Информационная безопасность. Отчет по лабораторной работе № 5

Дискреционное разграничение прав в Linux. Исследование влияния дополнительных атрибутов

Горбунова Ярослава Михайловна

Содержание

[1 Цель работы 1](#_Toc115364619)

[2 Теоретическое введение 1](#_Toc115364620)

[2.1 Подготовка лабораторного стенда 1](#_Toc115364621)

[2.2 Компилирование программ 2](#_Toc115364622)

[3 Выполнение лабораторной работы 4](#_Toc115364623)

[3.1 Создание программы 4](#_Toc115364624)

[3.2 Исследование Sticky-бита 11](#_Toc115364625)

[4 Выводы 13](#_Toc115364626)

[5 Список литературы 14](#_Toc115364627)

# 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].

# 2 Теоретическое введение

## 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 (fig. 1)). Если же gcc не установлен, то его необходимо установить, например, командой

yum install gcc

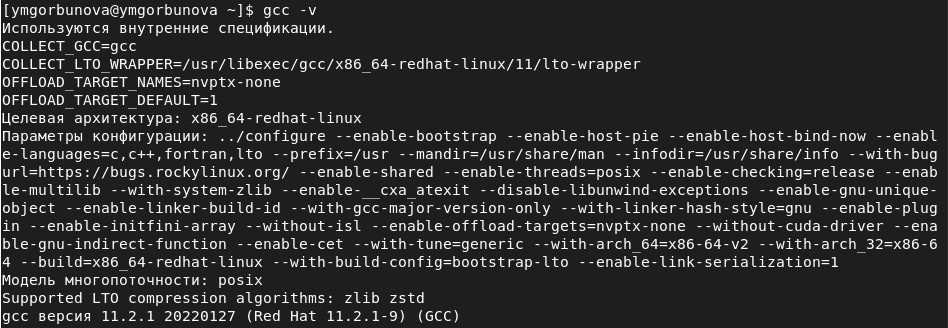


Figure 1: В системе установлен компилятор gcc

которая определит зависимости и установит следующие пакеты: gcc, cloogppl, срр, glibc-devel, glibc-headers, kernel-headers, libgomp, ppl, cloog-ppl, срр, 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 (fig. 2). В этой работе система SELinux рассматриваться не будет.

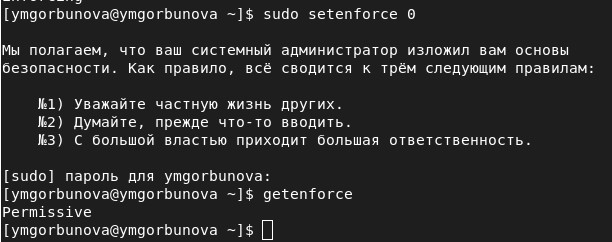


Figure 2: Отключение системы запретов до очередной перезагрузки системы

## 2.2 Компилирование программ

Для выполнения четвёртой части задания вам потребуются навыки программирования, а именно, умение компилировать простые программы, написанные на языке С (С++), 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-e 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 Compiller Collection, подробнее см. http://gcc.gnu.org). В неё входят компиляторы языков С, С++, Java, Objective-C, Fortran и Chill. Будем использовать лишь первые два.

Компилятор языка С называется gcc. Компилятор языка С++ 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.с

В случае успешного выполнения команды (отсутствие ошибок в коде) полученный объектный файл будет называться file.о.

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 С++, 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 С, 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 +х).

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 называние 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.

# 3 Выполнение лабораторной работы

## 3.1 Создание программы

1. Войдите в систему от имени пользователя guest.
2. Создайте программу simpleid.c (fig. 4, fig. 3):

* #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;  
    
   }



Figure 3: Создание программы. Программа simpleid.c

1. Скомплилируйте программу и убедитесь, что файл программы создан (fig. 4): gcc simpleid.c -o simpleid
2. Выполните программу simpleid (fig. 4): ./simpleid
3. Выполните системную программу id (fig. 4): id и сравните полученный вами результат с данными предыдущего пункта задания.

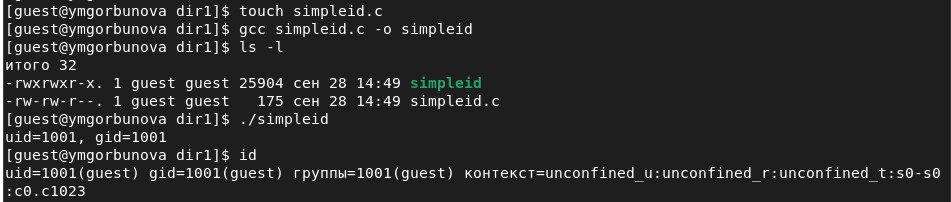


Figure 4: Создание программы. Пункты 2-5

1. Усложните программу, добавив вывод действительных идентификаторов (fig. 6, fig. 5):

* #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.

