Operating System Design & Implementation

Lab 6: Data Encryption and Decryption in the TCP Layer

TA: 羅濟韋

Objective:

In this Lab, you can

- Learn how to add a system call
- Learn the details of the socket structure
- Learn the data packet flow in the TCP layer

Lab 6.1 prepare experiment environment

In the following labs, we'll use *linux 2.6.34.15* as our experiment system kernel. Please switch your environment by yourself or follow the steps below to setup environment.

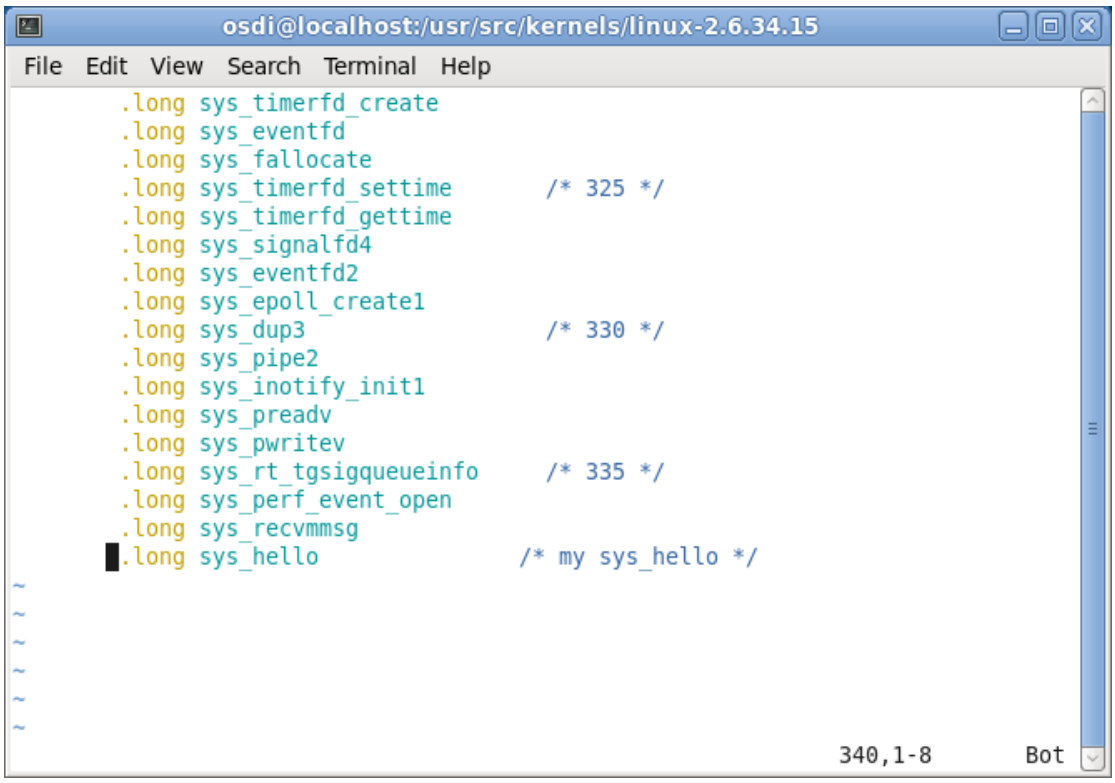
```
# cd /usr/src/kernels/
# wget
https://www.kernel.org/pub/linux/kernel/v2.6/longterm/v2.6.34/linux-2.6.34.15.tar.xz
# tar -xf linux-2.6.34.15.tar.xz
# cd linux-2.6.34.15
# make menuconfig    // just save and exit
# make modules
# make modules_install
# make
# make install
# vim /boot/grub/grub.conf    // modify timeout to 60 sec
```

After all, reboot your system and then you can switch to the new kernel.

