SYNACKTIV
DIGITAL SECURITY

Binder

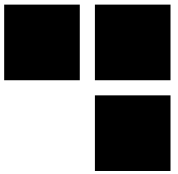
Étude du mécanisme de communication interprocessus d'Android et de ses vulnérabilités
-
Binder IPC and its vulnerabilities

Présenté 06/03/2020

Pour THCON 2020

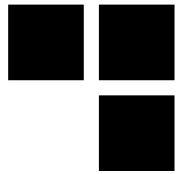
Par Jean-Baptiste Cayrou

Who I am



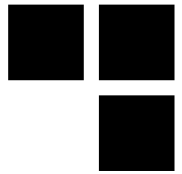
- **Jean-Baptiste Cayrou (@jbcayrou)**
- **Synacktiv:**
  - Offensive security company
  - > 60 ninjas
  - 3 teams : pentest, reverse engineering, development
- **Reverser at Synacktiv:**
  - Focus on low level reverse, vulnerability research, source code audit
  - Work since several years on Android
- **Binder articles on Synacktiv blog**

Introduction

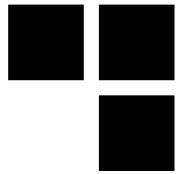


- **Binder : Kernel Module for communications between Android processes in Android**
- **Hot topic**
  - Exploitation in the wild discovered by Google
  - Recent critical vulnerabilities
- **A lot of documentation for high level parts but missing for low level behavior :(**
  - => Start to study Binder internals

Summary

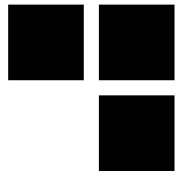


- **Part I : Binder presentation**
- **Part II : Binder vulnerabilities**
- **Part III : Study of two binder patches**



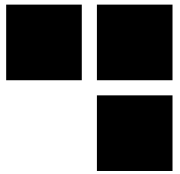
PART I - Presentation of Binder

History



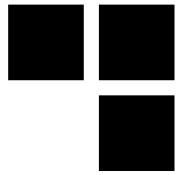
- **Android was bought by Google in 2008**
- **Android is based on the Linux kernel with specific drivers**
  - Binder
  - Ashmem
  - Low Memory Killer
- **Binder is based on OpenBinder implementation**
  - Developed by Be Inc and Palm.
  - Lead by Dianne Hackborn now working at Google

Binder Features



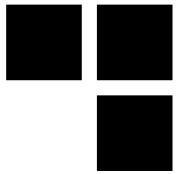
- **Kernel Module for IPC/RPC**
  - ~ 6000 lines of code in *linux/drivers/android/binder_*...
- **Features :**
  - Send messages between applications (sync/async)
  - Call remote function (RPC)
  - Share file descriptors (file, ashmem)
  - Manage references (strong, weak) on remote and local objects
- **Binder messages are called ‘Transactions’**

Binder transaction payload



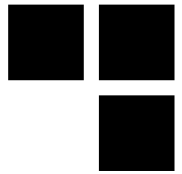
- **Up to 1 MB**
- **Basic types**
  - Integer, long, strings, simple data (sequence of bytes)
- **Binder Objects**
  - Data relative to a process
  - Need a transformation by the Kernel for the receiver (filedescriptor, local memory, references)

Binder Objects



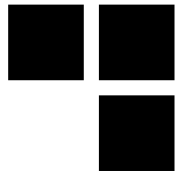
- **Local Object**
  - BINDER_TYPE_BINDER
  - BINDER_TYPE_WEAK_BINDER
- **Remote object**
  - BINDER_TYPE_HANDLE
  - BINDER_TYPE_WEAK_HANDLE
- **File Descriptors**
  - BINDER_TYPE_FD
  - BINDER_TYPE_FDA
- **Buffer**
  - BINDER_TYPE_PTR

Android Framework Interactions



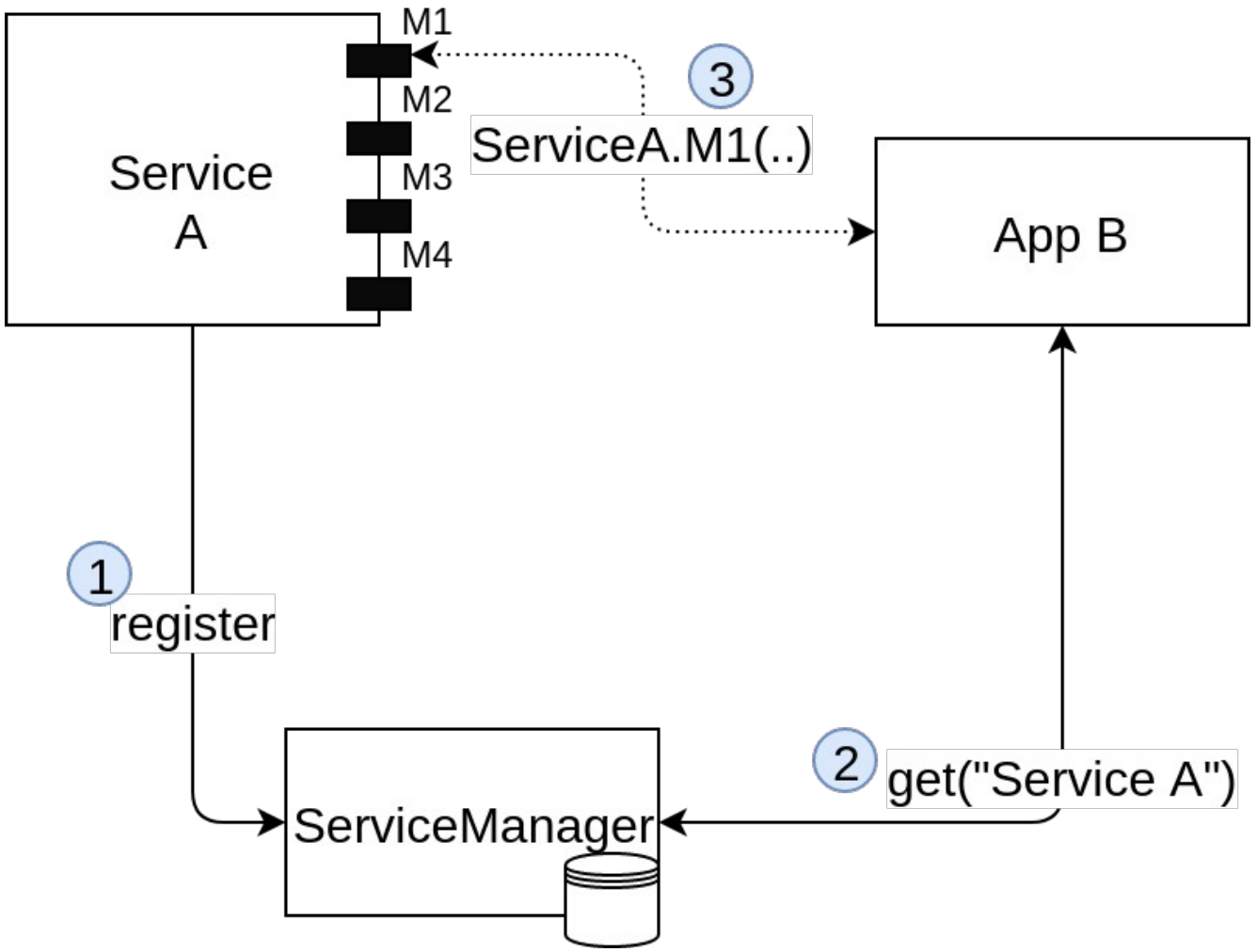
- **Activities**
  - Part of an application (user interface screen)
  - Optionally have arguments
  - Example : Open the browser at this address
- **Content Provider**
  - Database like, accessible by others applications (query, insert, update, remove)
  - Uri : ‘content://<authority>/<path>/<id>’
  - Example : contacts

Android Framework Interactions

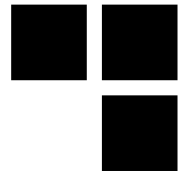


- **Broadcast :**
  - publish-subscribe design pattern
  - Broadcast events to applications (Incoming call, network connection changed ...)
- **Service**
  - A Background application which exposes commands to others (RPC)
  - Main IPC/RPC component, **based on Binder** !
  - Example : ActivityManager, ContentService
- **Activities, Content Providers and Broadcasts are based on Services**

Android Service Interaction

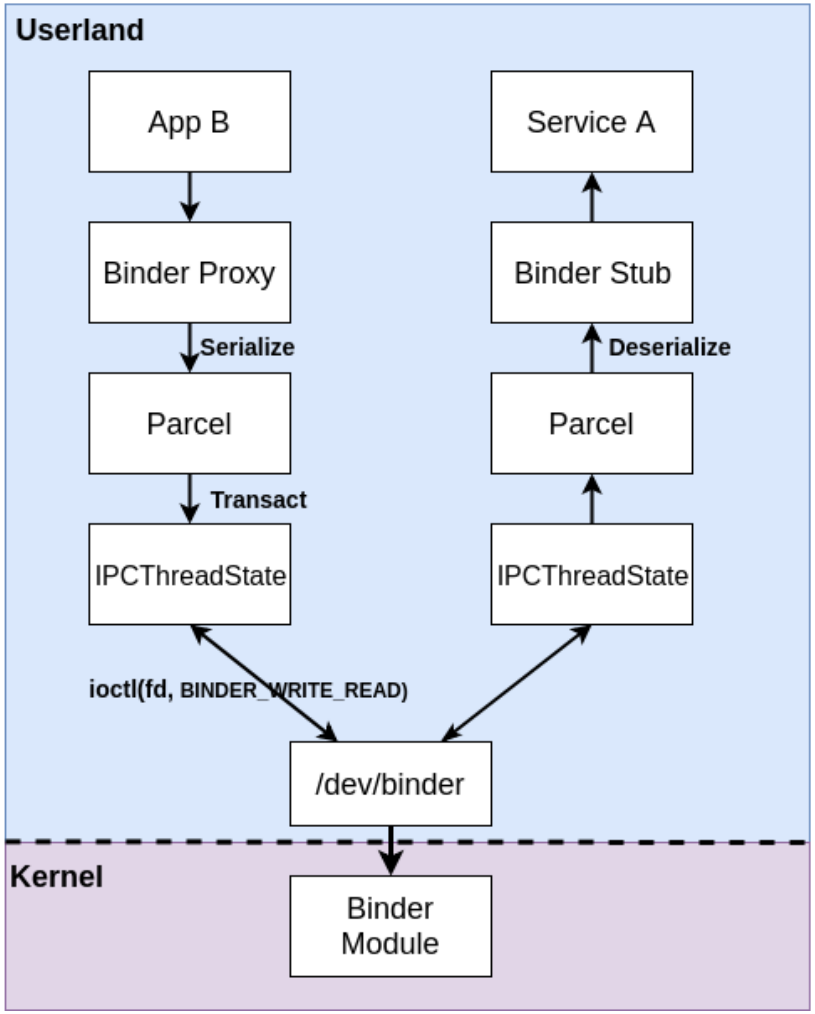


Android Service Interaction

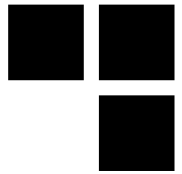


- **How applications know services interfaces ?**
- **Using Interface Definition Languages :**
  - AIDL : For Framework Applications
  - HIDL : For Hardware Service (for vendors)
- **AIDL and HIDL describe RPC functions**
- **Compilers for these languages generate code (C++ and Java):**
  - Binder Proxy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

Binder Call WorkFlow

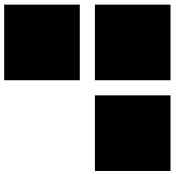


AIDL - Parcel



- **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
  - ...

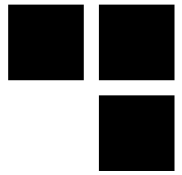
AIDL - File Example



```
// 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);
}
```

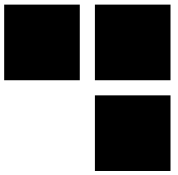
```
::android::binder::Status BpRemoteService::basicTypes(int32_t anInt, int64_t aLong, bool aBoolean, float aFloat,
::android::Parcel _aidl_data;
::android::Parcel _aidl_reply;
::android::status_t _aidl_ret_status = ::android::OK;
::android::binder::Status _aidl_status;
_aidl_ret_status = _aidl_data.writeInterfaceToken(getInterfaceDescriptor());
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::BASICTYPES, _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)



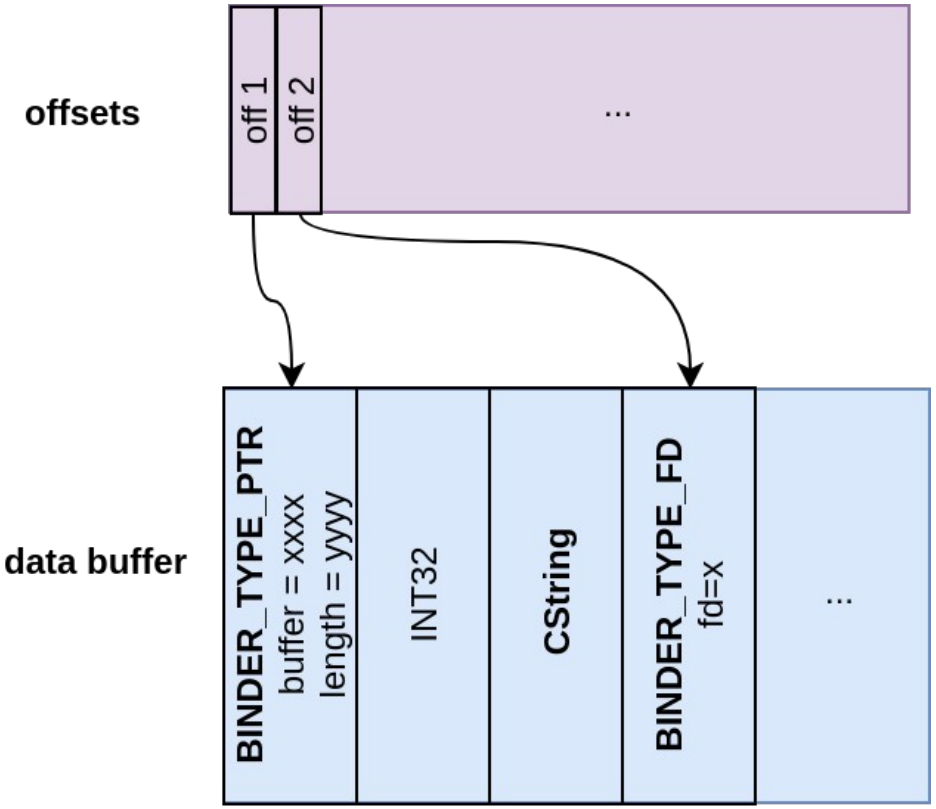
- **Serialization library for HwBinder transactions (C++ and Java)**
  - system/libhwbinder/include/hwbinder/Parcel.h
  - android/os/HwParcel.java
- **Based on the Parcel Framework**
- **Support of data buffer binder object**
  - For instance, C structures containing pointers to others buffers
- **More complex types !**

HIDL – File Format

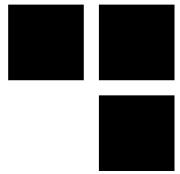


```
interface IFoo {
    uint32_t[3][4][5][6] multidimArray;

    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

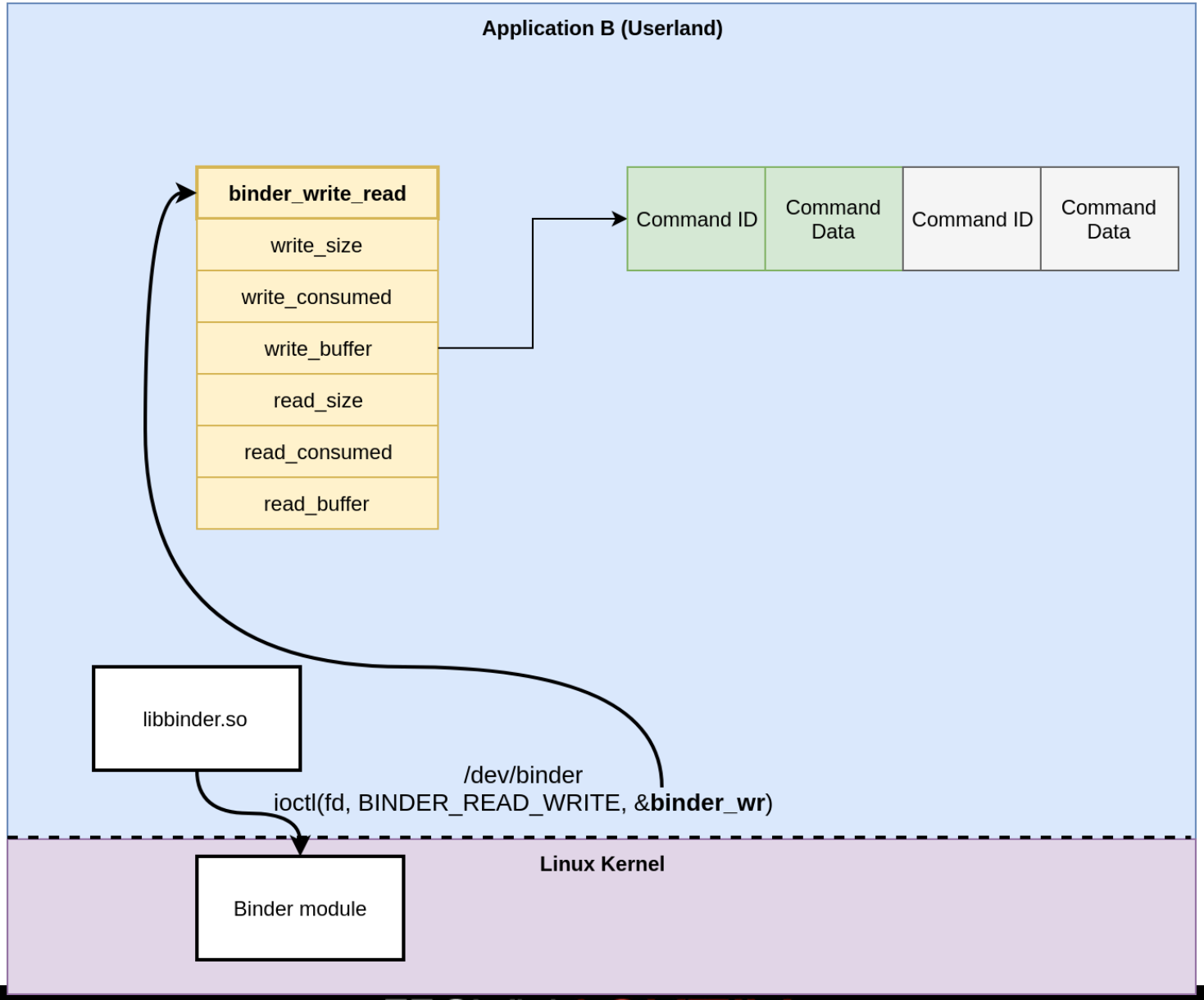


Binder device

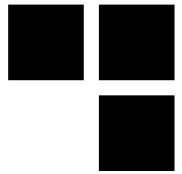


- ***Device : /dev/*binder, */dev/hwbinder, /dev/vndbinder***
- **Mapped as read-only in process memory to receive binder messages**
- **Ioctl commands :**
  - **BINDER_WRITE_READ** => Used for IPC
  - BINDER_SET_MAX_THREADS
  - BINDER_SET_CONTEXT_MGR
  - BINDER_THREAD_EXIT
  - BINDER_VERSION

