

What Bugs **Matter**?

Continuous Linux Kernel Bug Monitoring

SPEAKER

Yuan Tan

CONTRIBUTORS

Nan Li · Ren Wei · Yang Yang · Ao Zhou · Luxing Yin · Xin Liu*

bugtracker.nebusec.ai

continuous kernel bug monitoring & triage

Kernel bug reports in the **AI era**

Finding and reporting bugs got cheap. Verifying them did not.

- ◇ **Volume.** AI patch reviewers and LLM agents now report bugs at a scale fuzzers and humans never did.
- ◇ **Plausible, not proven.** Reports are well-argued and confident — but many describe code that no running kernel can reach.
- ◇ **Impact unclear.** A root-only corner case and an unprivileged privilege escalation read almost the same in a report.
- ◇ **The cost of checking** — reproduce, judge scope, judge severity — is still paid by hand, report by report.

The question is no longer “**are there bugs?**” — it's “**which of these bugs matter?**”

llm-report.txt

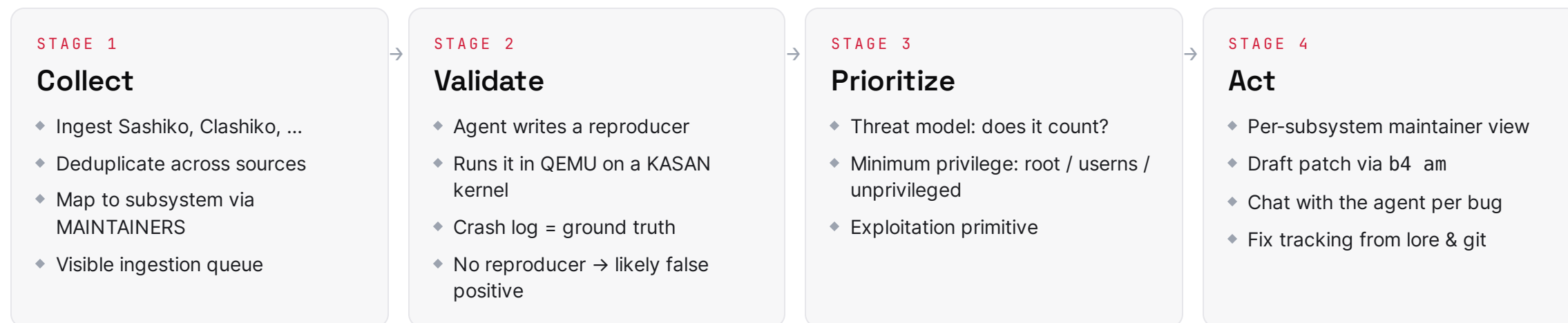
```
# AI review of [PATCH net-next] foo: ...
"Potential use-after-free in foo_release():
 the refcount may drop to zero while the
 object is still on the pending queue."

# Looks convincing. But:
? Can a running kernel reach foo_release()?
? Can an unprivileged user trigger it?
? Has someone already reported or fixed it?
? Does it actually matter?

-----
Someone has to check – by hand.
```

