LINUX SECURITY SUMMIT

SyzScope

Revealing High-Risk Security Impacts of Fuzzer-Exposed Bugs in Linux kernel

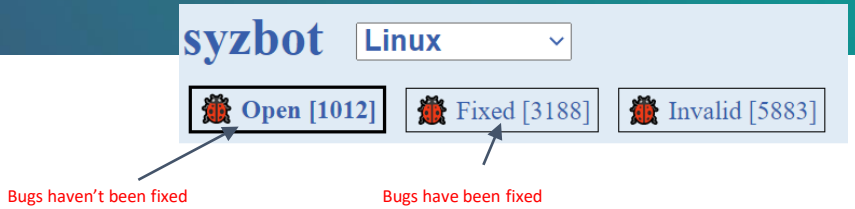
Xiaochen Zou

University of California, Riverside

@ETenal7

THE LINUX FOUNDATION

Background



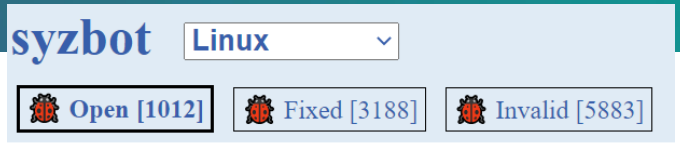
Focus of our study: Linux kernel bugs

Fuzzer: Syzkaller

Continuous fuzzing platform: Syzbot

- Memory bug ~1000
- WARNING + INFO ~900
- GPF + invalid ptr defer ~500

Background



Syzbot discovered mainly 8 types of bugs

- KASAN bug: use-after-free, out-of-bounds, double free
- KCSAN bug: data-race bug
- KMSAN bug: uninitialized use
- UBSAN: undefined behavior
- Kernel assertion bug: WARNING
- Kernel assertion bug: INFO
- Kernel assertion bug: BUG
- General protection fault:

KASAN: use-after-free Read in sctp_auth_free

KCSAN: data-race in bpf_lru_pop_free / bpf_lru_push_free

KMSAN: uninit-value in bpf_iter_prog_supported

UBSAN: array-index-out-of-bounds in ieee80211_del_key

WARNING in io_uring_cancel_task_requests

INFO: task hung in tcf_action_init_1

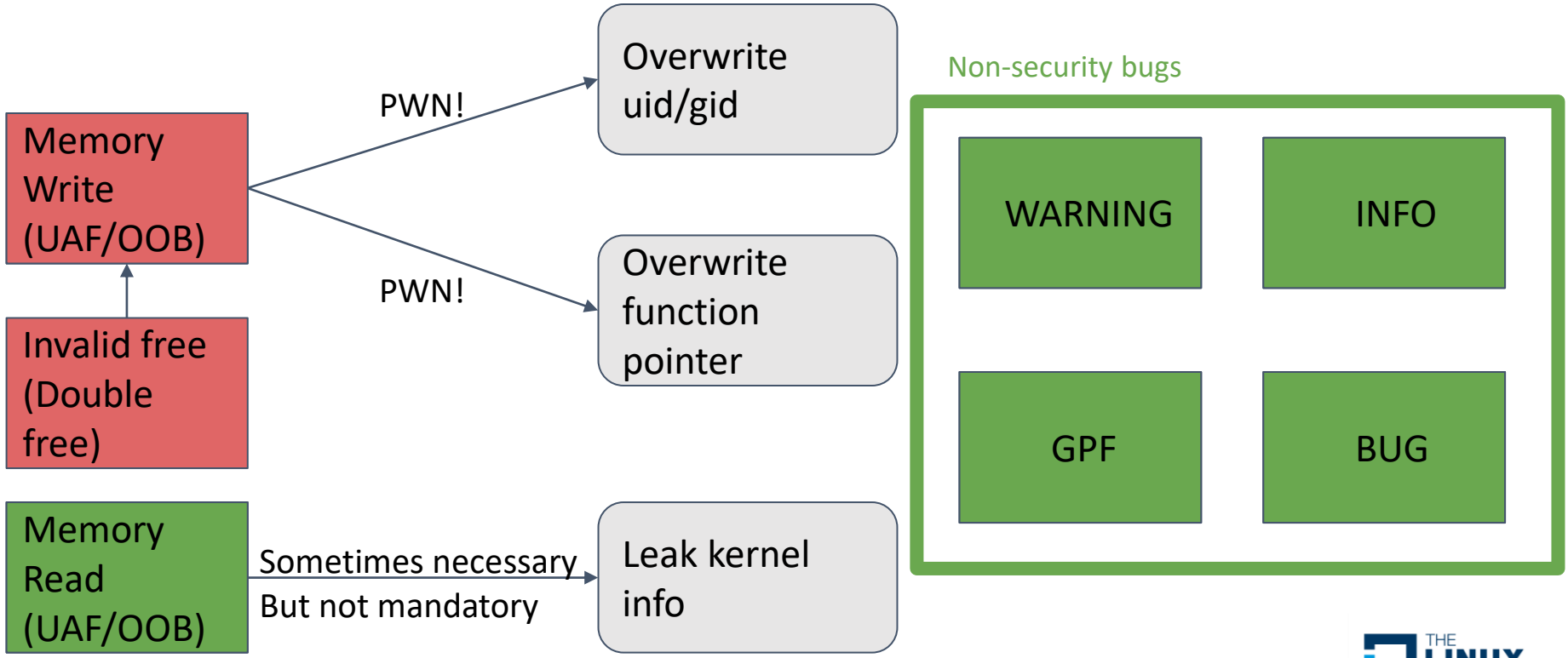
BUG: receive list entry not found for dev vxcan1, id 003, mask C000…

