Лабораторная работа №5
Работа с правами доступа, ACL

Введение

Из этой лабораторной работы Вы узнаете, чем отличается управление доступом в Windows и Linux, что такое модель UGO и как на самом деле работает алгоритм проверки прав доступа к файлам в Linux.

Правильная настройка прав доступа крайне важна для безопасной эксплуатации ИТ-инфраструктуры. Например, если всем пользователям Linux будет разрешено чтение файла `/etc/shadow`, в котором содержатся хеши паролей, то злоумышленник сможет взломать любой аккаунт, а если пользователям будет доступна запись на исполняемый файл какой-либо службы, то злоумышленник сможет выполнять в системе любые действия с повышенными привилегиями, ни в чем себе не отказывая.

Управление доступом через идентификаторы, ресурсы и разрешения

Система Linux разрабатывалась как многопользовательская операционная система, поэтому в ней изначально был реализован механизм дискреционного разграничения прав доступа. **Дискреционное** или избирательное управление доступом (*от англ. discretion – по усмотрению*) — это управление доступом **субъектов** безопасности к **ресурсам** на основе **разрешений**. Рассмотрим каждый из компонентов этой триады подробнее.

В операционной системе Linux мы работаем с такими **субъектами**, как пользователи и группы, причем так же, как в Windows, в состав локальных групп можно включать только учетные записи пользователей, а вложенные группы доступны только при работе компьютера в составе домена.

Как мы уже знаем, для того чтобы мы могли отличить один субъект от другого, пользователям назначаются идентификаторы пользователей (User Identifier, UID), а группам — идентификаторы групп (Group Identifier, GID), и оба вида POSIX-идентификаторов представляют собой целые числа в диапазоне от 0 до 2^{32} (4 294 967 296).

В части работы с идентификаторами субъектов в Linux много отличий от Windows. Например, в Windows идентификаторы пользователей и групп находятся в одном пространстве, а в Linux это два разных множества, поэтому пользователь root с идентификатором 0 и одноименная группа root с идентификатором 0 — это два

совершенно разных субъекта безопасности. Еще одной интересной особенностью является то, что один и тот же uid можно назначить нескольким пользователям, в то время как Windows строго следит за этим и не позволяет использовать один и тот же SID дважды. Но все эти моменты мы уже разбирали в предыдущей лабораторной работе.

Вторым элементом избирательного управления доступом являются **ресурсы**. В случае обычной файловой системы ресурсами являются файлы и папки, а в случае Linux, как мы знаем, файлы являются также интерфейсами для доступа к устройствам, ядру операционной системы, другим процессам и т. д. Метаинформация о файлах (такая как дата и время создания, количество жестких ссылок, владелец, группа и права доступа), как вы помните, хранится в структуре данных, которая называется айнодой. И эти айноды похожи на записи MFT с некоторыми отличиями.

И, наконец, **разрешения** определяют, какие действия могут быть выполнены субъектами по отношению к ресурсам. И в соответствии с базовой моделью безопасности в Linux доступны три разрешения — чтение (**R**ead), запись (**W**rite) и выполнение (e**X**ecute).

Значение разрешений rwx зависит от контекста:

1. Разрешения применительно к файлам (см. таблицу).

- **Чтение (r)** — дает право на чтение содержимого файла.
- **Запись (w)** — позволяет полностью переопределить содержимое файла или добавить новые данные в конец. При этом не обязательно иметь права на чтение файла.
- **Выполнение (x)** — актуально для бинарных исполняемых файлов. Если файл является скриптом sh или python, то для его выполнения интерпретатором достаточно, чтобы файл был доступен для чтения. Если же запускать скрипт напрямую по имени `./test.sh`, то вам потребуются права на выполнение.

Таблица. Пример выполнения команд с разными правами доступа на файл

Пользователь	**root**	**обыч.**	**обыч.**	**обыч.**	**обыч.**
Права на папку	**- - -**	**- - x**	**- - x**	**- - x**	**- - x**
Права на файл	**- - -**	**- - -**	**r - -**	**- w -**	**- - x**
`cat /folder/file.txt`	+	-	+	-	-
`echo "hello" > /folder/file.txt`	+	-	-	+	-
`echo "hello" >> /folder/file.txt`	+	-	-	+	-
`sh /folder/my.sh`	+	-	+	-	-

`/folder/cmd.bin (запуск)`	-	-	-	-	-

1. Значение разрешений применительно к папкам:

• **Чтение (r)** — дает право на чтение имен дочерних объектов, но вы не сможете обратиться к айнодам и прочитать их атрибуты.

• **Запись (w)** — имеет силу, только если есть доступ на выполнение. Расширяет этот доступ правами на создание новых и переименование/удаление существующих файлов и папок.

• **Выполнение (x)** — дает право на вход в папку (cd), чтение метаинформации по дочерним объектам и доступ к ним в соответствии с установленными правами доступа.

Другими словами, если у пользователя есть право на чтение файла, то для того чтобы он смог воспользоваться этим правом, ему нужны права на выполнение для всех вышестоящих каталогов.

