

# Computer Architecture and Operating Systems

## MONSOON SEMESTER 2019

INSTRUCTOR: DR. Sambuddho Chakravarty

### Custom System Call

To find and print content from the `task_struct` of a process from its "pid" and save it to a file.

---

#### Synopsis of Implementation:

The `/kernel/pid.c` has defined the method `pid_task` which can be used to retrieve the `task_struct` of a process.

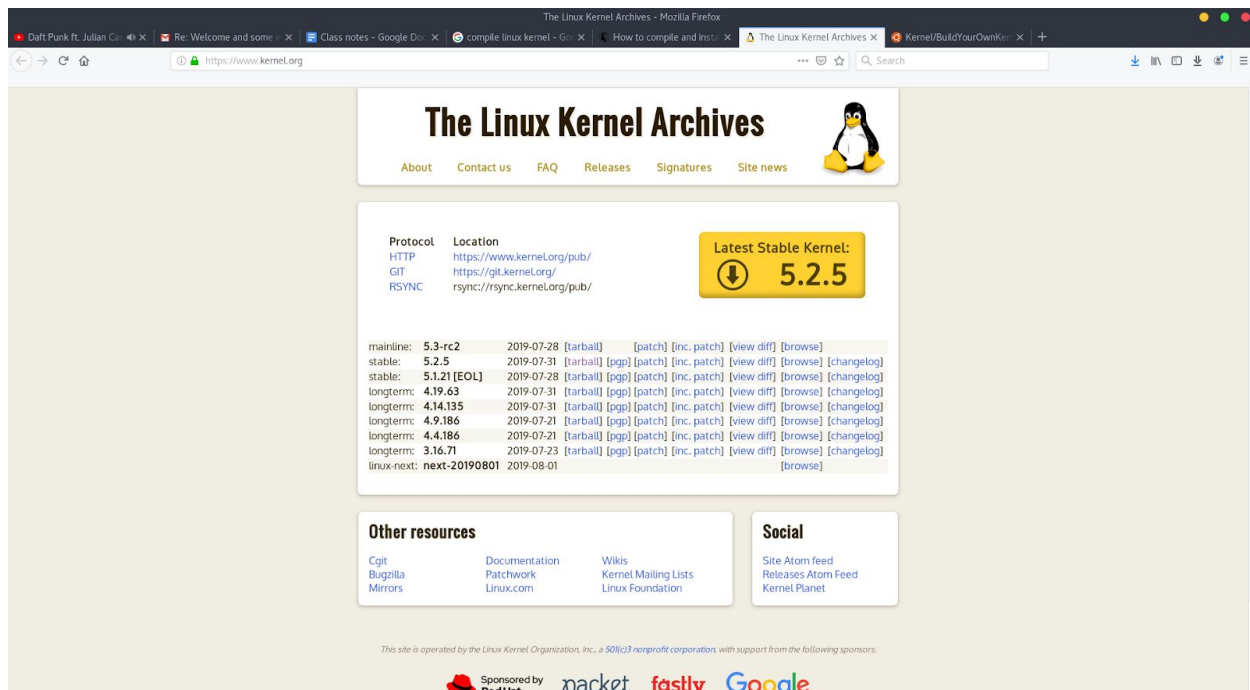
```
struct task_struct *pid_task(struct pid *pid, enum pid_type
type)
{
    //
}
```

The input process is then loaded in the task struct. The task struct loads from the `/include/linux/sched.h`. Any of the parameters of the `task_struct` specified in `sched.h` can be printed.

Since only other kernel system calls can be used in a system call, hence to write content to a file, a pointer to array of pointers of long is passed in the system call which is then populated by parameters of `task_struct` using the `copy_from_user` and `copy_to_user` syscalls.

---

## Step 1: Start by downloading the linux kernel from kernel.org.



## Step 2: Extract the kernel and make a copy of it.

```
root@Alium:~/Downloads# unxz -v linux-5.2.5.tar.xz balls
linux-5.2.5.tar.xz (1/1) with gpg command
100 % 102.1 MiB / 831.0 MiB = 0.123 127 MiB/s 0:06
root@Alium:~/Downloads#
```

---

### Step 3: Install the necessary development tools and GCC.

```
root@kali:~/Downloads/linux-5.2.5# apt-get install build-essential libncurses-dev bison flex libssl-dev libelf-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu cpp cpp-8 g++ g++-8 gcc gcc-8
  gcc-8-base gcc-9-base libasan5 libatomic1 libbinutils libbison-dev libcc1-0 libdw1
  libelf1 libfl-dev libfl2 libgcc-8-dev libgcc1 libgomp1 libisl19 libitm1 liblsan0
```

### Step 4: Create a new directory in one of the copies of the extracted linux kernel. Create the system call in this dir.

Step 5: Using `asmlinkage`, we tell compiler to look on the CPU stack for the function parameters, instead of registers.