general protection fault in strncasecmp

Bug type	Bug Impact
Sanitizer: KASAN	use-after-free (UAF) out-of-bounds (OOB) double free
Sanitizer: KCSAN	data race
Sanitizer: KMSAN	uninitialized use
Sanitizer: UBSAN	variety*
Kernel: WARNING / INFO / BUG	Assertions on any unexpected behaviors
Kernel: GPF	corrupted pointer dereference

* UBSAN (Undefined Behavior Sanitizer) can detect a variety of impacts

Bug's security impact



Motivation - Too many bugs to fix

Imagine a WARNING bug that can exploit the kernel but somehow we lower its priority for fixing due to misunderstanding its severity

UAF Read
63 days

UAF Write
37 days

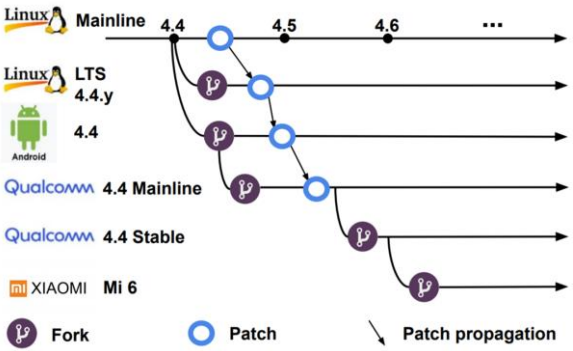
OOB Read
89 days

OOB Write
29 days

Prioritization on more serious bugs

THE LINUX FOUNDATION

Motivation - Too many patches to port



CVE-2019-2215. a UAF read and fixed in Linux upstream in 52 days. Unfortunately, it took over a year for the patch to propagate to downstream

Patch delay between upstream and Ubuntu of OOB write bugs is 59 days

Patch delay between upstream and Ubuntu of WARNING errors is 83 days

Because of misunderstanding the severity of a potential high-risk bug, a downstream kernel may remain unpatched for quite a long time or forever

Syzbot overwhelm downstream maintainers with a huge number of patches.

Questions to answer

- **Are all seemingly low-risk bugs actually low-risk?** No
- **Do bug reports always reveal the real impact of bugs?** No
- **Can we convert a seemingly low-risk bug to a high-risk bug automatically?** Yes

Goal and non-goal

Goal: SyzScope aims to reveal high-risk impacts of a seemingly low-risk bug

Non-goal: SyzScope does NOT aim to produce an end-to-end exploit automatically.

KOOBE: Towards Facilitating Exploit Generation of Kernel Out-Of-Bounds Write Vulnerabilities

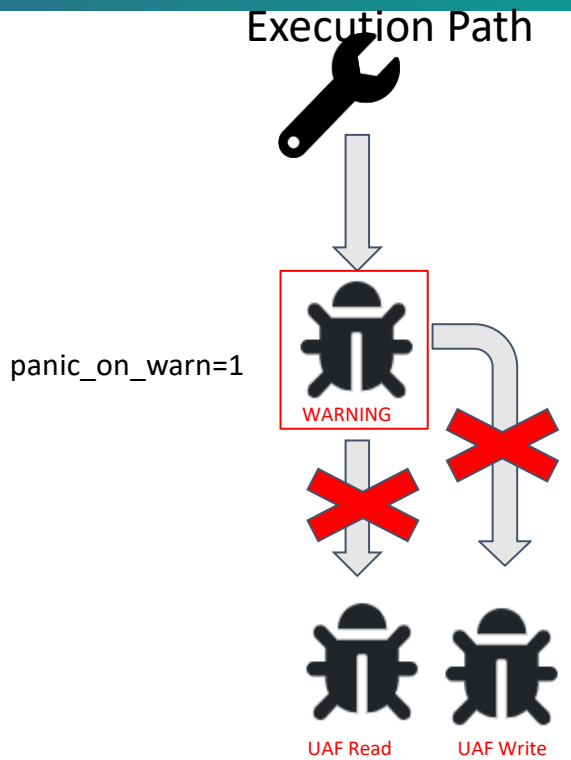
FUZE: Towards Facilitating Exploit Generation for Kernel Use-After-Free Vulnerabilities

Insight

1. Fuzzing only present the first impact of a bug

2. Bug's title on syzbot only shows the first impact instead of the most risky impact

WARNING in qp_broker_alloc



Insight

High-risk impacts:

- UAF/OOB Write
- Invalid Free
- Control flow hijacking
- Arbitrary/Constrained value write
- Arbitrary/Constrained address write

