Net Shredder:
Coverage-Guided Network Fuzzing for the Linux Kernel

BSides Vienna 2026 | Slava Moskvin

whoami

Slava Moskvin

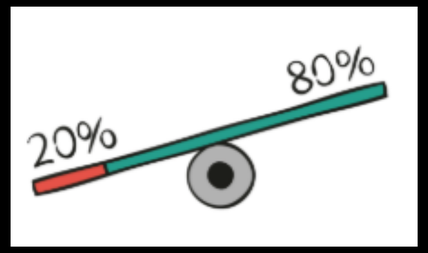
Security researcher in embedded and automotive

- 60+ CVEs

Linux kernel research as a hobby

What this talk is about

- Mutational, coverage-guided network fuzzer for the Linux kernel
- 3 remote kernel bugs
  - including CVE-2025-22037 in ksmbd
- Pareto Principle in effect



Motivation

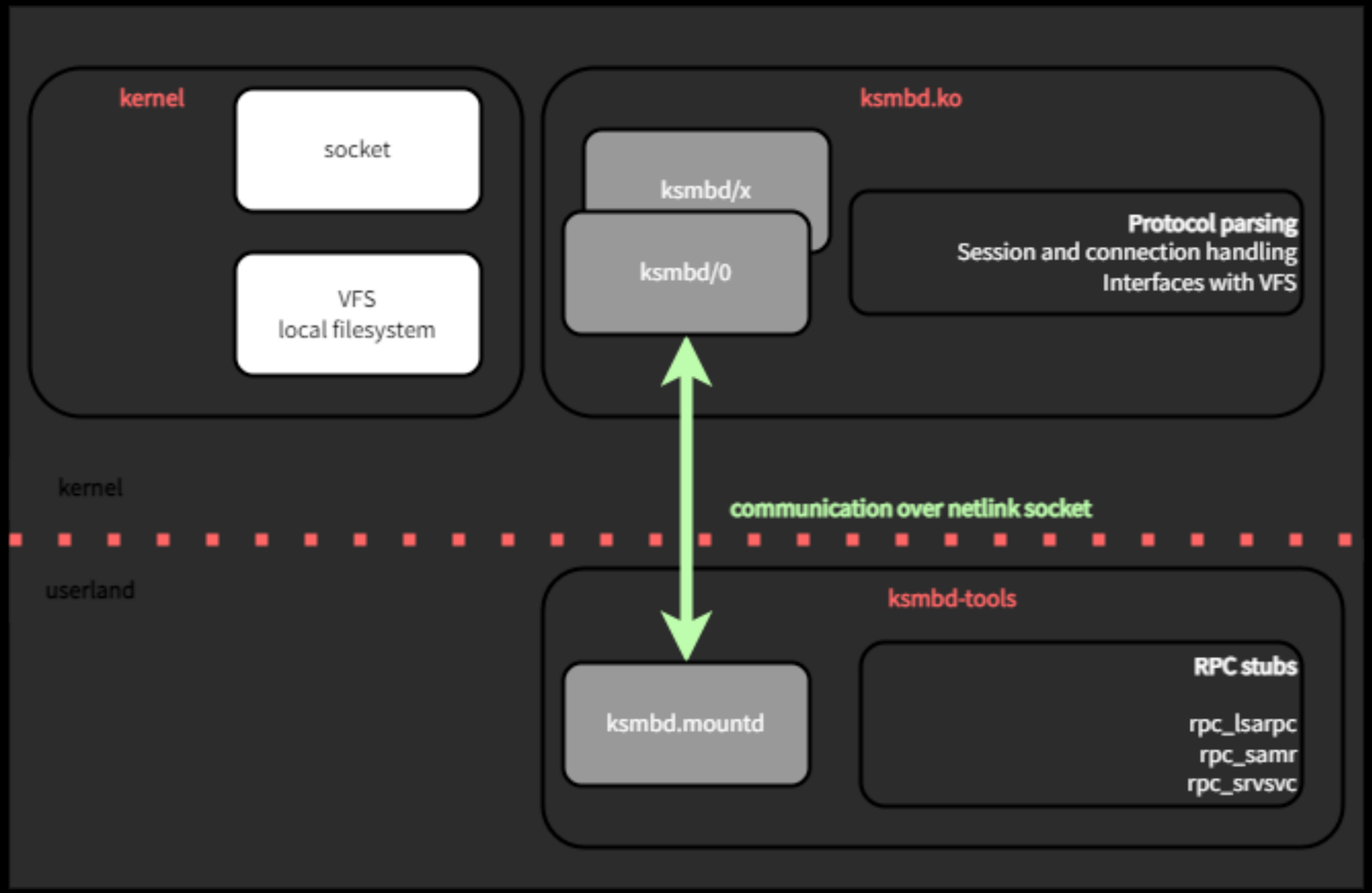
- Manual code analysis -> only good for 3rd-party modules
- Off-the-shelf tools -> limited results
- Options:
  - forking existing tools
  - writing my own

