# So You Wanna
# Find Bugs In The Linux Kernel?

# About Me

- Sam (@sam4k1)
- Background in VR and exploit dev
- I like Linux, security, games & food

# What’s The Plan?

- Cover the state of kernel VR in 2024
- Explore the kernel attack surface for targets
- Dive into approaches and workflow for actually finding bugs

# The State of Kernel VR Today

# State of Kernel VR | What's The Appeal?

- Lots of users (1.6+ billion)
- The kernel has ultimate control over the device
- Broad, open source attack surface with a history of bugs


Source(s): Statista

# State of Kernel VR | The Challenges

- Continuous hardening efforts, technology improvements
  - 150+ hardening options over the last 20 years[4]
  - Goal posts for finding + exploiting kernel vulns constantly shifting
  - Attempts to reduce availability of generic techniques
- Increased awareness + vested interests in security
  - Continual fuzzing, bounty programs etc.
- Culminates in increasing complexity to “weaponize” a vulnerability

# State of Kernel VR | So Where Are The Bugs?!

- How do we track what bugs are found where?

# State of Kernel VR | So Where Are The Bugs?!

- How do we track what bugs are found where?
  - Not all CVEs are created equal[1][10]

```
* Re: CVE-2024-26904: btrfs: fix data race at btrfs_use_block_rsv() when accessing block reserve
      [not found] ` <Zkt48ug3KKOTQk42@debian0.Home>
@ 2024-05-21  7:05   ` Greg Kroah-Hartman
  0 siblings, 0 replies; 2+ messages in thread
From: Greg Kroah-Hartman @ 2024-05-21  7:05 UTC (permalink / raw)
  To: Filipe Manana; +Cc: cve, linux-kernel, linux-cve-announce

On Mon, May 20, 2024 at 05:23:14PM +0100, Filipe Manana wrote:
> On Wed, Apr 17, 2024 at 12:29:19PM +0200, Greg Kroah-Hartman wrote:
> > Description
> > ===========
> >
> > In the Linux kernel, the following vulnerability has been resolved:
> >
> > btrfs: fix data race at btrfs_use_block_rsv() when accessing block reserve
>
> May I ask why is this classified a CVE?
>
> How can a malicious user exploit this to do something harmful?
>
> The race was solved to silence KCSAN warnings, as from time to time we have
> someone reporting it, but other than that, it should be harmless.

Oops, you are right, the line "BUG:" triggered our review to tag this as
a CVE.  I'll go reject it now, thanks for the review.

greg k-h
```

**Brad Spengler**
@spendergrsec

On the heels of 200 CVEs Friday and Sunday, 62 more yesterday, now today 378 new CVEs have been published, 640 in total in less than a week: lore.kernel.org/linux-cve-anno.... These 378 from today are what they're calling "backfilled" CVEs, old results from their also-automated GSD dataset

5:23 PM · May 21, 2024 · **7,705** Views

**GrapheneOS**
@GrapheneOS

Linux kernel becoming their own CVE Numbering Authority (CNA) is wasting resources they'd have previously put towards higher quantity and quality backporting. We've noticed a drop in both for the stable/longterm branches and particularly Android Generic Kernel Image LTS branches.

1:52 AM · May 23, 2024 · **9,965** Views

# State of Kernel VR | So Where Are The Bugs?!

- How do we track what bugs are found where?
  - Not all CVEs are created equal[1]
  - Neither are all kernel commits[2][11][12]

**netfilter: nf_tables: use timestamp to check for set element timeout**

```
Add a timestamp field at the beginning of the transaction, store it
in the nftables per-netns area.

Update set backend .insert, .deactivate and sync gc path to use the
timestamp, this avoids that an element expires while control plane
transaction is still unfinished.

.lookup and .update, which are used from packet path, still use the
current time to check if the element has expired. And .get path and dump
also since this runs lockless under rcu read size lock. Then, there is
async gc which also needs to check the current time since it runs
asynchronously from a workqueue.

Fixes: c3e1b00 ("netfilter: nf_tables: add set element timeout support")
Signed-off-by: Pablo Neira Ayuso <pablo@netfilter.org>
```

master
v6.9 ... v6.8-rc4

ummakynes committed on Feb 8

**tipc: fix UAF in error path**

```
Sam Page (sam4k) working with Trend Micro Zero Day Initiative reported
a UAF in the tipc_buf_append() error path:

BUG: KASAN: slab-use-after-free in kfree_skb_list_reason+0x47e/0x4c0
linux/net/core/skbuff.c:1183
Read of size 8 at addr ffff88804d2a7c80 by task poc/8034

CPU: 1 PID: 8034 Comm: poc Not tainted 6.8.2 #1
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS
1.16.0-debian-1.16.0-5 04/01/2014
Call Trace:
 <IRQ>
 // SNIP FOR TYPHOONCON SLIDES
 </TASK>

In the critical scenario, either the relevant skb is freed or its
ownership is transferred into a frag_lists. In both cases, the cleanup
code must not free it again: we need to clear the skb reference earlier.

Fixes: 1149557 ("tipc: eliminate unnecessary linearization of incoming buffers")
Cc: stable@vger.kernel.org
Reported-by: zdi-disclosures@trendmicro.com # ZDI-CAN-23852
Acked-by: Xin Long <lucien.xin@gmail.com>
Signed-off-by: Paolo Abeni <pabeni@redhat.com>
Reviewed-by: Eric Dumazet <edumazet@google.com>
Link: https://lore.kernel.org/r/752f1ccf762223d109845365d07f55414058e5a3.1714484273.git.pabeni@redhat.com
Signed-off-by: Jakub Kicinski <kuba@kernel.org>
```

