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

Hardware-Software Communication Support

Resource Management (CPU/Memory/Disk)

Userland

User Permission Separation

System information & Resource Protection

Hardware

2022 Vulnerabilities exploit by Zero-day attack (Google TAG)

~50% Vulns are OS vuln
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...

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 users), 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)
- User namespace awareness
  - For LTS, +$20,000 for not using userns
  - For COS: +$10,500 for not using userns
- Novel techniques
  - Irrelevant with vulnerabilities
  - $0 ~ $20,000 payout

$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

2014-05-19
Vuln introduced

2023-06-08
Vuln patch commit on net-next & mainline

2023-06-20~21
Exploit crafted for v6.1.31 & COS-105

2023-06-21
Patch backported to LTS (v6.1.35 and v5.15.118)

2023-06-28
CVE assigned

2023-07-27
Exploit published

2 Week patch gap
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
The Vulnerability - CVE-2023-3390

- Operations handled in batch (transaction)

```
batch start
  nft_trans
    NFT_MSG_NEWTABLE
      - protocol: INET
      - table name: "filter"
  nft_trans
    NFT_MSG_NEWCHAIN
      - protocol: INET
      - table name: "filter"
      - chain name: "chain1"
  nft_trans
    NFT_MSG_NEWRULE
      - protocol: INET
      - table name: "filter"
      - chain name: "chain1"
      - expr: lookup expression...
batch end
```

sendmsg syscall
The Vulnerability - CVE-2023-3390

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

```c
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 c1092c3549d ("netfilter\nf_tables\ndeactivate 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 convit 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
diff --git a/net/netfilter/nf_tables_api.c b/net/netfilter/nf_tables_api.c
index 96b0800d..3b68fde5 100644
--- a/net/netfilter/nf_tables_api.c
+++ b/net/netfilter/nf_tables_api.c
@@ -354,7 +354,8 @@ err_destroy_flow_rule:
     if (!ctx)
         nft_flow_rule.destroy(tlflow);
     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

```c
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 (set->use == 0)
            nft_deactivate_next(ctx, net, set);
        if (set->use == 1)
            return;
    case NFT_TRANS_ABORT:
        case NFT_TRANS_RELEASE:
            set->use = 0;
            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);
}
```
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:
 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_unbind_set(const struct nft_ctx *ctx, struct nft_set *set,
     struct nft_set_binding *binding, bool event)
 {
     list_del_rcu(binding->list);
     (__obj__)->genmask = nft_genmask_next(__net)

The Vulnerability - CVE-2023-3390

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);
        else
            nft_set_delete(set);
        return;
    case NFT_TRANS_ABORT:
        nft_set_release(set); 
        return;
    case NFT_TRANS_COMMIT:
        default:
            nf_tables_unbind_set(ctx, set, binding, 
                phase == NFT_TRANS_COMMIT);
            return;
    }
}

#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

Vuln path - Unbind the set
The Vulnerability - CVE-2023-3390

```
diff --git a/net/netfilter/nf_tables_api.c b/net/netfilter/nf_tables_api.c
index 3bb00000b3943a...635beceaa5c00 100644
--- a/net/netfilter/nf_tables_api.c
+++ b/net/netfilter/nf_tables_api.c
@@ -3044,7 +3044,8 @@ err_destroy_flow_rule;
     if (flow)
         nf_flow_rule_destroy(flow);
     err_release_rule:
-    nf_tables_rule_release(&ctx, rule);
+    nf_tables_rule_deactivate(&ctx, rule, NFT_TRANS_PREPARE);
     nf_tables_rule_destroy(&ctx, rule);
     err_release_expr:
         for (i = 0; i < n; i++) {
```

```
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_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);
}
```

Set destroyed and freed due to unbinding
The Vulnerability - CVE-2023-3390

Freed set still accessible after improper destroy
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

```
batch start
  nft_trans
    NFT_MSG_NEWSET
    - Allocate set object
  nft_trans
    NFT_MSG_NEWRULE
    - Add rule with set
    - Trigger error condition
  nft_trans
    NFT_MSG_NEWRULE
    - Add rule with set
batch end

sendmsg syscall
```

Commit

Abort
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

sendmsg syscall
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

```
batch start

nft_trans
NFT_MSG_NEWSSET
- Allocate set object

nft_trans
NFT_MSG_NEWRULE
- Add rule with set
- Trigger error condition

nft_trans
NFT_MSG_NEWRULE
- Add rule with set

batch end
```

```
Commit

Abort
```

```
sendmsg syscall
```
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

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

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

    freeptr_addr = (unsigned long) kasan_reset_tag((void *) freeptr_addr);
    freeptr_addr = freelist_ptr(s, fp, freeptr_addr);
}
```
The Vulnerability - CVE-2023-3390

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

1. set_A allocated
2. set_B allocated
3. set_A freed
4. set_B freed
5. set_A freed

sendmsg syscall

Transaction aborted
The Vulnerability - CVE-2023-3390

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

```c
/* 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()`
The Exploit: LTS 6.1.31 instance

- Leverage double free to *msg_msg* overlap

![Diagram showing msgqids and msgqids overlapping nodes with double free vulnerability]
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