The Idea

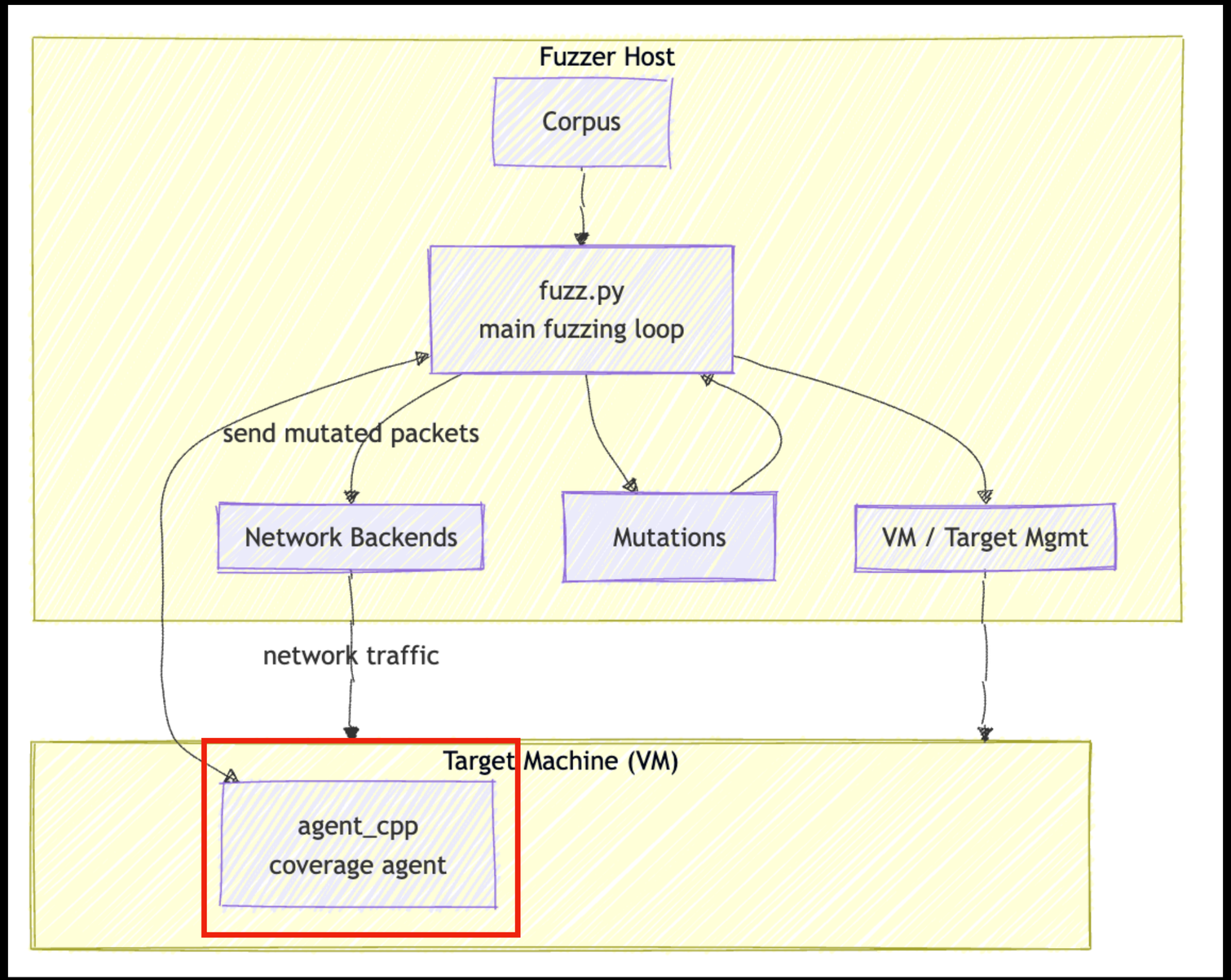
- Network fuzzing, with coverage!
- Simplicity over everything:
  - Minimal target modification
  - No extensive protocol research required
  - Cutting corners
- Not syzkaller with extra steps:
  - No grammar

