Визитка

ПАВЕЛ ЗАКЛЯКОВ, ИТ-специалист

Дискреционное разграничение прав в Linux

Часть 3. Исследование влияния дополнительных атрибутов

В данной части исследуются механизмы изменения идентификаторов, применение SetUID- и Sticky-битов

В первых двух частях [23, 24] были рассмотрены основные теоретические сведения о дискреционной политике безопасности в современных ОС с открытым кодом, основные и расширенные атрибуты файлов, а также приведены задания для самостоятельного выполнения в целях закрепления на практике теоретических знаний. В данной, завершающей, части предлагается расширенно рассмотреть работу механизма смены идентификатора процессов пользователей, а также влияние бита Sticky на запись и удаление файлов. В работе сохранена сквозная нумерация заданий и списка литературы. Обещанное в прошлой части исследование расширенного атрибута s в данной части производиться не будет в силу ограничений редакции на размер публикаций.

Подготовка лабораторного стенда

Возьмём за основу стенд, описанный в [24]. Для проведения данной части работы необходима дополнительная 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 сделать это просто, например, командой:

```
$ yum install gcc
```

которая определит зависимости и установит следующие пакеты: gcc, cloog-ppl, cpp, glibc-devel, glibc-headers, kernel-headers, libgomp, ppl. Но лучше сразу ставить пакет gcc-c++, для установки которого требуются пакеты: 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. Если это так, то задания 42-50 выполнить не удастся.

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 командой:

```
# setenforce 0
```

После этого команда getenforce должна выводить Permissive. В этой работе система SELinux рассматриваться не будет.

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 выполняется работа, а является вспомогательной частью, позволяющей увидеть, как реализуются на практике те или иные механизмы дискреционного разграничения доступа. Если при написании (или исправление существующих) скриптов на bash-е у большинства системных администраторов не возникает проблем, то процесс компилирования, как показывает практика, вызывает необоснованные затруднения.

Использование «yum install ...», «aptget install ...», а то и графического интерфейса снижает планку требований к администраторам и приводит к тому, что многие очень мало знают о существовании исходных кодов и магической последовательности действий «./configure, make, make install», не говоря о написании собственных программ.

Замечание 1

Предполагаю, что здесь мы сталкиваемся с ещё одной проблемой сегодняшнего образования и становления специалистов. Исторически наша система образования готовила думающих людей, специалистов широкого профиля, без узкой привязки к конкретным задачам. Такие люди могли заниматься любой проблемой в рамках своей области, но на конкретизацию у них могло уходить какое-то время.

При нынешних ритмах жизни показалось не выгодным готовить специалистов n лет, а потом ещё и адаптировать их полгода-год под решение тех или иных проблем. Большим фирмам удобнее, чтобы специалист мог сразу работать по какой-то проблеме и был легкозаменим (одно из требований стандарта ISO 9001:2008). Исходя из выше сказанного можно предположить, что разделение специалистов на программистов-разработчиков и администраторов готовых систем идёт с Запада и навязано жизненными потребностями крупных фирм. Несомненно, мы к этому придём, но не следует забывать, что российская действительность зачастую сильно отличается от принятых «там» стандартов.

Мы не печатаем деньги в том объёме, как это делает ФРС, не имеем дешёвых кредитов, из-за чего средств постоянно не хватает, и на всём приходится экономить. Зачастую самоучки и «левши» становятся нашей гордостью, поэтому считаю, что плох тот администратор, который не может написать маленькой программы. Хотите быть лучше? Изучайте глубже ваши основные обязанности и больше узнавайте о смежных областях. В любом случае знания, полученные вами из последующих нескольких абзацев, не будут лишними.

В системах Linux доступен широкий диапазон языков программирования, многие из них свободно распространяются и общедоступны. Вот лишь короткий перечень используемых языков [26, стр 20]: Ada, C, C++, Eiffel, Forth, Fortran, Icon, Java, JacaScript, Lisp, Modula 2, Modula 3, Oberon, Objective C, Pascal, Perl, PostScript, Prolog, Python, Ruby, Smalltalk, PHP, Tcl/Tk, Bourne Shell и другие. Все эти языки можно разделить на две большие группы – компилируемые и интерпретируемые.

