УДК 004.451

Linux: о жизненном цикле процессов и разграничении доступа

А. М. Каннер
ЗАО «ОКБ САПР», Москва, Россия

Описан жизненный цикл процессов, и приведены особенности, возникающие при разработке средств разграничения доступа в операционную систему (ОС) Linux. Предложен способ "отслеживания" процессов ОС для однозначного и точного определения субъекта доступа в момент получения доступа к объектам ОС.

Ключевые слова: Linux, разграничение доступа, fork, exec, setuid.

Для корректной реализации системы защиты информации от несанкционированного доступа (СЗИ НСД) в Linux вне зависимости от реализуемой политики разграничения доступа (дискреционная, мандатная или любая другая) ко всему прочему, свойственному только операционной системе (ОС) на базе ядра Linux [1—4], необходимо для любого типа доступа к объектам ОС ответить на главный вопрос — какой именно субъект доступа в данный момент осуществляет доступ? На первый взгляд этот вопрос может показаться не столь сложным, однако тут есть множество нюансов.

Итак, субъектом доступа всегда является процесс ОС, который выполняется от имени некоторого субъекта доступа — пользователя ОС, которому соответствует некоторый уникальный идентификатор (UID, *User identifier*). У каждого процесса на уровне ядра ОС существует описывающая его структура *task_struct*, в которой содержатся такие значения, как реальный и эффективный UID (для версий ядра Linux >= 2.6.29 эти значения перенесены в структуру *cred* внутри структуры *task_struct*). Не вдаваясь в отличия этих двух UID-в, отметим только то, что в зависимости от их значений процесс выполняется от имени того или иного пользователя ОС.

Таким образом, в общем случае присутствие определенного пользователя в ОС (т. е. пользователя, прошедшего процедуру идентификации и аутентификации и имеющего незавершенные задачи или запущенный интерактивный шелл) означает наличие хотя бы одного процесса с UID, соответствующим UID-у этого пользователя.

Каннер Андрей Михайлович, программист группы программирования ядра СЗИ.
E-mail: kanner@okbsapr.ru

Статья поступила в редакцию 14 июня 2014 г.

Жизненный цикл процессов ОС

Какой UID имеют процессы изначально? Откуда возникают процессы с различными UID?

На самом раннем этапе загрузки ОС Linux, а именно, в момент загрузки ядра создается самый первый процесс — *swapper* или *sched* (так называемый процесс 0, процесс имеющий Process ID/PID = 0). Этот процесс, вообще говоря, нельзя считать реальным процессом, скорее, это функция ядра ОС, ответственная за организацию управления памятью и некоторые другие возможности. В ходе дальнейшей загрузки ядра ОС (*start_kernel()*) запускается, по сути, первый процесс пользовательского режима *init*, который имеет PID = 1. Этот процесс в дальнейшем является родителем всех процессов, запускаемых в ОС (PID-ы всех остальных процессов нефиксированы), в том числе и процессов реальных пользователей. Само собой по умолчанию и процесс *swapper/sched*, и *init* имеют UID = 0 (т. е. выполняются от имени пользователя root в ОС Linux).

Сами процессы в ходе своей жизни могут пребывать в следующих состояниях (рис. 1): состояние выполнения (running), состояние сна (uninterruptible/interruptible sleep), состояние остановки выполнения (stopped), zombie-состояние.

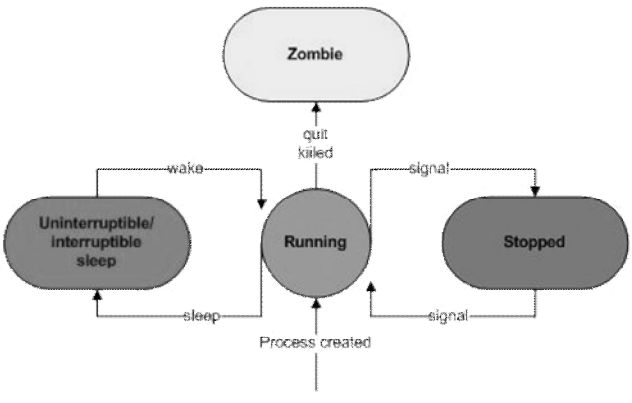

Рис. 1. Жизненный цикл процесса

При этом в процессе выполнения каждый процесс может создавать новые процессы (child process), по отношению к которым он будет предком-родителем, т. е. parent process (рис. 2). Эти новые процессы могут запускать на выполнение другие задачи (*fork()&exec()*) или выполняться дальше (*fork()*) с возможностью порождения других процессов в обоих случаях.