BINDER_WRITE_READ



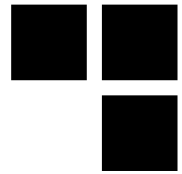
Binder commands



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

- **Tips :**
  - ‘BC_’ : Binder Command
  - ‘BR_’ : Binder Return

BC_TRANSACTION

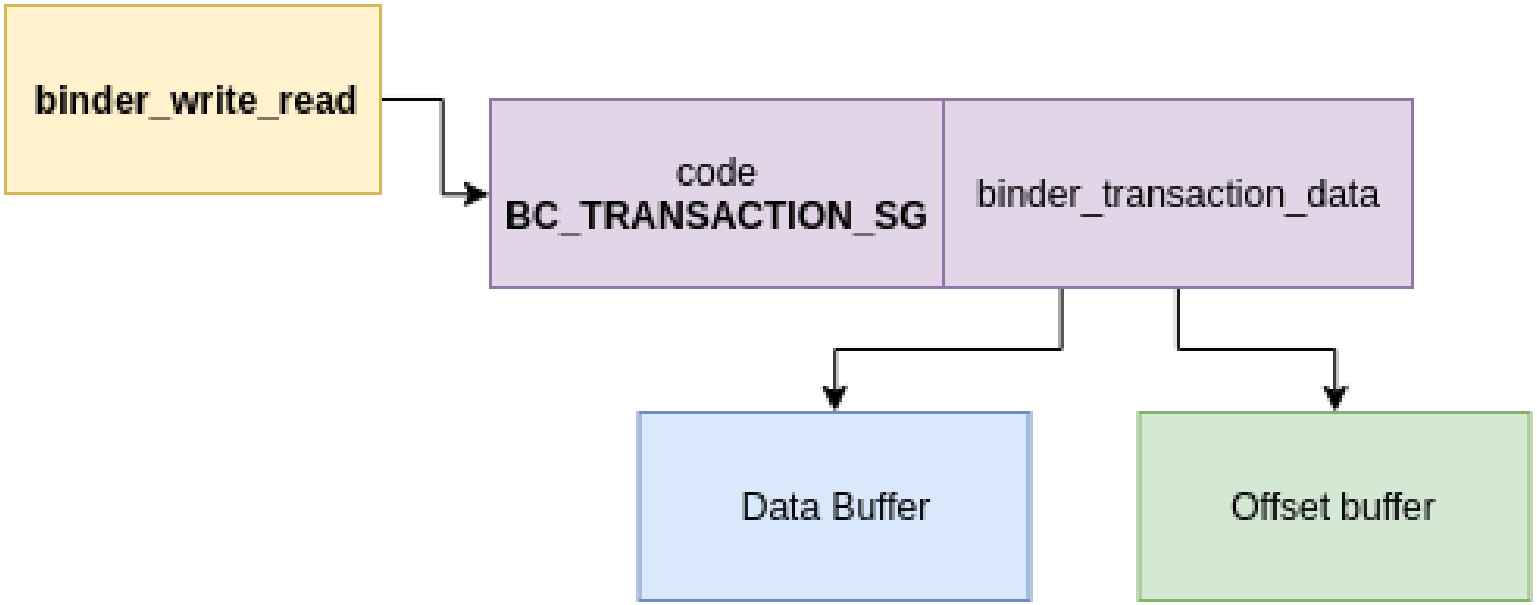


- **Handle :** Remote service ID
- **Code :** Remote method id
- **Buffer** : Message data
- **Offsets :** Objects list

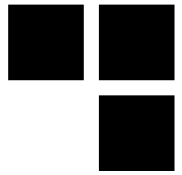
binder_transaction_data
handle
cookie
code
sender_pid
sender_euid
data size
offsets_size
buffer
offsets

- BC_TRANSACTION_SG :
  - + extra_size

Recap of userland view

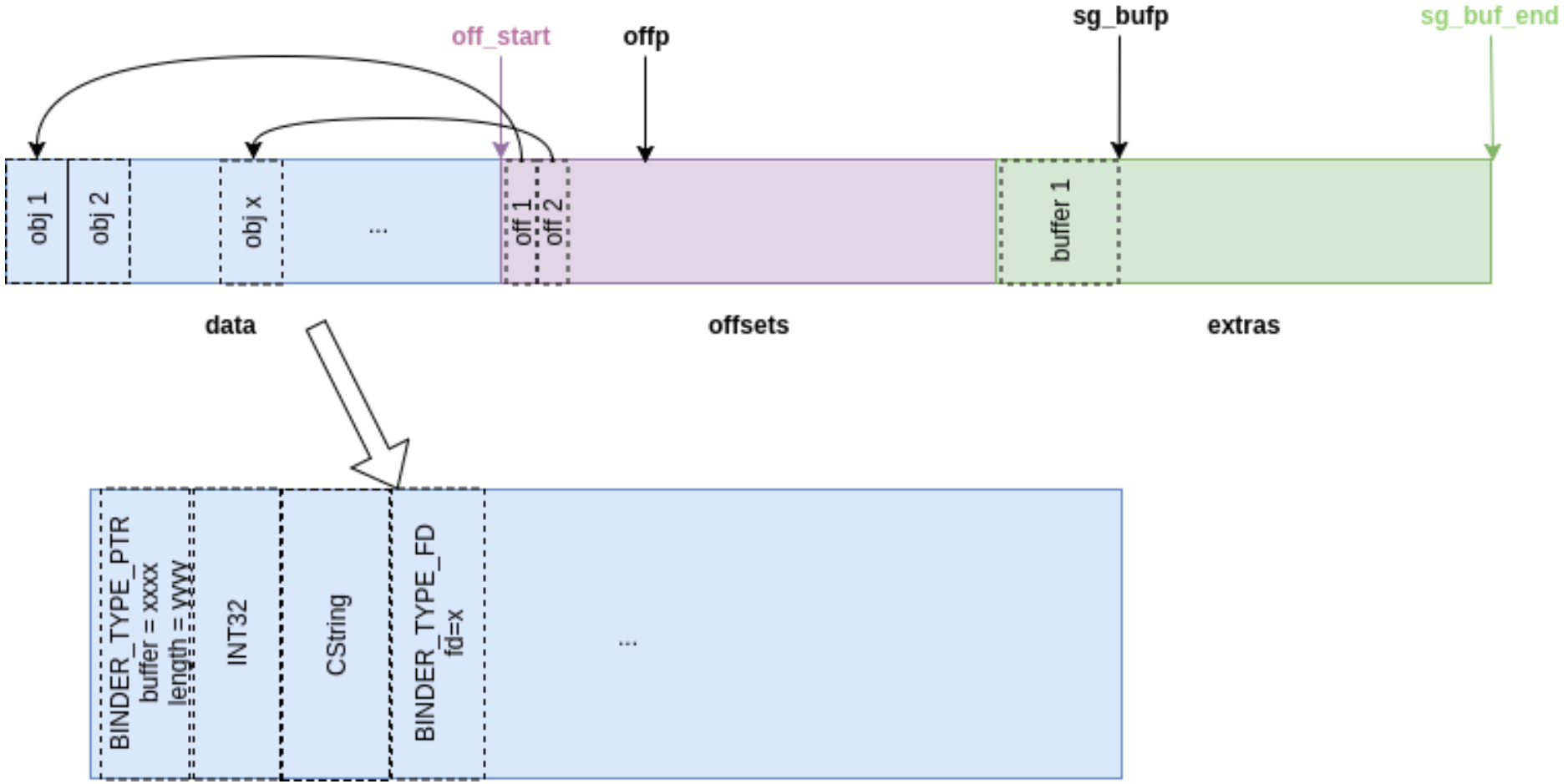


Entering the Kernel !

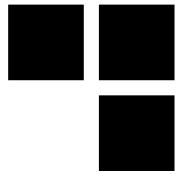


- **The kernel allocates the necessary size in the targeted process (size : data + offsets + extra) and copies the transaction**

- **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)


off_start
offp
sg_bufp
sg_buf_end
obj 1
obj 2
obj x
...
off 1
off 2
buffer 1
data
offsets
extras
BINDER_TYPE_PTR
buffer = xxxx
length = yyyy
INT32
CString
BINDER_TYPE_FD
fd=x
...

Example !

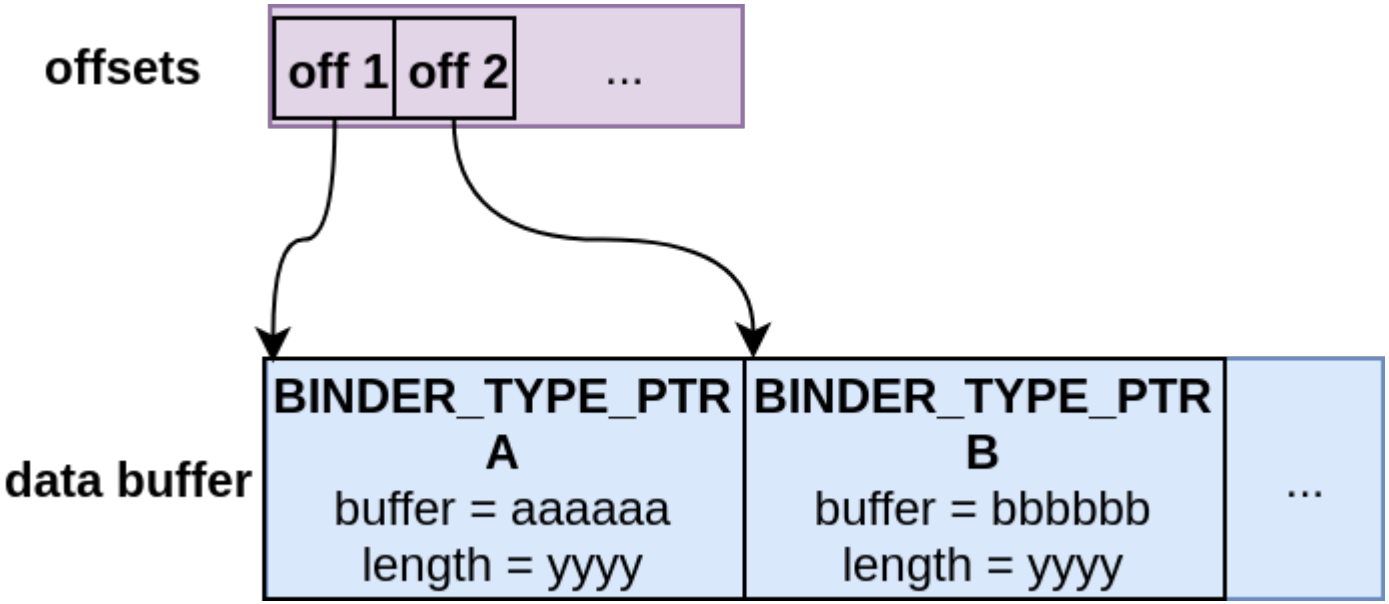


- **Send this hidl_string object :**

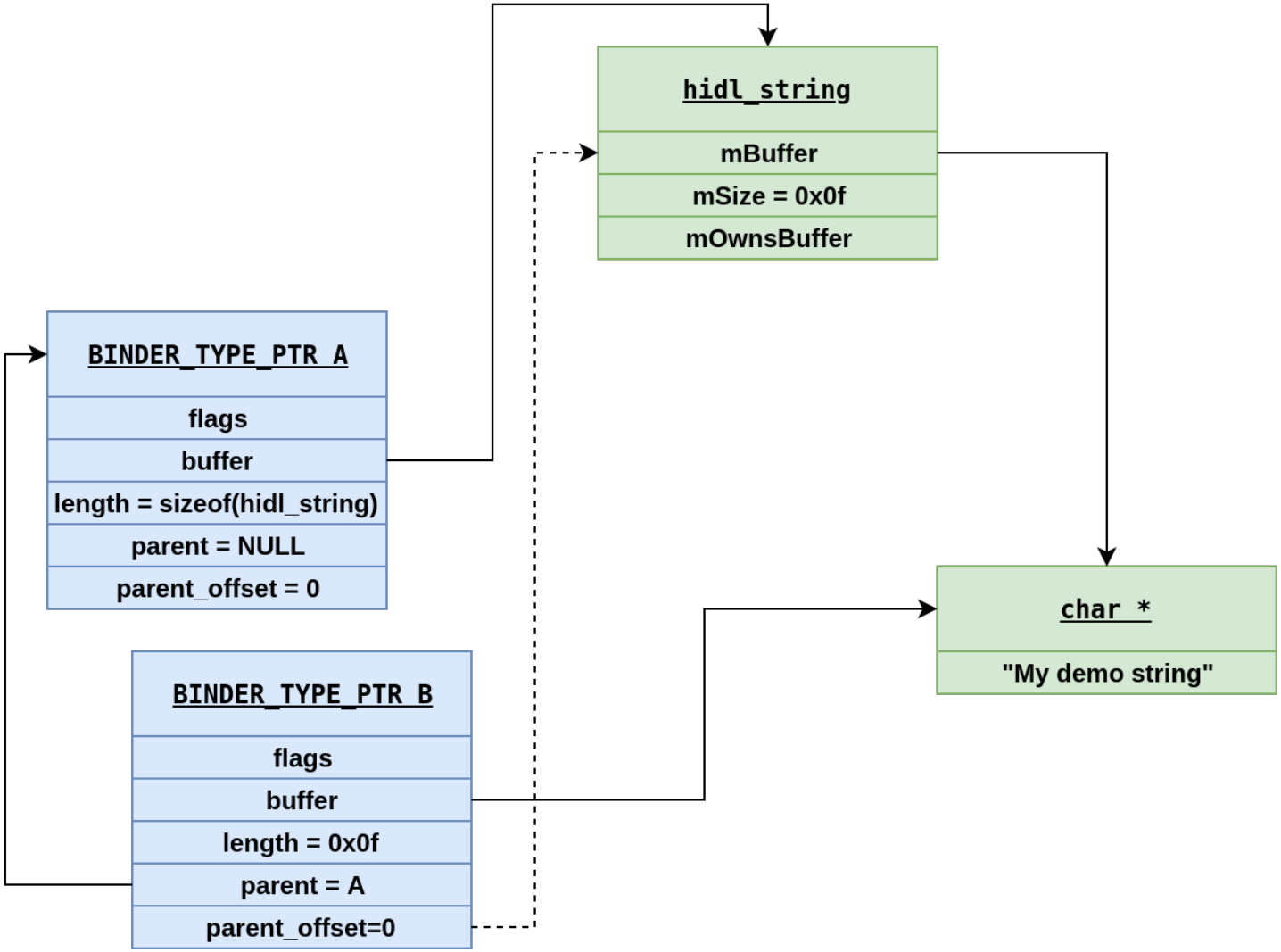
```
struct hidl_string {
    // copy from a C-style string. nullptr will create an empty string
    hidl_string(const char *);
    // ...
private:
    details::hidl_pointer<const char> mBuffer; // Pointer to the real char string
    uint32_t mSize;  // NOT including the terminating '\0'.
    bool mOwnsBuffer; // if true then mBuffer is a mutable char *
};

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

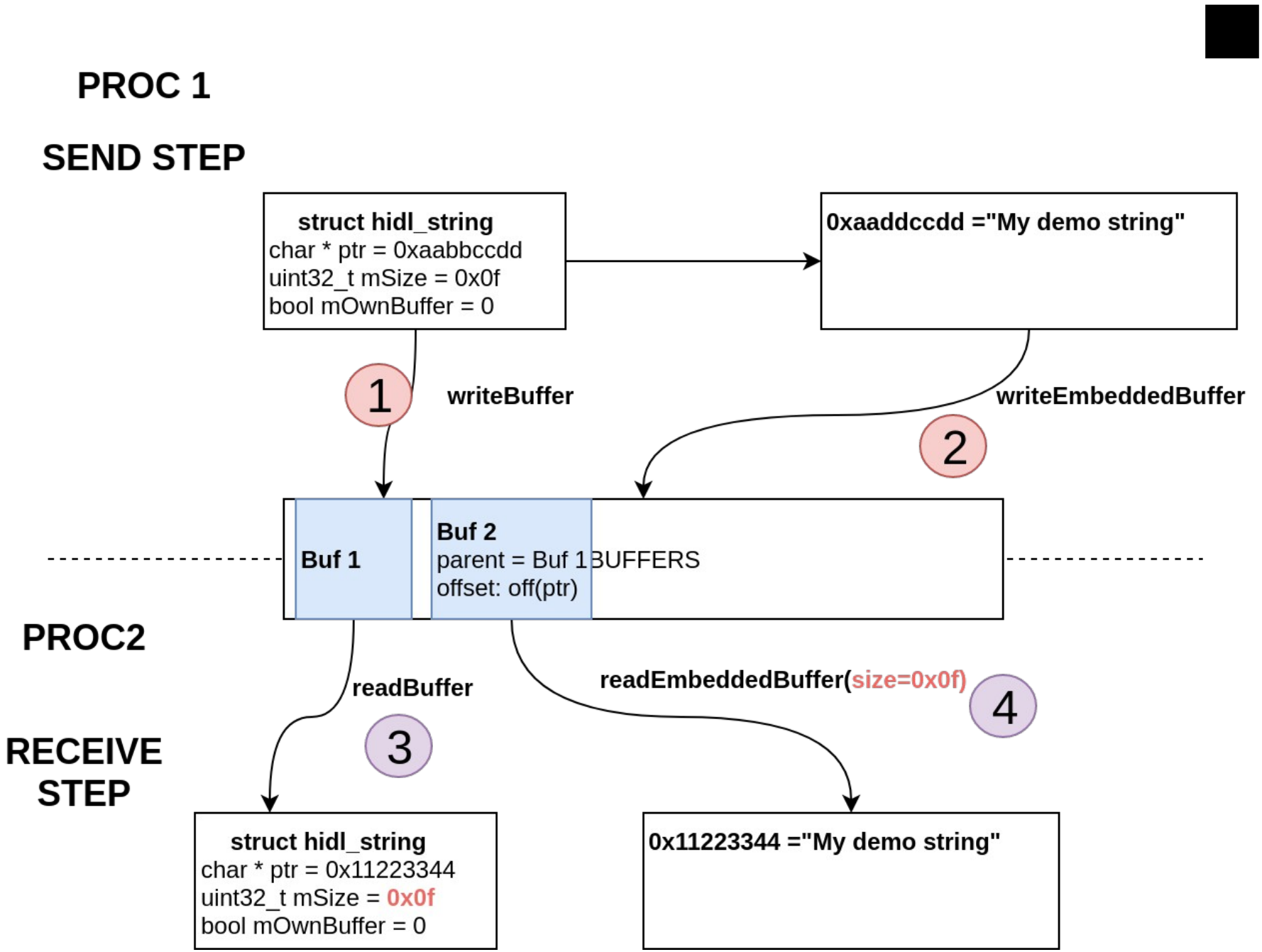
- **When ‘my_obj’ is created, a heap allocation is performed by the constructor to store the real string address in *mBuffer***

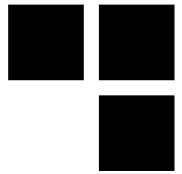
HIDL Parcel


offsets
off 1
off 2
...
data buffer
BINDER_TYPE_PTR
A
buffer = aaaaaa
length = yyyy
BINDER_TYPE_PTR
B
buffer = bbbbbb
length = yyyy
...

HIDL Parcel

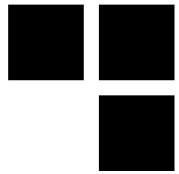

hidl_string
mBuffer
mSize = 0x0f
mOwnsBuffer
BINDER_TYPE_PTR_A
flags
buffer
length = sizeof(hidl_string)
parent = NULL
parent_offset = 0
BINDER_TYPE_PTR_B
flags
buffer
length = 0x0f
parent = A
parent_offset=0
char *
"My demo string"


PROC 1
SEND STEP
struct hidl_string
char * ptr = 0xaabbccdd
uint32_t mSize = 0x0f
bool mOwnBuffer = 0
0xaaddccdd ="My demo string"
1
writeBuffer
writeEmbeddedBuffer
2
Buf 1
Buf 2
parent = Buf 1
offset: off(ptr)
BUFFERS
PROC2
readBuffer
3
readEmbeddedBuffer(size=0x0f)
4
RECEIVE STEP
struct hidl_string
char * ptr = 0x11223344
uint32_t mSize = 0x0f
bool mOwnBuffer = 0
0x11223344 ="My demo string"



PART II - Binder vulnerabilities

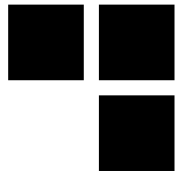
Critical component



- **Binder is the base of Android**
  - All applications use binder (even *unstrusted_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

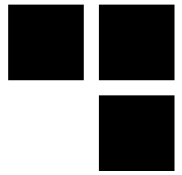
    Libbinder : Parcel

    Libhwbinder : HwParcel

Explore Android Security Bulletins

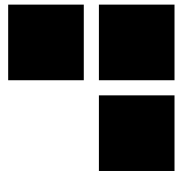
Advisory Data	Patch date	Patch to Advisory	Component	CVE	Type	Severity
01/03/2020	15/12/2019	~ 3 months	Binder Driver	CVE-2020-0041	Eop	High
01/02/2020	16/02/2018	~12 months	Binder Driver	CVE-2020-0030	Eop	High
01/02/2020	15/10/2019	~ 4 months	libbinder	CVE-2020-0026	EoP	High
01/11/2019	12/06/2019	~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

Explore Android Security Bulletins



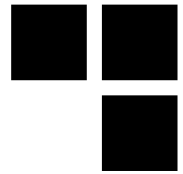
- **20 CVE from 01/2014 to 03/2020 :**
  - 14 Binder Driver
  - 4 libbinder
  - 2 libhwbinder
- **80 % CVE are HIGH (20 % Moderate)**
  - But notation changed in 2017
- **Privilege escalation (EoP) or Information disclosure (ID)**
- **In average 5 months between the patch and the advisory**

Obversations



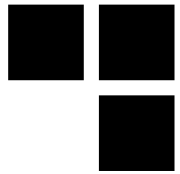
- **Security patches don’t always have a CVE**
  - Difficult to backport patches in the linux kernel !
- **Backports are not always done.**
  - Even on google references branches (kernel/msm)

Example 1 : CVE-2019-2215 (bad binder)



- **Exploits found in the wild by Google**
  - https://googleprojectzero.blogspot.com/2019/11/bad-binder-android-in-wild-exploit.html
- **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)

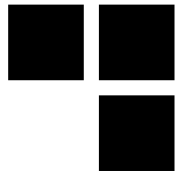


- **Discovered by C0RE Team, Qihoo 360**

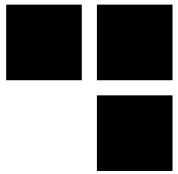
  http://blogs.360.cn/post/Binder_Kernel_Vul_EN.html

- **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

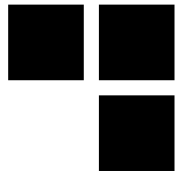


- **Vulnerabilities in kernel are difficult to follow and patch**
  - Vendors have their own kernel
- **Vulnerabilities in AOSP (libbinder/libhwbinder) are less critical and easier to patch**
- **Public patches give an advantage to attackers !**

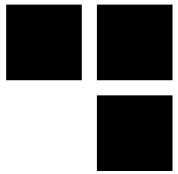


PART III -Study of two binder patches

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)***

binder: create node flag to request sender's security context

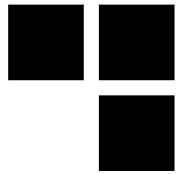
```
To allow servers to verify client identity, allow a node
flag to be set that causes the sender's security context
to be delivered with the transaction. The BR_TRANSACTION
command is extended in BR_TRANSACTION_SEC_CTX to
contain a pointer to the security context string.