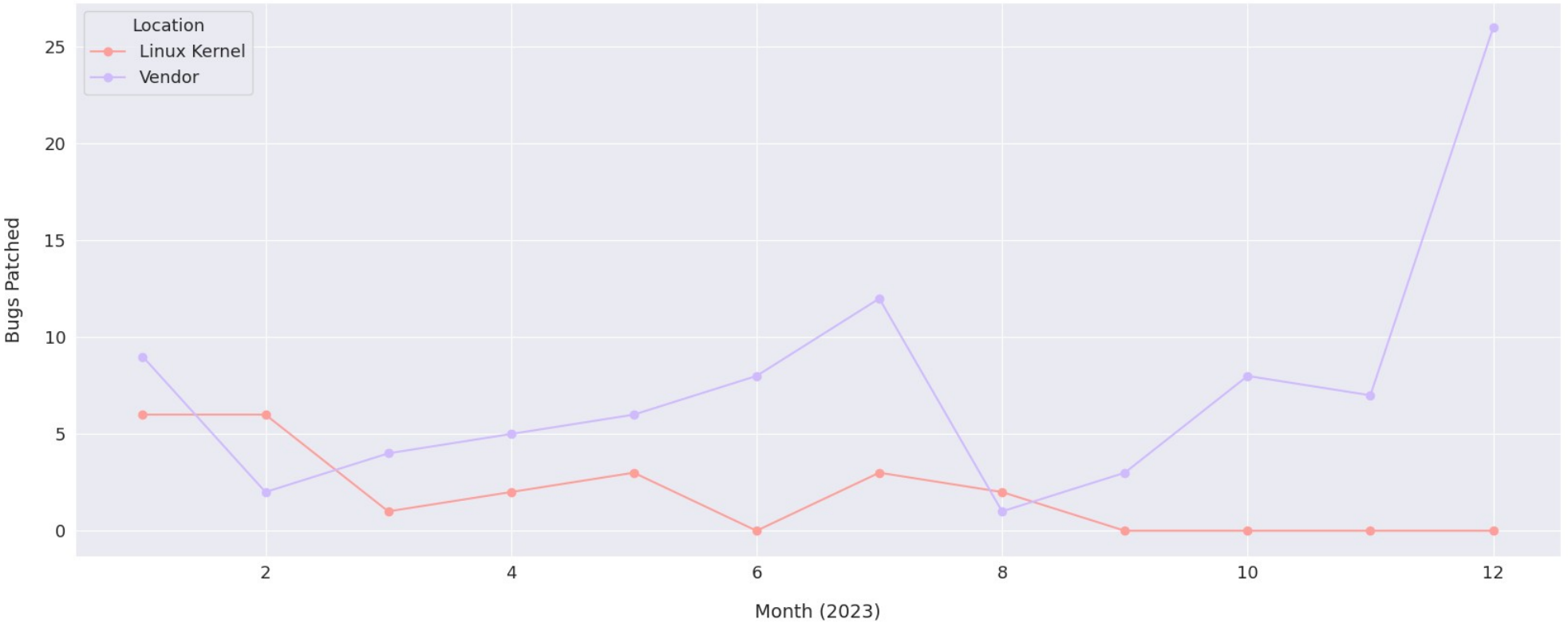
master
v6.9 v6.9-rc7

Paolo Abeni authored and kuba-moo committed 3 weeks ago

# State of Kernel VR | So Where Are The Bugs?!

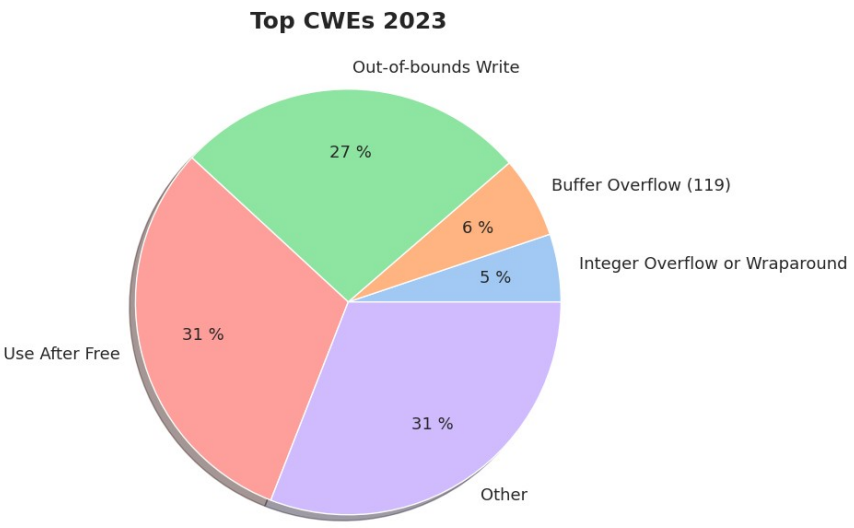
- How do we track what bugs are found where?
  - Not all CVEs are created equal[1]
  - Neither are all kernel commits[2]
  - What about common 3rd parties/vendors?
  - And ofc then there's the 0days…
- Android Security Bulletin?
  - Monthly list of impactful kernel vulns, incl upstream + vendors

