black hat EUROPE 2021
november 10-11, 2021
BRIEFINGS

Your Trash Kernel Bug, My Precious 0-day

Zhenpeng Lin, Yueqi Chen, Xinyu Xing, and Kang Li

Agenda

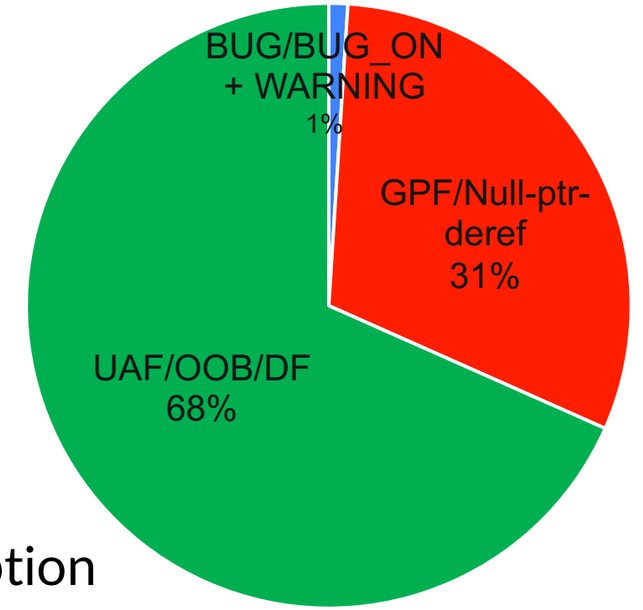
- A bug example that shows different memory corruption
- Definition of ***M**ultiple **E**rror **B**ehavior* of bugs
- A new approach to turning “unexploitable” bugs to “exploitable” ones
- Exploitation of a seemingly ”unexploitable” bug in CentOS kernel
- The implication of ***MEB***

“Unexploitable”

- Less likely to exploit
- The initial error cannot propagate
- Null-ptr-deref
- GPF
- Warning
- BUG_ON/BUG

Bugs without memory corruption

- Statistics from historical CVEs
  - Smaller portion
  - Lower score
- Less attention
  - Limited memory corruption ability
  - Exploitation requires memory corruption



	cvss2 avg. score
BUG/BUG_ON + WARNING	3.75
GPF/Null-ptr-deref	5.48
UAF/OOB/DF	6.17

A real kernel bug

```c
static void tun_attach(struct tun_struct *tun, ...)
{
    if (tun->flags & IFF_NAPI) {
        // initialize a timer
        hrtimer_init(&napi->timer, CLOCK_MONOTONIC,
            HRTIMER_MODE_REL_PINNED);
        // link current napi to the device's napi list
        list_add(&napi->dev_list, &dev->napi_list);
    }
}

static void tun_detach(struct tun_file *tfile, ...)
{
    struct tun_struct *tun = rtnl_dereference(tfile->tun);
    if (tun->flags & IFF_NAPI) {
        // GPF happens if timer is uninitialized
        hrtimer_cancel(&tfile->napi->timer);
        // remove the current napi from the list
        netif_napi_del(&tfile->napi);
    }
    destroy(tfile); // free napi
}

void free_netdev(struct net_device *dev) {
    list_for_each_entry_safe(p, n,
                &dev->napi_list, dev_list)
        netif_napi_del(p); // use-after-free
}
```


```c
static void tun_attach(struct tun_struct *tun, ...)
{
    if (tun->flags & IFF_NAPI) {
        // initialize a timer
        hrtimer_init(&napi->timer, CLOCK_MONOTONIC,
            HRTIMER_MODE_REL_PINNED);
        // link current napi to the device's napi list
        list_add(&napi->dev_list, &dev->napi_list);
    }
}
```

A real kernel bug

```c
static void tun_attach(struct tun_struct *tun, ...)
{
    if (tun->flags & IFF_NAPI) {
        // initialize a timer
        hrtimer_init(&napi->timer, CLOCK_MONOTONIC,
            HRTIMER_MODE_REL_PINNED);
        // link current napi to the device's napi list
        list_add(&napi->dev_list, &dev->napi_list);
    }
}

static void tun_detach(struct tun_file *tfile, ...)
{
    struct tun_struct *tun = rtnl_dereference(tfile->tun);
    if (tun->flags & IFF_NAPI) {
        // GPF happens if timer is uninitialized
        hrtimer_cancel(&tfile->napi->timer);
        // remove the current napi from the list
        netif_napi_del(&tfile->napi);
    }
    destroy(tfile); // free napi
}

void free_netdev(struct net_device *dev) {
    list_for_each_entry_safe(p, n,
                &dev->napi_list, dev_list)
        netif_napi_del(p); // use-after-free
}
```

```c
static void tun_detach(struct tun_file *tfile, ...)
{
    struct tun_struct *tun = rtnl_dereference(tfile->tun);
    if (tun->flags & IFF_NAPI) {
        // GPF happens if timer is uninitialized
        hrtimer_cancel(&tfile->napi->timer);
        // remove the current napi from the list
        netif_napi_del(&tfile->napi);
    }
    destroy(tfile); // free napi
}
```