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>
```

master (#213) v5.6-rc2 ... v5.1-rc1

Todd Kjos authored and gregkh committed on Jan 14, 2019 1 pare

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

Origin



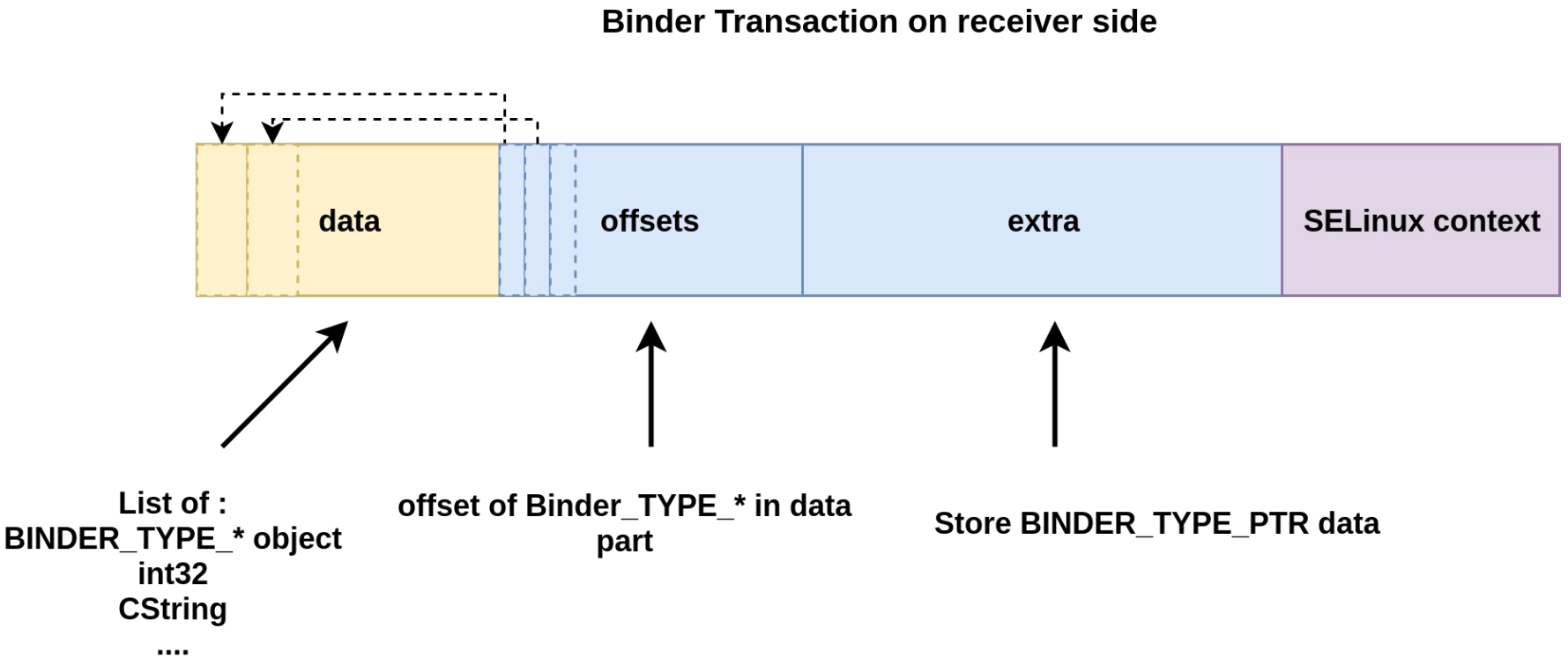
- **Fix CVE-2019-2023 (EoP High)**
  - ACL (Access Control List) bypass due to an insecure permission check, based on the PID of the caller
- **Binder design issue : How to know the identity of the caller ?**
  - Currently using its PID *getpidcon()*
  - However if the caller is dead and the PID is reused the context will be incorrect … (see Jann Horn POC)

    https://bugs.chromium.org/p/project-zero/issues/detail?id=851

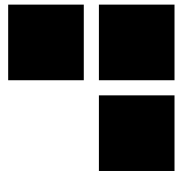
Main part of the patch

```
//@@ -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



Vulnerability 1 : Integer Overflow

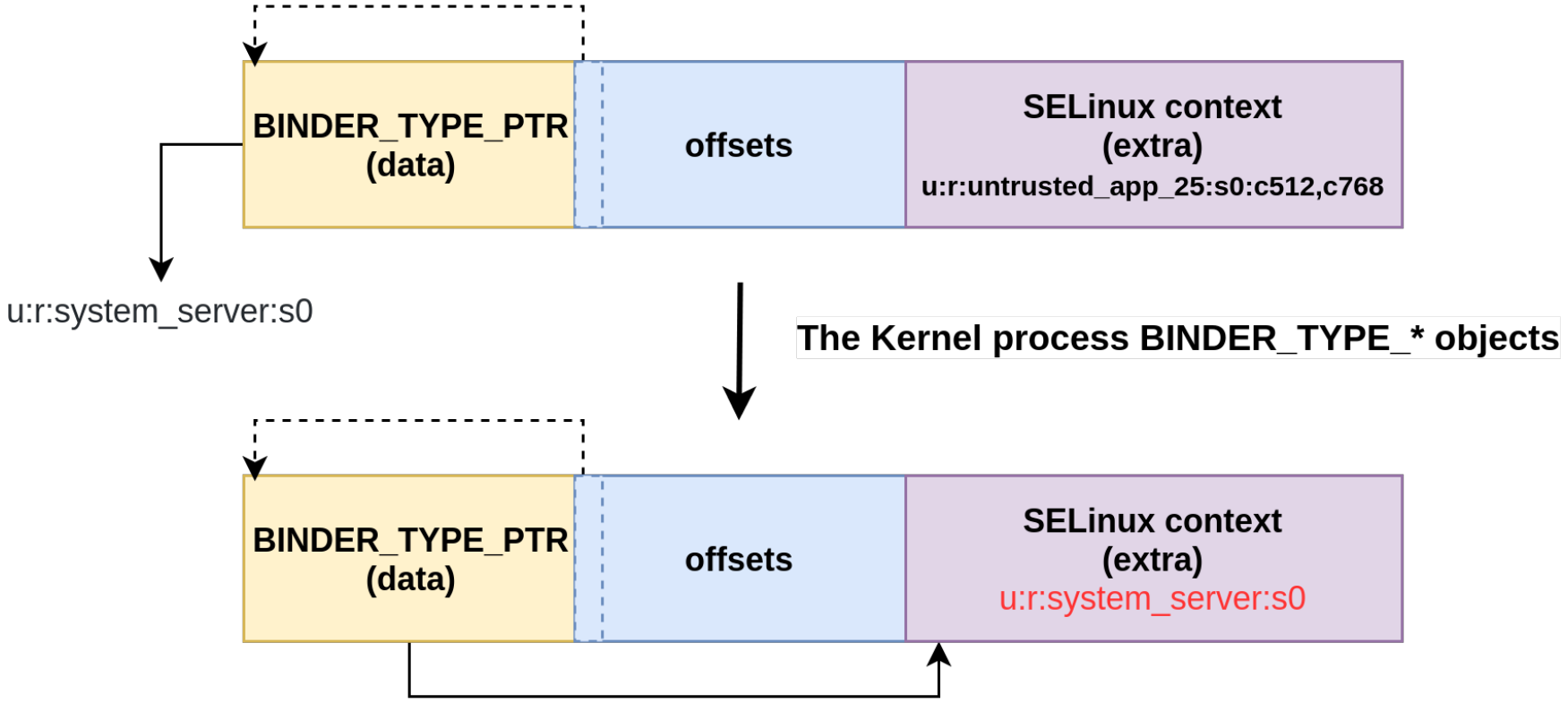


```
extra_buffers_size += ALIGN(secctx_sz, sizeof(u64));
// ...
  size_t buf_offset = ALIGN(tr->data_size, sizeof(void *)) +
            ALIGN(tr->offsets_size, sizeof(void *)) +
            ALIGN(extra_buffers_size, sizeof(void *)) -
            ALIGN(secctx_sz, sizeof(u64));
  char *kptr = t->buffer->data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

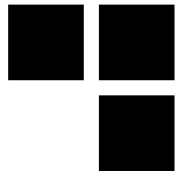
- ***extra_size* is controlled by the user**
  - ***buf_offset* can be set with an invalid value**
- **Patched the April 24 2019**
- **Identified as CVE-2019-2181 in September 2019**

Vulnerability 2 : ACL bypass

- Using BINDER_TYPE_PTR

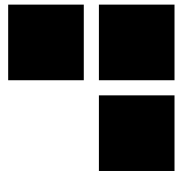


Vulnerability 2 : ACL bypass

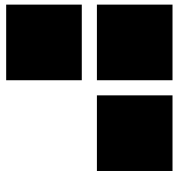


- **This is an easier way to bypass ACL than the *getpidcon()* race condition !!**
- **Fixed by commit a565870650 (Jul 9, 2019)**
- **CVE-2019-2214 (November 2019)**

Vulnerability 2 Status

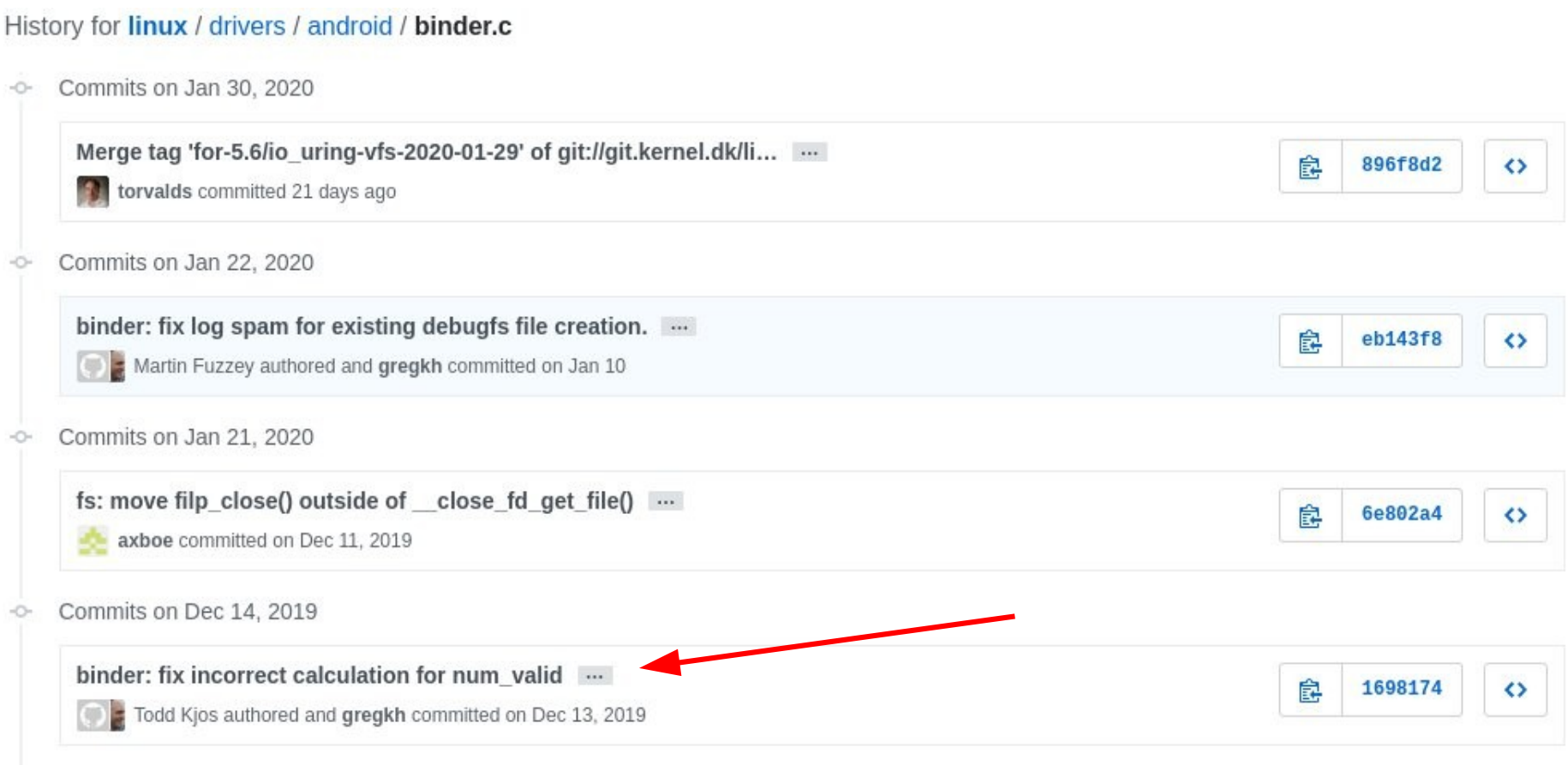


- **1 security bug patched => 2 new security bugs**
- **1 trivial bug ! Code review !?**

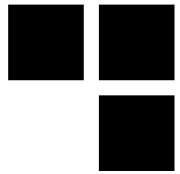


PART III -Study of two binder patches
b) fix incorrect calculation for num_valid

Last commits



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 * => /

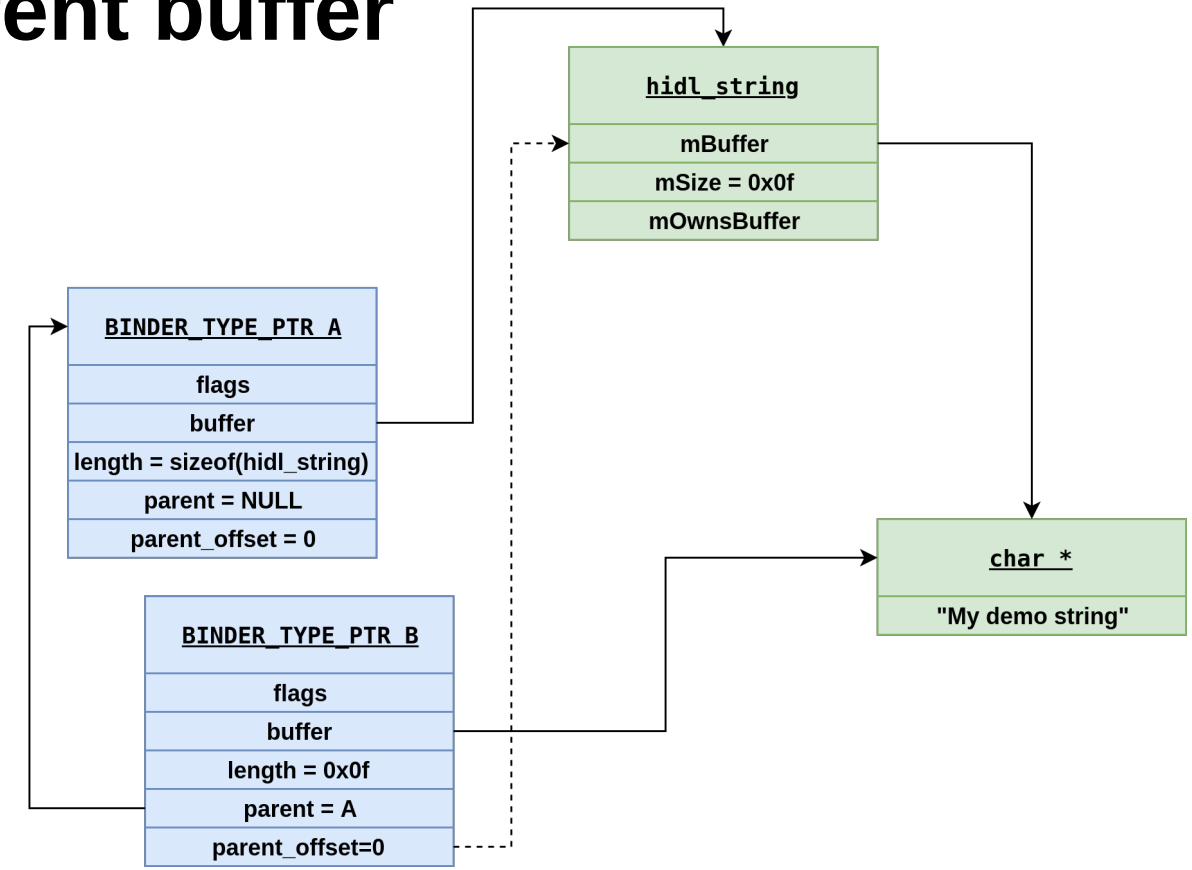
4 drivers/android/binder.c