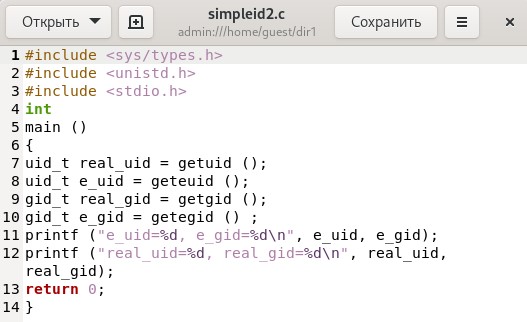


Figure 5: Создание программы. Программа simpleid2.c

1. Скомпилируйте и запустите simpleid2.c (fig. 6):

gcc simpleid2.c -o simpleid2

./simpleid2

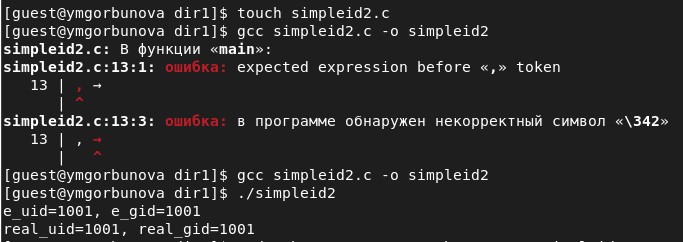


Figure 6: Создание программы. Пункты 6-7

1. От имени суперпользователя выполните команды (fig. 7):

chown root:guest /home/guest/simpleid2

chmod u+s /home/guest/simpleid2

1. Используйте sudo или повысьте временно свои права с помощью su. Поясните, что делают эти команды (fig. 7). – Первая команда меняет владельца файла на root, вторая - устанавливает UID-бит.
2. Выполните проверку правильности установки новых атрибутов и смены владельца файла simpleid2 (fig. 7): ls -l simpleid2

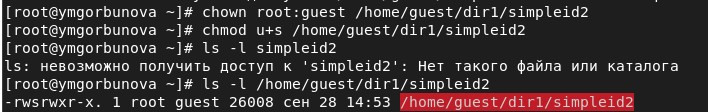


Figure 7: Создание программы. Пункты 8-10

1. Запустите simpleid2 и id (fig. 8):

./simpleid2

id

Сравните результаты. – Результаты полностью совпали.

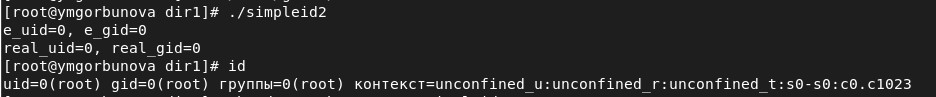


Figure 8: Создание программы. Пункт 11

1. Проделайте тоже самое относительно SetGID-бита (fig. 9).

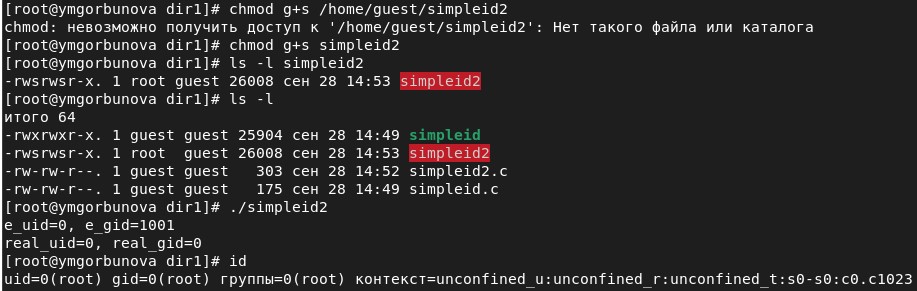


Figure 9: Создание программы. Пункт 12

1. Создайте программу readfile.c (fig. 10):

* #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;  
    