A real kernel bug

```c
static void tun_attach(struct tun_struct *tun, ...)
{
    if (tun->flags & IFF_NAPI) {
        // initialize a timer
        hrtimer_init(&napi->timer, CLOCK_MONOTONIC,
            HRTIMER_MODE_REL_PINNED);
        // link current napi to the device's napi list
        list_add(&napi->dev_list, &dev->napi_list);
    }
}

static void tun_detach(struct tun_file *tfile, ...)
{
    struct tun_struct *tun = rtnl_dereference(tfile->tun);
    if (tun->flags & IFF_NAPI) {
        // GPF happens if timer is uninitialized
        hrtimer_cancel(&tfile->napi->timer);
        // remove the current napi from the list
        netif_napi_del(&tfile->napi);
    }
    destroy(tfile); // free napi
}

void free_netdev(struct net_device *dev) {
    list_for_each_entry_safe(p, n,
                &dev->napi_list, dev_list)
        netif_napi_del(p); // use-after-free
}
```

```c
void free_netdev(struct net_device *dev) {
    list_for_each_entry_safe(p, n,
                &dev->napi_list, dev_list)
        netif_napi_del(p); // use-after-free
}
```

An “unexploitable” crash

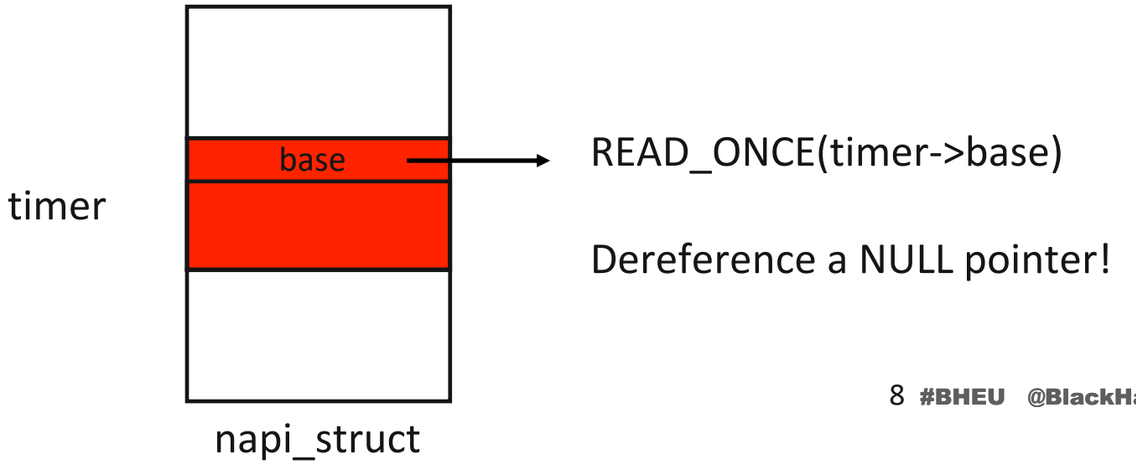
```c
static void tun_attach(struct tun_struct *tun, ...)
{
    if (tun->flags & IFF_NAPI) {
        // initialize a timer
        hrtimer_init(&napi->timer, CLOCK_MONOTONIC,
            HRTIMER_MODE_REL_PINNED);
        // link current napi to the device's napi list
        list_add(&napi->dev_list, &dev->napi_list);
    }
}

static void tun_detach(struct tun_file *tfile, ...)
{
    struct tun_struct *tun = rtnl_dereference(tfile->tun);
    if (tun->flags & IFF_NAPI) {
        // GPF happens if timer is uninitialized
        hrtimer_cancel(&tfile->napi->timer);
        // remove the current napi from the list
        netif_napi_del(&tfile->napi);
    }
    destroy(tfile); // free napi
}

void free_netdev(struct net_device *dev) {
    list_for_each_entry_safe(p, n,
                &dev->napi_list, dev_list)
        netif_napi_del(p); // use-after-free
}
```

1. *tun_attach* with *IFF_NAPI* **disabled**
   - ***timer uninitialized***
   - ***current napi not in the list***

2. *tun_detach* with IFF_NAPI **enabled**
   - ***cancel the timer***



An “unexploitable” crash

```c
static void tun_attach(struct tun_struct *tun, ...)
{
    if (tun->flags & IFF_NAPI) {
        // initialize a timer
        hrtimer_init(&napi->timer, CLOCK_MONOTONIC,
            HRTIMER_MODE_REL_PINNED);
        // link current napi to the device's napi list
        list_add(&napi->dev_list, &dev->napi_list);
    }
}

static void tun_detach(struct tun_file *tfile, ...)
{
    struct tun_struct *tun = rtnl_dereference(tfile->tun);
    if (tun->flags & IFF_NAPI) {
        // GPF happens if timer is uninitialized
        hrtimer_cancel(&tfile->napi->timer);
        // remove the current napi from the list
        netif_napi_del(&tfile->napi);
    }
    destroy(tfile); // free napi
}

void free_netdev(struct net_device *dev) {
    list_for_each_entry_safe(p, n,
                &dev->napi_list, dev_list)
        netif_napi_del(p); // use-after-free
}
```

1. *tun_attach* with *IFF_NAPI* **disabled**
   - ***timer uninitialized***
   - ***current napi not in the list***

2. *tun_detach* with IFF_NAPI **enabled**
   - ***cancel the timer***

Exploitability?