Step 6: We pass the `pid(int)`, the `filename(char*)` and a `buffer(long**)` as parameters for the system call.

Step 7: Using `find_vpid`, we search for our valid pid in the `task_struct`. If the struct value returns `NULL`, then the pid is not valid.

---

**Step 8:** Since, only other system calls can be used, we use *printk* to print messages from the kernel which can be checked using *dmesg*.

**Step 9:** To populate our buffer, we use the *copy\_from\_user* and *copy\_to\_user* syscalls.

**Step 10:** Compile the syscall with the Makefile and fix any error that shows up.

**Step 11:** The custom syscall is added to the syscall table located at `/arch/x86/entry/syscalls/syscall_64.tbl`. Add the syscall at a new id.

```
333  common      sys_sh_task_info      sys_sh_task_info
```

**Step 12:** The custom syscall is to be now added to `syscalls.h` which is located at `/include/linux/syscalls.h`. Add the syscall before the `#endif`.

```
asmlinkage long sys_sh_task_info(int pid,char *filename,long **buf);  
  
#endif
```

---

**Step 14: Configure the Makefile of the kernel located at the parent directory. Find the below string:**

```
core-y          += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/  
block/
```

**And Replace it with :**

```
core-y          += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/  
block/ <directory with custom syscall>
```

## Step 15: Compile the linux kernel. Save the .config file using *make menuconfig* command.

```
.config - Linux/x86 5.2.5 Kernel Configuration

Linux/x86 5.2.5 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus
---). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for
Search. Legend: [*] built-in [ ] excluded <M> module < > module capable

*** Compiler: gcc (Debian 8.3.0-19) 8.3.0 ***
General setup --->
[*] 64-bit kernel
Processor type and features --->
Power management and ACPI options --->
Bus options (PCI etc.) --->
Binary Emulations --->
Firmware Drivers --->
[*] Virtualization --->
General architecture-dependent options --->
[*] Enable loadable module support --->
[*] Enable the block layer --->
IO Schedulers --->
Executable file formats --->
Memory Management options --->
[*] Networking support --->
Device Drivers --->
File systems --->
Security options --->
-*- Cryptographic API --->
Library routines --->
Kernel hacking --->

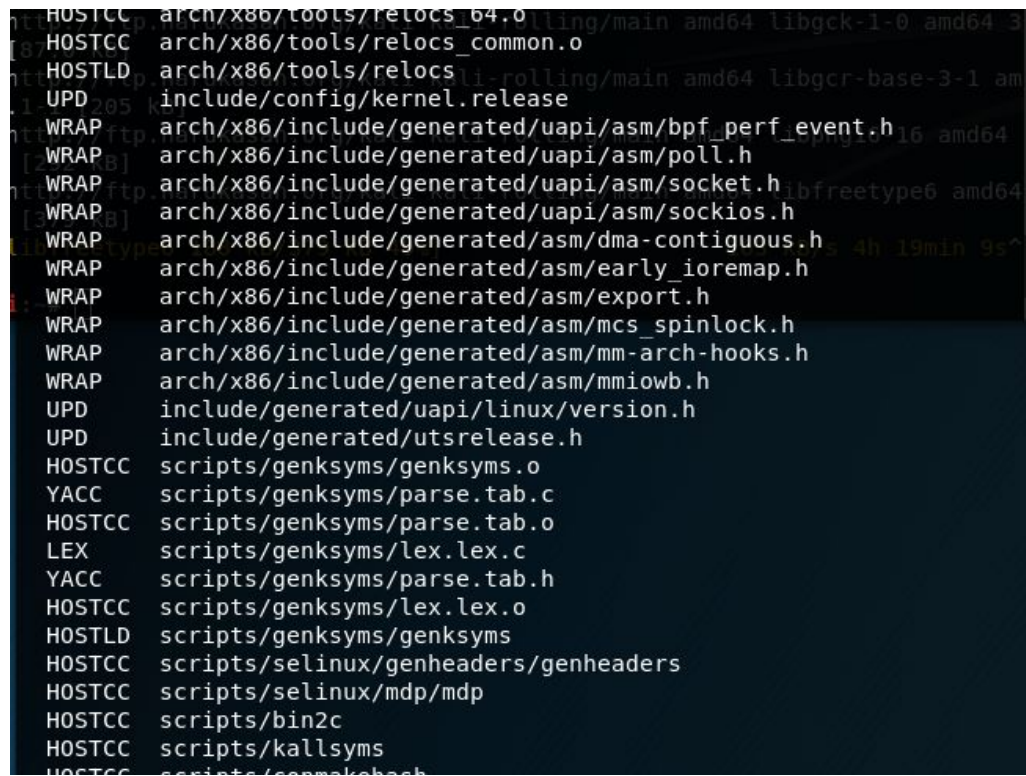
<Select> < Exit > < Help > < Save > < Load >
```

