

Лабораторная работа № 5. Дискреционное разграничение прав в Linux. Исследование влияния дополнительных атрибутов

5.1. 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.¹

5.2. Указания к работе

5.2.1. 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, cloog-ppl, 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.

¹При составлении работы использовались материалы [2—4].

5.2.2. Компилирование программ

Для выполнения четвёртой части задания вам потребуются навыки программирования, а именно, умение компилировать простые программы, написанные на языке С (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 скомпоновать. 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 плохо, потому что это позволит запускать любые команды от пользователя `user`.

5.3. Порядок выполнения работы

5.3.1. Создание программы

1. Войдите в систему от имени пользователя `guest`.
2. Создайте программу `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;
}
```

3. Скомпилируйте программу и убедитесь, что файл программы создан:

```
gcc simpleid.c -o simpleid
```

4. Выполните программу simpleid:

```
./simpleid
```

5. Выполните системную программу id:

```
id
```

и сравните полученный вами результат с данными предыдущего пункта задания.

6. Усложните программу, добавив вывод действительных идентификаторов:

```
#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;
}
```

Получившуюся программу назовите simpleid2.c.

7. Скомпилируйте и запустите simpleid2.c:

```
gcc simpleid2.c -o simpleid2
./simpleid2
```

8. От имени суперпользователя выполните команды:

```
chown root:guest /home/guest/simpleid2
```

```
chmod u+s /home/guest/simpleid2
```

9. Используйте `sudo` или повысьте временно свои права с помощью `su`. Поясните, что делают эти команды.
10. Выполните проверку правильности установки новых атрибутов и смены владельца файла `simpleid2`:

```
ls -l simpleid2
```

11. Запустите `simpleid2` и `id`:

```
./simpleid2
```

```
id
```

Сравните результаты.

12. Прodelайте тоже самое относительно `SetGID`-бита.
13. Создайте программу `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;
```

```
}
```

14. Откомпилируйте её.

```
gcc readfile.c -o readfile
```

15. Смените владельца у файла `readfile.c` (или любого другого текстового файла в системе) и измените права так, чтобы только суперпользователь (`root`) мог прочитать его, а `guest` не мог.
16. Проверьте, что пользователь `guest` не может прочитать файл `readfile.c`.
17. Смените у программы `readfile` владельца и установите `SetUID`-бит.
18. Проверьте, может ли программа `readfile` прочитать файл `readfile.c`?
19. Проверьте, может ли программа `readfile` прочитать файл `/etc/shadow`? Отрадите полученный результат и ваши объяснения в отчёте.

5.3.2. Исследование Sticky-бита

1. Выясните, установлен ли атрибут Sticky на директории /tmp, для чего выполните команду
`ls -l / | grep tmp`
2. От имени пользователя guest создайте файл file01.txt в директории /tmp со словом test:
`echo "test" > /tmp/file01.txt`
3. Просмотрите атрибуты у только что созданного файла и разрешите чтение и запись для категории пользователей «все остальные»:
`ls -l /tmp/file01.txt`
`chmod o+rw /tmp/file01.txt`
`ls -l /tmp/file01.txt`
4. От пользователя guest2 (не являющегося владельцем) попробуйте прочитать файл /tmp/file01.txt:
`cat /tmp/file01.txt`
5. От пользователя guest2 попробуйте дозаписать в файл /tmp/file01.txt слово test2 командой
`echo "test2" > /tmp/file01.txt`
Удалось ли вам выполнить операцию?
6. Проверьте содержимое файла командой
`cat /tmp/file01.txt`
7. От пользователя guest2 попробуйте записать в файл /tmp/file01.txt слово test3, стерев при этом всю имеющуюся в файле информацию командой
`echo "test3" > /tmp/file01.txt`
Удалось ли вам выполнить операцию?
8. Проверьте содержимое файла командой
`cat /tmp/file01.txt`
9. От пользователя guest2 попробуйте удалить файл /tmp/file01.txt командой
`rm /tmp/file01.txt`
Удалось ли вам удалить файл?
10. Повысьте свои права до суперпользователя следующей командой
`su -`
и выполните после этого команду, снимающую атрибут t (Sticky-бит) с директории /tmp:
`chmod -t /tmp`
11. Покиньте режим суперпользователя командой
`exit`
12. От пользователя guest2 проверьте, что атрибута t у директории /tmp нет:
`ls -l / | grep tmp`
13. Повторите предыдущие шаги. Какие наблюдаются изменения?
14. Удалось ли вам удалить файл от имени пользователя, не являющегося его владельцем? Ваши наблюдения занесите в отчёт.

15. Повысьте свои права до суперпользователя и верните атрибут `t` на директорию `/tmp`:
- ```
su -
chmod +t /tmp
exit
```

## 5.4. Содержание отчёта

Отчёт должен включать:

1. титульный лист;
2. формулировку цели работы;
3. описание процесса выполнения задания. Для каждого действия, производимого в командной строке, в отчёт следует включить:
  - краткое описание действия;
  - вводимая команда или команды;
  - результаты выполнения команд (снимок экрана);
4. листинг программы;
5. выводы, согласованные с целью работы.