- Always access to an ***uninitialized*** timer
- Mapping memory at 0 is not allowed
- Very ***less likely*** to be exploitable

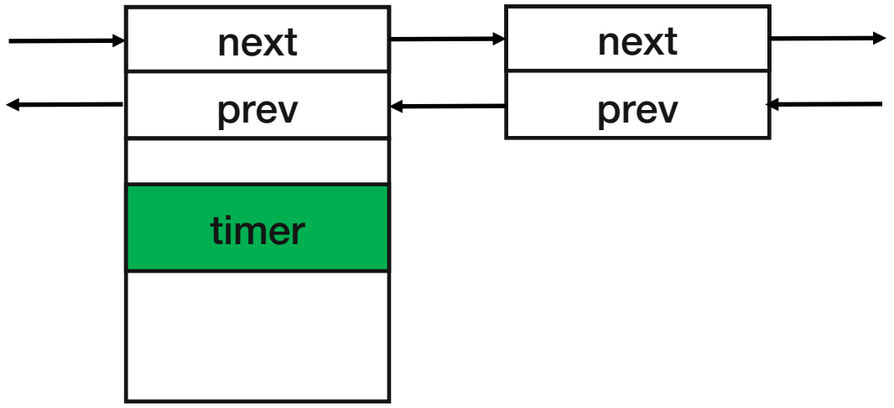
Other than “unexploitable”

```c
static void tun_attach(struct tun_struct *tun, ...)
{
    if (tun->flags & IFF_NAPI) {
        // initialize a timer
        hrtimer_init(&napi->timer, CLOCK_MONOTONIC,
            HRTIMER_MODE_REL_PINNED);
        // link current napi to the device's napi list
        list_add(&napi->dev_list, &dev->napi_list);
    }
}

static void tun_detach(struct tun_file *tfile, ...)
{
    struct tun_struct *tun = rtnl_dereference(tfile->tun);
    if (tun->flags & IFF_NAPI) {
        // GPF happens if timer is uninitialized
        hrtimer_cancel(&tfile->napi->timer);
        // remove the current napi from the list
        netif_napi_del(&tfile->napi);
    }
    destroy(tfile); // free napi
}

void free_netdev(struct net_device *dev) {
    list_for_each_entry_safe(p, n,
                &dev->napi_list, dev_list)
        netif_napi_del(p); // use-after-free
}
```

1. *tun_attach* with *IFF_NAPI* **enabled**
   - ***initialize the timer***
   - ***current napi lnked in the list***



napi_sruct

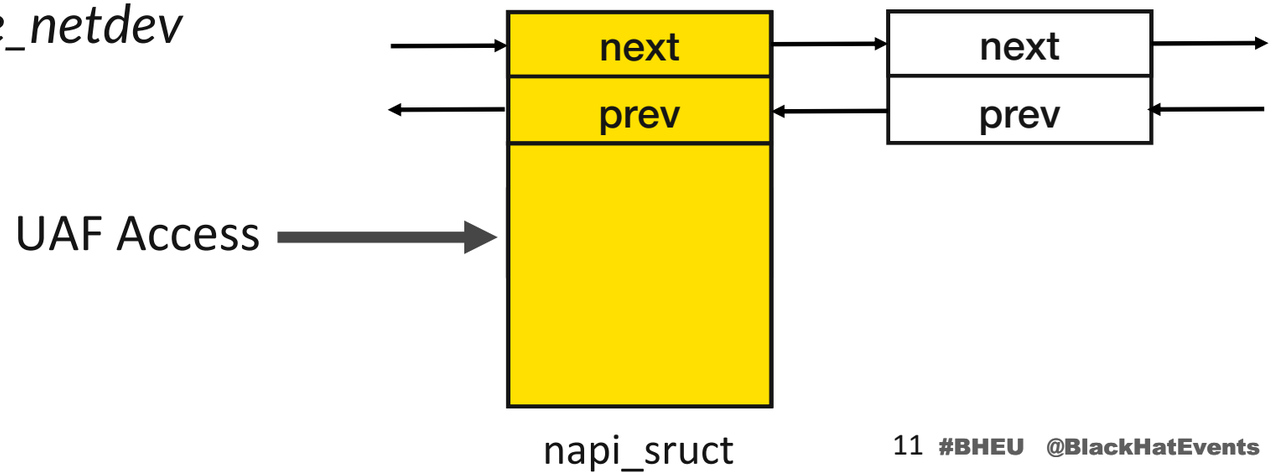
Other than “unexploitable”

```c
static void tun_attach(struct tun_struct *tun, ...)
{
    if (tun->flags & IFF_NAPI) {
        // initialize a timer
        hrtimer_init(&napi->timer, CLOCK_MONOTONIC,
            HRTIMER_MODE_REL_PINNED);
        // link current napi to the device's napi list
        list_add(&napi->dev_list, &dev->napi_list);
    }
}

static void tun_detach(struct tun_file *tfile, ...)
{
    struct tun_struct *tun = rtnl_dereference(tfile->tun);
    if (tun->flags & IFF_NAPI) {
        // GPF happens if timer is uninitialized
        hrtimer_cancel(&tfile->napi->timer);
        // remove the current napi from the list
        netif_napi_del(&tfile->napi);
    }
    destroy(tfile); // free napi
}

void free_netdev(struct net_device *dev) {
    list_for_each_entry_safe(p, n,
                &dev->napi_list, dev_list)
        netif_napi_del(p); // use-after-free
}
```