  }

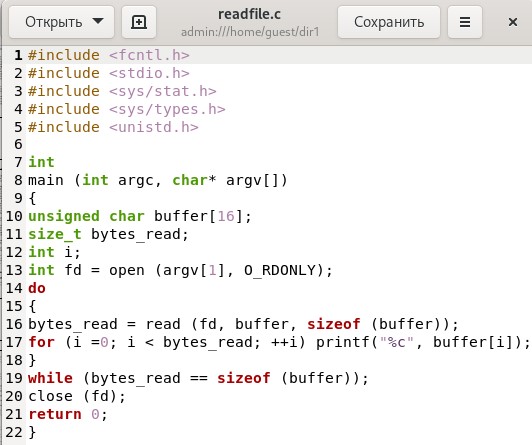


Figure 10: Создание программы. Пункт 13. Программа readfile.c

1. Откомпилируйте её (fig. 11). gcc readfile.c -o readfile

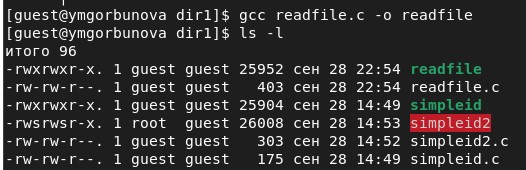


Figure 11: Создание программы. Пункт 14

1. Смените владельца у файла readfile.c (или любого другого текстового файла в системе) и измените права так, чтобы только суперпользователь (root) мог прочитать его, a guest не мог (fig. 12).

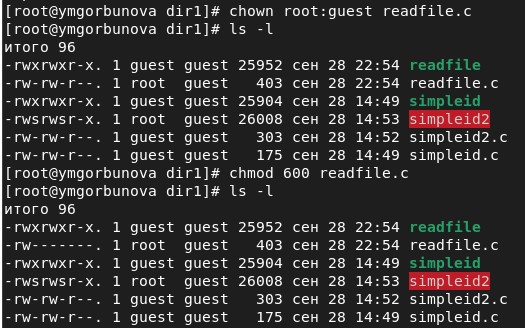


Figure 12: Создание программы. Пункт 15

1. Проверьте, что пользователь guest не может прочитать файл readfile.c (fig. 13).

Figure 13: Создание программы. Пункт 16

Figure 13: Создание программы. Пункт 16

1. Смените у программы readfile владельца и установите SetU’D-бит (fig. 14).

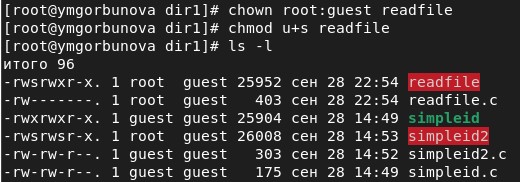


Figure 14: Создание программы. Пункт 17

1. Проверьте, может ли программа readfile прочитать файл readfile.c (fig. 15)? – Программа может прочитать файл.

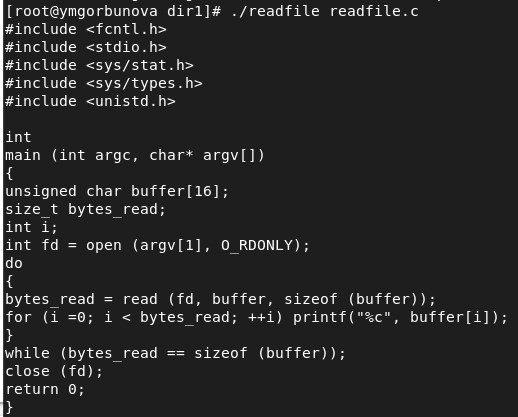


Figure 15: Создание программы. Пункт 18

1. Проверьте, может ли программа readfile прочитать файл /etc/shadow (fig. 16)? – Программа может прочитать файл. Отразите полученный результат и ваши объяснения в отчёте.

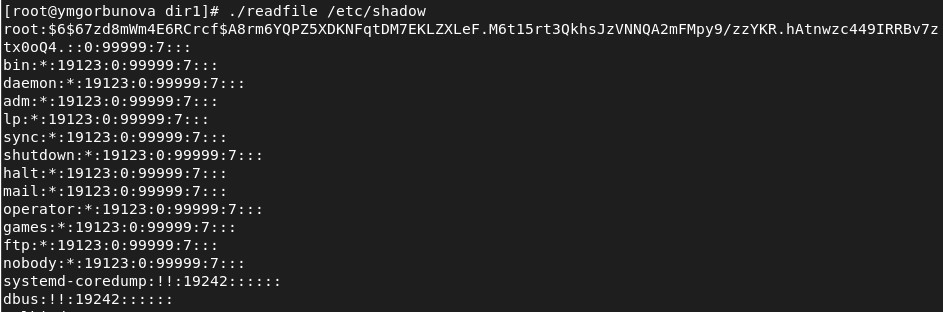


Figure 16: Создание программы. Пункт 19

## 3.2 Исследование Sticky-бита

1. Выясните, установлен ли атрибут Sticky на директории /tmp, для чего выполните команду (fig. 17) ls -l / | grep tmp
2. От имени пользователя guest создайте файл file01.txt в директории /tmp со словом test (fig. 17): echo “test” > /tmp/file01.txt
3. Просмотрите атрибуты у только что созданного файла и разрешите чтение и запись для категории пользователей «все остальные» (fig. 17):