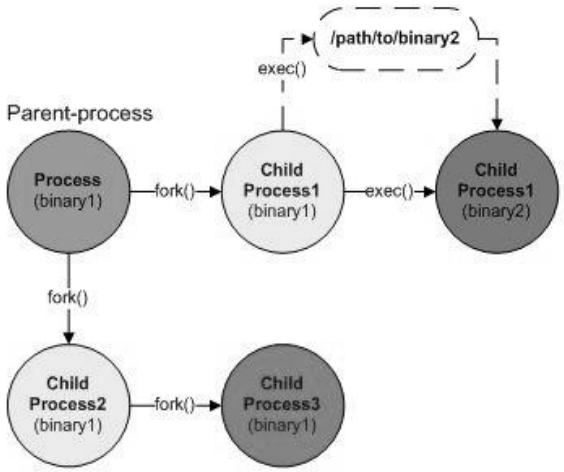

Рис. 2. Создание новых процессов (fork() и fork()&exec())

Таким образом, все процессы ОС можно представить в виде дерева, корнем которого является процесс *init*. Необходимо отметить, что при создании child process наследуют большинство параметров от своего предка-родителя, в том числе и UID. Однако далеко не все процессы даже во время загрузки ОС Linux будут постоянно иметь UID = 0, наследуя его от процесса *init*.

Существует возможность изменить текущий UID процесса с помощью системного вызова *setuid()*. Так, в ходе загрузки ОС Linux некоторые процессы будут иметь UID, соответствующий некоторым сервисам/демонам, для которых существуют учетные записи в ОС (например, apache, mysql и другие учетные записи, у которых обычно в /etc/passwd вместо реального шелла прописаны /sbin/nologin или /bin/false). Также в ходе штатной процедуры идентификации и аутентификации в ОС (login, gdm и т. п.) UID некоторого процесса, который в дальнейшем запустит шелл или сессию пользователя, будет заменен на UID реального пользователя.

Контроль создания и выполнения процессов в СЗИ НСД

Чтобы в любой момент времени иметь возможность однозначно ответить на вопрос "Какой именно субъект доступа осуществляет доступ?" и при этом реализовывать некоторое внешнее по отношению к ОС СЗИ НСД в ОКБ САПР в рамках продукта ПАК СЗИ НСД, "Аккорд-X" было решено реализовать следующее:

1. Начиная с момента загрузки СЗИ НСД, на раннем этапе загрузки ОС помечать все существующие и вновь создаваемые (во время операций *fork()* и/или *exec()*) процессы собственными дескрипторами безопасности. В таком случае в каждый момент времени в ОС не будет существовать процессов, которые не помечены каким-либо дескриптором безопасности.

2. Для всех уже существующих процессов ОС на момент загрузки ставить в соответствие дескрипторы с субъектом доступа типа *shadow* с именем *root* (это специальный субъект доступа, от которого выполняется процесс init и большинство процессов во время загрузки ОС).

3. Во время выполнения *setuid()* можно изменять субъект доступа в дескрипторе безопасности. При этом необходимо придерживаться определенных правил смены субъектов доступа, которые описаны ниже.

В продукте ПАК СЗИ НСД "Аккорд-X" на данный момент используется 2 типа субъектов доступа:

- *user* — реальные пользователи ОС, которые могут проходить процедуру идентификации и аутентификации;
- *shadow* — псевдопользователи, от имени которых могут выполняться определенные процессы в ОС, но которые не могут проходить процедуру идентификации и аутентификации (примером таких пользователей являются, например, различные сервисы/демоны).

В связи с этим логика условий и правил для смены субъекта доступа в рамках *setuid()* будет разной в зависимости от того, какой тип субъекта доступа соответствует текущему процессу.

- *Изменять субъект доступа типа user на другой субъект доступа типа user* можно только в случае успешного прохождения процедуры идентификации/аутентификации в СЗИ НСД одним из процессов-предков текущего процесса и при условии, что UID в *setuid()* совпадает с UID из процедуры идентификации/аутентификации. Так как в разных дистрибутивах ОС Linux процедуры идентификации/аутентификации и, собственно, смены UID-а могут призводить разные, даже достаточно далекие "в родстве", процессы необходимо вместе с дескриптором безопасности наследовать (при операциях *fork()* и *fork()&exec()*) от процесса-родителя, в том числе и информацию о пользователе, который осуществил успешную процедуру идентификации / аутентификации.

• *Изменять субъект доступа типа shadow после идентификации/аутентификации в СЗИ НСД*:

– *можно на субъект доступа типа user*, если UID в *setuid()* совпадает с UID из процедуры идентификации и аутентификации (такая замена является следствием обычного логина пользователя в ОС);

– *необходимо на другой субъект доступа типа shadow*, если текущий и заменяющий UID оба равны 0 и UID в *setuid()* не совпадает с UID из процедуры идентификации и аутентификации.