Primary Target: ksmbd

SMB server/client implementation in kernel space

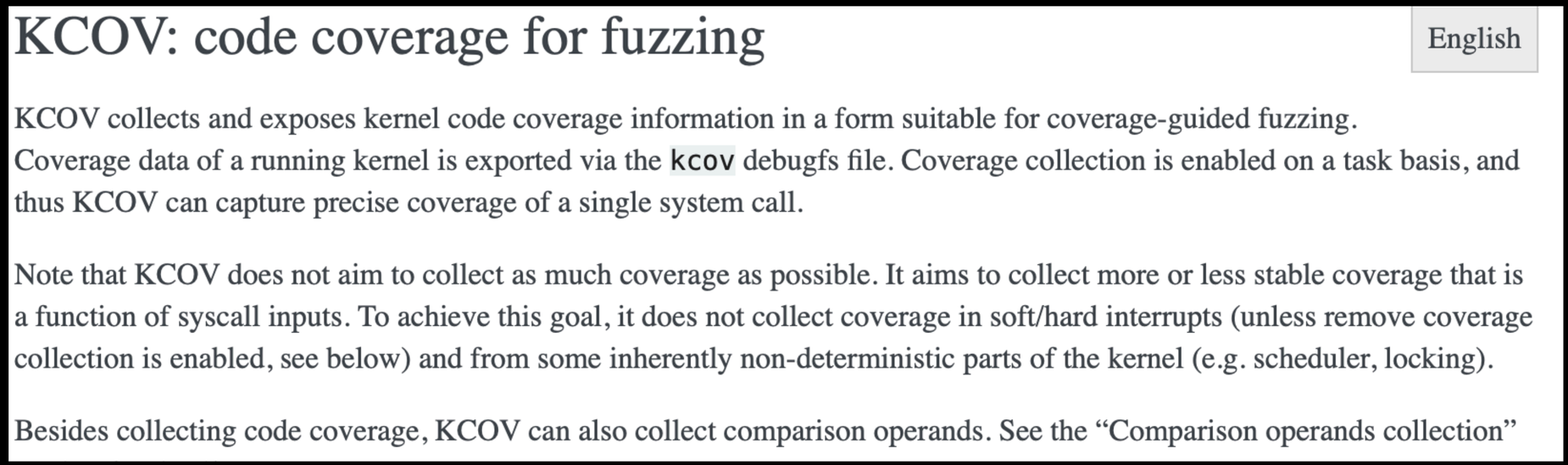


Architecture



Coverage: KCOV

- https://docs.kernel.org/dev-tools/kcov.html



KCOV: code coverage for fuzzing

English

KCOV collects and exposes kernel code coverage information in a form suitable for coverage-guided fuzzing.
Coverage data of a running kernel is exported via the `kcov` debugfs file. Coverage collection is enabled on a task basis, and thus KCOV can capture precise coverage of a single system call.

Note that KCOV does not aim to collect as much coverage as possible. It aims to collect more or less stable coverage that is a function of syscall inputs. To achieve this goal, it does not collect coverage in soft/hard interrupts (unless remove coverage collection is enabled, see below) and from some inherently non-deterministic parts of the kernel (e.g. scheduler, locking).

Besides collecting code coverage, KCOV can also collect comparison operands. See the “Comparison operands collection”

Normal KCOV

```
fd = open("/sys/kernel/debug/kcov", O_RDWR);
ioctl(fd, KCOV_ENABLE, KCOV_TRACE_PC);
// fuzz here
ioctl(fd, KCOV_DISABLE, 0);
n = __atomic_load_n(&cover[0], __ATOMIC_RELAXED);
for (i = 0; i < n; i++)
        printf("0x%lx\n", cover[i + 1]);
```

Remote KCOV

```
int ksmbd_conn_handler_loop(void *p)
{
    ...
    kcov_remote_start(handle);
    ...
    kcov_remote_stop();

out:
...
}
```

Remote KCOV

```
int ksmbd_conn_handler_loop(void *p)
{
    ...
    kcov_remote_start(handle);
    ...
    kcov_remote_stop();

out:
...
}
```

