**Цель работы:** Изучить системные вызовы chmod, chown, lchown, chroot в версиях ядра Linux 2.6 и 4.8. Для данных системных вызовов изучить их исходный код, продемонстрировать их применение в коде программы, заменить вызов своим.

Используемые системы

Lubuntu 16.04

|  |
| --- |
| System: Ubuntu 16.10 x86\_64  Kernel: 4.8.0-22-generic DE: LXDE Session: Lubuntu  ----------------------------------  Processor: Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz  Memory (Gb): 0.45  Video: 00:0f.0 VGA compatible controller: VMware SVGA II Adapter Subsystem: VMware SVGA II Adapter Kernel driver in use: vmwgfx Kernel modules: vmwgfx  ---------------------------------- |

Ubuntu 10.04

|  |
| --- |
| System: Ubuntu 10.04 LTS x86\_64  Kernel: 2.6.32-21-generic DE: Session: gnome  ----------------------------------  Processor: Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz  Memory (Gb): 0.96  Video: 00:0f.0 VGA compatible controller: VMware SVGA II Adapter 00:10.0 SCSI storage controller: LSI Logic / Symbios Logic 53c1030 PCI-X Fusion-MPT Dual Ultra320 SCSI (rev 01) Kernel driver in use: mptspi  ---------------------------------- |

1. Системные вызовы chmod, chown, lchown, chroot

**Chmod**

|  |
| --- |
| #include <sys/types.h>  #include <sys/stat.h>  int chmod(const char \*path, mode\_t mode); |

Изменяет права доступа к файлу, заданному параметром path.[1]

Права задаются применением логической операции OR (битовое сложение -- прим. пер.) к следующим константам:

S\_ISUID 04000 установить при выполнении идентификатор пользователя (suid бит -- прим.пер.)

S\_ISGID 02000 установить при выполнении идентификатор группы (sgid бит -- прим.пер.)

S\_ISVTX 01000 sticky бит

S\_IRUSR (S\_IREAD) 00400 владелец может читать

S\_IWUSR (S\_IWRITE) 00200 владелец может писать

S\_IXUSR (S\_IEXEC) 00100 владелец может выполнять файл или искать в каталоге

S\_IRGRP 00040 группа-владелец может читать

S\_IWGRP 00020 группа-владелец может писать

S\_IXGRP 00010 группа-владелец может выполнять файл или искать в каталоге

S\_IROTH 00004 все остальные могут читать

S\_IWOTH 00002 все остальные могут писать

S\_IXOTH 00001 все остальные могут выполнять файл или искать в каталоге

Эффективный идентификатор пользователя (UID) для вызывающего процесса должен быть нулем или совпадать с UID владельца файла.

Если эффективный UID процесса не равен нулю, а группа-владелец файла не совпадает с фактическим GID процесса или одним из его дополнительных GID'ов, то бит S\_ISGID будет сброшен, но ошибки при этом не возникнет.

**Chown, lchown**

|  |
| --- |
| #include <unistd.h>  int chown(const char \*path, uid\_t owner, gid\_t group);  int lchown(const char \*path, uid\_t owner, gid\_t group); |

Аргумент path является указателем на маршрутное имя файла. Идентификаторы владельца и группы указанного файла устанавливаются равными числовым значениям, соответственно, аргументов owner и group.

Изменить хозяев файла (владельца и группу) может только процесс, имеющий действующий идентификатор пользователя, равный идентификатору владельца файла или суперпользователя.

В версиях Linux до 2.1.81 (кроме 2.1.46), chown не следовал по символьным ссылкам. Начиная с версии Linux 2.1.81 chown следует по символьным ссылкам, и существует новый системный вызов lchown, который не следует по символьным ссылкам. Начиная с Linux 2.1.86 этот новый вызов (имеющий ту же смысл, что и старый chown), имеет тот же самый номер системного вызова, а chown получил новый номер.

**Chroot**

|  |
| --- |
| #include <unistd.h>  int chroot(const char \*path); |

Изменяет корневой каталог вызывающего процесса. Корневой каталог наследуется всеми дочерними процессами вызывающего процесса.

При успешном завершении возвращается ноль. При ошибке возвращается -1, a значение errno устанавливается соответствующим образом.

1. Реализация системных вызовов в различных версиях ядра

**Chmod**

Версия ядра 2.6.32-21.