Таблица. Пример выполнения команд с разными правами доступа на папку

Пользователь	**root**	**обыч.**	**обыч.**	**обыч.**	**обыч.**
Права на папку	**- - -**	**r - -**	**- w -**	**- - x**	**- w x**
Права на файл	**- - -**	**- - -**	**- - -**	**- - -**	**- - -**
`ls /folder/file.txt`	+	-	-	+	+
`ls /folder`	+	+	-	-	-
`cd /folder`	+	-	-	+	+
`stat /folder/file.txt`	+	-	-	+	+
`mkdir /folder/subfolder`	+	-	-	-	+
`mv /folder/file /folder/file2`	+	-	-	-	+
`touch /folder/newfile.txt`	+	-	-	-	+
`rm /folder/newfile.txt`	+	-	-	-	+

Разрешения rwx группируются по три бита в восьмеричное число от 0 до 7, причем биты следуют в обратном порядке:

• $2^0 = 1$ — Исполнение
• $2^1 = 2$ — Запись
• $2^2 = 4$ — Чтение

Полному доступу, например, соответствует число 7 (4 чтение + 2 запись + 1 исполнение), а на общие каталоги (обычно) полных прав не дают и урезают до 5 (4 чтение + 1 исполнение).

римечание

Лучшей практикой считается выдавать пользователям минимально возможные права, которых будет достаточно для выполнения ими должностных обязанностей. То же самое относится и к служебным приложениям: если на сервере работает Apache, то крайне желательно, чтобы эта служба работала не от суперпользователя, а из-под своей собственной учетной записи, например, www-data. Тогда в случае взлома веб-сервера злоумышленники будут ограничены правами учетной записи www-data, что значительно сократит количество дальнейших векторов атак.

Модель UGO «пользователь — группа — остальные»

Операционная система Linux унаследовала от UNIX простую модель безопасности «пользователь — группа — остальные» (от англ. User — Group — Others), в соответствии с которой у каждого объекта файловой системы есть три категории пользователей:

- **Пользователь** — это пользователь, который считается владельцем объекта. Владелец не только получает права, определенные для этой категории, но и может изменять права доступа на объект с помощью команды chmod (от англ. change mode - изменить режим).
- **Группа** — это группа пользователей, участникам которой можно назначить уникальные права доступа. Эта группа считается владельцем объекта, но, несмотря на это, участники группы не могут изменять права доступа к объекту. Изменять права может только пользователь-владелец и суперпользователь.
- **Остальные** — это все остальные пользователи, не вошедшие в две предыдущие категории.

римечание

Иногда эту модель называют «владелец — группа-владелец — остальные» (owner — group — others), но во избежание путаницы лучше придерживаться стандартной терминологии.

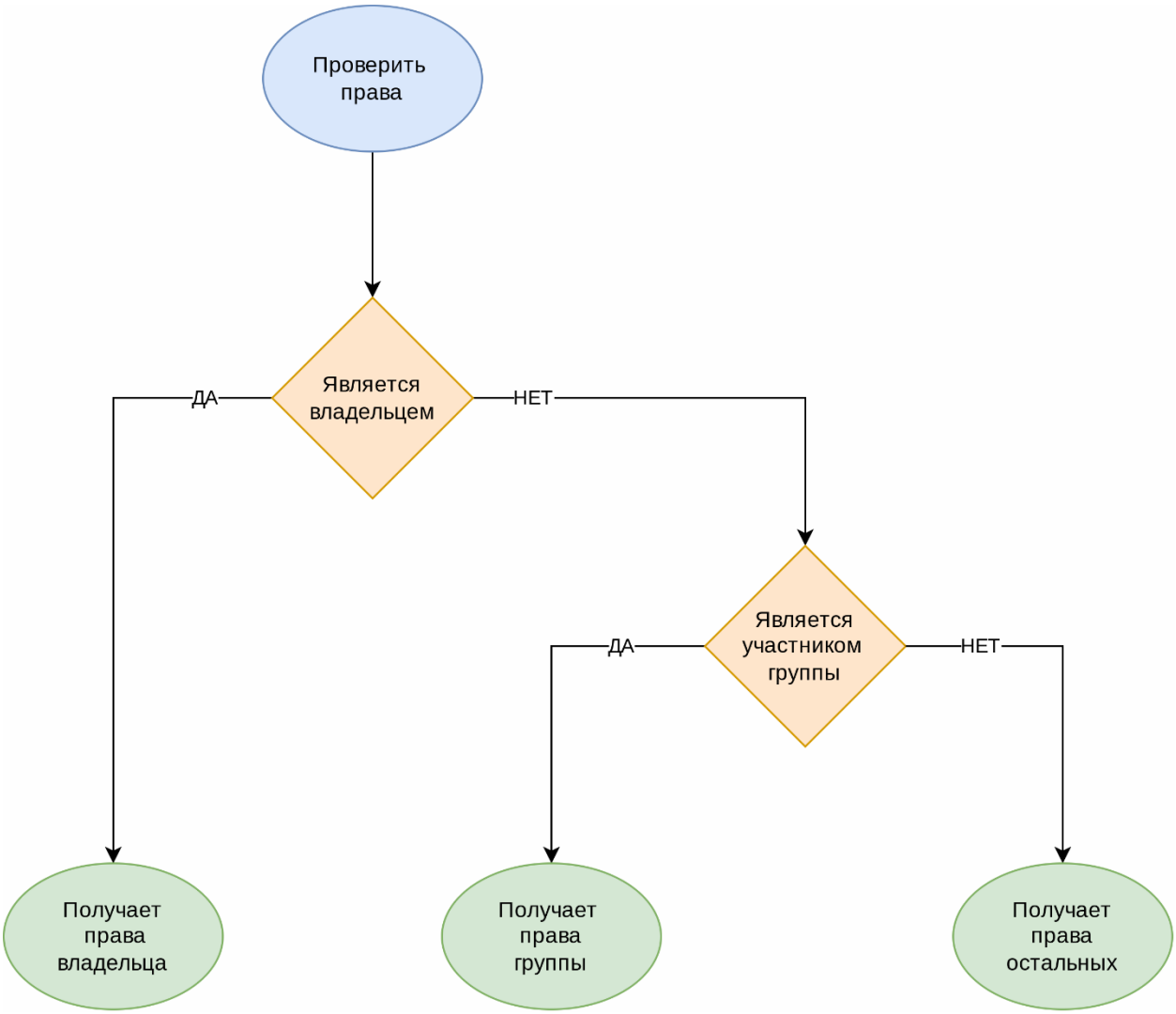
Сменить владельца можно с помощью утилиты `chown`, но привилегия CAP_CHOWN, позволяющая произвольно менять значения UID/GID для любых файлов, по умолчанию доступна только суперпользователю. Владелец файла с помощью `chown` может изменить только группу-владельца и только на ту группу, участником которой является сам. Если можно было бы устанавливать любую группу, то в сочетании с флагами sgid и execute это создавало бы большую угрозу безопасности.