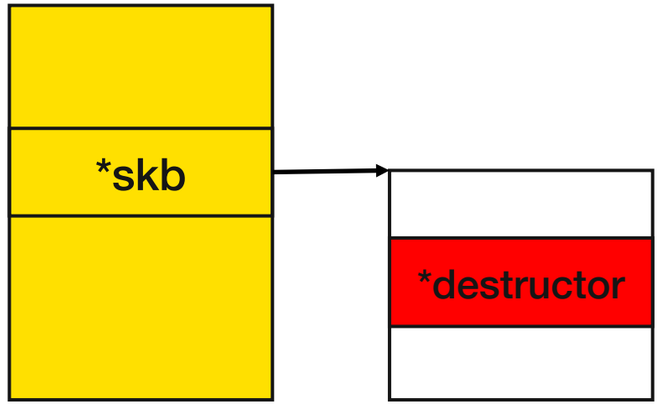
1. *tun_attach* with *IFF_NAPI* **enabled**
   - ***initialize the timer***
   - ***current napi lnked in the list***
2. *tun_detach* with IFF_NAPI **disabled**
   - ***napi still in the list***
   - ***napi freed by destroy(tfile)***
3. *free_netdev*



Exploitability of this UAF

- The vulnerable object is allocated in general cache (kmalloc-2048)
- Free of napi_struct will free the ***sk_buff*** inside
- Control hijacking when freeing ***sk_buff***

```
napi_struct {
...
struct sk_buff      *skb;
...
}
```



napi_sruct

Turning “unexploitable” to “exploitable”

- Why?
  - *Multiple **E**rror **B**ehaviors* if triggered differently
  - **MEB** demonstrates different bug effects (memory corruption ability)
  - Different bug effects have different exploit potential
- How?
  - Coverage-based fuzzer is not suitable
  - Object-driven: use error-related objects to bound the fuzzing scope!
  - A customized fuzzer based on Syzkaller

Object-driven kernel fuzzing

- Find objects from crash report
- Instrument BBs which contain the object
- New seed selection and input mutation based on object feedback
- Our tool and other details will be released soon!

base = READ_ONCE(timer->base) in ***hrtimer_active***

hrtimer_try_to_cancel(timer) in ***hrtimer_cancel***

hrtimer_cancel(&tfile->napi->timer) in ***tun_detach***

What it means

- To vendors
  - Do not rely on signal input/behavior on exploitability estimation
  - Fix all the bugs!
- To offensive technique researchers
  - “Trash” bug could be exploitable
  - A better primitive could be found automatically
  - A lot of ***unfixed exploitable*** bugs in vendors’ kernel

CVE-2021-3715

- fixed in upstream in March 2020
- Firstly found by Syzkaller with a WARNING error
- UAF if triggered differently
- Unfixed in Centos 7 and many other Centos-based distroes (TencentOS, AliOS) by August 2021