Файл /fs/open.c[2]

|  |
| --- |
| //line 662  //определение chmod  SYSCALL\_DEFINE2(chmod, const char \_\_user \*, filename, mode\_t, mode)  {  return sys\_fchmodat(AT\_FDCWD, filename, mode);  } |

Переопределение chmod системную функцию \_\_NR\_fchmodat

Файл /include/asm-generic/unistd.h

|  |
| --- |
| //системный вызов №53  #define \_\_NR\_fchmodat 53  \_\_SYSCALL(\_\_NR\_fchmodat, sys\_fchmodat) |

Функция имеет номер системного вызова 53. Это значит, что системные вызовы вызываются по номеру, а не по адресу, как при обычном вызове функции внутри собственной программы или функции в библиотеке, связанной с программой. Это используется для перехода от пользователя к пространству ядра, принимая вместе с ним и ваши параметры.

Для каждого процессора существует свое переопределение функции \_\_NR\_fchmodat. Для x86 оно определется в файле /arch/x86/include/asm/unistd\_64.h

Файл /arch/x86/include/asm/unistd\_64.h

|  |
| --- |
| //Для x86 процессора системный вызов имеет номер 268  #define \_\_NR\_fchmodat 268  \_\_SYSCALL(\_\_NR\_fchmodat, sys\_fchmodat) |

Версия ядра 4.8.0\_22

Файл /fs/open.c

|  |
| --- |
| //line 562  SYSCALL\_DEFINE2(chmod, const char \_\_user \*, filename, umode\_t, mode)  {  return sys\_fchmodat(AT\_FDCWD, filename, mode);  } |

Файл /arch/arm64/include/asm/unistd32.h

|  |
| --- |
| //line 690  #define \_\_NR\_fchmodat 333  \_\_SYSCALL(\_\_NR\_fchmodat, sys\_fchmodat) |

Как видно, для новой версии ядра изменился номер системного вызова. И тип mode\_t заменен на umode\_t

**Chown**

Версия ядра 2.6.32-21.

Файл fs/open

|  |
| --- |
| //line 692  SYSCALL\_DEFINE3(chown, const char \_\_user \*, filename, uid\_t, user, gid\_t, group)  {  //struct path {  // struct vfsmount \*mnt;  // struct dentry \*dentry;  //};  struct path path;  int error;  //записывает в path путь к файлу  error = user\_path(filename, &path);  //если ошибка то выход  if (error)  goto out;  //Говорит системе что далее последует запись в нее  //Проверяет на возможность записи  //После записи должен быть вызван mnt\_drop\_write  //Файл /fs/namespace.c строка 255  error = mnt\_want\_write(path.mnt);  if (error)  goto out\_release;  error = chown\_common(path.dentry, user, group);  mnt\_drop\_write(path.mnt);  out\_release:  path\_put(&path);  out:  return error;  }  //line 667  static int chown\_common(struct dentry \* dentry, uid\_t user, gid\_t group)  {  struct inode \*inode = dentry->d\_inode;  int error;    //Это структура атрибутов Inode, используемая для notify\_change ().  //Она использует флаги, чтобы знать, какие значения были изменены.  struct iattr newattrs;  //Далее производится установка флагов  newattrs.ia\_valid = ATTR\_CTIME;  //Проверка пользователя  if (user != (uid\_t) -1) {  newattrs.ia\_valid |= ATTR\_UID;  newattrs.ia\_uid = user;  }  //Проверка группы  if (group != (gid\_t) -1) {  newattrs.ia\_valid |= ATTR\_GID;  newattrs.ia\_gid = group;  }  //Если файл - папка, то добавить флаги  if (!S\_ISDIR(inode->i\_mode))  newattrs.ia\_valid |=  ATTR\_KILL\_SUID | ATTR\_KILL\_SGID | ATTR\_KILL\_PRIV;  //Блокировка доступа другим процесам  mutex\_lock(&inode->i\_mutex);  //Внесение изменений в систему  error = notify\_change(dentry, &newattrs);  mutex\_unlock(&inode->i\_mutex);  return error;  } |

Версия ядра 4.8.0\_22

Файл /fs/open.c

|  |
| --- |
| SYSCALL\_DEFINE3(chown, const char \_\_user \*, filename, uid\_t, user, gid\_t, group)  {  return sys\_fchownat(AT\_FDCWD, filename, user, group, 0);  } |

Файл /arch/arm64/include/asm/unistd32.h

|  |
| --- |
| #define \_\_NR\_fchownat 325  \_\_SYSCALL(\_\_NR\_fchownat, sys\_fchownat) |

Как видно, вызов chown в новых версиях ядра заменен на системный вызов по номеру функции. Это значит что chown реализован в виде ассемблерной вставки, а не как код на языке С в более ранних версиях.

