Binder

Étude du mécanisme de communication interprocessus d'Android et de ses vulnérabilités

Binder IPC and its vulnerabilities

Présenté 06/03/2020
Pour THCON 2020
Par Jean-Baptiste Cayrou
Who I am

- Jean-Baptiste Cayrou (@jbcayrou)

- **Synacktiv:**
  - Offensive security company
  - > 60 ninjas
  - 3 teams: pentest, reverse engineering, development

- **Reverser at Synacktiv:**
  - Focus on low level reverse, vulnerability research, source code audit
  - Work since several years on Android

- **Binder articles on Synacktiv blog**
Introduction

- Binder: Kernel Module for communications between Android processes in Android

- Hot topic
  - Exploitation in the wild discovered by Google
  - Recent critical vulnerabilities

- A lot of documentation for high level parts but missing for low level behavior :(  
  - => Start to study Binder internals
Summary

- Part I: Binder presentation
- Part II: Binder vulnerabilities
- Part III: Study of two binder patches
PART I - Presentation of Binder
History

- Android was bought by Google in 2008
- Android is based on the Linux kernel with specific drivers
  - Binder
  - Ashmem
  - Low Memory Killer
- Binder is based on OpenBinder implementation
  - Developed by Be Inc and Palm.
  - Lead by Dianne Hackborn now working at Google
Binder Features

- **Kernel Module for IPC/RPC**
  - ~ 6000 lines of code in `linux/drivers/android/binder_...`

- **Features**:
  - Send messages between applications (sync/async)
  - Call remote function (RPC)
  - Share file descriptors (file, ashmem)
  - Manage references (strong, weak) on remote and local objects

- **Binder messages are called ‘Transactions’**
Binder transaction payload

- Up to 1 MB
- Basic types
  - Integer, long, strings, simple data (sequence of bytes)
- Binder Objects
  - Data relative to a process
  - Need a transformation by the Kernel for the receiver (filedescriptor, local memory, references)
Binder Objects

- Local Object
  - BINDER_TYPE_BINDER
  - BINDER_TYPE_WEAK_BINDER
- Remote object
  - BINDER_TYPE_HANDLE
  - BINDER_TYPE_WEAK_HANDLE
- File Descriptors
  - BINDER_TYPE_FD
  - BINDER_TYPE_FDA
- Buffer
  - BINDER_TYPE_PTR
Android Framework Interactions

- **Activities**
  - Part of an application (user interface screen)
  - Optionally have arguments
  - Example: Open the browser at this address

- **Content Provider**
  - Database like, accessible by other applications (query, insert, update, remove)
  - Uri: ‘content://<authority>/<path>/<id>’
  - Example: contacts
Android Framework Interactions

- **Broadcast**:  
  - publish-subscribe design pattern  
  - Broadcast events to applications (Incoming call, network connection changed ...)

- **Service**  
  - A Background application which exposes commands to others (RPC)  
  - Main IPC/RPC component, **based on Binder**!  
  - Example: ActivityManager, ContentService

- **Activities, Content Providers and Broadcasts** are based on Services
Android Service Interaction

1. register

Service A

M1
M2
M3
M4

ServiceA.M1(..)

App B

2. get("Service A")

ServiceManager

3.
Android Service Interaction

- How applications know services interfaces?
- Using Interface Definition Languages:
  - AIDL: For Framework Applications
  - HIDL: For Hardware Service (for vendors)
- AIDL and HIDL describe RPC functions
- Compilers for these languages generate code (C++ and Java):
  - Binder Proxy for client part
  - Binder Stub for service implementation
Binder Call WorkFlow
AIDL - Parcel

- Serialization library for Binder transactions
  - JAVA : android.os.Parcel
  - C/C++ : frameworks/native/include/binder/Parcel.h
- Basic types
  - writeInt/ readInt
  - writeString/readString
  - WriteInArray / readIntArray
- Filedescriptor and references:
  - WriteFileDescriptor / readFileDescriptor
  - ...

