

Лабораторная работа №5

**Дискреционное разграничение прав в Linux. Исследование влияния
дополнительных атрибутов**

Хватов Максим

Содержание

1	Цель работы	5
2	Подготовка лабораторного стенда	6
3	Компилирование программ	7
4	Выполнение работы	10
5	Вывод	16

Список иллюстраций

4.1	Рис. 1	10
4.2	Рис. 2	11
4.3	Рис. 3	12
4.4	Рис. 4	13
4.5	Рис. 5	15

Список таблиц

1 Цель работы

Изучение механизмов изменения идентификаторов, применения SetUID- и Sticky-битов. Получение практических навыков работы в консоли с дополнительными атрибутами. Рассмотрение работы механизма смены идентификатора процессов пользователей, а также влияние бита Sticky на запись и удаление файлов.

2 Подготовка лабораторного стенда

Помимо прав администратора для выполнения части заданий потребуются средства разработки приложений. В частности, при подготовке стенда следует убедиться, что в системе установлен компилятор gcc (для этого, например, можно ввести команду `gcc -v`). Если же gcc не установлен, то его необходимо установить, например, командой `yum install gcc` которая определит зависимости и установит следующие пакеты: gcc, cloogpp, cpp, glibc-devel, glibc-headers, kernel-headers, libgomp, ppl, cloog-ppl, cpp, gcc, glibc-devel, glibc-headers, kernel-headers, libgomp, libstdc++-devel, mpfr, ppl, glibc, glibc-common, libgcc, libstdc++. Файловая система, где располагаются домашние директории и файлы пользователей (в частности, пользователя guest), не должна быть смонтирована с опцией `nosuid`. Так как программы с установленным битом SetUID могут представлять большую брешь в системе безопасности, в современных системах используются дополнительные механизмы защиты. Проследите, чтобы система защиты SELinux не мешала выполнению заданий работы. Если вы не знаете, что это такое, просто отключите систему запретов до очередной перезагрузки системы командой `setenforce 0`

После этого команда `getenforce` должна выводить `Permissive`. В этой работе система SELinux рассматриваться не будет.

3 Компилирование программ

Для выполнения четвёртой части задания вам потребуются навыки программирования, а именно, умение компилировать простые программы, написанные на языке C (C++), используя интерфейс CLI. Само по себе создание программ не относится к теме, по которой выполняется работа, а является вспомогательной частью, позволяющей увидеть, как реализуются на практике те или иные механизмы дискреционного разграничения доступа. Если при написании (или исправлении существующих) скриптов на `bash`-е у большинства системных администраторов не возникает проблем, то процесс компилирования, как показывает практика, вызывает необоснованные затруднения. Компиляторы, доступные в Linux-системах, являются частью коллекции GNU-компиляторов, известной как GCC (GNU Compiler Collection, подробнее см. <http://gcc.gnu.org>). В неё входят компиляторы языков C, C++, Java, Objective-C, Fortran и Chill. Будем использовать лишь первые два. Компилятор языка C называется `gcc`. Компилятор языка C++ называется `g++` и запускается с параметрами почти так же, как `gcc`. Проверить это можно следующими командами: `whereis gcc whereis g++` Первый шаг заключается в превращении исходных файлов в объектный код: `gcc -c file.c` В случае успешного выполнения команды (отсутствие ошибок в коде) полученный объектный файл будет называться `file.o`. Объектные файлы невозможно запускать и использовать, поэтому после компиляции для получения готовой программы объектные файлы необходимо компоновать. Компоновать можно один или несколько файлов. В случае использования хотя бы одного из файлов, написанных на C++, компоновка производится с помощью компилятора

g++. Строго говоря, это тоже не вполне верно. Компоновка объектного кода, сгенерированного чем бы то ни было (хоть вручную), производится линкером ld, g++ его просто вызывает изнутри. Если же все файлы написаны на языке C, нужно использовать компилятор gcc. Например, так: gcc -o program file.o В случае успешного выполнения команды будет создана программа program (исполняемый файл формата ELF с установленным атрибутом +x). Компилирование — это процесс. Компилятор gcc (g++) имеет множество параметров, влияющих на процесс компиляции. Он поддерживает различные режимы оптимизации, выбор платформы назначения и пр. Также возможно использование make-файлов (Makefile) с помощью утилиты make для упрощения процесса компиляции.