The Bug

- The vulnerable object
  - Allocated by kzalloc at *route4_change*
  - Freed by workqueue at *route4_delete_filter*
  - The work is queued at *route4_change*

```
Allocated by task 6915:
 save_stack+0x1b/0x40 mm/kasan/common.c:49
 set_track mm/kasan/common.c:57 [inline]
 __kasan_kmalloc mm/kasan/common.c:492 [inline]
 __kasan_kmalloc.constprop.0+0xc2/0xd0 mm/kasan/common.c:465
 kmem_cache_alloc_trace+0x156/0x3d0 mm/slab.c:3551
 kmalloc include/linux/slab.h:555 [inline]
 kzalloc include/linux/slab.h:669 [inline]
 route4_change+0x2a8/0x2250 net/sched/cls_route.c:493
 tc_new_tfilter+0xa64/0x20c0 net/sched/cls_api.c:2103
 rtnetlink_rcv_msg+0x80f/0xad0 net/core/rtnetlink.c:5431
 netlink_rcv_skb+0x15a/0x410 net/netlink/af_netlink.c:2478
 netlink_unicast_kernel net/netlink/af_netlink.c:1303 [inline]
 netlink_unicast+0x533/0x740 net/netlink/af_netlink.c:1329
 netlink_sendmsg+0x856/0xd90 net/netlink/af_netlink.c:1918
 sock_sendmsg_nosec net/socket.c:652 [inline]
 sock_sendmsg+0xcf/0x120 net/socket.c:672
 ____sys_sendmsg+0x6bb/0x7d0 net/socket.c:2343
 ___sys_sendmsg+0x100/0x170 net/socket.c:2397
 __sys_sendmsg+0xec/0x1b0 net/socket.c:2430
 do_syscall_64+0xf6/0x790 arch/x86/entry/common.c:295
 entry_SYSCALL_64_after_hwframe+0x49/0xbe

Freed by task 7:
 save_stack+0x1b/0x40 mm/kasan/common.c:49
 set_track mm/kasan/common.c:57 [inline]
 kasan_set_free_info mm/kasan/common.c:314 [inline]
 __kasan_slab_free+0xf5/0x140 mm/kasan/common.c:453
 __cache_free mm/slab.c:3426 [inline]
 kfree+0x10b/0x2b0 mm/slab.c:3757
 route4_delete_filter_work+0x17/0x20 net/sched/cls_route.c:266
 process_one_work+0x94f/0x1690 kernel/workqueue.c:2266
 worker_thread+0x96/0xe20 kernel/workqueue.c:2412
 kthread+0x357/0x430 kernel/kthread.c:255
 ret_from_fork+0x24/0x30 arch/x86/entry/entry_64.S:352
```

The Bug

- What *route4_change* does?
  - Allocate **route4_filter**
  - Update **route4_filter**
    - CORRECT: remove the old one from linked list, then free
    - ERROS: remove the ***new*** one from linked list, then free the ***old*** one

```
static int route4_change(struct net *net, struct sk_buff *in_skb,
    fp = &b->ht[h];
    for (pfp = rtnl_dereference(*fp); pfp;
         fp = &pfp->next, pfp = rtnl_dereference(*fp)) {
            if (pfp == f) {
                    *fp = f->next;
            if (pfp == fold) {
                    rcu_assign_pointer(*fp, fold->next);
                    break;
            }
    }
```

The initial bug report

- Unsuspicious WARNING
- Happened at calling RCU
- Requires collision
- might be unstable?

```
ODEBUG: activate active (active state 1) object type: rcu_head hint: 0x0
WARNING: CPU: 1 PID: 7214 at lib/debugobjects.c:485 debug_print_object+0x160/0x
Kernel panic - not syncing: panic_on_warn set ...
CPU: 1 PID: 7214 Comm: syz-executor526 Not tainted 5.6.0-rc7-syzkaller #0
Hardware name: Google Google Compute Engine/Google Compute Engine, BIOS Google
Call Trace:
 __dump_stack lib/dump_stack.c:77 [inline]
 dump_stack+0x188/0x20d lib/dump_stack.c:118
 panic+0x2e3/0x75c kernel/panic.c:221
 __warn.cold+0x2f/0x35 kernel/panic.c:582
 report_bug+0x27b/0x2f0 lib/bug.c:195
 fixup_bug arch/x86/kernel/traps.c:174 [inline]
 fixup_bug arch/x86/kernel/traps.c:169 [inline]
 do_error_trap+0x12b/0x220 arch/x86/kernel/traps.c:267
 do_invalid_op+0x32/0x40 arch/x86/kernel/traps.c:286
 invalid_op+0x23/0x30 arch/x86/entry/entry_64.S:1027
RIP: 0010:debug_print_object+0x160/0x250 lib/debugobjects.c:485
Code: dd 00 f2 51 88 48 89 fa 48 c1 ea 03 80 3c 02 00 0f 85 bf 00 00 00 48 8b 1
RSP: 0018:ffffc90000f87178 EFLAGS: 00010282
RAX: 0000000000000000 RBX: 0000000000000003 RCX: 0000000000000000
RDX: 0000000000000000 RSI: ffffffff815c06c1 RDI: fffff520001f0e21
RBP: 0000000000000001 R08: ffff8880a26ee140 R09: ffffed1015ce6659
R10: ffffed1015ce6658 R11: ffff8880ae7332c7 R12: ffffffff897acba0
R13: 0000000000000000 R14: dffffc0000000000 R15: 1ffff920001f0e3c
 debug_object_activate+0x346/0x470 lib/debugobjects.c:652
 debug_rcu_head_queue kernel/rcu/rcu.h:176 [inline]
 __call_rcu kernel/rcu/tree.c:2597 [inline]
 call_rcu+0x2c/0x690 kernel/rcu/tree.c:2683
 queue_rcu_work+0x82/0xa0 kernel/workqueue.c:1744
 route4_change+0x19e8/0x2250 net/sched/cls_route.c:550
```