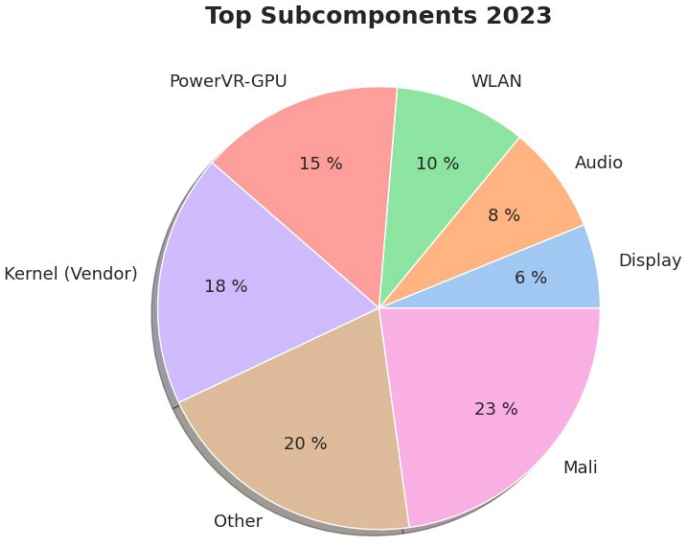
# State of Kernel VR | So Where Are The Bugs?!




Source(s): Android Security Bulletin

# State of Kernel VR | So Where Are The Bugs?!




Source(s): Android Security Bulletin

# Picking A Kernel Target

# Picking A Kernel Target | Some Context

At a high-level, there are several different tiers of attack surface in the Android ecosystem. Here are some of the important ones:

- *Tier:* Ubiquitous

  *Description:* Issues that affect all devices in the Android ecosystem.
  *Example:* Core Linux kernel bugs like Dirty COW, or vulnerabilities in standard system services.

- *Tier:* Chipset

  *Description:* Issues that affect a substantial portion of the Android ecosystem, based on which type of hardware is used by various OEM vendors.
  *Example:* Snapdragon SoC perf counter vulnerability, or Broadcom WiFi firmware stack overflow.

- *Tier:* Vendor

  *Description:* Issues that affect most or all devices from a particular Android OEM vendor
  *Example:* Samsung kernel driver vulnerabilities

- *Tier:* Device

  Description: Issues that affect a particular device model from an Android OEM vendor
  *Example*: Pixel 4 face unlock "attention aware" vulnerability

Source(s): Attacking the Qualcomm Adreno GPU by Ben Hawkes [3]

# Picking A Kernel Target | Defining The Attack Surface

- What's our goal? Define scope/target (E.g. Ubuntu 22.04.3 LTS)
  - E.g. specific device, bug bounty, vibes
- We want to consider:
  - The kernel version (and arch) we're interested in
  - The typical kernel configuration
  - Additional distro/vendor surface that might be present
  - The surface exposed to an unprivileged user/our chosen context
  - Reliability, privesc vs crash etc.

# Picking A Kernel Target | Kconfig & Narrowing Down Attack Surface

- Mitigations: FORTIFY_SOURCE, CFI, stack protector, heap hardening etc.[4]
    - Consider probabilistic vs deterministic mitigations
- Attack Surface: SELinux, Seccomp, unpriv namespaces etc.
- Exploitation Techniques: FUSE,  STATIC_USERMODEHELPER, generally reducing kernel surface (e.g. less gadgets, heap feng shui objects) etc.

# Picking A Kernel Target | Target Considerations

- Explore target history: past bugs, commits, recent features?
- Maturity and complexity: is it a tiny module that's been around forever?
- Syzkaller coverage: has it been fuzzed into oblivion already?

# Okay, How About Finding Bugs?

# Kernel Auditing | An Overview

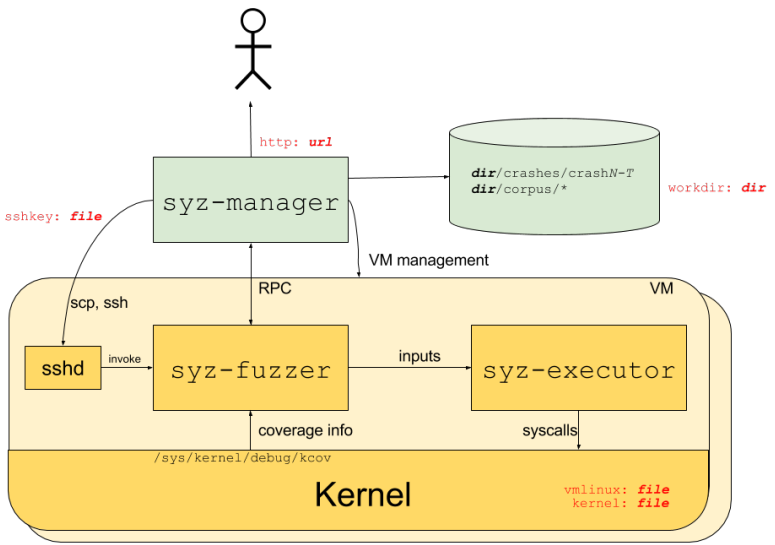
- Understand the tools and techniques available to us
- Use the knowledge gained so far to inform our approach
    - Bug classes, complexity, areas of interest etc.
    - It's an iterative process of trial and error!
- Remember this stuff is HARD (right???)