Чтобы установить пользователя ivanov и группу drivers владельцами файла `test.txt`, используя привилегии суперпользователя:

```
sa@astra:~$ touch test.txt
sa@astra:~$ sudo chown ivanov:drivers test.txt
```

Информация о том, какой пользователь является владельцем и какой группе принадлежит файл, записана в айноде. Там же хранится информация о том, какие права доступа определены этим категориям пользователей.

Алгоритм определения прав доступа работает строго «слева направо», о чем (к величайшему сожалению) не догадывается подавляющее большинство администраторов Linux (см. рис).



Базовый алгоритм определения прав доступа к объекту файловой системы Linux

- Если пользователь является владельцем, он получает права пользователя и процедура проверки прав доступа завершается.

• Если пользователь не является владельцем, но входит в состав участников группы, то он получает права группы и процедура завершается.

• Если пользователь не является ни владельцем, ни участником группы, то он получает права, предназначенные для всех остальных пользователей.

Это означает, что пользователь не сможет прочитать файл, на который установлены права 007, если этот пользователь является владельцем этого файла. Владелец просто не попадает в категорию остальных пользователей, он сразу получает «ноль», и на этом проверка прав доступа завершается. Однако, являясь владельцем файла, пользователь всегда сможет поменять права и после этого получить доступ к содержимому файла.

Специальные флаги suid, sgid, sticky

Когда мы запускаем какую-нибудь утилиту (например, `cat`), операционная система создает процесс и передает ему соответствующие параметры. Посмотреть список процессов можно с помощью команды `ps -aux`.

Каждый процесс Linux запускается от имени конкретного пользователя и получает права доступа, соответствующие этому пользователю и группам, участником которых он является. Если процесс создает объекты файловой системы, то в айноде этих объектов в качестве владельца вписывается uid пользователя, запустившего процесс, а в качестве группы — gid первичной группы этого пользователя.

Кроме стандартных прав доступа в айноде хранятся три специальных флага SetUID, SetGID и Sticky bit, которые позволяют изменить описанную выше логику. Значение этих флагов зависит от контекста.

Значения специальных флагов к исполняемым файлам

SetUID (suid) на исполняемый файл:

Если установлен флаг suid на исполняемый файл, то при запуске приложения операционная система выполнит его от имени того пользователя, который является владельцем этого файла (т.е. подменит эффективный идентификатор пользователя на uid владельца исполняемого файла).

Такой фокус нам показывает, например, утилита `passwd` – напрямую у нас нет прав редактировать файл /etc/shadow, но через suid все невозможное становится возможным.

Продемонстрируем работу suid с помощью утилиты `id`:

Вначале определяем место расположения утилиты:

```
sa@astra:~$ which id
/usr/bin/id
```

Копируем утилиту в домашний каталог, меняем владельца и выставляем suid:

```
sa@astra:~$ cp /usr/bin/id ~/
sa@astra:~$ sudo chown root:root ~/id
sa@astra:~$ sudo chmod u=s,g=,o=x ~/id
```

Вызываем штатную утилиту

```
sa@astra:~$ id

uid=1000(sa) gid=1000(sa) группы=1000(sa),24(cdrom),25(floppy),29(audio),30(dip),44(video)
,46(plugdev),109(netdev),113(lpadmin),114(scanner),333(astra-console),1001(astra-admin)
```

Вызываем утилиту, для которой установлен suid, см. euid=0(root)

```
sa@astra:~$ ~/id

uid=1000(sa) gid=1000(sa) euid=0(root) группы=1000(sa),24(cdrom),25(floppy),29(audio),30(d
ip),44(video),46(plugdev),109(netdev),113(lpadmin),114(scanner),333(astra-console),1001(as
tra-admin)
```

SetGID (sgid) на исполняемый файл:

Если установлены флаги sgid и execute на исполняемый файл, то при запуске приложения операционная система заменит у процесса его эффективный gid на тот gid, который задан для файла, и добавит этот gid в список групп, участником которых является владелец процесса, поэтому процесс получит в том числе те права, которые будут назначены группе, являющейся владельцем исполняемого файла. Пожалуй, без полбита и не разберешься.