```
@@ -3310,7 +3310,7 @@ static void binder_transaction(struct binder_proc *proc,
3310 3310                    binder_size_t parent_offset;
3311 3311                    struct binder_fd_array_object *fda =
3312 3312                            to_binder_fd_array_object(hdr);
3313      -                  size_t num_valid = (buffer_offset - off_start_offset) *
     3313 +                  size_t num_valid = (buffer_offset - off_start_offset) /
3314 3314                                               sizeof(binder_size_t);
3315 3315                    struct binder_buffer_object *parent =
3316 3316                            binder_validate_ptr(target_proc, t->buffer,
@@ -3384,7 +3384,7 @@ static void binder_transaction(struct binder_proc *proc,
3384 3384                            t->buffer->user_data + sg_buf_offset;
3385 3385                    sg_buf_offset += ALIGN(bp->length, sizeof(u64));
3386 3386
3387      -                  num_valid = (buffer_offset - off_start_offset) *
     3387 +                  num_valid = (buffer_offset - off_start_offset) /
3388 3388                                           sizeof(binder_size_t);
3389 3389                    ret = binder_fixup_parent(t, thread, bp,
3390 3390                                              off_start_offset,
```

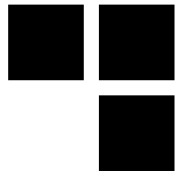
num_valid is used as parameter of *binder_fixup_parent(…)* call

binder_fixup_parent

- **Remember : BINDER_TYPE_PTR allows to patch a parent buffer**