# Kernel Auditing | Code Auditing

- Take the time to *understand* what the code is trying to do
  - Continually ask questions, be curious!
  - Object lifetimes, locking, userspace interactions, state etc.
  - New features, complex interactions with other subystems etc.
- Factor in all of the context we've built up so far
  - Are we expecting low hanging fruit or complex bugs?
  - Are there bug classes that we should avoid completely?
- Don't neglect tooling, workflow & documentation

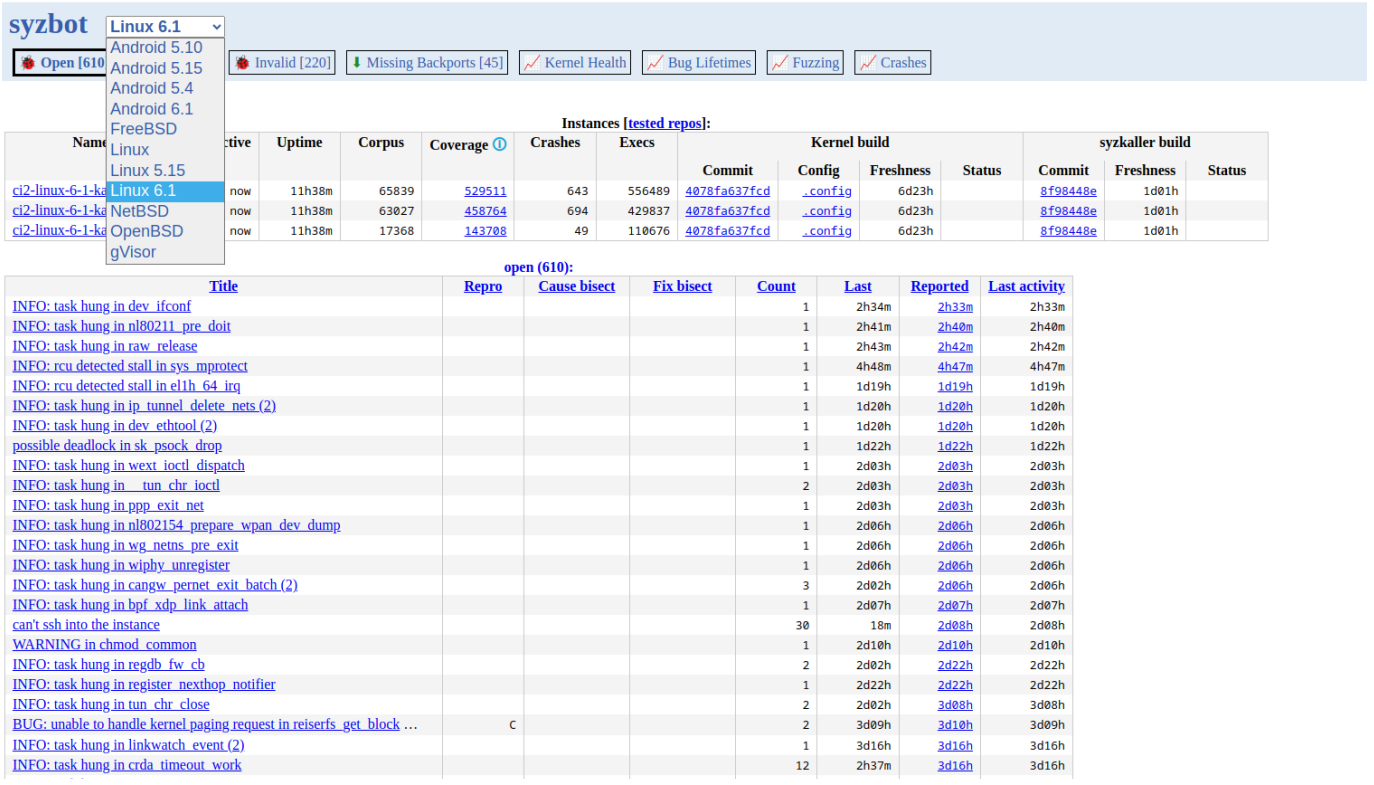
# Kernel Auditing | Fuzzing with Syzkaller

- syzkaller is an unsupervised coverage-guided kernel fuzzer[5]
- syzbot continuously fuzzes main Linux kernel branches[6]
- We can use the understanding developed to extend its coverage




Source(s): https://github.com/google/syzkaller/blob/master/docs/internals.md

# Kernel Auditing | Syzbot Dashboard



**syzbot** Linux 6.1

Android 5.10
Android 5.15
Android 5.4
Android 6.1
FreeBSD
Linux
Linux 5.15
Linux 6.1
NetBSD
OpenBSD
gVisor

Open [610] | Invalid [220] | Missing Backports [45] | Kernel Health | Bug Lifetimes | Fuzzing | Crashes

**Instances [tested repos]:**

| Name | ctive | Uptime | Corpus | Coverage | Crashes | Execs | Kernel build | | | | syzkaller build | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | Commit | Config | Freshness | Status | Commit | Freshness | Status |
| ci2-linux-6-1-ka | now | 11h38m | 65839 | 529511 | 643 | 556489 | 4078fa637fcd | .config | 6d23h | | 8f98448e | 1d01h | |
| ci2-linux-6-1-ka | now | 11h38m | 63027 | 458764 | 694 | 429837 | 4078fa637fcd | .config | 6d23h | | 8f98448e | 1d01h | |
| ci2-linux-6-1-ka | now | 11h38m | 17368 | 143708 | 49 | 110676 | 4078fa637fcd | .config | 6d23h | | 8f98448e | 1d01h | |

