GREBE: Unveiling Exploitation Potential for Linux Kernel Bugs

Zhenpeng Lin, Yueqi Chen, Yuhang Wu, Dongliang Mu, Chensheng Yu,
Xinyu Xing, Kang Li

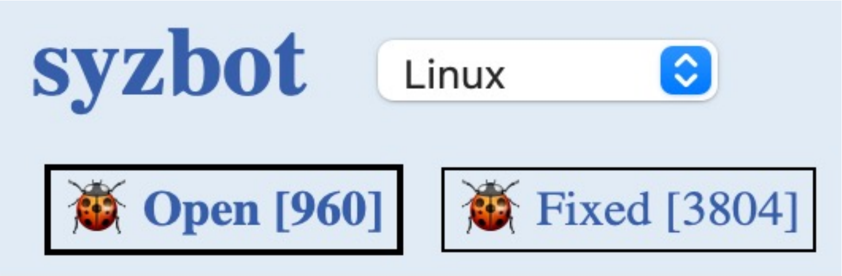
Linux Kernel is Security-Critical but Buggy

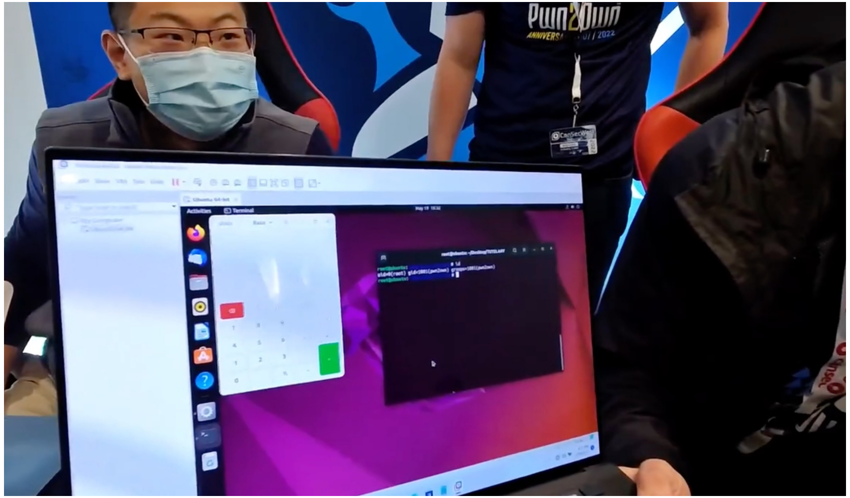
Security-Critical

- 85% smartphones run on Linux kernel
- ~39% websites are powered by Linux kernel
- etc

Buggy

- Syzbot reported ~5k bugs in past 4 years
- ~1k bugs are still unfixed
- Often gets *pwned* at Pwn2Own





Knowing Exploitability is Important but Challenging

Guide the design of hardening

- Eliminate exploit component

Promote bug fix and fix adoption

- Severe bugs not fixed in upstream
- Severe bugs fixed in upstream, unfixed in vendor's kernel

🌹 Roses are red, Violets are blue 💙 Giving leets 🧑‍💻 more sweets 🍭 All of 2022!

February 14, 2022

Posted by Eduardo Vela, Vulnerability Matchmaker

Until December 31 2022 we will pay 20,000 to 91,337 USD for exploits of vulnerabilities in the Linux Kernel, Kubernetes, GKE or kCTF that are exploitable on our test lab.

Knowing true exploitability is hard

- Kernel is complex
- Writing exploits is time-consuming

Practical Exploitability Assessment

Approximate the exploitability

- *Likely to exploit* : UAF/OOB/DF
- ***Less*** *Likely to exploit* : GPF/Null Ptr Dereference/BUG/WARN/INFO

Level	Type	Example	CVSS Score
0	Exploit Kernel	KASAN (e.g., use-after-free, double-free, out-of-bound access)	6.2
1	Terminate Process	BUG, GPF, NULL ptr dereference	5.3
2	Logging Errors	WARN, wrappers (e.g., pr_err)	1.9

Approximation May Underestimate Exploitability

- A severe bug may not show memory corruption capability
- A severe bug may only show limited memory corruption capability