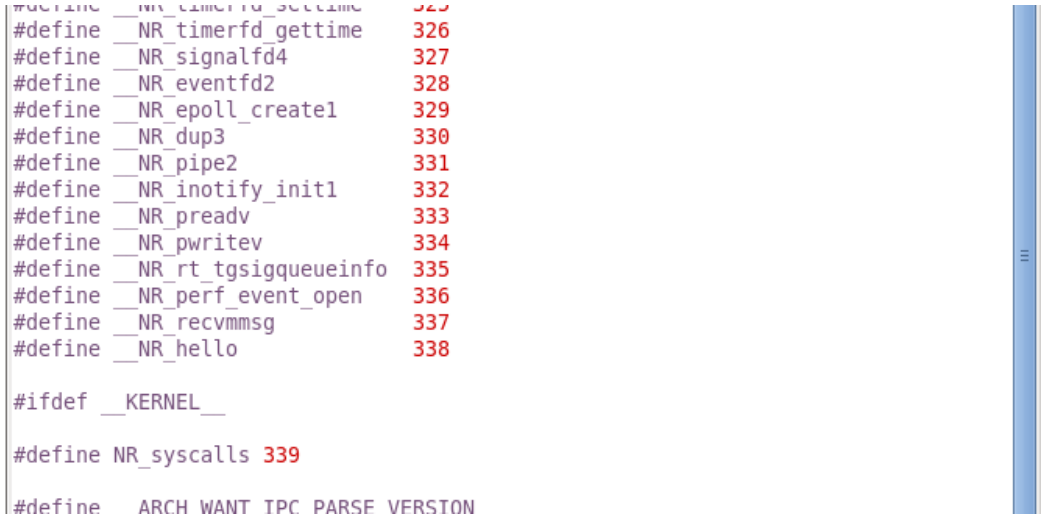
Lab 6.2 add system call

User space program can use system call to read/write/call kernel space data or function. In lab 6.2, you can learn how to implement a new system call.

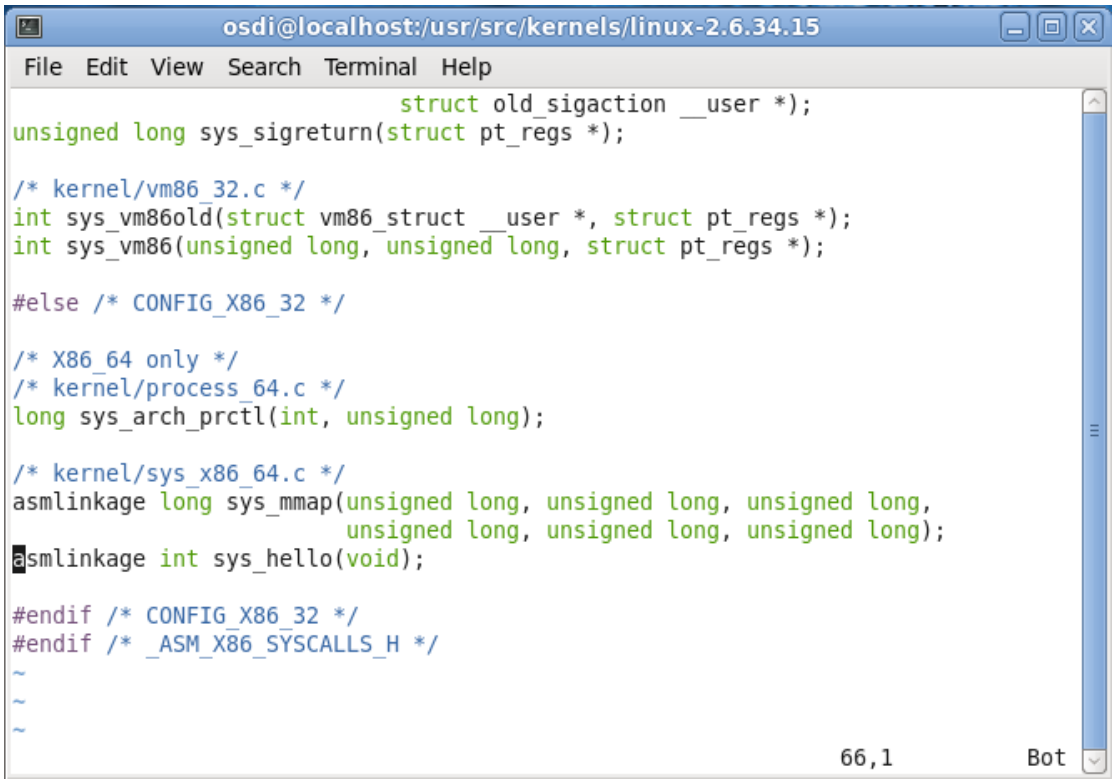
1. Define your new system call in system call table by vim arch/x86/kernel/syscall_table_32.S



