LINUX SECURITY SUMMIT

Finding Multiple Bug Effects for More Precise Exploitability Estimation

Zhenpeng Lin & Yueqi Chen
The Pennsylvania State University

THE LINUX FOUNDATION

Kernel bugs found by Syzkaller

- syzbot: continuous kernel fuzzing
  - Public
  - ~4400 bugs for 4 years
  - ~3000 fixed bugs, ~1000 open bugs
  - Exploitability of these bugs are unknown
    - zero day in upstream
    - zero day in vendors' kernel

Exploitability

- Estimate the consequence of bugs
- Promote bug fixes and fixes adoption
- Guide the design of hardening

Challenges of knowing exploitability

- Proving exploitability is hard
  - write the exploit!
- Proving unexploitability?
  - is even harder
  - no path leading to exploitation
  - talked by some [academic research]
  - not realistic for kernel

Exploitability approximation

- Approximate the likelihood of exploitation
- Based on the read/write ability of UAF/OOB bugs?
  - No
  - Exploitability of UAF
    - Transfer UAF object to others
    - From UAF bugs to information leaking
  - Exploitability of OOB
    - Exploit kernel with 4 zero-bytes overflow

Exploitability approximation (cont.)

- Based on the type of bug
  - Likely to exploit
    - UAF, double/invalid free
    - OOB
  - Less likely to exploit
    - WARNING
    - INFO
    - GPF
    - Null-ptr-deference
    - ...

The reliability of approximation

- How bugs are underestimated
  - severe bug doesn’t show memory corruption
  - severe bug shows limited memory corruption ability

- How to improve the reliability
  - Find the true effect of bugs

Exploitability being underestimated

- Syzkaller generates incomplete errors
  - Misses KASAN errors when “*panic_on_warn*” is set

Exploitability being underestimated

```
Kernel panic - not syncing: panic_on_warn set ...
CPU: 0 PID: 59 Comm: kworker/u4:2 Not tainted 5.1.0-rc6+ #83
Hardware name: Google Google Compute Engine/Google Compute Engine, BIOS G
Workqueue: netns cleanup_net
Call Trace:
 __dump_stack lib/dump_stack.c:77 [inline]
 dump_stack+0x172/0x1f0 lib/dump_stack.c:113
 panic+0x2cb/0x65c kernel/panic.c:214
 __warn.cold+0x20/0x45 kernel/panic.c:571
 report_bug+0x263/0x2b0 lib/bug.c:186
 fixup_bug arch/x86/kernel/traps.c:179 [inline]
 fixup_bug arch/x86/kernel/traps.c:174 [inline]
 do_error_trap+0x11b/0x200 arch/x86/kernel/traps.c:272
 do_invalid_op+0x37/0x50 arch/x86/kernel/traps.c:291
 invalid_op+0x14/0x20 arch/x86/entry/entry_64.S:973
RIP: 0010:xfrm_state_fini+0x218/0x280 net/xfrm/xfrm_state.c:2389
Code: 41 5e 5d c3 e8 29 b0 66