AIDL - File Example

// IRemoteService.aidl
package com.example.android;

// Declare any non-default types here with import statements

/** Example service interface */
interface IRemoteService {
    /** Request the process ID of this service, to do evil things with it. */
    int getPid();

    /** Demonstrates some basic types that you can use as parameters
     * and return values in AIDL.
     */
    void basicTypes(int anInt, long aLong, boolean aBoolean, float aFloat,
                    double aDouble, String aString);
}
::android::binder::Status BpRemoteService::basicTypes(int32_t anInt, int64_t aLong, bool aBoolean, float aFloat,
::android::Parcel _aidl_data;
::android::Parcel _aidl_reply;
::android::status_t _aidl_ret_status = ::android::OK;
::android::binder::Status _aidl_status;
_aidl_ret_status = _aidl_data.writeInterfaceToken(getInterfaceDescriptor());
if ((_aidl_ret_status) != (::android::OK)) {
    goto _aidl_error;
}
_aidl_ret_status = _aidl_data.writeInt32(anInt);
if ((_aidl_ret_status) != (::android::OK)) {
    goto _aidl_error;
}
_aidl_ret_status = _aidl_data.writeInt64(aLong);
if ((_aidl_ret_status) != (::android::OK)) {
    goto _aidl_error;
}
_aidl_ret_status = _aidl_data.writeBool(aBoolean);
if ((_aidl_ret_status) != (::android::OK)) {
    goto _aidl_error;
}
_aidl_ret_status = _aidl_data.writeFloat(aFloat);
if ((_aidl_ret_status) != (::android::OK)) {
    goto _aidl_error;
}
_aidl_ret_status = _aidl_data.writeDouble(aDouble);
if ((_aidl_ret_status) != (::android::OK)) {
    goto _aidl_error;
}
_aidl_ret_status = _aidl_data.writeString16(aString);
if ((_aidl_ret_status) != (::android::OK)) {
    goto _aidl_error;
}
_aidl_ret_status = remote()->transact(IRemoteService::BASIC_TYPES, _aidl_data, &_aidl_reply);
if ((_aidl_ret_status) != (::android::OK)) {
    goto _aidl_error;
}
_aidl_ret_status = _aidl_status.readFromParcel(_aidl_reply);
if ((_aidl_ret_status) != (::android::OK)) {
    goto _aidl_error;
}
HIDL – Parcel (HwParcel)

- Serialization library for HwBinder transactions (C++ and Java)
  - system/libhwbinder/include/hwbinder/Parcel.h
  - android/os/HwParcel.java
- Based on the Parcel Framework
- Support of data buffer binder object
  - For instance, C structures containing pointers to others buffers
- More complex types!
interface IFoo {
    uint32_t[3][4][5][6] multidimArray;

    vec<vec<vec<int8_t>>>() multidimVector;

    vec<bool[4]> arrayVec;

    struct foo {
        struct bar {
            uint32_t val;
        };
        bar b;
    }

    struct baz {
        foo f;
        foo.bar fb; // HIDL uses dots to access nested type names
    }

    ...
}
Transaction buffers
Binder device

- Device: /dev/binder, /dev/hwbinder, /dev/vndbinder
- Mapped as read-only in process memory to receive binder messages
- loctl commands:
  - BINDER_WRITE_READ => Used for IPC
  - BINDER_SET_MAX_THREADS
  - BINDER_SET_CONTEXT_MGR
  - BINDER_THREAD_EXIT
  - BINDER_VERSION
Binder commands

- BC_TRANSACTION
- BC_TRANSACTION_SG (SG : Scatter Gather)
- BC_REPLY
- BC_FREE_BUFFER
- ...

