

Отчет по лабораторной работе 5

Даниил Генералов, 1032212280

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	10

Список иллюстраций

2.1	id	6
2.2	simpleid2	7
2.3	readfile	7
2.4	setuid	8
2.5	sticky	9

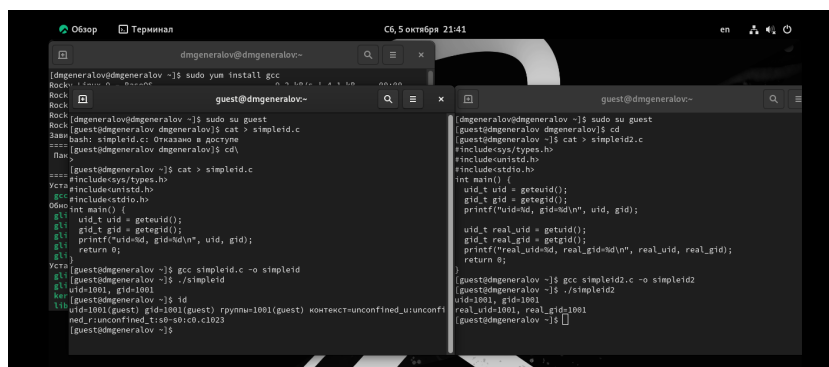
Список таблиц

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 Выполнение лабораторной работы

Сначала требуется установить компилятор gcc с помощью `yum install`, а также временно выключить SELinux: это нужно, чтобы мы могли скомпилировать программу и затем сделать так, чтобы она запускалась с атрибутом `setuid`. После того, как мы установили их, мы компилируем и запускаем две программы: `simpleid` и `simpleid2`: они выводят идентификаторы пользователя и группы, под которыми запущены. Эти числа совпадают с выводом команды `id`, что можно увидеть на рис. fig. 2.1.



```
dmgeneralov@dmgeneralov- ~$ sudo yum install gcc
Rock
Rock
Rock
dmgeneralov@dmgeneralov- ~$ sudo su guest
[guest@dmgeneralov- ~]$ cat > simpleid.c
#include<sys/types.h>
#include<unistd.h>
int main() {
    uid_t uid = getuid();
    gid_t gid = getgid();
    printf("uid=%d, gid=%d\n", uid, gid);
    return 0;
}
[guest@dmgeneralov- ~]$ gcc simpleid.c -o simpleid
[guest@dmgeneralov- ~]$ ./simpleid
uid=1001, gid=1001
[guest@dmgeneralov- ~]$ id
uid=1001(guest) gid=1001(guest) rpnw-1001(guest) контекст=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0-c1023
[guest@dmgeneralov- ~]$

[dmgeneralov@dmgeneralov- ~]$ sudo su guest
[guest@dmgeneralov- ~]$ cd
[guest@dmgeneralov- ~]$ cat > simpleid2.c
#include<sys/types.h>
#include<unistd.h>
int main() {
    uid_t uid = getuid();
    gid_t gid = getgid();
    printf("uid=%d, gid=%d\n", uid, gid);

    uid_t real_uid = getuid();
    gid_t real_gid = getgid();
    printf("real_uid=%d, real_gid=%d\n", real_uid, real_gid);
    return 0;
}
[guest@dmgeneralov- ~]$ gcc simpleid2.c -o simpleid2
[guest@dmgeneralov- ~]$ ./simpleid2
uid=1001, gid=1001
real_uid=0, real_gid=0
[guest@dmgeneralov- ~]$
```

Рис. 2.1: id

После этого мы меняем исполняемый файл `simpleid2`: у него теперь будет владелец `root:guest` и для пользователя будет поставлен `setuid`-бит. Теперь в выводе программы будет видно, что настоящий `id` пользователя сохранился как `1001`, но эффективный `id` – `0` (`root`) благодаря `setuid`-биту; аналогично, если файл имеет `setgid`-бит для группы, и группа этого файла равна `root`, то эффективный `id` группы будет равен `0` (см. рис. fig. 2.2).

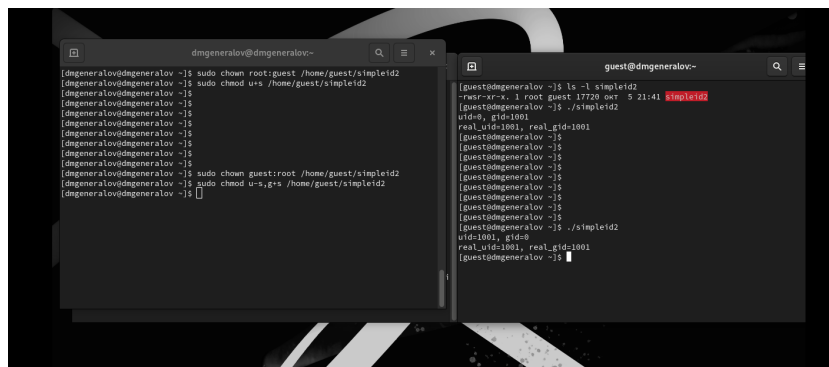


Рис. 2.2: simpleid2

После этого мы компилируем новую программу, которая называется readfile: эта программа принимает на вход название файла и пытается прочитать его. Сначала мы пытаемся с помощью этой программы прочитать файл, который доступен пользователю guest, и это удастся. После этого мы делаем этот файл недоступным для пользователя guest, и программа ломается, потому что она получает ошибку при открытии файла, и вместо этого начинает читать оперативную память рядом с переменной buffer, в итоге выходя за границы разрешенного сегмента памяти и умирая (рис. fig. 2.3).

