



## گزارش آزمایش ۸: آشنایی با توابع سیستمی

ملیکا علیزاده ۴۰۱۱۰۶۲۵۵

الینا هژبری ۴۰۱۱۷۰۶۶۱

### ۳.۸ آزمایش ۱

در این بخش باید با نوشتن یک ماژول سطح هسته آدرس تمامی توابع سیستمی موجود در هسته‌ی سیستم عامل لینوکس را چاپ کنیم. ولی قبل از نوشتن ماژول، باید دستور زیر را اجرا کنیم تا هدر لینوکس با توجه به نسخه‌ی هسته‌ی فعلی آپدیت شود:

```
sudo apt install build-essential linux-headers-$(uname -r)
```

بعد باید دو فایل ماژول و Makefile آن را بنویسیم. در کد ماژول ابتدا ماژول را به صورت GPL تعریف می‌کنیم تا هسته بعضی محدودیت‌ها را برای ماژول در نظر بگیرد. بعد متغیرهای زیر را تعریف می‌کنیم:

fn kallsyms\_lookup\_name یک اشاره‌گر به تابع است تا آدرس واقعی kallsyms\_lookup\_name را ذخیره کند. ساختار kprobe برای آدرس نمادین kallsyms\_lookup\_name syscall\_table اشاره‌گر به جدول syscall‌ها و num\_syscalls تعداد syscall‌ها را نگه می‌دارد.

```
۱ #include <linux/init.h>
۲ #include <linux/module.h>
۳ #include <linux/kernel.h>
۴ #include <linux/kprobes.h>
۵ #include <linux/kallsyms.h>
۶ #include <linux/unistd.h>
۷
۸ MODULE_LICENSE("GPL");
۹
۱۰ static unsigned long (*kallsyms_lookup_name_fn)(const char *name);
۱۱ static struct kprobe kp = { .symbol_name = "kallsyms_lookup_name" };
۱۲ static unsigned long *syscall_table;
۱۳ static size_t num_syscalls;
۱۴
```

در تابع زیر تعداد syscall‌ها مشخص می‌شود. به این صورت که اگر NR\_syscalls یا NR\_syscalls\_\_ باشد آن را در ادامه استفاده می‌کنیم تا در هنگام چاپ از تعداد واقعی توابع سیستمی بیشتر جلو نرویم.

---

```

1 static size_t detect_syscall_count(void)
2 {
3     #if defined(NR_syscalls)
4         return NR_syscalls;
5     #elif defined(__NR_syscalls)
6         return __NR_syscalls;
7     #else
8         return 512;
9     #endif
10 }
11

```

---

در ادامه دو تابع اصلی ماژول را داریم. تابع `init` با لود شدن ماژول اجرا می‌شود و ابتدا تعداد کل `syscall` ها را پیدا می‌کند. حال با استفاده از `kprobe` آدرس واقعی `kallsyms_lookup_name` را در اشاره‌گر آن ذخیره می‌کنیم و بعد `kprobe` را حذف می‌کنیم. حال با استفاده از تابع `kallsyms_lookup_name`، آدرس جدول `syscall` های هسته را می‌گیریم. بعد در یک حلقه روی کل جدول `syscall` ها آدرس و اسم توابع را چاپ می‌کنیم. در تابع `exit` هم وقتی ماژول از هسته خارج می‌شود یک پیغام چاپ می‌کنیم.

---

```

1 static int __init syslist_init(void)
2 {
3     int i;
4     num_syscalls = detect_syscall_count();
5
6     register_kprobe(&kp);
7     kallsyms_lookup_name_fn = (void *)kp.addr;
8     unregister_kprobe(&kp);
9
10    syscall_table = (unsigned long *)kallsyms_lookup_name_fn("sys_call_table");
11
12    for (i = 0; i < num_syscalls; i++)
13        if (syscall_table[i])
14            printk("syscalls_lister: syscall[%d] = %px (%pS)\n",
15                  i, (void *)syscall_table[i], (void *)syscall_table[i]);
16
17    return 0;
18 }
19
20 static void __exit syslist_exit(void)
21 {
22     printk("syscalls_lister: module unloaded.\n");
23 }
24
25 module_init(syslist_init);
26 module_exit(syslist_exit);
27
28

