Дискреционное разграничение прав в Linux. Исследование влияния дополнительных атрибутов

Лабораторная 5

Прокопьева М. Е

Российский университет дружбы народов, Москва, Россия

Информация

Докладчик

- Прокопьева Марина Евгеньевна
- студент
- Российский университет дружбы народов

Цель работы

Цель работы

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

Подготовка лабораторного стенда

Подготовка лабораторного стенда

Помимо прав администратора для выполнения части заданий потребу- ются средства разработки приложений. В частности, при подготовке стенда следует убедиться, что в системе установлен компилятор gcc (для этого, на- пример, 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 смонти- рована с опцией nosuid. Так как программы с установленным битом SetUID могут представлять большую брешь в системе безопасности, в современных системах исполь- зуются дополнительные механизмы защиты. Проследите, чтобы система защиты SELinux не мешала

Компилирование программ

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 вы- полняется работа, а является вспомогательной частью, позволяющей уви- деть, как реализуются на практике те или иные механизмы дискреционного разграничения доступа. Если при написании (или исправлении существу- ющих) 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 коллек- ции GNU-компиляторов, известной как GCC (GNU Compiller 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.

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 необхо- димо скомпоновать. Компоновать можно один или несколько файлов. В слу- чае использования хотя бы одного из файлов, написанных на C++, компо- новка 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 вызы- вает изнутри. Если же все файлы написаны на языке C, нужно использовать компилятор gcc. Например, так:

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,

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.

Выполнение лабораторной работы

Создание программы

1. Войдите в систему от имени пользователя guest.
2. Создайте программу simpleid.c:
3. Скомплилируйте программу и убедитесь, что файл программы создан:
4. Выполните программу simpleid:
5. Выполните системную программу id:

и сравните полученный вами результат с данными предыдущего пункта задания.

6. Усложните программу, добавив вывод действительных идентификато- ров
7. Скомпилируйте и запустите simpleid2.c:
8. От имени суперпользователя выполните команды: Используйте sudo или повысьте временно свои права с помощью su. Поясните, что делают эти команды.
9. Выполните проверку правильности установки новых атрибутов и смены владельца файла simpleid2: ls -l simpleid2
10. Запустите simpleid2 и id: ./simpleid2 id Сравните результаты.

12. Проделайте тоже самое относительно SetGID-бита.
13. Запустите simpleid2 и id:
14. Создайте программу readfile.c:
15. Откомпилируйте её.
16. 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 SetU’D-бит.
18. Проверьте, может ли программа readfile прочитать файл readfile.c?
19. Проверьте, может ли программа readfile прочитать файл /etc/shadow?
    Отразите полученный результат и ваши объяснения в отчёт

Исследование Sticky-бита

1. Выясните, установлен ли атрибут Sticky на директории /tmp, для чего выполните команду
2. От имени пользователя guest создайте файл file01.txt в директории /tmp со словом test:
3. Просмотрите атрибуты у только что созданного файла и разрешите чтение и запись для категории пользователей «все остальные»:
4. От пользователя guest2 (не являющегося владельцем) попробуйте прочитать файл /tmp/file01.txt:
5. От пользователя guest2 попробуйте дозаписать в файл

6. Проверьте содержимое файла командой
7. От пользователя guest2 попробуйте записать в файл /tmp/file01.txt слово test3, стерев при этом всю имеющуюся в файле информацию ко- мандой
8. Проверьте содержимое файла командой
9. От пользователя guest2 попробуйте удалить файл /tmp/file01.txt ко- мандой

10. Повысьте свои права до суперпользователя следующей командой su - и выполните после этого команду, снимающую атрибут t (Sticky-бит) с директории /tmp:

11. Покиньте режим суперпользователя командой exit

12. От пользователя guest2 проверьте, что атрибута t у директории /tmp нет:

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. Информационная безопасность компьютерных сетей 39

15. Повысьте свои права до суперпользователя и верните атрибут t на директорию /tmp:

Выводы

Выводы

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