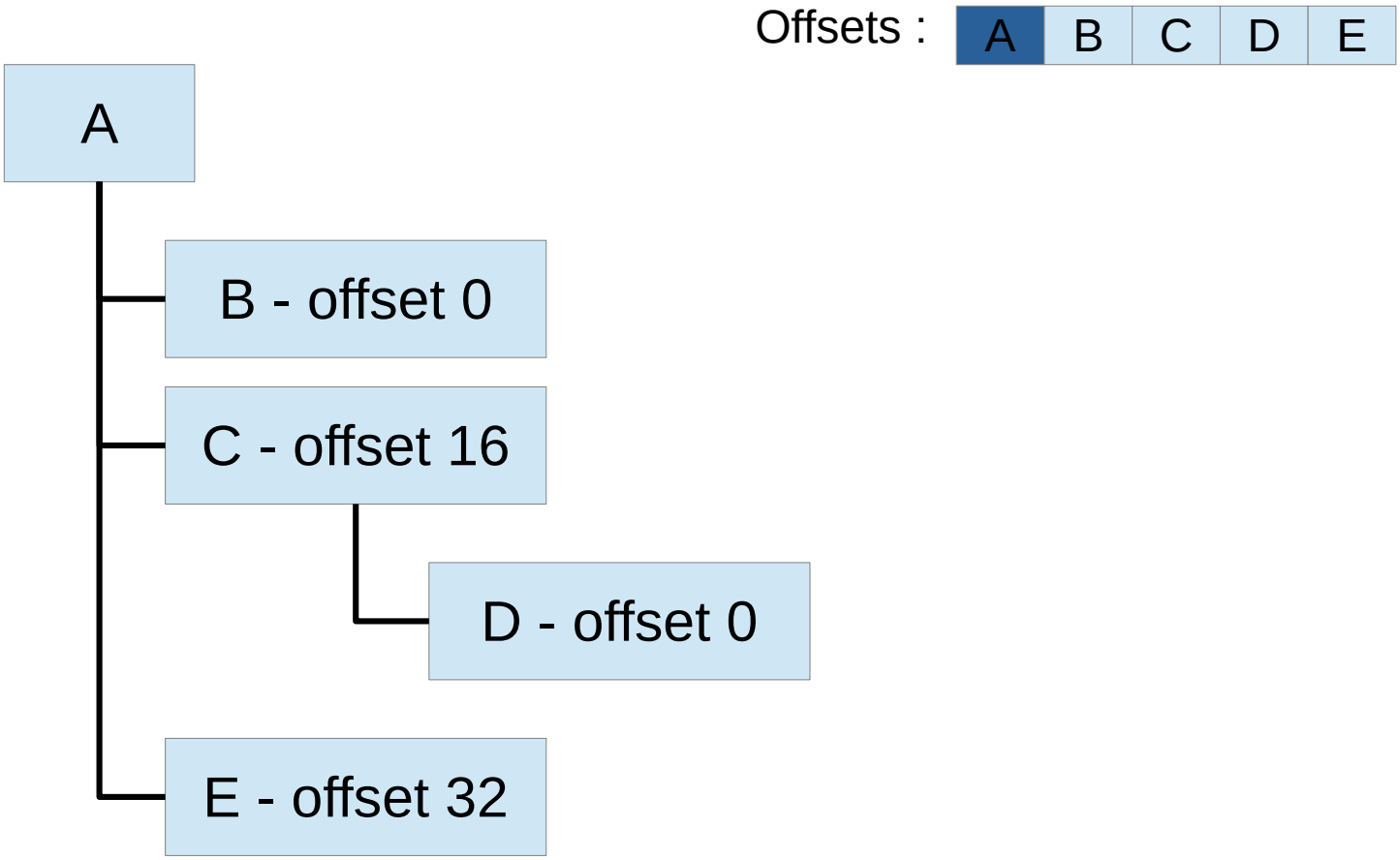


binder_fixup_parent rules

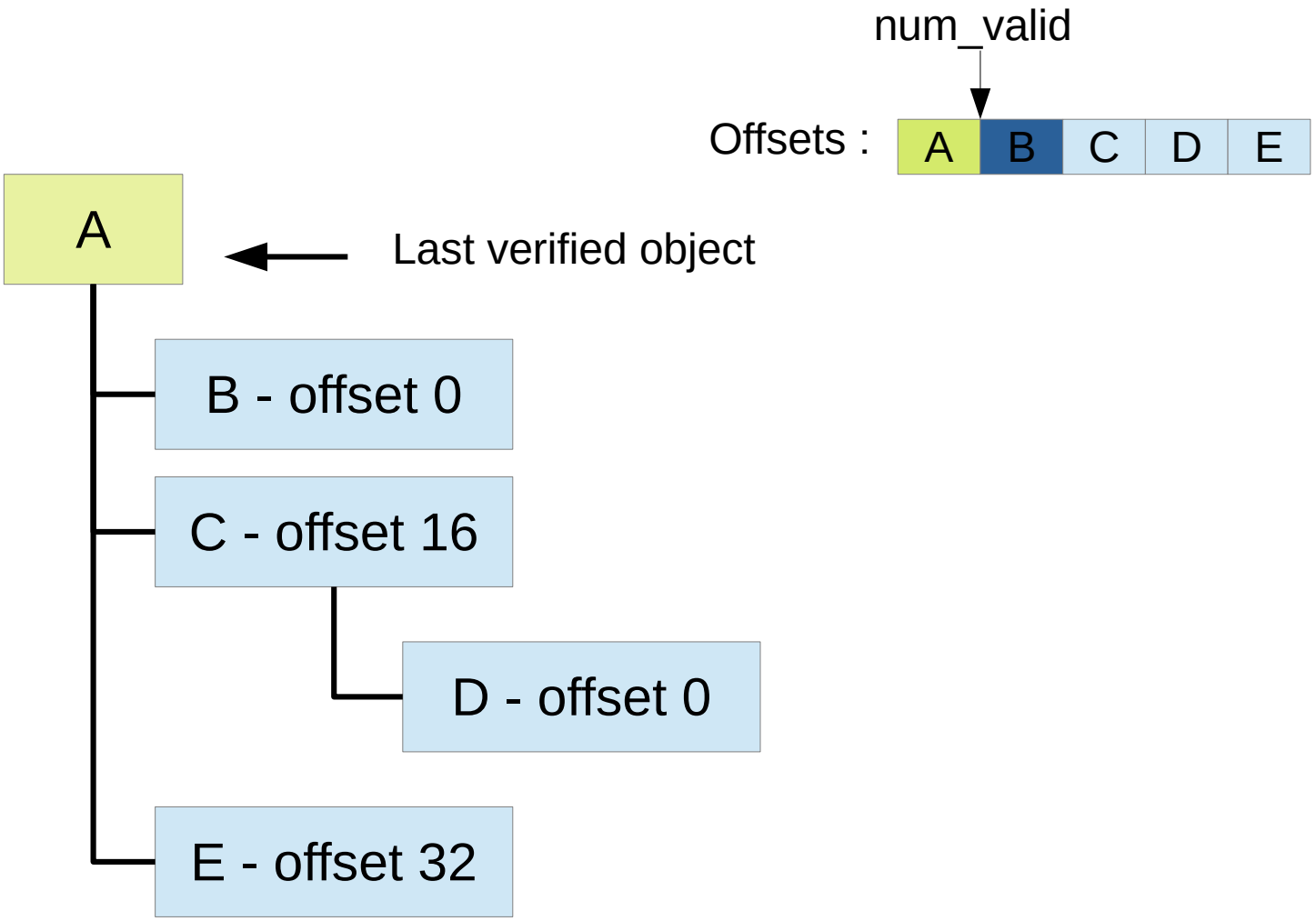


- **1 - binder_validate_ptr()**
  - Parent index < num_valid
- **2 - binder_validate_fixup()**
  - Only allow fixup on the last buffer object that was verified, or one of its parents
  - We only allow fixups inside a buffer to happen at increasing offsets

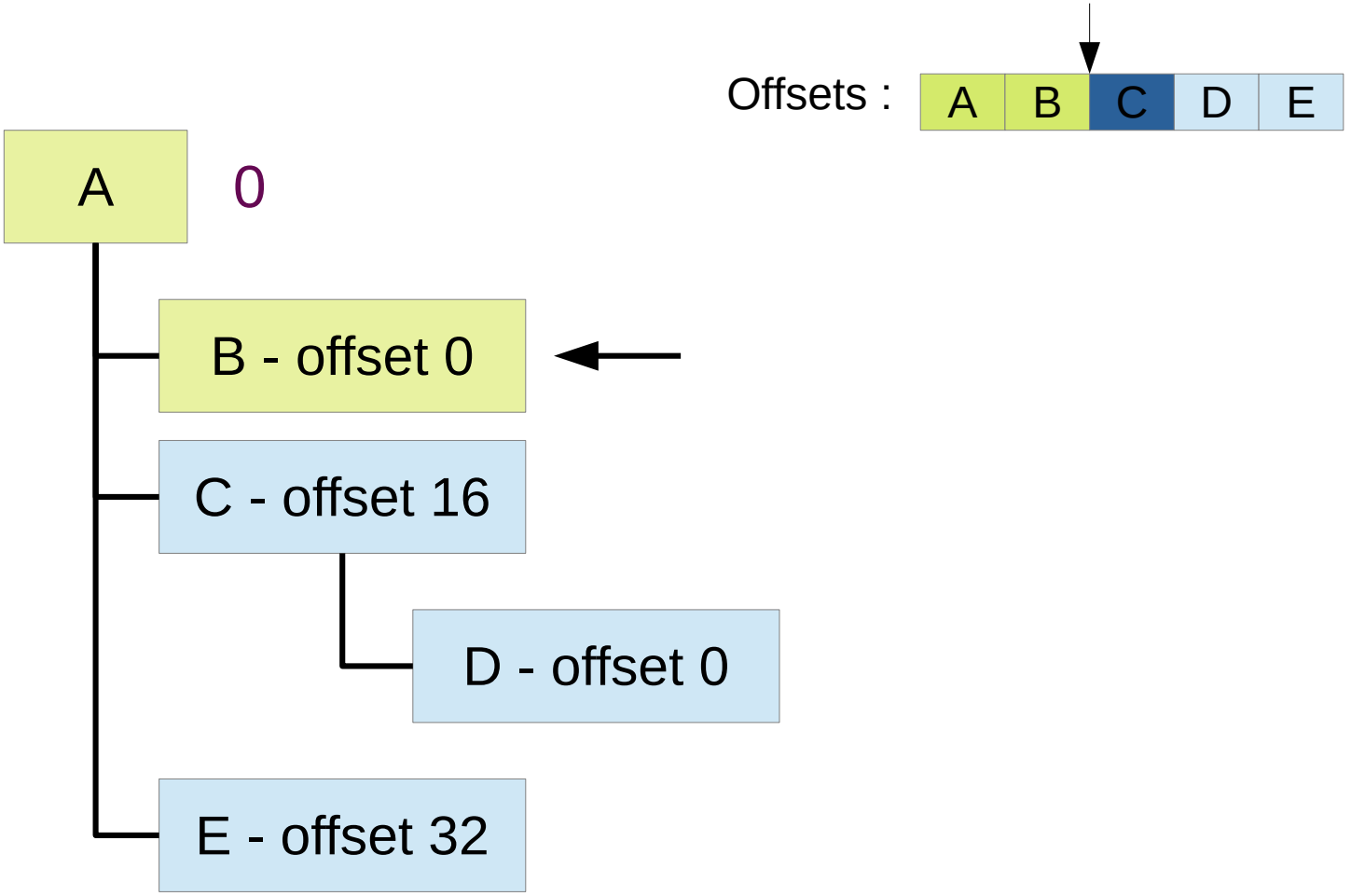
Rule example : Valid



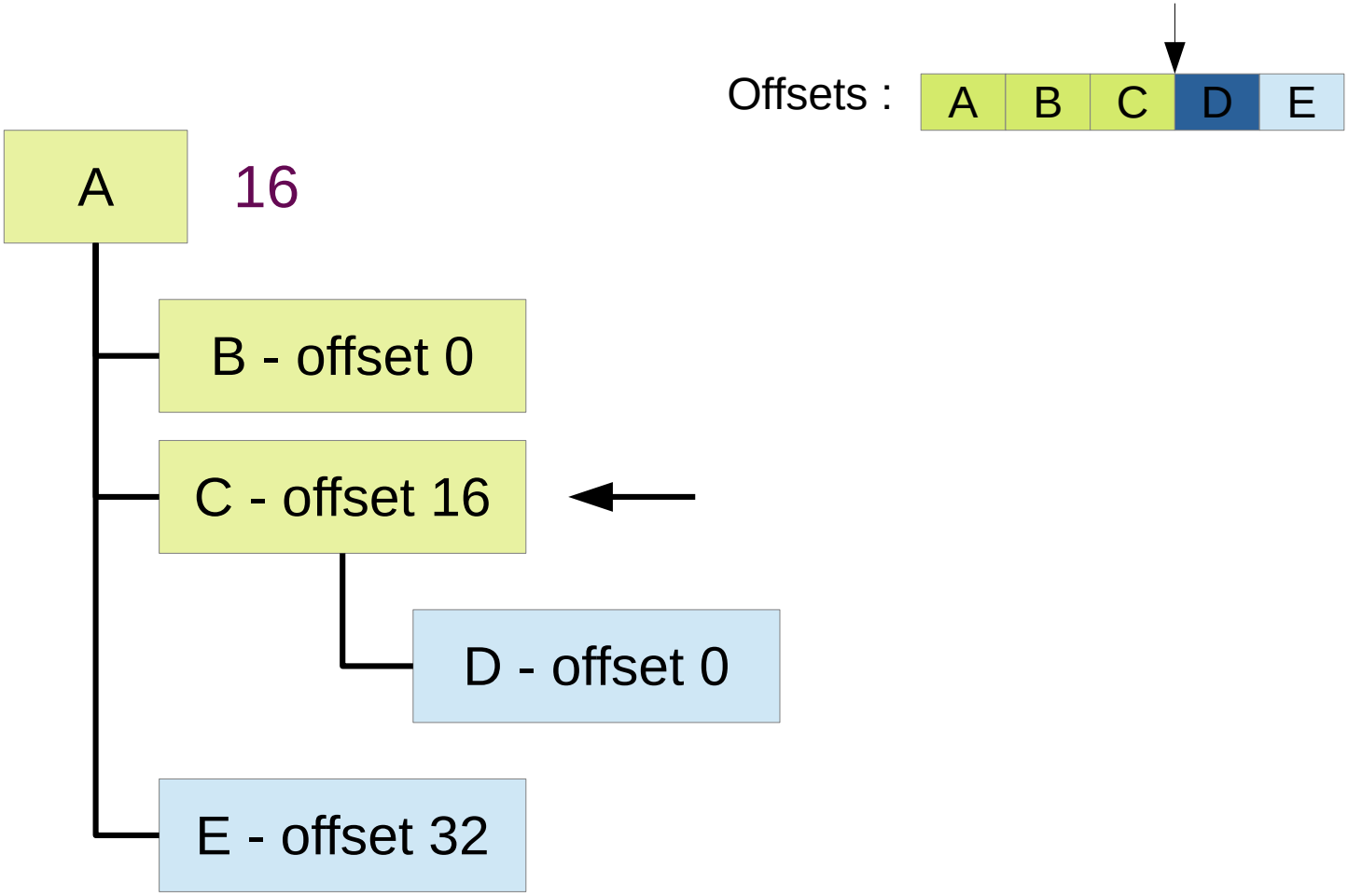
Rule example : Valid



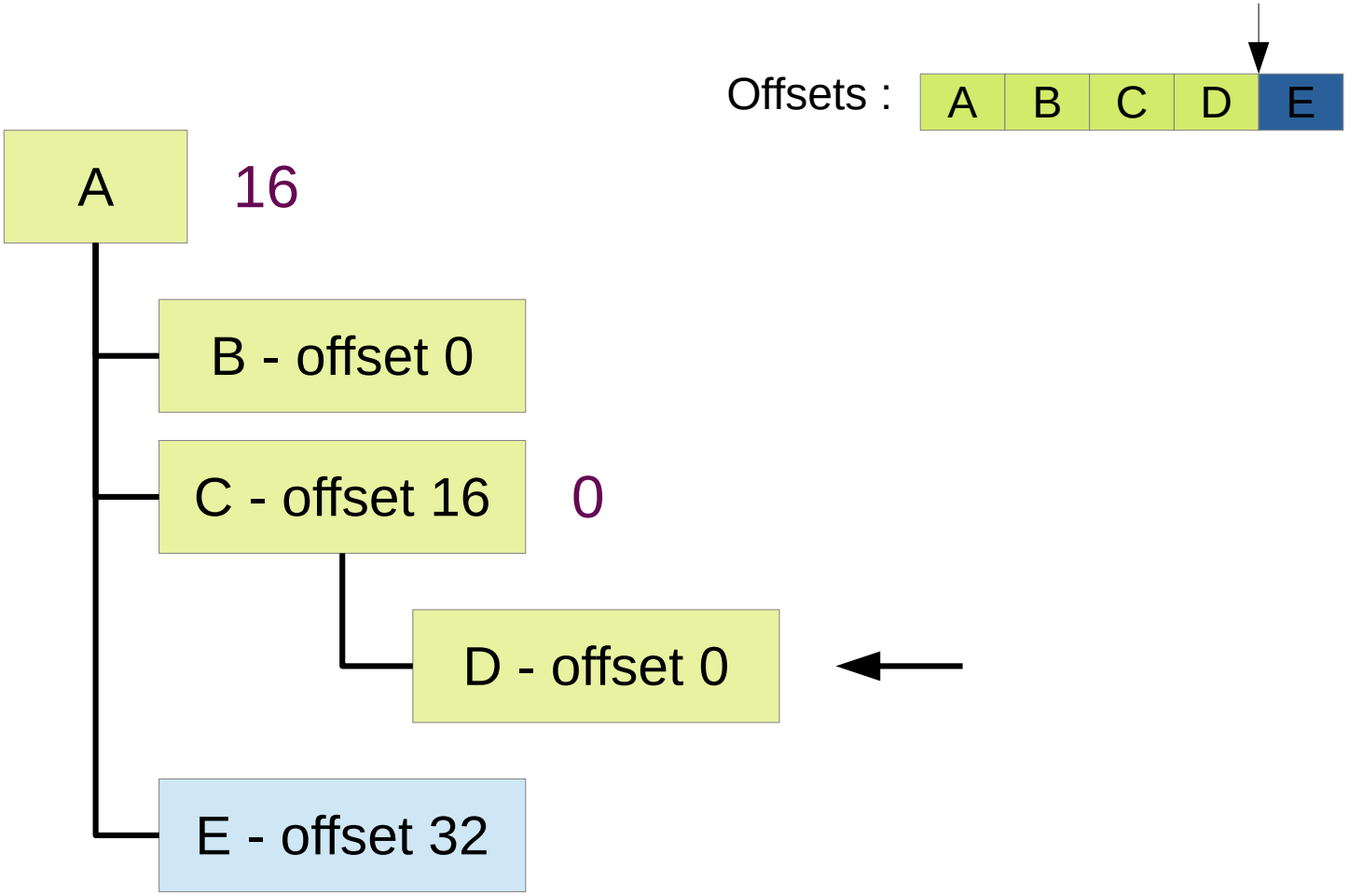
Rule example : Valid



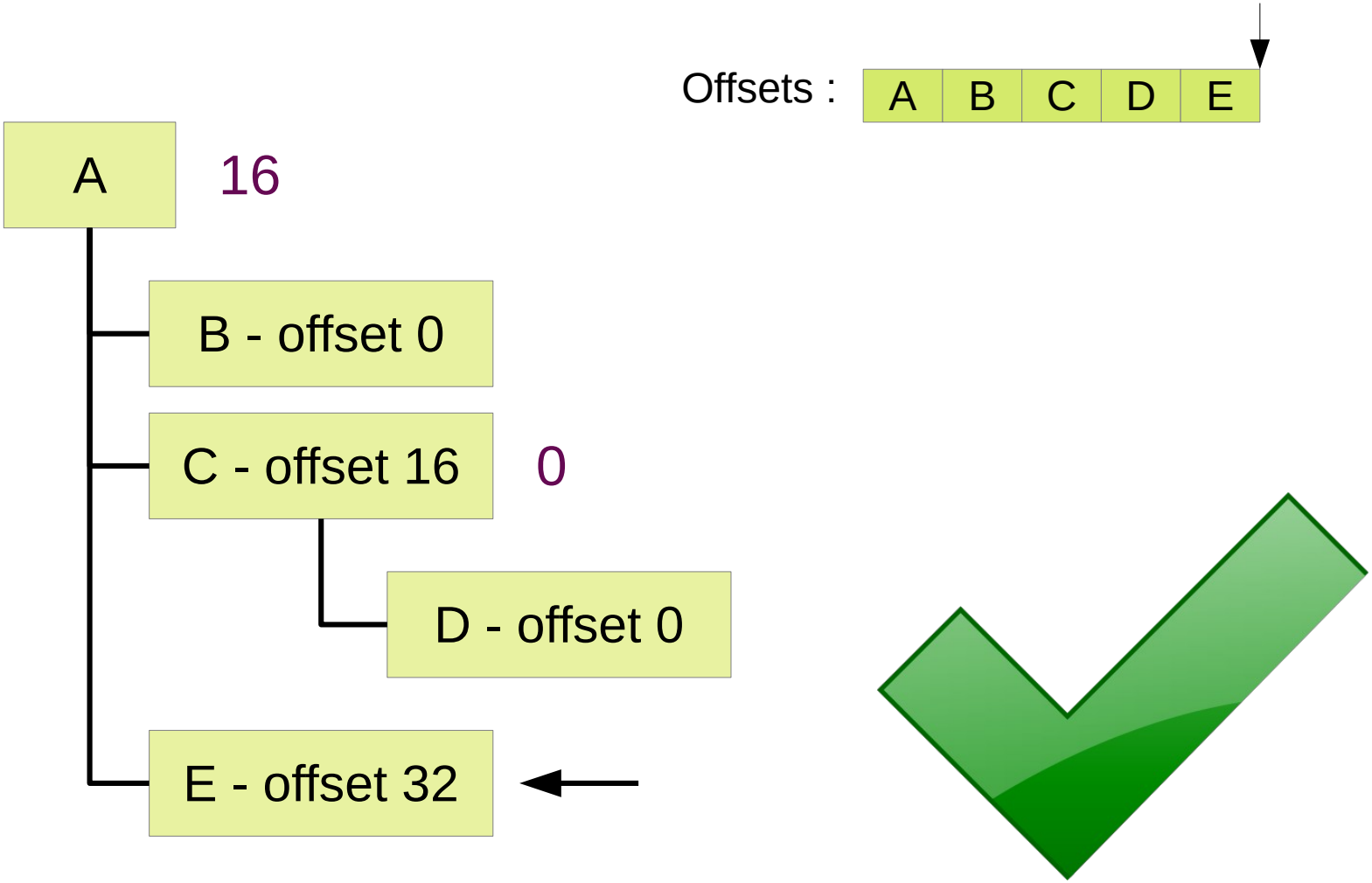
Rule example : Valid



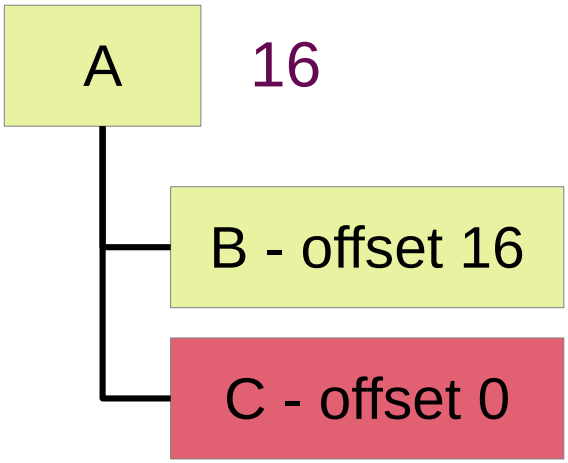
Rule example : Valid



Rule example : Valid

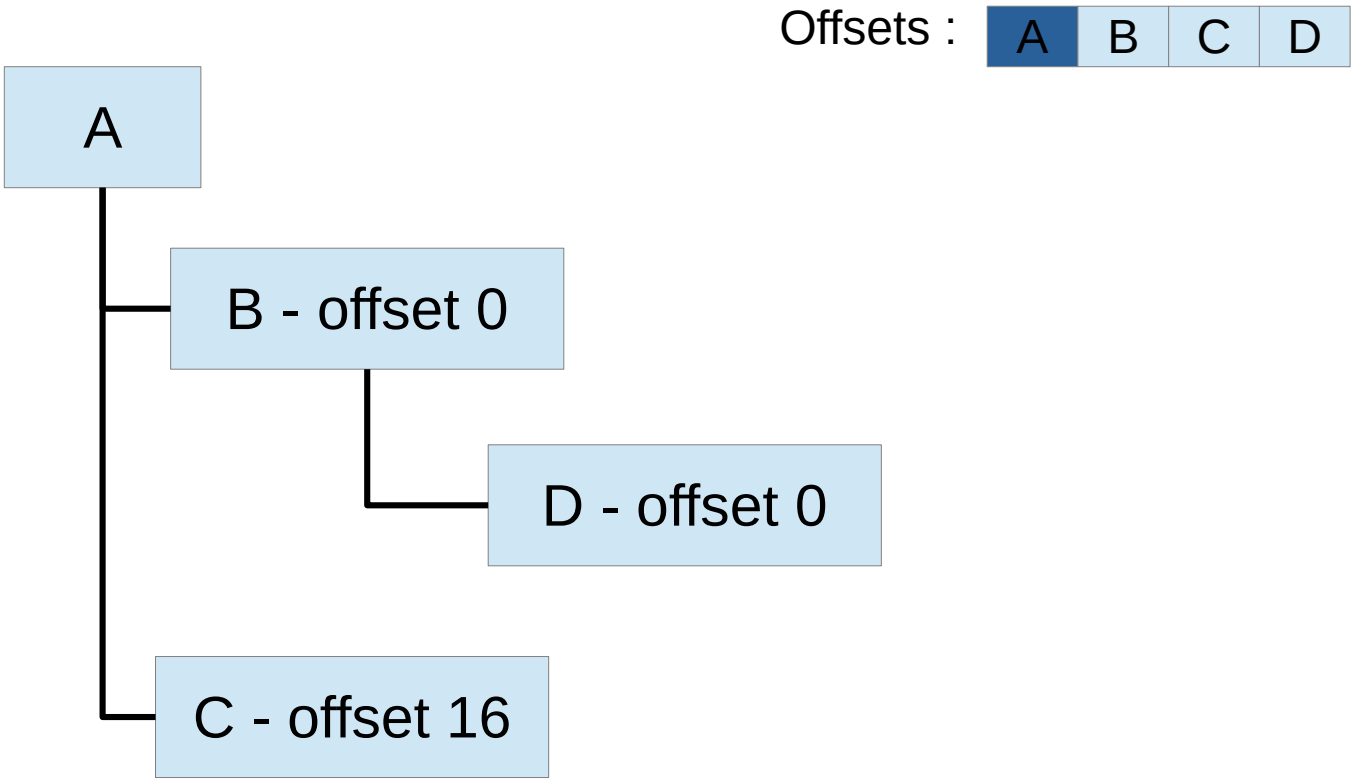


Rule example : Invalid

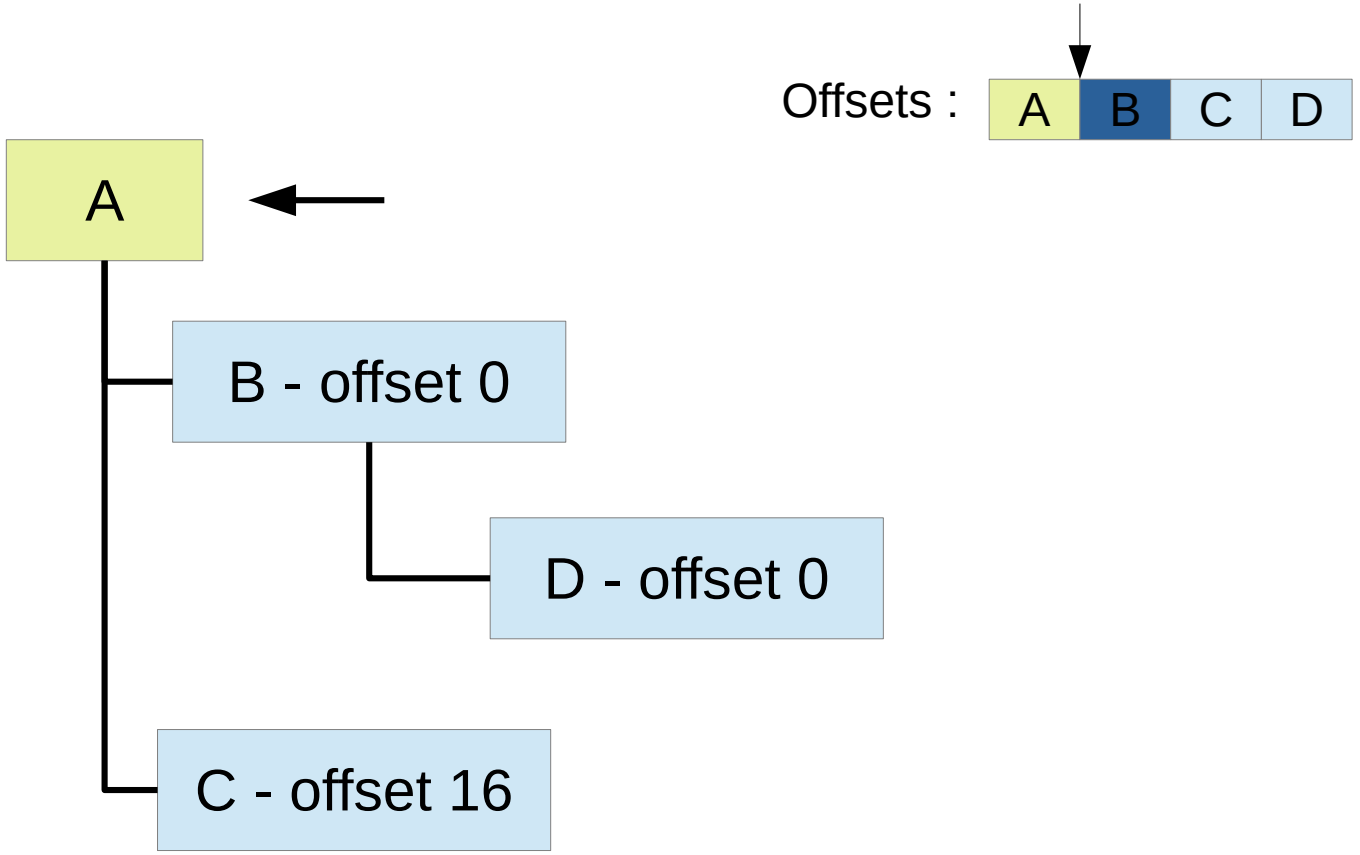


- Rule : We only allow fixups inside a buffer to happen at increasing offsets

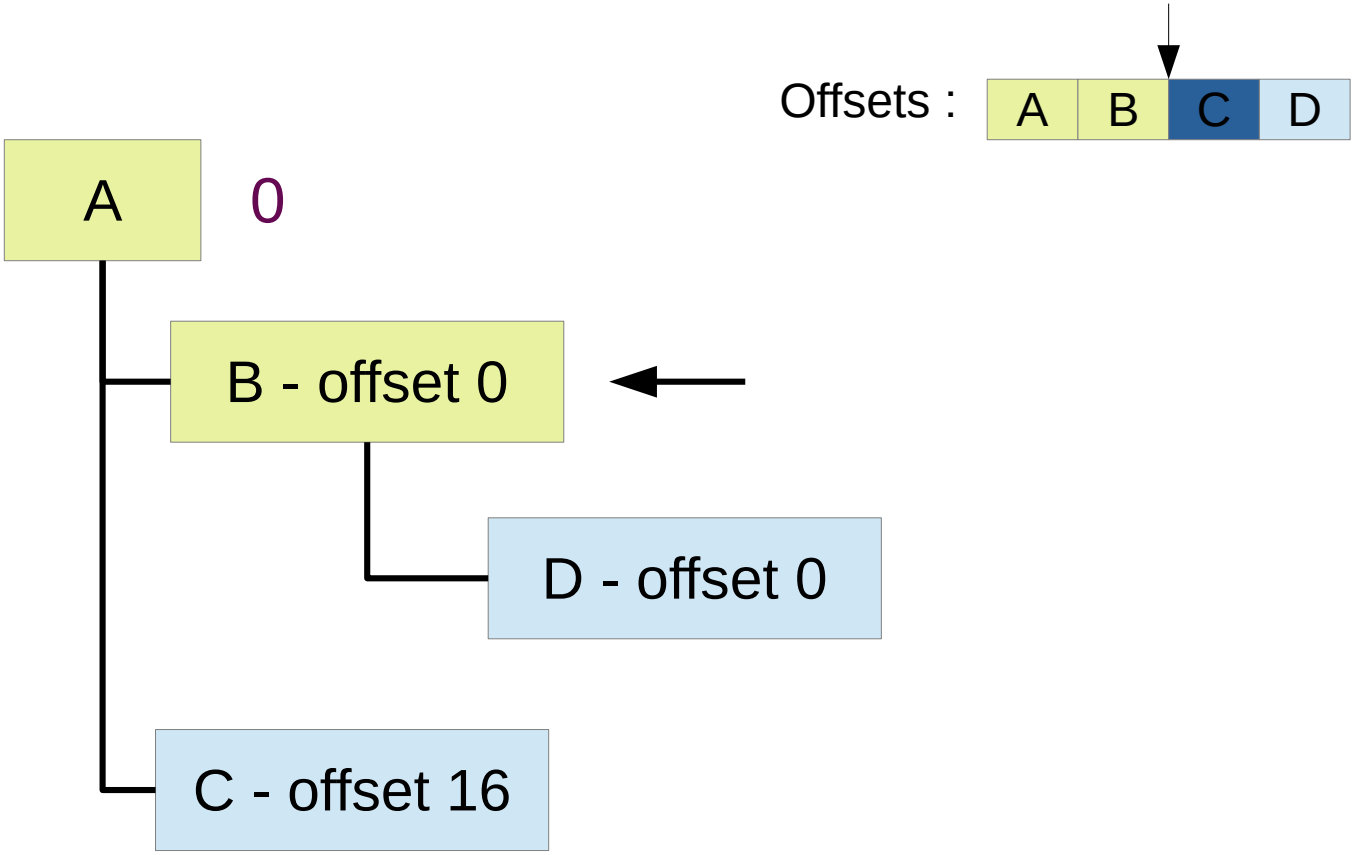
Rule example : Invalid



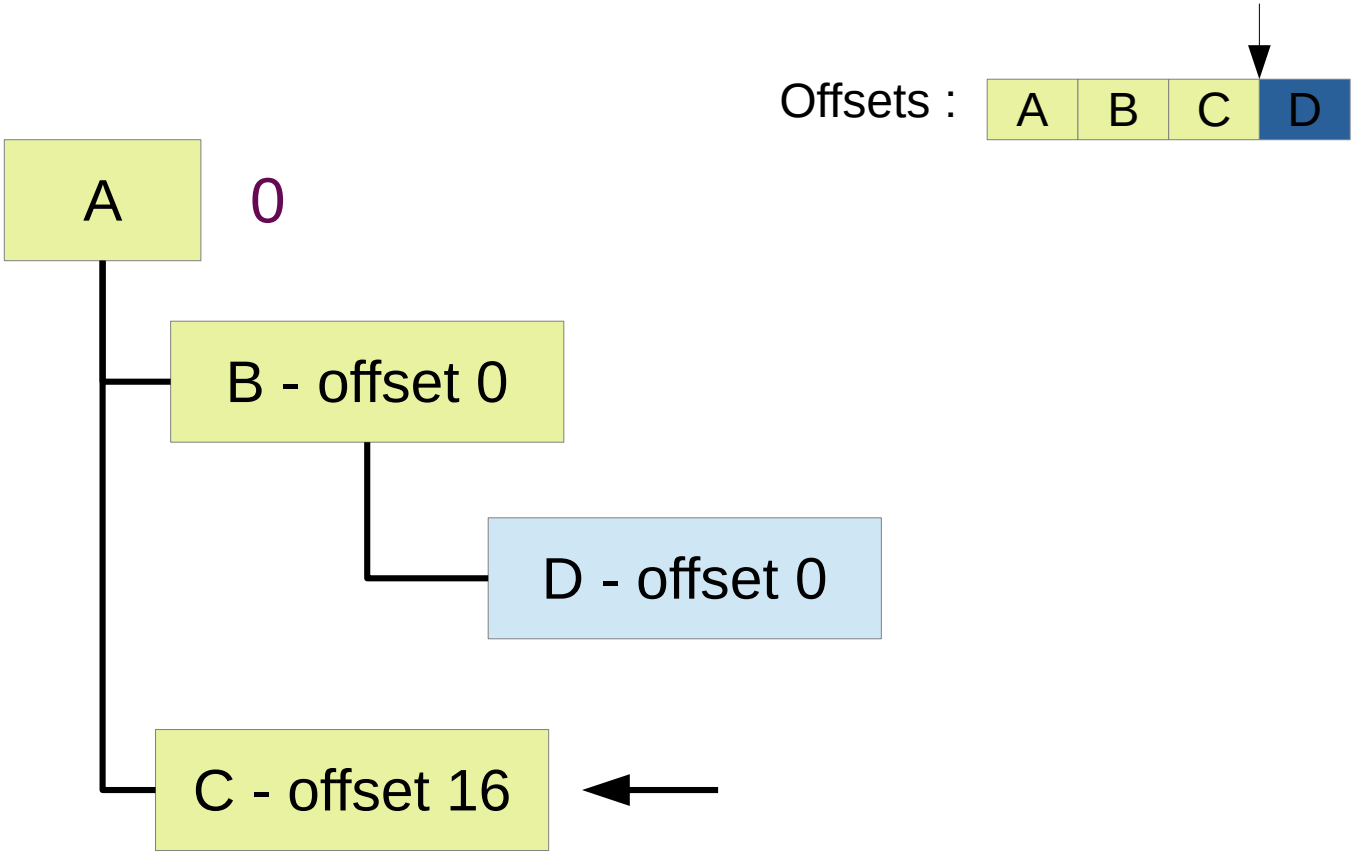
Rule example : Invalid



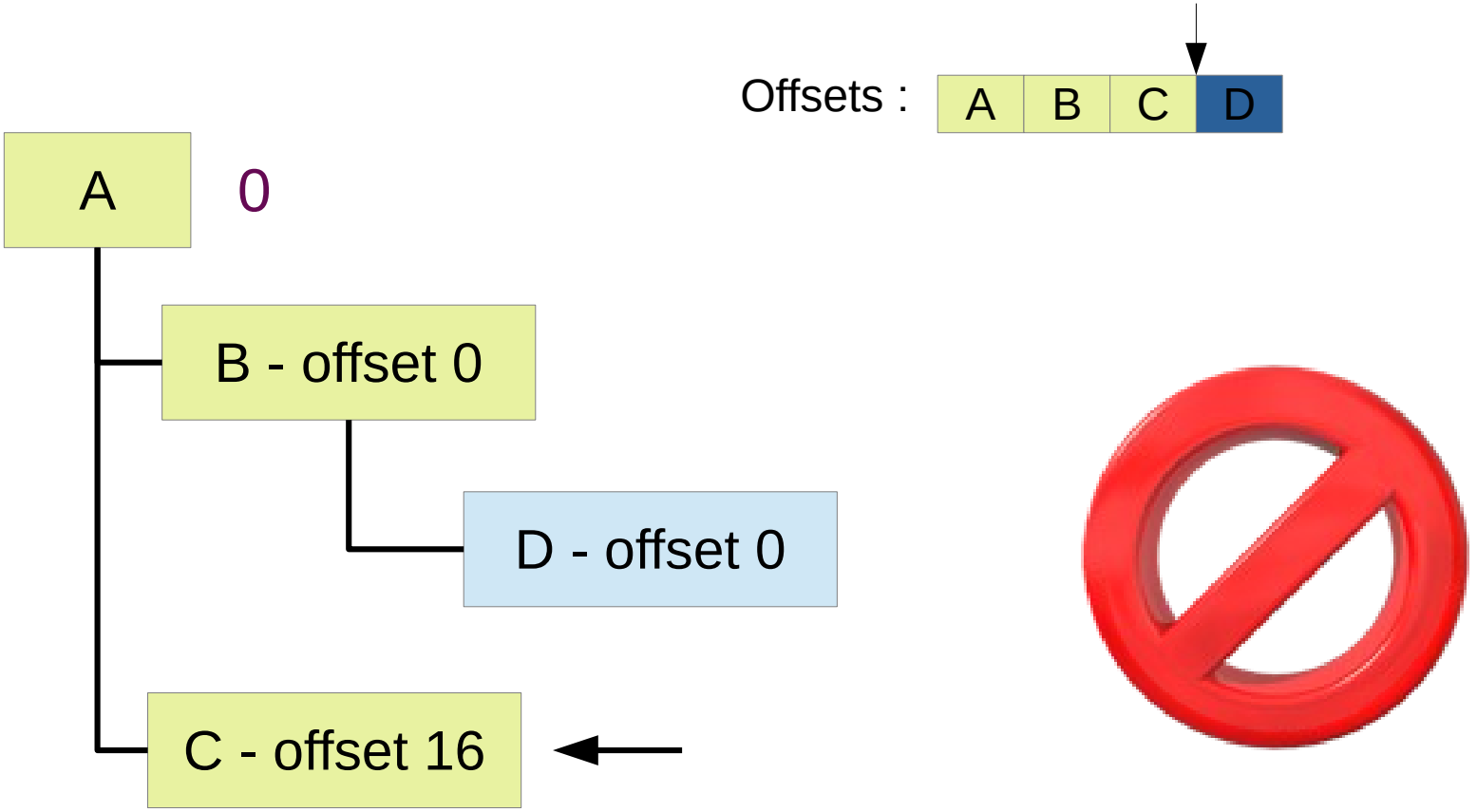
Rule example : Invalid



Rule example : Invalid



Rule example : Invalid

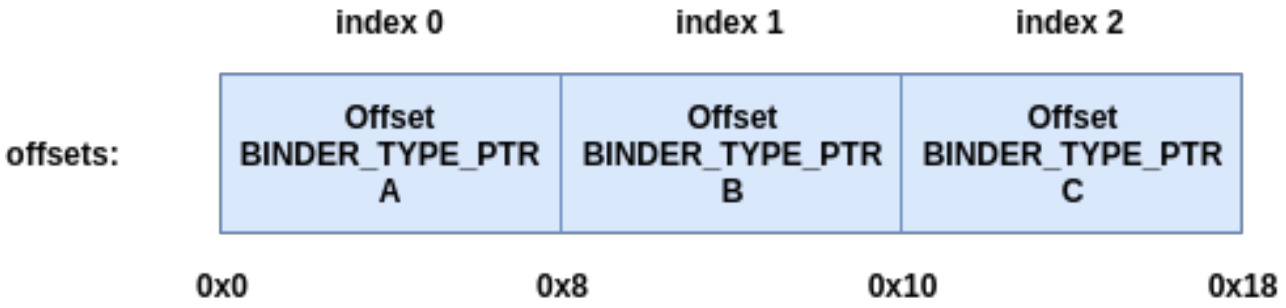


- Only allow fixup on the last buffer object that was verified, or one of its parents

What is the bug ?

- **Confusion between index in a table and offsets**

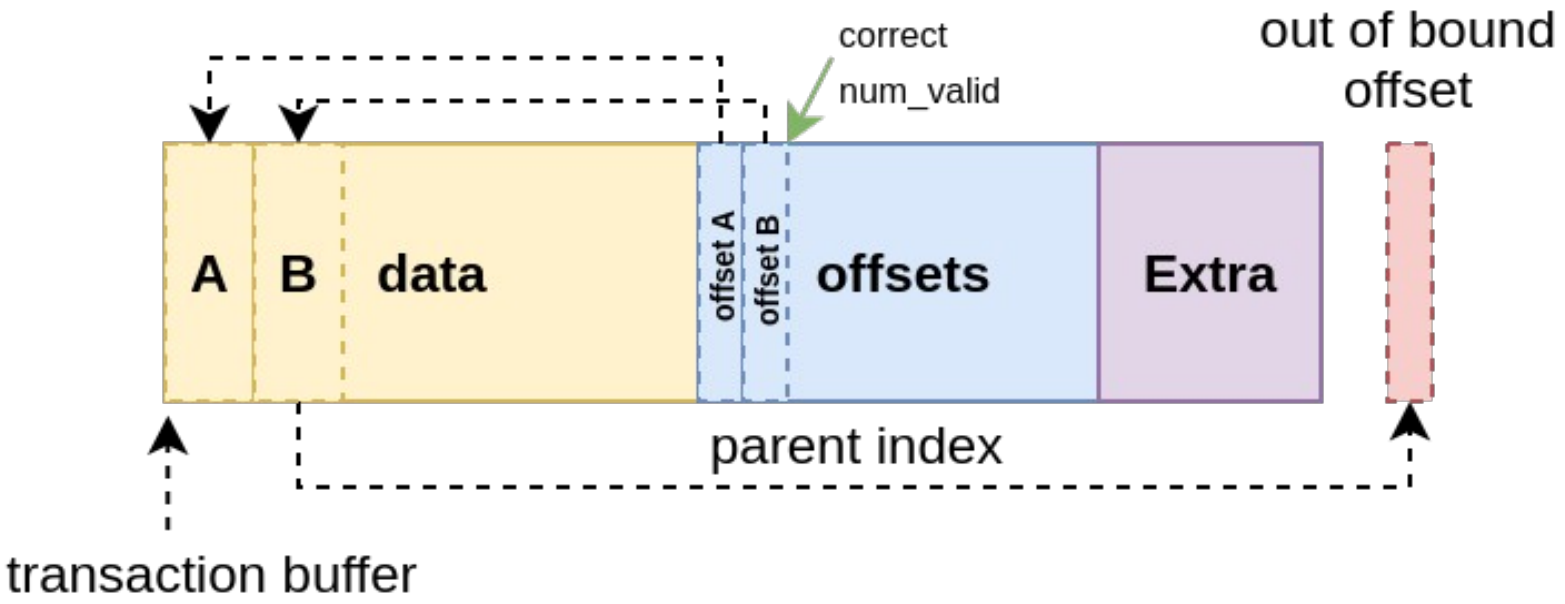
```
//vulnerable code
size_t num_valid = (buffer_offset - off_start_offset) * sizeof(binder_size_t);