Tips:
- ‘BC_’ : Binder Command
- ‘BR_’ : Binder Return
BC TRANSACTION

- **Handle**: Remote service ID
- **Code**: Remote method id
- **Buffer**: Message data
- **Offsets**: Objects list

**BC TRANSACTION_SG**:
- + extra_size

```
<table>
<thead>
<tr>
<th>binder_transaction_data</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
</tr>
<tr>
<td>cookie</td>
</tr>
<tr>
<td>code</td>
</tr>
<tr>
<td>sender_pid</td>
</tr>
<tr>
<td>sender_euid</td>
</tr>
<tr>
<td>data size</td>
</tr>
<tr>
<td>offsets_size</td>
</tr>
<tr>
<td>buffer</td>
</tr>
<tr>
<td>offsets</td>
</tr>
</tbody>
</table>
```
Recap of userland view
Entering the Kernel!

- The kernel allocates the necessary size in the targeted process (size : data + offsets + extra) and copies the transaction

- Lookup the offsets list to patch all binder objects
  - Convert local and remote references
  - Install file descriptors in the target process
  - Copies BINDER_TYPE_PTR buffers in the target process (in extra part)
Example!

Send this hidl_string object:

```c
struct hidl_string {
    // copy from a C-style string. nullptr will create an empty string
    hidl_string(const char *);
    // ...

private:
    details::hidl_pointer<const char> mBuffer; // Pointer to the real char string
    uint32_t mSize; // NOT including the terminating '\0'.
    bool mOwnsBuffer; // if true then mBuffer is a mutable char *
};

hidl_string my_obj("My demo string");
```

When ‘my_obj’ is created, a heap allocation is performed by the constructor to store the real string address in `mBuffer`
HIDL Parcel

offsets

<table>
<thead>
<tr>
<th>off 1</th>
<th>off 2</th>
<th>...</th>
</tr>
</thead>
</table>

data buffer

BINDER_TYPE_PTR A
buffer = aaaaaa
length = yyyy

BINDER_TYPE_PTR B
buffer = bbbbbbb
length = yyyy

...
HIDL Parcel

```
BINDER_TYPE_PTR_A
  flags
  buffer
  length = sizeof(hidl_string)
  parent = NULL
  parent_offset = 0

BINDER_TYPE_PTR_B
  flags
  buffer
  length = 0x0f
  parent = A
  parent_offset = 0

hidl_string
  mBuffer
  mSize = 0x0f
  mOwnsBuffer

char *
"My demo string"
```
PROC 1

SEND STEP

Buf 1
parent = Buf 1
offset: off(ptr)

Buf 2
BUFFERS

Struct hidl_string
char * ptr = 0xaabbccddd
uint32_t mSize = 0x0f
bool mOwnBuffer = 0

0xaaddccddd = "My demo string"

PROC 2

RECEIVE STEP

Struct hidl_string
char * ptr = 0x11223344
uint32_t mSize = 0x0f
bool mOwnBuffer = 0

0x11223344 = "My demo string"
PART II - Binder vulnerabilities
Critical component

- **Binder is the base of Android**
  - All applications use binder (even `untrusted_app` or `isolated_app`)
  - Generic code on all devices

- **Binder vulnerabilities => Generic exploits !**
Attack Surface

Where can we find bugs?

- In the Kernel: Binder driver
- In the serialization libraries
  - Libbinder: Parcel
  - Libhwbinder: HwParcel
## Explore Android Security Bulletins