A Real-World Example — CVE-2021-3715

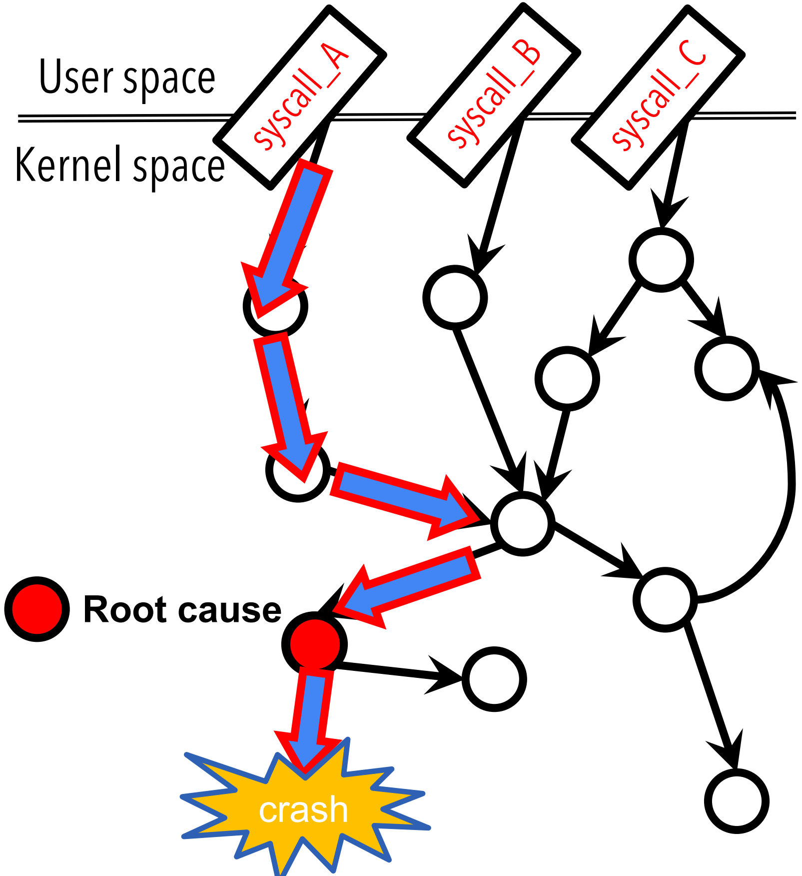
- Reported as a ***warning*** error by Syzbot
- Fixed in upstream kernel, but unfixed in some vendors’ kernel
- No CVE assigned, no discussion, no public exploit

A Real-World Example — CVE-2021-3715

- Reported as a ***warning*** error by Syzbot
- Fixed in upstream kernel, but unfixed in some vendors’ kernel
- No CVE assigned, no discussion, no public exploit
- ***UAF*** error identified by our tool — **GREBE** and being exploited by us
- Responsibly disclosed to RedHat
- RedHat notified affected vendors and *CVE assigned*

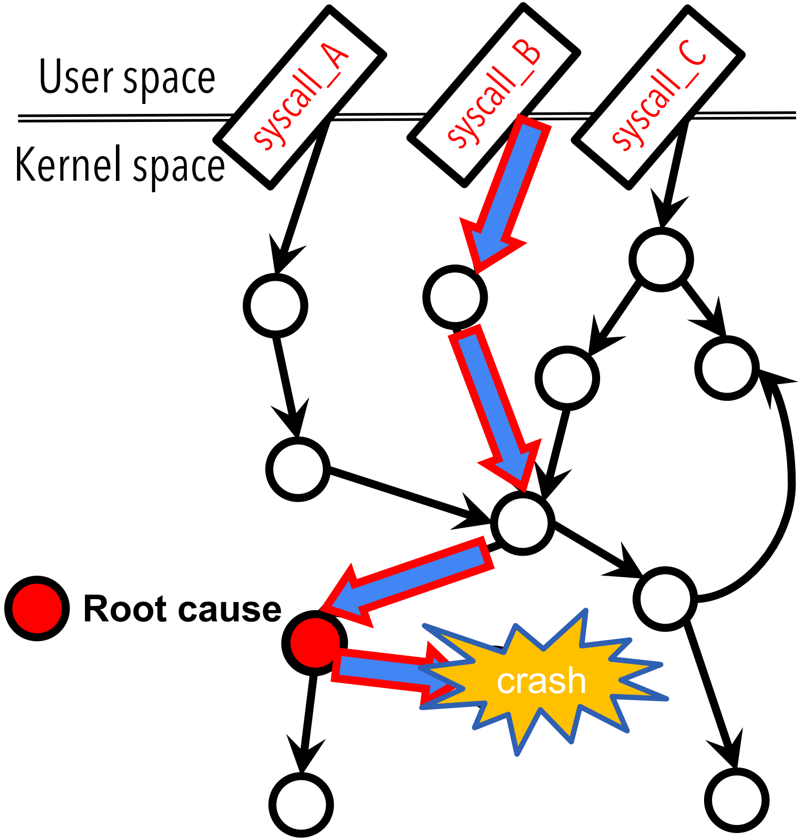
Kernel Bugs Have Multiple Error Behaviors

- CVE-2021-3715 shows warning error and UAF error.
- With the ***same*** root cause, ***different*** inputs causing ***different*** errors.