**Lchown**

Версия ядра 2.6.32-21.

Отличие от chown только в функции user\_lpath вместо user\_path.

user\_lpath не проследует по символьным ссылкам.

Файл fs/open.c

|  |
| --- |
| SYSCALL\_DEFINE3(lchown, const char \_\_user \*, filename, uid\_t, user, gid\_t, group)  {  struct path path;  int error;  **error = user\_lpath(filename, &path);**  if (error)  goto out;  error = mnt\_want\_write(path.mnt);  if (error)  goto out\_release;  error = chown\_common(path.dentry, user, group);  mnt\_drop\_write(path.mnt);  out\_release:  path\_put(&path);  out:  return error;  } |

Версия ядра 4.8.0\_22

/fs/open.c

|  |
| --- |
| SYSCALL\_DEFINE3(lchown, const char \_\_user \*, filename, uid\_t, user, gid\_t, group)  {  return sys\_fchownat(AT\_FDCWD, filename, user, group,  AT\_SYMLINK\_NOFOLLOW);  } |

/arch/arm64/include/asm/unistd32.h

|  |
| --- |
| #define \_\_NR\_lchown 16  \_\_SYSCALL(\_\_NR\_lchown, sys\_lchown16) |

Принцип работы – ассемблерный системный вызов.

**Chroot**

Версия ядра 2.6.32-21.

Файл fs/open.c

|  |
| --- |
| //line 574  SYSCALL\_DEFINE1(chroot, const char \_\_user \*, filename)  {  struct path path;  int error;  //Получение пути к дирректории  error = user\_path\_dir(filename, &path);  //Если ошибка то выходим  if (error)  goto out;  Проверить права доступа к данному inode  error = inode\_permission(path.dentry->d\_inode, MAY\_EXEC | MAY\_ACCESS);  //Если ошибка то выходим  if (error)  goto dput\_and\_out;  error = -EPERM;  //Определите, является ли текущая задача выполнимой  if (!capable(CAP\_SYS\_CHROOT))  goto dput\_and\_out;  //Заменяет fs->{rootmnt,root} на {mnt,dentry}(новые).  set\_fs\_root(current->fs, &path);  error = 0;  dput\_and\_out:  path\_put(&path);  out:  return error;  } |

Версия ядра 4.8.0\_22

/fs/open.c

|  |
| --- |
| SYSCALL\_DEFINE1(chroot, const char \_\_user \*, filename)  {  struct path path;  int error;  unsigned int lookup\_flags = LOOKUP\_FOLLOW | LOOKUP\_DIRECTORY;  retry:  //получить путь к новой папке  error = user\_path\_at(AT\_FDCWD, filename, lookup\_flags, &path);  //Если ошибка выйти  if (error)  goto out;  //Проверить права доступа к данному inode  error = inode\_permission(path.dentry->d\_inode, MAY\_EXEC | MAY\_CHDIR);  if (error)  goto dput\_and\_out;  error = -EPERM;  //Определите, является ли текущая задача выполнимой  //в текущем пространстве имен пользователя  if (!ns\_capable(current\_user\_ns(), CAP\_SYS\_CHROOT))  goto dput\_and\_out;  //Проверка безопасной замены дирректории  error = security\_path\_chroot(&path);  if (error)  goto dput\_and\_out;  //Заменяет fs->{rootmnt,root} на {mnt,dentry}(новые).  set\_fs\_root(current->fs, &path);  error = 0;  dput\_and\_out:  path\_put(&path);  if (retry\_estale(error, lookup\_flags)) {  lookup\_flags |= LOOKUP\_REVAL;  goto retry;  }  out:  return error;  } |

Таким образом, в новой версии ядра системный вызов изменился не сильно. Функция user\_path\_dir заменена на user\_path\_at. Добавлена дополнительная проверка security\_path\_chroot.

