人用户

An elevation of privilege vulnerability in a MediaTek APK could enable a local malicious application to execute arbitrary code within the context of a privileged process. This issue is rated as High due to the possibility of local arbitrary code execution in a privileged process.

CVE	References	Severity	Updated Google devices	Date reported
CVE-2017-0522	A-32916158* M-ALPS03032516	High	None**	Nov 15, 2016

FOTA Vulnerability

- ADUPS provide Firmware Over the Air (FOTA) updates service, but exist a privilege elevation vulnerability
- At com.fw.upgrade.sysoper
- Gain system privilege easily
- Affect the world's at least 500 million Android phones

Vulnerability details

```
public abstract class b extends Binder implements a {
public b() {
super();
this.attachInterface(((IInterface)this), "com.adups.fota.sysoper.IService");
public boolean onTransact(int arg3, Parcel arg4, Parcel arg5, int arg6) {
boolean v0 = true;
switch(arg3) {
case 1: {
arg4.enforceInterface("com.adups.fota.sysoper.IService");
int v1 = this.a(arg4.readString());
arg5.writeNoException();
arg5.writeInt(v1);
break;
}
case 1598968902: {
arg5.writeString("com.adups.fota.sysoper.IService");
break;
}
default: {
v0 = super.onTransact(arg3, arg4, arg5, arg6);
break;
}
}
return v0;
```

Vulnerability details

```
private static int a(String[] arg7) {
int v5_2;
Process v4;
InputStream v3_1;
ByteArrayOutputStream v1_2;
ProcessBuilder v3;
InputStream v2 = null;
int v0 = -1;
try {
v3 = new ProcessBuilder(arg7);
v1_2 = new ByteArrayOutputStream();
}
catch(Throwable v0_1) {
v1_2 = ((ByteArrayOutputStream)v2);
v3_1 = v2;
v4 = ((Process)v2);
goto label_59;
}
```

```
try {
v4 = v3.start();
}
catch(Throwable v0_1) {
v3_1 = v2;
v4 = ((Process)v2);
goto label_59;
}
```

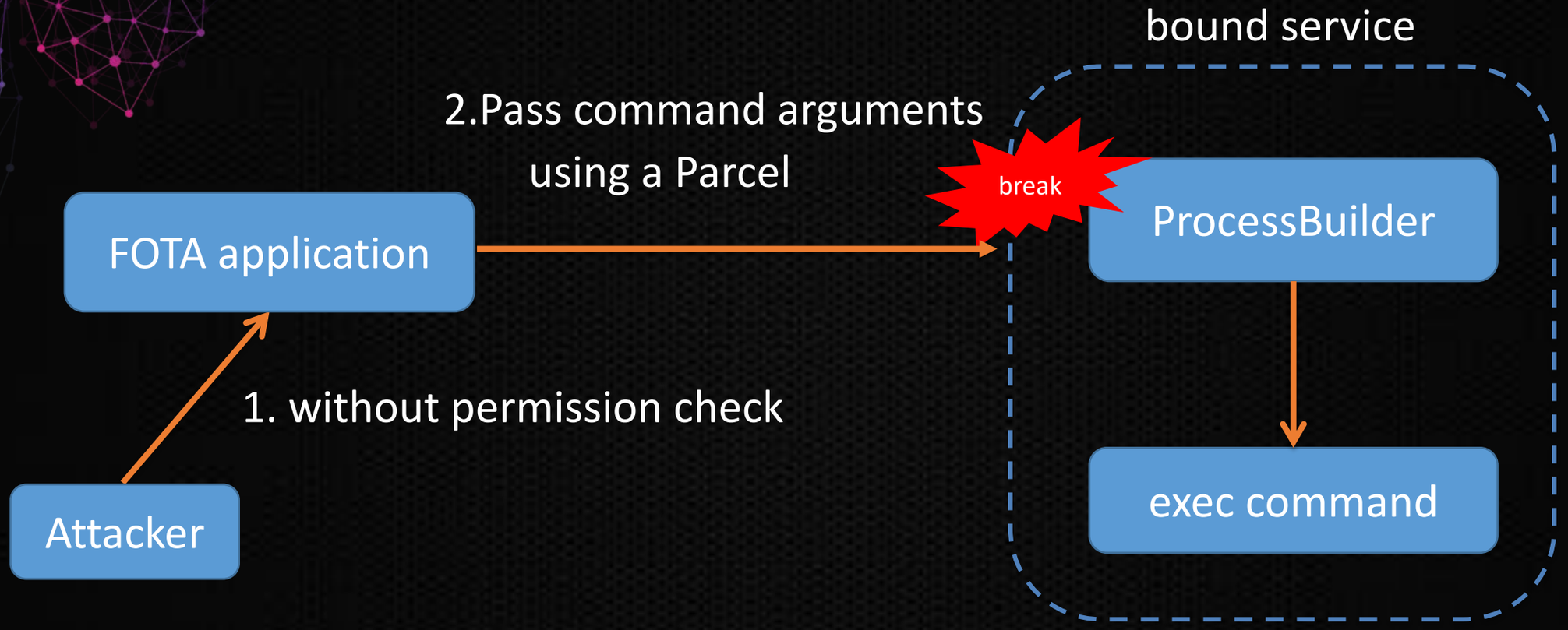
CVE-2017-0522

FOTA Vulnerability

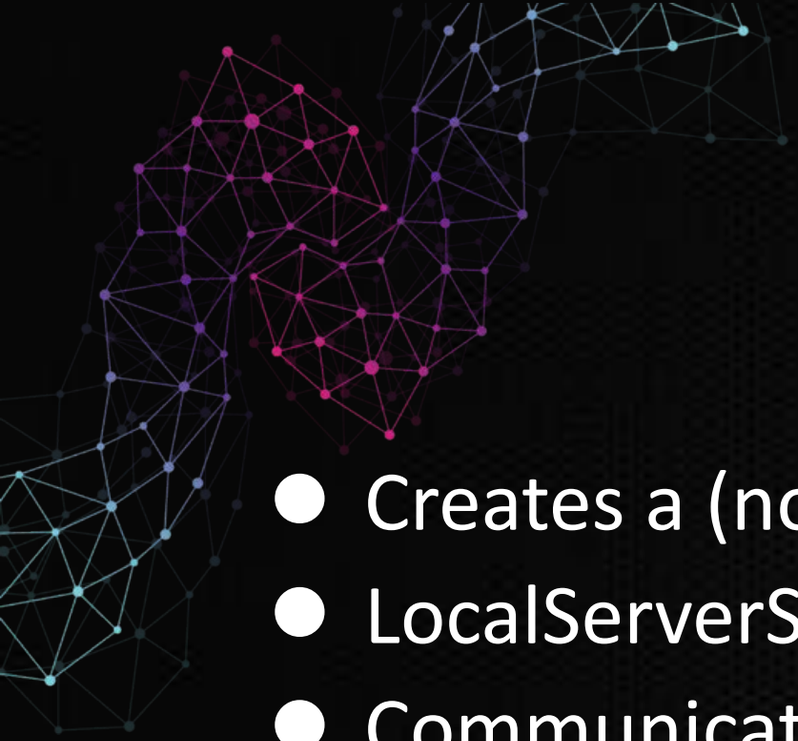
- sharedUserId=“android.uid.system”
- Local command execute as system user

```
<manifest android:sharedUserId="android.uid.system" package="com.fw.upgrade.sysoper"
xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-sdk android:minSdkVersion="11" android:targetSdkVersion="19" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.DEVICE_POWER" />
```

FOTA vulnerability exploit



CVE-2017-0522 exploit



Local socket to remote socket

- Creates a (non-server) socket in the UNIX-domain namespace
- LocalServerSocket and LocalSocket
- Communication between native and framework