Рассмотрим на практическом примере, но обратите внимание на то, что флаг sgid не будет работать без прав на execute для группы, т.е. в выводе утилиты `ls` вы должны видеть строчную «s»:

Устанавливаем флаг sgid без execute, работать не будет, а прописная S указывает, что флаг выполнения не установлен:

```
sa@astra:~$ sudo chmod u=,g=s,o=x ~/id
sa@astra:~$ ls -l ~/id
------S--x 1 root root 48336 мар 22 00:00 /home/sa/id
```

Устанавливаем флаг sgid вместе с execute, теперь будет работать, и строчная буква s указывает, что флаг выполнения установлен:

```
sa@astra:~$ sudo chmod u=,g=sx,o=x ~/id
sa@astra:~$ ls -l ~/id
------s--x 1 root root 48336 мар  2 14:15 /home/admin/id
```

Вызываем штатную утилиту id для просмотра прав:

```
sa@astra:~$ id
uid=1000(sa) gid=1000(sa) группы=1000(sa),24(cdrom),25(floppy),29(audio),30(dip),44(video),
46(plugdev),109(netdev),113(lpadmin),114(scanner),333(astra-console),1001(astra-admin)
```

Вызываем утилиту, для которой установлены флаги:

```
sa@astra:~$ ~/id
uid=1000(sa) gid=1000(sa) группы=1000(sa),24(cdrom),25(floppy),29(audio),30(dip),44(video)
,46(plugdev),109(netdev),113(lpadmin),114(scanner),333(astra-console),1001(astra-admin)
```

Sticky bit на исполняемый файл:

Когда-то давно этот флаг позволял оставлять программу в памяти после завершения процесса для ускорения повторного запуска приложения, отсюда и его название «липкий бит». Но в современных Linux системах этот флаг более не имеет значения для исполняемых файлов и используется только для папок.

Значения специальных флагов к папкам

SetUID (suid) для папок:

Данный флаг на папку не имеет значения.

SetGID (sgid) для папок:

Если на папку установлен флаг sgid, то при создании объектов внутри этой папки в качестве группы-владельца им будет назначаться та же группа, которая является владельцем родительской папки.

```
sa@astra:~$ mkdir folder
sa@astra:~$ sudo chown root:root folder
sa@astra:~$ sudo chmod u=,g=s,o=rwx folder
sa@astra:~$ ls -l
...
d-----Srwx 2 root  root   4096 мар  2 14:21 folder

...
sa@astra:~$ touch folder/test.txt
sa@astra:~$ ls -l folder/test.txt
-rw-r--r-- 1 sa root 0 мар  2 14:22 folder/test.txt
```

Sticky bit для папок: — если на общую папку установлен липкий бит, то удалить или переименовать объекты внутри этой папки сможет только владелец этого объекта, (реальное применение папка /tmp).

```
sa@astra:~$ sudo mkdir folder
sa@astra:~$ sudo touch folder/test.txt

sa@astra:~$ sudo chmod u=,g=,o=rwxt folder/
sa@astra:~$ sudo chmod u=,g=,o=rwx folder/test.txt
sa@astra:~$ mv folder/test.txt folder/test2.txt
mv: невозможно переместить 'folder/test.txt' в 'folder/test2.txt': Операция не позволена

sa@astra:~$ sudo chmod u=,g=,o=rwx folder/
sa@astra:~$ mv folder/test.txt folder/test2.txt
sa@astra:~$ ls -l folder/
итого 0
-------rwx 1 sa root 0 мар  2 14:25 test2.txt

sa@astra:~$ sudo chmod u=,g=,o=rwxt folder/
sa@astra:~$ sudo chown sa:sa folder/test2.txt
sa@astra:~$ mv folder/test2.txt folder/test3.txt
sa@astra:~$ ls -l folder/
```

```
итого 0
-------rwx 1 sa sa 0 мар  2 14:25 test3.txt

root@astra:~# ls -la /tmp | head -2
итого 56
drwxrwxrwt 13 root              root              4096 мар  2 14:02 .
```

Несмотря на то, что специальные флаги назначают с помощью операторов +/-/= применительно к разным категориям пользователей, например, suid через u+s, sgid через g+s, а sticky через o+t, эти флаги так же, как разрешения rwx, группируются в отдельное восьмеричное число в следующем порядке:

- $2^0 = 1$ — Sticky
- $2^1 = 2$ — SetGID
- $2^2 = 4$ — SetUID

При назначении прав доступа через число специальные флаги задаются в первом разряде, поэтому для назначения липкого бита на папку с правами rw-r–r– нужно использовать число 1644:

```
sa@astra:~$ sudo chmod 1644 folder/
sa@astra:~$ ls -l
drw-r-Sr-T 2 root  root   4096 мар  2 14:29 folder
```

Понятие umask

Новые объекты файловой системы создаются с **правами по умолчанию**, значение которых зависит от типа создаваемого объекта: каталоги создаются с правами **777**, а обычные текстовые файлы с правами **666**. Однако, если вы проверите это на практике, то обнаружите, что у новых каталогов права 755, а у новых файлов 644, и происходит так по той простой причине, что на значение прав по умолчанию дополнительно **накладывается** так называемая **маска**.