2. Define your new system call in unistd_32.h by vim arch/x86/include/asm/unistd_32.h



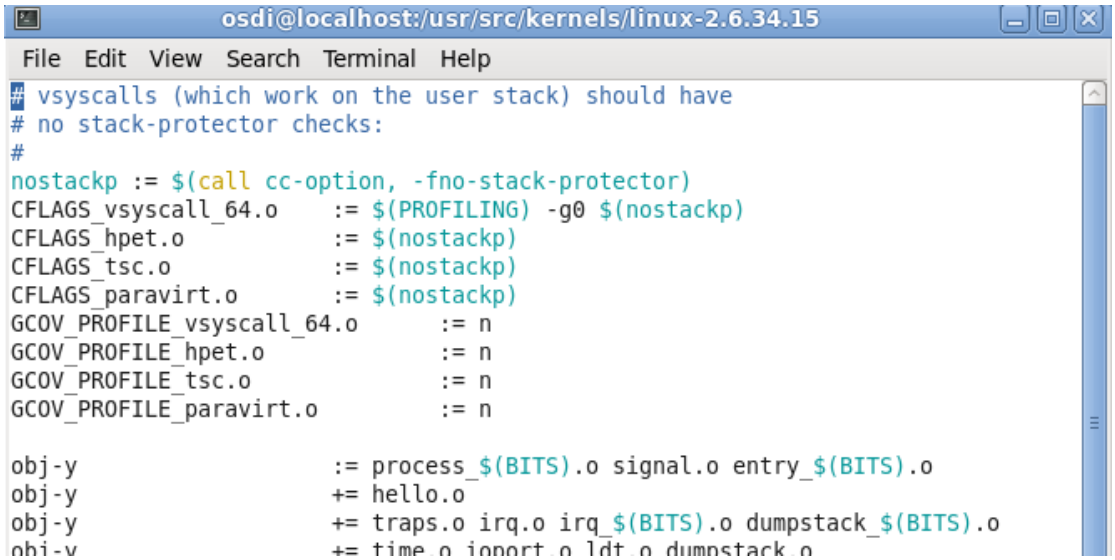
3. Define your new system call in syscalls.h by vim arch/x86/include/asm/syscalls.h



4. Implement your system calls in kernel.

<Hint> Please make sure to include <linux/kernel.h> and <linux/linkage.h> in your C file.

5. Add the new file to Makefile by vim arch/x86/kernel/Makefile



6. Define your system calls number for user space program in unistd_32.h by vim /usr/include/asm/unistd_32.h

```
#define __NR_utimensat          320
#define __NR_signalfd           321
#define __NR_timerfd_create     322
#define __NR_eventfd            323
#define __NR_fallocate          324
#define __NR_timerfd_settime    325
#define __NR_timerfd_gettime    326
#define __NR_signalfd4          327
#define __NR_eventfd2           328
#define __NR_epoll_create1      329
#define __NR_dup3               330
#define __NR_pipe2              331
#define __NR_inotify_init1      332
#define __NR_preadv             333
#define __NR_pwritev            334
#define __NR_rt_tgsigqueueinfo  335
#define __NR_perf_event_open    336
#define __NR_recvmmsg           337
#define __NR_hello              338

#endif /* _ASM_X86_UNISTD_32_H */
<els/linux-2.6.34.15/usr/include/asm/unistd_32.h" 348L, 9897C 346,1          Bot
```

7. Define your system calls for user program in syscall.h by vim /usr/include/bits/syscall.h, then you can use syscall(__NR_hello, argv…) in user space program

```
#define SYS_uselib __NR_uselib
#define SYS_ustat __NR_ustat
#define SYS_utime __NR_utime
#define SYS_utimensat __NR_utimensat
#define SYS_utimes __NR_utimes
#define SYS_vfork __NR_vfork
#define SYS_vhangup __NR_vhangup
#define SYS_vm86 __NR_vm86
#define SYS_vm86old __NR_vm86old
#define SYS_vmsplice __NR_vmsplice
#define SYS_vserver __NR_vserver
#define SYS_wait4 __NR_wait4
#define SYS_waitid __NR_waitid
#define SYS_waitpid __NR_waitpid
#define SYS_write __NR_write
#define SYS_writev __NR_writev
#define SYS_hello __NR_hello
                                                    342,1         Bot
```

