Guide to Linux kernel exploitation

Ivor Canjuga

whoami

Twitter: @santaclzz

Undergrad student at FERI, University of Maribor

CTF player / challenge creator

Intern at Viris

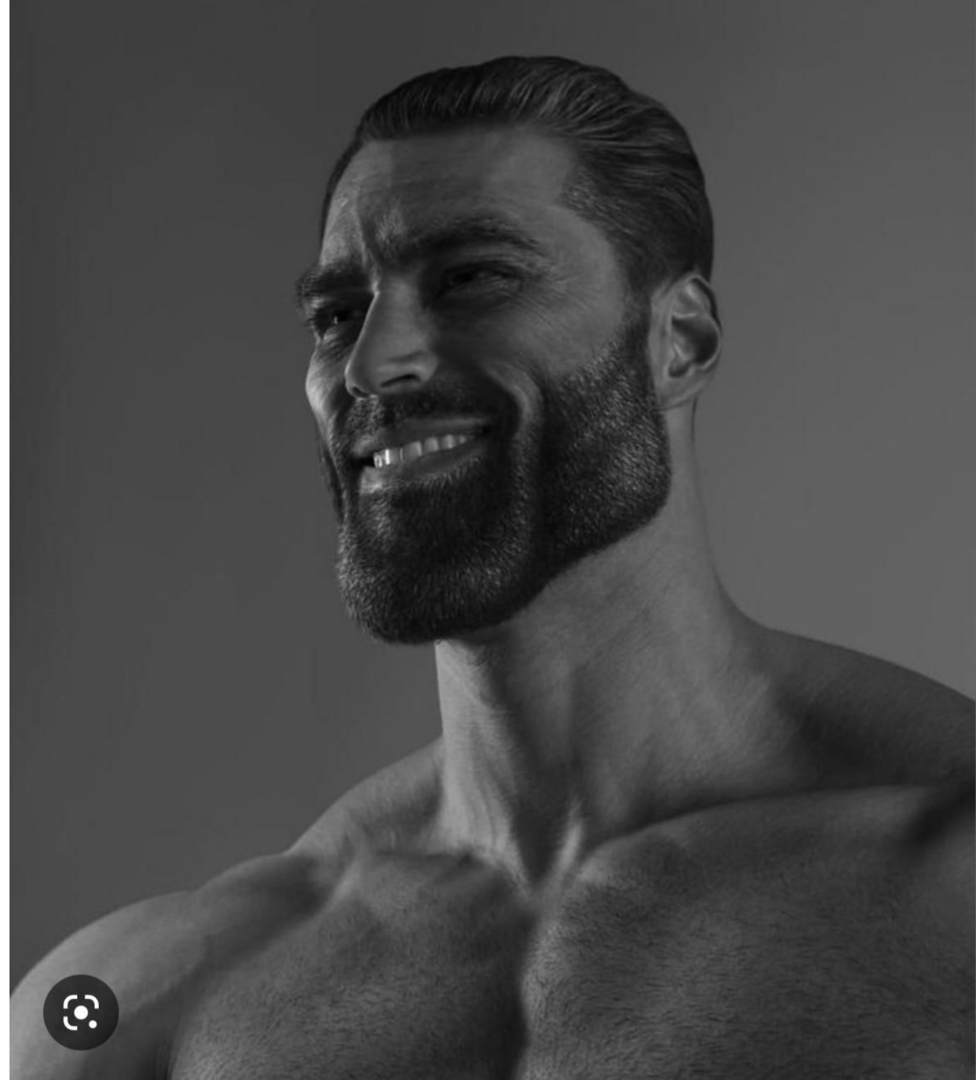
Why this presentation

To challenge myself

To share knowledge

Jk

To get a ticket before others :)



