

Лабораторная работа №5

Дисциплина: Информационная безопасность

Дорофеева Алёна Тимофеевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Исследование Sticky-бита	14
5	Выводы	17
	Список литературы	18

Список иллюстраций

4.1	Подготовка лабораторного стенда	9
4.2	Программа simpleid	10
4.3	Программа simpleid	10
4.4	Усложнение программы	11
4.5	Программа simpleid2	11
4.6	SetUID- и SetGID-биты	12
4.7	Файл readfile.c	13
4.8	Смена владельца файла readfile.c	13
4.9	Чтение файла readfile.c	13
4.10	Чтение файла /etc/shadow	14
4.11	Проверка наличия Sticky-бита на директории /tmp	15
4.12	Проверка возможности действий при наличии Sticky-бита	15
4.13	Снятие Sticky-бита	16
4.14	Проверка возможности действий при отсутствии Sticky-бита	16

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 Задание

- Изучить на практике применение SetUID- и Sticky-битов.

3 Теоретическое введение

Setuid – это бит разрешения, который позволяет пользователю запускать исполняемый файл с правами владельца этого файла. Другими словами, использование этого бита позволяет нам поднять привилегии пользователя в случае, если это необходимо. Классический пример использования этого бита в операционной системе это команда `sudo`.

Там, где обычно установлен классический бит `x` (на исполнение), выставляется специальный бит `s`. Это позволяет обычному пользователю системы выполнять команды с повышенными привилегиями без необходимости входа в систему как `root`, разумеется зная пароль пользователя `root`. Принцип работы **Setgid** очень похож на `setuid` с отличием, что файл будет запускаться пользователем от имени группы, которая владеет файлом.

Биты **setuid** и **setgid** выставляются с помощью команд `chmod u+s` и `chmod g+s` соответственно.[01?]

Sticky-bit — дополнительный атрибут файлов или каталогов в операционных системах семейства UNIX. В настоящее время `sticky bit` используется в основном для каталогов, чтобы защитить в них файлы. Из такого каталога пользователь может удалить только те файлы, владельцем которых он является. Примером может служить каталог `/tmp`, в который запись открыта для всех пользователей, но нежелательно удаление чужих файлов. Установка атрибута производится утилитой `chmod`.

В операционной системе Solaris для файлов, не являющихся программами, имеет строго противоположное действие — запрещает сохранение данных этого

файла в системном кэше.[02?]

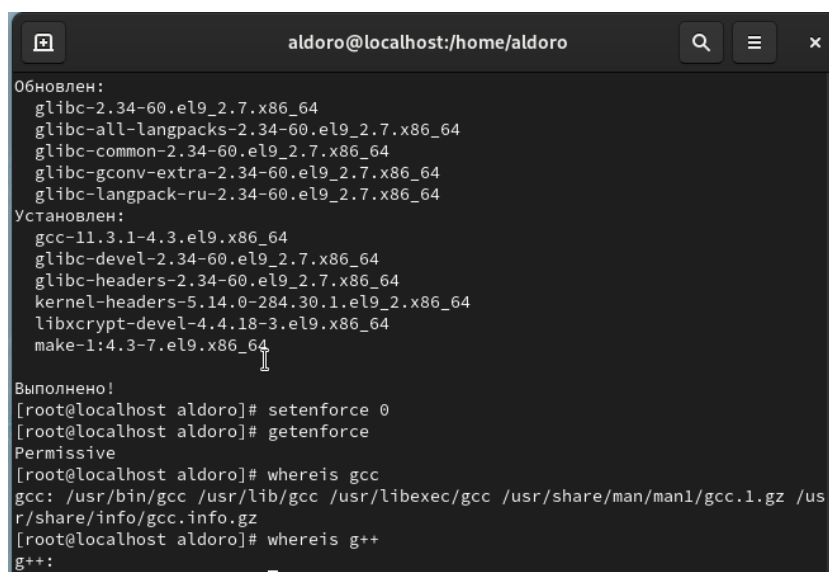
Это разрешение полезно для защиты файлов от случайного удаления в среде, где несколько пользователей имеют права на запись в один и тот же каталог. Если применяется закрепленный sticky bit, пользователь может удалить файл, только если он является пользователем-владельцем файла или каталога, в котором содержится файл. По этой причине он применяется в качестве разрешения по умолчанию для каталога /tmp и может быть полезен также для каталогов общих групп.

Когда вы применяете sticky bit, пользователь может удалять файлы, только если выполняется одно из следующих условий:

- Пользователь является владельцем файла;
- Пользователь является владельцем каталога, в котором находится файл.[03?]