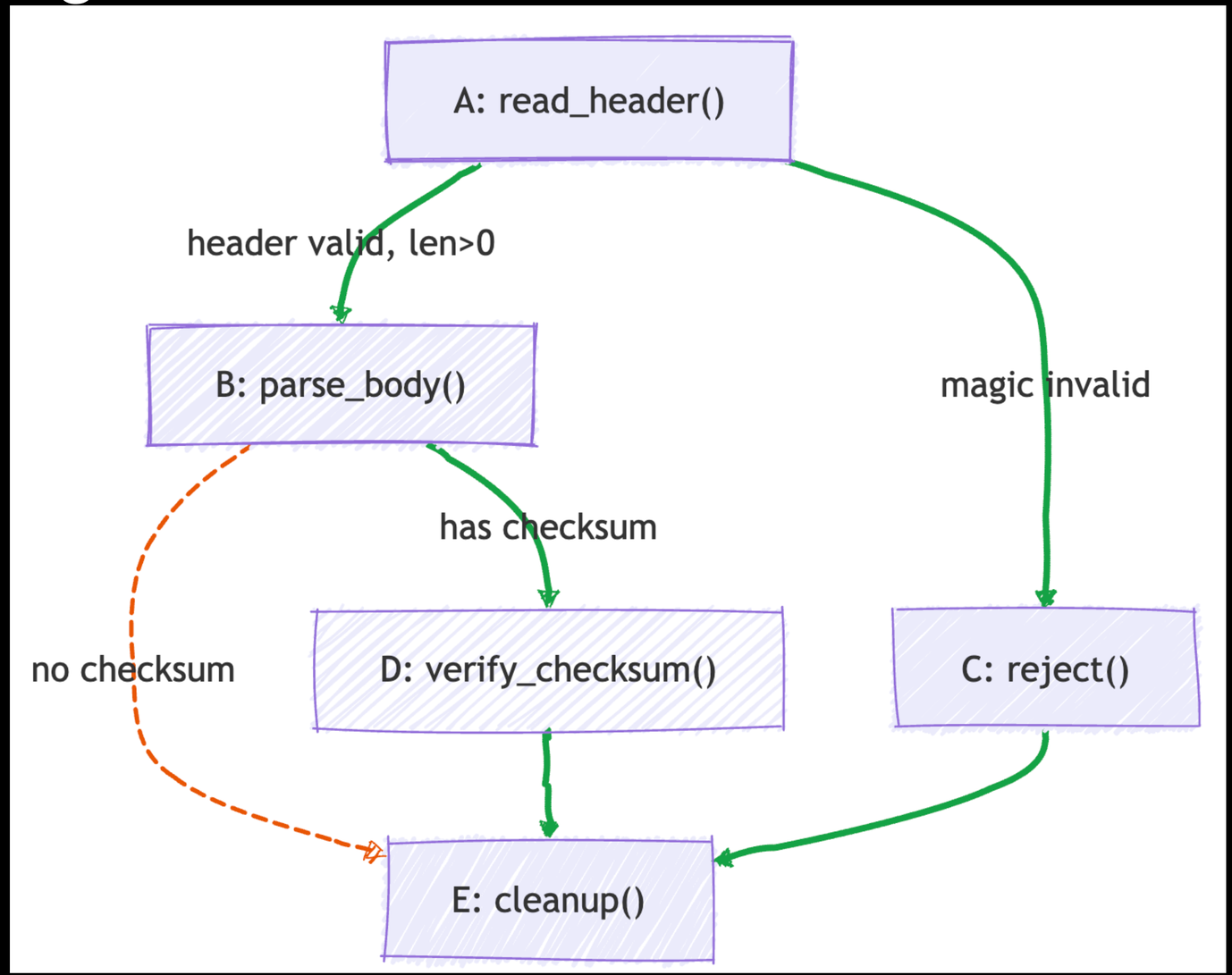
PCs -> Feedback

```
→  hfs+ sudo ./kcov_test 2>/dev/null
0xffff80008005f614
0xffff80008005f12c
0xffff80008005f168
0xffff80008005f184
0xffff80008005f2c8
0xffff80008005e4dc
0xffff80008005e528
0xffff80008005e56c
0xffff80008005e594
0xffff80008005e5e4
```

PCs -> Feedback

Basic block vs Edge coverage

- BB coverage:
  - Marking BBs hit
  - A->B->D->E = A->B->E
- Edge coverage:
  - Marking branches
  - A->B->D->E != A->B->E



PCs -> Feedback

- Basic block coverage

```
uint64_t ncov = __atomic_load_n(&kcov_data[0], __ATOMIC_RELAXED);
for (size_t i = 0; i < ncov; i++) {
    idx = pc % STORED_COVER_SIZE;
    cov_bitmap[idx] = 1;
}
```

- Edge coverage = ?

```
cur_location = <COMPILE_TIME_RANDOM>;
shared_mem[cur_location ^ prev_location]++;
prev_location = cur_location >> 1;
```

PCs -> Feedback

Edge coverage

- Solution: hash PCs

```
idx = (size_t)(skcov_hash(prev_pc, pc) % ctx->cfg.cover_size);
ctx->curr_map[idx] = 1;
```

Problems Start

- Attempt 1: Per-susystem KCOV
  - echo "KCOV_INSTRUMENT := y" >> Makefile
  - 0 coverage
- Attempt 2: KCOV_INSTRUMENT_ALL=y
  - Great coverage
  - Either a genius fuzzer, or noise

Problems Start

Great coverage?

```
INFO: Time spent fuzzing: 1468.17 seconds
INFO: Generated 11579, discovered 5151, crashes: 0
INFO: Strategy stats: bit_flip: 922, byte_flip: 1098, radamsa: 997, dictionary: 1036, havoc: 1098, shuffle_packets:
INFO: Mutating session 1363428922350900019 with strategy shuffle_packets
DEBUG:CoverageCollector:HASN response: b'\x00\x00'
INFO:CoverageCollector:HASN: has_cov=False, has_crash=False
```