Local socket to remote socket

Install process

- native server：

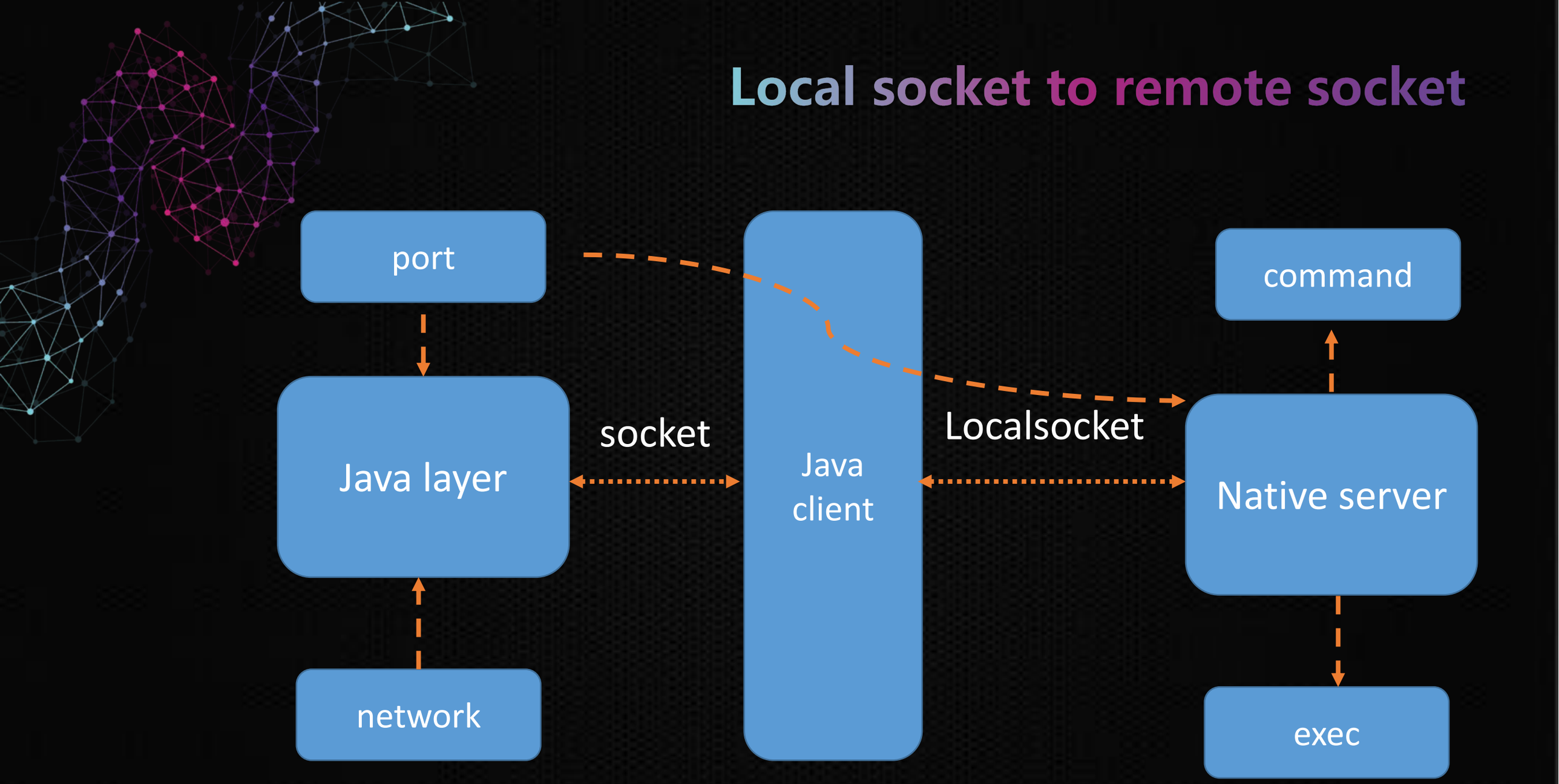
  \frameworks\base\cmds\installd\installd.c

- Framework client:

  \frameworks\base\services\java\com\android\server\pm\Installer.java

```
mSocket = new LocalSocket();
LocalSocketAddress address = new LocalSocketAddress("installd",
LocalSocketAddress.Namespace.RESERVED);
mSocket.connect(address);
```