• *Изменять субъект доступа типа shadow без идентификации/аутентификации в СЗИ НСД*:

– *на субъект доступа типа user нельзя*;

– *можно на другой субъект доступа типа shadow*, если такой субъект доступа *shadow* существует и у текущего субъекта доступа есть разрешение делать *setuid()*.

В соответствии с приведенными правилами выполнения *setuid()* в системе при использовании СЗИ НСД должно присутствовать некоторое количество пользователей типа *shadow*. Для корректного создания таких пользователей *shadow* (сервисов, демонов и т. п.) необходимо предусмотреть режим работы СЗИ НСД, в котором все переключения на пользователей типа *shadow* будут фиксироваться и заноситься в журнал (так называемый лог работы СЗИ НСД в "мягком" режиме). В дальнейшем из журнала необходимо заносить таких пользователей типа *shadow* в список пользователей СЗИ НСД.

"Мягкий" режим работы является средством обучения СЗИ НСД, поэтому все субъекты доступа типа *shadow*, которые не были отображены в журнале и, соответственно, не были созданы в самом СЗИ НСД, не смогут корректно работать в обычном режиме работы с включенными политиками разграничения доступа. В связи с этим в "мягком" режиме работы СЗИ НСД желательно достаточно точно эмулировать работу пользователей ОС с целью создания как можно более точного соответствия/наличия в системе пользователей типа *shadow*. В противном случае при возникновении каких-либо ошибок придется повторно "дообучать" СЗИ НСД при дальнейшем использовании.

Как было отмечено выше, желательно для всех пользователей типа *shadow* завести параметры с разрешением/запрещением делать операции *setuid()* (так называемый setuid_ability) и делать *setuid()* на UID = 0 (так называемый setuid_root_ability). Эти параметры призваны исключить возможность эскалации привилегий и должны запрещать все действия *setiud()*, если выбранному субъекту доступа типа *shadow* в своей работе не требуется переключения на другие учетные записи (как правило, это требуется всего нескольким *shadow*, например, при входе пользователей для открытия /etc/passwd и т. п.).

Заключение

В заключении хотелось бы отметить, что описанный выше способ является не единственным способом для корректной «пометки» процессов соответствующими дескрипторами безопасности. Вторым, более простым вариантом можно считать аналогичную маркировку процессов в начале работы СЗИ НСД и при всех операциях *fork()* и/или *exec()*, но со сменой субъекта доступа на этапе успешного прохождения процедуры идентификации и аутентификации (т. е. по сигналу от специального PAM-модуля), без учета каких-либо событий *setuid()*. Этот способ является вполне корректным, но в меньшей степени соответствует реальному поведению процессов, так как, например, после прохождения идентификации и аутентификации процессы, помеченные дескриптором зашедшего пользователя, будут производить множество системных действий, на которые пользователю, вообще говоря, нельзя выдавать права доступа (доступ к /etc/passwd, различным сервисам, активирующимся при старте графической пользовательской сессии и т. д.).

Рассмотренный способ "отслеживания" процессов ОС является практически полноценной эмуляцией реального поведения процессов и не требует назначения пользователю каких-либо лишних прав в системе. Очень часто для защиты ОС используются внешние по отношению к ОС СЗИ НСД — такая эмуляция необходима, так как без нее нельзя с достаточной степенью уверенности сказать, что определенный доступ осуществил в действительности тот или иной пользователь, а не другой процесс/сервис/демон ОС.

Литература

1. *Каннер А. М.* Linux: о доверенной загрузке загрузчика ОС // Безопасность информационных технологий. — М., 2013. № 2. С. 41—46.

2. *Каннер А. М.* Linux: объекты контроля целостности // Информационная безопасность. Матер. XIII Междунар. конф. — Таганрог, 2013. Часть 1. С. 112—118.

3. *Каннер А. М.* Linux: к вопросу о построении системы защиты на основе абсолютных путей к объектам доступа // Информационная безопасность. Матер. XIII Междун. конф. — Таганрог, 2013. Часть 1. С. 118—121.

4. *Каннер А. М., Ухлинов Л. М.* Управление доступом в ОС GNU/Linux // Вопросы защиты информации. — М., 2012. № 3. С. 35—38.

Linux: process life cycle and access control

A. M. Kanner
OKB SAPR JSC, Moscow, Russia

Author describes life cycle of processes and singularities arising during development of tools for access control in Linux operating systems. Article provides method of "tracking" processes for unambiguous and precise definition of subject which gains access to objects in operating system.

Keywords: Linux, access control, fork, exec, setuid.

Bibliography — 4 references. *Received June 14, 2014*