Рассмотрим первую группу. Для выполнения программ, написанных для неё, требуется компилятор, который считывает текст программы, написанной на одном языке – исходном, и транслирует (переводит) его в эквивалентный текст на другом языке – целевом [27, стр. 22]. В более привычном понимании компилятор превращает исходный текст программы, понятный человеку, в объектный код, исполняемый компьютером [3, стр. 19]. (Подробнее о процессе компиляции см. [28].)

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 C, C++, Java, Objective-C, Fortran и Chill [3, стр.19].

Будем использовать лишь первые два. Компилятор языка C называется gcc (файл /usr/bin/gcc, пакет gcc-4.4.2-7.fc12.i686). Компилятор языка C++ называется g++ и запускается с параметрами почти так же, как gcc (файл /usr/bin/g++, пакет gcc-c++-4.4.2-7.fc12.i686) [3, стр.20].

Проверить это можно следующими командами:

```
$ whereis gcc
gcc: /usr/lib/gcc /usr/libexec/gcc /usr/share/man/man1/gcc.1.gz
$ whereis g++
g++: /usr/bin/g++ /usr/share/man/man1/g++.1.gz
$ rpm -qf /usr/bin/gcc
gcc-4.4.2-7.fc12.i686
$ rpm -qf /usr/bin/g++
gcc-c++-4.4.2-7.fc12.i686
```

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 [3, стр.21].

Например, так:

```
$ gcc -o new_program file.o
```

В случае успешного выполнения команды будет создана программа new_program (исполняемый файл формата ELF с установленным атрибутом +x).

Замечание 2

Компилирование – это процесс, если не сказать искусство. Компилятор gcc (g++) имеет множество параметров, влияющих на процесс компиляции. Он поддерживает различные режимы оптимизации, выбор платформы назначения и пр. (подробнее о gcc см. [28]).

Также возможно использование make-файлов (Makefile) с помощью утилиты make для упрощения процесса компиляции.

У читателя, немного разбирающегося в компилировании, может возникнуть желание не разбивать процесс компиляции на стадии и, например, сразу использовать команду вроде gcc -o program program.c или g++ -o program program.cpp.

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.

Замечание 3

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. То есть 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

```
# ls -l /bin/bash
```

```
-rwxr-xr-x. 1 root root 927352 Май 21  2010 /bin/bash
```

/bin/bash является root:
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

```
#!/bin/bash
```

```
/usr/bin/id
/usr/bin/whoami
```

файл prog1.sh следующего содержания:
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.

Действительные и эффективные идентификаторы

В предыдущих частях [23, 24] мы рассмотрели различные атрибуты файлов и их влияние на работу в системе. При этом мы намеренно не рассматривали технические детали того, как эти атрибуты хранятся в системе и как осуществляется проверка, так как указанный процесс интуитивно понятен из-за своей простоты отношения субъектов и объектов.

К сожалению, в жизни не всё так просто и однозначно. Если взять файл на диске, то скорее всего вы будете иметь дело с объектом, а если этому файлу было передано управление, то это уже субъект, унаследовавший права того, кто его запустил.

Жизненная необходимость (например, пример, рассмотренный в [23], хранение учётных записей пользователей и механизм смены пароля) требует, чтобы наследование прав происходило не всегда или чтобы субъекты после запуска могли сами понижать свои права. (Известный принцип минимизации привилегий.)

Для того чтобы рассмотреть в подробностях, как это происходит, введём определения, взятые из [27]:

Действительный ID пользователя. UID пользователя, породившего процесс.