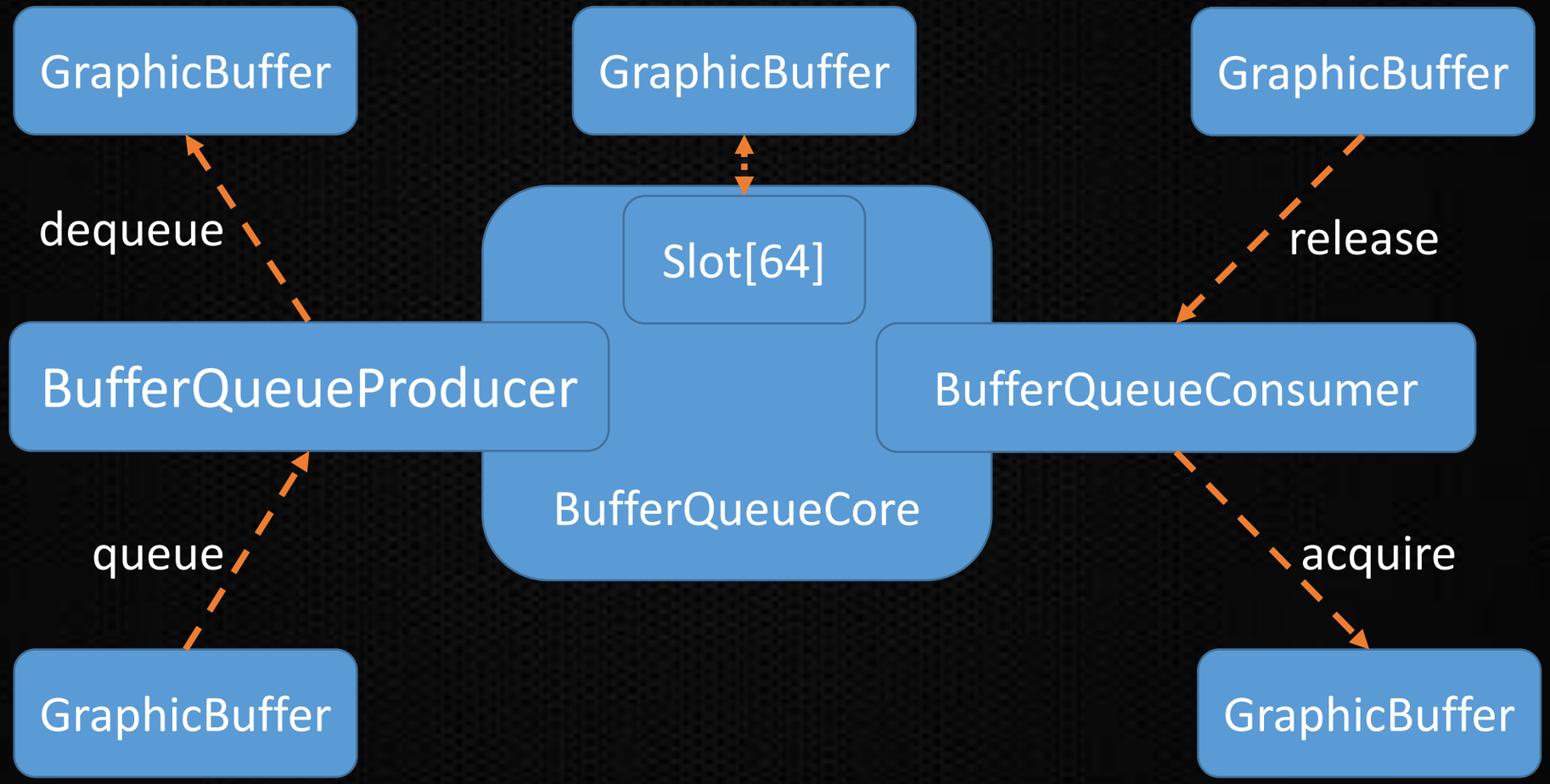
Local socket to remote socket



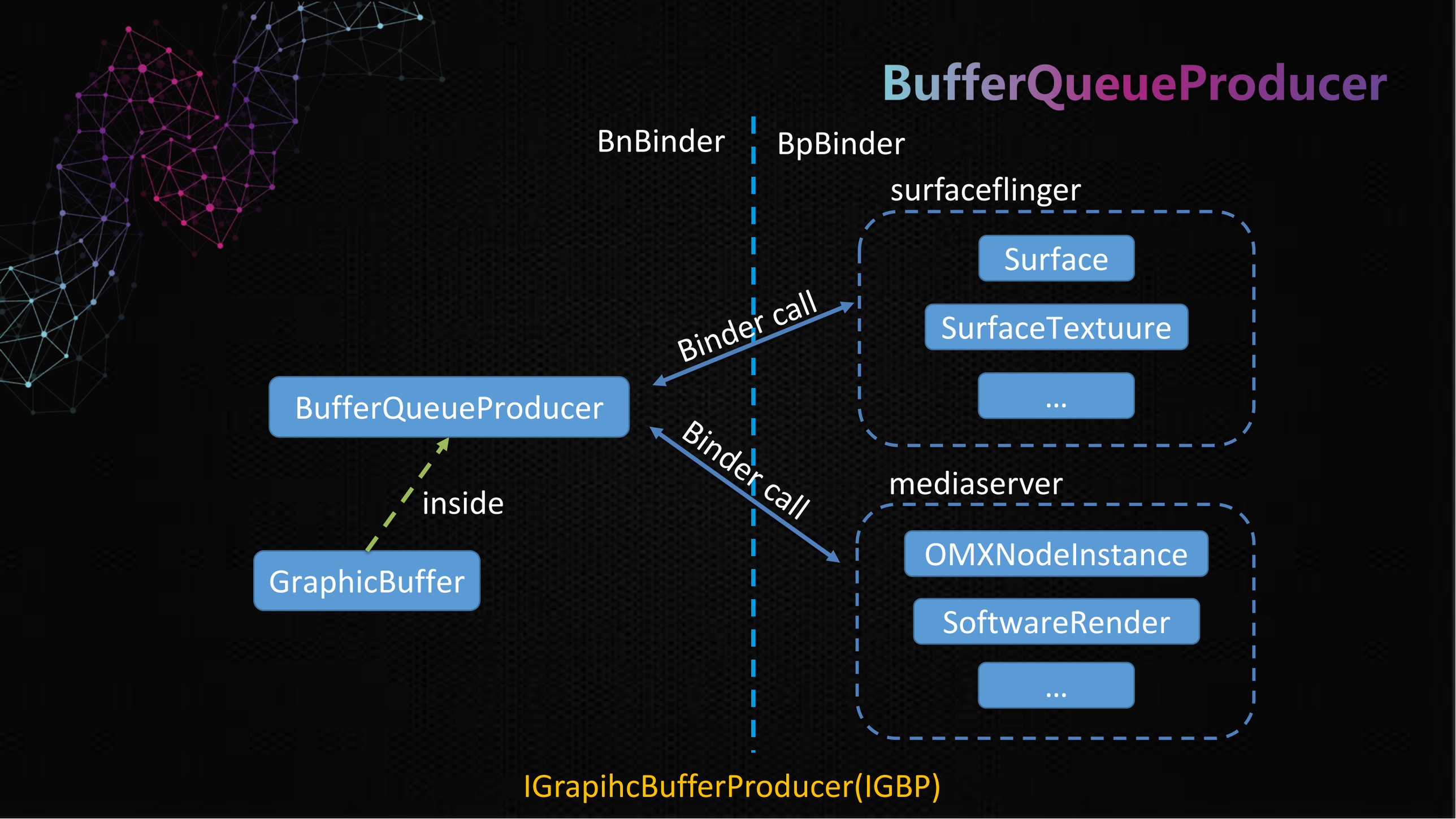
Local socket to remote socket

- Here simply describes the attack path
- We found a number of high-risk vulnerabilities on this basis, but the specific case is not disclosed because of information sensitivity.

BufferQueue in Android



BufferQueueProducer

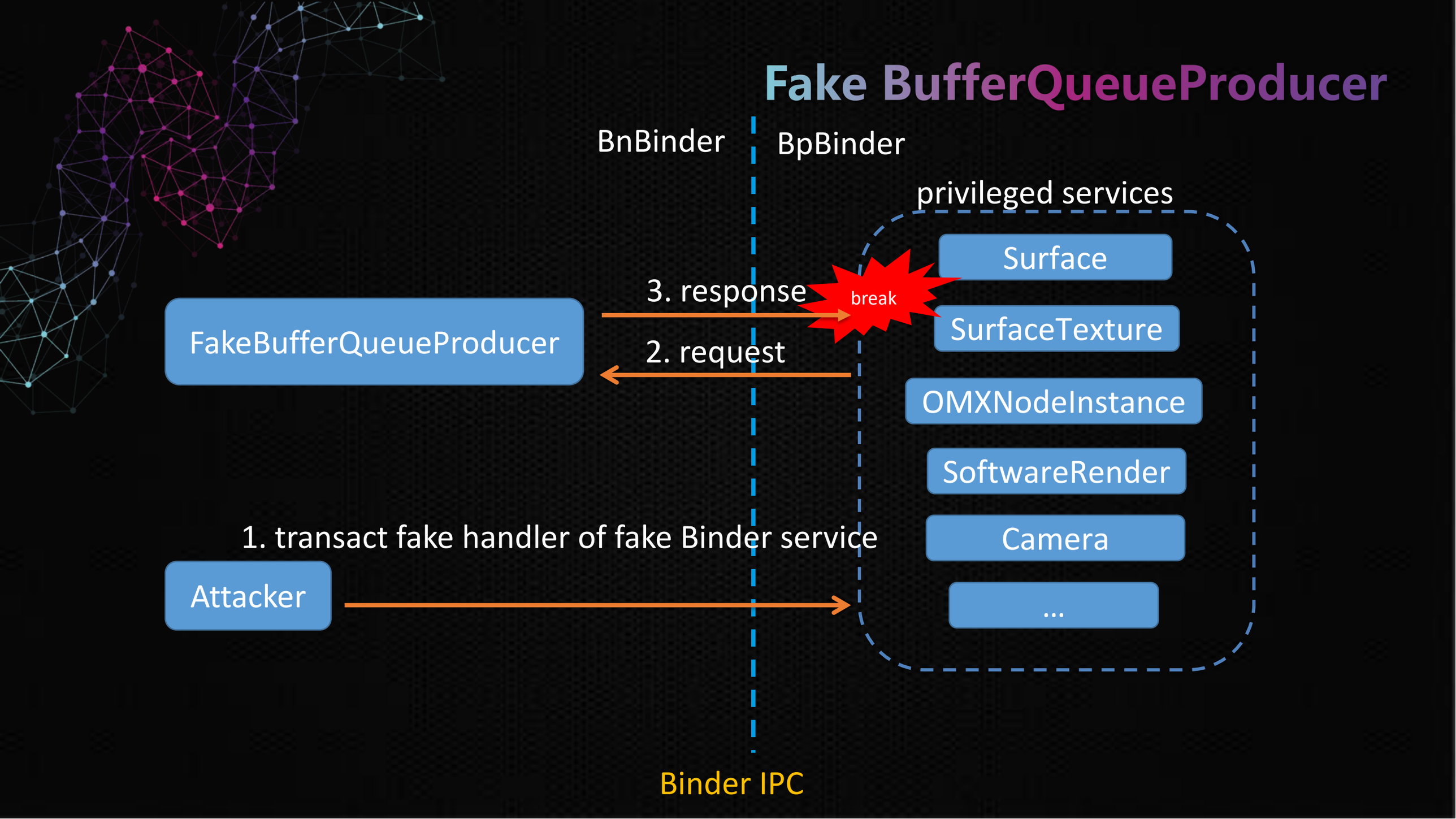


IPC interfaces in IGBP

```
class IGraphicBufferProducer : public IInterface
{
public:
DECLARE_META_INTERFACE(GraphicBufferProducer);
...

virtual status_t requestBuffer(int slot, sp<GraphicBuffer>* buf) = 0;

virtual status_t dequeueBuffer(int* slot, sp<Fence>* fence, uint32_t w,
uint32_t h, PixelFormat format, uint32_t usage) = 0;

virtual status_t attachBuffer(int* outSlot, const sp<GraphicBuffer>& buffer) = 0;

...
};
```

