Архитектура ЭВМ. Операционные системы

Власов Е.Е.

Аутентификация и права доступа в UNIX

Аутентификация (от греч. αὐθεντικός [authentikos] – реальный, подлинный; от αὐθέντης [authentes] – автор) – процедура проверка подлинности пользователя.

Права доступа - совокупность правил, регламентирующих порядок и условия доступа субъекта к объектам информационной системы, установленных правовыми документами или собственником, владельцем информации.

Права доступа определяют набор действий (например, чтение, запись, выполнение), разрешённых для выполнения субъектам (например, пользователям системы) над объектами данных. Для этого требуется некая система для предоставления субъектам различных прав доступа к объектам. Это система разграничения доступа субъектов к объектам, которая рассматривается в качестве главного средства защиты от несанкционированного доступа к информации или порче самой системы.

Субъект – процесс, взаимодействующий с объектами ОС.

Объект – любой ресурс ОС, к которому может обратиться субъект.

В UNIX роль номинального (зарегистрированного) субъекта играет пользователь. Каждому пользователю выдается (обычно - одно) входное имя (login name). Каждому входному имени соответствует единственное число, идентификатор пользователя (User IDentifier, UID). Это число и есть **ярлык субъекта**, которым система пользуется для определения прав доступа.

Каждый пользователь входит в одну или более групп.

Группа - это образование, которое имеет собственный идентификатор группы (Group IDentifier), объединяет нескольких пользователей системы, а стало быть, соответствует понятию множественный субъект. Значит, GID - это ярлык множественного субъекта, каковых у действительного субъекта может быть более одного. Список GID субъекта однозначно соответствует его **UID**.

Роль действительного (работающего с объектами) субъекта играет процесс. Каждый процесс снабжен единственным UID: это идентификатор запустившего процесс пользователя. Любой процесс, порожденный некоторым процессом, наследует его UID. Таким образом, все процессы, запускаемые по желанию пользователя, будут иметь его идентификатор. UID учитываются, например, когда один процесс посылает другому сигнал. В общем случае разрешается посылать сигналы "своим" процессам (тем, что имеют такой же UID). Классическое "Я тебя породил, я тебя и убью!" иллюстрируется еще и тем, что если процесс пытается убить родителя (послав ему тот или иной сигнал), то в лучшем случае ничего не происходит, чаще умирают оба, а иногда процесс, лишившийся родителя, превращается в зомби (его приходится убивать системе).

Роль объекта в UNIX играют многие реальные объекты, в частности представленные в файловой системе: файлы, каталоги, устройства, каналы и т. п. Каждый файл снабжен UID - идентификатором пользователя - **хозяина**. Вдобавок у файла есть **единственный** GID, определяющий группу, которой он принадлежит.

Виды доступа

На уровне файловой системы в UNIX определяется три вида доступа: чтение (**read, r**), запись (**write, w**) и исполнение (**execution, x**).

Право на чтение из файла дает доступ к содержащейся в нем информации, а право записи - возможность ее изменять.

При каждом файле имеется список того, что с ним может делать хозяин (если совпадает UID процесса и файла), член группы (если совпадает GID) и кто угодно (если ничего не совпадает). Такой список весьма невелик (три категории субъектов по три способа доступа итого 9 записей вида "можно/нельзя", в целом чуть больше одного байта). Установленное право того или иного доступа называется атрибутом (соответственно **r**, **w** или **x**).

Исполнение файла означает возможность **запустить** его на выполнение (обычно атрибут x выдается программам и командным сценариям); именно среди файлов, которые пользователю разрешено выполнять, shell ищет утилиту, с имени которой началась командная строка, а заставить выполниться файл, не имеющий атрибута **x** из командной строки, вообще невозможно. Право на чтение из каталога позволяет узнать только список файлов в нем (можно, например, написать cat каталог и увидеть малопонятную мешанину символов, в которой, однако, встретятся имена всех файлов в каталоге).

```
drwx--x--x 2 george staff 512 Dec 4 09:58 .
drwxr-xr-x 3 george staff 512 Dec 4 09:57 ..
-rw-r--r-- 1 george staff 135 Dec 4
```

Запись в каталог означает право изменять список имен файлов: переименовывать, создавать и удалять все, что в нем может содержаться. Это значит, что, имея право записи в каталог, пользователь (точнее, запущенный им процесс) может переименовать или удалить (!) оттуда любой файл, даже если ни писать в этот файл, ни читать из него он не может. В обратной ситуации, когда есть право записи в файл, а в содержащий его каталог - нет, пользователь сможет изменять содержимое и атрибуты, но не имя этого файла. Это вполне логичная схема, если учитывать, что атрибуты и содержимое файла суть свойства файлов, а атрибуты каталога и его содержимое - **имена** файлов - это свойство каталога.