<table>
<thead>
<tr>
<th>Advisory Data</th>
<th>Patch date</th>
<th>Patch to Advisory</th>
<th>Component</th>
<th>CVE</th>
<th>Type</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/03/2020</td>
<td>15/12/2019</td>
<td>~ 3 months</td>
<td>Binder Driver</td>
<td>CVE-2020-0041</td>
<td>Eop</td>
<td>High</td>
</tr>
<tr>
<td>01/02/2020</td>
<td>16/02/2018</td>
<td>~12 months</td>
<td>Binder Driver</td>
<td>CVE-2020-0030</td>
<td>Eop</td>
<td>High</td>
</tr>
<tr>
<td>01/02/2020</td>
<td>15/10/2019</td>
<td>~ 4 months</td>
<td>libbinder</td>
<td>CVE-2020-0026</td>
<td>EoP</td>
<td>High</td>
</tr>
<tr>
<td>01/11/2019</td>
<td>12/06/2019</td>
<td>~ 7 months</td>
<td>Binder Driver</td>
<td>CVE-2019-2213</td>
<td>EoP</td>
<td>High</td>
</tr>
<tr>
<td>01/11/2019</td>
<td>09/06/2019</td>
<td>~ 5 months</td>
<td>Binder Driver</td>
<td>CVE-2019-2214</td>
<td>EoP</td>
<td>High</td>
</tr>
<tr>
<td>01/10/2019</td>
<td>05/01/2018</td>
<td>~ 9 months</td>
<td>Binder Driver</td>
<td>CVE-2019-2215</td>
<td>EoP</td>
<td>High</td>
</tr>
<tr>
<td>01/09/2019</td>
<td>24/04/2019</td>
<td>~ 4 months</td>
<td>Binder Driver</td>
<td>CVE-2019-2181</td>
<td>EoP</td>
<td>High</td>
</tr>
<tr>
<td>01/08/2019</td>
<td>17/05/2019</td>
<td>~ 3 months</td>
<td>libbinder</td>
<td>CVE-2019-2136</td>
<td>ID</td>
<td>High</td>
</tr>
<tr>
<td>01/07/2019</td>
<td>18/04/2019</td>
<td>~ 3 months</td>
<td>libhwbinder</td>
<td>CVE-2019-2118</td>
<td>ID</td>
<td>High</td>
</tr>
<tr>
<td>01/03/2019</td>
<td>05/12/2018</td>
<td>~ 3 months</td>
<td>libhwbinder</td>
<td>CVE-2019-2011</td>
<td>EoP</td>
<td>High</td>
</tr>
<tr>
<td>01/03/2019</td>
<td>06/11/2018</td>
<td>~ 4 months</td>
<td>Binder Driver</td>
<td>CVE-2019-2025</td>
<td>EoP</td>
<td>High</td>
</tr>
<tr>
<td>01/02/2019</td>
<td>23/08/2018</td>
<td>~ 5 months</td>
<td>Binder Driver</td>
<td>CVE-2019-1999</td>
<td>EoP</td>
<td>High</td>
</tr>
<tr>
<td>01/02/2019</td>
<td>11/11/2017</td>
<td>~ 3 months</td>
<td>Binder Driver</td>
<td>CVE-2019-2000</td>
<td>EoP</td>
<td>High</td>
</tr>
<tr>
<td>01/08/2018</td>
<td>15/11/2017</td>
<td>~ 9 months</td>
<td>Binder Driver</td>
<td>CVE-2018-9465</td>
<td>EoP</td>
<td>High</td>
</tr>
<tr>
<td>01/04/2018</td>
<td>29/06/2017</td>
<td>~ 9 months</td>
<td>Binder Driver</td>
<td>CVE-2017-17770</td>
<td>EoP</td>
<td>High</td>
</tr>
<tr>
<td>01/12/2017</td>
<td>06/06/2017</td>
<td>~ 7 months</td>
<td>Binder Driver</td>
<td>CVE-2017-13162</td>
<td>EoP</td>
<td>High</td>
</tr>
<tr>
<td>01/01/2017</td>
<td>?</td>
<td></td>
<td>Binder Driver</td>
<td>CVE-2016-8468</td>
<td>EoP</td>
<td>Moderate</td>
</tr>
<tr>
<td>01/10/2016</td>
<td>?</td>
<td></td>
<td>Binder Driver</td>
<td>CVE-2016-6683</td>
<td>ID</td>
<td>Moderate</td>
</tr>
<tr>
<td>01/10/2016</td>
<td>?</td>
<td></td>
<td>libbinder</td>
<td>CVE-2016-6684</td>
<td>ID</td>
<td>Moderate</td>
</tr>
<tr>
<td>01/05/2016</td>
<td>?</td>
<td></td>
<td>libbinder</td>
<td>CVE-2016-2440</td>
<td>EoP</td>
<td>High</td>
</tr>
</tbody>
</table>
Explore Android Security Bulletins