Nice pointer ☺

Fake BufferQueueProducer



Fake BufferQueueProducer

```
void createBufferQueue(sp<IGraphicBufferProducer>* outProducer,
sp<IGraphicBufferConsumer>* outConsumer) {
sp<BufferQueueCore> core(new BufferQueueCore(NULL));
sp<IGraphicBufferProducer> producer(new EvilBufferQueueProducer(core));
sp<IGraphicBufferConsumer> consumer(new BufferQueueConsumer(core));
*outProducer = producer;
*outConsumer = consumer;
}
```

```
class EvilBufferQueueProducer : public BnGraphicBufferProducer,
private IBinder::DeathRecipient
//or EvilBufferQueueConsumer
{
public:
EvilBufferQueueProducer(const sp<BufferQueueCore>& core);
virtual ~EvilBufferQueueProducer();
virtual status_t dequeueBuffer(int *outSlot, sp<Fence>* outFence,
uint32_t width, uint32_t height, PixelFormat format,
uint32_t usage);
...
};
```

Arbitrary read in Surface

```
virtual status_t dequeueBuffer(int *buf, sp<Fence>* fence, uint32_t width,
uint32_t height, PixelFormat format, uint32_t usage) {
Parcel data, reply;
...
status_t result = remote()->transact(DEQUEUE_BUFFER, data, &reply);
if (result != NO_ERROR) {
return result;
}
*buf = reply.readInt32();
...
return result;
}
```

```
int Surface::dequeueBuffer(android_native_buffer_t** buffer, int*
fenceFd) {
...
status_t result = mGraphicBufferProducer->dequeueBuffer(&buf, &fence,
reqWidth, reqHeight, reqFormat, reqUsage);
...
sp<GraphicBuffer>& gbuf(mSlots[buf].buffer);
...
*buffer = gbuf.get();
...
}
```

CVE-2017-0665

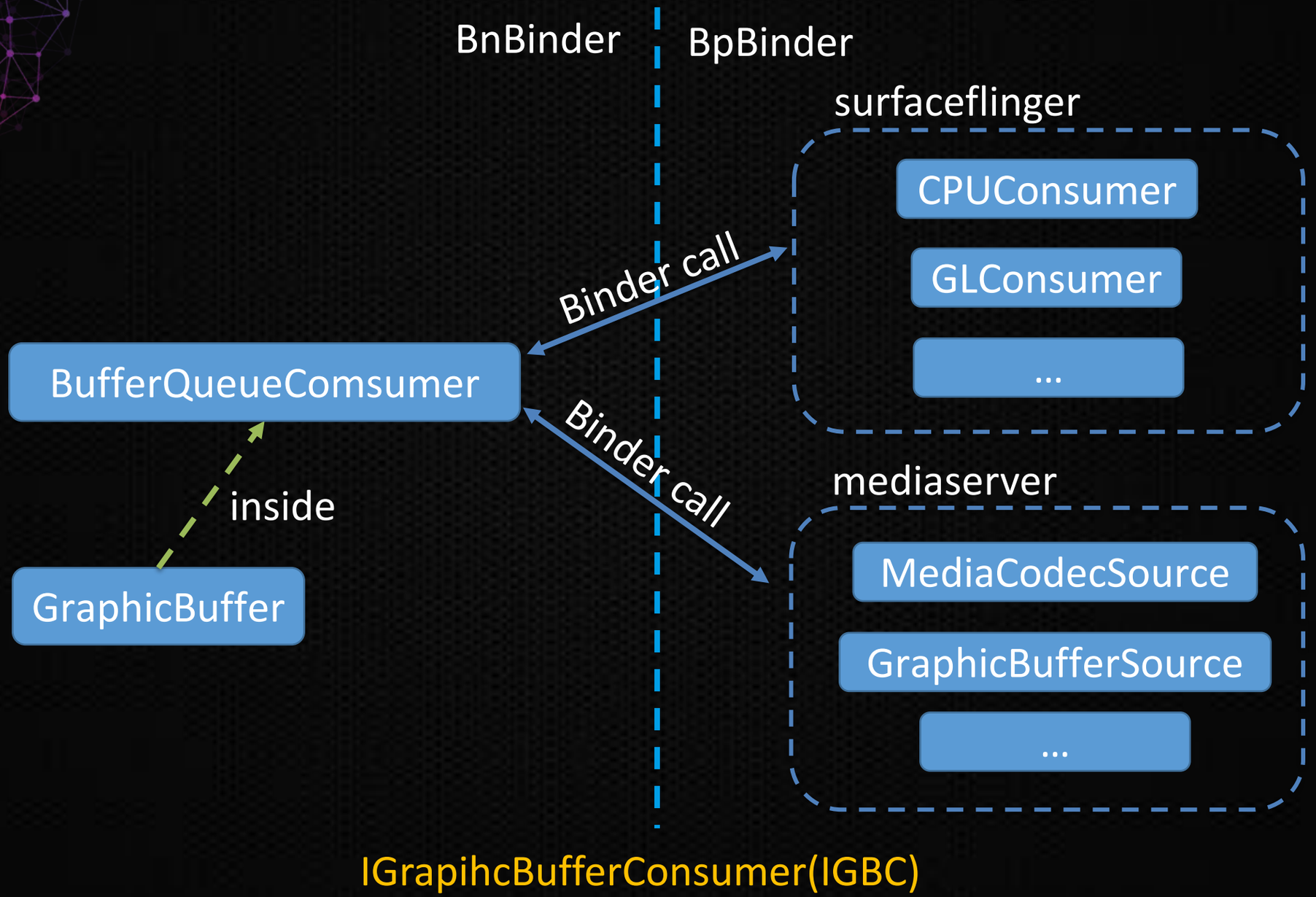
Arbitrary write in Surface

```
virtual status_t attachBuffer(int* slot, const sp<GraphicBuffer>& buffer)
{
...
status_t result = remote()->transact(ATTACH_BUFFER, data, &reply);
if (result != NO_ERROR) {
return result;
}
*slot = reply.readInt32();
result = reply.readInt32();
...
}
```

```
int Surface::attachBuffer(ANativeWindowBuffer* buffer)
{
...
sp<GraphicBuffer> graphicBuffer(static_cast<GraphicBuffer*>(buffer));
...
status_t result = mGraphicBufferProducer->attachBuffer(&attachedSlot, graphicBuffer);
...
mSlots[attachedSlot].buffer = graphicBuffer;
...
}
```

CVE-2017-0667(DUP)

BufferQueueConsumer



IPC interfaces in IGBC

```
class IGraphicBufferConsumer : public IInterface {

public:
DECLARE_META_INTERFACE(GraphicBufferConsumer);
...

virtual status_t acquireBuffer(BufferItem* buffer, nsecs_t presentWhen, uint64_t
maxFrameNumber = 0) = 0;

virtual status_t attachBuffer(int *outSlot, const sp<GraphicBuffer>& buffer) = 0;

virtual status_t consumerConnect(const sp<IConsumerListener>& consumer, bool
controlledByApp) = 0;

...
};
```

Again ☺

Arbitrary write in GraphicBufferSource

```
virtual status_t acquireBuffer(BufferItem *buffer, nsecs_t presentWhen,
uint64_t maxFrameNumber) {
Parcel data, reply;
...
status_t result = remote()->transact(ACQUIRE_BUFFER, data, &reply);
if (result != NO_ERROR) {
return result;
}
result = reply.read(*buffer);
...
}
```

```
void GraphicBufferSource::onFrameAvailable(const BufferItem& /*item*/) {
...
status_t err = mConsumer->acquireBuffer(&item, 0);
...
mBufferSlot[item.mSlot] = item.mGraphicBuffer;
mBufferUseCount[item.mSlot] = 0;
...
}
```