Kernel exploits: where are they used

Threat actors: to escalate privileges

Pentesters: to demonstrate impact

Defenders: coming up with detections and mitigations

Kernel / driver developers: to write patches

Android / iOS superusers: to customize their phone

Linux kernel oversimplified

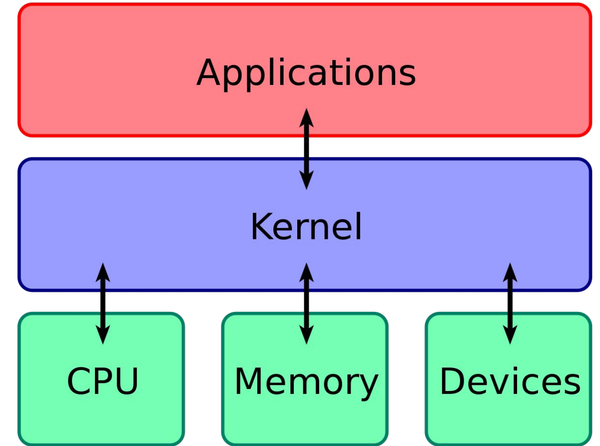
Layer between user applications and HW

Manages CPU, memory, devices,

file system, networking, process control...

Complex project with over 8 million lines of code

Still evolving



https://en.wikipedia.org/wiki/Kernel_(operating_system)

Differences from userspace

More instructions

More registers

More patience

Resources are shared

Bugs are the same

Goal of exploitation

Get root

Escape docker / k8 container

Escape seccomp / sandbox

Persistence / stealth

Goal of exploitation

```
commit_creds(prepare_kernel_cred(0))

current->thread_info.flags &= ~(1 << TIF_SECCOMP)

run_cmd(“/path_to_command”)
```

Attack surface

Kernel modules - read, write, ioctl

Syscalls - 398 syscalls

File system

Network drivers

USB device

Setup

Build the Linux kernel with debug symbols

a. git clone https://github.com/torvalds/linux

b. cd linux && make defconfig && make menuconfig

c. Ensure that kernel hacking --> Compile-time checks and compiler options --> Compile the kernel with debug symbols is checked.

d. make

Setup

filesystem

busybox

qemu

gdb

Debug

From userspace
From kernelspace
Symbols in /proc/kallsyms

```
1 #!/bin/sh
2
3 qemu-system-x86_64 \
4         -kernel ./bzImage \
5         -initrd ./dist.cpio.gz \
6         -monitor /dev/null \
7         -nographic -append "console=ttyS0 nokaslr" \
8         -s
9
```

-s Shorthand for -gdb tcp::1234, i.e. open a gdbserver on TCP port 1234.

(gdb) b *0x401d05

```
► 0x401d05    endbr64
  0x401d09    push   rbp
  0x401d0a    mov    rbp, rsp
  0x401d0d    lea    rdi, [rip + 0x932f0]
  0x401d14    call   0x411660

  0x401d19    mov    eax, 0
  0x401d1e    pop    rbp
  0x401d1f    ret

  0x401d20    push   rbx
  0x401d21    sub    rsp, 0x88
  0x401d28    test   rdi, rdi
```

Shellcode

Write kernel module

Compile it

Reverse engineer it

objdump -M intel -d test.ko

```c
 1 #include <linux/module.h>
 2 #include <linux/kernel.h>
 3 #include <linux/cred.h>
 4
 5 MODULE_LICENSE("GPL");
 6
 7 int test(void) {
 8         current->thread_info.flags &= ~(1 << TIF_SECCOMP);
 9         return 0;
10 }
```

```
Disassembly of section .text:

0000000000000000 <test>:
   0:   65 48 8b 04 25 00 00    mov    rax,QWORD PTR gs:0x0
   7:   00 00
   9:   48 81 20 ff fe ff ff    and    QWORD PTR [rax],0xfffffffffffffeff
  10:   31 c0                   xor    eax,eax
  12:   c3                      ret
```

```
[nya@neko kernel ]$ pwn asm -c amd64 -f string "mov    rax,QWORD PTR gs:0x0
and    QWORD PTR [rax],0xfffffffffffffeff
xor    eax,eax
ret
"
b'eH\x8b\x04%\x00\x00\x00\x00H\x81 \xff\xfe\xff\xff1\xc0\xc3'
[nya@neko kernel ]$
```