```

---

فایل Makefile هم به صورت زیر می‌نویسیم.

```

۱ obj-m += syscalls_lister.o
۲
۳ KDIR := /lib/modules/$(shell uname -r)/build
۴ PWD := $(shell pwd)
۵
۶ all:
۷     $(MAKE) -C $(KDIR) M=$(PWD) modules
۸
۹ clean:
۱۰    $(MAKE) -C $(KDIR) M=$(PWD) clean
۱۱
۱۲
۱۳

```

اگر از فایل `unistd.h` استفاده کنیم فقط اسم توابع و شماره‌ی آنها را بدست می‌آوریم و چون به آدرس توابع هم نیاز داریم پس از جدول `syscall`ها استفاده کردیم. حال با استفاده از دستورات زیر ماژول را لود و اجرا می‌کنیم.  
آماده کردن فایل اجرایی یا `.ko` ماژول:

`make`

لودکردن ماژول در هسته:

`sudo insmod syscalls_lister.ko`

مشاهده‌ی خروجی:

`sudo dmesg`

و برای آنلود ماژول و پاک کردن فایل اجرایی آن:

`sudo rmmod syscalls_lister`

`make clean`

```

401106255-401170661@ubuntu24:~/oslab8/8.3$ nano syscalls_lister.c
401106255-401170661@ubuntu24:~/oslab8/8.3$ nano Makefile
401106255-401170661@ubuntu24:~/oslab8/8.3$ make
make -C /lib/modules/6.14.0-27-generic/build M=/home/401106255-401170661/oslab8/8.3 modules
make[1]: Entering directory '/usr/src/linux-headers-6.14.0-27-generic'
make[2]: Entering directory '/home/401106255-401170661/oslab8/8.3'
warning: the compiler differs from the one used to build the kernel
The kernel was built by: x86_64-linux-gnu-gcc-13 (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
You are using:          gcc-13 (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
CC [M]  syscalls_lister.o
MODPOST Module.symvers
CC [M]  syscalls_lister.mod.o
CC [M]  .module-common.o
LD [M]  syscalls_lister.ko
BTF [M] syscalls_lister.ko
Skipping BTF generation for syscalls_lister.ko due to unavailability of vmlinux
make[2]: Leaving directory '/home/401106255-401170661/oslab8/8.3'
make[1]: Leaving directory '/usr/src/linux-headers-6.14.0-27-generic'
401106255-401170661@ubuntu24:~/oslab8/8.3$ sudo insmod syscalls_lister.ko