RSP: 0018:ffff8880a9987bd0 EF
RAX: ffff8880a9970080 RBX: ff
RDX: 0000000000000000 RSI: ff
RBP: ffff8880a9987bf0 R08: ff
R10: 0000000000000000 R11: 0000000000000000 R12: ffff8880a7cd1a80
R13: ffff8880a9987cc8 R14: ffffffff893ea658 R15: dffffc0000000000
 xfrm_net_exit+0x25/0x70 net/xfrm/xfrm_policy.c:3934
 ops_exit_list.isra.0+0xb0/0x160 net/core/net_namespace.c:153
 cleanup_net+0x3fb/0x960 net/core/net_namespace.c:552
 process_one_work+0x98e/0x1790 kernel/workqueue.c:2269
 worker_thread+0x98/0xe40 kernel/workqueue.c:2415
 kthread+0x357/0x430 kernel/kthread.c:253
 ret_from_fork+0x3a/0x50 arch/x86/entry/entry_64.S:352
Kernel Offset: disabled
Rebooting in 86400 seconds..
```

```
WARNING: CPU: 1 PID: 5920 at net/xfrm/xfrm_state.c:2389 xfrm_state_fini+0x1f1/0x260
Modules linked in:
CPU: 1 PID: 5920 Comm: kworker/u4:4 Not tainted 5.1.0-rc6 #1
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.13.0-1ubuntu1.1 04/01/2014
Workqueue: netns cleanup_net
RIP: 0010:xfrm_state_fini+0x1f1/0x260
Code: 41 5d 5d c3 e8 90 ea b5 fb 0f 0b e9 24 ff ff ff e8 84 ea b5 fb 0f 0b e9 75 ff ff ff e8 78 ea
RSP: 0018:ffff88805fdbfbe8 EFLAGS: 00010293
RAX: ffff8880660185c0 RBX: ffff88806105c180 RCX: ffffffff85b40c91
RDX: 0000000000000000 RSI: ffff888066018e90 RDI: ffffffff88324180
RBP: ffff88805fdbfc00 R08: 0000000000000001 R09: 0000000000000000
R10: 0000000000000000 R11: 0000000000000000 R12: ffff88806105dbc0
R13: ffff88805fdbfcf8 R14: ffffffff88d648f8 R15: dffffc0000000000
FS:  0000000000000000(0000) GS:ffff88806c900000(0000) knlGS:0000000000000000
CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
CR2: 00007f2a56ab6000 CR3: 0000000065400000 CR4: 00000000000006e0
Call Trace:
 ? xfrm_policy_fini+0x380/0x380
 xfrm_net_exit+0x25/0x70
 ops_exit_list.isra.4+0xb0/0x160
 ? process_one_work+0x1620/0x1620
 kthread+0x354/0x430
 ? kthread_cancel_delayed_work_sync+0x20/0x20
 ret_from_fork+0x27/0x50
irq event stamp: 17508478
hardirqs last  enabled at (17508477): [<ffffffff8140b440>] __local_bh_enable_ip+0x120/0x270
hardirqs last disabled at (17508478): [<ffffffff81006087>] trace_hardirqs_off_thunk+0x1a/0x1c
softirqs last  enabled at (17508476): [<ffffffff85b2cef7>] xfrm_state_flush+0x497/0x550
softirqs last disabled at (17508474): [<ffffffff85b2ca98>] xfrm_state_flush+0x38/0x550
---[ end trace 292c972ebedd4fdb ]---

==================================================================
BUG: KASAN: use-after-free in __lock_acquire+0x3b96/0x3d10
Read of size 8 at addr ffff88806105e018 by task swapper/0/0
```