CVE-2017-0737

UAF in BufferQueueConsumerListener

```
status_t OMXNodeInstance::createGraphicBufferSource(//Binder call entry
OMX_U32 portIndex, sp<IGraphicBufferConsumer> bufferConsumer, MetadataBufferType
*type) {
...
sp<GraphicBufferSource> bufferSource = new GraphicBufferSource(this,
def.format.video.nFrameWidth,
def.format.video.nFrameHeight,
def.nBufferCountActual,
usageBits,
bufferConsumer);
...
}
```

```
GraphicBufferSource::GraphicBufferSource(
...
const sp<IGraphicBufferConsumer> &consumer):
...
mConsumer(consumer),
...
```

```
GraphicBufferSource::PersistentProxyListener::PersistentProxyListener(
const wp<IGraphicBufferConsumer> &consumer,
const wp<ConsumerListener>& consumerListener) :
mConsumerListener(consumerListener),
mConsumer(consumer) {}
```

```
wp<BufferQueue::ConsumerListener> listener =
static_cast<BufferQueue::ConsumerListener*>(this);
sp<IConsumerListener> proxy;
if (!mIsPersistent) {
proxy = new BufferQueue::ProxyConsumerListener(listener);
} else {
proxy = new PersistentProxyListener(mConsumer, listener);
}
mInitCheck = mConsumer->consumerConnect(proxy, false); //we can control mConsumer.
```

UAF in BufferQueueConsumerListener

```
virtual status_t consumerConnect(const sp<IConsumerListener>& consumer, bool
controlledByApp) {
Parcel data, reply;
data.writeInterfaceToken(IGraphicBufferConsumer::getInterfaceDescriptor());
data.writeStrongBinder(IInterface::asBinder(consumer));
data.writeInt32(controlledByApp);
status_t result = remote()->transact(CONSUMER_CONNECT, data, &reply);
if (result != NO_ERROR) {
return result;
}
return reply.readInt32();
}
```

```
void GraphicBufferSource::PersistentProxyListener::onFrameAvailable(
const BufferItem& item) {
sp<ConsumerListener> listener(mConsumerListener.promote());
...
}
```

```
template<typename T>
sp<T>::~sp() {
if (m_ptr)
m_ptr->decStrong(this);
}
```

```
void RefBase::decStrong(const void* id) const
{
weakref_impl* const refs = mRefs;
refs->removeStrongRef(id);
const int32_t c = refs->mStrong.fetch_sub(1, std::memory_order_release);
...
if (c == 1) {
...
delete this;
...
}
```

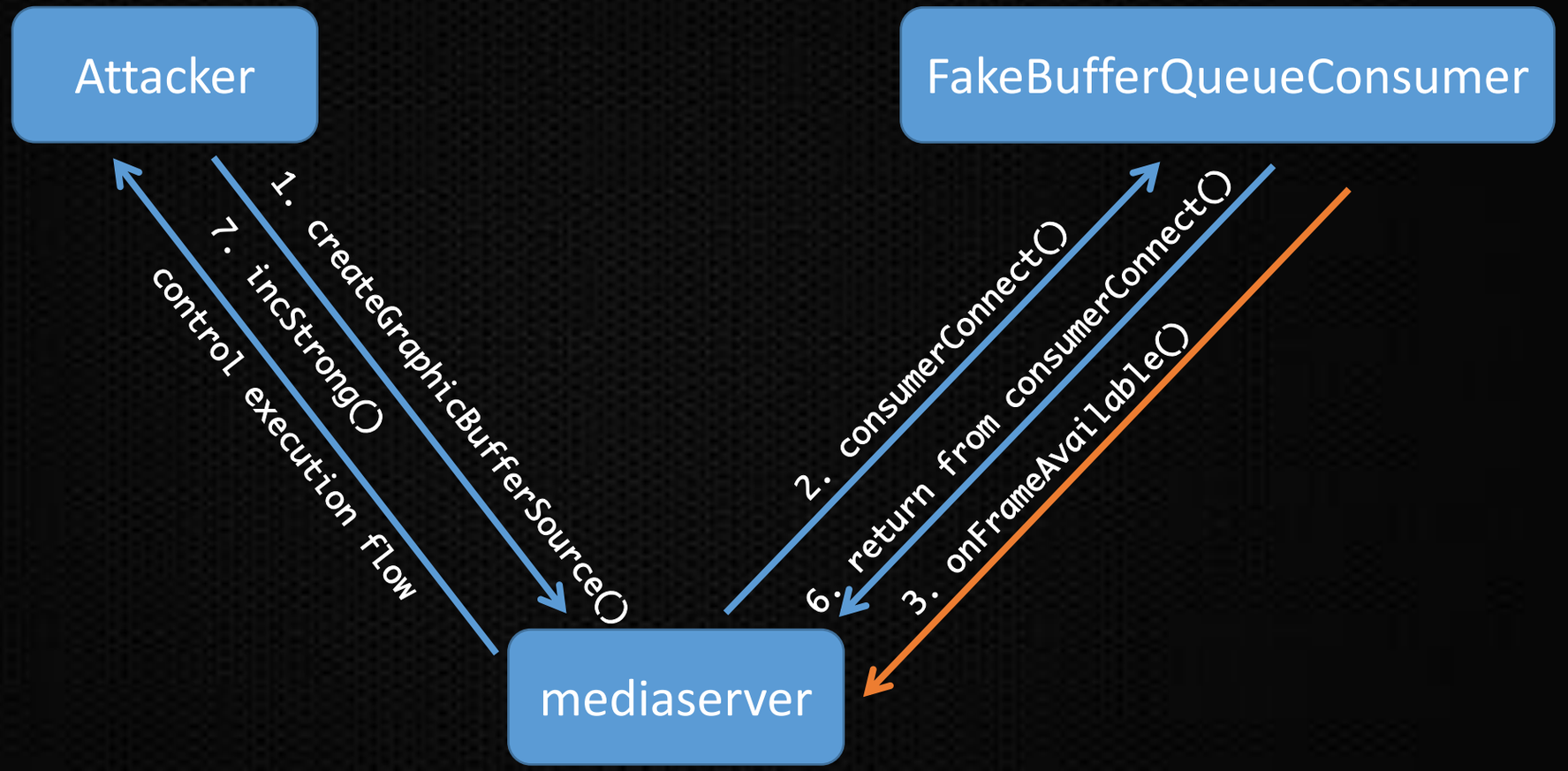
UAF in BufferQueueConsumerListener

```
status_t OMXNodeInstance::createGraphicBufferSource(//Binder call entry
OMX_U32 portIndex, sp<IGraphicBufferConsumer> bufferConsumer, MetadataBufferType
*type) {
...
sp<GraphicBufferSource> bufferSource = new GraphicBufferSource(this,
def.format.video.nFrameWidth,
def.format.video.nFrameHeight,
def.nBufferCountActual,
usageBits,
bufferConsumer);
...
}
```

```
class RefBase
{
...
virtual void onFirstRef();
virtual void onLastStrongRef(const void* id);
...
weakref_impl* const mRefs;
}

void RefBase::incStrong(const void* id) const
{
weakref_impl* const refs = mRefs;
...
refs->mBase->onFirstRef();// we control mBase
}
```

```
class RefBase::weakref_impl
        :public RefBase::weakref_type
{
public:
std::atomic<int32_t> mStrong;
std::atomic<int32_t> mWeak;
RefBase* const mBase;
std::atomic<int32_t> mFlags;
...
}

template<typename T>
sp<T>& sp<T>::operator =(T* other)
{
if (other)
other->incStrong(this);
if (m_ptr)
m_ptr->decStrong(this);
m_ptr = other;
return *this;
}
```