Primitives discovered *automatically*

- UAF Read in ***route4_get***

```
==================================================================
BUG: KASAN: use-after-free in route4_get+0x3e1/0x420 net/sched/cls_route.c:235
Read of size 4 at addr ffff88806a474040 by task syz-executor.2/7402

CPU: 1 PID: 7402 Comm: syz-executor.2 Not tainted 5.6.0-rc3-next-20200228 #1
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.10.2-1ubuntu1 04/
Call Trace:
 __dump_stack lib/dump_stack.c:77 [inline]
 dump_stack+0x188/0x20d lib/dump_stack.c:118
 print_address_description.constprop.0.cold+0xd3/0x315 mm/kasan/report.c:374
 __kasan_report.cold+0x1a/0x32 mm/kasan/report.c:506
 kasan_report+0xe/0x20 mm/kasan/common.c:618
 route4_get+0x3e1/0x420 net/sched/cls_route.c:235
 tc_new_tfilter+0x7b3/0x20c0 net/sched/cls_api.c:2082
 rtnetlink_rcv_msg+0x80f/0xad0 net/core/rtnetlink.c:5431
 netlink_rcv_skb+0x15a/0x410 net/netlink/af_netlink.c:2478
 netlink_unicast_kernel net/netlink/af_netlink.c:1303 [inline]
```

Primitives discovered *automatically*

- UAF Read in ***route4_destroy***

```
==================================================================
BUG: KASAN: use-after-free in route4_destroy+0x6d1/0x820 net/sched/cls_route.c:295
Read of size 8 at addr ffff88806a9fde00 by task syz-executor.0/8011

CPU: 1 PID: 8011 Comm: syz-executor.0 Not tainted 5.6.0-rc3-next-20200228 #1
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.10.2-1ubuntu1 04/01/2
Call Trace:
 __dump_stack lib/dump_stack.c:77 [inline]
 dump_stack+0x188/0x20d lib/dump_stack.c:118
 print_address_description.constprop.0.cold+0xd3/0x315 mm/kasan/report.c:374
 __kasan_report.cold+0x1a/0x32 mm/kasan/report.c:506
 kasan_report+0xe/0x20 mm/kasan/common.c:618
 route4_destroy+0x6d1/0x820 net/sched/cls_route.c:295
 tcf_proto_destroy+0x6e/0x310 net/sched/cls_api.c:296
 tcf_proto_put+0x8c/0xc0 net/sched/cls_api.c:308
 tcf_chain_flush+0x266/0x390 net/sched/cls_api.c:600
 tcf_block_flush_all_chains net/sched/cls_api.c:1052 [inline]
 __tcf_block_put+0x1a6/0x540 net/sched/cls_api.c:1214
 tcf_block_put_ext net/sched/cls_api.c:1414 [inline]
 tcf_block_put+0xb3/0x100 net/sched/cls_api.c:1429
 hfsc_destroy_qdisc+0xe0/0x280 net/sched/sch_hfsc.c:1501
 qdisc_destroy+0x118/0x690 net/sched/sch_generic.c:958
```

Primitives discovered *automatically*

- UAF Read in ***route4_dump***

```
==================================================================
BUG: KASAN: use-after-free in route4_dump net/sched/cls_route.c:605 [inline]
BUG: KASAN: use-after-free in route4_dump+0x571/0x630 net/sched/cls_route.c:595
Read of size 4 at addr ffff88806b35d340 by task syz-executor.1/7520

CPU: 0 PID: 7520 Comm: syz-executor.1 Not tainted 5.6.0-rc3-next-20200228 #2
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.10.2-1ubuntu1 04/(
Call Trace:
 __dump_stack lib/dump_stack.c:77 [inline]
 dump_stack+0x188/0x20d lib/dump_stack.c:118
 print_address_description.constprop.0.cold+0xd3/0x315 mm/kasan/report.c:374
 __kasan_report.cold+0x1a/0x32 mm/kasan/report.c:506
 kasan_report+0xe/0x20 mm/kasan/common.c:618
 route4_dump net/sched/cls_route.c:605 [inline]
 route4_dump+0x571/0x630 net/sched/cls_route.c:595
 tcf_fill_node+0x574/0x930 net/sched/cls_api.c:1814
 tcf_node_dump+0x1ce/0x2c0 net/sched/cls_api.c:2441
 route4_walk+0x249/0x540 net/sched/cls_route.c:584
 tcf_chain_dump+0x31f/0x790 net/sched/cls_api.c:2494
```