```
msgqids1[0]
```

```
msgqid3
```

```
msgqids2[x]
```

```
msgqids2[y]
```

```
kmalloc-cg-4k
```

```
kmalloc-cg-1k
```

```
kmalloc-cg-512
```
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

```
  msgqids1[0]
  msgqid3
  msgqids2[x]
  msgqids2[y]
  ...
  kmalloc-cg-4k
  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
The Exploit: LTS 6.1.31 instance

- Place the pipe_buffer by pipe operation

```c
struct pipe_buffer {
    struct page *page;
    unsigned int offset, len;
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```
The Exploit: LTS 6.1.31 instance

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

```
struct msgqids2 {
    int32_t next;
    int32_t prev;
    int32_t m_type;
    int32_t m_ts;
    int32_t security;
}
```
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
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
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
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`

```c
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.

```
The Exploit: COS 105 instance

- **user_key_payload** struct

```c
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()

```c
int user_preparse(struct key_preparsed_payload *prep) {
    struct user_key_payload *upayload = prep->data;
    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;
}
```
The Exploit: COS 105 instance

- Leverage double free to chunk overlap
  - user_key_payload vs nft_set

```
struct user_key_payload
    rcu.head
    rcu.func
    datalen
(overwritten) user-controlled data

struct nft_set
    list.next
    list.prev
    bindings.next
    bindings.prev
    table
    net
    ops
    ...
    catchall_list.next
    catchall_list.prev

kmalloc-1k
```
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`

```
struct user_key_payload
    rcu.head       rcu.func
    datalen
(overwritten) user-controlled data

struct nft_set
    list.next   list.prev
    bindings.next bindings.prev
    table       net
    ...         ...
    ops
    ...
    catchall_list.next catchall_list.prev
```

`kmalloc-1k`
The Exploit: COS 105 instance

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

Mitigation?

Not a chance

(Exploit just worked lololololol)
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_SLABVIRTUAL

- “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

<table>
<thead>
<tr>
<th>nft_set</th>
<th>pipe_buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (ops-&gt;privsize != NULL)</td>
<td>bufs = kcalloc(nr_slots, sizeof(*bufs),</td>
</tr>
<tr>
<td>size = ops-&gt;privsize(nla, &amp;desc);</td>
<td>GFP_KERNEL_ACCOUNT</td>
</tr>
<tr>
<td>alloc_size = sizeof(*set) + size + udlens;</td>
<td></td>
</tr>
<tr>
<td>set = kvzalloc(alloc_size, GFP_KERNEL_ACCOUNT)</td>
<td></td>
</tr>
<tr>
<td>msg_msg</td>
<td>msg_msgseg</td>
</tr>
<tr>
<td>alen = min(len, DATALEN_MSG);</td>
<td>alen = min(len, DATALEN_SEG);</td>
</tr>
<tr>
<td>msg = kmalloc(sizeof(*msg) + alen, GFP_KERNEL_ACCOUNT);</td>
<td>seg = kmalloc(sizeof(*seg) + alen, GFP_KERNEL_ACCOUNT);</td>
</tr>
</tbody>
</table>
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()

```c
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

```c
def 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 */
#if CONFIG_SLUB_CPU_PARTIAL
    struct page *partial; /* Partially allocated frozen slabs */
#endif
#if 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()`

<table>
<thead>
<tr>
<th>cleanup routines of NFT_MSG_NEWRULE</th>
<th>transaction abort routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>nf_tables_newrule</td>
<td>nf_tables_abort</td>
</tr>
<tr>
<td>nf_tables_rule_release</td>
<td>nf_tables_abort</td>
</tr>
<tr>
<td>nf_rule_expr_deactivate</td>
<td>nf_rule_expr_deactivate</td>
</tr>
<tr>
<td>nf_tables_deactivate_set</td>
<td>nf_tables_deactivate_set</td>
</tr>
<tr>
<td>nf_tables_unbind_set</td>
<td>nf_tables_unbind_set</td>
</tr>
<tr>
<td>list_del_rcu</td>
<td>list_del_rcu</td>
</tr>
<tr>
<td>nf_tables_rule_destroy</td>
<td>nf_tables_abort_release</td>
</tr>
<tr>
<td>nf_tables_expr_destroy</td>
<td>nf_tables_rule_destroy</td>
</tr>
<tr>
<td>nft_set_destroy</td>
<td>nft_set_destroy</td>
</tr>
</tbody>
</table>

1. `list_del_rcu`
2. `nft_set_destroy`
3. `list_del_rcu`
4. `nft_set_destroy`
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?

```c
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);
}
```
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, ffff8880506e400->prev is LIST_POISON2 (dead00000000122)
[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+0x1f2/0xd0
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 0xdead0000000000122: 0000 [#1] PREEMPT SMP PTI
[ 5.582038] 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
[ 5.587084] ? 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

```c
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

```
cpu #0 -> encode(A) -> ??? -> decode(???)
A
???
B
A
```
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

```
cpu #0  decode(???)
```

```
???
A

???
B

???
A
```
Demystifying kernel exploit mitigations:
**CONFIG_SLAB_FREELIST_HARDENED** invariant

- On LTS instance, further allocation in this slab results in #GP fault
Demystifying kernel exploit mitigations: CONFIG_SLAB_FREELIST_HARDENED invariant

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

```c
#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;
}))
```
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
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_VAR_SIZE=y` enabled with additional existing hardenings: `CONFIG_BUG_ON_DATA_CORRUPTION=y`, `CONFIG_FORTIFY_SOURCE=y`, `CONFIG_DEBUG_NX=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