Too much coverage:

- noise from unrelated subsystems

Limiting the noise

- address ranges

```
~/proj/net-shredder (main*) » uv run python utils/parse_dwarfdump_clusters.py utils/dwarfdump_fs_smb.txt 0x100
Found 1589 address range(s):
Range 1: 0xffff800080022fec - 0xffff800080022fec
Range 2: 0xffff800080dc2780 - 0xffff800080dc2780
Range 3: 0xffff800080dc2d80 - 0xffff800080dc2d80
```

...

```
Range 1585: 0xffff800084e2cba0 - 0xffff800084e2cba0
Range 1586: 0xffff800084e2cd20 - 0xffff800084e2cd20
Range 1587: 0xffff800084e2cf60 - 0xffff800084e2d020
Range 1588: 0xffff800084e2d1a0 - 0xffff800084e2d1a0
Range 1589: 0xffff800084e2d7a0 - 0xffff800084e2d7e0
```

...

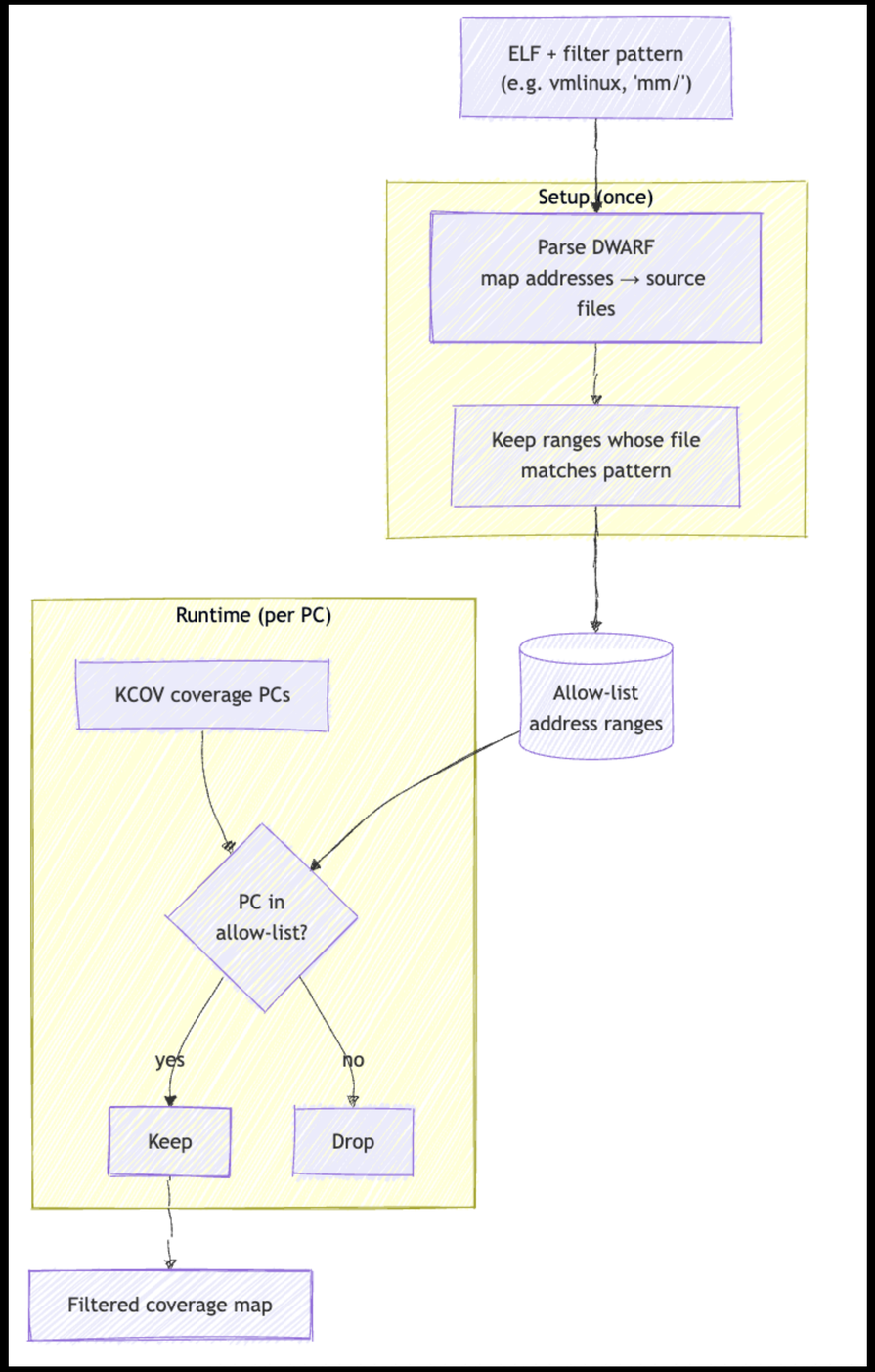
- addr2line

```
~/proj/net-shredder/data (main*) » addr2line -e vmlinux -a 0xffff80008138b570
0xffff80008138b570
/root/linux/fs/smb/server/smb2pdu.c:8924
```