4 Выполнение лабораторной работы

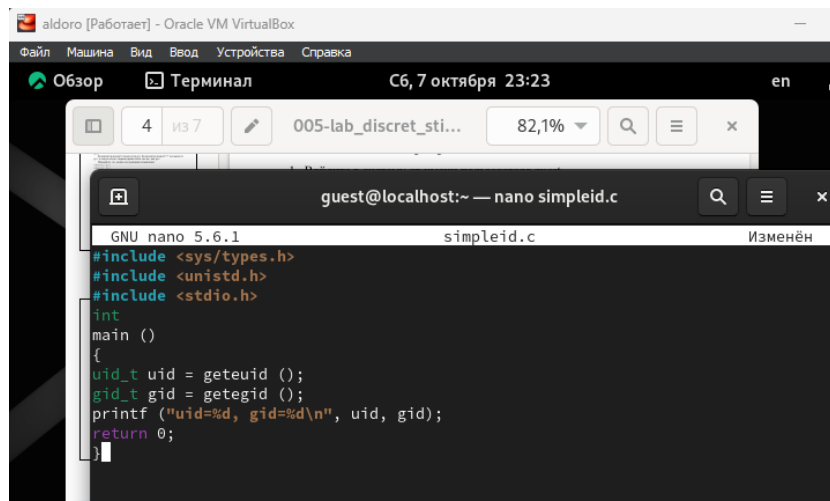
1. Вошла в систему от имени пользователя guest (рис. ??). Убедилась, что в системе установлен компилятор gcc, проверив версию командой `gcc -v` (рис. ??). Видим, что он установлен. Отключила систему запретов до очередной перезагрузки командой `setenforce 0` (рис. ??). После этого команда `getenforce` выводит “Permissive” (рис. ??).



```
aldoro@localhost:/home/aldoro
Обновлен:
glibc-2.34-60.el9_2.7.x86_64
glibc-all-langpacks-2.34-60.el9_2.7.x86_64
glibc-common-2.34-60.el9_2.7.x86_64
glibc-gconv-extra-2.34-60.el9_2.7.x86_64
glibc-langpack-ru-2.34-60.el9_2.7.x86_64
Установлен:
gcc-11.3.1-4.3.el9.x86_64
glibc-devel-2.34-60.el9_2.7.x86_64
glibc-headers-2.34-60.el9_2.7.x86_64
kernel-headers-5.14.0-284.30.1.el9_2.x86_64
libxcrypt-devel-4.4.18-3.el9.x86_64
make-1:4.3-7.el9.x86_64
Выполнено!
[root@localhost aldoro]# setenforce 0
[root@localhost aldoro]# getenforce
Permissive
[root@localhost aldoro]# whereis gcc
gcc: /usr/bin/gcc /usr/lib/gcc /usr/libexec/gcc /usr/share/man/man1/gcc.1.gz /usr/share/info/gcc.info.gz
[root@localhost aldoro]# whereis g++
g++:
```