Let's build a pipeline: from raw reports to **actions**

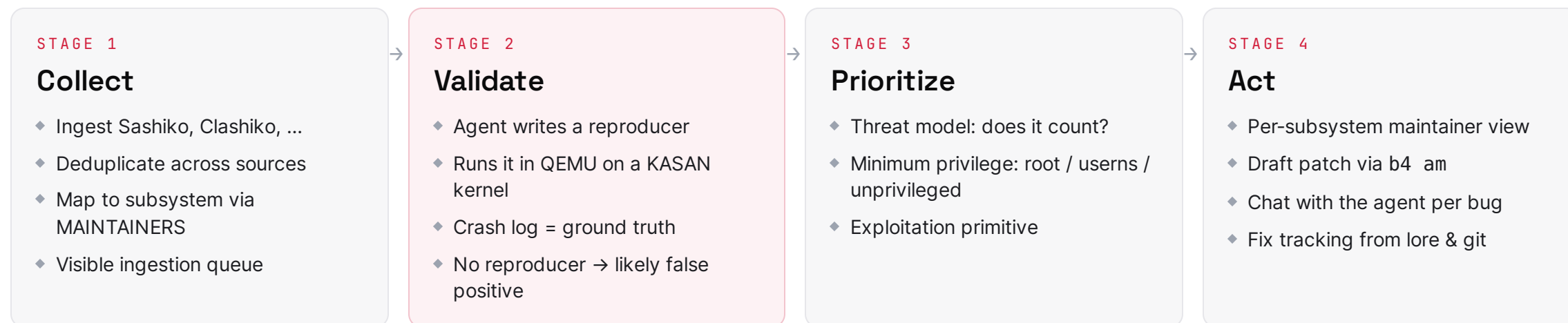
A repeatable workflow: collect reports, validate bugs, prioritize impact and act.



Every stage runs automatically. Humans step in only where judgment is needed — bugs with unusual trigger conditions, and the final patch.

Let's build a pipeline: from raw reports to **actions**

A repeatable workflow: collect reports, validate bugs, prioritize impact and act.



Every stage runs automatically. Humans step in only where judgment is needed — bugs with unusual trigger conditions, and the final patch.

Plausible ≠ Real

For every AI-generated report, a maintainer has to answer four questions before it is worth their time.

CHALLENGE 1

Unreachable

The bug can't actually be triggered; the path is dead.

CHALLENGE 2

Out of threat model

Only fires under assumptions no real attacker has.

CHALLENGE 3

Severity

Root vs. namespace vs. unprivileged: very different risk.

CHALLENGE 4

Unusual trigger conditions

Real, but only under odd, demanding conditions — an experienced developer has to decide if it's worth fixing.

Make the agent **crash the kernel**

- ◇ Run the agent inside **QEMU with KASAN** and kernel debug checks enabled, and let it attempt to trigger the bug for real.
- ◇ Any **kernel panic report** — a KASAN splat, a `BUG()`, or a hard panic — is our ground-truth signal that the bug genuinely fired.
- ◇ No crash, no confirmed bug.

The lesson that makes this work. A state-of-the-art agent already reproduces real kernel bugs from the patch alone, reliably enough that a reproducer is a trustworthy signal, and a persistent **failure** to reproduce is signal too.

Empirical basis on the next slide: 100 real-world KernelCTF vulnerabilities.

qemu · KASAN

```
$ ./repro_attempt_3
[ 12.441] repro: racing close() against queue
=====
BUG: KASAN: slab-use-after-free in foo_release
Read of size 8 at addr ffff888012a4c1e8
   by task repro/214
Call Trace:
  foo_release+0x1a4/0x2c0
  __fput+0x2f1/0xb70
  task_work_run+0x16a/0x260
=====
→ verified: REAL · crash log attached
```

Agents already reproduce real kernel bugs

Reproducing real kernel bugs is now something strong models do well — up to **95%** on the standard KernelCTF benchmark.

- ◇ Public studies set the floor: **over 50%** of 100 KernelCTF vulnerabilities reproduced from the patch alone, fully autonomous.

So reproduction becomes a **filter**: a bug the agent can trigger is confirmed real; one it repeatedly can't is treated as a **false positive** and dropped.

— REPRODUCTION · KERNELCTF

K-Repro · patch input

56%

GPT-5.1 Codex Max · 2026¹

THIS PLATFORM

Reproducer agent · bug-report input

95%

GPT-5.4 · 2026

¹ Baseline: Pu, Li, Liang, Cox, Wu, Shehada, Srivastav, Qian. *Patch-to-PoC: A Systematic Study of Agentic LLM Systems for Linux Kernel N-Day Reproduction*, arXiv:2602.07287 (2026).

A crash is not always a vulnerability

Whether a trigger “counts” depends entirely on the attacker model.