Limiting the noise

Limiting the noise

Solution = reimplement important bits with libdwarf



Agent: Communication

Network Protocol

- COON/COOF - KCOV on/off
- HASN - new cov? new crash?
- GECO/GECU - get best coverage map, get current coverage map
- GECR - get crash data

skcov (Simple KCOV)

```
cfg.kcov_trace_mode = KCOV_TRACE_PC;
cfg.type = SKCOV_TYPE_EDGE_WITHOUT_COUNT;
cfg.cover_size = 256u * 1024u;
cfg.dwarf_elf_path = "/boot/vmlinux";
cfg.filter_pattern = "fs/smb";
cfg.dump_covered_pcs = true;
cfg.pc_dump_path = "/tmp/skcov-filtered-pcs.txt";
```

skcov (Simple KCOV)

```
skcov_init(&cfg, &ctx);
skcov_start(ctx);
/* Trigger your target code here. */
skcov_stop(ctx);

new_coverage = skcov_has_new_coverage(ctx);
current = skcov_get_current_coverage(ctx);
best = skcov_get_best_coverage(ctx);
```

skcov (Simple KCOV)

```python
def main() -> None:
    config = create_filtered_config(
        dwarf_elf_path="/boot/vmlinux",
        filter_pattern="fs/",
        dump_covered_pcs=True,
        pc_dump_path="/tmp/skcov-filtered-pcs.txt",
    )

    with Skcov(config) as cov:
        cov.start()
        # Trigger your target code here.
        cov.stop()
        cov.has_new_coverage()
        cov.dump_covered_pcs()
        print("covered pcs:", cov.get_covered_pcs_count())
```

Bugs found

ksmbd Remote DoS

```
INFO: Time spent fuzzing: 9853.21 seconds
INFO: Generated 73974, discovered 5972, rejected: 2473, corpus size: 4423, crashes: 3
INFO: Strategy stats: bit_flip: 1032, byte_flip: 1225, radamsa: 1129, dictionary: 1187,
INFO: Mutating session -8569154040998113951 with strategy dictionary
DEBUG:CoverageCollector:HASN response: b'\x00\x00'
```

```
[  258.945471] ==================================================================
[  258.956361] BUG: KASAN: null-ptr-deref in kmemdup_noprof+0x60/0x80
[  258.960464] Read of size 64 at addr 0000000000000002 by task kworker/2:1/106

[  258.964310] CPU: 2 UID: 0 PID: 106 Comm: kworker/2:1 Not tainted 6.14.0-rc6-ksmbd-fuzz #13
[  258.964328] Hardware name: QEMU QEMU Virtual Machine, BIOS 0.0.0 02/06/2015
[  258.964337] Workqueue: ksmbd-io handle_ksmbd_work
[  258.964360] Call trace:
[  258.964365]  show_stack+0x2c/0x40 (C)
[  258.964388]  dump_stack_lvl+0xd4/0x12c
[  258.964408]  print_report+0x520/0x540
[  258.964428]  kasan_report+0xb8/0x1a8
[  258.964447]  kasan_check_range+0x100/0x1a8
[  258.964466]  __asan_memcpy+0x3c/0xa0
[  258.964485]  kmemdup_noprof+0x60/0x80
[  258.964498]  smb2_sess_setup+0x1090/0x3808
[  258.964512]  handle_ksmbd_work+0x1000/0x11e4
[  258.964527]  process_one_work+0x85c/0x1a70
[  258.964545]  worker_thread+0x554/0xcc8
[  258.964559]  kthread+0x414/0x7a0
[  258.964581]  ret_from_fork+0x10/0x20
[  258.964604] ==================================================================
[  258.990839] Disabling lock debugging due to kernel taint
[  258.990857] generate_preauth_hash:1331: preauth_hash=00000000bea51736
[  258.990872] ksmbd_gen_preauth_integrity_hash
[  258.990892] Unable to handle kernel paging request at virtual address dfff800000000000
[  258.993646] KASAN: null-ptr-deref in range [0x0000000000000000-0x0000000000000007]
[  258.995895] Mem abort info:
```

Bugs found

ksmbd Remote DoS

```
static int alloc_preauth_hash(struct ksmbd_session *sess,
                              struct ksmbd_conn *conn)
{
    if (sess->Preauth_HashValue)
        return 0;

    sess->Preauth_HashValue = kmemdup(conn->preauth_info->Preauth_HashValue,
                                      PREAUTH_HASHVALUE_SIZE, KSMBD_DEFAULT_GFP);
    if (!sess->Preauth_HashValue)
        return -ENOMEM;

    return 0;
}
```