Follow up impacts

Low-risk impacts:

- UAF/OOB Read
- WARNING/INFO
- General protection fault
- BUG
- All other non-security bugs

Blue means this impact **can** be detected by fuzzing

Red means this impact **can not** be detected by fuzzing

Motivating example

```
1  static void tcindex_free_perfect_hash(struct
       tcindex_data *cp) {
2    for (int i = 0; i < cp->hash; i++)
3      tcf_exts_destroy(&cp->perfect[i].exts);
4    kfree(cp->perfect);
5  }
6
7  void tcf_exts_destroy(struct tcf_exts *exts) {
8    if (exts->actions)
9      tcf_action_destroy(exts->actions);
10 }
11
```

KASAN: slab-out-of-bounds Read in tcf_exts_destroy

cp->hash is bigger than the length of **cp->perfect** arrary

cp->perfect[i] is out-of-bound when i is bigger than length of cp->perfect

OOB read captured by KASAN

Motivating example

```
1  static void tcindex_free_perfect_hash(struct
       tcindex_data *cp) {
2    for (int i = 0; i < cp->hash; i++)
3      tcf_exts_destroy(&cp->perfect[i].exts);
4    kfree(cp->perfect);
5  }
6
7  void tcf_exts_destroy(struct tcf_exts *exts) {
8    if (exts->actions)
9      tcf_action_destroy(exts->actions);
10 }
11
```

```
12 int tcf_action_destroy(struct tc_action *
       actions[]) {
13   struct tc_action *a;
14   for (i = 0; i < TCA_ACT_MAX_PRIO &&
         actions[i]; i++) {
15     a = actions[i];
16     actions[i] = NULL;   // AAW
17     ret = tcf_action_cleanup(a);
18   }
19 }
```

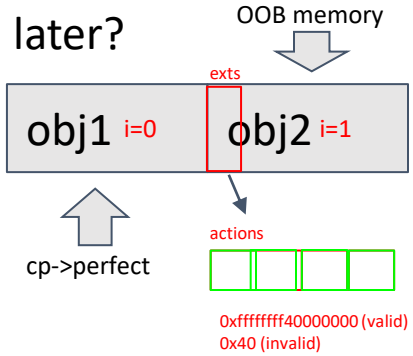
Kernel panic or action[i] == 0

AAW = Arbitrary address

KASAN: slab-out-of-bounds Read in tcf_exts_destroy

What if we let fuzzing **continue** instead of stop at the first bug report?

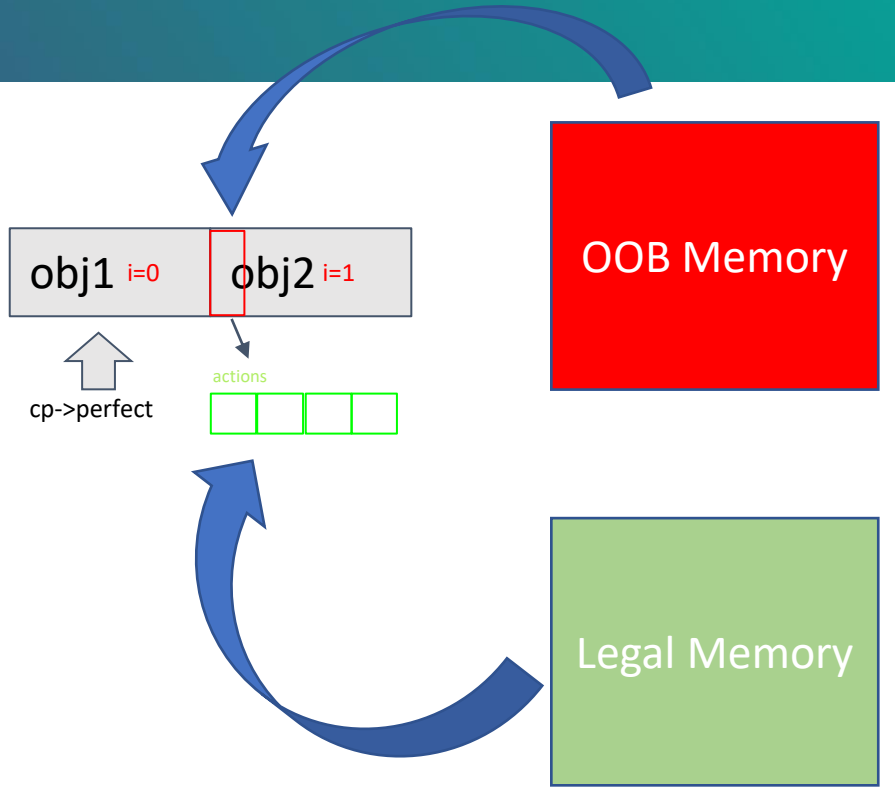
Can we capture the AAW write later?