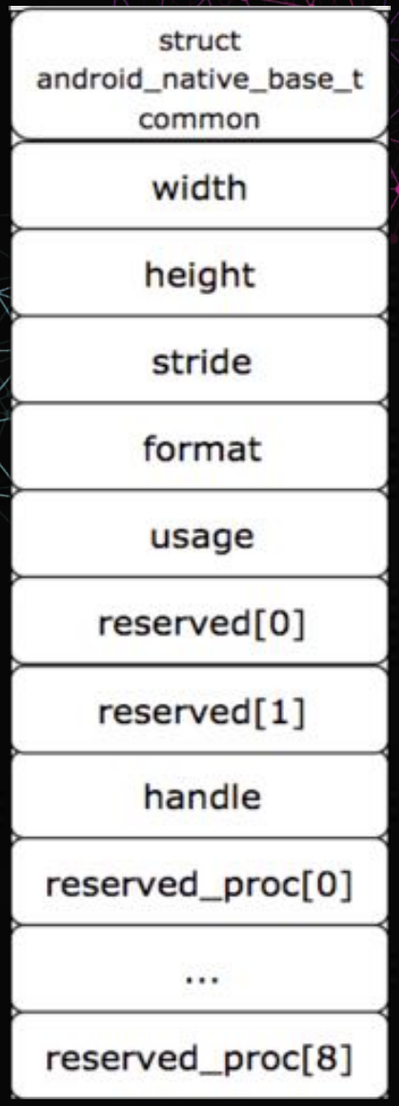
An time-free UAF



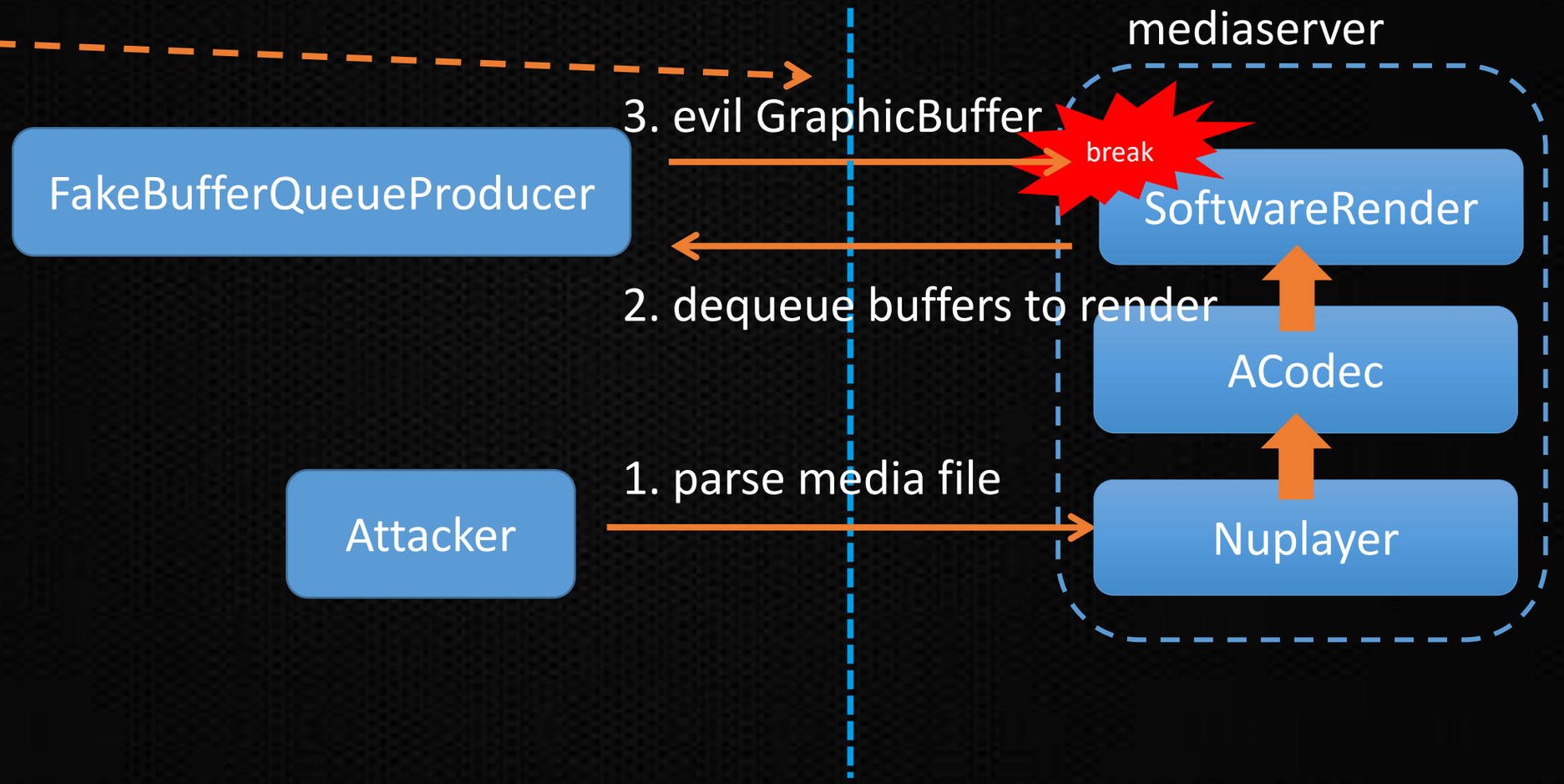
4. delete the GraphicBufferSource
5. sharp the memory by heap spray

CVE-2017-0732(DUP)