**open (610):**

| Title | Repro | Cause bisect | Fix bisect | Count | Last | Reported | Last activity |
|---|---|---|---|---|---|---|---|
| INFO: task hung in dev_ifconf | | | | 1 | 2h34m | 2h33m | 2h33m |
| INFO: task hung in nl80211_pre_doit | | | | 1 | 2h41m | 2h40m | 2h40m |
| INFO: task hung in raw_release | | | | 1 | 2h43m | 2h42m | 2h42m |
| INFO: rcu detected stall in sys_mprotect | | | | 1 | 4h48m | 4h47m | 4h47m |
| INFO: rcu detected stall in el1h_64_irq | | | | 1 | 1d19h | 1d19h | 1d19h |
| INFO: task hung in ip_tunnel_delete_nets (2) | | | | 1 | 1d20h | 1d20h | 1d20h |
| INFO: task hung in dev_ethtool (2) | | | | 1 | 1d20h | 1d20h | 1d20h |
| possible deadlock in sk_psock_drop | | | | 1 | 1d22h | 1d22h | 1d22h |
| INFO: task hung in wext_ioctl_dispatch | | | | 1 | 2d03h | 2d03h | 2d03h |
| INFO: task hung in __tun_chr_ioctl | | | | 2 | 2d03h | 2d03h | 2d03h |
| INFO: task hung in ppp_exit_net | | | | 1 | 2d03h | 2d03h | 2d03h |
| INFO: task hung in nl802154_prepare_wpan_dev_dump | | | | 1 | 2d06h | 2d06h | 2d06h |
| INFO: task hung in wg_netns_pre_exit | | | | 1 | 2d06h | 2d06h | 2d06h |
| INFO: task hung in wiphy_unregister | | | | 1 | 2d06h | 2d06h | 2d06h |
| INFO: task hung in cangw_pernet_exit_batch (2) | | | | 3 | 2d02h | 2d06h | 2d06h |
| INFO: task hung in bpf_xdp_link_attach | | | | 1 | 2d07h | 2d07h | 2d07h |
| can't ssh into the instance | | | | 30 | 18m | 2d08h | 2d08h |
| WARNING in chmod_common | | | | 1 | 2d10h | 2d10h | 2d10h |
| INFO: task hung in regdb_fw_cb | | | | 2 | 2d02h | 2d22h | 2d22h |
| INFO: task hung in register_nexthop_notifier | | | | 1 | 2d22h | 2d22h | 2d22h |
| INFO: task hung in tun_chr_close | | | | 2 | 2d02h | 3d08h | 3d08h |
| BUG: unable to handle kernel paging request in reiserfs_get_block ... | C | | | 2 | 3d09h | 3d10h | 3d09h |
| INFO: task hung in linkwatch_event (2) | | | | 1 | 3d16h | 3d16h | 3d16h |
| INFO: task hung in crda_timeout_work | | | | 12 | 2h37m | 3d16h | 3d16h |

Source(s): https://syzkaller.appspot.com/linux-6.1

# Kernel Auditing | Syzbot Dashboard

| | | |
|---|---|---|
| ▶ arch/x86 | 17% | of 47506 |
| ▶ block | 20% | of 16392 |
| ▶ certs | 24% | of 21 |
| ▶ crypto | 25% | of 8814 |
| ▶ drivers | 6% | of 624560 |
| ▶ fs | 21% | of 345243 |
| ▶ include | 100% | of 588 |
| ▶ init | 3% | of 342 |
| ▼ io_uring | 41% | of 5467 |
| advise.c | 64% | of 19 |
| alloc_cache.h | 100% | of 1 |
| cancel.c | 57% | of 92 |
| epoll.c | 20% | of 10 |
| fdinfo.c | 45% | of 81 |
| filetable.c | 45% | of 49 |
| filetable.h | 100% | of 1 |
| fs.c | 62% | of 60 |
| io-wq.c | 22% | of 510 |
| io-wq.h | 100% | of 1 |
| io_uring.c | 41% | of 1987 |
| io_uring.h | 100% | of 1 |
| kbuf.c | 45% | of 212 |
| kbuf.h | 100% | of 1 |
| msg_ring.c | 15% | of 47 |
| net.c | 34% | of 488 |
| nop.c | 100% | of 2 |
| notif.c | --- | of 38 |
| opdef.c | 40% | of 5 |
| openclose.c | 58% | of 92 |
| poll.c | 45% | of 349 |
| refs.h | 100% | of 1 |
| rsrc.c | 60% | of 423 |
| rsrc.h | 100% | of 1 |
| rw.c | 46% | of 356 |
| slist.h | 100% | of 1 |
| splice.c | 30% | of 30 |
| sqpoll.c | 36% | of 166 |
| statx.c | 67% | of 12 |
| sync.c | 47% | of 32 |
| tctx.c | 44% | of 113 |
| tctx.h | 100% | of 1 |
| timeout.c | 42% | of 191 |
| timeout.h | 100% | of 1 |
| uring_cmd.c | --- | of 42 |
| xattr.c | 24% | of 51 |
| ▶ ipc | 42% | of 2564 |
| ▶ kernel | 25% | of 70494 |
| ▶ lib | 19% | of 31717 |
| ▶ mm | 26% | of 48808 |
| ▶ net | 25% | of 373956 |
| ▶ security | 25% | of 16597 |
| ▶ sound | 17% | of 34732 |
| ▶ tools/lib/bpf | --- | of 430 |
| ▶ virt | 30% | of 3375 |

```
1615
1616 static __cold void io_drain_req(struct io_kiocb *req)
1617         __must_hold(&ctx->uring_lock)
  38 1618 {
1619         struct io_ring_ctx *ctx = req->ctx;
1620         struct io_defer_entry *de;
1621         int ret;
  38 1622         u32 seq = io_get_sequence(req);
1623
1624         /* Still need defer if there is pending req in defer list. */
1625         spin_lock(&ctx->completion_lock);
  37 1626         if (!req_need_defer(req, seq) && list_empty_careful(&ctx->defer_list)) {
  24 1627                 spin_unlock(&ctx->completion_lock);
1628 queue:
1629                 ctx->drain_active = false;
1630                 io_req_task_queue(req);
1631                 return;
1632         }
1633         spin_unlock(&ctx->completion_lock);
1634
  16 1635         io_prep_async_link(req);
1636         de = kmalloc(sizeof(*de), GFP_KERNEL);
1637         if (!de) {
1638                 ret = -ENOMEM;
1639                 io_req_complete_failed(req, ret);
1640                 return;
1641         }
1642
1643         spin_lock(&ctx->completion_lock);
  16 1644         if (!req_need_defer(req, seq) && list_empty(&ctx->defer_list)) {
1645                 spin_unlock(&ctx->completion_lock);
1646                 kfree(de);
1647                 goto queue;
1648         }
1649
  16 1650         trace_io_uring_defer(req);
1651         de->req = req;
1652         de->seq = seq;
  16 1653         list_add_tail(&de->list, &ctx->defer_list);
1654         spin_unlock(&ctx->completion_lock);
1655 }
1656
1657 static void io_clean_op(struct io_kiocb *req)
 109 1658 {
 108 1659         if (req->flags & REQ_F_BUFFER_SELECTED) {
1660                 spin_lock(&req->ctx->completion_lock);
1661                 io_put_kbuf_comp(req);
1662                 spin_unlock(&req->ctx->completion_lock);
1663         }
1664
  90 1665         if (req->flags & REQ_F_NEED_CLEANUP) {
1666                 const struct io_op_def *def = &io_op_defs[req->opcode];
1667
1668                 if (def->cleanup)
  20 1669                         def->cleanup(req);
1670         }
 105 1671         if ((req->flags & REQ_F_POLLED) && req->apoll) {
   4 1672                 kfree(req->apoll->double_poll);
1673                 kfree(req->apoll);
1674                 req->apoll = NULL;
1675         }
  94 1676         if (req->flags & REQ_F_INFLIGHT) {
```