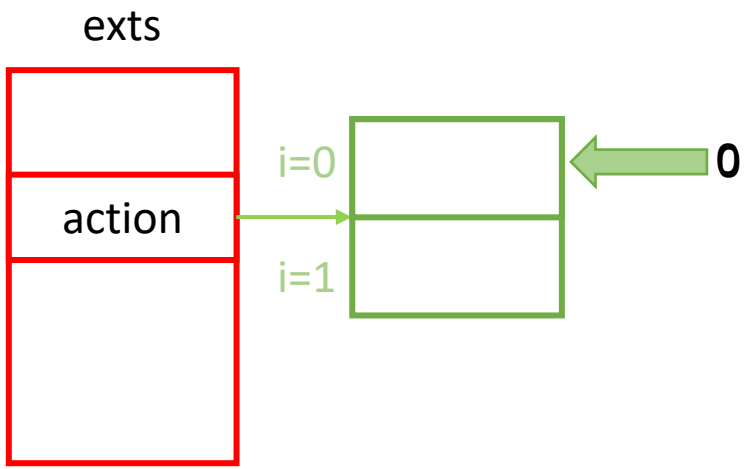


THE LINUX FOUNDATION

Motivating example

```
12  int tcf_action_destroy(struct tc_action *
        actions[]) {
13    struct tc_action *a;
14    for (i = 0; i < TCA_ACT_MAX_PRIO &&
          actions[i]; i++) {
15      a = actions[i];
16      actions[i] = NULL;   // AAW
17      ret = tcf_action_cleanup(a);
18    }
19  }
```





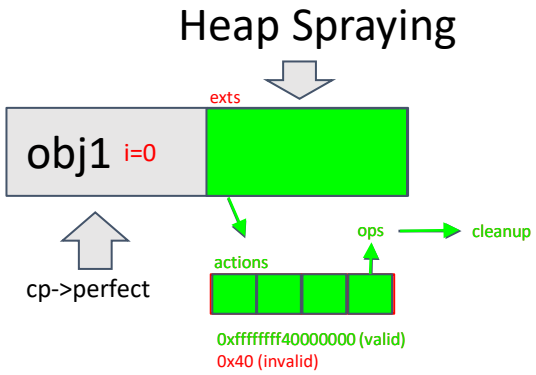
Motivating example

```
1  static void tcindex_free_perfect_hash(struct
       tcindex_data *cp) {
2    for (int i = 0; i < cp->hash; i++)
3      tcf_exts_destroy(&cp->perfect[i].exts);
4    kfree(cp->perfect);
5  }
6
7  void tcf_exts_destroy(struct tcf_exts *exts) {
8    if (exts->actions)
9      tcf_action_destroy(exts->actions);
10 }
11
```

KASAN: slab-out-of-bounds Read in tcf_exts_destroy

```
21 static void tcf_action_cleanup(struct
       tc_action *p) {
22   if (p->ops->cleanup)
23     p->ops->cleanup(p);   // FPD
24 }
```

```
12 int tcf_action_destroy(struct tc_action *
       actions[]) {
13   struct tc_action *a;
14   for (i = 0; i < TCA_ACT_MAX_PRIO &&
         actions[i]; i++) {
15     a = actions[i];
16     actions[i] = NULL;   // AAW
17     ret = tcf_action_cleanup(a);
18   }
19 }
```



Assumption: UAF/OOB objects are controlled by attackers using heap spraying

Problem: UAF/OOB objects cannot be controlled by fuzzer alone

Problem: Fuzzer was blocked by kernel panic or complicated constraints

```
11
12  int tcf_action_destroy(struct tc_action *
        actions[]) {
13    struct tc_action *a;
14    for (i = 0; i < TCA_ACT_MAX_PRIO &&
          actions[i]; i++) {
15      a = actions[i];
16      actions[i] = NULL;    // AAW
17      ret = __tcf_idr_release(a);
18    }
19  }
```

Take the true branch

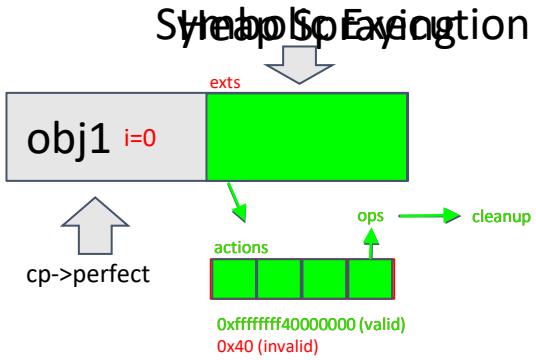
Take the false branch

actions+i is a valid memory address OR actions[i] == 0

actions+i must be a valid memory address AND actions[i] != 0

```
BUG: kernel NULL pointer dereference, address: 0000000000000000
#PF: supervisor instruction fetch in kernel mode
#PF: error_code(0x0010) - not-present page
PGD 1d5b1067 P4D 1d5b1067 PUD 13a4d067 PMD 0
Oops: 0010 [#1] PREEMPT SMP KASAN
```

Symbolic execution is a program analysis technique that uses symbolic value instead of a concrete value during program execution.



Two modes of operations - Open vs Fixed bugs

Upstream - open bugs

SyzScope package **static analysis** and **symbolic execution** to explore high-risk impacts behind the low-risk impact.

