Эссе по курсу "Защита информации", кафедра радиотехники, Московский физико-технический институт (ГУ МФТИ), http://www.re.mipt.ru/infsec

# RSBAC - система для построения модели безопасности в ОС Linux

**Бажин Алексей, 113 группа, 20.05.2005.**

RSBAC (Rule Set Based Access Control - контроль доступа на основе набора правил) - это система для построения модели безопасности в ОС Линукс. Она состоит из набора патчей к ядру линукс и утилит для ее администрирования. Она предназначена для гибкого контроля доступа к ресурсам компьютера. Ее правильное использование решает часть проблем связанных с безопасностью ОС Линукс.

Так как линукс был создан unix-подобной системой, то он унаследовал и ряд проблем безопасности связанных с доступом к ресурсам системы. В линукс доступ к системным ресурсам таким как память или файлы получают процессы. Для выполнения необходимых задач процессу даются права для их выполнения, как правило на основе пользователя с правами которого он запущен и его групп. Соответственно все процессы данного пользователя получают одинаковые права, которых больше чем необходимо для выполнения задачи одного из процессов. Таким образом если злоумышленник получает контроль над одним из процессов, то он получает и все права данного пользователя и может получить доступ к секретной информации пользователя. Это становится серьезной проблемой безопасности. Один из путей предотвращения таких ситуаций, это давать процессам ровно столько прав, сколько необходимо для успешного выполнения их задач. В стандартной линукс системе ядро решает может или нет процесс получить доступ к данному ресурсу (файлу, памяти, устройству, итд) основываясь на unix-правах доступа, являющимся частью файловой системы. Unix-права определяют доступ только для трех категорий пользователей(каждая категория получает одинаковый доступ): пользователя-владельца, определенной группы пользователей и “всех остальных”, и позволяют только задать права на чтение, запись и выполнение. Например если одному пользователю мы хотим дать права на запись, другому на чтение и третьему на выполнение, а остальным запретить доступ, то выполнить такое при помощи unix-прав мы не можем. Такая модель является причиной того что процесс имеет “слишком много” прав. Если нужно более четкое разделение прав, то мы вынуждены решать это на уровне приложений. Но такая возможность не всегда имеется (например при использовании чужих приложений с закрытым кодом). Или если программист допускает ошибку, то при определенных условиях злоумышленником может быть получен контроль над процессом. Также в системе имеется суперпользователь root, процессы которого имеют неограниченные права и могут делать в системе что угодно включая изменение системных данных чтобы скрыть свои действия. Очевидное решение – не давать процессам привилегий суперпользователя, но оно не может быть реализовано, так как доступ ко многим системным объектам может получать только суперпользователь. Например слушать на портах меньше 1024, а для http-демона необходимо слушать 80й порт. Конечно “правильный” демон сбросит привилегии сразу после необходимых действий, но это не обязательно и не может быть принуждено ядром. Также стандартная линукс система считает что пользователь владелец файла всегда имеет права дать другим пользователям доступ к своим файлам, что может увеличить риск раскрытия секретных данных. Таким образом получается что стандартная модель безопасности линукс(как в прочем и других unix-систем) имеет серьезные недостатки. Для их

решения был создан проект RSBAC.

В RSBAC системе существуют два администратора, первый – администратор системы (root), а второй - администратор безопасности, называемый офицером безопасности (по умолчанию uid 400), которые не могут влиять друг на друга. Системный администратор все еще будет заниматься администрированием системы: менять настройки, создавать/удалять пользователей итд. А офицер безопасности будет ограничивать всех пользователей, в том числе и системного администратора в доступе к ресурсам системы. Например можно разрешить root-у создавать/удалять пользователей при помощи определенных утилит, но запретить редактирование фалов passwd и shadow вручную. Так в чем же разница, если у нас в конечном итоге все равно есть пользователь, который может дать кому угодно, в том числе себе любые права на доступ к системе? - Разница в том, как можно получить эти права. Как уже было упомянуто, в стандартной линукс системе для многих задач требуются права суперпользователя root, соответственно процессы которым необходимы такие права их получают. Соответственно если скажем в suid-ном бинарнике, который общается с внешним миром есть уязвимость, то злоумышленник получит полный контроль над системой. А в RSBAC системе невозможно стать офицером безопасности через уязвимость в демоне(итд), так у них нет прав чтобы сменить uid на офицера безопасности, даже если у них есть привилегии суперпользователя root. В правильно сконфигурированной RSBAC системе можно стать офицером безопасности только залогинившись им (а в системе с особой безопасностью например только с локальной консоли). Разумеется надо сконфигурировать систему так чтобы root не смог никаким образом(setuid, сменой пароля, итд) стать офицером безопасности, иначе от RSBAC-а толку будет мало.