- 20 CVE from 01/2014 to 03/2020:
  - 14 Binder Driver
  - 4 libbinder
  - 2 libhwbinder
- 80% CVE are HIGH (20% Moderate)
  - But notation changed in 2017
- Privilege escalation (EoP) or Information disclosure (ID)
- In average 5 months between the patch and the advisory
Obversations

- Security patches don’t always have a CVE
  - Difficult to backport patches in the linux kernel!
- Backports are not always done.
  - Even on google references branches (kernel/msm)
Example 1 : CVE-2019-2215
(bad binder)

- Exploits found in the wild by Google

- The bug
  - Discovered in November 2017
  - Patched in February 2018
  - Never included in the security bulletin !
  - => No security backport on several devices

- Pixel devices : 19 months since the patch !
Example 2 : CVE-2019-2025 (waterdrop)

- Discovered by C0RE Team, Qihoo 360
  http://blogs.360.cn/post/Binder_Kernel_Vul_EN.html
- Universal Android root ! ( versions > 11/2014)
- Kernel patch : 06/11/2018
- CVE publication : 01/03/2019
- Attackers : 4 months to make a generic root !
Weakness of bulletins

- Vulnerabilities in kernel are difficult to follow and patch
  - Vendors have their own kernel
- Vulnerabilities in AOSP (libbinder/libhwbinder) are less critical and easier to patch
- Public patches give an advantage to attackers!
PART III - Study of two binder patches
Patch

- [ ] https://github.com/torvalds/linux/
- [ ] Review Upstream kernel binder.c patches
- [ ] Can we find commits that fix recent vulnerabilities (and not patched yet)?
PART III - Study of two binder patches
   a) Binder secctx patch analysis
Binder secctx patch analysis

- **Commit ec74136ded (January 14 2019)**

  ```text
  binder: create node flag to request sender's security context
  
  To allow servers to verify client identity, allow a node flag to be set that causes the sender's security context to be delivered with the transaction. The BR_TRANSACTION command is extended in BR_TRANSACTION_SEC_CTX to contain a pointer to the security context string.
  
  Signed-off-by: Todd Kjos <tkjos@google.com>
  Reviewed-by: Joel Fernandes (Google) <joel@joelfernandes.org>
  Signed-off-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>
  ```

- **Add a security context (selinux) to a binder transaction**
**Origin**

- **Fix CVE-2019-2023 (EoP High)**
  - ACL (Access Control List) bypass due to an insecure permission check, based on the PID of the caller

- **Binder design issue : How to know the identity of the caller ?**
  - Currently using its PID `getpidcon()`
  - However if the caller is dead and the PID is reused the context will be incorrect … (see Jann Horn POC)

https://bugs.chromium.org/p/project-zero/issues/detail?id=851
Main part of the patch