Недостатки субъект-субъективной модели UNIX

За схемой прав доступа к файлам UNIX можно разглядеть механизм **доменной ответственности**, который тесно связан с О. Поскольку речь идет о правах, а не о **сеансах** доступа, мы имеем дело со **статической** моделью субъект-субъектных отношений со множественным субъектом. Множественный субъект в UNIX реализован не до конца. Дело в том, что при трех различных способах доступа мы имеем возможность задать объекту только **одну** группу. Это означает, что средствами chmod/chown **невозможно** создать такое положение вещей, когда одна группа пользователей могла бы только читать из файла, другая - только запускать его, а всем остальным файл вообще не был бы доступен.

Авторизация и аутентификация

Процесс определения того, имеет или не имеет некоторый субъект доступ к некоторому объекту, называется авторизацией. Выше описана статическая схема авторизации в UNIX, основанная на постоянных правах доступа. В статической схеме вопрос о доступе решается один раз, когда права задаются или изменяются. Во время работы пользователя достаточно четко поставить ему в соответствие некоторый номинальный субъект системы, чтобы заработал **механизм авторизации** и система автоматически отказывала в доступе или предоставляла его.

Динамическая авторизация - принятие решения о доступе при **каждом** запросе со стороны действительного субъекта - тоже имеет место в UNIX, хотя она строго не стандартизирована и больше зависит от состояния системы вообще и от характеристик некоторых иных объектов, нежели сам объект доступа, в частности. Право пользоваться входной телефонной линией может быть отнято у абонента при неуплате или перерасходе отведенного времени, для некоторых пользователей может быть ограничено время сеанса или право запускать определенные программы в определенное время (например, игры в рабочие часы), можно ограничить максимальный объем оперативной памяти и дискового пространства

Учетные записи

Все данные о пользователях UNIX хранит в файле /etc/passwd в текстовом виде. Каждому пользователю соответствует одна строка, поля которой разделяются двоеточиями

LOGNAME:*:UID:GID:GECOS:HOME:SHELL

Суперпользователь. Подмена идентификатора

Пользователь root (он же "суперпользователь") имеет нулевые UID и GID и играет роль **доверенного субъекта** UNIX. Это значит, что он не подчиняется законам, которые управляют правами доступа, и может по своему усмотрению эти права изменять. Большинство настроек системы доступны для записи только суперпользователю (даже если файл имеет права доступа 0444, root все равно может в него писать). Вообще, root - страшный человек! Он может удалить все ваши файлы, хотите вы того или нет. Он может отредактировать /etc/passwd и вообще может все. Как правило, пароль root знает **только** системный администратор. В полном согласии с **О**, он отвечает за все, что творится в системе, - раз уж он все это в состоянии изменить. Именно суперпользователю принадлежит файл /etc/group, который определяет, в какие еще группы, помимо отмеченных в /etc/passwd, входят пользователи системы.

Именно с нулевыми идентификаторами пользователя и группы запускается login: это позволяет ему в дальнейшем "становиться любым пользователем", меняя собственные UID и GID. Многие другие системные действия тоже требуют прав root, но по здравом рассуждении могут быть доверены обычному (не супер) пользователю. Например, управлять очередью отсылаемых электронных писем и передавать эти письма по назначению может процесс, не обладающий правами root, однако ему нужен полный доступ к очереди писем. С другой стороны, хорошо бы, чтобы никакой настоящий пользователь системы - человек не мог даже подглядеть в эту очередь. Так возникают псевдопользователи - учетные записи, к которым не подходит **никакой** пароль. Поле SHELL у псевдопользователя часто равно /sbin/nologin (программа выдает This account is currently not available и немедленно завершается), а поле HOME - /nonexistent (каталог, которого в системе нет). Зато система, запуская процесс "от имени" такого псевдопользователя, будет уверена, что ничего крамольного вне своей компетенции этот процесс не совершит даже в результате ошибки.

Пользователь может сам поменять себе пароль (а иногда SHELL и GECOS) с помощью команды passwd. Это простое и довольно обыденное действие, с учетом всего сказанного выше, **невозможно**. В самом деле: процесс, запущенный пользователем, будет иметь его UID, а файл passwd принадлежит root, и только процессам с нулевым UID доступен для записи. Значит, необходим механизм подмены идентификатора, позволяющий обычным пользователям запускать процессы "от имени" других, в частности от имени root .

Для этого в файловой системе предусмотрено еще два атрибута - setuid и setgid. При запуске файла, имеющего атрибут setuid, система создает процесс, UID которого равен UID **этого файла**, а не UID процесса-родителя. Такова программа passwd: запустив ее, пользователь получает права root, но его действия ограничены возможностями этой программы.

ls отображает setuid как s на месте пользовательского x-бита (x-бит никуда не делся, просто без него setuid все равно не имеет смысла). Сходным образом работает и setgid, наследуя GID процесса от выполняемого файла. Подмена GID нужна в тех случаях, когда необходимо и открыть доступ к файлу, и сохранить реальный идентификатор пользователя - например, для записи рекордов в игре rogue. Кстати, бессмысленно ставить setuid или setgid **сценариям**, поскольку процесс запустится из файла, содержащего **интерпретатор**, а файл со сценарием будет всего лишь параметром командной строки.