**Covered: black (#000000)**

All PC values associated to that line are covered. There is number on the left side indicating how many programs have triggered executing the PC values associated to this line. You can click on that number and it will open last executed program. Example below shows how single line which is fully covered is shown.

```
static inline bool drive_no_geom(int drive)
{
4       return !current_type[drive] && !ITYPE(UDRS->fd_device);
}
```

**Both: orange (#c86400)**

There are several PC values associated to the line and not all of these are executed. Again there is number left to the source code line that can clicked to open last program triggering associated PC values. Example below shows single line which has both executed and non-executed PC values associated to it.

```
static void process_fd_request(void)
{
13      cont = &rw_cont;
        schedule_bh(redo_fd_request);
}
```

**Weak-uncovered: crimson red (#c80000)**

Function (symbol) this line is in doesn't have any coverage. I.e. the function is not executed at all. Please note that if compiler have optimized certain symbol out and made the code inline instead symbol associated with this line is the one where the code is compiled into. This makes it sometimes real hard to figure out meaning of coloring. Example below shows how single line which is uncovered and PC values associated to it are in function(s) that are not executed either is shown.

```
static void reset_intr(void)
{
        pr_info("weird, reset interrupt called\n");
}
```

**Uncovered: red (#ff0000)**

Line is uncovered. Function (symbol) this line is in is executed and one of the PC values associated to this line. Example below shows how single line which is not covered is shown.

```
static void cancel_activity(void)
{
        do_floppy = NULL;
        cancel_delayed_work_sync(&fd_timer);
        cancel_work_sync(&floppy_work);
}
```

**Not instrumented: grey (#505050)**

PC values associated to the line are not instrumented or source line doesn't generate code at all. Example below shows how all not instrumented code is shown.

```
#ifndef fd_eject
static inline int fd_eject(int drive)
{
        return -EINVAL;
}
#endif
```

Source(s): [8][9]

# Kernel Auditing | Modifying Syzkaller

- Broadly speaking, three things to consider:
    - Descriptions: describe syscalls, their arguments, possible values and any order they need to be called in

```
# snippets from syzkaller/sys/linux/sys.txt
include <linux/fcntl.h> # header includes for defs

resource fd[int32]: -1 # define resource

# syscall descriptions for open(2) and close(2)
open(file ptr[in, filename], flags flags[open_flags], mode flags[open_mode]) fd
close(fd fd)

# defs for description args
open_flags = O_WRONLY, O_RDWR, O_APPEND, FASYNC, O_CLOEXEC, O_CREAT, O_DIRECT, O_DIRECTORY, O_EXCL, O_LARGEFILE,
O_NOATIME, O_NOCTTY, O_NOFOLLOW, O_NONBLOCK, O_PATH, O_SYNC, O_TRUNC, __O_TMPFILE
open_mode = S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH
```

# Kernel Auditing | Modifying Syzkaller

- Broadly speaking, three things to consider:
  - Descriptions: describe syscalls, their arguments, possible values and any order they need to be called in
  - Pseudo-syscalls: wrappers around syscalls to carry out any additional setup or state-tweaking to get desired coverage
  - Adding KCOV: subsystem for collecting coverage; may need to add remote coverage for code run outside the process context

# Kernel Auditing | Code Querying with CodeQL

- CodeQL lets you query code as though it were data[7]
- Need to create a database for the code we want to query
- Can be used to query for vuln patterns, variant analysis etc.
- But also can be used to augment code audit & enumeration
- As well as exploit development! (out of scope for this talk tho :()