1. Использование системных вызовов из пользовательского кода

**Chmod**

Код программы

|  |
| --- |
| #include <sys/types.h>  #include <sys/stat.h>  #include <errno.h>  #include <string.h>  #include <stdio.h>  int main(void)  {  int mod=0;  mod = S\_IXOTH;  printf("PRECHANGE\n");  if (chmod ("/home/user/Documents/OS/SysCall/testFile",mod) == 0)  printf("chmod seccesful\n");  else  {  printf("Error chmod: %s\n", strerror(errno));  }  return 0;  } |

Выполнение кода

|  |
| --- |
| user@user-virtual-machine:~/Documents/OS/SysCall$ ls -l testFile  -rw-rw-r-- 1 user user 0 май 24 21:30 testFile  user@user-virtual-machine:~/Documents/OS/SysCall$ ./chmod  PRECHANGE  chmod seccesful  user@user-virtual-machine:~/Documents/OS/SysCall$ ls -l testFile  ---------x 1 user user 0 май 24 21:30 testFile |

Версия ядра 4.8

Трассировка программы

|  |
| --- |
| user@user-virtual-machine:~/Documents/OS/SysCall$ strace ./chmod  …  **chmod("/home/user/Documents/OS/SysCall/testFile", 01) = 0**  fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(136, 1), ...}) = 0  brk(NULL) = 0x559971c3c000  brk(0x559971c5d000) = 0x559971c5d000  write(1, "chmod seccesful\n", 16chmod seccesful  ) = 16  exit\_group(0) = ?  +++ exited with 0 +++ |

Виден системный вызов chmod, его входные параметры и возвращаемое значение.

Создадим файл testFile от пользователя root, чтобы вызвать ошибку при выполнении программы от обычного пользователя.

Трассировка программы

|  |
| --- |
| user@user-virtual-machine:~/Documents/OS/SysCall$ strace ./chmod  …  write(1, "PRECHANGE\n", 10PRECHANGE  ) = 10  chmod("/home/user/Documents/OS/SysCall/testFile", 01) = -1 EPERM (Operation not permitted)  write(1, "Error chmod: Operation not permi"..., 37Error chmod: Operation not permitted  ) = 37  exit\_group(0) = ?  +++ exited with 0 +++ |

Из вывода видно, значение ошибки (-1) и значение errno, указывающего на ошибку прав доступа.

Ядро 2.6

Трассировка программы

|  |
| --- |
| write(1, "PRECHANGE\n", 10PRECHANGE  ) = 10  **chmod("/home/stas/Documents/OS/SysCall/testFile", 01) = 0**  write(1, "chmod seccesful\n", 16chmod seccesful  ) = 16  exit\_group(0) = ?  +++ exited with 0 +++ |

Как видно из вывода, системный вызов внешне ничем не отличается от вызова в новой версии ядра.

**Chown**

Код программы

|  |
| --- |
| #include <unistd.h>  #include <errno.h>  #include <string.h>  #include <stdio.h>  int main(void)  {  printf("PRECHOWN\n");  if (chown ("/home/user/Documents/OS/SysCall/testFile",0,0) == 0)  printf("chown seccesful\n");  else  {  printf("Error chown: %s\n", strerror(errno));  }  return 0;  } |

Версия ядра 4.8

Трассировка программы

|  |
| --- |
| brk(NULL) = 0x563c4b649000  brk(0x563c4b66a000) = 0x563c4b66a000  write(1, "PRECHOWN\n", 9PRECHOWN) = 9  **chown("/home/user/Documents/OS/SysCall/testFile", 0, 0) = -1 EPERM (Operation not permitted)**  write(1, "Error chown: Operation not permi"..., 37Error chown: Operation not permitted  ) = 37  exit\_group(0) = ?  +++ exited with 0 +++ |

Трассировка программы от root

|  |
| --- |
| brk(0x55964e2e4000) = 0x55964e2e4000  write(1, "PRECHOWN\n", 9PRECHOWN  ) = 9  **chown("/home/user/Documents/OS/SysCall/testFile", 0, 0) = 0**  write(1, "chown seccesful\n", 16chown seccesful  ) = 16  exit\_group(0) = ?  +++ exited with 0 +++ |