Такое решение подойдёт лишь для простых случаев. Если говорить про пример выше, то компилирование одного файла из двух шагов можно сократить вообще до одного, например: gcc file.c В этом случае готовая программа будет иметь название a.out. Механизм компилирования программ в данной работе не мог быть не рассмотрен потому, что использование программ, написанных на bash, для изучения SetUID- и SetGID- битов, не представляется возможным. Связано это с тем, что любая bash-программа интерпретируется в процессе своего выполнения, т.е. существует сторонняя программа-интерпретатор, которая выполняет считывание файла сценария и выполняет его последовательно. Сам интерпретатор выполняется с правами пользователя, его запустившего, а значит, и выполняемая программа использует эти права. При этом интерпретатору абсолютно всё равно, установлены SetUID-, SetGID-биты у текстового файла сценария, атрибут разрешения запуска «x» или нет. Важно, чтобы был установлен лишь атрибут, разрешающий чтение «r». Также не важно, был ли вызван интерпретатор из командной строки (запуск файла, как bash file1.sh), либо внутри файла была указана строчка #!/bin/bash. Логично спросить: если установление SetUID- и SetGID- битов на сценарий не приводит к нужному результату как с исполняемыми файлами, то что мешает установить эти биты на сам интерпретатор? Ничего не мешает, только их установление приведёт к тому, что, так как

владельцем `/bin/bash` является `root`: `ls -l /bin/bash` все сценарии, выполняемые с использованием `/bin/bash`, будут иметь возможности суперпользователя — совсем не тот результат, который хотелось бы видеть. Если сомневаетесь в выше сказанном, создайте простой файл `progl.sh` следующего содержания: `#!/bin/bash /usr/bin/id /usr/bin/whoami` и попробуйте поменять его атрибуты в различных конфигурациях. Подход вида: сделать копию `/bin/bash`, для нее `chown user:users` и потом SUID также плох, потому что это позволит запускать любые команды от пользователя `user`

4 Выполнение работы

Захожу в систему от имени пользователя guest и создаю файл simpleid.c с помощью команды touch