Dig into GraphicBuffer



Struct GraphicBuffer



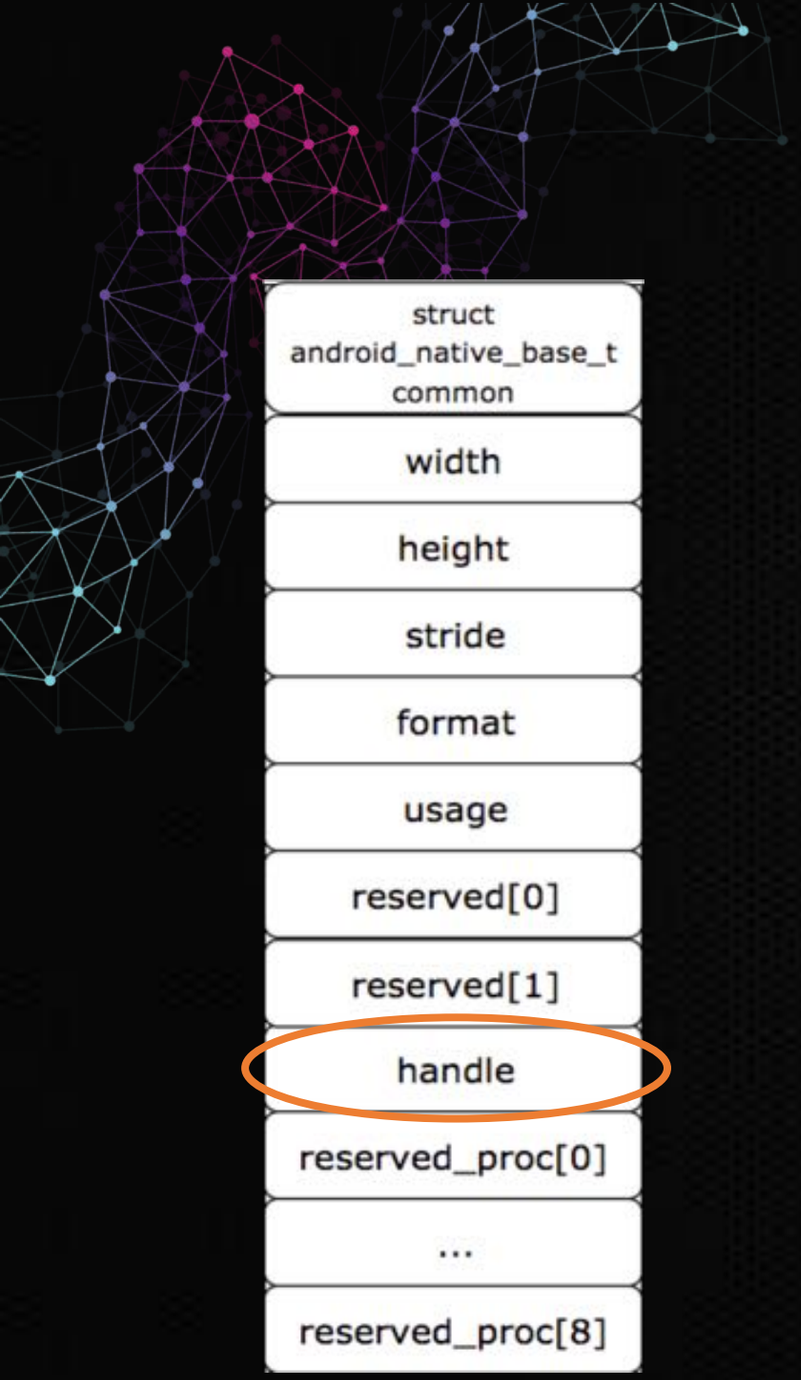
OOB Writes in GraphicBuffer

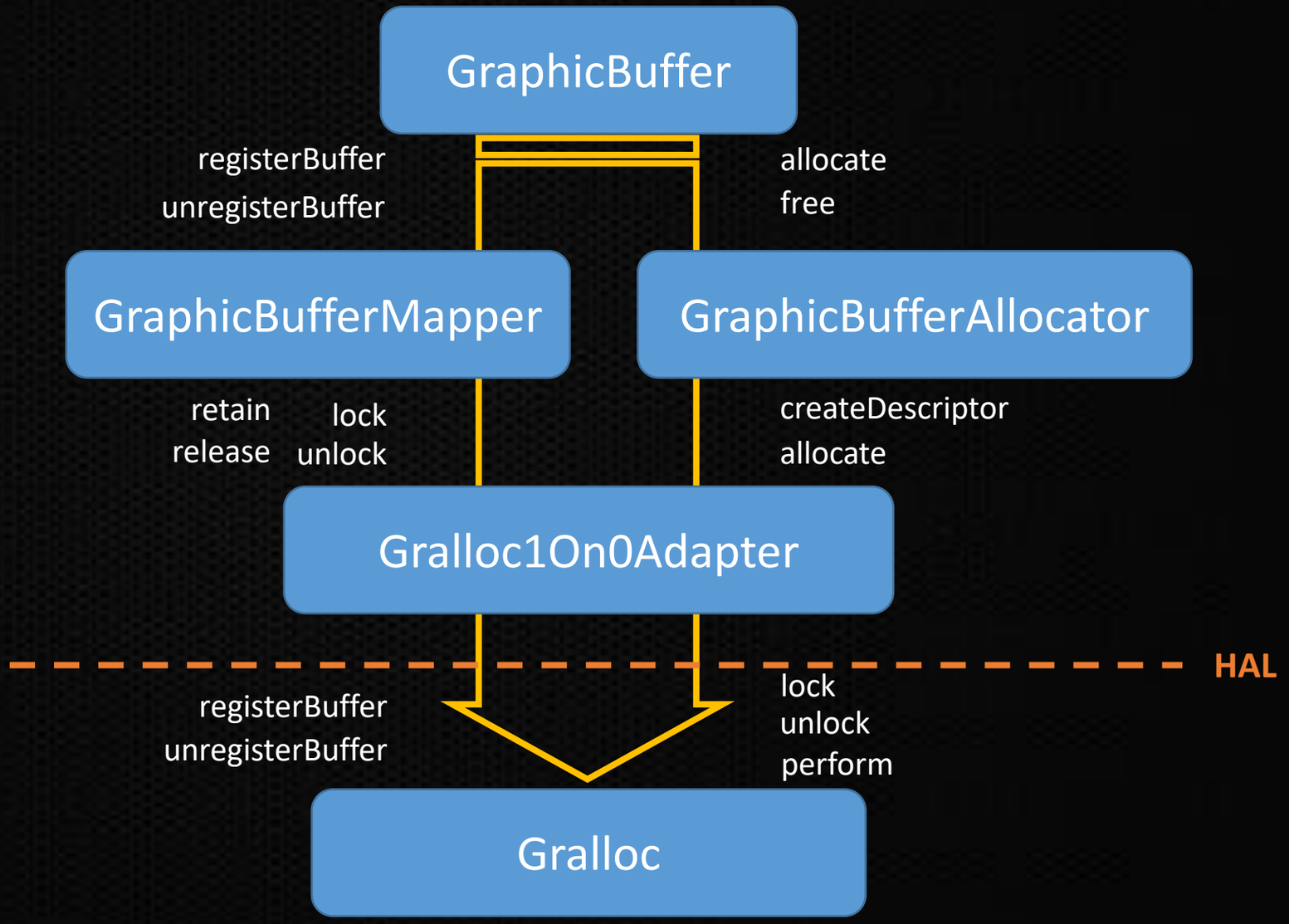
```
std::list<FrameRenderTracker::Info> SoftwareRenderer::render(
const void *data, size_t size, int64_t mediaTimeUs, nsecs_t renderTimeNs,
void* /*platformPrivate*/, const sp<AMessage>& format) {
...
ANativeWindowBuffer *buf;
int fenceFd = -1;
int err = mNativeWindow->dequeueBuffer(mNativeWindow.get(), &buf, &fenceFd);
...
void *dst;
CHECK_EQ(0, mapper.lock(
buf->handle, GRALLOC_USAGE_SW_WRITE_OFTEN, bounds, &dst));
...
const uint8_t *src_y = (const uint8_t *)data;
const uint8_t *src_u = (const uint8_t *)data + mWidth * mHeight;
const uint8_t *src_v = src_u + (mWidth / 2 * mHeight / 2);
uint8_t *dst_y = (uint8_t *)dst;
size_t dst_y_size = buf->stride * buf->height;
size_t dst_c_stride = ALIGN(buf->stride / 2, 16);
size_t dst_c_size = dst_c_stride * buf->height / 2;
uint8_t *dst_v = dst_y + dst_y_size;
uint8_t *dst_u = dst_v + dst_c_size;
...
for (int y = 0; y < (mCropHeight + 1) / 2; ++y) {
memcpy(dst_u, src_u, (mCropWidth + 1) / 2);
memcpy(dst_v, src_v, (mCropWidth + 1) / 2);
...
}
```

AndroidID-62084097

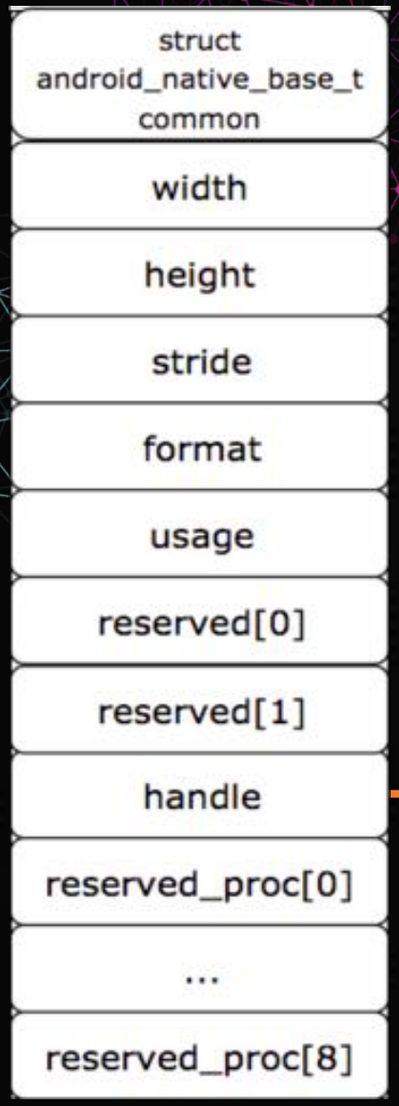
AndroidID-62540707

From GraphicBuffer to NativeHandle

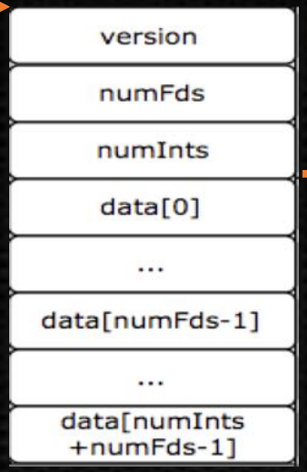




NativeHandle under Gralloc



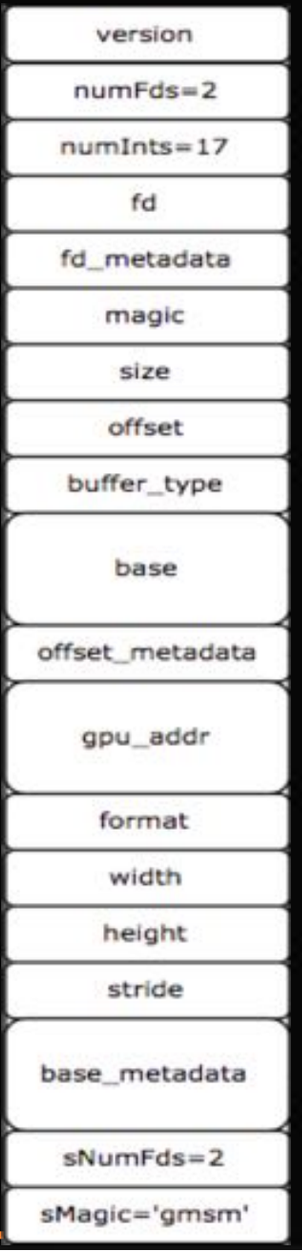
Struct GraphicBuffer



int

Struct NativeHandle

We can fake all fields !