Маска позволяет ограничить права доступа «сверху», указывая, какие биты следует сбрасывать. Если маска будет равна 000, то она не будет никак ограничивать права доступа к новым объектам. Если же выставить маску 777, то при создании новых объектов ни у кого не будет доступа к ним.

Чтобы получить текущее значение маски, вызовите утилиту `umask` с ключом -p или -S:

```
sa@astra:~$ umask -p
umask 0022
sa@astra:~$ umask -S
u=rwx,g=rx,o=rx
```

Обратите внимание, что утилита возвращает четырехразрядное число, но первый разряд не является маской для специальных флагов, маска так не работает. Ведущий ноль в этой строке просто придает числу формат восьмеричной константы в стиле языка

программирования Си. Утилиты оболочки bash всегда интерпретируют разрешения как числа в восьмеричной системе, поэтому ведущий ноль не является обязательным.

Чтобы установить значение маски 0022, мы можем использовать любую из следующих команд:

```
sa@astra:~$ umask 022
sa@astra:~$ umask 0022
sa@astra:~$ umask u=rwx,g=rwx,o=
```

Задавая маску в текстовом представлении, категории пользователей можно объединять или даже задавать маску сразу для всех категорий с помощью параметра a (all).

```
sa@astra:~$ umask ug=rwx,o=rx
sa@astra:~$ umask a=rwx
```

Также имеется возможность работы с отдельными битами. С помощью операторов +/- вы можете включить/отключить определенный бит маски, оставляя другие биты без изменений.

```
sa@astra:~$ umask ug-w
sa@astra:~$ umask a+w
```

Помимо прочего, вы можете комбинировать два предыдущих способа. Например, разрешить пользователю все операции, а группе и остальным пользователям убрать право на чтение.

```
sa@astra:~$ umask u=rwx,go-r
```

Утилита `umask` влияет на работу пользователя только в пределах одной сессии терминала. Для установления значения маски по умолчанию вы можете задать значение параметра UMASK в файлах `/etc/default/login` или `/etc/login.defs`.

римечание

Вам следует обратить внимание на различия между утилитами `umask` и `chmod`. Утилита `chmod` устанавливает права доступа на уже существующие объекты, а утилита `umask` влияет на то, с какими правами доступа будут создаваться новые объекты.

Атрибуты файлов

Помимо прав доступа, каждый из файлов стандартной файловой системы Linux имеет набор атрибутов, регламентирующих особенности работы с ним. Атрибуты поддерживаются такими файловыми системами Linux, как Ext4, Btrfs и XFS. Текущие установленные атрибуты можно посмотреть с помощью утилиты `lsattr`:

```
sa@astra:~$ touch test.txt
sa@astra:~$ lsattr test.txt
--------------e---- test.txt
```

Наиболее важными параметрами утилиты `lsattr` являются:

- `-R` — позволяет рекурсивно выводить атрибуты файлов в дереве папок;
- `-a` — позволяет выводить информацию и об атрибутах скрытых файлов;
- `-d` — позволяет выводить информацию об атрибутах папок вместо обработки их содержимого.

В том случае, если утилите не передаются имена файлов, она выводит информацию об атрибутах файлов из текущей папки.

Основные файловые атрибуты приведены в таблице:

Таблица. Файловые атрибуты

Атрибут	Значение
A	Запрещает обновлять метку времени доступа к файлу.
a	Автоматически устанавливает режим дополнения при открытии файла для записи.
C	Запрещает использовать механизм копирования при записи (Copy-on-Write) при модификации содержимого файла.
D	При применении к папке активирует режим синхронной записи изменений содержащихся в ней файлов.
d	Запрещает утилите `dump` создавать резервную копию файла.
E	Указывает на ошибку сжатия содержимого файла ядром ОС (не может быть установлен пользователем).
e	Указывает на использование экстентов для ссылок на соответствующие файлу дисковые блоки (не может быть установлен пользователем).
h	Указывает на то, что размер файла исчисляется в количестве блоков ФС, а не ее секторов, то есть размер файла превышал или превышает в данный момент 2 ТБ (не может быть установлен пользователем).
I	Указывает на то, что содержимое директории было проиндексировано утилитой `htree`.
i	Запрещает всем пользователям, в том числе супрепользователю, модифицировать файл, а именно: записывать в него данные, удалять, переименовывать или создавать ссылки на него. Например, если вы хотите запретить скриптам установки службы каталога вносить правки в файл `/etc/resolv.conf`, вы можете установить этот атрибут для указанного файла.
j	Принудительно активирует режим журналирования ФС при записи данных в файл.
s	Активирует механизм надежного удаления, автоматически записывающий нулевые блоки на диск после удаления файла пользователем.
S	Активирует режим синхронной записи изменений содержимого файла на диск.

T	При применении к папке указывает на то, что ее подпапки не связаны и могут размещаться в отдельных группах блоков.
t	Запрещает оптимизации использования блоков файла.
u	Запрещает удаление содержимого файла при его удалении из ФС с целью получения возможности его последующего восстановления.
X	Указывает на возможность прямого доступа к содержимому сжатого ядром ОС файла (не может быть установлен пользователем).
Z	Указывает на неактуальность сжатого ядром ОС файла (не может быть установлен пользователем).