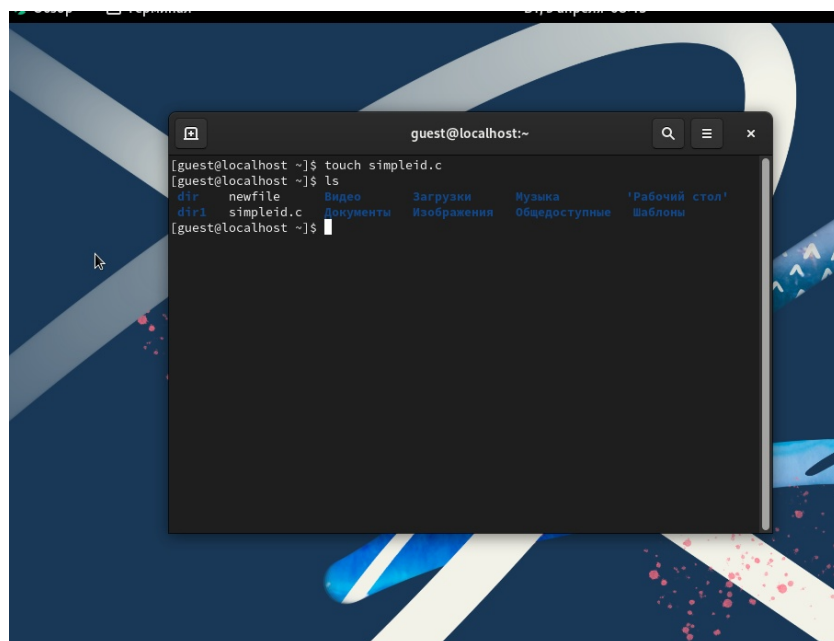


Рис. 4.1: Рис. 1

Затем пишу код в этом файле

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int
main ()
```

```

{
uid_t uid = geteuid ();
gid_t gid = getegid ();
printf ("uid=%d, gid=%d\n", uid, gid);
return 0;
}

```

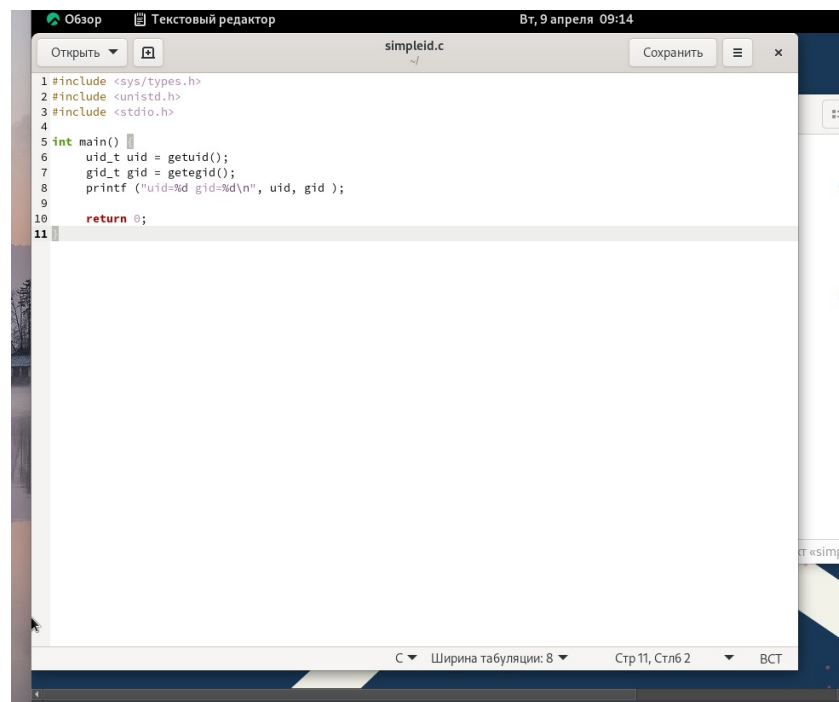


Рис. 4.2: Рис. 2

Компилирую программу с помощью gcc и команды gcc simpleid.c -o simpleid

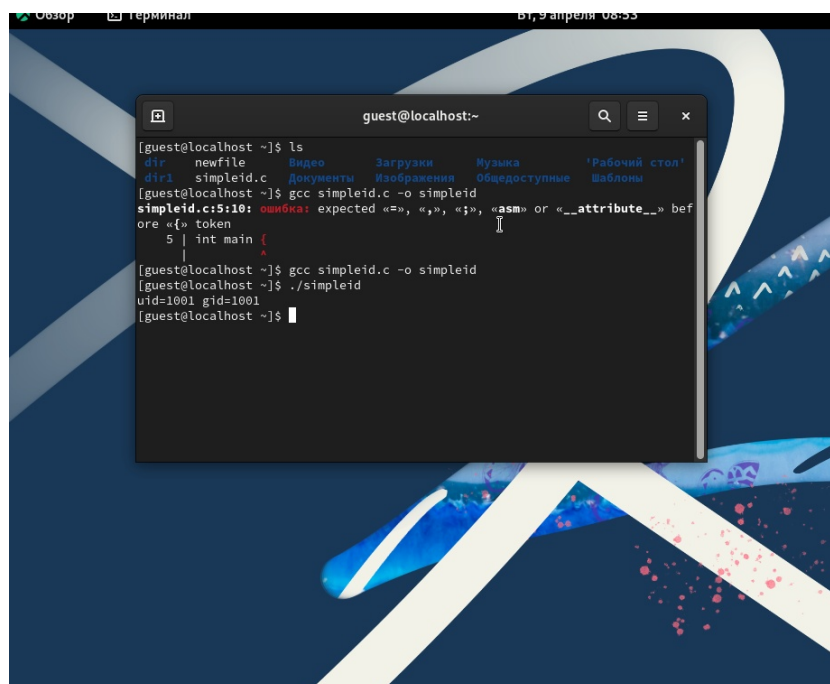


Рис. 4.3: Рис. 3

После этого запускаю полученный исполняемый файл с помощью команды `./simpleid` и выполняю затем команду `id`.

Результат на рисунке 3. Значения, выдаваемые программой и системной командой одинаковые.

Усложняю программу, создав новый файл `simpleid.c`. Вписываю код