@@ -3020,6 +3027,20 @@ static void binder_transaction(struct binder_proc *proc,
+   if (target_node && target_node->txn_security_ctx) {
+     u32 secid;
+
+     security_task_getsecid(proc->tsk, &secid);
+     ret = security_secid_to_secctx(secid, &secctx, &secctx_sz);
+     if (ret) {
+         return_error = BR_FAILED_REPLY;
+         return_error_param = ret;
+         return_error_line = __LINE__;
+         goto err_get_secctx_failed;
+     }
+     extra_buffers_size += ALIGN(secctx_sz, sizeof(u64));
+   }
+
+   if (secctx) {
+     size_t buf_offset = ALIGN(tr->data_size, sizeof(void *)) +
+                        ALIGN(tr->offsets_size, sizeof(void *)) +
+                        ALIGN(extra_buffers_size, sizeof(void *)) -
+                        ALIGN(secctx_sz, sizeof(u64));
+     char *kptr = t->buffer->data + buf_offset;
+     t->security_ctx = (uintptr_t)kptr +
+                        binder_alloc_get_user_buffer_offset(&target_proc->alloc);
+     memcpy(kptr, secctx, secctx_sz);
+     security_release_secctx(secctx, secctx_sz);
+     secctx = NULL;
+   }
Secctx diagram

Binder Transaction on receiver side

- data
- offsets
- extra
- SELinux context

List of:
- BINDER_TYPE_* object
- int32
- CString
- ....

Offset of Binder_TYPE_* in data part

Store BINDER_TYPE_PTR data
Vulnerability 1: Integer Overflow

- **extra_size** is controlled by the user
  - **buf_offset** can be set with an invalid value
- Patched the April 24 2019
- Identified as CVE-2019-2181 in September 2019

```c
extra_buffers_size += ALIGN(secctx_sz, sizeof(u64));
// ...
size_t buf_offset = ALIGN(tr->data_size, sizeof(void *)) +
  ALIGN(tr->offsets_size, sizeof(void *)) +
  ALIGN(extra_buffers_size, sizeof(void *)) -
  ALIGN(secctx_sz, sizeof(u64));
char *kptr = t->buffer->data + buf_offset;
// ...
memcpy(kptr, secctx, secctx_sz);
```
Vulnerability 2: ACL bypass

- Using BINDER_TYPE_PTR

- The Kernel process BINDER_TYPE_* objects
Vulnerability 2 : ACL bypass

- This is an easier way to bypass ACL than the `getpidcon()` race condition !!
- Fixed by commit a565870650 (Jul 9, 2019)
- CVE-2019-2214 (November 2019)
Vulnerability 2 Status

- 1 security bug patched => 2 new security bugs
- 1 trivial bug! Code review!?
PART III - Study of two binder patches

b) fix incorrect calculation for num_valid
Last commits

History for linux / drivers / android / binder.c

Commits on Jan 30, 2020

Merge tag 'for-5.6/io_uring-vfs-2020-01-29' of git://git.kernel.dk/li... 895f8d2
slovens committed 21 days ago

Commits on Jan 22, 2020

binder: fix log spam for existing debugfs file creation. eb143f8
Martin Fuzzey authored and gregkh committed on Jan 10

Commits on Jan 21, 2020

fs: move flip_close() outside of __close_fd get_file() 6e002a4
axbnc committed on Dec 11, 2019

Commits on Dec 14, 2019

binder: fix incorrect calculation for num_valid 1698174
Todd Kjcs authored and gregkh committed on Dec 13, 2019
Security patch?

- It seems a security patch
- Date: December, 13 2019
- No CVE, No advisory
  - Edit 03/03/2020: CVE-2020-0041!
- No public informations

=> Let’s study the bug!
num_valid invalid * => /

num_valid is used as parameter of binder_fixup_parent(...) call
Remember: `BINDER_TYPE_PTR` allows to patch a parent buffer.
binder_fixup_parent rules

1 - binder_validate_ptr()
   - Parent index < num_valid

2 - binder_validate_fixup()
   - Only allow fixup on the last buffer object that was verified, or one of its parents
   - We only allow fixups inside a buffer to happen at increasing offsets
Rule example : Valid

Offsets :

A - offset 0
B - offset 0
C - offset 16
D - offset 0
E - offset 32
Rule example: Valid

Offsets:

- A - offset 0
- B - offset 0
- C - offset 16
- D - offset 0
- E - offset 32

Last verified object

num_valid
Rule example : Valid

Offsets :