```

شکل ۱: لود ماژول

```

401106255-401170661@ubuntu24:~/oslab8/8.3$ sudo dmesg
[ 6921.432671] syscalls_list: syscall[0] = ffffffff8e1c0340 (__x64_sys_read+0x0/0x30)
[ 6921.432698] syscalls_list: syscall[1] = ffffffff8e1c04c0 (__x64_sys_write+0x0/0x30)
[ 6921.432707] syscalls_list: syscall[2] = ffffffff8e1bb460 (__x64_sys_open+0x0/0xa0)
[ 6921.432719] syscalls_list: syscall[3] = ffffffff8e1b7600 (__x64_sys_close+0x0/0x90)
[ 6921.432727] syscalls_list: syscall[4] = ffffffff8e1c87f0 (__x64_sys_newstat+0x0/0x20)
[ 6921.432737] syscalls_list: syscall[5] = ffffffff8e1c84f0 (__x64_sys_newfstat+0x0/0x20)
[ 6921.432745] syscalls_list: syscall[6] = ffffffff8e1c8b20 (__x64_sys_newlstat+0x0/0x20)
[ 6921.432754] syscalls_list: syscall[7] = ffffffff8e1e20c0 (__x64_sys_poll+0x0/0x150)
[ 6921.432761] syscalls_list: syscall[8] = ffffffff8e1bc370 (__x64_sys_lseek+0x0/0x30)
[ 6921.432771] syscalls_list: syscall[9] = ffffffff8dce1710 (__x64_sys_mmap+0x0/0x70)
[ 6921.432780] syscalls_list: syscall[10] = ffffffff8e0fe7d0 (__x64_sys_mprotect+0x0/0x30)
[ 6921.432790] syscalls_list: syscall[11] = ffffffff8e0f8270 (__x64_sys_munmap+0x0/0x30)
[ 6921.432800] syscalls_list: syscall[12] = ffffffff8e0f90e0 (__x64_sys_brk+0x0/0x20)
[ 6921.432809] syscalls_list: syscall[13] = ffffffff8dda5420 (__x64_sys_rt_sigaction+0x0/0x120)
[ 6921.432821] syscalls_list: syscall[14] = ffffffff8dd9d040 (__x64_sys_rt_sigprocmask+0x0/0xe0)
[ 6921.432866] syscalls_list: syscall[15] = ffffffff8dcd6260 (__do_sys_rt_sigreturn+0x0/0x100)
[ 6921.432879] syscalls_list: syscall[16] = ffffffff8e1de450 (__x64_sys_ioctl+0x0/0xe0)
[ 6921.432890] syscalls_list: syscall[17] = ffffffff8e1bf4f0 (__x64_sys_pread64+0x0/0xd0)
[ 6921.433031] syscalls_list: syscall[18] = ffffffff8e1c0090 (__x64_sys_pwrite64+0x0/0xd0)
[ 6921.433042] syscalls_list: syscall[19] = ffffffff8e1bd560 (__x64_sys_readv+0x0/0x30)
[ 6921.433048] syscalls_list: syscall[20] = ffffffff8e1be040 (__x64_sys_writev+0x0/0x30)
[ 6921.433054] syscalls_list: syscall[21] = ffffffff8e1b7b10 (__x64_sys_access+0x0/0x30)
[ 6921.433061] syscalls_list: syscall[22] = ffffffff8e1cf1f0 (__x64_sys_pipe+0x0/0x20)
[ 6921.433070] syscalls_list: syscall[23] = ffffffff8e1e29f0 (__x64_sys_select+0x0/0x40)
[ 6921.433076] syscalls_list: syscall[24] = ffffffff8de05fd0 (__do_sys_sched_yield+0x0/0x20)
[ 6921.433087] syscalls_list: syscall[25] = ffffffff8e1010f0 (__x64_sys_mremap+0x0/0x40)
[ 6921.433096] syscalls_list: syscall[26] = ffffffff8e1014c0 (__x64_sys_msync+0x0/0x30)
0x0/0x120)
[ 6921.437426] syscalls_list: syscall[447] = ffffffff8e1ac280 (__x64_sys_memfd_secret+0x0/0xc0)
[ 6921.437433] syscalls_list: syscall[448] = ffffffff8e088ad0 (__x64_sys_process_mrelease+0x0/0x
20)
[ 6921.437445] syscalls_list: syscall[449] = ffffffff8decc210 (__x64_sys_futex_waitv+0x0/0x40)
[ 6921.437453] syscalls_list: syscall[450] = ffffffff8e165bf0 (__x64_sys_set_mempolicy_home_node
+0x0/0x30)
[ 6921.437465] syscalls_list: syscall[451] = ffffffff8e07ddd0 (__x64_sys_cachestat+0x0/0x200)
[ 6921.437476] syscalls_list: syscall[452] = ffffffff8e1b9d00 (__x64_sys_fchmodat2+0x0/0x30)
[ 6921.437485] syscalls_list: syscall[453] = ffffffff8dd4da90 (__x64_sys_map_shadow_stack+0x0/0x
d0)
[ 6921.437495] syscalls_list: syscall[454] = ffffffff8decbe0 (__x64_sys_futex_wake+0x0/0xb0)
[ 6921.437502] syscalls_list: syscall[455] = ffffffff8decbe2a0 (__x64_sys_futex_wait+0x0/0x140)
[ 6921.437509] syscalls_list: syscall[456] = ffffffff8decc3b0 (__x64_sys_futex_requeue+0x0/0x30)
[ 6921.437516] syscalls_list: syscall[457] = ffffffff8e1f66a0 (__x64_sys_statmount+0x0/0x30)
[ 6921.437527] syscalls_list: syscall[458] = ffffffff8e1f6960 (__x64_sys_listmount+0x0/0x30)
[ 6921.437537] syscalls_list: syscall[459] = ffffffff8e3c40d0 (__x64_sys_lsm_get_self_attr+0x0/0
x30)
[ 6921.437546] syscalls_list: syscall[460] = ffffffff8e3c4050 (__x64_sys_lsm_set_self_attr+0x0/0
x30)
[ 6921.437553] syscalls_list: syscall[461] = ffffffff8e3c4250 (__x64_sys_lsm_list_modules+0x0/0x
f0)
[ 6921.437560] syscalls_list: syscall[462] = ffffffff8e11b170 (__x64_sys_mseal+0x0/0x30)
[ 6921.437569] syscalls_list: syscall[463] = ffffffff8e201c50 (__x64_sys_setxattrat+0x0/0x120)
[ 6921.437577] syscalls_list: syscall[464] = ffffffff8e2023a0 (__x64_sys_getxattrat+0x0/0x120)
[ 6921.437585] syscalls_list: syscall[465] = ffffffff8e1ffe50 (__x64_sys_listxattrat+0x0/0x40)
[ 6921.437597] syscalls_list: syscall[466] = ffffffff8e200b80 (__x64_sys_removexattrat+0x0/0x30)
401106255-401170661@ubuntu24:~/oslab8/8.3$