— CASE · OOB WRITE IN GET_OPTIONS()

```
lib/cmdline.c · KASAN

# boot cmdline: st=0-2147483647
# range count wraps INT_MAX → INT_MIN
# ints + i now points before the array
-----
BUG: unable to handle page fault
      for address: fffffffd...
RIP: get_option+0x1a/0x90
Call Trace: get_options ← st_setup
Kernel panic - not syncing
→ verified: real OOB write
```

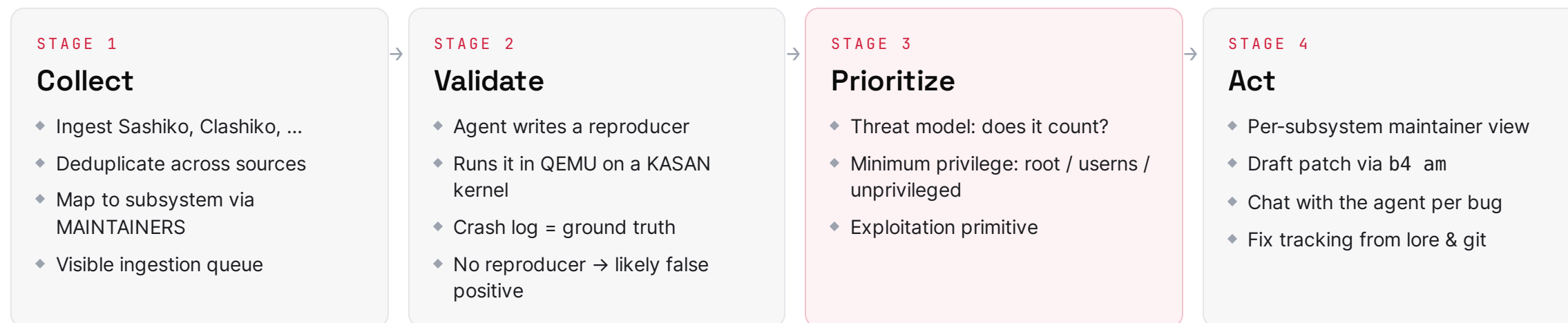
— REAL — BUT OUT OF SCOPE

- ◇ A genuine **out-of-bounds write**, crashing under KASAN — present since **v2.6.20**.
- ◇ It only fires if you **modify the boot command line** — not a runtime, unprivileged trigger.
- ◇ Whoever can set your boot args **already owns the machine** — so this crash **doesn't count**.

This is where verification stops. A reproducer proves a bug is **real and reachable**; it can't tell you whether it's **in scope**. Scope is a policy we set **in the prompt** — verification doesn't settle it.

Let's build a pipeline: from raw reports to **actions**

A repeatable workflow: collect reports, validate bugs, prioritize impact and act.



Every stage runs automatically. Humans step in only where judgment is needed — bugs with unusual trigger conditions, and the final patch.

Exploit primitives change **priority**

CVE-2026-43502 · ZcopyReaper

— THE PRIMITIVE

```
net/rds/message.c · pre-fix

bool zcopy = false;
// ...
if (!zcopy)
    __free_page(sg_page(&rm->data.op_sg[i]));
else
    put_page(sg_page(&rm->data.op_sg[i]));
```

Early send failure frees a pinned page through the wrong path.

Initial assessment by openSUSE

DoS

Identified primitive

Invalid free

Actual impact

Local privilege escalation

A bug's **actual impact** can be far more dangerous than it first appears.

Sources: [pre-fix code](#) · [our exploit](#) · [NebuSec disclosure](#) · [earlier availability-only assessment](#).

Grade severity by **privilege** and **exploit primitive**

We rate each bug on two axes: how much privilege it takes to trigger, and what capability it grants.

— WHO CAN TRIGGER IT · PRIVILEGE FLOOR

- ◇ Get a **root reproducer** first, then **step privilege down**, level by level.
- ◇ The **lowest floor** that still triggers becomes the reported severity.