ls -l /tmp/file01.txt

chmod o+rw /tmp/file01.txt

ls -l /tmp/file01.txt

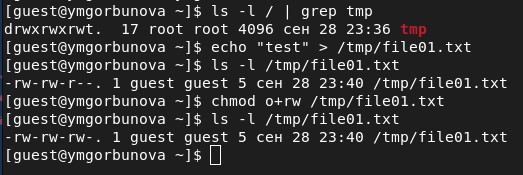


Figure 17: Исследование Sticky-бита. Пункты 1-3

1. От пользователя guest2 (не являющегося владельцем) попробуйте прочитать файл /tmp/file01.txt (fig. 18): cat /tmp/file01.txt
2. От пользователя guest2 попробуйте дозаписать в файл /tmp/file01.txt слово test2 командой (fig. 18)

echo “test2” >> /tmp/file01.txt

Удалось ли вам выполнить операцию? – Удалось.

1. Проверьте содержимое файла командой (fig. 18) cat /tmp/file01.txt
2. От пользователя guest2 попробуйте записать в файл /tmp/file01.txt слово test3, стерев при этом всю имеющуюся в файле информацию командой (fig. 18)

echo “test3” > /tmp/file01.txt

Удалось ли вам выполнить операцию? – Удалось.

1. Проверьте содержимое файла командой (fig. 18) cat /tmp/file01.txt

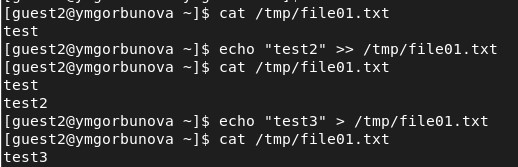


Figure 18: Исследование Sticky-бита. Пункты 4-8

1. От пользователя guest2 попробуйте удалить файл /tmp/file01.txt командой (fig. 19) rm /tmp/file01.txt

Удалось ли вам удалить файл? – Не удалось.

1. Повысьте свои права до суперпользователя следующей командой (fig. 19) su - и выполните после этого команду, снимающую атрибут t(Sticky-бит) с директории /tmp: chmod -t /tmp
2. Покиньте режим суперпользователя командой (fig. 19) exit
3. От пользователя guest2 проверьте, что атрибута t у директории /tmp нет (fig. 19): ls -l / | grep tmp
4. Повторите предыдущие шаги (fig. 19). Какие наблюдаются изменения? – Удается выполнить удаление файла.
5. Удалось ли вам удалить файл от имени пользователя, не являющегося его владельцем? Ваши наблюдения занесите в отчёт (fig. 19). – Удается выполнить удаление файла.

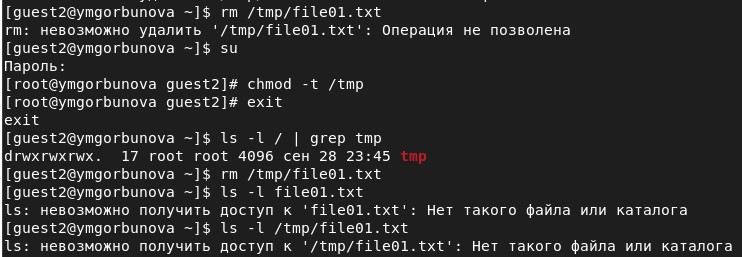


Figure 19: Исследование Sticky-бита. Пункты 9-14

1. Повысьте свои права до суперпользователя и верните атрибут t на директорию /tmp (fig. 20):

su -

chmod +t /tmp

exit

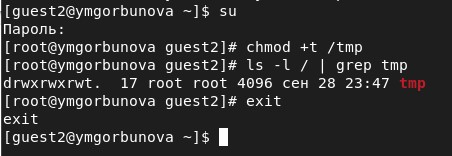


Figure 20: Исследование Sticky-бита. Пункт 15

# 4 Выводы

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

# 5 Список литературы

1. Методические материалы курса