В трассировке видим системный вызов chown

Тест программы на символьной ссылке

|  |
| --- |
| user@user-virtual-machine:~/Documents/OS/SysCall$ ls -l  total 32  …  -rw-rw-r-- 1 user user 0 май 24 22:12 linkedFile  lrwxrwxrwx 1 user user 10 май 24 22:13 testFile -> linkedFile  user@user-virtual-machine:~/Documents/OS/SysCall$ sudo ./chown  PRECHOWN  chown seccesful  user@user-virtual-machine:~/Documents/OS/SysCall$ ls -l  total 32  …  -rw-rw-r-- 1 **root root** 0 май 24 22:12 linkedFile  lrwxrwxrwx 1 user user 10 май 24 22:13 testFile -> linkedFile |

Ядро 2.6.

Трассировка программы

|  |
| --- |
| write(1, "PRECHOWN\n", 9PRECHOWN  ) = 9  **chown("/home/stas/Documents/OS/SysCall/testFile", 0, 0) = 0**  write(1, "chown seccesful\n", 16chown seccesful  ) = 16  exit\_group(0) = ?  +++ exited with 0 +++ |

Отличий в вызове системной функции нет.

**Lchown**

Версия ядра 4.8

Код программы

|  |
| --- |
| #include <unistd.h>  #include <errno.h>  #include <string.h>  #include <stdio.h>  int main(void)  {  printf("PRELCHOWN\n");  if (lchown ("/home/user/Documents/OS/SysCall/testFile",0,0) == 0)  printf("Lchown seccesful\n");  else  {  printf("Error Lchown: %s\n", strerror(errno));  }  return 0;  } |

|  |
| --- |
| user@user-virtual-machine:~/Documents/OS/SysCall$ ls -l  total 48  -rw-rw-r-- 1 user user 0 май 24 22:12 linkedFile  lrwxrwxrwx 1 user user 10 май 24 22:13 testFile -> linkedFile  user@user-virtual-machine:~/Documents/OS/SysCall$ sudo ./lchown  PRELCHOWN  lchown seccesful  user@user-virtual-machine:~/Documents/OS/SysCall$ ls -l  total 48  -rw-rw-r-- 1 user user 0 май 24 22:12 linkedFile  lrwxrwxrwx 1 **root root** 10 май 24 22:13 testFile -> linkedFile |

Трассировка программы

|  |
| --- |
| write(1, "PRELCHOWN\n", 9PRELCHOWN  ) = 9  **lchown("/home/user/Documents/OS/SysCall/testFile", 0, 0) = 0**  write(1, "lchown seccesful\n", 16chown seccesful  ) = 16  exit\_group(0) = ?  +++ exited with 0 +++ |

При трассировке виден системный вызов lchown.

Ядро 2.6

Трассировка программы

|  |
| --- |
| write(1, "PRELCHOWN\n", 10PRELCHOWN  ) = 10  lchown("/home/stas/Documents/OS/SysCall/testFile", 0, 0) = 0  write(1, "Lchown seccesful\n", 17Lchown seccesful  ) = 17  exit\_group(0) = ?  +++ exited with 0 +++ |

Системный вызов не отличается от версии ядра 4.8

**Chroot**

Тестовая программа

|  |
| --- |
| #include <unistd.h>  #include <errno.h>  #include <string.h>  #include <stdio.h>  int main(void)  {  printf("PRECHROOT\n");  if (chroot ("/home/user/Documents/OS/SysCall/chrootTest") == 0)  printf("Chroot seccesful\n");  else  {  printf("Error chroot: %s\n", strerror(errno));  }  while (1)  {  }  return 0;  } |

Выполнение без root

|  |
| --- |
| user@user-virtual-machine:~$ ps –aux  …  **user 19374 93.3 0.1 4344 720 pts/1 R+ 00:30 0:19 ./chroot** |

1. Перехват системных вызовов

Lubuntu 14.04

|  |
| --- |
| osboxes@osboxes:~/Documents/OS/replace$ cat /boot/System.map-\* | grep sys\_call\_table  c175f180 R sys\_call\_table |

Ubuntu 10.04

|  |
| --- |
| stas@ubuntu:~/Documents/OS/SysCall/callReplace$ cat /boot/System.map-2.6.32-21-generic | grep sys\_call\_table  ffffffff8154d380 R sys\_call\_table  ffffffff81556ef8 r ia32\_sys\_call\_table |