KASAN error is not shown

WARNING in xfrm_state_fini *v.s.* KASAN: **use-after-free** Read in __lock_acquire

Exploitability being underestimated

- Syzkaller generates incomplete errors
  - Misses KASAN errors when "*panic_on_warn*" is set
  - Only reports first error kernel triggers

Exploitability being underestimated

```
=========================
WARNING: held lock freed!
4.15.0-rc7+ #261 Not tainted
-------------------------
syzkaller113545/3666 is freeing memory 00000000c92133c6-0000000092cb75b3, with a lock still held there!
 (sk_lock-AF_INET6){+.+.}, at: [<00000000b1bf268b>] lock_sock include/net/sock.h:1461 [inline]
 (sk_lock-AF_INET6){+.+.}, at: [<00000000b1bf268b>] sctp_wait_for_sndbuf+0x509/0x8d0 net/sctp/socket.c:8056
1 lock held by syzkaller113545/3666:
 #0:  (sk_lock-AF_INET6){+.+.}, at: [<00000000b1bf268b>] lock_sock include/net/sock.h:1461 [inline]
 #0:  (sk_lock-AF_INET6){+.+.}, at: [<00000000b1bf268b>] sctp_wait_for_sndbuf+0x509/0x8d0 net/sctp/socket.c:8056

stack backtrace:
CPU: 0 PID: 3666 Comm: syzkaller113545 Not tainted 4.15.0-rc7+ #261
Hardware name: Google Google Compute Engine/Google Compute Engine, BIOS Google 01/01/2011
Call Trace:
 __dump_stack lib/dump_stack.c:17 [inline]
 dump_stack+0x194/0x257 lib/dump_stack.c:53
 print_freed_lock_bug kernel/locking/lockdep.c:4379 [inline]
 debug_check_no_locks_freed+0x32f/0x3c0 kernel/locking/lockdep.c:4412
 kmem_cache_free+0x68/0x2a0 mm/slab.c:3743
 sk_prot_free net/core/sock.c:1504 [inline]
 __sk_destr
 sk_destruc
 __sk_free
 sk_free+0x
 sock_put i
 sctp_associ
 sctp_association_put+0x14c/0x2f0 net/sctp/associola.c:883
 sctp_wait_for_sndbuf+0x673/0x8d0 net/sctp/socket.c:8067
 sctp_sendmsg+0x277d/0x3360 net/sctp/socket.c:1974
 inet_sendmsg+0x11f/0x5e0 net/ipv4/af_inet.c:763
 sock_sendmsg_nosec net/socket.c:638 [inline]
 sock_sendmsg+0xca/0x110 net/socket.c:648
 SYSC_sendto+0x361/0x5c0 net/socket.c:1729
 SyS_sendto+0x40/0x50 net/socket.c:1697
 entry_SYSCALL_64_fastpath+0x23/0x9a
RIP: 0033:0x4457e9
RSP: 002b:00007fe7bf12bda8 EFLAGS: 00000216 ORIG_RAX: 000000000000002c
RAX: ffffffffffffffda RBX: 00000000006dac6c RCX: 00000000004457e9
RDX: 0000000000000001 RSI: 000000002010bf14 RDI: 0000000000000004
RBP: 00000000006dac68 R08: 00000000204d9000 R09: 000000000000001c
R10: 0000000000000000 R11: 0000000000000216 R12: 0000000000000000
R13: 00007ffc3c438f0f R14: 00007fe7bf12c9c0 R15: 0000000000000001
==================================================================
BUG: KASAN: use-after-free in debug_spin_lock_before kernel/locking/spinlock_debug.c:83 [inline]
BUG: KASAN: use-after-free in do_raw_spin_lock+0x1e0/0x220 kernel/locking/spinlock_debug.c:112
Read of size 4 at addr ffff8801bbbae08c by task syzkaller113545/3666
```

KASAN error is ignored

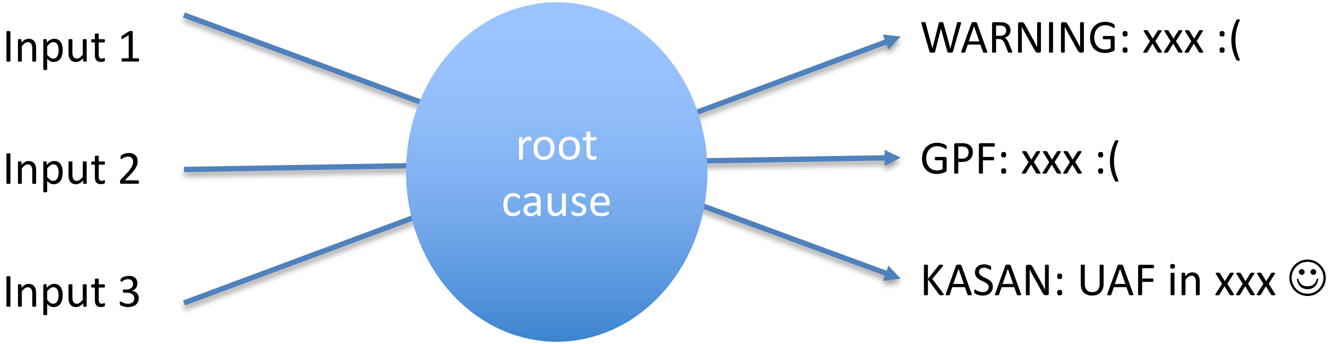
WARNING: held lock freed!