```

#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int
main ()
{
    uid_t real_uid = getuid ();
    uid_t e_uid = geteuid ();
    gid_t real_gid = getgid ();

```

```

gid_t e_gid = getegid ( ) ;
printf ("e_uid=%d, e_gid=%d\n", e_uid, e_gid);
printf ("real_uid=%d, real_gid=%d\n", real_uid,
,→ real_gid);
return 0;
}

```

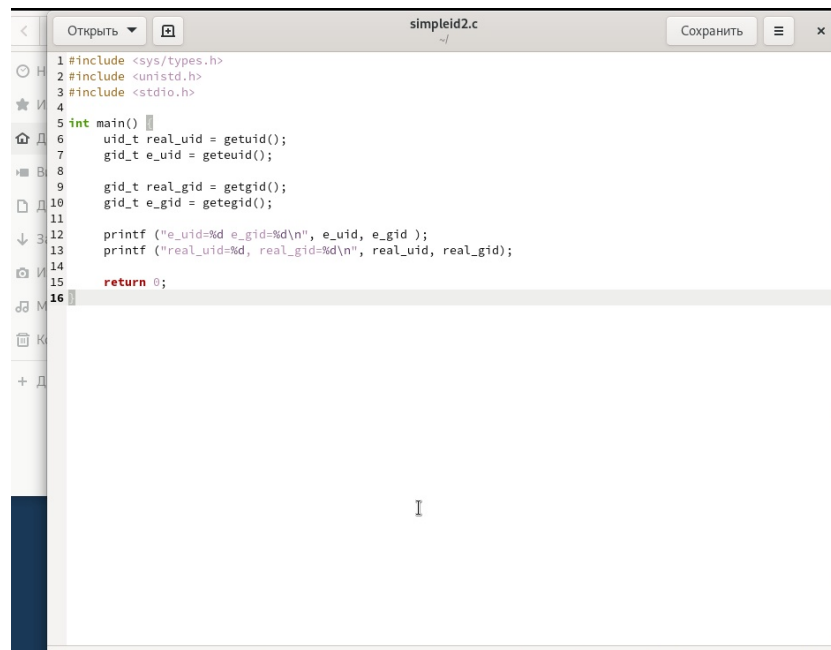


Рис. 4.4: Рис. 4

Компилирую с помощью команды `gcc simpleid2.c -o simpleid2`, получаю исполняемый файл и запускаю его командой `./simpleid2`

Создаю файл `readfile.c` с помощью команды `touch`