В коде программы изменяется только адрес на таблицу системных вызовов

Код программы[3]

|  |
| --- |
| #include <linux/kernel.h>  #include <linux/module.h>  #include <linux/moduleparam.h>  #include <linux/unistd.h>  #include <asm/cacheflush.h>  #include <linux/kallsyms.h>  #include <linux/utsname.h>  #define GPF\_DISABLE write\_cr0(read\_cr0() & (~ 0x10000))  #define GPF\_ENABLE write\_cr0(read\_cr0() | 0x10000)  // sys\_call\_table address in System.map  void \*\*sys\_call\_table = (void\*)0xc175f180;  long unsigned int addr = 0xc175f180;  asmlinkage int (\*original\_call) (const char\*, int, int);  asmlinkage long (\*original\_fchmodat) (int, const char \_\_user \*, mode\_t);  asmlinkage long (\*original\_fchownat) (int dfd, const char \_\_user \*filename, uid\_t user, gid\_t group, int flag);  asmlinkage long my\_fchmodat(int dfd, const char \_\_user \* filename, mode\_t mode)  {  printk("System call \_fchmodat was catched: path = %s, mode = %o\n", filename, mode);  return original\_fchmodat(dfd, filename, mode);  }  asmlinkage long my\_fchownat(int dfd, const char \_\_user \*filename, uid\_t user, gid\_t group, int flag)  {  printk("System call \_fchownat was catched: path = %s, user = %d, group = %d\n", filename, user, group);  return original\_fchownat(dfd, filename, user, group, flag);  }  int init\_module()  {  original\_fchmodat = sys\_call\_table[\_\_NR\_fchmodat];  original\_fchownat = sys\_call\_table[\_\_NR\_fchownat];  GPF\_DISABLE;  sys\_call\_table[\_\_NR\_fchmodat] = my\_fchmodat;  sys\_call\_table[\_\_NR\_fchownat] = my\_fchownat;  GPF\_ENABLE;  return 0;  }  void cleanup\_module()  {  // Restore the original calls  GPF\_DISABLE;  sys\_call\_table[\_\_NR\_fchmodat] = original\_fchmodat;  sys\_call\_table[\_\_NR\_fchownat] = original\_fchownat;  GPF\_ENABLE;  } |

Makefile

|  |
| --- |
| obj-m += replace.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 |

Загрузка в ядро

|  |
| --- |
| osboxes@osboxes:~/Documents/OS/replace$ make  make -C /lib/modules/4.4.0-31-generic/build M=/home/osboxes/Documents/OS/replace modules  make[1]: Entering directory `/usr/src/linux-headers-4.4.0-31-generic'  CC [M] /home/osboxes/Documents/OS/replace/replace.o  Building modules, stage 2.  MODPOST 1 modules  CC /home/osboxes/Documents/OS/replace/replace.mod.o  LD [M] /home/osboxes/Documents/OS/replace/replace.ko  make[1]: Leaving directory `/usr/src/linux-headers-4.4.0-31-generic'  osboxes@osboxes:~/Documents/OS/replace$ sudo insmod replace.ko  osboxes@osboxes:~/Documents/OS/replace$ sudo rmmod replace.ko |

Тестирование

|  |
| --- |
| osboxes@osboxes:~/Documents/OS$ chmod 666 testFile  osboxes@osboxes:~/Documents/OS$ dmesg | tail  …  [ 1598.108907] systemd-hostnamed[3762]: Warning: nss-myhostname is not installed. Changing the local hostname might make it unresolveable. Please install nss-myhostname!  [ 1673.303015] System call \_fchmodat was catched: path = testFile, mode = 666  osboxes@osboxes:~/Documents/OS$ sudo chown root:root testFile  osboxes@osboxes:~/Documents/OS$ dmesg | tail  …  [ 1598.108907] systemd-hostnamed[3762]: Warning: nss-myhostname is not installed. Changing the local hostname might make it unresolveable. Please install nss-myhostname!  [ 1673.303015] System call \_fchmodat was catched: path = testFile, mode = 666  [ 1700.585488] System call \_fchownat was catched: path = testFile, user = 0, group = 0  osboxes@osboxes:~/Documents/OS$ |