Struct PrivateNativeHandle
(Gralloc of Qcom's Implement)

PrivateNativeHandle in Qcom Gralloc Impl.

```
int gralloc_lock(gralloc_module_t const* module,
buffer_handle_t handle, int usage,
int /*l*/, int /*t*/, int /*w*/, int /*h*/,
void** vaddr)
{
...
private_handle_t* hnd = (private_handle_t*)handle;
int err = gralloc_map_and_invalidate(module, handle,
usage);
...
}
```

```
typedef const native_handle_t* buffer_handle_t;
```

```
static int gralloc_map_and_invalidate (gralloc_module_t
const* module,
buffer_handle_t handle, int usage)
{
...
if (!module || private_handle_t::validate(handle) < 0)
return -EINVAL;
...
err = gralloc_map(module, handle);
...
}
```

```
static int gralloc_map(gralloc_module_t const* module,
buffer_handle_t handle)
{
...
private_handle_t* hnd = (private_handle_t*)handle;
...
size = hnd->size;
err = memalloc->map_buffer(&mappedAddress, size,
hnd->offset, hnd->fd);
...
hnd->base = uint64_t(mappedAddress) + hnd->offset;
...
}
```

```
static int validate(const native_handle* h) {
const private_handle_t* hnd = (const private_handle_t*)h;
if (!h || h->version != sizeof(native_handle) ||
h->numInts != sNumInts() || h->numFds != sNumFds ||
hnd->magic != sMagic)
{
...
return -EINVAL;
}
return 0;
}
```

fragile check

PrivateNativeHandle in Qcom Gralloc Impl.

```
int IonAlloc::map_buffer(void **pBase, unsigned int size,
unsigned int offset, int fd)
{
...
void *base = 0;
...
base = mmap(0, size, PROT_READ| PROT_WRITE,
MAP_SHARED, fd, 0);
*pBase = base;
...
}
```

```
int IonAlloc::unmap_buffer(void *base, unsigned int size,
unsigned int /*offset*/)
{
...
if(munmap(base, size)) {
...
}
```

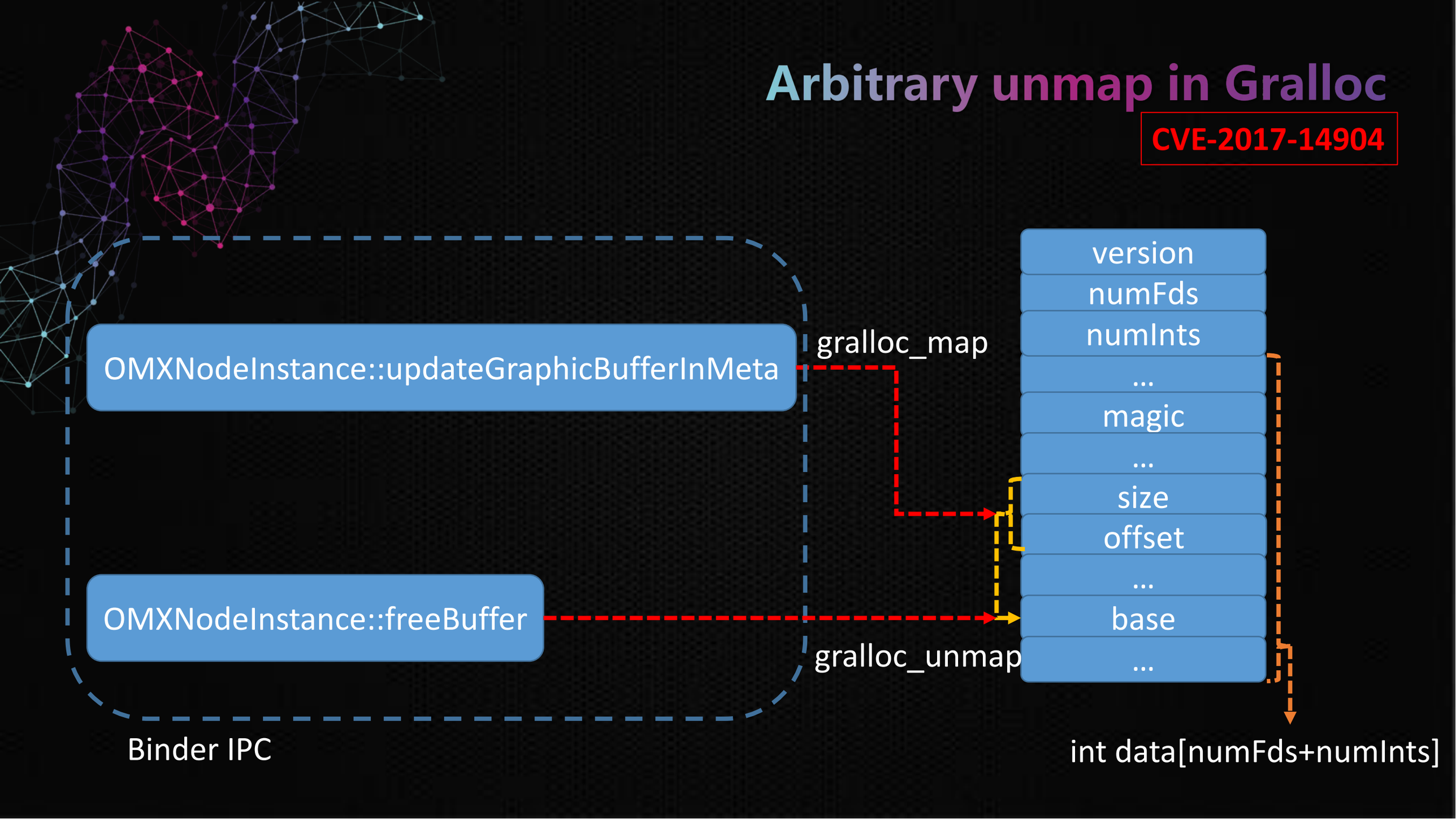
```
int gralloc_unregister_buffer(gralloc_module_t const*
module,
buffer_handle_t handle)
{
ATRACE_CALL();
if (!module || private_handle_t::validate(handle) < 0)
return -EINVAL;
...
return gralloc_unmap(module, handle);
}
```

```
typedef const native_handle_t* buffer_handle_t;

static int gralloc_unmap(gralloc_module_t const* module,
buffer_handle_t handle)
{
...
private_handle_t* hnd = (private_handle_t*)handle;
...
if(hnd->base) {
err = memalloc->unmap_buffer((void*)hnd->base, hnd->size,
hnd->offset);
...
}
```

direct convert

Arbitrary unmap in Gralloc

CVE-2017-14904



Defects in Gralloc

Party @ Gralloc

	Default	Qcom	Nivida
OOB/Arbitrary Write	AndroidID-63107384 AndroidID-63662157	AndroidID-62997699 CVE-2017-14904	AndroidID-62539574 AndroidID-62540032
Infoleak	AndroidID-63104333	AndroidID-63107128 AndroidID-63527797	N/A

Summary

- Gralloc defects impact default implement in AOSP, and also in vendor-specific implement
- Mismatch between NativeHandle and it’s buffer based derived class also exists in other HAL components implement
- We should take more concern when transmit data cross layer/border

Acknowledge & Reference

- Acknowledge
  - Joint work with C0RE Team(@C0RETeam)
  - All of the team members in Ant-financial Light-Year Security Lab

- Reference
  - https://source.android.com/devices/graphics/arch-bq-gralloc
  - http://www.commverge.com/Solutions/SubscribersServicesManagement/OverTheAirOTASolutions/tabid/176/Default.aspx

Q&A

THANK YOU

蚂蚁金服 ANT FINANCIAL | 支付宝 ALIPAY