Bugs found

ksmbd Remote DoS

```
// int smb2_handle_negotiate(struct ksmbd_work *work)

    if (conn->dialect == SMB311_PROT_ID) {
        unsigned int nego_ctxt_off = le32_to_cpu(req->NegotiateContextOffset);

        if (smb2_buf_len < nego_ctxt_off) {
            ...
            goto err_out;
        }
        if (smb2_neg_size > nego_ctxt_off) {
            ...
            goto err_out;
        }
        if (smb2_neg_size + le16_to_cpu(req->DialectCount) * sizeof(__le16) >
            nego_ctxt_off) {
            ...
            goto err_out;
        }
    } else {
        if (smb2_neg_size + le16_to_cpu(req->DialectCount) * sizeof(__le16) >
            smb2_buf_len) {
            ...
            goto err_out;
        }
    }
```

Bugs found

ksmbd Remote DoS

```
// int smb2_handle_negotiate(struct ksmbd_work *work)
    if (conn->dialect == SMB311_PROT_ID) {
        ...
    }

    switch (conn->dialect) {
    case SMB311_PROT_ID:
        conn->preauth_info =
            kzalloc(sizeof(struct preauth_integrity_info),
                KSMBD_DEFAULT_GFP);
```

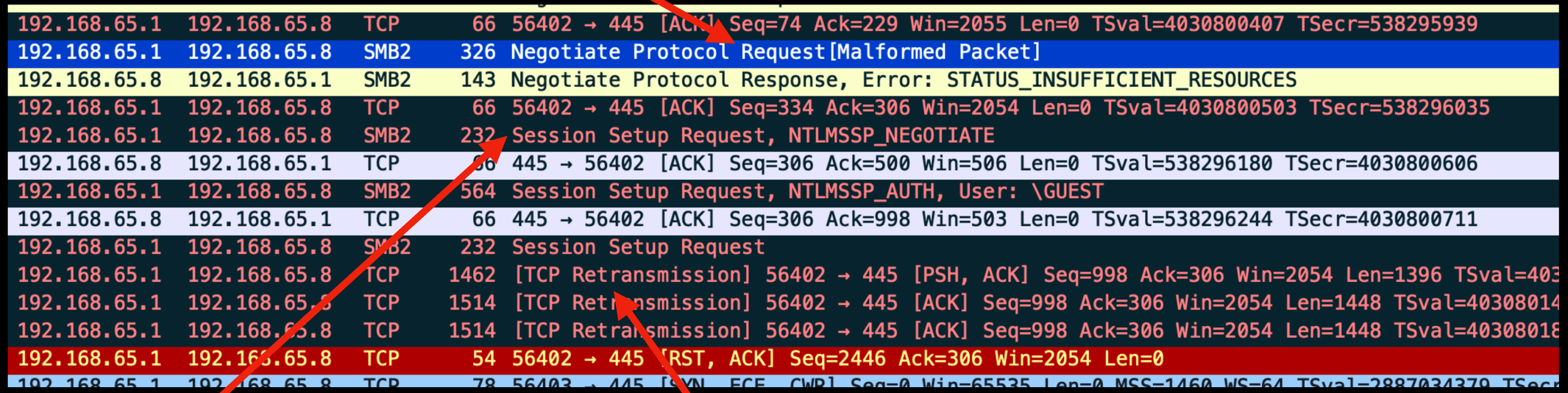
Bugs found
ksmbd Remote DoS

- smb2_handle_negotiate()
  - connection init
  - conn->preauth_info = 0
- smb2_sess_setup() -> generate_preauth_hash() -> alloc_preauth_hash()
  - NULL deref
- Remote kernel panic

Bugs found

ksmbd Remote DoS

smb2_handle_negotiate()