Подмена идентификатора, особенно на суперпользовательский (т. н. setuid root), - дело весьма деликатное. Что если программа passwd имеет какие-нибудь **еще** способности, кроме как изменять /etc/passwd строго в соответствии с документацией? Имея дело со свободно распространяемыми системами, мы всегда можем заглянуть в исходные тексты этой программы и убедиться, что авторы **не имели в виду** ничего предосудительного. Но вдруг у них **случайно** так вышло, что при определенных условиях passwd может запустить из текущего каталога программу с именем hack'em'all? Тогда все действия этой программы будут выполняться от имени root (наследование UID!) - действия, предусмотренные не системой, а каким-то бесправным пользователем, которому всего только и разрешено было что менять себе пароль.

Функции определения реального и эффективного UID

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

uid_t getuid(void);
uid_t geteuid(void);
```

`getuid` возвращает фактический идентификатор ID пользователя в текущем процессе. `geteuid` возвращает эффективный идентификатор ID пользователя в текущем процессе.

Фактический ID соответствует ID пользователя, который вызвал процесс. Эффективный ID соответствует установленному setuid биту на исполняемом файле.

Системные вызовы для изменения прав доступ для файлов

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

int chmod(const char *path, mode_t mode);
int fchmod(int fildes, mode_t mode);
```

Изменяет права доступа к файлу, заданному параметром *path* или файловым дескриптором *fildes*. Права задаются применением логической операции *OR* к константам.

Константы для функции chmod

S_ISUID	**04000**	**установить при выполнении идентификатор пользователя**
S_ISGID	02000	установить при выполнении идентификатор группы
S_ISVTX	01000	sticky бит
S_IRUSR (S_IREAD)	00400	владелец может читать
S_IWUSR (S_IWRITE)	00200	владелец может писать
S_IXUSR (S_IEXEC)	00100	владелец может выполнять файл или искать в каталоге
S_IRGRP	00040	группа-владелец может читать
S_IWGRP	00020	группа-владелец может писать
S_IXGRP	00010	группа-владелец может выполнять файл или искать в каталоге
S_IROTH	00004	все остальные могут читать
S_IWOTH	00002	все остальные могут писать
S_IXOTH	00001	все остальные могут выполнять файл или искать в каталоге

Получение информации о файле

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

int stat(const char *file_name, struct stat *buf);
int fstat(int filedes, struct stat *buf);
int lstat(const char *file_name, struct stat *buf);
```

Функции возвращают информацию об указанном файле. Для этого не требуется иметь права доступа к файлу, хотя потребуются права поиска во всех каталогах, указанных в полном имени файла.

`stat` возвращает информацию о файле *file_name* и заполняет буфер *buf*. `lstat` идентична `stat`, но в случае символьных сылок она возвращает информацию о самой ссылке, а не о файле, на который она указывает. `fstat` идентична `stat`, только возвращается информация об открытом файле, на который указывает *filedes* (возвращаемый **open**(2)), а не о *file_name*.

```
struct stat {
   dev_t st_dev; /* устройство */
   ino_t st_ino; /* inode */
   mode_t st_mode; /* режим доступа */
   nlink_t st_nlink; /* количество жестких ссылок */
   uid_t st_uid; /* идентификатор пользователя-владельца */
   gid_t st_gid; /* идентификатор группы-владельца */
   dev_t st_rdev; /* тип устройства (если это устройство) */
   off_t st_size; /* общий размер в байтах */
   blksize_t st_blksize; /* размер блока ввода-вывода в файловой системе */
   blkcnt_t st_blocks; /* количество выделенных блоков */
   time_t st_atime; /* время последнего доступа */
   time_t st_mtime; /* время последней модификации */
   time_t st_ctime; /* время последнего изменения */
 };
```

Флаги для поля *st_mode*

S_IFMT	0170000	битовая маска для полей типа файла
S_IFSOCK	0140000	сокет
S_IFLNK	0120000	символьная ссылка
S_IFREG	0100000	обычный файл
S_IFBLK	0060000	блочное устройство
S_IFDIR	0040000	каталог
S_IFCHR	0020000	символьное устройство
S_IFIFO	0010000	канал FIFO
S_ISUID	0004000	бит setuid
S_ISGID	0002000	бит setgid (смотри ниже)
S_ISVTX	0001000	бит принадлежности (смотри ниже)
S_IRWXU	00700	маска для прав доступа пользователя
S_IRUSR	00400	пользователь имеет право чтения
S_IWUSR	00200	пользователь имеет право записи
S_IXUSR	00100	пользователь имеет право выполнения
S_IRWXG	00070	маска для прав доступа группы
S_IRGRP	00040	Группа имеет права чтения
S_IWGRP	00020	Группа имеет права записи
S_IXGRP	00010	группа имеет права выполнения
S_IRWXO	00007	маска прав доступа всех прочих (не находящихся в группе)
S_IROTH	00004	все прочие имеют права чтения
S_IWOTH	00002	все прочие имеют права записи
S_IXOTH	00001	все прочие имеют права выполнения