Рис. 4.1: Подготовка лабораторного стенда

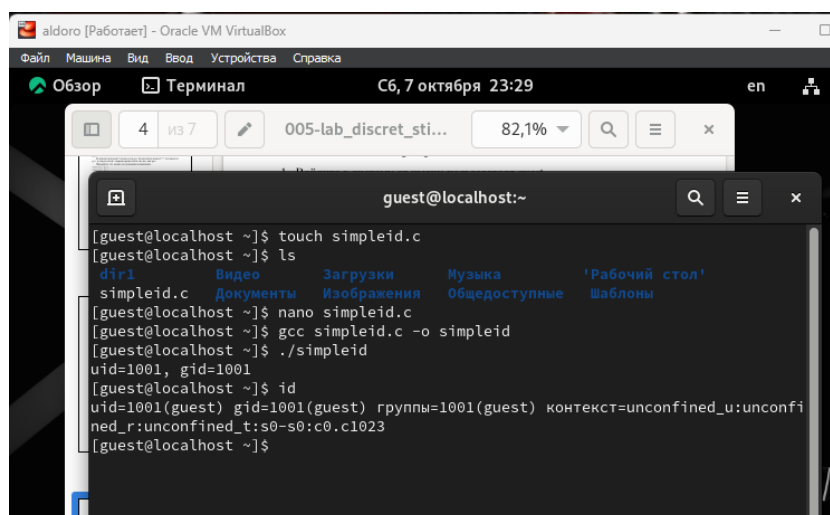
2. Создала программу `simpleid.c` в текстовом редакторе `nano` (рис. 4.2).



```
GNU nano 5.6.1 simpleid.c
#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: Программа simpleid

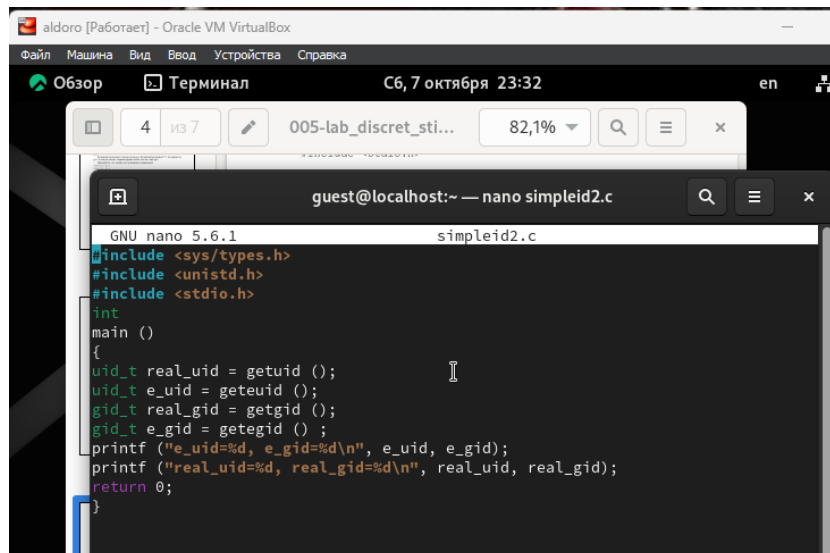
3. Скомпилировала программу и убедилась, что файл программы создан, командой `gcc simpleid.c -o simpleid` (рис. 4.3).
4. Выполнила программу `simpleid`, введя `./simpleid` (рис. 4.3).
5. Выполнила системную программу `id` (рис. 4.3). Видим, что выводимая информация о `uid` и `gid` идентична, но команда `id` также выводит `groups=1001(guest)`.



```
[guest@localhost ~]$ touch simpleid.c
[guest@localhost ~]$ ls
dir1 Видео Загрузки Музыка 'Рабочий стол'
simpleid.c Документы Изображения Общедоступные Шаблоны
[guest@localhost ~]$ nano simpleid.c
[guest@localhost ~]$ gcc simpleid.c -o simpleid
[guest@localhost ~]$ ./simpleid
uid=1001, gid=1001
[guest@localhost ~]$ id
uid=1001(guest) gid=1001(guest) группы=1001(guest) контекст=unconfined_u:unconfi
ned_r:unconfined_t:s0-s0:c0.c1023
[guest@localhost ~]$
```

Рис. 4.3: Программа simpleid

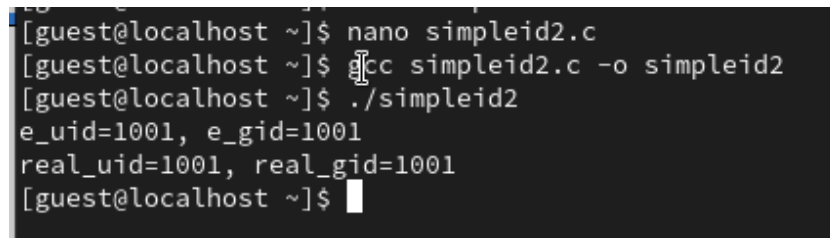
6. Усложнила программу, добавив вывод действительных идентификаторов (рис. 4.4). Получившуюся программу назвала simpleid2.c (рис. 4.4).



```
GNU nano 5.6.1 simpleid2.c
#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: Усложнение программы

7. Скомпилировала программу командой `gcc simpleid2.c -o simpleid2` (рис. ??). Запустила программу, введя `./simpleid2` (рис. ??).



```
[guest@localhost ~]$ nano simpleid2.c
[guest@localhost ~]$ gcc simpleid2.c -o simpleid2
[guest@localhost ~]$ ./simpleid2
e_uid=1001, e_gid=1001
real_uid=1001, real_gid=1001
[guest@localhost ~]$
```

Рис. 4.5: Программа simpleid2

8. Выполнила команды `sudo chown root:guest /home/guest/simpleid2` и `sudo chmod u+s /home/guest/simpleid2` (рис. 4.6).
9. Первая команда меняет у файла владельца и группу. Вторая команда ставит SetUID-бит.