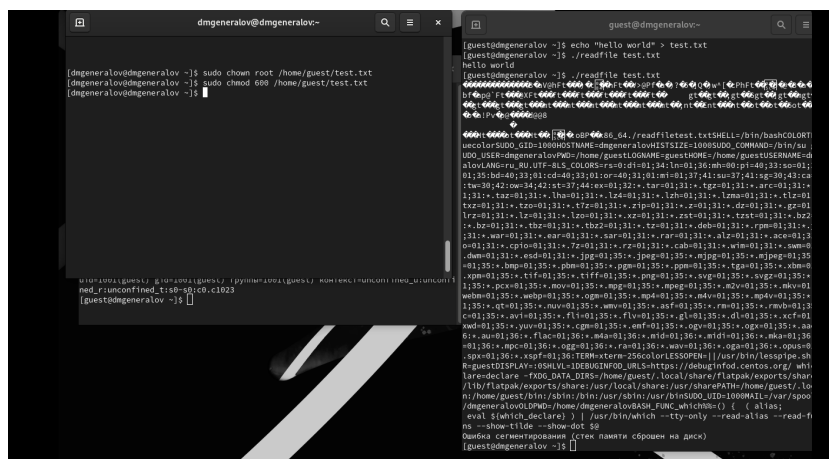


Рис. 2.3: readfile

Но после этого мы добавляем setuid-бит для этой программы, и теперь мы можем прочитать файлы, которые закрыты от нас, как можно увидеть на рис. fig. 2.4.

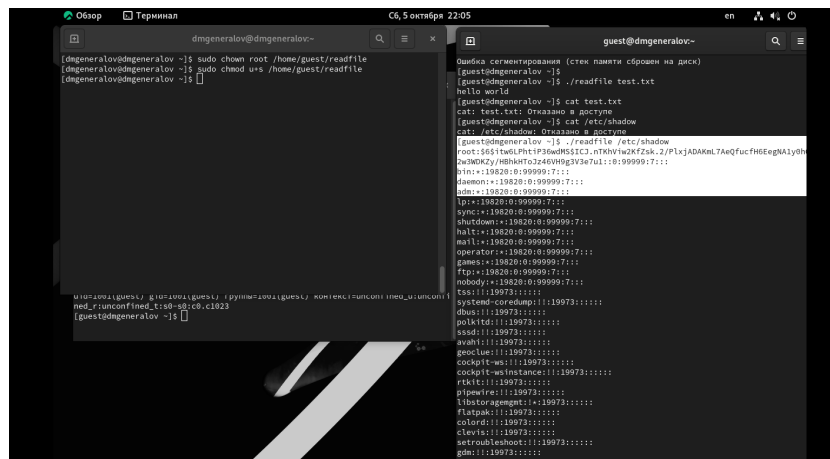


Рис. 2.4: setuid

Это связано с тем, что когда setuid-бит задан, то программа запускается не с правами того пользователя, который ее запустил, а с правами владельца этой программы. Поскольку в данном случае владельцем является root, который имеет права на доступ ко всем файлам в системе, то программа может прочитать файлы, которые закрыты от нас.

Теперь мы проверяем поведение sticky-бита. Для этого мы создаем файл в папке /tmp (которая по умолчанию имеет этот sticky-бит) от имени пользователя guest. После этого мы даем доступ к этому файлу от группы “все остальные”, и после этого используем пользователя guest2, чтобы добавить текст к этому файлу, а затем попробовать его перезаписать. Обе этих вещи не получаются. После этого мы пробуем прочитать файл (что получается), а затем удалить его (что не получается благодаря sticky-биту). Затем мы снимаем sticky-бит с папки /tmp и создаем файл заново: как результат, хотя мы все еще не можем изменить содержимое этого файла от имени пользователя guest2, он все-таки может удалить этот файл. Это можно увидеть на рис. fig. 2.5.

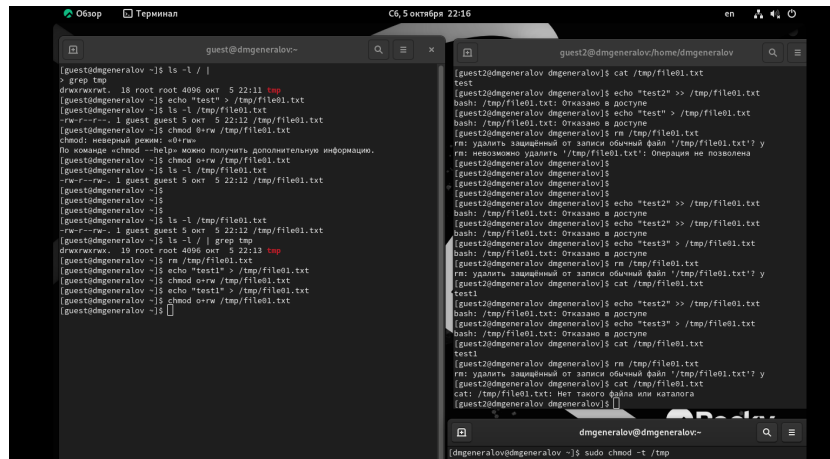


Рис. 2.5: sticky

Смысл sticky-бита заключается в том, чтобы файлы в папке принадлежали ее владельцам, и другие пользователи – даже те, кто имеет доступ к этой папке – не могли удалять их. Это очень нужно для /tmp и других общих директорий, потому что тогда один пользователь не может удалить файлы другого пользователя.

3 Выводы

В этой лабораторной работе мы использовали механизмы sticky, setuid и setgid-битов, чтобы посмотреть на некоторые продвинутые механизмы работы с разрешениями файлов в Linux.