Downstream - fixed bugs

SyzScope package **fuzzing**, **static analysis** and **symbolic execution** to explore high-risk impacts behind the low-risk impact.

Workflow - Fixed bugs & Open bugs

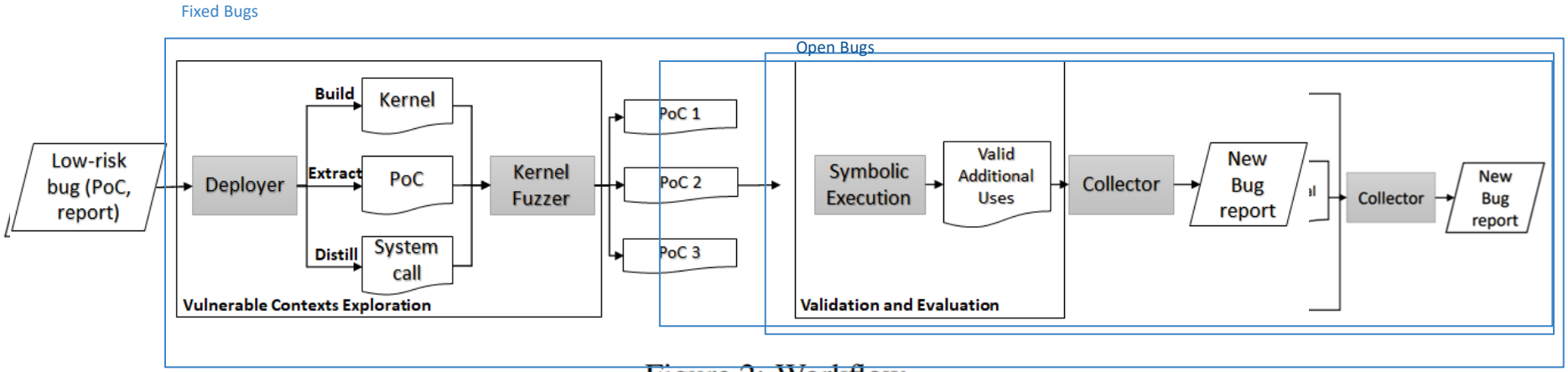
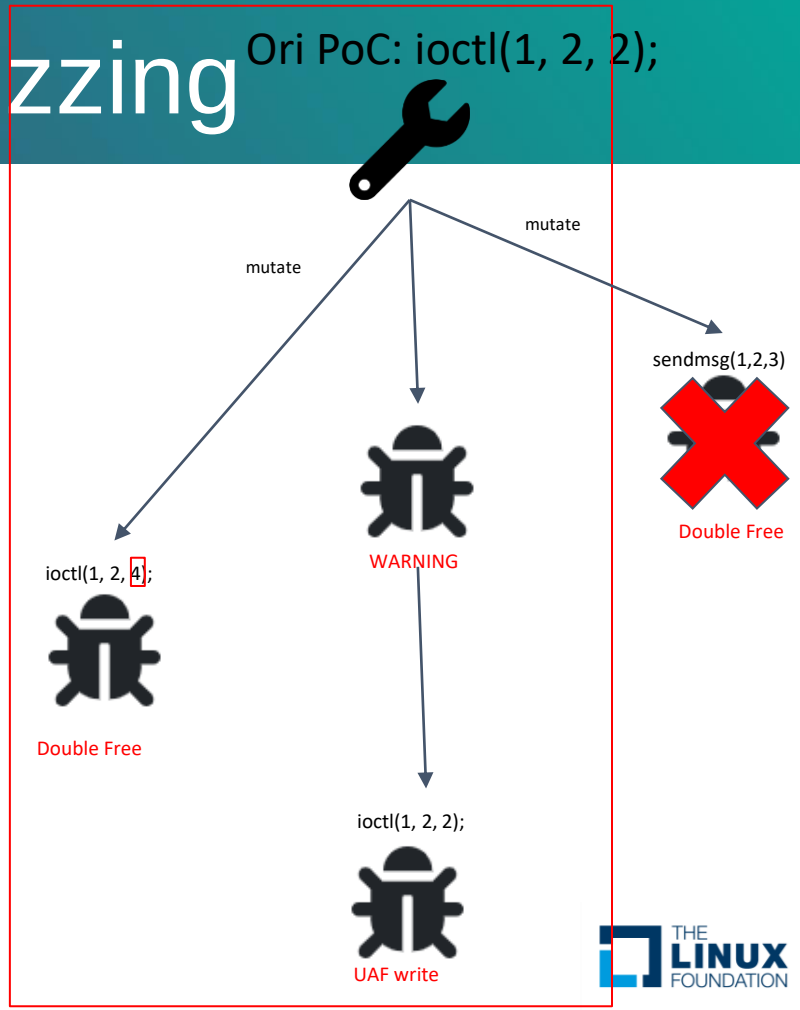