Эффективный ID пользователя. UID, использующийся при большинстве проверок прав доступа. В большинстве случаев эффективный и действительный UID являются одним и тем же. Эффективный UID может отличаться от действительного при запуске, если установлен бит setuid у файла исполняемой программы и файл не принадлежит пользователю, запускающему программу. (Подробнее рассмотрим ниже.)

Сохранённый set-user ID. Первоначальный эффективный UID при запуске программы (после выполнения exec). Имеет значение при проверке прав доступа, когда процессу нужно менять действительный и эффективный UID в ходе работы. Эта концепция пришла из System V.

Действительный ID группы. GID пользователя, создавшего процесс, аналогично действительному UID

Эффективный ID группы. GID, использующийся для проверка прав доступа, аналогично эффективному UID.

Сохранённый set-group ID. Первоначальный эффективный GID при запуске программы, аналогично сохранённому set-user ID.

Набор дополнительных групп. 4.2 BSD ввело понятие набора групп. Помимо действительного и эффективного GID, у каждого процесса есть некоторый набор дополнительных групп, которым он принадлежит в одно и то же время. Таким образом, когда проверка прав доступа осуществляется для группы файла, ядро проверяет не только эффективный GID, но также все GID в наборе групп.

Каждый процесс может получить все из этих значений. Обычный (не принадлежащий суперпользователю) процесс может переключать (менять, устанавливать) свои действительные и эффективные ID пользователя и группы. Процесс root (с эффективным UID, равным 0) может также устанавливать значения, как ему нужно (хотя это может оказаться односторонней операцией).

Получение и изменение идентификаторов пользователя и группы

Приведём группу системных вызовов из [30, стр. 307, man getuid, man getgid, man setreuid], позволяющих оперировать с введёнными выше понятиями (данными) на языке C. Информация по указанным функциям далее. Эти сведения понадобятся при написании небольших программ ниже, по ходу выполнения задания.

Имена функций:
getuid, geteuid – получить идентифкатор пользователя.
getgid, getegid – получить группу процесса.
setreuid, seteuid – установить действительный и/или фактический идентификатор пользователя.
Примеры использования:

```
#include <unistd.h>
#include <sys/types.h>

// getuid возвращает фактический идентификатор
// ID-пользователя в текущем процессе
uid_t getuid(void);

// geteuid возвращает эффективный идентификатор
// ID-пользователя в текущем процессе
uid_t geteuid(void);

// getgid возвращает действительный идентификатор группы
// текущего процесса
gid_t getgid(void);

// getegid возвращает эффективный идентификатор группы
// текущего процесса
gid_t getegid(void);

// Фактический ID соответствует ID-пользователя, который
// вызвал процесс. Эффективный ID соответствует
// установленному согласно setuid-биту на исполняемом файле.
// Действительный идентификатор соответствует
// идентификатору вызывающего процесса. Эффективный
// идентификатор соответствует установленному согласно биту
// setuid на исполняемом файле. Setreuid устанавливает
// действительный и фактический идентификаторы владельца
// текущего процесса. Непривилегированные пользователи
// могут изменять действительный идентификатор владельца
// на фактический и наоборот
int setreuid(uid_t ruid, uid_t euid);
int seteuid(uid_t euid);
```

Вопросы для самоконтроля

1. Как узнать идентификатор пользователя, от которого вы работаете?
2. Что такое действительный ID пользователя?
3. Что такое эффективный ID пользователя?
4. Могут ли действительный ID пользователя и эффективный ID пользователя отличаться?
5. Если ответ на п.4 «да», то при каких условиях?
6. Как из командной строки установить атрибут SetUID?
7. Как из командной строки установить атрибут SetGID?
8. Чем отличаются атрибуты SetUID и SetGID?
9. Как просмотреть, установлены SetUID- и/или и SetGID-атрибуты?
10. Возможно ли установить SetUID-атрибут у текстового файла, имеет ли это смысл?
11. Влияет ли SetUID-бит на чтение из файла?