Leak kernel data

- UAF Read in ***route4_dump***
  - nla_put_32
  - Send kernel data to user space
  - Spray pointers in the leaking field

```
static int route4_dump(...)
{
    ...

    if (!(f->handle & 0x8000)) {
        id = f->id & 0xFF;
        if (nla_put_u32(skb, TCA_ROUTE4_TO, id))
            goto nla_put_failure;
    }
    if (f->handle & 0x80000000) {
        if ((f->handle >> 16) != 0xFFFF &&
            nla_put_u32(skb, TCA_ROUTE4_IIF, f->iif))
            goto nla_put_failure;
    } else {
        id = f->id >> 16;
        if (nla_put_u32(skb, TCA_ROUTE4_FROM, id))
            goto nla_put_failure;
    }
    if (f->res.classid &&
        nla_put_u32(skb, TCA_ROUTE4_CLASSID, f->res.classid))
        goto nla_put_failure;
    ...
}
```

Hijack control flow

- UAF in ***route4_destory***
  - Queue work at ***route4_destory***
  - Func pointer inside *route4_filter* object
  - Spray objects to overwrite func pointer after queuing.

```
struct route4_filter {
    ...
    struct rcu_work     rwork;
};

struct rcu_work {
    struct work_struct work;
    ...
};

struct work_struct {
    atomic_long_t data;
    struct list_head entry;
    work_func_t func;
    ...
};

static void route4_destroy(...)
{
    ...

    if (tcf_exts_get_net(&f->exts))
        route4_queue_work(f);
    else
        __route4_delete_filter(f);
    ...
}
```

Multiple Error Behaviors

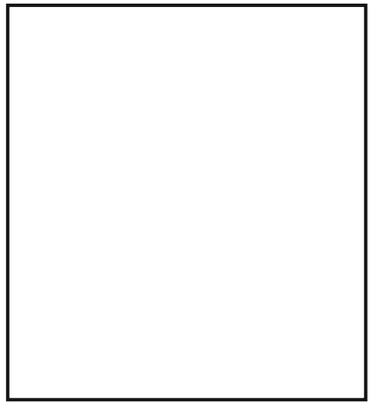
- Different errors if triggered differently
- Find the worst bug effects
- Discover exploit primitive

A generic UAF exploitation

- Only requires free ability from the UAF
- Leak and overwrite with universal spray object

Steps

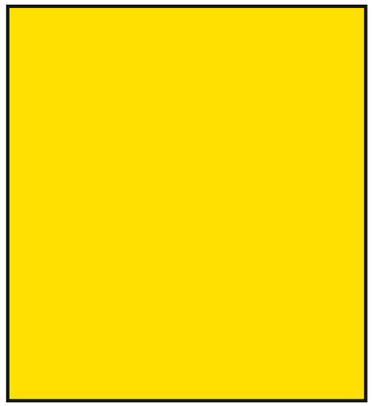
- Free the vuln object



route4_filter

Steps

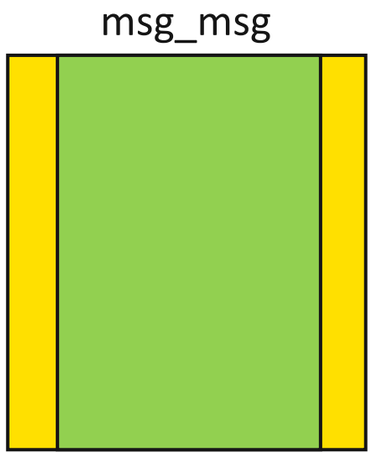
- Free the vuln object



route4_filter

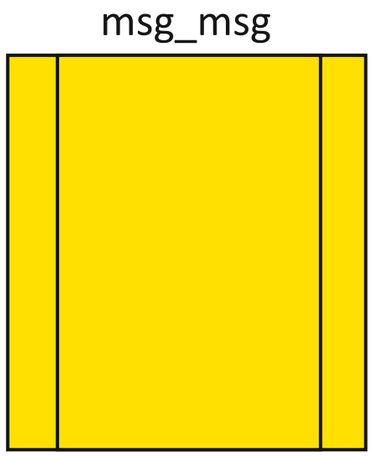
Steps

- Free the vuln object
- Spray with *struct msg_msg (or msg_msgseg)*, **it can be in any general cache**



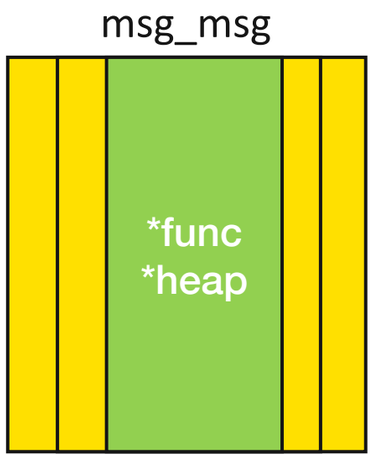
Steps

- Free the vuln object
- Spray with *struct msg_msg (or struct msg_msgseg)*, **it can be in any cache**
- Free *object msg_msg* through the vulnerability