Kernel Bugs Have Multiple Error Behaviors

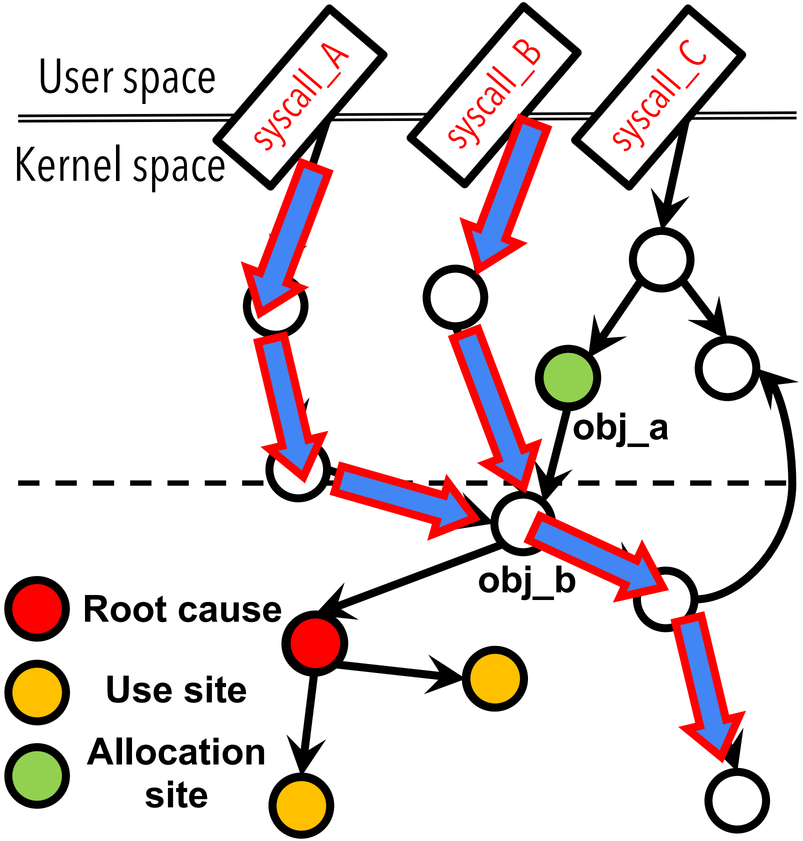
- CVE-2021-3715 shows warning error and UAF error.
- With the ***same*** root cause, ***different*** inputs causing ***different*** errors.



GREBE: An Object-driven Kernel Fuzzer

Insight

- Linux kernel implementation is object-oriented
- Operation on kernel objects are necessary to trigger the bug
- Data in kernel propagate through kernel objects