Bugs

Race conditions everywhere 🚨

modules

syscalls

SpinlockSafeExample.c

```c
1 DEFINE_SPINLOCK(mylock);
2 spin_lock(&mylock);
3
4 if (*a == 0) {
5   return -1;
6 }
7 ...
8 creds->uid = *a;
9 ...
10 commit_creds(creds);
11 spin_unlock(&mylock);
```

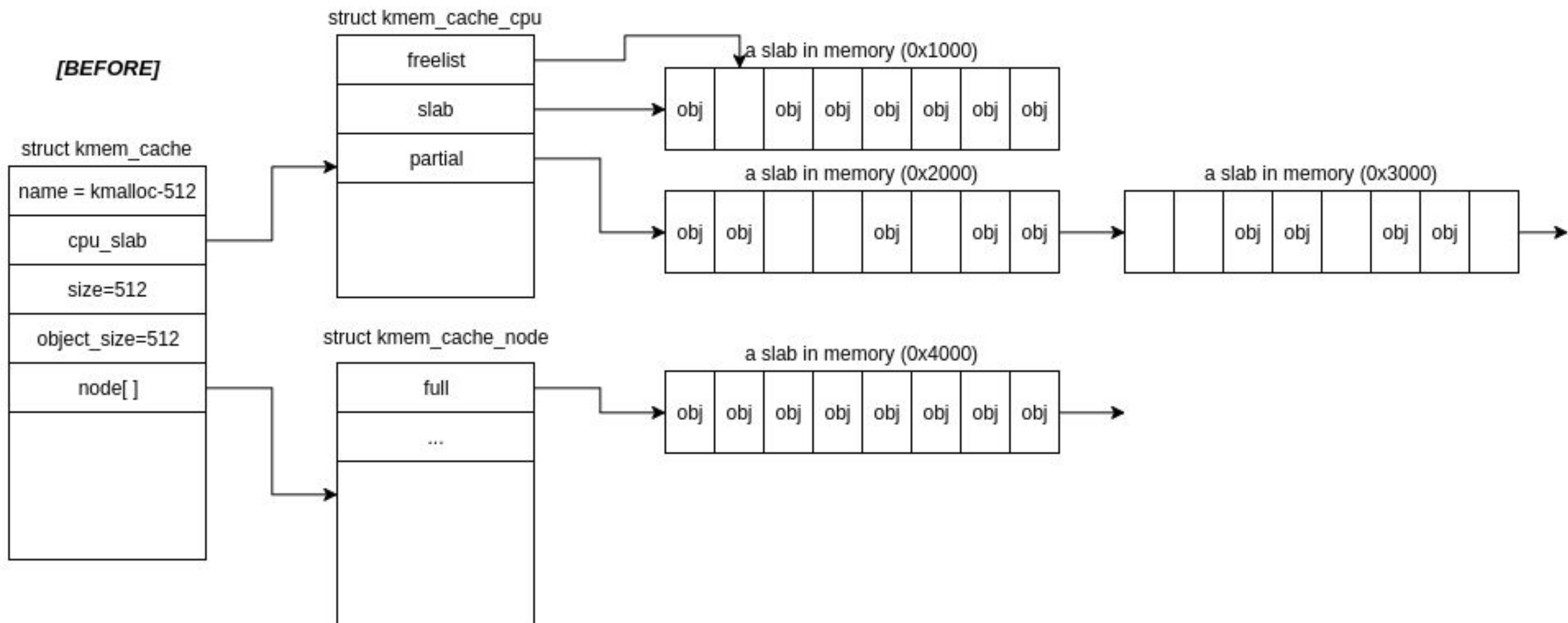
Kernel Heap

SLOB, SLUB, SLAB allocator

kmalloc() and vmalloc()