---

## Step 16: Customize the kernel at the menuconfig.

## Step 17: Compile the kernel.

Command: **make -j \$(nproc)**

A terminal window showing the output of the 'make' command during kernel compilation. The output lists various files being processed, including headers, source files, and object files. The files are organized into categories like 'HOSTCC', 'YACC', 'LEX', and 'WRAP'. The terminal has a dark background with light-colored text. A vertical scrollbar is visible on the right side of the terminal window.

```
HOSTCC arch/x86/tools/relocs_164.o
HOSTCC arch/x86/tools/relocs_common.o
HOSTLD arch/x86/tools/relocs
UPD include/config/kernel.release
WRAP arch/x86/include/generated/uapi/asm/bpf_perf_event.h
WRAP arch/x86/include/generated/uapi/asm/poll.h
WRAP arch/x86/include/generated/uapi/asm/socket.h
WRAP arch/x86/include/generated/uapi/asm/sockios.h
WRAP arch/x86/include/generated/asm/dma-contiguous.h
WRAP arch/x86/include/generated/asm/early_ioremap.h
WRAP arch/x86/include/generated/asm/export.h
WRAP arch/x86/include/generated/asm/mcs_spinlock.h
WRAP arch/x86/include/generated/asm/mm-arch-hooks.h
WRAP arch/x86/include/generated/asm/mmiowb.h
UPD include/generated/uapi/linux/version.h
UPD include/generated/utsrelease.h
HOSTCC scripts/genksyms/genksyms.o
YACC scripts/genksyms/parse.tab.c
HOSTCC scripts/genksyms/parse.tab.o
LEX scripts/genksyms/lex.lex.c
YACC scripts/genksyms/parse.tab.h
HOSTCC scripts/genksyms/lex.lex.o
HOSTLD scripts/genksyms/genksyms
HOSTCC scripts/selinux/genheaders/genheaders
HOSTCC scripts/selinux/mdp/mdp
HOSTCC scripts/bin2c
HOSTCC scripts/kallsyms
HOSTCC scripts/genmakehash
```

---

## Step 18: On successful compilation, install the modules.

Command: **make modules\_install**

```
root@kali:~/Downloads/linux-5.2.6# make -j $(nproc)
CALL      scripts/checksyscalls.sh
CALL      scripts/atomic/check-atomics.sh
DESCEND    objtool
CHK        include/generated/compile.h
Kernel: arch/x86/boot/bzImage is ready  (#1)
Building modules, stage 2.
MODPOST 3297 modules
root@kali:~/Downloads/linux-5.2.6# make modules_install
INSTALL arch/x86/crypto/aes-x86_64.ko
INSTALL arch/x86/crypto/aesni-intel.ko
INSTALL arch/x86/crypto/blowfish-x86_64.ko
INSTALL arch/x86/crypto/camellia-aesni-avx-x86_64.ko
INSTALL arch/x86/crypto/camellia-aesni-avx2.ko
INSTALL arch/x86/crypto/camellia-x86_64.ko
INSTALL arch/x86/crypto/cast5-avx-x86_64.ko
INSTALL arch/x86/crypto/cast6-avx-x86_64.ko
INSTALL arch/x86/crypto/chacha-x86_64.ko
INSTALL arch/x86/crypto/crc32-pclmul.ko
INSTALL arch/x86/crypto/crc32c-intel.ko
INSTALL arch/x86/crypto/crc10dif-pclmul.ko
```

## Step 19: Install the kernel and reboot.

Command: **make install**



---

```
root@kali:~/Downloads/linux-5.2.6# sudo make install
sh ./arch/x86/boot/install.sh 5.2.6 arch/x86/boot/bzImage \
    System.map "/boot"
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 5.2.6 /boot/vmlinuz-5.2.6
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 5.2.6 /boot/vmlinuz-5.2.6
update-initramfs: Generating /boot/initrd.img-5.2.6
WARNING: Setting CRYPTSETUP in /etc/initramfs-tools/initramfs.conf is deprecated
and will stop working in the future. Use /etc/cryptsetup-initramfs/conf-hook instead.
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 5.2.6 /boot/vmlinuz-5.2.6
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 5.2.6 /boot/vmlinuz-5.2.6
Generating grub configuration file ...
Found background image: /usr/share/images/desktop-base/desktop-grub.png
Found linux image: /boot/vmlinuz-5.2.6
Found initrd image: /boot/initrd.img-5.2.6
Found linux image: /boot/vmlinuz-4.15.0-kali2-amd64
Found initrd image: /boot/initrd.img-4.15.0-kali2-amd64
done
```