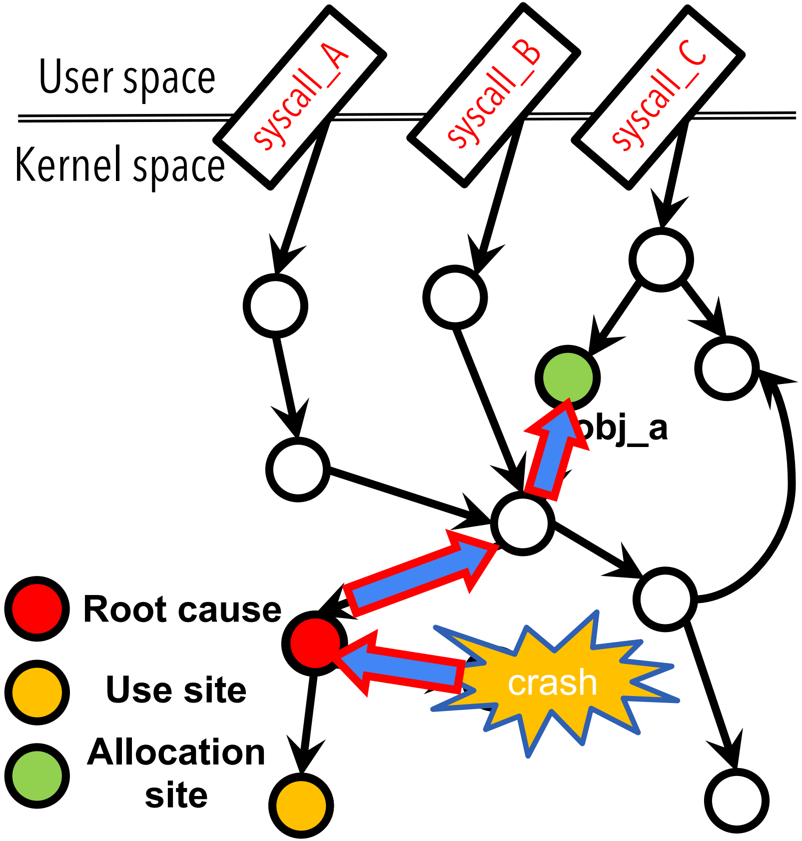
GREBE: An Object-driven Kernel Fuzzer

Insight

- Linux kernel implementation is object-oriented
- Operation on kernel objects are necessary to trigger the bug
- Data in kernel propagate through kernel objects

GREBE's solution in High-level

- Identify critical kernel objects given the bug report
- Guide the kernel fuzzing with the objects
    - set up context
    - bound fuzzing



Backward Taint Analysis to Identify Critical Objects

Taint source identification

- Kernel complains when checks unsatisfied
- Use variables in the checking conditions as taint source

```
// in drivers/vhost/vhost.c
void vhost_dev_cleanup(struct vhost_dev *dev)
{
    WARN_ON(!list_empty(&dev->work_list));
    if (dev->worker) {
        kthread_stop(dev->worker);
        dev->worker = NULL;
        dev->kcov_handle = 0;
    }
}
```

```
// source code
walk->offset = sg->offset;

// pseudo binary code after instrumentation
kasan_check_read(&sg->offset, sizeof(var));
tmp = LOAD(&sg->offset, sizeof(var)); // first access
kasan_check_write(&walk->offset, sizeof(var));
STORE(tmp, &walk->offset); // second access
```

Backward Taint Analysis to Identify Critical Object

Taint propagation

- Taint to parent structure variables
- Taint to loop counter

```
hrtimer_cancel(&tfile->napi->timer)

hrtimer_try_to_cancel(timer)

base = READ_ONCE(timer->base)
```

```
int func() {
    for (int i=0; i<vuln->size; i++) {
        // buffer overflow in vuln->buff
        vuln->buff[i] = src[j++];
    }
}
```

Backward Taint Analysis to Identify Critical Object

Taint sink

- The definition of a variable
- Syscall entry, or interrupt handler

Backward Taint Analysis to Identify Critical Object

Taint sink

- The definition of a variable
- Syscall entry, or interrupt handler

Object filtering

- Object popularity ranking
- Filter out “popular” objects
- More details in our paper

Object-driven Kernel Fuzzing

- Instrument basic blocks involved with critical objects
- Maximize object coverage instead of code coverage

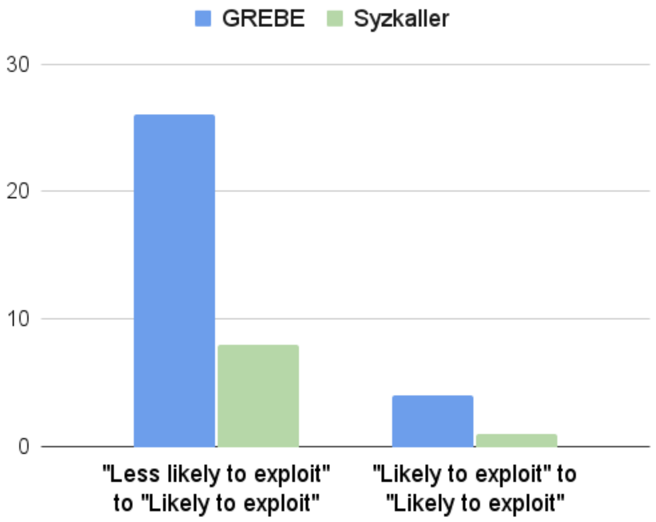
Experiment

Setup

- Used 60 kernel bugs (2017-2021)
- Compared with Syzkaller
- Manually triage the results

Results

- Exploitability escalation
  - From "*less likely to exploit*" to "*likely to exploit*"
  - GREBE (26) *vs.* Syzkaller (4)
- More exploit potential
  - From one "*likely to exploit*" to more "*likely to exploit*"
  - GREBE (8) *vs.* Syzkaller (1)



Takeaway

- A kernel bug could have Multiple Error Behaviors (***MEB***).
- Exposing ***MEB*** contributes to more precise exploitability estimation.
- Utilizing kernel objects to find ***MEB*** is effective and efficient.

GREBE is available at: https://github.com/Markakd/GREBE

zplin@u.northwestern.edu

https://zplin.me