10. Выполнила проверку правильности установки новых атрибутов и смены владельца файла `simpleid2` с помощью команды `ls -l simpleid2` (рис. 4.6). Видим, что все установлено верно.
11. Запустила файл `simpleid2` командой `./simpleid2` (рис. 4.6). Также выполнила системную команду `id` (рис. 4.6). Видим, что файл выводит информацию не только о реальном идентификаторе, но и эффективных ID - EUID и EGID. Фактические ID пользователя и группы соответствуют ID пользователя, который вызвал процесс (пользователь `guest`, группа `guest`). Эффективный ID пользователя соответствует установленному SetUid биту на исполняемом файле (привилегии суперпользователя). Эффективный ID группы соответствует установленному SetGid биту на исполняемом файле. Команда `id` выводит информацию только о реальном идентификаторе.
12. Прodelала то же самое относительно SetGID-бита (рис. 4.6). Видим, что теперь при исполнении файла выводится EUID = 1001, EGID = 0, так как мы установили SetGid бит.

```
[guest@localhost ~]$ su aldoro
Пароль:
[aldoro@localhost guest]$ sudo -i

Мы полагаем, что ваш системный администратор изложил вам основы
безопасности. Как правило, всё сводится к трём следующим правилам:

    №1) Уважайте частную жизнь других.
    №2) Думайте, прежде что-то вводить.
    №3) С большой властью приходит большая ответственность.

[sudo] пароль для aldoro:
[root@localhost ~]# chown root:guest /home/guest/simpleid2
[root@localhost ~]# chmod u+s /home/guest/simpleid2
[root@localhost ~]#
```

Рис. 4.6: SetUID- и SetGID-биты

13. Создала программу `readfile` с помощью текстового редактора `nano` (рис. 4.7).
14. Откомпилировала её командой `gcc readfile.c -o readfile` (рис. 4.7).

```
[root@localhost guest]# ls -l simpleid2
-rwsr-xr-x. 1 root guest 26008 окт  7 23:33 simpleid2
[root@localhost guest]#
```

Рис. 4.7: Файл readfile.c

15. Сменила владельца у файла readfile.c на root командой `sudo chown root /home/guest/readfile` и изменила права так, чтобы только суперпользователь (в моем случае владелец) мог прочитать его, а guest (член группы) не мог командой `sudo chmod 733 /home/guest/readfile` (рис. 4.8).
16. Проверила, что пользователь guest не может прочитать файл readfile.c командой `cat readfile.c` (рис. 4.8). Видим, что нам действительно отказано в доступе.
17. Сменила у программы readfile владельца обратно на guest и установила SetUID-бит командами `sudo chown guest:guest /home/guest/readfile` и `sudo chmod u+s /home/guest/readfile` (рис. 4.8).

```
[root@localhost guest]# ./simpleid2
e_uid=0, e_gid=0
real_uid=0, real_gid=0
[root@localhost guest]# id
uid=0(root) gid=0(root) группы=0(root) контекст=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[root@localhost guest]#
```

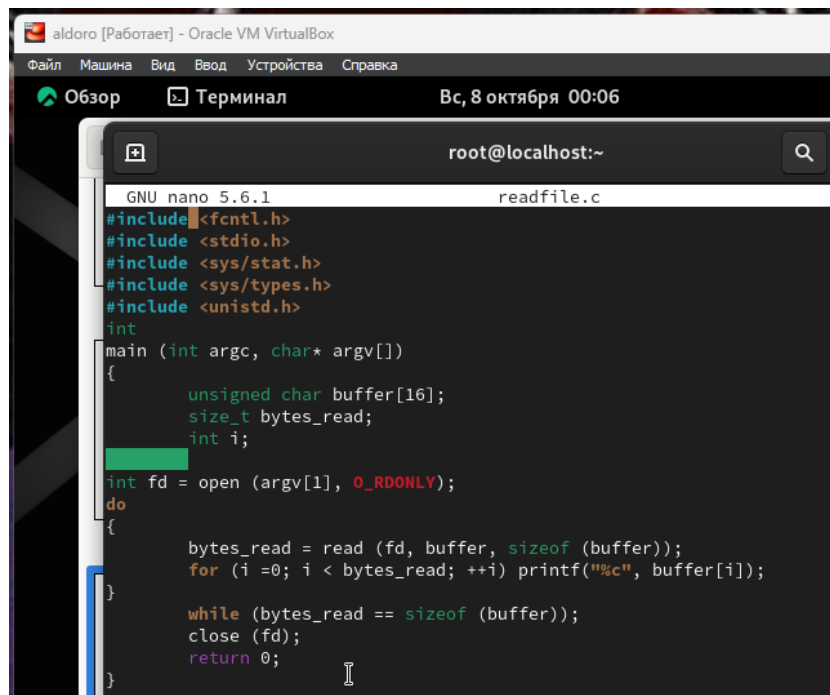
Рис. 4.8: Смена владельца файла readfile.c