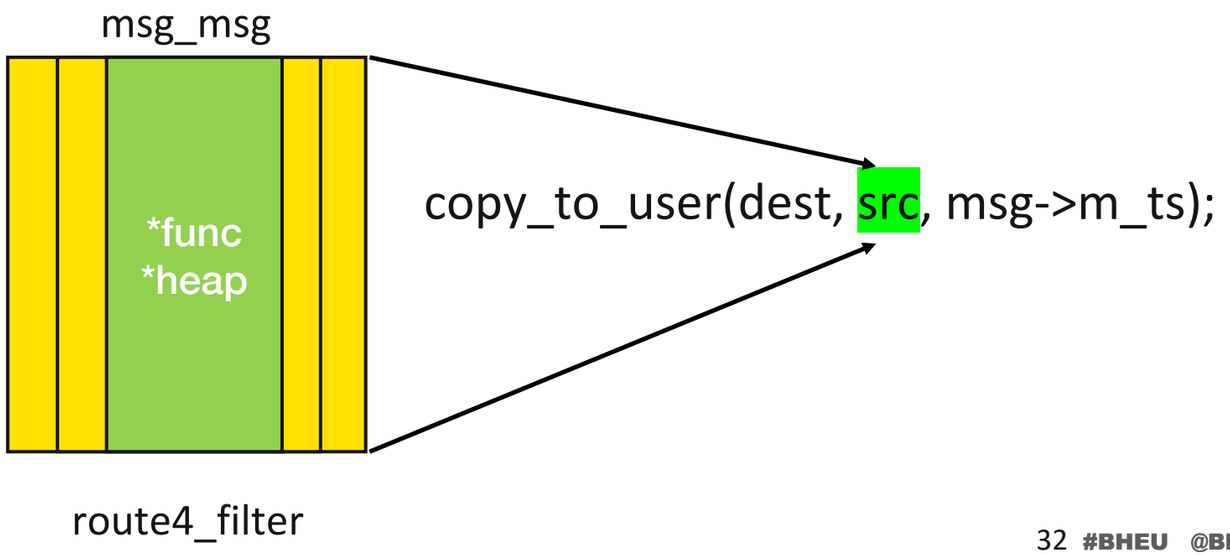
Steps

- Free the vuln object
- Spray with *struct msg_msg (or struct msg_msgseg)*, **it can be in any cache**
- Free *object msg_msg* through the vulnerability
- Spray victim objects



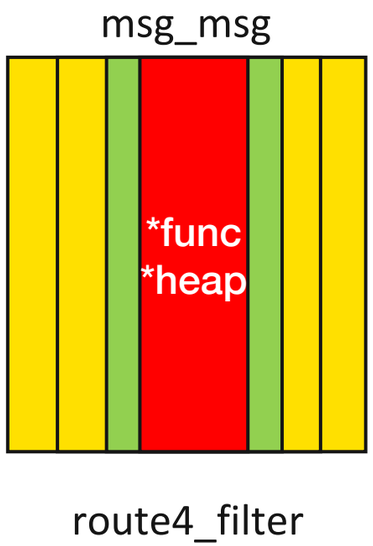
Steps

- Free the vuln object
- Spray with *struct msg_msg (or struct msg_msgseg)*, **it can be in any cache**
- Free *object msg_msg* through the vulnerability
- Spray victim objects
- Leak kernel data (*store_msg*)



Steps

- Free the vuln object
- Spray with *struct msg_msg (or struct msg_msgseg)*, **it can be in any cache**
- Free *object msg_msg* through the vulnerability
- Spray victim objects
- Leak kernel data (*store_msg*)
- Hijack control flow



One more thing...

- Argument is not controlled when hijacking, e.g. vtm->func(vtm)
- Cannot call to *run_cmd* or *native_write_cr4*
- Try ***install_exec_creds*** before v5.8 or ***commit_nsset*** after v5.7

One more thing...

- Argument is r
- Cannot call tc
- Try ***install_ex***

tm)

,

```
static void commit_nsset(struct nsset *nsset)
{
    ...
    commit_creds(nsset_cred(nsset));
    ...
}

void install_exec_creds(struct linux_binprm *bprm)
{
    ...
    commit_creds(bprm->cred);
    bprm->cred = NULL;
    ...
}
```

One more thing...

- Argument is not controlled when hijacking, e.g. vtm->func(vtm)
- Cannot call to *run_cmd* or *native_write_cr4*
- Try ***install_exec_creds*** before v5.8 or ***commit_nsset*** after v5.7

- Exploit available at https://github.com/Markakd/kernel_exploit

Takeaway

- “Trash” bug could be exploitable, fix them!
- Exploit primitives could be found automatically
- There might be other ***unfixed exploitable*** bugs in vendors’ kernel

Takeaway

- “Trash” bug could be exploitable, fix them!
- Exploit primitives could be found automatically
- There might be other ***unfixed exploitable*** bugs in vendors’ kernel

Zhenpeng Lin (@Markak_)
https://zplin.me
Looking for summer internship!