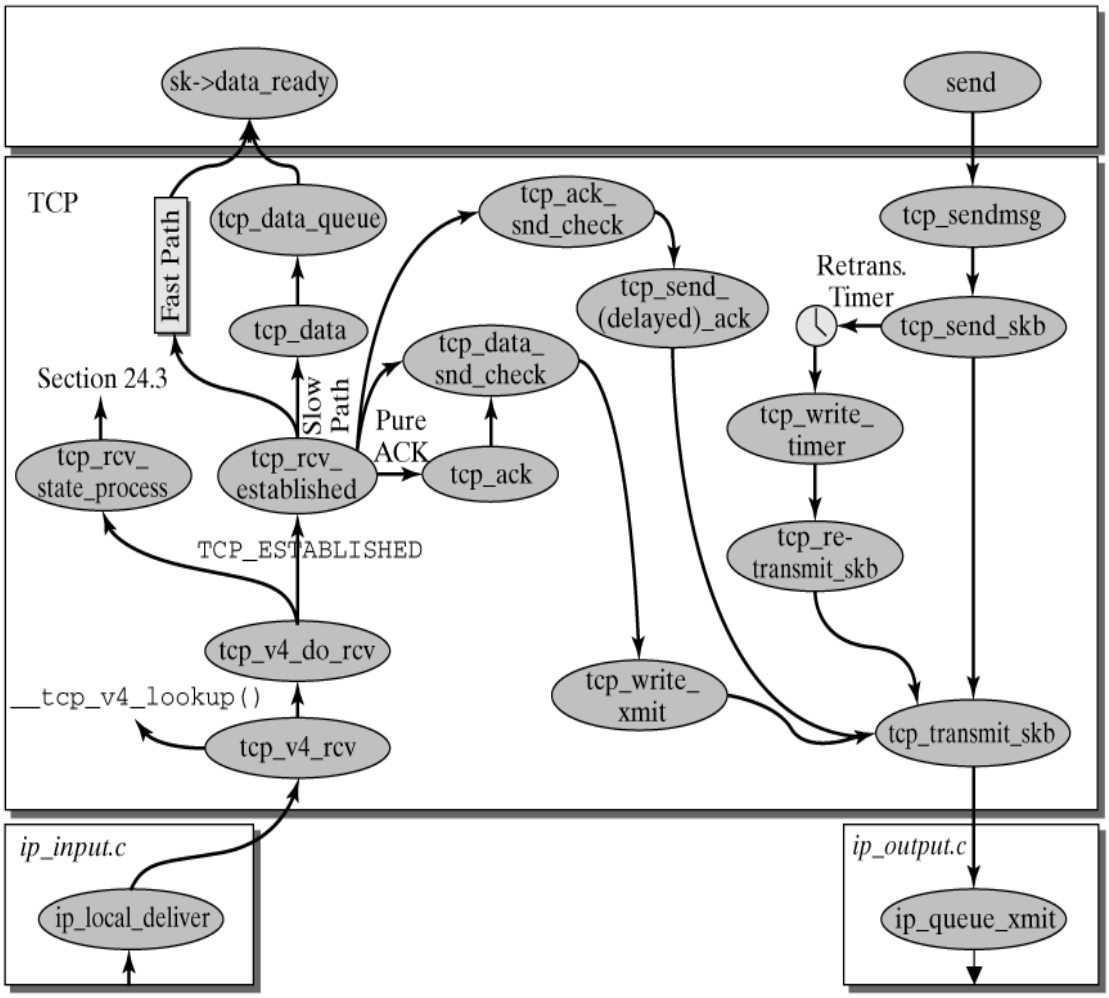
8.After all, recompile kernel by make ; make modules_install ; make install and reboot.

Lab 6.3 Encrypt and Decrypt data packet in TCP layer

You are requested to implement a system call **mysetsockopt**, which does the same thing as “setsockopt” and stores the encrypt and decrypt key into the socket structure.

- **To-Do**
  1. First of all, please choose a place to encrypt/decrypt the outgoing/incoming data packet in the TCP layer. You can reference this “rough” flow in the TCP layer.

     Note: The place you encrypt/decrypt the data packet is not restricted. The only restriction is “in the TCP layer”.