L0	root / CAP_SYS_ADMIN	start	BASELINE
L1	user namespace	↓ step down	ELEVATED
L2	unprivileged user	↓ step down	CRITICAL

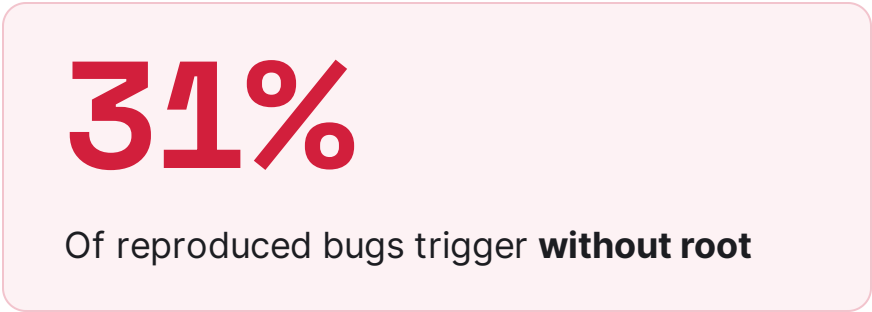
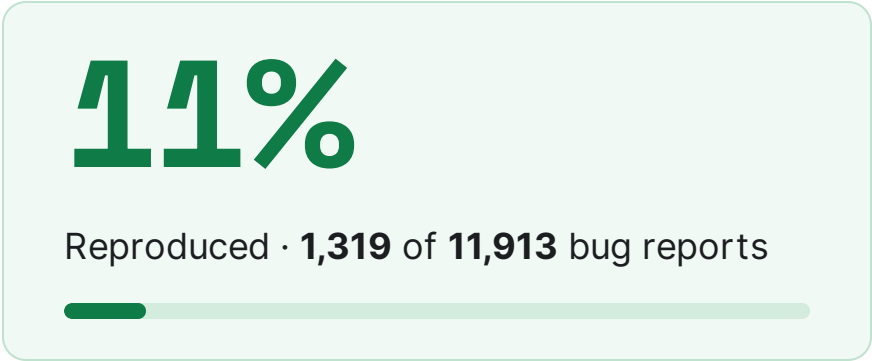
— WHAT IT GRANTS · EXPLOITATION PRIMITIVE

- ◇ The same crash can grant very different power, from a **constrained write** to **arbitrary write or control-flow hijack**.
- ◇ Static reasoning spots the **memory corruption**; a debugger-driven check **confirms the primitive** before it counts.

IF	Invalid free	WRITE
OUW	OOB / UAF write	WRITE
CFH	Control-flow hijack	CONTROL
AAW	Arbitrary write	CONTROL

Only 11% of bug reports reproduce

And most of the reproduced bugs still need root to trigger.



MINIMUM TRIGGER PRIVILEGE · REPRODUCED BUGS			
PRIVILEGE		BUGS	SHARE
Root		911	69.07%
Namespace privileges		324	24.56%
Unprivileged		84	6.37%

Trigger conditions affect priority

Even with dynamic checks and a threat model, some bugs only fire under odd, demanding conditions.

— CASE · A TIME-RANGE PACKET MATCH

```
● ● ● cross-day contiguous match

// cross-day range backdates a day
stamp -= SECONDS_PER_DAY;

// if the clock sits near 1970:
stamp < 0 → monthday wraps > 31

val = 1U << monthday;
// out-of-range shift – undefined behavior
```