```

شکل ۲: نمایش خروجی کد

```
401106255-401170661@ubuntu24:~/oslab8/8.3$ sudo rmmod syscalls_lister.ko
401106255-401170661@ubuntu24:~/oslab8/8.3$ make clean
make -C /lib/modules/6.14.0-27-generic/build M=/home/401106255-401170661/oslab8/8.3 clean
make[1]: Entering directory '/usr/src/linux-headers-6.14.0-27-generic'
make[2]: Entering directory '/home/401106255-401170661/oslab8/8.3'
CLEAN Module.symvers
make[2]: Leaving directory '/home/401106255-401170661/oslab8/8.3'
make[1]: Leaving directory '/usr/src/linux-headers-6.14.0-27-generic'
401106255-401170661@ubuntu24:~/oslab8/8.3$ sudo dmesg -C
401106255-401170661@ubuntu24:~/oslab8/8.3$
```

شکل ۳: آنلود ماژول

## ۴.۸ آزمایش ۲

در این آزمایش با استفاده از LD\_PRELOAD یک کتابخانه اشتراکی ایجاد کرده که رفتار دستور ls را تغییر دهد. هدف از این کد این است که وقتی ls یک پوشه خالی را بررسی می‌کند، به جای اینکه چیزی نمایش دهد، یک سری فایل تقلبی نمایش دهد. ابتدا با استفاده از دستور strace ls توابع سیستمی‌ای که توسط ls استفاده می‌شود را بررسی می‌کنیم. به طور کلی دستور ls ۲ کار را انجام می‌دهد.

- بررسی نام فایل‌های موجود
- بررسی اطلاعات فایل مثل سایز و زمان تغییر و ...

در ابتدا توابع سیستمی readdir و opendir و access را تغییر دادیم ولی این توابع در هنگام چک کردن اطلاعات فایل دچار ارور می‌شدند و مشخص می‌شد که فایل وجود ندارد پس در ادامه توابع دیگری مثل getdents64 و \_\_xstat را تغییر دادیم. در کد زیر ابتدا یک سری هدر فایل برای دسترسی به توابع کتابخانه‌ای و ساختارهای داده سیستم را فراخوانی می‌کنیم. در ادامه یک ساختار تعریف کردیم تا اطلاعات هر پوشه یا فایل در آن ذخیره شود. این اطلاعات شامل شماره‌ی اینود فایل، آفست، طول فایل، نوع و نام فایل می‌شود. اسامی فایل‌های تقلبی هم تعریف شده‌است.

```
۱ #define _GNU_SOURCE
۲
۳ #include <dlfcn.h>
۴ #include <stdio.h>
۵ #include <string.h>
۶ #include <errno.h>
۷ #include <sys/stat.h>
۸ #include <stdint.h>
۹ #include <sys/syscall.h>
۱۰ #include <dirent.h>
۱۱ #include <unistd.h>
۱۲ #include <fcntl.h>
۱۳
۱۴ struct linux_dirent64 {
۱۵     uint64_t d_ino;
۱۶     int64_t d_off;
۱۷     uint16_t d_reclen;
۱۸     uint8_t d_type;
۱۹     char d_name[];
۲۰ };
۲۱
۲۲ static const char* fake_file_name = "faketext.txt";
۲۳ static const char* fake_dir_name = "fakefile";
۲۴
۲۵ static ssize_t (*real_getdents64)(int, void*, size_t) = NULL;
۲۶ static int (*real___xstat)(int, const char*, struct stat*) = NULL;
۲۷ static int (*real_lstat)(const char*, struct stat*) = NULL;
۲۸
۲۹ static int fake_file_returned = 0;
۳۰ static int fake_dir_returned = 0;
۳۱
```