Дополнительные вопросы:

12. Возможно ли установить SetUID-атрибут на директорию?
13. Каким образом влияет SetUID-атрибут, если он установлен на директории?
14. Возможно ли удаление файлов с установленным атрибутом SetUID?
15. Как Sticky-бит влияет на директории?

Дополнительные атрибуты

Исследуем влияние часто используемых дополнительных атрибутов.

Исследование механизма SetUID

Из статьи [24] выполните пункты 1 и 2:

1. Получите у руководителя занятия информацию об учётных записях пользователей и их паролях. Постарайтесь последовательно выполнить все последующие пункты, занося ваши ответы на поставленные вопросы и замечания в отчёт.

2. Войдите в систему от имени полученного пользователя (в нашем случае guest).

Продолжим:

36. Создайте программу simpleid.c, листинг 10.1 из [29]:

```
#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;
}
```

37. Скомплилируйте программу и убедитесь, что файл программы создан:

```
$ gcc -c simpleid.c
$ gcc -o simpleid simpleid.o
$ ls -l simpleid
```

```
-rwxrwxr-x. 1 guest guest 6992 Дек 17 19:36 simpleid
```

38. Выполните программу simpleid:

```
$ ./simpleid
```

```
uid=500 gid=500
```

39. Выполните системную программу id и сравните полученный вами результат с данными предыдущего пункта задания:

```
$ id
```

```
uid=500(guest) gid=500(guest) группы=500(guest),501(guest2)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c102
```

40. Усложните программу, добавив вывод действительных идентификаторов. Получившуюся программу назовите 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;
}
```

41. Скомпилируйте и запустите simpleid2.c:

```
$ gcc -c simpleid2.c
```

```
$ gcc -o simpleid2 simpleid2.o
$ ./simpleid2

e_uid=500 e_gid=500
real_uid=500 real_gid=500
```

42. От имени суперпользователя выполните команды:

```
# chown root:guest /home/guest/simpleid2
# chmod u+s /home/guest/simpleid2
```

Используйте sudo или повысьте временно свои права с помощью su-. Поясните, что делают эти команды.

43. Выполните проверку правильности установки новых атрибутов и смены владельца файла simpleid2:

```
$ ls -l simpleid2

-rwsrwxr-x. 1 root pasha 7313 Дек 17 19:46 simpleid2
```

44. Запустите simpleid2 и id, сравните результаты:

```
$ ./simpleid2

e_uid=0 e_gid=500
real_uid=500 real_gid=500

$ id

uid=500(guest) gid=500(guest) группы=500(guest),501(guest2)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

45. Повторите шаги 42-44 относительно SetGID-бита.

46. Возьмём за основу листинг Б.4 hexdump.c из [29], немного модифицируем (уберем лишнее) и скомпилируем его. Новую программу назовём 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;
}

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

Если мы хотим получить функционал программы близкий к hexdump.c, а не readfile.c, то есть выводить не только текстовые файлы, то замените в функции printf «%c» на «%2X».

47. 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.

48. Проверьте, что пользователь guest не может прочитать файл readfile.c.

49. Смените у программы readfile владельца и установите SetUID-бит. (Аналогично пункту 42 задания.)

50. Проверьте, может ли программа readfile прочитать файл readfile.c?

Проверьте, может ли программа readfile прочитать файл /etc/shadow?

Отразите полученный результат и ваши объяснения в отчёте.

Исследование Sticky-бита

51. Выясните, установлен ли атрибут Sticky на директории /tmp, для чего выполните команду:

```
$ ls -l /|grep tmp

drwxrwxrwt.  31 root root     4096 Дек 26 17:51 tmp
```

52. От имени пользователя guest создайте файл file01.txt в директории /tmp со словом test:

```
$ echo test>/tmp/file01.txt
```

53. Просмотрите атрибуты у только что созданного файла и разрешите чтение и запись для категории пользователей «все остальные»:

```
$ ls -l /tmp/file01.txt

-rw-rw-r--. 1 guest guest 5 Дек 26 19:40 /tmp/file01.txt

$ chmod o+rw /tmp/file01.txt
$ ls -l /tmp/file01.txt

-rw-rw-rw-. 1 guest guest 5 Дек 26 19:40 /tmp/file01.txt
```

54. От пользователя guest2 (не являющегося владельцем) попробуйте прочитать файл /tmp/file01.txt:

```
$ cat /tmp/file01.txt

test
```

55. От пользователя guest2 попробуйте дозаписать в файл /tmp/file01.txt слово test2 командой:

```
$ echo test2>>/tmp/file01.txt
```

Удалось ли вам выполнить операцию?