```

#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

```

```

int
main (int argc, char* argv[])
{
unsigned char buffer[16];
size_t bytes_read;
int i;
int fd = open (argv[1], O_RDONLY);
do
{
bytes_read = read (fd, buffer, sizeof (buffer));
for (i =0; i < bytes_read; ++i) printf("%c", buffer[i]);
}
while (bytes_read == sizeof (buffer));
close (fd);
return 0;
}

```

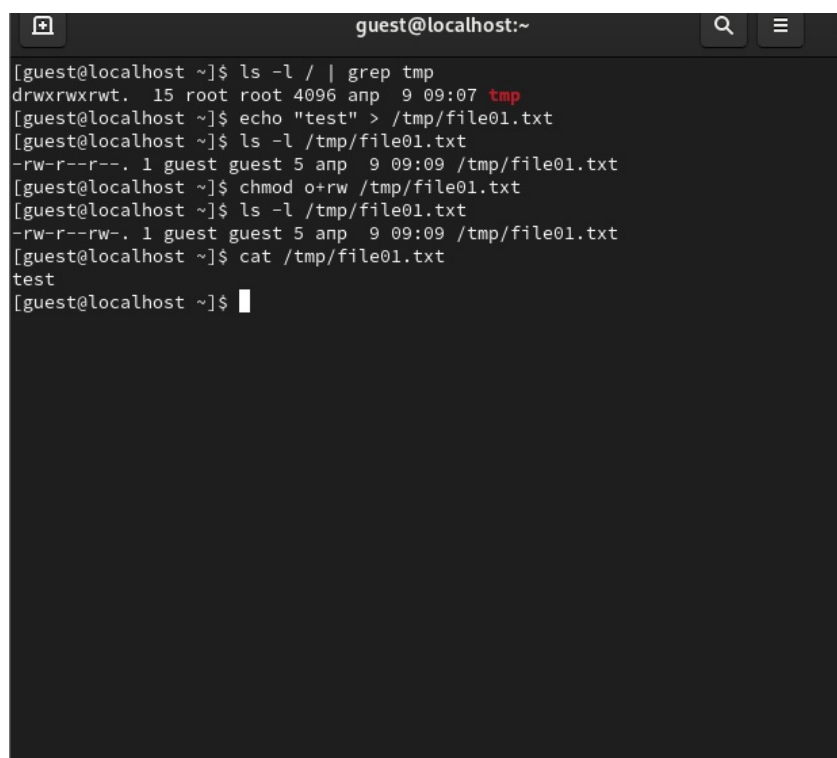
Компилирую файл командой `gcc readfile.c -o readfile`

Выполняю команду `ls -l / | grep tmp` и выясняю, что атрибут установлен. От имени пользователя `guest` создаю файл и ввожу в него текст командой `echo "test"`
`> /tmp/file01.txt`

Просматриваю атрибуты и разрешаю чтение и запись для категории пользователей "все остальное", выполняя команды

```
ls -l /tmp/file01.txt chmod o+rw /tmp/file01.txt ls -l /tmp/file01.txt
```

Проверяю содержимое файла командой `cat /tmp/file01.txt`



```
guest@localhost:~  
[guest@localhost ~]$ ls -l / | grep tmp  
drwxrwxrwt. 15 root root 4096 anp  9 09:07 tmp  
[guest@localhost ~]$ echo "test" > /tmp/file01.txt  
[guest@localhost ~]$ ls -l /tmp/file01.txt  
-rw-r--r--. 1 guest guest 5 anp  9 09:09 /tmp/file01.txt  
[guest@localhost ~]$ chmod o+rw /tmp/file01.txt  
[guest@localhost ~]$ ls -l /tmp/file01.txt  
-rw-r--rw-. 1 guest guest 5 anp  9 09:09 /tmp/file01.txt  
[guest@localhost ~]$ cat /tmp/file01.txt  
test  
[guest@localhost ~]$
```

Рис. 4.5: Рис. 5

5 Вывод

Я изучил механизмы изменения идентификаторов, применения SetUID-и Sticky-битов. Получил практические навыки работы в консоли с дополнительными атрибутами. Рассмотрел работы механизма смены идентификатора процессов пользователей, а также влияние бита Sticky на запись и удаление файлов.