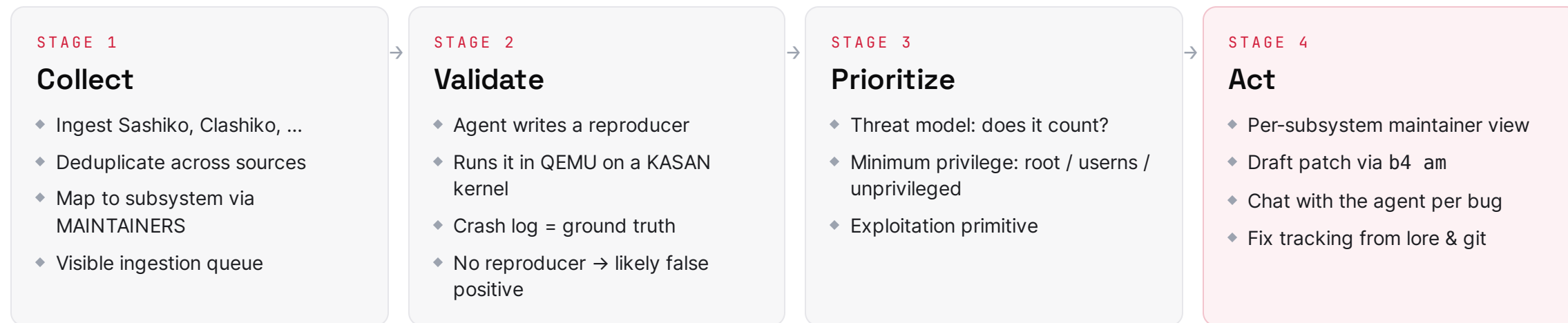
— REAL CRASH – PRACTICALLY UNREACHABLE

- ◇ It genuinely **crashes the kernel** — an out-of-range **1U << monthday** shift, real and reproducible.
- ◇ But it only fires if the **system clock sits near 1970** — a corner that **virtually never happens** in the real world.

Reality: whether bugs like this are **worth fixing** takes an **experienced developer** to judge. Verification narrows the field; it doesn't remove the expert.

Let's build a pipeline: from raw reports to **actions**

A repeatable workflow: collect reports, validate bugs, prioritize impact and act.



Every stage runs automatically. Humans step in only where judgment is needed — bugs with unusual trigger conditions, and the final patch.

From finding to fix, the workflow is **fragmented**

Knowing which bug matters is only the start. Reports, evidence and fixes need to stay connected.

— WHERE THE WORKFLOW LIVES

REPORTS

syzbot, AI review & mailing lists

Separate dashboards, tools and discussion threads

EVIDENCE

Reproducers, crash logs & impact

Verification results must stay linked to the report

FIXES

Patch threads & git commits

A proposed patch and its merge may appear elsewhere

— WHAT MAINTAINERS MUST PIECE TOGETHER

- ◇ **Who owns it?** — route the bug to the right maintainer, even when reports overlap.
- ◇ **Where is the evidence?** — connect the report, reproducer and impact assessment.
- ◇ **Has it been fixed?** — follow the patch discussion and track whether the fix has landed.

A unified tracking platform keeps each bug's reports, evidence and fix history in one record, updated continuously.

We built a unified bug-tracking platform

bugtracker.nebusec.ai

Kernel Bug Tracker

Sashiko queue

Guide

Chats

Auto

Yuan Tan

SUBSYSTEMS

Clear

Your name, e-mail, or st

Bugs are mapped to MAINTAINERS entries by the files they touch. Pick one or more subsystems to scope the list.

Select all

☒ MAC80211 224

☐ NETWORKING... 11624

☐ NETFILTER 1271

☐ CRYPTO API 884

☐ NETWORKING D... 501

☐ TC subsystem 458

☐ MEMORY MANA... 412

☐ LIBRARY CODE 378

☐ NETWORKING [I... 310

☐ BLUETOOTH SU... 271

☐ NETWORKING [I... 252

☐ NFS, SUNRPC, A... 243

223

Bugs in scope

194

Open

29

Fixed

All 16

Mine 0

Open 2

Fixed 14

Draft patch generated 2

PoC Run Result ?

All 194

Crash 2 x

Triggered, no crash 34

Couldn't reproduce 57

Falsified 101

Out of scope 1

Trigger Required Privilege ?

All 2 x

unprivileged 0

user namespace 1

root 1

Exploitability ?

All 2 x

Exploit ready 1

Exploit very likely 0

Exploit likely 0

Not exploitable 0

Pending analysis 1

Search bugs: id, file, description...