Для изменения атрибутов используется команда `chattr`.

Примеры использования:

```
sa@astra:~$ sudo -i
root@astra:~# id
uid=0(root) gid=0(root) группы=0(root)

root@astra:~# touch test.txt
root@astra:~# chattr +i test.txt
root@astra:~# lsattr test.txt
----i---------e---- test.txt

root@astra:~# echo "qwe" > test.txt
bash: test.txt: Операция не позволена
root@astra:~# rm -f test.txt
rm: невозможно удалить 'test.txt': Операция не позволена
root@astra:~# touch test2.txt
root@astra:~# echo "Text 1" > test2.txt
root@astra:~# cat test2.txt
Text 1
root@astra:~# chattr +a test2.txt
root@astra:~# rm -f test2.txt
rm: невозможно удалить 'test2.txt': Операция не позволена
root@astra:~# echo "Text NEW" > test2.txt
bash: test2.txt: Операция не позволена
root@astra:~# echo "Text NEW" >> test2.txt
root@astra:~# cat test2.txt
Text 1
Text NEW
```

Списки доступа ACL

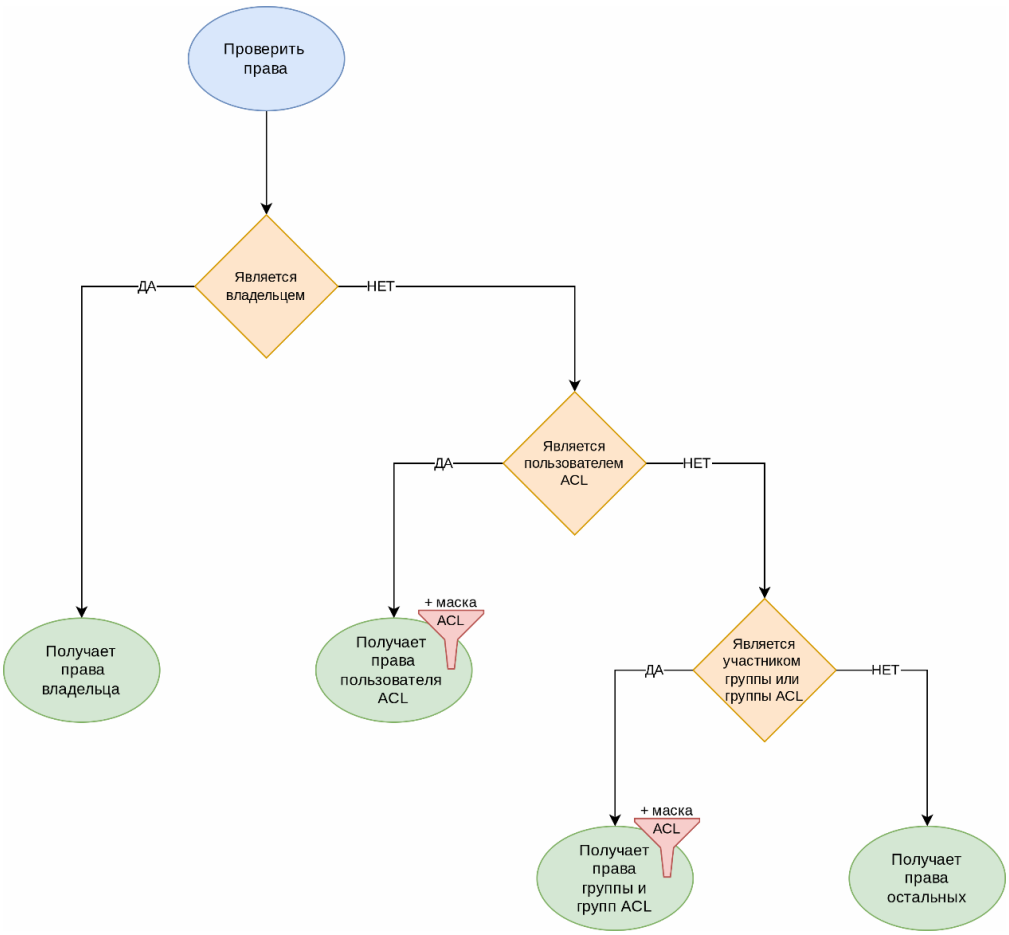
Базовая модель безопасности Linux «пользователь — группа — остальные» проста и эффективна, но не обладает достаточной гибкостью, чтобы с ее помощью эффективно решать задачи по организации совместного доступа сотрудников к файлам.

Например, если у вас есть группа IT сотрудников и группа бухгалтеров, то для совместной работы над счетами от поставщиков техники вам потребуется создать третью группу, куда следует поместить всех пользователей первых двух групп, т.к. локальные группы не могут быть участниками других групп. Внедрение домена частично решает эту задачу, т.к. появляется возможность использовать вложенные группы, но более простым и

эффективным решением является использование списков доступа (Access Control Lists, ACL).

Впервые списки доступа появились в 1994 году в файловой системе NTFS от Microsoft, что стало одним из факторов, который помог этой операционной системе занять лидирующие позиции в корпоративном сегменте рынка. В Linux этот механизм появился в 2002 году, поэтому отставания уже давно нет. В Astra Linux ядро скомпилировано уже с поддержкой POSIX ACL, поэтому данные функции доступны «из коробки» и ничего дополнительно делать не потребуется.