# Kernel Auditing | CodeQL Example

- Example of query to find kmalloc calls taking 16-bit arguments (easier to overflow) for further analysis:

```
import cpp

from FunctionCall fc // Select all Function Calls
where fc.getTarget().getName() = "kmalloc" // Where the target function is called kmalloc
and fc.getArgument(0).getType().getSize() = 2 // and the supplied size argument is a 16-bit int
select fc, fc.getLocation() // Select the call location and the string of the location to know what file it's in

// src: https://www.sentinelone.com/labs/tipc-remote-linux-kernel-heap-overflow-allows-arbitrary-code-execution/
```

Source(s): https://www.sentinelone.com/labs/tipc-remote-linux-kernel-heap-overflow-allows-arbitrary-code-execution/

# Kernel Auditing | CodeQL Usecases

- Querying for vulnerabilities: variant analysis on bugs found, rule out low hanging fruit/easily query-able bug classes to free up audit time etc.
- Enumerate attack surface, highlight areas of interest: what objects are allocated, where are they accessed, which are ref counted, have fptrs etc.
- Automate code auditing process: check if certain fields are accessed, function is called with certain args, if a certain condition is guarded etc.

# Kernel Auditing | A Case Study

- Transparent Inter-Process Communication (TIPC)
- Non-default network protocol (RCE is cool right?)
- Low Syzkaller coverage
- Previous experience with it

| | | |
|---|---|---|
| ▼ tipc | 27%(47%) | of 8041(4492) |
| addr.c | 51%(51%) | of 51(51) |
| addr.h | 100%(0%) | of 1(0) |
| bcast.c | 25%(33%) | of 221(167) |
| bearer.c | 26%(41%) | of 607(382) |
| bearer.h | 100%(0%) | of 1(0) |
| core.c | --- | of 28 |
| core.h | 100%(0%) | of 1(0) |
| crypto.c | 7%(23%) | of 826(249) |
| diag.c | 67%(67%) | of 12(12) |
| discover.c | 13%(40%) | of 77(25) |
| eth_media.c | 34%(100%) | of 6(2) |
| group.c | 58%(59%) | of 326(321) |
| ib_media.c | --- | of 6 |
| link.c | 3%(43%) | of 892(61) |
| monitor.c | 8%(25%) | of 314(97) |
| msg.c | 32%(46%) | of 223(153) |
| msg.h | 100%(0%) | of 1(0) |
| name_distr.c | 25%(56%) | of 83(36) |
| name_table.c | 47%(55%) | of 599(516) |
| net.c | 42%(44%) | of 68(64) |
| netlink.c | --- | of 1 |
| netlink_compat.c | 54%(57%) | of 248(232) |
| node.c | 11%(28%) | of 1200(457) |
| socket.c | 52%(55%) | of 1493(1419) |
| subscr.c | 49%(53%) | of 37(34) |
| sysctl.c | --- | of 2 |
| topsrv.c | 23%(43%) | of 148(77) |
| trace.c | --- | of 335 |
| trace.h | 100%(0%) | of 1(0) |
| udp_media.c | 27%(46%) | of 233(137) |

Source(s): https://syzkaller.appspot.com/linux-6.1

# Kernel Auditing | A Case Study

- Used understanding gained via code audit to determine key interactions which lacked coverage and why this was
- Implemented proper message formatting and TIPC handshake boilerplate
- discover.c (+29%), link.c (+32%), monitor.c (+15%), name_distr.c (+46%), node.c (+17%)

| | | |
|---|---|---|
| ▼ tipc | 16%(51%) | of 7423(2285) |
| addr.c | 7%(29%) | of 33(7) |
| addr.h | 100%(0%) | of 1(0) |
| bcast.c | 14%(56%) | of 233(58) |
| bcast.h | 100%(0%) | of 1(0) |
| bearer.c | 9%(77%) | of 437(46) |
| bearer.h | 100%(0%) | of 1(0) |
| core.c | --- | of 14 |
| core.h | 100%(0%) | of 1(0) |
| crypto.c | 7%(18%) | of 684(244) |
| diag.c | --- | of 12 |
| discover.c | 42%(67%) | of 77(48) |
| eth_media.c | --- | of 6 |
| group.c | --- | of 270 |
| link.c | 35%(53%) | of 954(631) |
| monitor.c | 23%(47%) | of 229(113) |
| msg.c | 23%(60%) | of 264(100) |
| msg.h | 100%(0%) | of 1(0) |
| name_distr.c | 71%(76%) | of 78(73) |
| name_table.c | 35%(78%) | of 424(189) |
| net.c | --- | of 71 |
| netlink.c | --- | of 1 |
| netlink_compat.c | --- | of 264 |
| node.c | 28%(52%) | of 1041(563) |
| socket.c | 4%(36%) | of 1463(153) |
| subscr.c | --- | of 58 |
| sysctl.c | --- | of 3 |
| topsrv.c | --- | of 210 |
| trace.c | --- | of 382 |
| trace.h | 100%(0%) | of 1(0) |
| udp_media.c | 12%(42%) | of 209(60) |

# Kernel Auditing | A Case Study

**Crashes:**

| Description | Count |
|---|---|
| INFO: rcu detected stall in [REDACTED] | 9 |
| INFO: rcu detected stall in [REDACTED] | 10 |
| KASAN: slab-use-after-free Read in [REDACTED] | 99 |
| KASAN: slab-use-after-free Read in [REDACTED] | 13 |
| KASAN: slab-use-after-free Write in [REDACTED] | 1 |
| KASAN: stack-out-of-bounds Read in [REDACTED] | 100 |