kfree(), vfree() and kvfree()

```
[nya@neko ~ ]$ sudo head -n 2 /proc/slabinfo && sudo grep "^kmalloc-[0-9]" /proc/slabinfo
slabinfo - version: 2.1
# name            <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab> : tunables <limit>
kmalloc-8k           350    364   8192    4    8 : tunables    0    0    0 : slabdata     91     91
kmalloc-4k          3939   3976   4096    8    8 : tunables    0    0    0 : slabdata    497    497
kmalloc-2k          2657   2704   2048   16    8 : tunables    0    0    0 : slabdata    169    169
kmalloc-1k          2675   2688   1024   32    8 : tunables    0    0    0 : slabdata     84     84
kmalloc-512        27589  27680    512   32    4 : tunables    0    0    0 : slabdata    865    865
kmalloc-256        88155  88192    256   32    2 : tunables    0    0    0 : slabdata   2756   2756
kmalloc-192       157500 157500    192   21    1 : tunables    0    0    0 : slabdata   7500   7500
kmalloc-128         3040   3040    128   32    1 : tunables    0    0    0 : slabdata     95     95
kmalloc-96          5040   5040     96   42    1 : tunables    0    0    0 : slabdata    120    120
kmalloc-64         19783  20864     64   64    1 : tunables    0    0    0 : slabdata    326    326
kmalloc-32         54784  54784     32  128    1 : tunables    0    0    0 : slabdata    428    428
kmalloc-16         18176  18176     16  256    1 : tunables    0    0    0 : slabdata     71     71
kmalloc-8          13312  13312      8  512    1 : tunables    0    0    0 : slabdata     26     26
```



https://sam4k.com/linternals-memory-allocators-0x02/

Heap exploitation

Double Free, Use After Free, Heap Overflow

1) Find struct with the “same” size
2) See what you can do with it
3) Spray the heap

Mitigations

KASLR, FG-KASLR

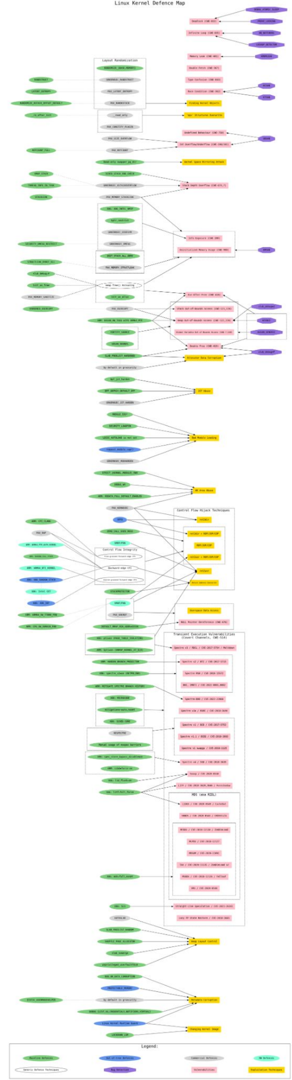
Kernel Stack Canary

SMEP - Execution Protection, cr4

SMAP - Access Protection, cr4

KPTI - isolated page tables

https://github.com/a13xp0p0v/linux-kernel-defence-map



Ret2user

Bypass everything

Return to userland

```
syscall   call
swapgs    leave
iretq     ret
```

Side channel attacks

spectre & meltdown

still not seen in the wild

/proc/cpuinfo

```
vmx flags       : vnmi preemption_timer invvpid ept_x_only ept_ad ept_1g
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l
bogomips        : 3999.93
```

Fuzzing

KCOV

KASAN

AFL

Syzkaller

Syzbot

Buzzer - eBPF

References

https://blog.trailofbits.com/2019/07/19/understanding-docker-container-escapes/

https://sam4k.com/linternals-memory-allocators-0x02/

https://lkmidas.github.io/posts/20210205-linux-kernel-pwn-part-3/

https://seal9055.com/blog/kernel/return_oriented_programming

https://breaking-bits.gitbook.io/breaking-bits/exploit-development/linux-kernel-exploit-development/kernel-page-table-isolation-kpti#kpti-trampoline

https://ptr-yudai.hatenablog.com/entry/2020/03/16/165628

https://pwn.college/system-security/kernel-security

https://github.com/google/syzkaller/

https://research.nccgroup.com/2018/09/11/ncc-groups-exploit-development-capability-why-and-what/

https://lwn.net/Articles/824307/

https://meltdownattack.com/meltdown.pdf