Если администраторов, разбирающихся в работе базового алгоритма проверки прав доступа Linux, просто мало, то специалистов, которые хорошо ориентируются в версии, усложненной POSIX ACL, не сыщешь днем с огнем. Отчасти так происходит еще и потому, что этот алгоритм практически нигде нормально не описан, поэтому давайте разбираться в деталях, (см. рис).



Алгоритм определения прав доступа к объекту файловой системы Linux с учетом ACL

- На первом шаге все то же самое. Если пользователь является владельцем, он получает права пользователя и процедура завершается.
- А со второго шага начинаются отличия. Если пользователь не является владельцем, но включен в ACL как пользователь, то он получает права, указанные в списке доступа с

учетом маски, и процедура завершается. Маска — это специальное правило POSIX ACL, которое определяет максимально возможные права доступа.

- Если пользователь не является владельцем и не включен в список ACL как пользователь, то проверка переходит на группы. Если пользователь является участником группы-владельца или одной из групп ACL, то он получает сумму этих прав с учетом маски и процедура завершается.
- Если пользователь не попал ни в одну из предыдущих категорий, то он получает права, предназначенные для всех остальных пользователей.

Для просмотра списков доступа предназначена утилита `getfacl` (Get File ACL), для редактирования этих списков предназначена утилита `setfacl` (Set File ACL). Продемонстрируем, как установка маски wx препятствует возможности прочитать файл.

```
sa@astra:~$ echo "hello" > test.txt
sa@astra:~$ sudo chown root:sa test.txt
sa@astra:~$ sudo chmod u=,g=r,o= test.txt
sa@astra:~$ sudo setfacl -m m::wx test.txt
sa@astra:~$ ls -l test.txt
-----wx---+ 1 root sa 6 мар  2 14:49 test.txt
```

Посмотрим расширенные права командой getfacl:

```
sa@astra:~$ getfacl test.txt
```

Результат вывода:

```
# file: test.txt
# owner: root
# group: sa
user::---
group::r--                      #effective:---
mask::-wx
other::---

sa@astra:~$ cat test.txt

cat: test.txt: Отказано в доступе
sa@astra:~$ sudo setfacl -m m::rwx test.txt
sa@astra:~$ cat test.txt
hello
```

При вызове утилиты `setfacl -m m::wx test.txt` мы использовали следующие параметры:

- `-m` (modify) — указывает, что мы хотим изменить записи ACL. Может использоваться ключ `-x` для удаления записи из ACL.
- `m:` (mask) — указывает, что мы хотим изменить маску. Возможны еще значения u/g для пользователя/группы, но в этом случае после двоеточия нужно будет задать имя субъекта.
- `wx` — задает значение маски.
- `test.txt` — имя файла.

Списки доступа ACL хранятся в расширенных атрибутах файла, поэтому при обычном копировании эта информация теряется:

```
sa@astra:~$ echo "hello" > test.txt
sa@astra:~$ setfacl -m m::rwx test.txt
sa@astra:~$ cp test.txt test2.txt
sa@astra:~$ getfacl test2.txt
```

Результат вывода:

```
# file: test2.txt
# owner: sa
# group: sa
user::---
group::r-x
other::---
```

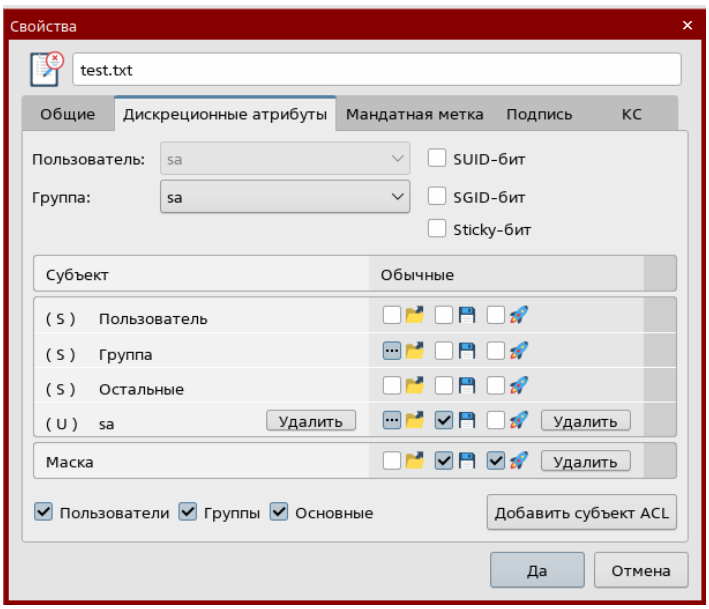
Для того чтобы скопировать файл с сохранением значений расширенных атрибутов, требуется установить ключ `--preserve`:

```
sa@astra:~$ cp --preserve=all test.txt test3.txt
sa@astra:~$ getfacl test3.txt
```

Результат вывода:

```
# file: test3.txt
# owner: sa
# group: sa
user::---
group::r--
mask::rwx
other::---
```

Следует отметить, что, в отличие от подавляющего большинства операционных систем, Astra Linux предоставляет файловый менеджер, с помощью которого можно из графики редактировать права доступа, включая расширенные POSIX ACL (см. рис.)