Source: all

Type: all

Show severity from sashiko

Crash x

MAC80211 x

PICKED BY	SOURCE · ID	DESCRIPTION	TYPE	FILE	TRIGGER	STATUS
Zhiling... x	nebusec #12147	Exploit ready Netdev debugfs recreation leaves default-key dentries dangling for later removal...	use-after-free	net/mac80211/debugfs_k...	Crash user ns	Open Fixed Out of scope
Pick up	nebusec #12145	AP-VLAN reenable erases inherited crypto tailroom requirements	memory-corrup...	net/mac80211/key.c	Crash root	Open Fixed Out of scope

2 of 223 bugs in scope · priority order

1 / 1

What the platform provides

Like syzbot for fuzzer bugs — an automated tracking layer for AI-reported bugs.

✓ Zero extra effort

Nothing to run, configure or maintain.
Nothing is posted to the mailing lists.

✓ Tracked automatically

Mailing lists and git are monitored; bugs are **marked fixed** automatically.

✓ Every bug reproduced

Each bug shown comes with a **reproducer** and a **KASAN crash log**.

Collect & deduplicate

Sashiko, Clashiko and more; visible ingestion queue.

Verify in QEMU

Agent writes a reproducer and runs it on a KASAN kernel.

Triage

Minimum privilege to trigger: root, namespace, or unprivileged.

Draft patch

A starting point for the fix; fetch with `b4 am`.

Per-subsystem view

Log in with a MAINTAINERS address; see only your bugs.

Chat with the agent

Discuss each bug and its fix on the bug page.

One bug, everything in one place

Severity at a glance

Exploitability and the minimum privilege needed to trigger it.

Tracked automatically

Flips to **Fixed** on its own once a patch appears on the mailing list.

Root cause traced

The commit and release that introduced the bug.

A proposed fix

With Fixes: and Cc: stable, ready for b4 am. Or chat with the agent about it.

bugtracker.nebusec.ai

↑
↓

12147
1 of 2 in view

Chat
X

nebusec
Exploit ready
Critical
● user namespace

Owner
Zhiling Zou
Clear

Status
Open
Fixed
Out of scope

Type
use-after-free

Found
2026-07-20 15:27 UTC (2 months ago)

Introduced by
c942398f95ef · v6.7

Tracked automatically — flips to Fixed on its own once a patch appears on the mailing list. Mark by hand only if tracking misses it.

PROPOSED FIX

1 patch

wifi: mac80211: reset default key debugfs pointers

1 file +4 -0 +++

FIXES c942398f95ef CC stable@vger.kernel.org

Where sanitizers go **blind**

A shared challenge: how do we validate real bugs that never trip KASAN?

PAGE OWNERSHIP / COPY-ON-WRITE

Writing to a page you don't own

Copy Fail (CVE-2026-31431) and variants: Dirty Frag, Fragnesia

BUG An in-place write lands on a **shared page-cache page** brought in by `splice()` / zero-copy.

WHY MISSED Every access is to valid, live memory — the bug is in **ownership semantics**, close to a logic bug.

OPEN QUESTION How can we check **page ownership** across zero-copy paths, using instrumentation or static reasoning?

STACK OBJECT LIFETIME

A pointer into a dead stack frame

GhostLock (CVE-2026-43499): futex PI, unprivileged → root, 15 years old

BUG A task keeps `pi_blocked_on` pointing at an `rt_mutex_waiter` on a stack frame that has already returned.

WHY MISSED KASAN has **no stack-lifetime checks**: use-after-scope detection was removed in 2019 (7771bdbbfd3d) for its cost and few findings.

OPEN QUESTION Can **stack tagging** or other lifetime checks detect these bugs at an acceptable runtime cost?

Let's build better validation signals together. We need new checks and shared regression tests for bugs beyond crash-based oracles.

Next steps

Live at bugtracker.nebusec.ai

01 Improve the quality of generated patches

02 Ingest syzbot bug reports

03 Find a way to validate driver bugs