**A custom Linux has been successfully installed.**

**Step 20: Check the custom syscall in a test C file.**

**FIN!**

---

## Files: sys\_task\_info.c

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

#include<linux/init.h>

#include<linux/sched.h>

#include<linux/syscalls.h>

#include<linux/sched.h>

#include <linux/slab.h>

asmlinkage long sys_sh_task_info(int pid,char *filename,long **buf)

{

    struct task_struct *task;

    struct pid *pid_str;

    pid_str=find_vpid(pid);

    task=pid_task(pid_str, PIDTYPE_PID);

    if (task == NULL)

    {

        printk(KERN_ERR "No Process with given PID\n");

        return 3;

    }

    else

    {

        long buf_kernel[50];
```

---

```
long *user_ptrs[50];

unsigned long res;

int i;

printk(KERN_ERR "Process: %s\n", task->comm);

printk(KERN_ERR "PID: %ld\n", (long)task_pid_nr(task));

printk(KERN_ERR "Process State: %ld\n", (long)task->state);

printk(KERN_ERR "Priority: %ld\n", (long)task->prio);

printk(KERN_ERR "Process Exit State: %ld\n",
(long)task->exit_state);

printk(KERN_ERR "Process Exit Code: %ld\n",
(long)task->exit_code);

printk(KERN_ERR "Process Exit Signal: %ld\n",
(long)task->exit_signal);

printk(KERN_ERR "Process RT Priority: %ld\n",
(long)task->rt_priority);

printk(KERN_ERR "Process Static Priority: %ld\n",
(long)task->static_prio);

printk(KERN_ERR "Process Normal Priority: %ld\n",
(long)task->normal_prio);

printk(KERN_ERR "Creating File....");

buf_kernel[0]=(long)task_pid_nr(task);

buf_kernel[1]=(long)task->state;

buf_kernel[2]=(long)task->prio;
```

---

---

```
    buf_kernel[3]=(long)task->exit_state;

    buf_kernel[4]=(long)task->exit_code;

    buf_kernel[5]=(long)task->exit_signal;

    buf_kernel[6]=(long)task->rt_priority;

    buf_kernel[7]=(long)task->static_prio;

    buf_kernel[8]=(long)task->normal_prio;

    res = copy_from_user(user_ptrs,buf,sizeof(long *)* 50);

    for (i=0; i<50; i++)

    {

        res=copy_to_user(user_ptrs[i],&buf_kernel[i],sizeof(long));

    }

    return 0;

}

}
```

---

## Makefile for System Call

```
obj-y:= sys_sh_task_info.o
```

```
all:
```

```
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD)
```

```
modules
```

```
clean:
```

```
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

---

## Test C file for System Call

```
#include <stdio.h>

#include <linux/kernel.h>

#include <sys/syscall.h>

#include <unistd.h>

#include <stdlib.h>


#define sys_sh_task_info 333


int main()
{

    const int size = 10;

    long **buf = malloc(sizeof(long *) * size);

    for(int i=0; i<size; i++) buf[i] = malloc(sizeof(long));


    int sys = syscall(sys_sh_task_info, "232145", "lol.txt", buf);

    printf("%d\n", sys);

    if (sys==3)
    {
```

---

```
    printf("Error Code: %ld\n", sys);

    printf("Error Name: ESRCH 3 No such process.PID Wrong!");

}

else if(sys== -1)

{

    printf("Error Code: %ld\n", sys);

    printf("Error Name: EIO 5 Input/output error");

}

else

{

    printf("PID: %ld\n", *buf[0]);

    printf("Process State: %ld\n", *buf[1]);

    printf("Priority: %ld\n", *buf[2]);

    printf("Process Exit State: %ld\n", *buf[3]);

    printf("Process Exit Code: %ld\n", *buf[4]);

    printf("Process Exit Signal: %ld\n", *buf[5]);

    printf("Process RT Priority: %ld\n", *buf[6]);

    printf("Process Static Priority: %ld\n", *buf[7]);
```

---

```
printf("Process Normal Priority: %ld\n", *buf[8]);

printf("Creating File....\n");

FILE *fp = fopen("lol.txt", "ab+");

fprintf(fp, "-----\n");

fprintf(fp, "PID: %ld\n", *buf[0]);

fprintf(fp, "Process State: %ld\n", *buf[1]);

fprintf(fp, "Priority: %ld\n", *buf[2]);

fprintf(fp, "Process Exit State: %ld\n", *buf[3]);

fprintf(fp, "Process Exit Code: %ld\n", *buf[4]);

fprintf(fp, "Process Exit Signal: %ld\n", *buf[5]);

fprintf(fp, "Process RT Priority: %ld\n", *buf[6]);

fprintf(fp, "Process Static Priority: %ld\n", *buf[7]);

fprintf(fp, "Process Normal Priority: %ld\n", *buf[8]);

fclose(fp);

}
```



---

```
// printf("%ld", *buf[0]);  
  
return 0;  
  
}
```