```



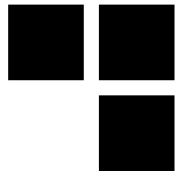
- **If current offset is 0x10**
  - Wanted num_valid = 0x10/8 = 2
  - Buggy code, num_valid = 0x10 * 8 = **0x80** !

What is the impact ?

- **An object can have an unverified parent offset**



Exploitation Idea

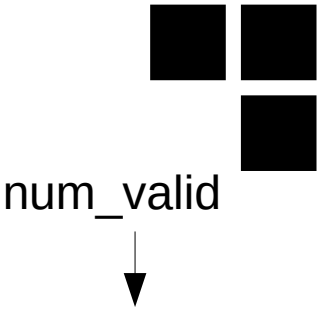


- **Objective :**
  - Bypass *binder_validate_fixup* validation

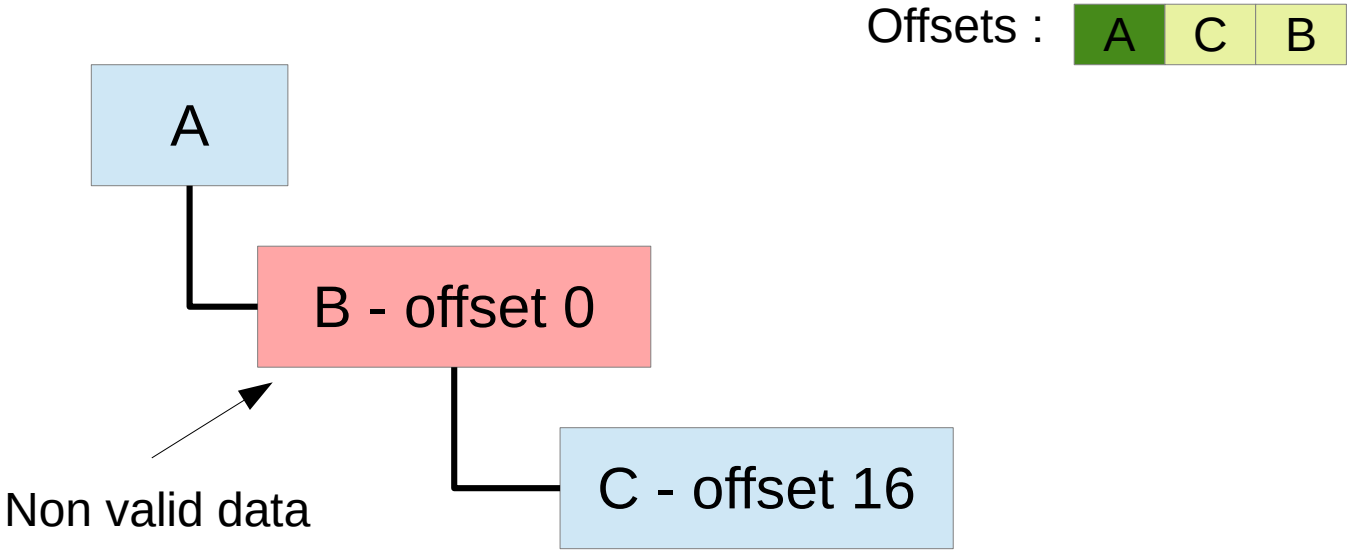
```
/* binder_validate_fixup comments :
* For safety reasons, we only allow fixups inside a buffer to happen
 * at increasing offsets; additionally, we only allow fixup on the last
 * buffer object that was verified, or one of its parents.
*/
```

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

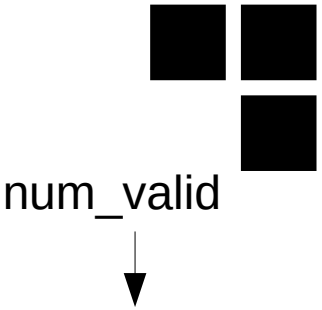
Naive try



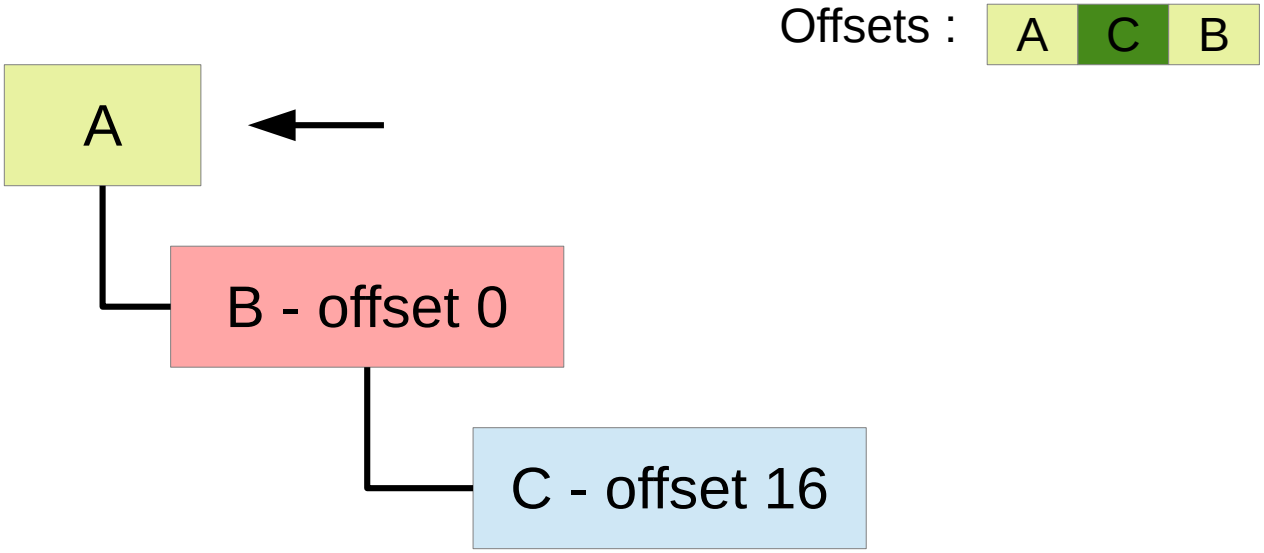
num_valid



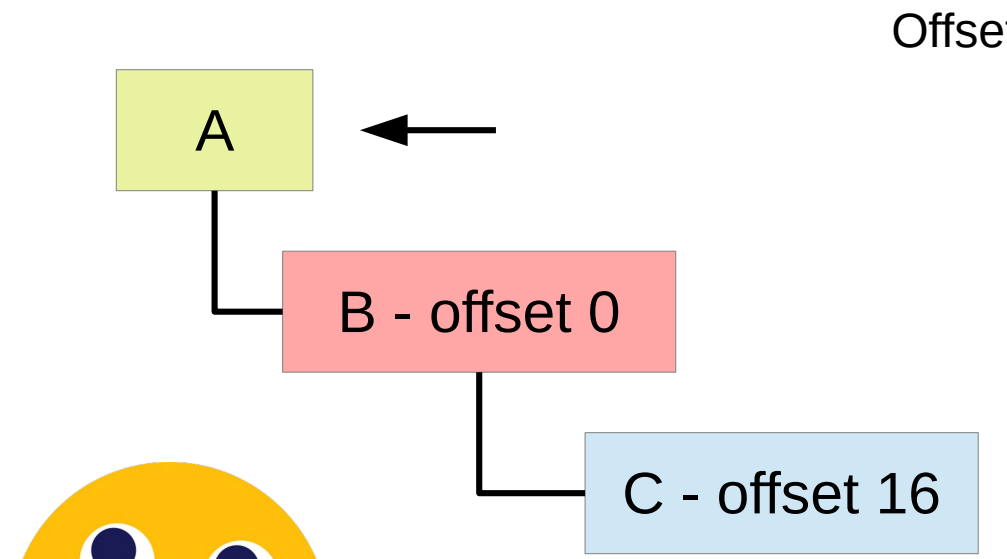
Naive try



num_valid



Naive try

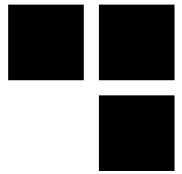


num_valid

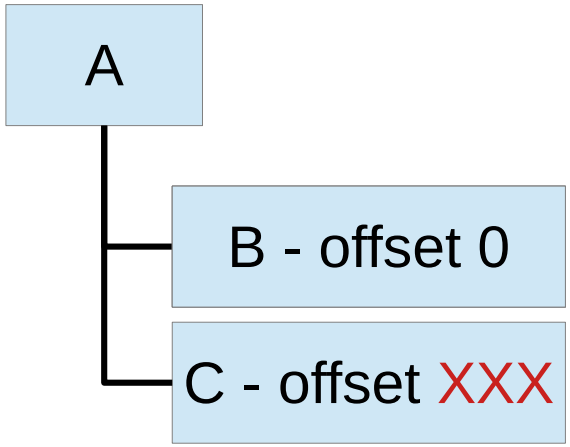
Offsets : A C B

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

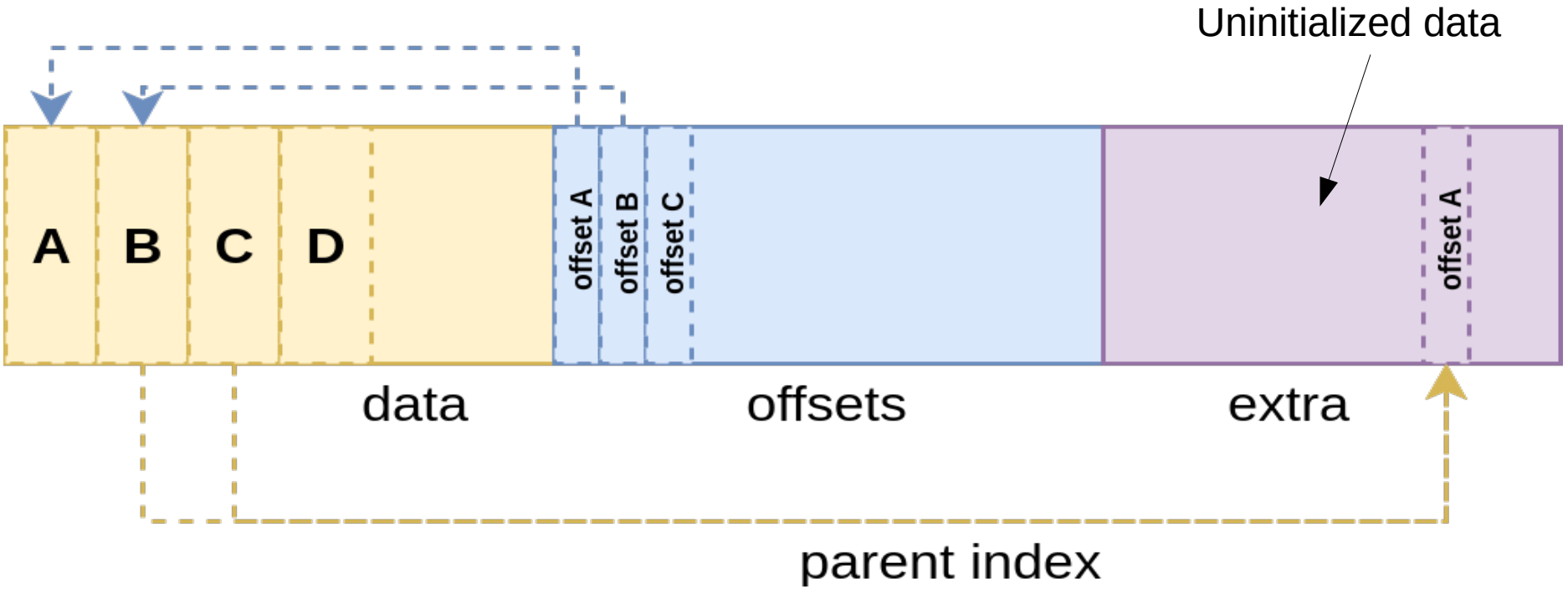
Solution

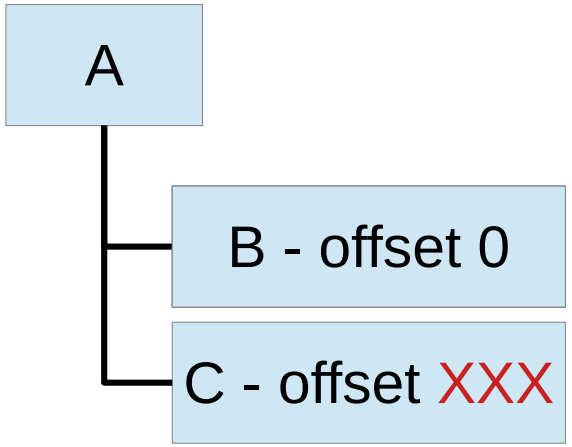


- **Change a parent during the validation !**
- **Using the extra buffer !**
  - Use a parent index which is in extra part
  - Each time a BINDER_TYPE_PTR is valid, its buffer is copied in extra part !

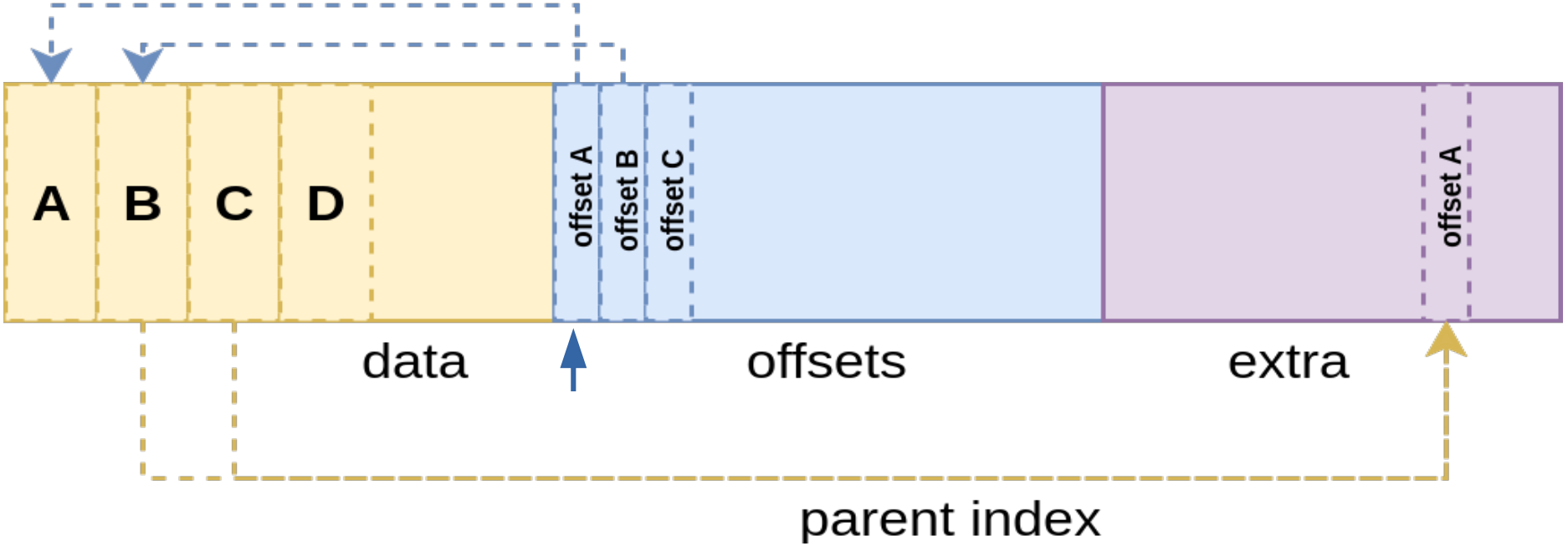


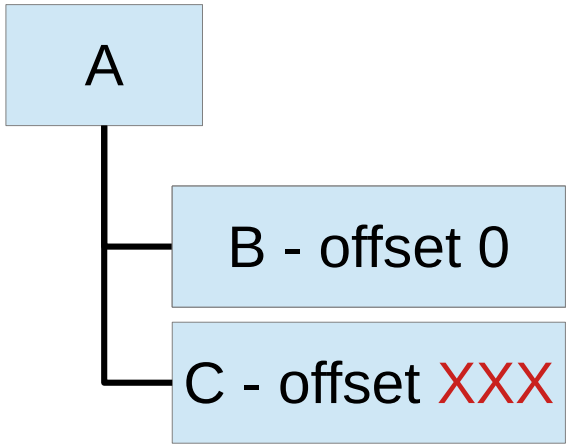