Проверьте содержимое файла командой:

```
$ cat /tmp/file01.txt
```

56. 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?

Проверьте содержимое файла командой:

```
$ cat /tmp/file01.txt
```

57. От пользователя guest2 попробуйте удалить файл /tmp/file01.txt командой:

```
$ rm /tmp/file01.txt
```

Удалось ли вам удалить файл?

Проверьте содержимое каталога /tmp командой:

```
ls
```

58. Повысьте свои права до суперпользователя следующей командой:

```
$ su -
```

и выполните после команду, снимающую атрибут t (Sticky-бит) с директории /tmp:

```
# chmod -t /tmp
```

покиньте режим суперпользователя командой:

```
# exit
```

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

```
$ ls -l /|grep tmp
drwxrwxrwx.  31 root root     4096 Дек 26 19:40 tmp
```

60. Повторите шаги 52-57. Какие наблюдаются изменения?

Удалось ли вам удалить файл от имени пользователя, не являющегося его владельцем?

Ваши наблюдения занесите в отчёт.

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

```
$ su -
# chmod +t /tmp
# exit
```

***

В данной части работы вы научились компилировать маленькие программы на C, исследовали механизм изменения идентификторов при использовании SetUID- и SetGID-битов, проверили работу Sticky-бита. EOF

1. Касперски К. Техника сетевых атак. – М.: Издательство «Солон-Р», 2001, ISBN 5-93455-078-0.
2. Тейнсли Д. Linux и UNIX: программирование в shell. Руководство разработчика/пер. с англ. – Издательская группа BHV, 2001, ISBN 966-522-085-7, ISBN 5-7315-0114-9.
3. Митчел М., Оулдем Д., Самьюэл А. Программирование для Linux. Профессиональный подход/ пер. с англ. – М.: Издательский дом «Вильямс», 2002, ISBN 5-8459-0243-6.
4. Доступный UNIX: Linux, FreeBSD, DragonFlyBSD, NetBSD, OpenBSD. – СПб: «БХВ-Петербург», 2006, ISBN 5-94157-876-8.
5. Пазизин С.В. Основы защиты информации в компьютерных системах: учебное пособие. – М.: Редакция «ОПиПМ», Научное издательство «ТВП», 2003, ISBN 5-85484-030-8.
6. Шаньгин В.Ф. Комплексная защита информации в корпоративных системах: учебное пособие.– М.: ИД «Форум»: ИНФРА-М, 2010, ISBN 978-5-8199-0411-4, ISBN 978-5-16-003746-2.
7. Filesystem Hierarchy Standard – домашняя страница проекта по иерархии файловых систем – http://www.pathname.com/fhs, последний стандарт 2.3 – http://www.pathname.com/fhs/pub/fhs-2.3.pdf.
8. Манн С., Митчелл Э.Л., Крелл М. Безопасность Linux/пер. с англ. – 2-е издание. – М.: Издательский дом «Вильямс», 2003, ISBN 5-8459-0485-4.
9. Бэндл Д. Защита и безопасность в сетях Linux. Для профессионалов (+CD). – СПб: «Питер», 2002, ISBN 5-318-00057-6.
10. Фликенгер Р. Взломы и настройка LINUX. 100 профессиональных советов и инструментов: практическое пособие. /пер. с англ. – М.: Издательство ЭКОМ, 2006, ISBN 5-7163-0121-5.