Exploitability being underestimated

- Incomplete error reported by Syzkaller
  - Misses KASAN errors when "*panic_on_warn*" is set
  - Only reports first error kernel triggers
- **Multiple Error Behaviors (MEB)**

Multiple Error Behaviors

- With the ***same*** root cause, but ***different*** errors at ***different*** sites



Exposing MEB to avoid underestimation

“Unexploitable” path *v.s.* “Exploitable” path

```
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
}
```


```
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);
    }
}
```

“Unexploitable” path *v.s.* “Exploitable” path

```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
}
```

“Unexploitable” path *v.s.* “Exploitable” path

```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
}
```

“Unexploitable” path *v.s.* “Exploitable” path

```
static void tun_attach(struct tun_struct *tun, ...)
{
    if (tun->flags & IFF_NAPI) {
        // initialize a timer

    }
}

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**
    - ***no timer***
    - ***current napi not in the list***

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

Null-ptr-def happens

“Unexploitable” path *v.s.* “Exploitable” path

```
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) {

    }
    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*
   - ***deference the dangling pointer***

UAF happens

Exploitability of two behaviors

- Exploit the Null-ptr-dereference
  - mapping at 0 is not allowed
- Exploit the UAF
  - netif_napi_del (napi)
    - kfree_skb(napi->skb)
      - napi->skb->destructor(napi->skb) (**Hijack control flow)**

Precise exploitability estimation needs to expose Multiple Error Behaviors of bug.

Finding Multiple Error Behaviors

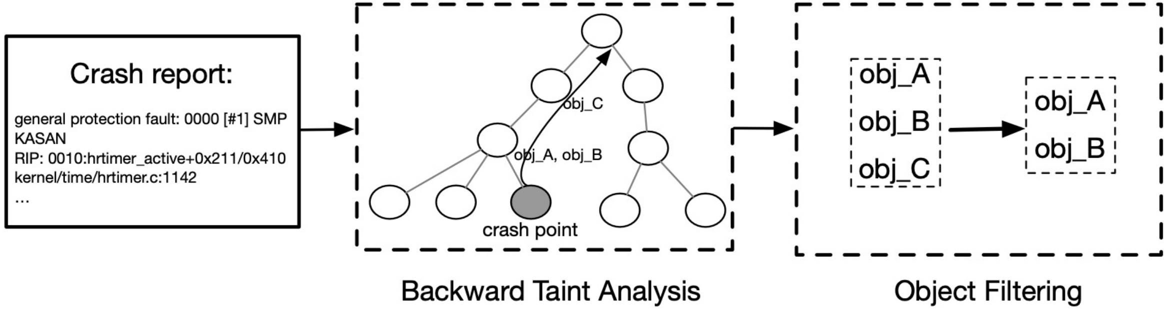
- Static analysis
  - A lot of false positives
  - No input
- Fuzzing
  - Code-coverage feedback will detour the path
  - How to restrict the fuzzing scope
  - What is the proper fuzzing scope

Finding Multiple Error Behaviors (cont.)

- Some observations
  - Linux kernel's design is [object-oriented]
  - Bugs result from incorrect usage of kernel object
  - Incorrectness propagates to different places
- Object-driven kernel fuzzing
  - Static analysis to find critical objects
  - Under-scope fuzzing based on the reachability of identified objects

Object-driven kernel fuzzing

- Static analysis to find critical objects



Object-driven kernel fuzzing (cont.)

- Static analysis to find critical objects

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

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

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

struct hrtimer

↓

struct hrtimer

↓

struct hrtimer,
struct napi_struct,
struct tun_file

Object-driven kernel fuzzing (cont.)

- Under-scope fuzzing based on *Syzkaller*
  - Instrument basic blocks involved with critical objects
  - Only inputs reaching these objects are interesting

Experiment setup

- 60 kernel bugs (2017-2021)
- Each cases comes with a patch
- 7 days for Syzkaller and our tool
- Manually categorize reports tied to the same bug

Experiment results

- Exploitability escalation
  - less likely to exploit bug (44/60)
    - 4 escalation found by Syzkaller
    - 26 escalation found by our tool, 3 error behaviors on avg.
- More exploit potential
  - likely to exploit bug (16/60)
    - Syzkaller found 1 bug has other exploitable behaviors
    - Our tool found 8 bugs have other exploitable behaviors

Takeaway

- A kernel bug could have ***M**ultiple **E**rror **B**ehaviors*
- ***MEB*** contribute to more precise exploitability estimation
- Finding ***MEB*** automatically is possible
- Utilizing kernel objects to find ***MEB*** is effective and efficient.

Zhenpeng Lin (@Markak_)

https://zplin.me

Looking for summer internship!

LINUX
SECURITY
SUMMIT