ZDI-CAN-23852 Linux CVSS: 9.0 2024-04-25 (18 days ago) 2024-08-23

Discovered by: Sam Page (sam4k)

**tipc: fix UAF in error path**

```
Sam Page (sam4k) working with Trend Micro Zero Day Initiative reported
a UAF in the tipc_buf_append() error path:

BUG: KASAN: slab-use-after-free in kfree_skb_list_reason+0x47e/0x4c0
linux/net/core/skbuff.c:1183
Read of size 8 at addr ffff88804d2a7c80 by task poc/8034

CPU: 1 PID: 8034 Comm: poc Not tainted 6.8.2 #1
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS
1.16.0-debian-1.16.0-5 04/01/2014
Call Trace:
 <IRQ>
 __dump_stack linux/lib/dump_stack.c:88
 dump_stack_lvl+0xd9/0x1b0 linux/lib/dump_stack.c:106
 print_address_description linux/mm/kasan/report.c:377
 print_report+0xc4/0x620 linux/mm/kasan/report.c:488
 kasan_report+0xda/0x110 linux/mm/kasan/report.c:601
 kfree_skb_list_reason+0x47e/0x4c0 linux/net/core/skbuff.c:1183
 skb_release_data+0x5af/0x880 linux/net/core/skbuff.c:1026
 skb_release_all linux/net/core/skbuff.c:1094
 __kfree_skb linux/net/core/skbuff.c:1108
 kfree_skb_reason+0x12d/0x210 linux/net/core/skbuff.c:1144
 kfree_skb linux/./include/linux/skbuff.h:1244
 tipc_buf_append+0x425/0xb50 linux/net/tipc/msg.c:186
 tipc_link_input+0x224/0x7c0 linux/net/tipc/link.c:1324
 tipc_link_rcv+0x76e/0x2d70 linux/net/tipc/link.c:1824
 tipc_rcv+0x45f/0x10f0 linux/net/tipc/node.c:2159
 tipc_udp_recv+0x73b/0x8f0 linux/net/tipc/udp_media.c:390
 // SNIP FOR TYPHOONCON SLIDES
 </IRQ>
 <TASK>
 // SNIP FOR TYPHOONCON SLIDES
 sock_sendmsg_nosec linux/net/socket.c:730
 __sock_sendmsg linux/net/socket.c:745
 __sys_sendto+0x42c/0x4e0 linux/net/socket.c:2191
 __do_sys_sendto linux/net/socket.c:2203
 __se_sys_sendto linux/net/socket.c:2199
 __x64_sys_sendto+0xe0/0x1c0 linux/net/socket.c:2199
 do_syscall_x64 linux/arch/x86/entry/common.c:52
 do_syscall_64+0xd8/0x270 linux/arch/x86/entry/common.c:83
 entry_SYSCALL_64_after_hwframe+0x6f/0x77 linux/arch/x86/entry/entry_64.S:120
 // SNIP FOR TYPHOONCON SLIDES
 </TASK>

In the critical scenario, either the relevant skb is freed or its
ownership is transferred into a frag_lists. In both cases, the cleanup
code must not free it again: we need to clear the skb reference earlier.

Fixes: 1149557 ("tipc: eliminate unnecessary linearization of incoming buffers")
Cc: stable@vger.kernel.org
Reported-by: zdi-disclosures@trendmicro.com # ZDI-CAN-23852
Acked-by: Xin Long <lucien.xin@gmail.com>
Signed-off-by: Paolo Abeni <pabeni@redhat.com>
Reviewed-by: Eric Dumazet <edumazet@google.com>
Link: https://lore.kernel.org/r/752f1ccf762223d109845365d07f55414058e5a3.1714484273.git.pabeni@redhat.com
```

# Wrapping Up

# Wrapping Up

- Ask questions, be curious!
- Pace yourself, (try to) enjoy the process
- Experiment with tools and techniques
- Sometimes there just isn't a bug! But the knowledge + exp carries over
- Feel free to ping me off/online :)

# Resources

- https://github.com/google/syzkaller
- https://codeql.github.com
- https://github.com/xairy/linux-kernel-exploitation (great collection of kernel exploitation resources)
- https://pwning.tech/ksmbd-syzkaller/ (good guide on extending syzkaller for ksmbd)

# Refs


1) “The bogus CVE problem”, “Supplementing CVEs with !CVEs” by Jake Edge at LWN
2) https://sam4k.com/analysing-linux-kernel-commits/#on-silent-security-fixes
3) https://googleprojectzero.blogspot.com/2020/09/attacking-qualcomm-adreno-gpu.html
4) https://github.com/a13xp0p0v/kernel-hardening-checker
5) https://github.com/google/syzkaller
6) https://github.com/google/syzkaller/blob/master/docs/syzbot.md
7) https://codeql.github.com
8) https://storage.googleapis.com/syzbot-assets/34c45129131f/ci2-linux-6-1-kasan-4078fa63.html#io_uring%2fio_uring.c
9) https://github.com/google/syzkaller/blob/master/docs/coverage.md
10) https://lore.kernel.org/linux-cve-announce/2024052155-raking-onshore-f6f3@gregkh/T/#t
11) https://github.com/torvalds/linux/commit/7395dfacfff65e9938ac0889dafa1ab01e987d15
12) https://github.com/torvalds/linux/commit/080cbb890286cd794f1ee788bbc5463e2deb7c2b