Figure 2: Workflow

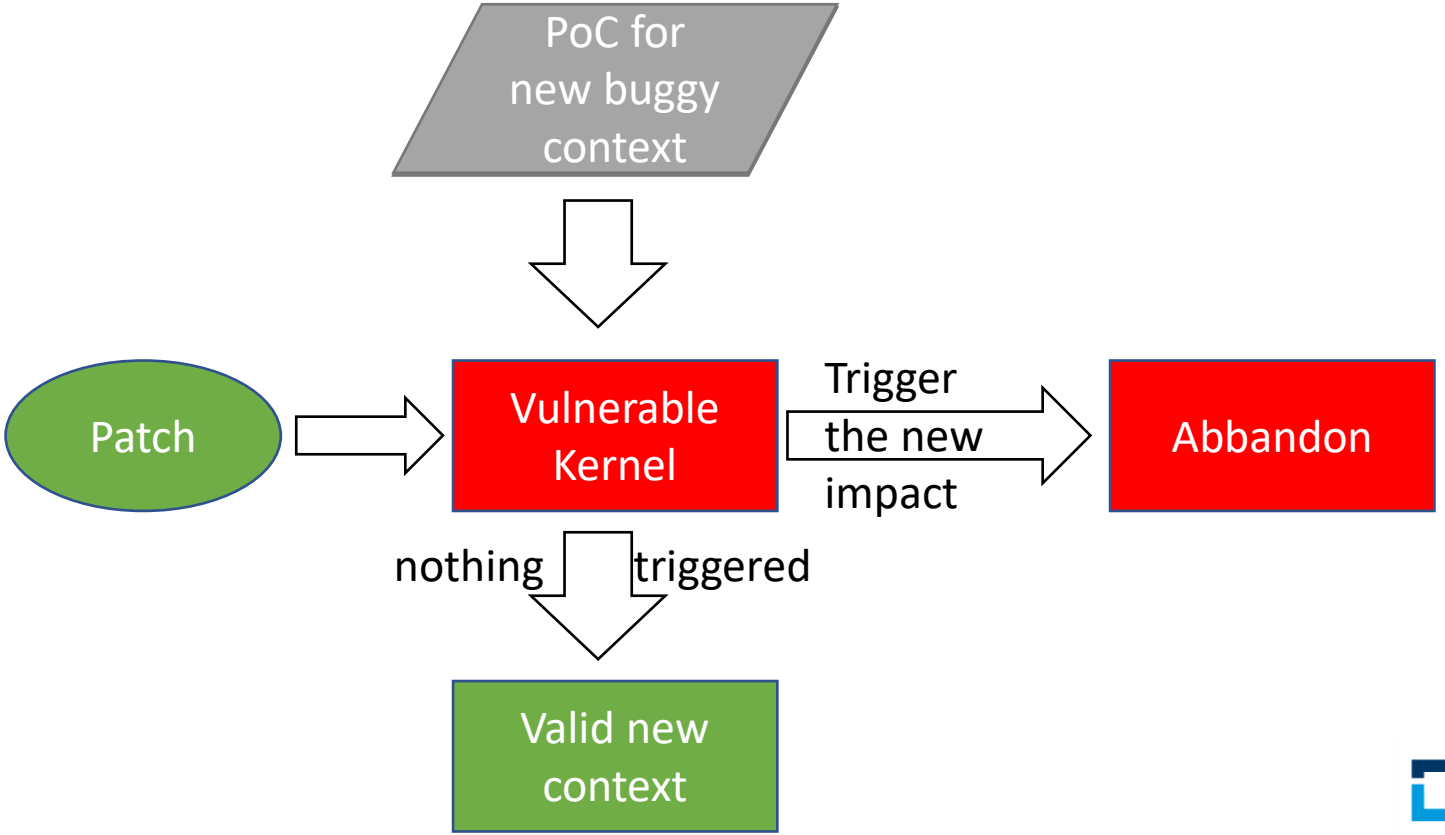
Fuzzing - Restricted fuzzing

1. Provides more UAF/OOB contexts to symbolic execution
2. Does not stop at the first impact
3. Conservative mutating strategy

Fuzzing is only enabled for cases with patches



Fuzzing - New contexts verification

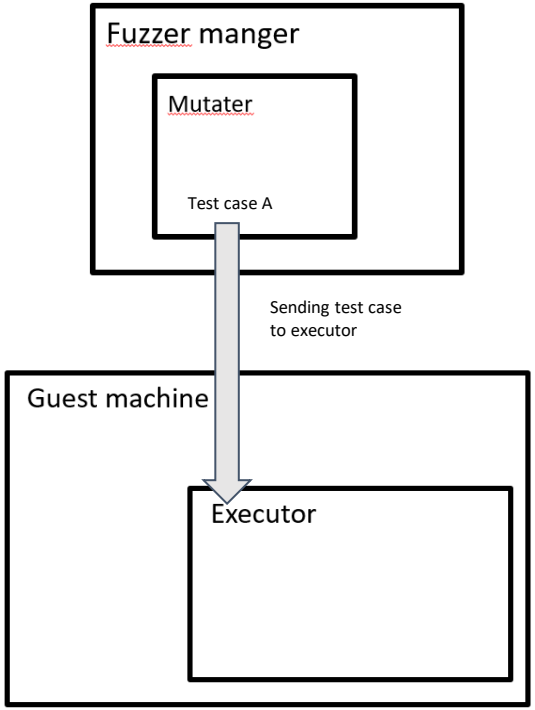


Fuzzer - Impact aware fuzzing

We use Syzkaller as our fuzzer

1. Let Syzkaller catch multiple bug impacts along one execution path and pick up the most high-risk one (eg. WARNING->UAF Write)