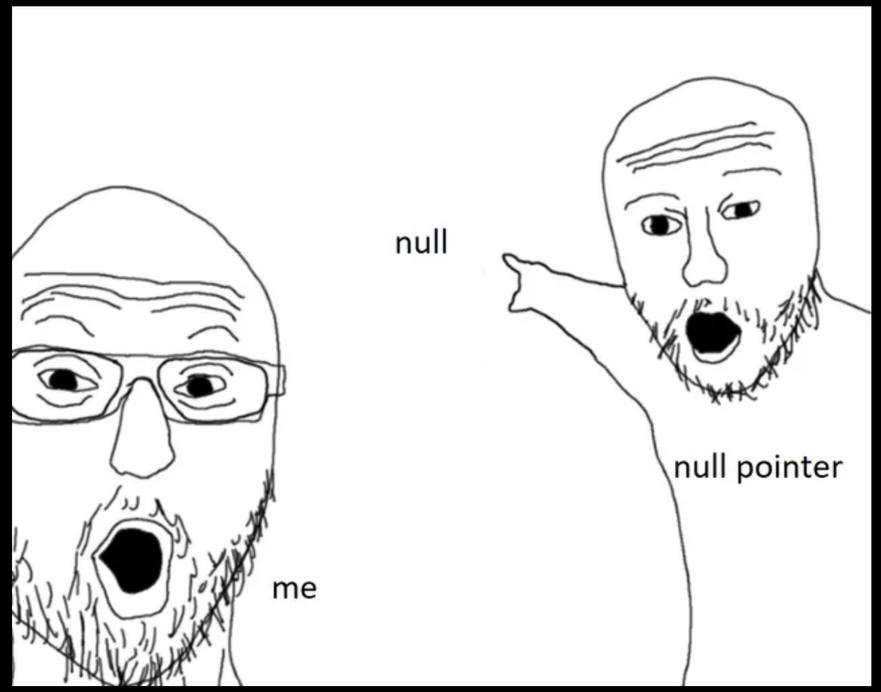


smb2_sess_setup()

target down

More Bugs

- ksmbd: another bug (NULL deref?):
  - service-level DoS
  - fixed by CVE-2025-22037 patch
- NFS: NULL deref in svc_process_common()
  - Triggers only on first connection to NFS after reboot



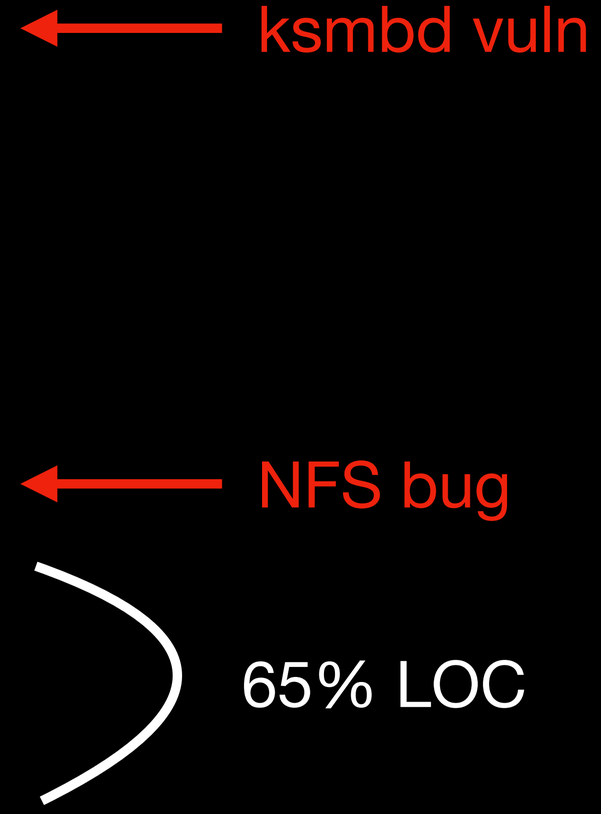

```
Job nfs-server.service/stop running (3min 31s / 4min 30s)
```

What worked well

- End-to-end fuzzing idea
- Bugs in high-profile targets
  - Network fuzzing is somewhat interesting*
  - *I’d have said in 2025

Development Timeline

Period	Dev focus	Activity (commits / LOC)
Jan 2025	Project scaffolding, crash detection, replay-pcap	2 / 2.0k 🟧
Feb 2025	Core engine — base mutation, dict, radamsa, AFL++ havoc, edge cov, packet checker, periodic reboot	**54 / 2.8k** 🔴
Mar 2025	Distributed fuzzing start — manager, VM spawning, python agent	13 / 1.6k 🟨
Apr 2025	Manager fixes	1 / 71 ⬜
May 2025	Client mode, config arg, frida-cov	3 / 362 ⬜
Jun 2025	Session/corpus rework, cov threshold, bias, nfs dict, UTM conn	20 / 1.8k 🟧
Jul 2025	Agent rewrite C++, libdwarf filter, stats DB, UTM backend	**31 / 8.3k** 🔴
Aug 2025	IPv6, ICMP support	20 / 1.2k 🟨
Sep 2025	skcov lib, local/remote cov collector, dump-PCs, fuzz-w/o-cov	**30 / 6.9k** 🔴



What I’d Change

- Fuzzing w/o becoming protocol expert
- Early results as oracle to future success
  - Depth <-> width
- Speed

Wrap-up

- skcov release
- Followed by net-shredder release
- Socials:
  - X/Twitter: @slava_moskvin_
  - LinkedIn: Vyacheslav Moskvin
- Questions?