Редактирование прав доступа к файлам из графического интерфейса

Сравнение возможностей POSIX и Windows ACL

Списки доступа POSIX ACL решают ту же задачу, что и NTFS ACL, но отличаются технической реализацией, поэтому для получения похожего результата вам потребуется

вручную анализировать доступные параметры. Однако, если вам нужно максимально близко подойти к возможностям файлового сервера Windows, то POSIX ACL точно не хватит, и можно рекомендовать воспользоваться набором приложений Samba, в котором реализована поддержка подключаемых модулей виртуальной файловой системы (Virtual File System, VFS).

Служба Samba действует на сервере от имени суперпользователя, но позволяет эмулировать несколько моделей безопасности. По умолчанию Samba выполняет проверки в стиле POSIX ACL, но при подключении модулей VFS может приближать свое поведение к NTFS. Следует обратить внимание на следующие VFS-модули:

- **Модуль acl_xattr** — использует одновременно как базовые права доступа к файлам (включая списки доступа ACL), так и расширенные атрибуты для поддержки списков доступа Windows. Но ввиду некоторой несовместимости моделей безопасности Windows/Linux при использовании этого модуля вы будете сталкиваться с рядом ограничений. Например, какие-то изменения настроек прав доступа, выполненные из Windows, будут настойчиво отказываться сохраняться.
- **Модуль nfs4acl_xattr** — использует только расширенные атрибуты, игнорируя базовые права доступа Linux и списки доступа POSIX ACL. Преимуществом модуля является практически полная совместимость со списками доступа Windows, но при подключении к файловому серверу напрямую через ssh/xrdp/vnc пользователи не смогут получить доступ к файлам, т.к. Linux игнорирует значения расширенных атрибутов, и владелец у всех объектов будет root.

Практические задания

Задание 1. Права rw на текстовый файл.

1. Создайте пользователя user1, назначьте для него пароль и выполните вход (команды `useradd`, `passwd`, `login`).
2. Создайте файл в домашней директории пользователя `~/file.txt` с контентом «hello». Владельцем файла должен быть user1:user1, права доступа по умолчанию 644 (u=rw,g=r,o=r).
3. Назначьте на файл права доступа 000 (u=,g=,o=) и проверьте, что вы
   a. не можете прочитать файл
   b. не можете записать в файл новую строку «world»
4. Назначьте на файл права доступа 400 (u=r,g=,o=) и проверьте, что запись все еще недоступна, но чтение появилось.
5. Назначьте на файл права доступа 600 (u=rw,g=,o=) и проверьте, что вы можете теперь и читать, и писать в файл.

6. Назначьте на файл права доступа 006 (u=,g=,o=rw) и проверьте, что вы снова не можете ни читать, ни писать в файл.

Задание 2. Права x на исполняемом файле.

1. Скопируйте исполняемый файл cat в домашнюю директорию пользователя.
2. Выполните чтение из файла `file.txt` с помощью утилиты `cat` из домашней директории.
3. Назначьте на исполняемый файл права доступа 600 (u=rw,g=,o=) и проверьте, что вы не можете запускать эту утилиту для чтения файлов.
4. Назначьте на исполняемый файл права 100 (u=x,g=,o=) и проверьте, что права доступа на запуск утилиты снова появились.

Задание 3. Права rwx на каталог.

1. Создайте каталог `~/folder` и файл `~/folder/file.txt` с контентом «hello».
2. Назначьте на каталог права 000 (u=,g=,o=) и проверьте, что вы не можете более просматривать содержимое папки.
3. Назначьте на каталог права 400 (u=r,g=,o=) и проверьте, что теперь вы можете просмотреть список файлов, но без прав доступа к ним. Создание новых файлов все так же недоступно.
4. Назначьте на каталог права 600 (u=rw,g=,o=) и проверьте, что ничего не изменилось. Запись все так же недоступна.
5. Назначьте на каталог права 700 (u=rwx,g=,o=) и проверьте, что x дает возможность читать права на дочерние файлы, а в сочетании wx появилась возможность создания файлов.

Задание 4. ACL на пользователя.

1. Откройте еще одно окно терминала с правами root и создайте файл `~/file2.txt` с контентом «hello». Владельцем файла должен быть root:root, права доступа по умолчанию 644 (u=rw,g=r,o=r).
2. Назначьте на файл права доступа 007 (u=,g=,o=rwx).
3. Добавьте пользователя user1 в ACL файла с правами 4 (u:user1:r).

Убедитесь, что пользователь потерял права на запись, которые ему ранее предоставляла категория others, и теперь у него права только на чтение через ACL.

4. Установите на файл через ACL маску с правами 0 (m::).

Убедитесь, что права на чтение, которые ему ранее предоставляла запись в ACL, перекрываются маской, поэтому он «выпадает» из ACL, и ему снова выдаются права в соответствии с правами категории others.

Вопросы

1. Какой командой можно изменить владельца файла/папки?
2. Какой командой можно изменить права доступа на файл/папку?
3. Что дает право read на файл?
4. Что дает право write на файл?
5. Что дает право execute на файл?
6. Что дает право read на папку?
7. Что дает право write на папку?
8. Что дает право execute на папку?
9. Что дает право write вместе с execute на папку?
10. Если обычный пользователь является владельцем файла и на него установлены права 007, сможет ли пользователь прочитать файл командой cat?