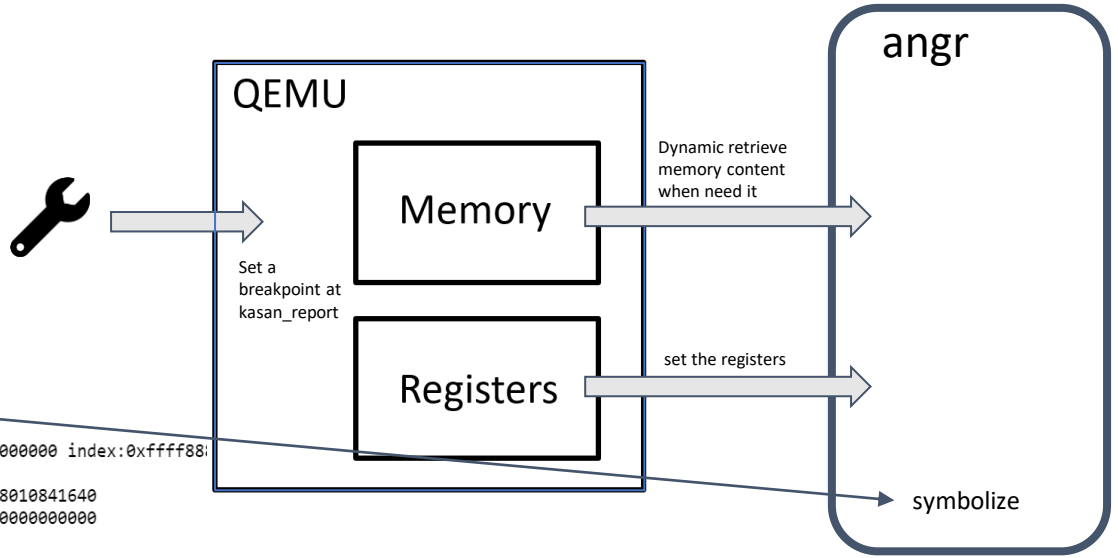
2. Not only does it have code coverage feedback, but we also added impact feedback.



We also consider a seed that triggers a new impact as an interesting candidate, and further mutate it

Symbolic Execution - Architecture

Our symbolic execution starts at bug's first KASAN report




```
The buggy address belongs to the object at ffff8881437d5700
 which belongs to the cache kmalloc-64 of size 64
The buggy address is located 16 bytes inside of
 64-byte region [ffff8881437d5700, ffff8881437d5740)
The buggy address belongs to the page:
page:000000003e519aab refcount:1 mapcount:0 mapping:0000000000000000 index:0xffff88
flags: 0x57ff00000000200(slab)
raw: 057ff00000000200 ffffea000511ff88 ffffea00051d0a48 ffff888010841640
raw: ffff8881437d5b00 0000000000200019 00000001ffffffff 0000000000000000
page dumped because: kasan: bad access detected

Memory state around the buggy address:
 ffff8881437d5600: 00 00 00 00 00 fc fc fc fc fc fc fc fc fc fc fc
 ffff8881437d5680: 00 00 00 00 00 fc fc fc fc fc fc fc fc fc fc fc
>ffff8881437d5700: fa fb fb fb fb fb fb fb fc fc fc fc fc fc fc fc
                  ^
 ffff8881437d5780: 00 00 00 00 00 00 fc fc fc fc fc fc fc fc fc fc
 ffff8881437d5800: 00 00 00 00 00 00 fc fc fc fc fc fc fc fc fc fc
==================================================================
```

THE LINUX FOUNDATION

Symbolic Execution - Impacts identification

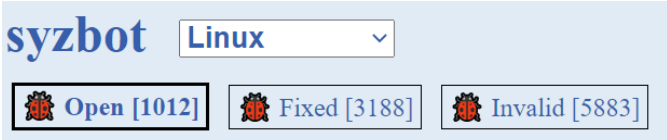
High-risk impacts:

- UAF/OOB Write → *mov [rbx], rax*
- Invalid Free → *mov rdi, xxx*
call _kfree
- Control flow hijacking → *call rax*
- Arbitrary/Constrained value write → *mov [rbx], rax*
- Arbitrary/Constrained address write → *mov [rbx], rax*

Dataset

Syzbot discovered mainly 8 types of bugs

- KASAN bug: use-after-free, out-of-bounds, double free
- Kernel assertion bug: WARNING
- Kernel assertion bug: INFO
- Kernel assertion bug: BUG
- General protection fault: GPF