D
buffer = XXXX



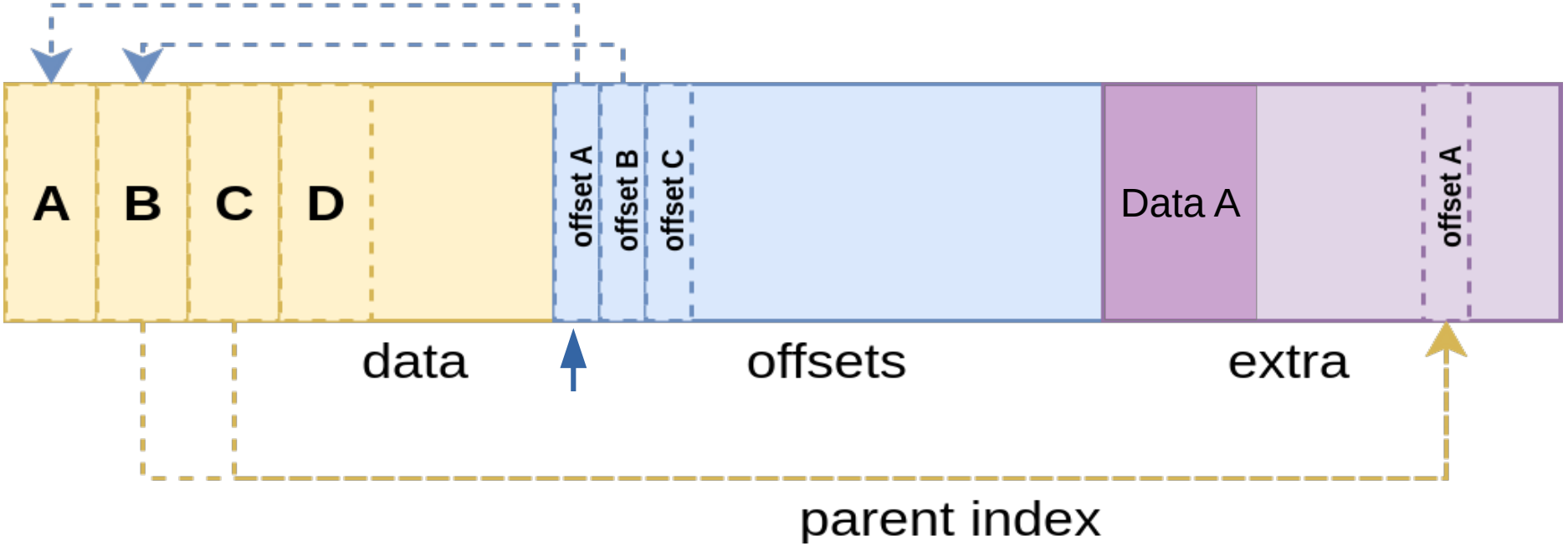


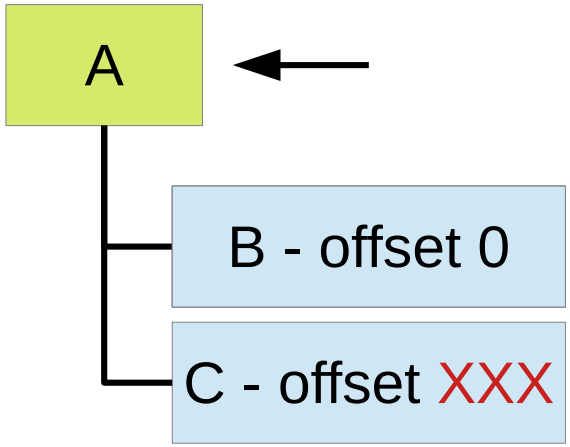
D
buffer = XXXX



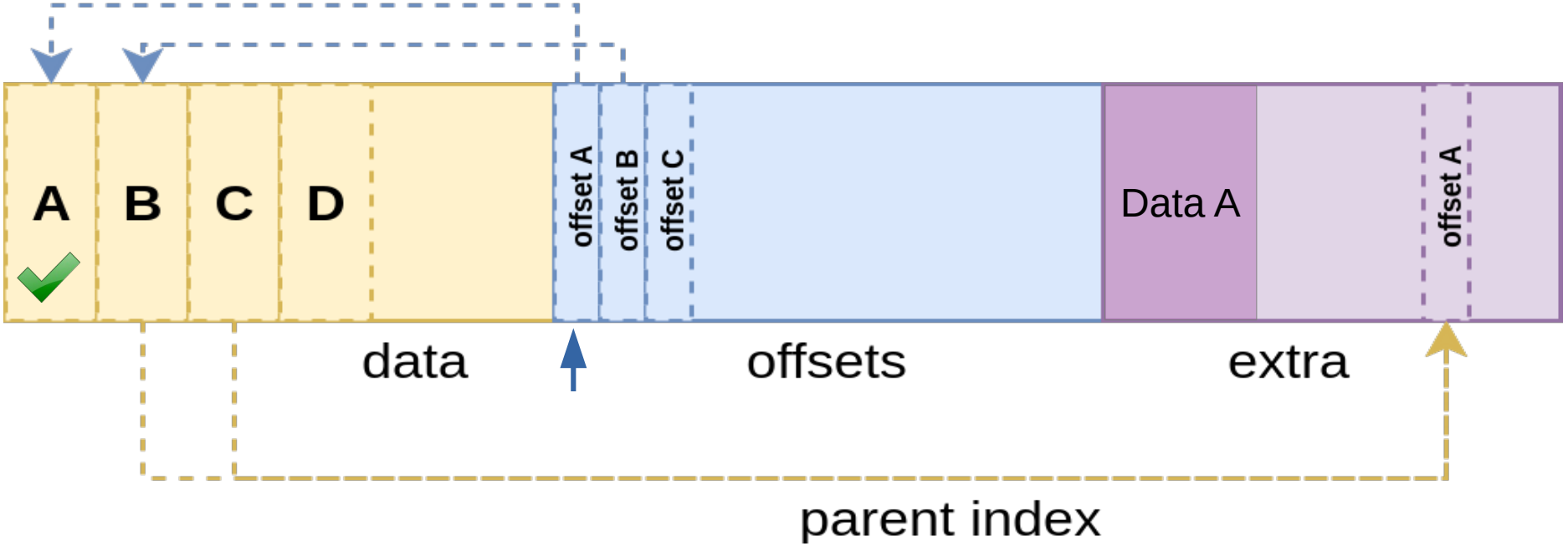


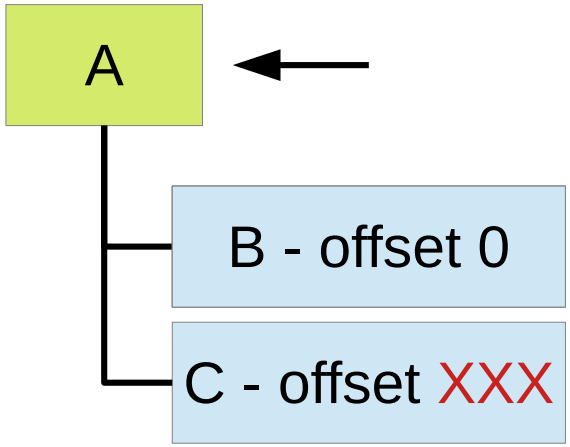
D
buffer = XXXX



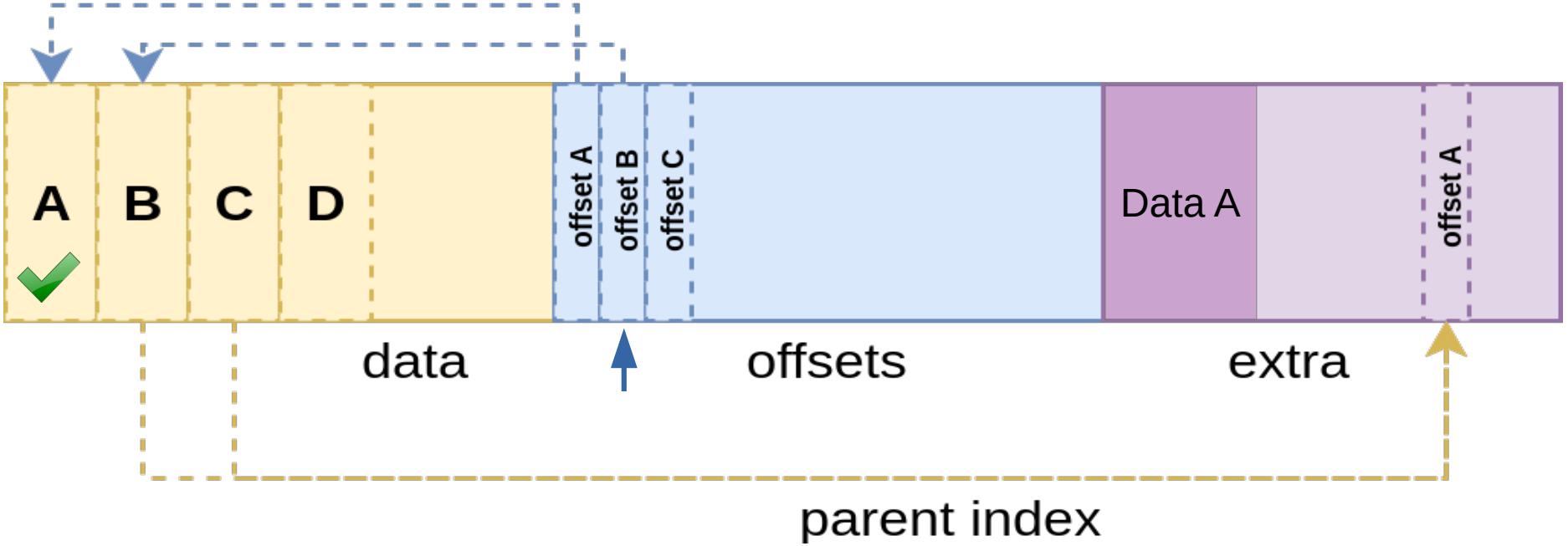


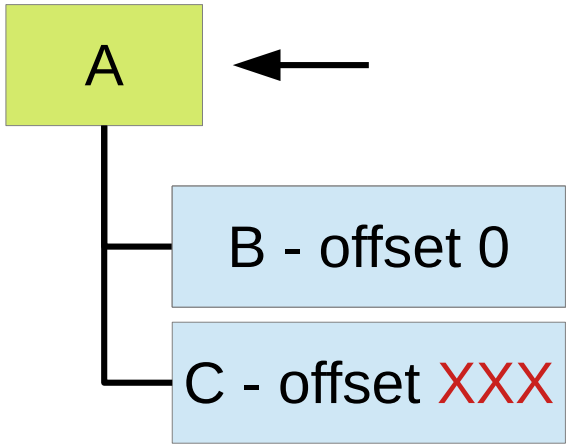
D
buffer = XXXX



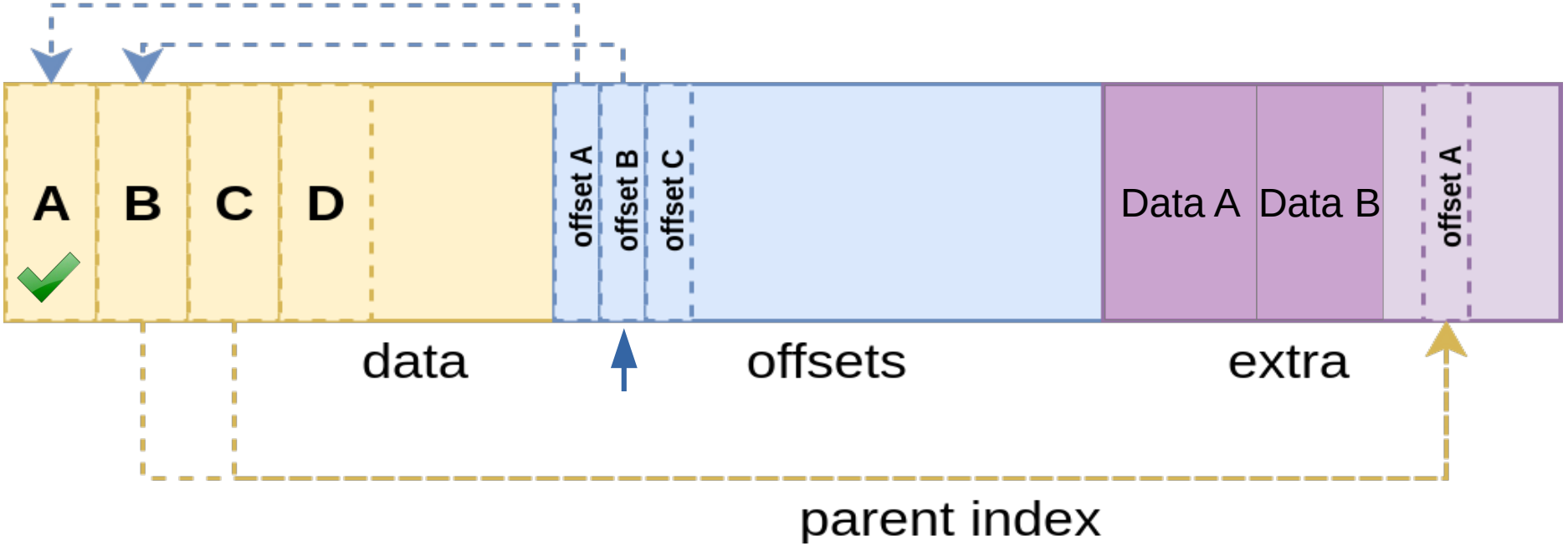


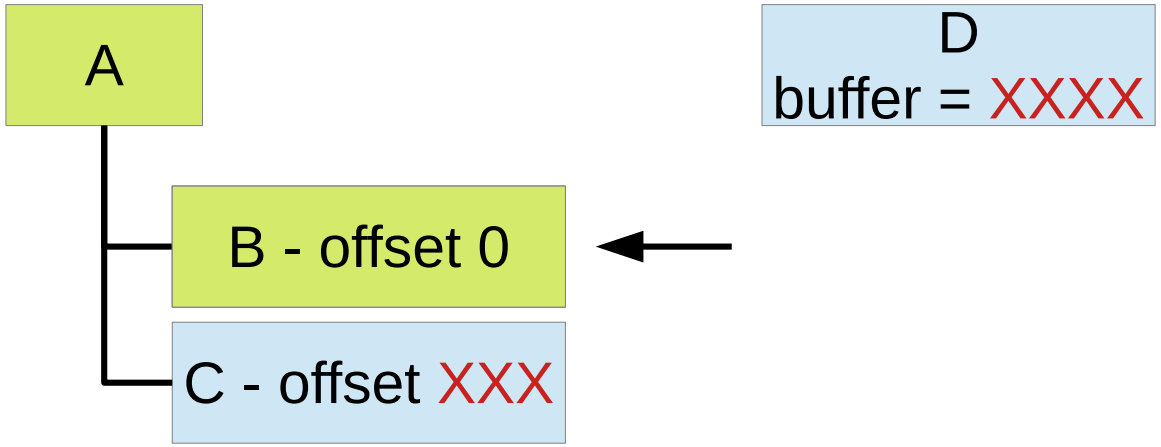
D
buffer = XXXX

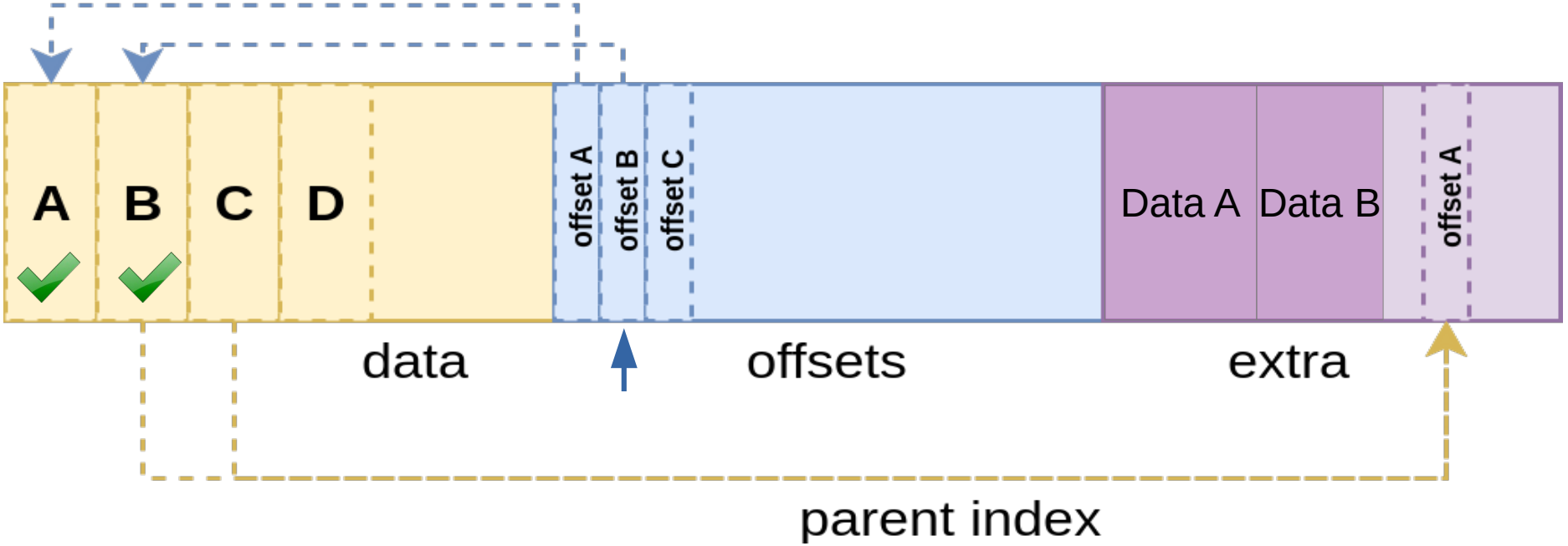


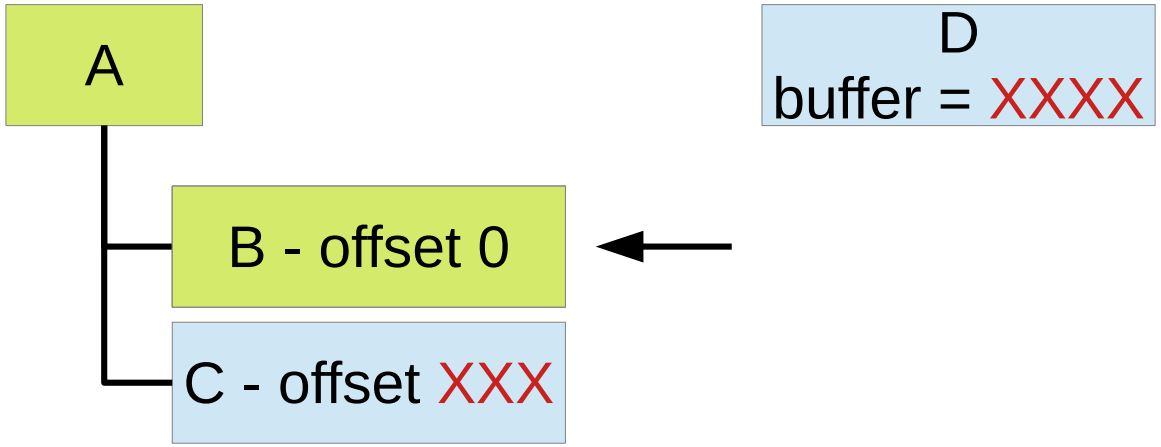


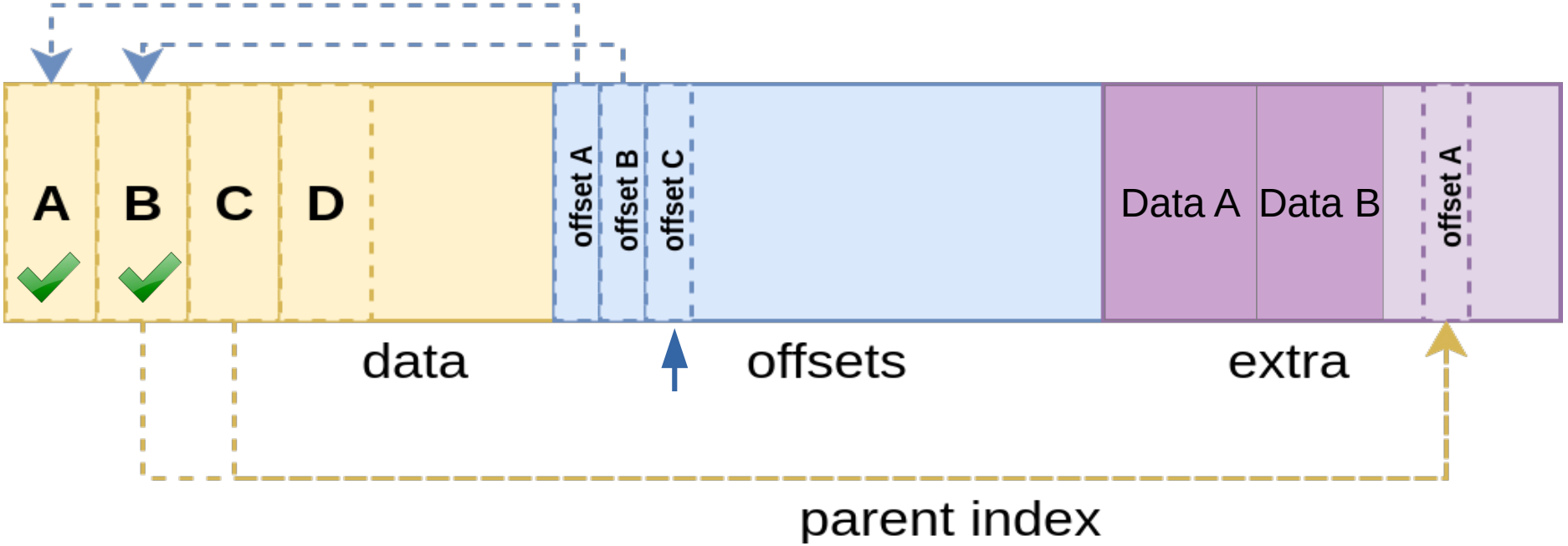
D
buffer = XXXX

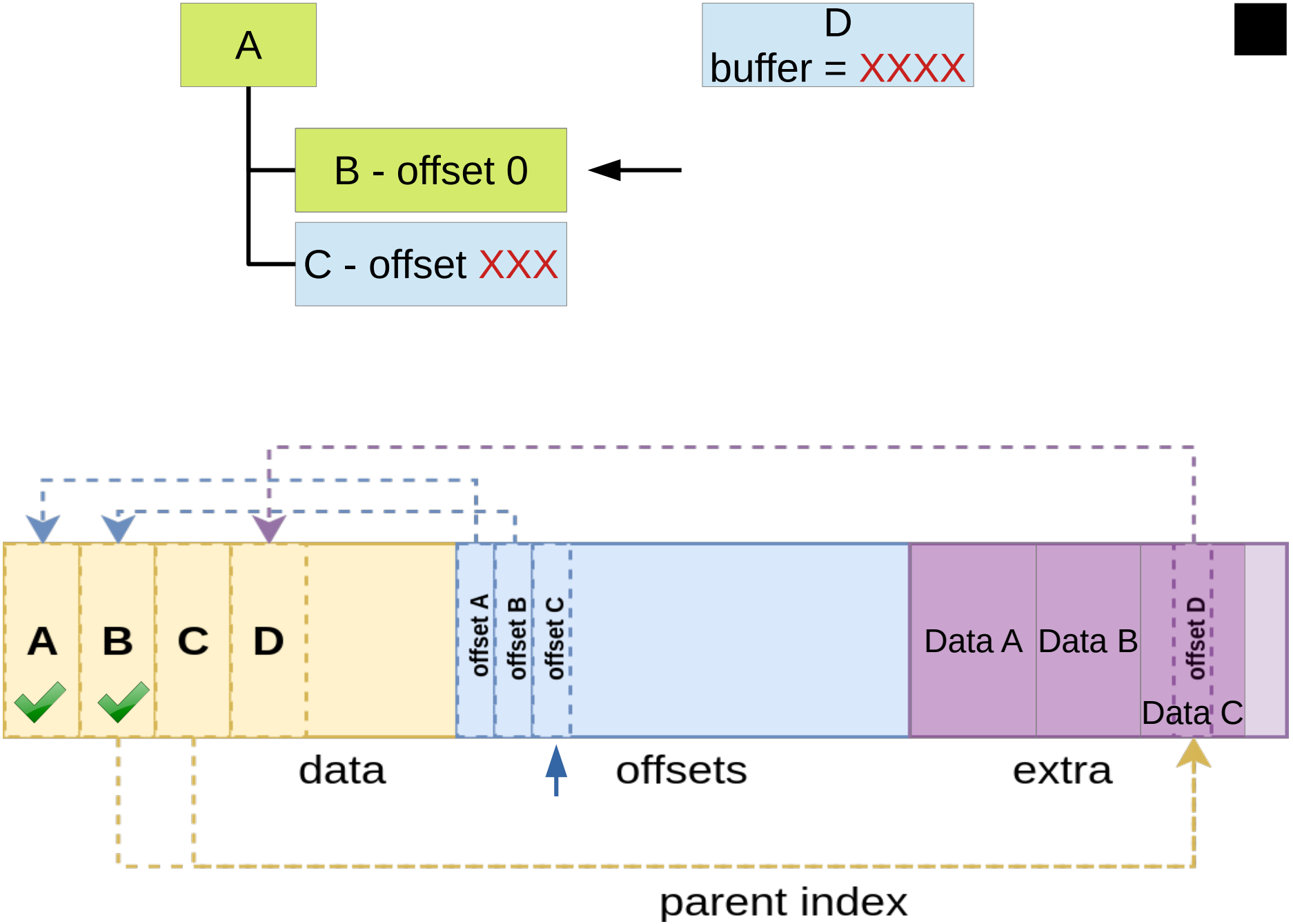


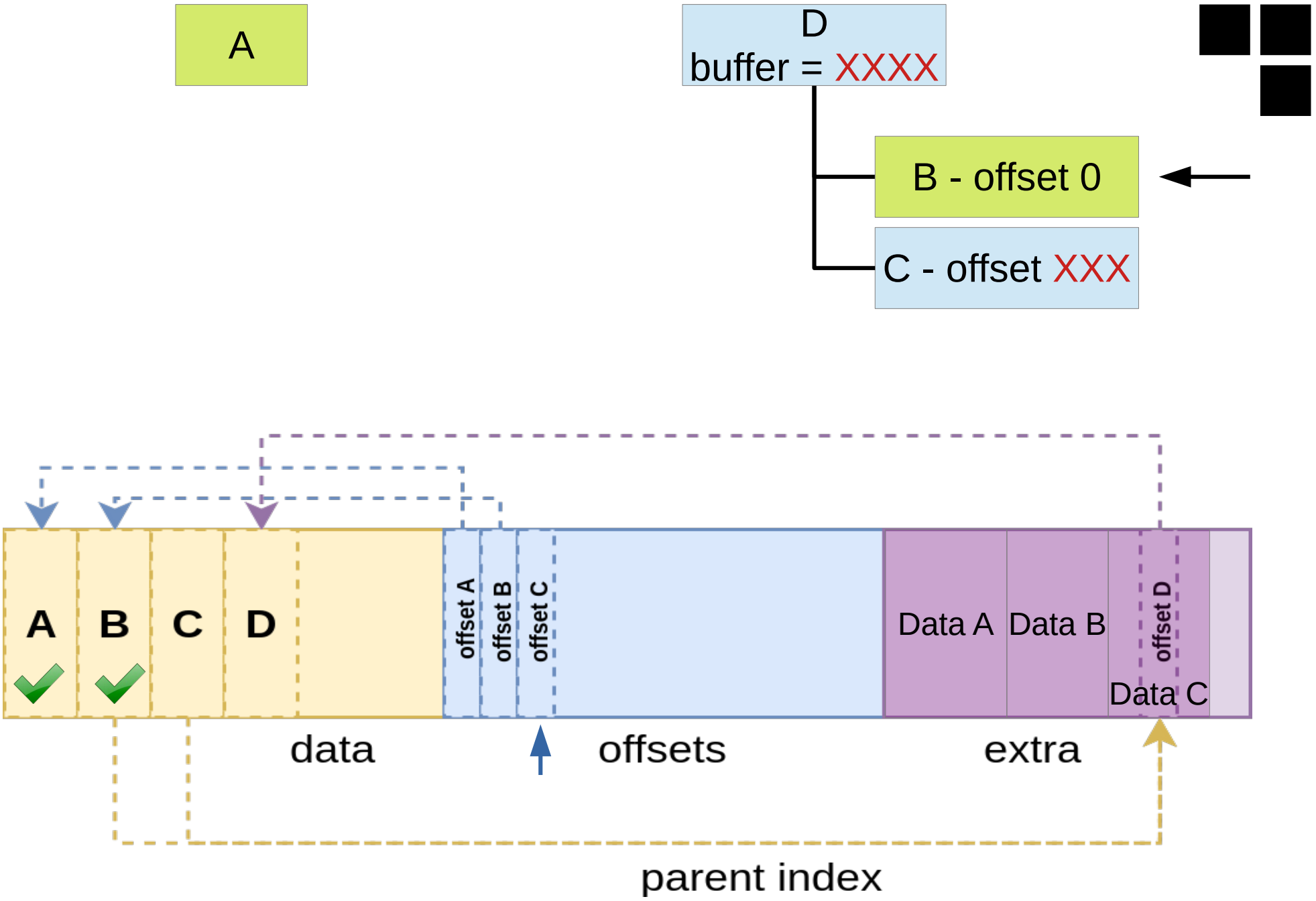

A
B - offset 0
C - offset XXX
D
buffer = XXXX

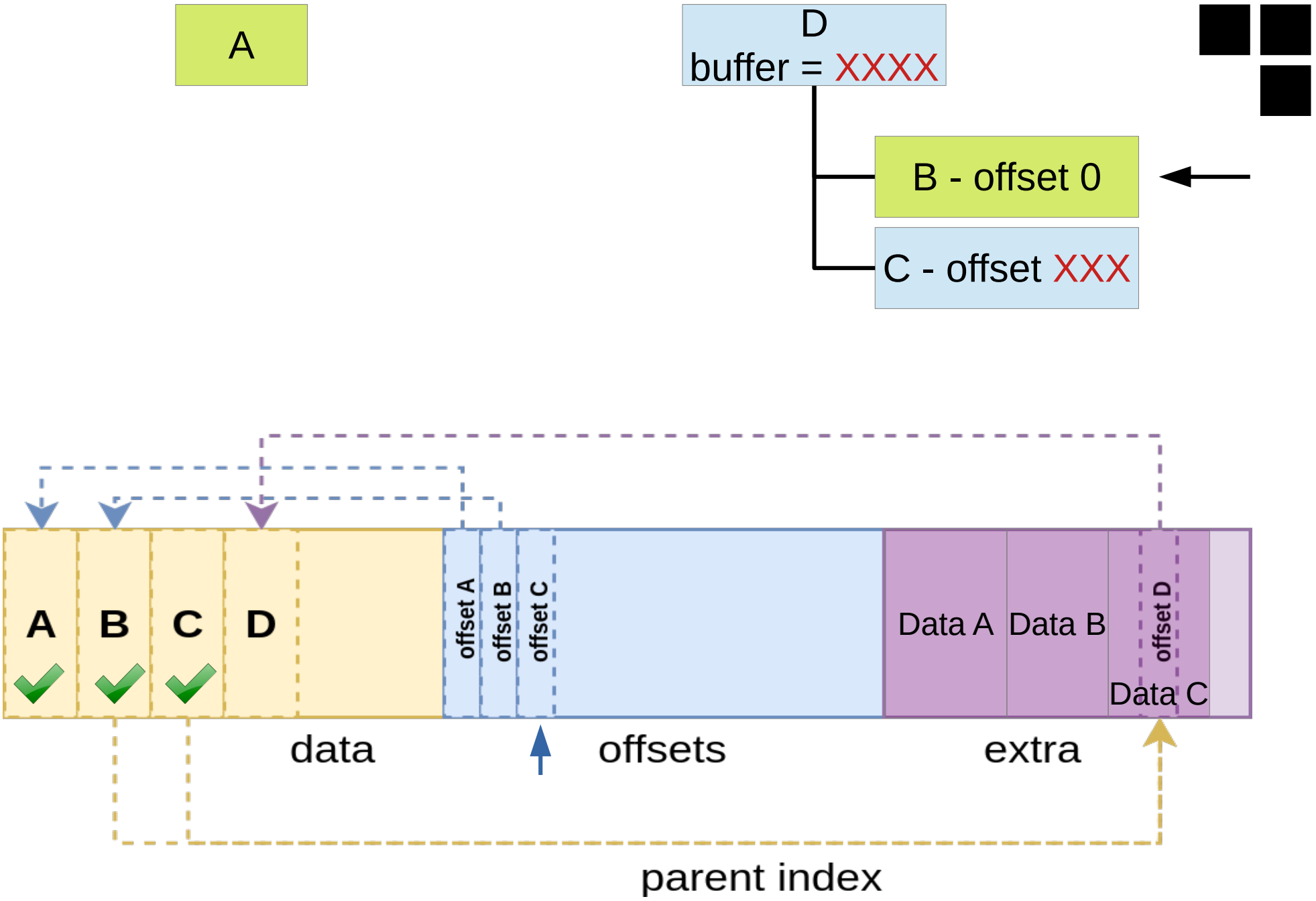

A
B
C
D
offset A
offset B
offset C
Data A
Data B
offset A
data
offsets
extra
parent index


A
B - offset 0
C - offset XXX
D
buffer = XXXX


A
B
C
D
offset A
offset B
offset C
Data A
Data B
offset A
data
offsets
extra
parent index


A
B - offset 0
C - offset XXX
D
buffer = XXXX
A
B
C
D
offset A
offset B
offset C
Data A
Data B
offset D
Data C
data
offsets
extra
parent index


A
D
buffer = XXXX
B - offset 0
C - offset XXX
A
B
C
D
offset A
offset B
offset C
Data A
Data B
offset D
Data C
data
offsets
extra
parent index

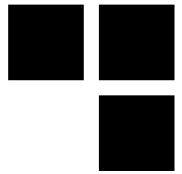

A
D
buffer = XXXX
B - offset 0
C - offset XXX
A
B
C
D
offset A
offset B
offset C
Data A
Data B
offset D
Data C
data
offsets
extra
parent index

Patch buffer