A B C D E

A

0

B - offset 0

C - offset 16

D - offset 0

E - offset 32
Rule example: Valid

Offsets:

A   B   C   D   E

A - offset 0
B - offset 0
C - offset 16
D - offset 0
E - offset 32

16
Rule example: Valid

Offsets:

- A: 16
- B: offset 0
- C: offset 16
- D: offset 0
- E: offset 32
Rule example : Valid

Offsets:

```
A B C D E
```

```
A 16
B - offset 0
C - offset 16
D - offset 0
E - offset 32
```
Rule example: Invalid

- Rule: We only allow fixups inside a buffer to happen at increasing offsets.
Rule example : Invalid

Offsets :

A

B - offset 0

D - offset 0

C - offset 16

A B C D
Rule example : Invalid

Offsets : A B C D

A

B - offset 0

D - offset 0

C - offset 16
Rule example : Invalid

Offsets : A B C D

A

B - offset 0

D - offset 0

C - offset 16
Rule example: Invalid

Offsets:

- A
- B - offset 0
- C - offset 16
- D - offset 0
Rule example: Invalid

- Only allow fixup on the last buffer object that was verified, or one of its parents
What is the bug?

- Confusion between index in a table and offsets

```
//vulnerable code
size_t num_valid = (buffer_offset - off_start_offset) * sizeof(binder_size_t);
```

- If current offset is 0x10
  - Wanted num_valid = 0x10/8 = 2
  - Buggy code, num_valid = 0x10 * 8 = 0x80!
What is the impact?

- An object can have an unverified parent offset
Objective:

- Bypass `binder_validate_fixup` validation

Use an arbitrary buffer parent to patch an invalid parent offset!

/* binder_validate_fixup comments:
   * For safety reasons, we only allow fixups inside a buffer to happen
   * at increasing offsets; additionally, we only allow fixup on the last
   * buffer object that was verified, or one of its parents.
*/
Naive try

Offsets:

A

B - offset 0

C - offset 16

Non valid data

num_valid
Naive try

Offsets: A C B

A

B - offset 0

C - offset 16
Naive try

- Only allow fixup on the last buffer object that was verified, or one of its parents
Solution

- Change a parent during the validation!
- Using the extra buffer!
  - Use a parent index which is in extra part
  - Each time a BINDER_TYPE_PTR is valid, its buffer is copied in extra part!
buffer = XXXX
buffer = XXXX
Data A

buffer = XXXX
A

B - offset 0

C - offset XXX

D
buffer = XXXX

buffer = XXXX

Data A

Data B

offset A
offset B
offset C

parent index

data

offsets

extra

offset A
A

buffer = XXXX

B - offset 0

C - offset XXX

Data A

Data B

Data C

parent index
Patch buffer

```c
buffer_offset = bp->parent_offset +
    (uintptr_t)parent->buffer - (uintptr_t)b->user_data;
if (binder_alloc_copy_to_buffer(&target_proc->alloc, b, buffer_offset,
    &bp->buffer, sizeof(bp->buffer))) {
    binder_user_error("%d:%d got transaction with invalid parent offset\n", proc->pid, thread->pid);
    return -EINVAL;
}
```

- **Value controlled:**
  - `parent\rightarrow buffer`
  - `bp\rightarrow parent\_offset`

- **Value writing:** pointer to C buffer (controlled) in extra data
  - `alloc\_buffer + buffer\_offset = @(C buffer)`
Exploit Limitations

- $\text{binder\_alloc\_copy\_to\_buffer}$ checks if $\text{buffer} + \text{offset}$ is in the allocated buffer of this transaction!
- Kernel memory is not reachable
- Need to know the target memory mapping!
  - Need a memory leak!
PoC Setup

- Android emulator (QEMU) X86_64
  
  ```bash
  ./emulator -avd Pixel_3a_XL_API_29_64b -kernel custom_bzImage -show-kernel -no-window -verbose -ranchu -no-snapshot
  ```