Bug type	Bug Impact
Sanitizer: KASAN	use-after-free (UAF) out-of-bounds (OOB) double free
Sanitizer: KCSAN	data race
Sanitizer: KMSAN	uninitialized use
Sanitizer: UBSAN	variety*
Kernel: WARNING / INFO / BUG	Assertions on any unexpected behaviors
Kernel: GPF	corrupted pointer dereference

* UBSAN (Undefined Behavior Sanitizer) can detect a variety of impacts

Evaluation

Experiment setup

- 1173 valid bugs
- All experiment are conducted in Ubuntu-18.04 with 1TB memory and Intel(R) Xeon(R) Gold 6248 20 Core CPU @ 2.50GHz * 2
- 3 hours kernel fuzzing, and 4 hours symbolic execution

Overall results

		High-risk bug found	OOB/UAF write	Arbitrary address write	Constrained address write	Arbitrary value write	Constrained value write	Control flow hijacking	Invalid Free
Fixed	**GPF and BUG**	9	164	5	26	19	25	1	3
	WARNING and INFO	19	67	10	13	51	27	10	2
	UAF and OOB Read	85	1749	142	113	531	192	28	1
Open	**GPF and BUG**	4	2	0	0	2	0	1	0
	WARNING and INFO	8	151	19	2	13	15	11	0
	UAF and OOB Read	22	242	2	21	38	52	0	2
	Total	147	2375	178	175	654	311	51	8

Evaluation - Fuzzing

Fuzzing alone discovered 61 high-risk bugs (53%) with 167 impacts (5%).
All impacts found by fuzzing are UAF/OOB write and invalid free.

- The average number of impacts
  fixed section (with patches) 27.9 vs open section (without patches) 16.9:
  This is because the open bugs did not go through the fuzzing.
- Buggy contexts in fixed section : 1.37 contexts
  Buggy context in open section: 1 context

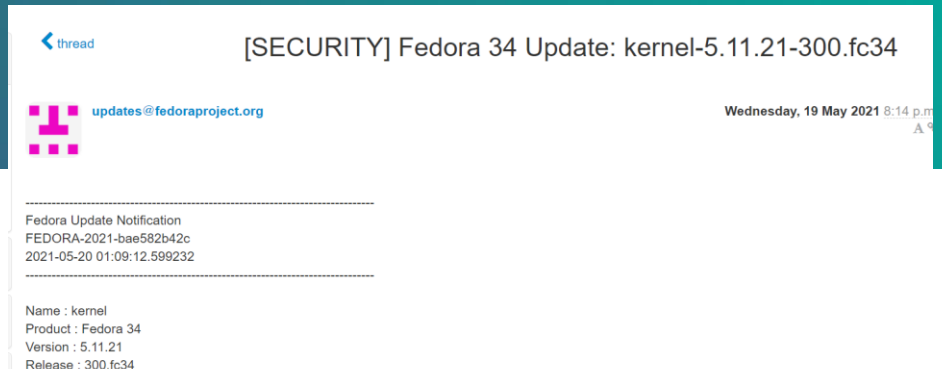
Evaluation - Symbolic execution

Initial bug impact	High-risk bugs by fuzzing	Primitives found by fuzzing	Extra high-risk bugs by S&S	Extra primitives found by S&S
GPF and BUG	6	6	3	237
WARNING and INFO	10	29	9	151
UAF and OOB Read	45	132	40	2624

S&S = Static analysis and symbolic execution

- 61 bugs that were turned into high-risk by fuzzing alone, the rest 52 bugs in fixed section can only be turned into high-risk by symbolic execution.
- Symbolic execution discovered 95% impacts including control flow hijacking, arbitrary address write.

Disclosure


thread

[SECURITY] Fedora 34 Update: kernel-5.11.21-300.fc34

updates@fedoraproject.org

Wednesday, 19 May 2021 8:14 p.m

Fedora Update Notification
FEDORA-2021-bae582b42c
2021-05-20 01:09:12.599232

Name : kernel
Product : Fedora 34
Version : 5.11.21
Release : 300.fc34

We submitted 32 high-risk bugs since
maintainers and 8 of them have been
still pending for respor

- CVE-2021-33034
- CVE-2021-33033
- CVE-2019-25044
- CVE-2020-36386
- CVE-2020-36385
- CVE-2018-25015
- CVE-2020-36387
- CVE-2019-25045

Affected Packages and Issued Red Hat Security Errata

Platform	Package	State
Red Hat Virtualization 4	redhat-virtualization-host moderate	Affected
Red Hat Enterprise Linux 7	kernel-rt	Affected
Red Hat Enterprise Linux 8	kernel-rt	Affected
Red Hat Enterprise Linux 5	kernel moderate CVSS v3: 7.7 See score details	Not affected
Red Hat Enterprise Linux 6	kernel	Not affected
Red Hat Enterprise Linux 7	kernel	Affected
Red Hat Enterprise Linux 8	kernel	Affected

Publish

Our paper will appear on USENIX security 2022.

THE LINUX FOUNDATION

Q&A

Thank you for listening, SyzScope is now open source on:

https://github.com/seclab-ucr/SyzScope.git

LINUX SECURITY SUMMIT