```
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;
}
```

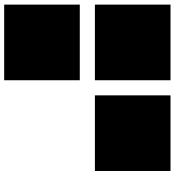
- **Value controlled :**
  - parent → buffer
  - bp → parent_offset
- **Value writing : pointer to C buffer (controlled) in extra data**
  - **alloc_buffer + buffer_offset =** @(C buffer)

Exploit Limitations



- ***binder_alloc_copy_to_buffer* checks if *buffer* + *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**

  ./emulator -avd Pixel_3a_XL_API_29_64b -kernel custom_bzImage -show-kernel -no-window -verbose -ranchu -no-snapshot

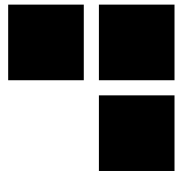
- **Build custom kernel to add debug log**

```
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

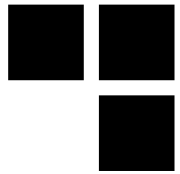
```
[  148.291702] binder: 3410:3410 ioctl c0306201 7fff98cb5f20 returned -22
[  148.295022] binder_alloc: [JB] check_buffer buffer_size : 0x10e0 bytes =  0x8
offset = 0x71829fdc8b8
[  148.299460] ------------[ cut here ]------------
[  148.301159] kernel BUG at drivers/android/binder_alloc.c:1133!
[  148.303042] invalid opcode: 0000 [#1] PREEMPT SMP NOPTI
[  148.304537] Modules linked in:
[  148.305422] CPU: 0 PID: 3410 Comm: poc Not tainted 4.14.150HELLO+ #28
[  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: 0000000000a1c204
[  148.313730] RIP: 0010:binder_alloc_do_buffer_copy+0x8d/0x15e
[  148.315692] RSP: 0018:ffffa11501effa48 EFLAGS: 00010246
[  148.317540] RAX: 0000000000000000 RBX: ffff9e98a62079c0 RCX: 0000000000000008
[  148.320403] RDX: ffff9e98aa0e5dd8 RSI: 0000000000000000 RDI: ffff9e98aa0e5da0
[  148.323268] RBP: ffffa11501effaa0 R08: 00000000000000f4 R09: 0000000000000000
[  148.325435] R10: 0000000000000000 R11: 0000000000000000 R12: 0000000000000008
[  148.328290] R13: 0000071829fdc8b8 R14: ffff9e98aa0e5da0 R15: ffff9e98a62079c0
[  148.330194] FS:  000000000048d648(0000) GS:ffff9e98bfc00000(0000)
knlGS:0000000000000000
[  148.331780] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[  148.332740] CR2: 0007435311239a0 CR3: 0000000010ee2000 CR4: 00000000000006b0
[  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



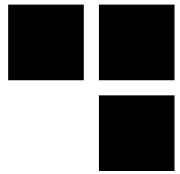
- **We can overwrite verified data in a binder transaction**
- **Overwrite existing objects :**
  - File descriptors
  - Binder reference => to a controlled object
  - Structures (like hild_string)

    Change the address

    Change the size

```
struct hidl_string {
    details::hidl_pointer<const char> mBuffer;
    uint32_t mSize;
    bool mOwnsBuffer;
};
```

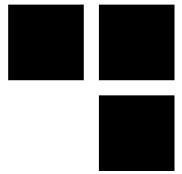
Vulnerable devices



- **Need a recent kernel**

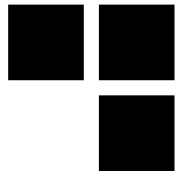
  commit bde4a19fc04f5 - Feb 8, 2019
- **Pixel 4 – msm-coral-4.14-android10**
- **Pixel 3/3a XL – msm-bonito-4.9-android10**
- **Fixed with the update of March 2020**

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



- **http://newandroidbook.com/files/Andevcon-Binder.pdf**
- **https://blog.zimperium.com/cve-2018-9411-new-critical-vulnerability-multiple-high-privileged-android-services/**
- **https://conference.hitb.org/hitbsecconf2019ams/materials/D2T2%20-%20Binder%20-%20The%20Bridge%20to%20Root%20-%20Hongli%20Han%20&%20Mingjian%20Zhou.pdf**
- **https://googleprojectzero.blogspot.com/2019/11/bad-binder-android-in-wild-exploit.html**
- **https://www.synacktiv.com/posts/systems/binder-transactions-in-the-bowels-of-the-linux-kernel.html**
- **https://www.synacktiv.com/posts/systems/binder-secctx-patch-analysis.html**

AVEZ-VOUS DES QUESTIONS ?

MERCI DE VOTRE ATTENTION,

SYNACKTIV
DIGITAL SECURITY