18. Проверила, может ли программа readfile прочитать файл readfile.c (рис. 4.9). Видим, что может, так как мы установили SetUID-бит.

```
[root@localhost ~]# chown root:root /home/guest/simpleid2
[root@localhost ~]# chmod g+s /home/guest/simpleid2
[root@localhost ~]#
```

Рис. 4.9: Чтение файла readfile.c

19. Проверил, может ли программа `readfile` прочитать файл `/etc/shadow` (рис. 4.10). Видим, что может, так как мы установили SetUID-бит.



```
GNU nano 5.6.1 readfile.c
#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;
}
```

Рис. 4.10: Чтение файла `/etc/shadow`

4.1 Исследование Sticky-бита

1. Выяснила, установлен ли атрибут Sticky на директории `/tmp`, для чего выполнила команду `ls -l / | grep tmp` (рис. 4.11). Видим, что он установлен (буква `t` в конце).
2. От имени пользователя `guest` создала файл `file01.txt` в директории `/tmp` со словом `test` командой `echo "test" > /tmp/file01.txt` (рис. 4.11).
3. Просмотрела атрибуты у только что созданного файла (команда `ls -l /tmp/file01.txt`) и разрешила чтение и запись для категории пользователей «все остальные» командой `chmod o+rw /tmp/file01.txt` (рис. 4.11). Проверила атрибуты (команда `ls -l /tmp/file01.txt`) (рис. 4.11).

```

[guest@localhost ~]$ touch /tmp/file01.txt
[root@localhost ~]# touch readfile.c
[root@localhost ~]# nano readfile.c
[root@localhost ~]# gcc readfile.c -o readfile
[root@localhost ~]#

```

Рис. 4.11: Проверка наличия Sticky-бита на директории /tmp

4. От пользователя guest2 (не являющегося владельцем) попробовала прочитать файл /tmp/file01.txt командой `cat /tmp/file01.txt` (рис. 4.12). Прочитать файл удалось.
5. От пользователя guest2 попробовала дозаписать в файл /tmp/file01.txt слово test2 командой `echo "test2" >> /tmp/file01.txt` (рис. 4.12). Выполнить данную операцию не удалось.
6. Проверила содержимое файла командой `cat /tmp/file01.txt` (рис. 4.12). Действие по сути бессмысленное, так как файл не менялся.
7. От пользователя guest2 попробовала записать в файл /tmp/file01.txt слово test3, стерев при этом всю имеющуюся в файле информацию командой `echo "test3" > /tmp/file01.txt` (рис. 4.12). Выполнить эту операцию не удалось.
8. Проверила содержимое файла командой `cat /tmp/file01.txt` (рис. 4.12). Действие по сути бессмысленное, так как файл не менялся.
9. От пользователя guest2 попробовала удалить файл /tmp/file01.txt командой `rm /tmp/file01.txt` (рис. 4.12).

```

[guest@localhost ~]$ ls -l readfile.c
-rwxr-x---. 1 root root 402 окт  8 00:17 readfile.c
[guest@localhost ~]$ cat readfile.c
cat: readfile.c: Отказано в доступе
[guest@localhost ~]$

```

Рис. 4.12: Проверка возможности действий при наличии Sticky-бита

10. Повысила свои права до суперпользователя следующей командой `su -` и выполнида после этого команду `chmod -t /tmp`, снимающую атрибут `t` (Sticky-бит) с директории `/tmp` (рис. 4.13).
11. Покинула режим суперпользователя командой `exit` (рис. 4.13)

```
andconda KS:erg readfile readfile  
[root@localhost ~]# chown root:guest /home/guest/readfile.c  
[root@localhost ~]# chmod u+s /home/guest/readfile.c  
[root@localhost ~]#
```

Рис. 4.13: Снятие Sticky-бита

12. От пользователя `guest2` проверила, что атрибута `t` у директории `/tmp` нет командой `ls -l / | grep tmp` (рис. 4.14).
13. Повторила предыдущие шаги (рис. 4.14). Теперь мы смогли удалить файл.
14. Нам удалось удалить файл, когда мы сняли Sticky-бит.

```
[root@localhost ~]# ls -l / | grep tmp  
drwxrwxrwt. 17 root root 4096 окт  8 00:26 tmp  
[root@localhost ~]#
```

Рис. 4.14: Проверка возможности действий при отсутствии Sticky-бита

5 Выводы

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

Список литературы