در تابع زیر یک مسیر را به عنوان ورودی گرفته و مسیر فایل تقلبی را ایجاد می‌کند. و در صورت تقلبی بودن مسیر مقدار نوع فایل را مشخص می‌کند و ۱ را برمی‌گرداند.

---

```

1  int is_fake_path(const char* path, int* type) {
2      size_t path_len = strlen(path);
3      size_t file_len = strlen(fake_file_name);
4      size_t dir_len = strlen(fake_dir_name);
5
6      if (path_len >= file_len && strcmp(path + path_len - file_len, fake_file_name) == 0) {
7          *type = DT_REG;
8          return 1;
9      }
10
11     if (path_len >= dir_len && strcmp(path + path_len - dir_len, fake_dir_name) == 0) {
12         *type = DT_DIR;
13         return 1;
14     }
15
16     *type = 0;
17     return 0;
18 }
19

```

---

در بخش زیر باید تابع سیستمی `getents64` را تغییر دهیم. به طوری که فایل‌ها و پوشه‌های تقلبی را پنهان کنیم تا وقتی `ls` خواست اطلاعات این فایل‌ها را با توابع سیستمی دیگر بررسی کند دچار خطا نشود. ابتدا تابع اصلی `getdents64` را فراخوانی می‌کند تا محتویات واقعی پوشه را بخواند. سپس اگر هنوز ورودی‌های تقلبی را برگردانده باشد، آن‌ها را به بافری که توسط `real_getdents64` پر شده است، اضافه می‌کند. که باعث می‌شود این فایل‌ها در خروجی نمایش داده شوند.

---

```

1  ssize_t getdents64(int fd, void* dirp, size_t count) {
2      if (!real_getdents64) {
3          real_getdents64 = dlsym(RTLD_NEXT, "getdents64");
4      }
5
6      ssize_t bytes_read = real_getdents64(fd, dirp, count);
7      if (bytes_read == -1) {
8          return -1;
9      }
10
11     struct linux_dirent64* d;
12     char* bpos;
13
14     if (!fake_file_returned && !fake_dir_returned) {
15         d = (struct linux_dirent64*) dirp;
16         bpos = (char*) dirp;
17
18         if (count - bytes_read >= sizeof(struct linux_dirent64) + strlen(fake_dir_name) + 1) {
19             d->d_ino = 12345679;
20             d->d_off = bytes_read;
21             d->d_type = DT_DIR;
22             strcpy(d->d_name, fake_dir_name);
23             d->d_reclen = sizeof(struct linux_dirent64) + strlen(d->d_name) + 1;
24
25             bytes_read += d->d_reclen;
26             d = (struct linux_dirent64*)((char*)d + d->d_reclen);
27

```



```

۲۷         fake_dir_returned = 1;
۲۸     }
۲۹
۳۰     if (count - bytes_read >= sizeof(struct linux_dirent64) + strlen(fake_file_name) + 1) {
۳۱         d->d_ino = 12345678;
۳۲         d->d_off = bytes_read;
۳۳         d->d_type = DT_REG;
۳۴         strcpy(d->d_name, fake_file_name);
۳۵         d->d_reclen = sizeof(struct linux_dirent64) + strlen(d->d_name) + 1;
۳۶
۳۷         bytes_read += d->d_reclen;
۳۸         fake_file_returned = 1;
۳۹     }
۴۰ }
۴۱
۴۲     return bytes_read;
۴۳ }
۴۴