- Build custom kernel to add debug log

```c
static void binder_alloc_do_buffer_copy(struct binder_alloc *alloc,
    bool to_buffer,
    struct binder_buffer *buffer,
    binder_size_t buffer_offset,
    void *ptr,
    size_t bytes)
{
    if (!check_buffer(alloc, buffer, buffer_offset, bytes)){
        size_t buffer_size = binder_alloc_buffer_size(alloc, buffer);
        pr_info("[JB] check_buffer buffer_size : 0x%lx bytes = 0x%lx offset = 0x%lx\n", buffer_size, bytes, buffer_offset);
    }
    /* All copies must be 32-bit aligned and 32-bit size */
    BUG_ON(!check_buffer(alloc, buffer, buffer_offset, bytes));
}
```
POC - Crash

```
[  148.291702] binder: 3410:3410 ioctl c0306201 7fff98cb5f20 returned -22
[  148.295022] binder_alloc: [JB] check_buffer buffer_size : 0x10e0 bytes = 0x8
    offset = 0x71829fda8b8
[  148.299460] ------------[ cut here ]-------------
[  148.301159] kernel BUG at drivers/android/binder_alloc.c:1133!
[  148.303042] invalid opcode: 0000 [#1] PREEMPT SMP NOPTI
[  148.304537] Modules linked in:
[  148.307397] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS rel-
  1.11.1-0-g0551a4be2c-prebuilt.qemu-project.org 04/01/2014
[  148.311690] task: 0000000086b3eedc task.stack: 000000000a1c204
[  148.313730] RIP: 0010:binder_alloc_do_buffer_copy+0x8d/0x15e
[  148.317540] RAX: 0000000000000000 RBX: ffff9e98a62079c0 RCX: 0000000000000008
[  148.320403] RDX: ffff9e98aa0e5dd8 RSI: 0000000000000000 RDI: ffff9e98aa0e5da0
[  148.323268] RBP: ffffa11501effaa0 R08: 0000000000000000 R09: 0000000000000000
[  148.328290] R13: 0000071829fda8b8 R14: ffff9e98aa0e5da0 R15: ffff9e98aa0e5da0
[  148.330194] FS: 0000000000000000 GS: ffff9e98bfc00000(0000)
knlGS:0000000000000000
[  148.331780] CS: 0010 DS: 0000 ES: 0000 CR0: 0000000000000000
[  148.332740] CR2: 00007435311239a0 CR3: 00000000000000000
[  148.333848] Call Trace:
[  148.334207] binder_alloc_copy_to_buffer+0x1a/0x1c
[  148.334895] binder_fixup_parent+0x186/0x1ac
```
We already have the leak

- In Android Java applications are forked from Zygote (or Zygote64)
- The memory mapping is the same!
- The reception buffer `/dev/binder` is known
- We can target all apps forked of the same Zygote
Ideas

- We can overwrite verified data in a binder transaction
- Overwrite existing objects:
  - File descriptors
  - Binder reference => to a controlled object
  - Structures (like hild_string)
    - Change the address
    - Change the size

```c
struct hild_string {
    details::hidl_pointer<const char> mBuffer;
    uint32_t mSize;
    bool mOwnsBuffer;
};
```
Vulnerable devices

- Need a recent kernel commit bde4a19fc04f5 - Feb 8, 2019
- Pixel 4 – msm-coral-4.14-android10
- Pixel 3/3a XL – msm-bonito-4.9-android10
- Fixed with the update of March 2020
Conclusion

- Binder is a critical Android component
- Attack surface is quite large (kernel + libs)
- Attack windows of several months
- Binder driver update ...
  - Depends on vendors !!
  - Many linux branches
  - Need CVE for backports !
References

AVEZ-VOUS DES QUESTIONS ?

MERCI DE VOTRE ATTENTION,

SYNACKTIV