Попробуем воспользоваться тем, что системный вызов запускается в пространстве ядра для того чтобы обойти запрос на проверку прав пользователя root. Для этого в обработчик вызова chmod внедрим вызов обработчика chown. C помощью chown попробуем поменять группу и владельца того же файла, к которому применяется chmod

Код модуля ядра

|  |
| --- |
| #include <linux/kernel.h>  #include <linux/module.h>  #include <linux/moduleparam.h>  #include <linux/unistd.h>  #include <asm/cacheflush.h>  #include <linux/kallsyms.h>  #include <linux/utsname.h>  #define GPF\_DISABLE write\_cr0(read\_cr0() & (~ 0x10000))  #define GPF\_ENABLE write\_cr0(read\_cr0() | 0x10000)  // sys\_call\_table address in System.map  void \*\*sys\_call\_table = (void\*)0xc175f180;  long unsigned int addr = 0xc175f180;  asmlinkage int (\*original\_call) (const char\*, int, int);  asmlinkage long (\*original\_fchmodat) (int, const char \_\_user \*, mode\_t);  asmlinkage long (\*original\_fchownat) (int dfd, const char \_\_user \*filename, uid\_t user, gid\_t group, int flag);  asmlinkage long my\_fchmodat(int dfd, const char \_\_user \* filename, mode\_t mode)  {    original\_fchownat = sys\_call\_table[\_\_NR\_fchownat];  original\_fchownat(dfd, filename, 0, 0, 0);  printk("System call \_fchmodat was catched: path = %s, mode = %o\n", filename, mode);  return original\_fchmodat(dfd, filename, mode);  }  asmlinkage long my\_fchownat(int dfd, const char \_\_user \*filename, uid\_t user, gid\_t group, int flag)  {  printk("System call \_fchownat was catched: path = %s, user = %d, group = %d\n", filename, user, group);  return original\_fchownat(dfd, filename, user, group, flag);  }  int init\_module()  {  original\_fchmodat = sys\_call\_table[\_\_NR\_fchmodat];  //original\_fchownat = sys\_call\_table[\_\_NR\_fchownat];  GPF\_DISABLE;  sys\_call\_table[\_\_NR\_fchmodat] = my\_fchmodat;  //sys\_call\_table[\_\_NR\_fchownat] = my\_fchownat;  GPF\_ENABLE;  return 0;  }  void cleanup\_module()  {  // Restore the original calls  GPF\_DISABLE;  sys\_call\_table[\_\_NR\_fchmodat] = original\_fchmodat;  sys\_call\_table[\_\_NR\_fchownat] = original\_fchownat;  GPF\_ENABLE;  } |

Загрузим модуль в ядро с помощью insmod и попробуем поменять флаги файла testFile.

|  |
| --- |
| osboxes@osboxes:~/Documents/OS$ touch testFile  osboxes@osboxes:~/Documents/OS$ ls -l  -rw-rw-r-- 1 osboxes osboxes 0 Jun 2 05:38 testFile  osboxes@osboxes:~/Documents/OS$ chmod 555 testFile  osboxes@osboxes:~/Documents/OS$ ls -l  -r-xr-xr-x 1 osboxes osboxes 0 Jun 2 05:38 testFile  osboxes@osboxes:~/Documents/OS$ sudo chmod 555 testFile  osboxes@osboxes:~/Documents/OS$ ls -l-r-xr-xr-x 1 root root 0 Jun 2 05:38 testFile |

Как видно из вывода, функция chown в обработчике chmod, не привела к желаемым результатам при запуске от обычного пользователя. Тем не менее модуль работоспособен, так как функциональность chown выполнена при запуске chmod от привилегированного пользователя.

Вывод

В данной лабораторной работе исследованы исходные коды системных вызовов chmod, chown, lchown и chroot для версии ядра 2.6 и 4. Выявлены незначительные отличия в их реализации. Созданы программы, демонстрирующие осуществление системного вызова из кода программы на языке С. С помощью таблицы системных вызовов осуществлена подмена стандартных обработчиков. Это позволило осуществить некоторые дополнительные действия перед вызовом стандартного обработчика.

Список источников

1. <http://man7.org/linux/man-pages/man2/chmod.2.html>
2. <http://elixir.free-electrons.com/linux/>
3. https://stackoverflow.com/questions/2103315/linux-kernel-system-call-hooking-example/6742086