```

---

توابع زیر برای ارائه اطلاعات در مورد یک فایل یا پوشه استفاده می‌شوند. ابتدا طبق تابع `is_fake_path` جعلی بودن یا نبودن مسیر را مشخص می‌کند. اگر بود یک ساختار ساختگی ایجاد می‌کند و در غیر اینصورت تابع اصلی صدا زده و اطلاعات درست را می‌دهد.

```

۱  int __xstat(int ver, const char *path, struct stat *buf) {
۲      if (!real__xstat) {
۳          real__xstat = dlsym(RTLD_NEXT, "__xstat");
۴      }
۵
۶      int type;
۷      if (is_fake_path(path, &type)) {
۸          memset(buf, 0, sizeof(struct stat));
۹          buf->st_uid = geteuid();
۱۰         buf->st_gid = getegid();
۱۱         buf->st_ino = (type == DT_REG) ? 12345678 : 12345679;
۱۲
۱۳         if (type == DT_REG) {
۱۴             buf->st_mode = S_IFREG | 0755;
۱۵             buf->st_size = 100;
۱۶         } else {
۱۷             buf->st_mode = S_IFDIR | 0755;
۱۸             buf->st_size = 4096;
۱۹         }
۲۰         return 0;
۲۱     }
۲۲     return real__xstat(ver, path, buf);
۲۳ }
۲۴
۲۵ int lstat(const char *path, struct stat *buf) {
۲۶     if (!real_lstat) {
۲۷         real_lstat = dlsym(RTLD_NEXT, "lstat");
۲۸     }
۲۹

```



```

۳۰     int type;
۳۱     if (is_fake_path(path, &type)) {
۳۲         memset(buf, 0, sizeof(struct stat));
۳۳         buf->st_uid = geteuid();
۳۴         buf->st_gid = getegid();
۳۵         buf->st_ino = (type == DT_REG) ? 12345678 : 12345679;
۳۶
۳۷         if (type == DT_REG) {
۳۸             buf->st_mode = S_IFREG | 0755;
۳۹             buf->st_size = 100;
۴۰         } else {
۴۱             buf->st_mode = S_IFDIR | 0755;
۴۲             buf->st_size = 4096;
۴۳         }
۴۴         return 0;
۴۵     }
۴۶     return real_lstat(path, buf);
۴۷ }
۴۸

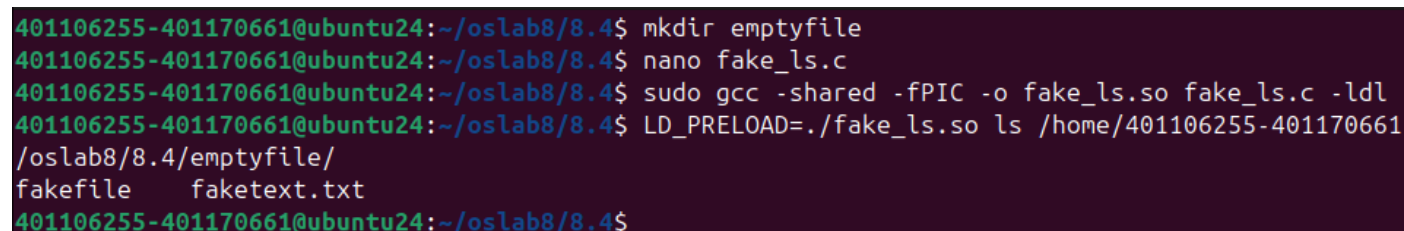
```

---

برای تست ابتدا یک فایل خالی ساخته و کد را کامپایل و به صورت یک فایل کتابخانه‌ی اشتراکی تبدیل می‌کنیم و سپس دستور `ls` را با استفاده از `LD_PRELOAD` اجرا می‌کنیم یا به عبارتی دستورات زیر را اجرا می‌کنیم:

```
gcc -shared -fPIC -o fake_ls.so fake_ls.c
```

```
LD_PRELOAD="./fake_ls.so" ls <path to emptydir>
```



```

401106255-401170661@ubuntu24:~/oslab8/8.4$ mkdir emptyfile
401106255-401170661@ubuntu24:~/oslab8/8.4$ nano fake_ls.c
401106255-401170661@ubuntu24:~/oslab8/8.4$ sudo gcc -shared -fPIC -o fake_ls.so fake_ls.c -ldl
401106255-401170661@ubuntu24:~/oslab8/8.4$ LD_PRELOAD=./fake_ls.so ls /home/401106255-401170661
/oslab8/8.4/emptyfile/
fakefile    faketext.txt
401106255-401170661@ubuntu24:~/oslab8/8.4$

```

شکل ۴: نمایش خروجی و دستورات استفاده شده