11. Мешков В. Архитектура файловой системы ext2. //«Системный администратор», №11, 2003 г. – С. 26-32 (http://www.samag.ru/cgi-bin/go.pl?q=articles;n=11.2003;a=04).
12. Костромин В. Права доступа к файлам и каталогам – http://www.linuxcenter.ru/lib/books/kostromin/gl_04_05.phtml, сентябрь 2010.
13. Frank Mayer, Karl MacMillan, David Caplan. SELinux by Example: Using Security Enhanced Linux – Publisher: Prentice Hall; Pub Date: July 27, 2006; ISBN-13: 978-0-13-196369-6.
14. Торчинский Ф. UNIX. Практическое пособие администратора. – СПб: «Символ-Плюс», 2003, ISBN 5-93286058-8.
15. In UNIX Everything is a File – http://ph7spot.com/musings/in-unix-everything-is-a-file, октябрь 2010.
16. Шрёдер К. Linux. Сборник рецептов. – СПб: «Питер», 2006, ISBN 5-46901188-7.
17. FAQ по FreeBSD 4.X, 5.X и 6.X – http://www.freebsd.org/doc/ru/books/faq/misc.html, октябрь 2010.
18. Колисниченко Д.Н. Linux-сервер своими руками. – СПб: «Наука и Техника», 2002, ISBN 5-94387-063-6.
19. Эбен М., Таймэн Б. FreeBSD. Platinum Edition. /пер. с англ.– СПб: ООО «ДиаСофтЮП», 2003, ISBN 5-93772-107-1.
20. Смит Р. Полный справочник по FreeBSD. /пер. с англ. – М.: Издательский дом «Вильямс», 2005, ISBN 5-8459-0576-1.
21. Федорчук А.В., Торн А.В. FreeBSD: установка, настройка, использование. – СПБ: «БХВ-Петербург», 2003, ISBN 5-94157-200-X.
22. Робачевский А.Н., Немнюгин С.А., Стесик О.Л. Операционная система UNIX. – 2-е изд. – СПб: «БХВ-Петербург», 2008.– 656 с. – С. 40-43, ISBN 978-5-94157-538-1.
23. Закляков П. Дискреционное разграничение прав в Linux. Часть 1. Основные теоретические сведения. //«Системный администратор», №11, 2010 г. – С. 98-103.
24. Закляков П. Дискреционное разграничение прав пользователей в Linux. Часть 2. Теоретические сведения. //«Системный администратор», №12, 2010 г. – С. 82-90.
25. Ахо А.В., Сети Р., Ульман Дж. Д. Компиляторы: принципы, технологии инструменты/ пер. с англ. – М.: Издательский дом «Вильямс», 2001, ISBN 5-8459-0189-8.
26. Мэтью Н. Основы программирования в Linux. /пер. с англ. /Н. Мэтью, Р. Стоунс. – 4-е изд., перераб. и доп. – СПб.: «БХВ-Петербург», 2009, ISBN 978-5-9775-0289-4.
27. Роббинс А. Linux: Программирование в примерах. /пер. с англ. – М.: «КУДИЦ-ОБРАЗ», 2005, ISBN 5-9579-0059-1.
28. Гриффитс Артур GCC. Полное руководство. /пер. с англ. /Артур Гриффитс. – К.: ООО «Тид «ДС», 2004, ISBN 966-7992-33-0.
29. Сайт книги Advanced Linux Programming (русское издание см. [2]) – http://www.advancedlinuxprogramming.com/, 2010. На сайте доступны английская PDF-версия (http://www.advancedlinuxprogramming.com/alp-folder) и коды программ – http://www.advancedlinuxprogramming.com/ALP-listings.tar.gz.
30. Рочкинд М. Программирование для UNIX. /пер. с англ. – 2-е изд. перераб. и доп.– М.: Издательско-торговый дом «Русская редакция»; СПб.: «БХВ-Петербург», 2005, ISBN 5-94157-749-4, ISBN 5-7502-0261-5.