  2. Your encrypt/decrypt algorithm should be invoked only after you use the “mysetsockopt” system call to set the mykey field. If you encrypt/decrypt any data packet before that (i.e. the 3-way hand shake message), the kernel won’t be able to recognize the transmitted message and this will cause a kernel panic

3. The encrypt algorithm is “shift each byte of the payload of the data packet to the right by mykey locations in the ASCII code table”. For example, if the data is “abcABC123” & mykey=3, then the encrypted data packet will look like “defDEF456”.

4. The Decrypt algorithm is “shift each byte of the payload of the data packet to the left by mykey locations in the ASCII code table”. For example, if the data is “abcABC123” & mykey=3, then the encrypted data packet will look like “^-‘>?@./0”.

- Step
  1. Please create a new field “mykey”, which stores the encrypt and decrypt key, in the socket structure. (it’s free to add this key in any place of the socket structure). The type of mykey is integer.
  2. Implement a system call named “mysetsockopt” to set your description/encryption key. The format of the system call “mysetsockopt” is not restricted. But it should at least has one parameter “mykey”. (i.e. mysetsockopt(…, mykey, …);) Note: The system call setsockopt is defined in net/socket.c
  3. Implement your descrypt & encrypt algorithm.
  4. Build your kernel & install
  5. Modify the user program “client.c” and “server.c”, use system call to set your key, then start to send / receive message, your client should echo the message you typed.
  6. Use “dmesg” to show your flow of decryption and encryption.
  7. If your result not correct,you can try to use 2 virtual machine as host and client. (Remember to stop the firewall)

     <hint : TA writes the encrypt/decrypt algorithm in tcp_sendmsg/tcp_v4_rcv function>

Demo requirement

You need to implement a system call to control encryption. And you are required to encrypt/decrypt only the TCP package body.

Run the client and server program then show the results.(you can use 2 individual VMs)

client

```
[root@localhost Desktop]# ./client 3 192.168.221.128 8780
Enter(q for exit):
123abcABC
Received(10): 123abcABC
Enter(q for exit):
ABCaba123
Received(10): ABCaba123
Enter(q for exit):
q

[root@localhost Desktop]#
```

server

```
[root@localhost Desktop]# ./server 3 8780
Starting server:  Hit return to shutdown
No echo requests for 100 secs...Server still alive
Request on port 0:  Handling client 192.168.221.129(4)
456defDEF

DEFded456

Connection 4 Shudown.
```

dmesg (client side)

```
[  359.564476] Send side, the original message is : 123abcABC
[  359.564479]
[  359.564482] Send side, the encrypted message is : 456defDEF
[  359.565087] Receive side, the encrypted data is : 456defDEF
[  359.565091] Receive side, the decrypted data is : 123abcABC
[  359.565092]
[  371.917251] Send side, the original message is : ABCaba123
[  371.917254]
[  371.917257] Send side, the encrypted message is : DEFded456
[  371.917790] Receive side, the encrypted data is : DEFded456
[  371.917794] Receive side, the decrypted data is : ABCaba123
[  371.917796]
```

Tips for kernel development (not necessary):

1. First, you can write your encrypt/decrypt algorithm as a function in a module. (i.e. myPacket.ko ; void myPacket(…,…,…,……))

2. Define your function & call your function at the place you want to encrypt/decrypt. For example

```
sk = __inet_lookup_skb(&tcp_hashinfo,
if (!sk)
        goto no_tcp_socket;

if(myPacket){
        myPacket(skb,sk->mykey);
}
```

3. Extern define your function at the sock.c (sock.c is just an example)

```
extern int(*myPacket)(struct sk_buff*,int mykey);
EXPORT_SYMBOL(myPacket);
```

4. Compile your kernel at the first time and only one time. After reboot, you only need to compile/insmod/rmmod your module to test your algorithm.

5. If you modify your algorithm in the module, you won't need to recompile the kernel. This skill can save lots of time in compiling the kernel.

Reference

http://in1.csie.ncu.edu.tw/~hsufh/COURSES/SUMMER2013/Compile_kernel.pdf
http://in1.csie.ncu.edu.tw/~hsufh/COURSES/FALL2007/syscall.html
http://blog.chinaunix.net/uid-27007766-id-3233371.html
http://blog.sina.com.cn/s/blog_52355d840100b6sd.html
http://lxr.free-electrons.com/ident?i=tcp_sendmsg