RSBAC имеет модульную структуру, и содержит несколько модулей, каждый из которых проверяет доступ своим способом и представляет собой свою модель безопасности. При помощи этих модулей можно создать необходимую модель безопасности, при этом можно одновременно использовать несколько модулей и финальное решение о доступе будет приниматься после опроса всех модулей. Доступ может быть предоставлен, только если все модули разрешили доступ. Любой модуль может запретить доступ, но не может разрешить доступ, запрещенный другим модулем. Все модули работают независимо, кроме модуля AUTH, который используется остальными модулями.

Модуль AUTH служит для контроля смены uid-ов процессами. Администрировать модуль можно при помощи утилит auth_set_cap (cli) или rsbac_menu (curses). Он имеет два параметра для каждого файла (разумеется меняют uid-ы запущенные процессы, а права на смену uid-а считываются с файла программы при запуске):

- auth_may_setuid – Процесс будет работать как при обычном ядре, то есть может менять uid на любой. Можно разрешить например для /bin/login.
- auth_capabilities – Список uid-ов на которые процесс может поменять.

Следующий по простоте модуль – FF (File Flags). Он позволяет выставлять для файлов (и директорий и fifo) дополнительные атрибуты. Модуль позволяет выставлять следующие атрибуты: no_execute, secure_delete, write_only, search_only, execute_only, read_only, no_delete_or_rename, add_inherited, append_only. Администрирование модуля производится при помощи утилит attr_set_fd или rsbac_menu. Интересные атрибуты: secure_delete – заполнять файл нулями при удалении, add_inherited – наследование атрибутов FF от вышестоящего объекта (то есть например можно сделать директорию ”только для чтения” с несколькими изменяемыми файлами).

Модуль RC (Role Compatibility). Модель реализуемая этим модулем напоминает то как мыслят люди, например “оператору разрешено только читать журнальные файлы”. Здесь оператор представляет собой “роль”. Модель RC также использует концепцию “ролей”. “Роль” можно рассматривать как название набора задач которые пользователь может выполнять (например webmaster, operator). “Роль” ассоциируется с доступом к определенным объектам. Объекты категоризируются “типами”, которым могут быть даны имена (например logfiles, webdocuments). “Роли” даются права на доступ к определенным “типам” (например operator имеет доступ к logfiles). Какой доступ имеет “роль” к “типу” также можно задать (например operator имеет доступ к logfiles, но может только читать их). Для того чтобы получить “роль” субъект (тот кто получает доступ к объекту)(процесс) должен быть авторизован. Каждому субъекту (процессу) дается начальная “роль”, и она наследуется любым субъектом порожденным данным. В линукс субъектом является процесс и его права наследуются потомками. Например при залогинивании процессу login пользователя joe дается начальная роль operator и теперь все процессы порожденные login-ом joe наследуют роль operator. Модель RC позволяет субъекту менять свою роль, для этого он должен выполнить специальный системный вызов или запустить программу с “forced role” (роль которую приобретет файл при запуске вне зависимости от того какая роль была первоначально у субъекта делающего запуск. Это аналогично suid для обычных unix-прав). Но изменить роль можно, только если текущая роль позволяет такую смену роли. (Например можно сделать так, что можно сменить роль с A на B, а наоборот нельзя). Между uid-ом и ролью нет прямой связи, например процесс того же пользователя joe может сменить uid на mary и при этом у него сохранится роль operator. В модели RC в качестве объектов могут быть FD (фалы и директории), IPC (средства межпроцессного взаимодействия: семафоры, очереди, сообщения, разделяемая память, ...), DEV (устройства), SCD (Системные данные: порты, журнал ядра, ...), PROCESS (процессы). Виды (типы) доступа совпадают с ACL правами (будут рассмотрены далее), например CHDIR, CREATE, DELETE, итд. То есть например есть вид доступа CHDIR (можно ли сменить директорию на...), а есть ACL право (можно ли со мной это сделать). Для администрирования ролей используются утилиты rc_copy_role, rc_get_item, rc_set_item или rsbac_menu.

Далее рассмотрим модуль ACL. Каждый файл имеет unix-права (wrxwrxwrx), но как это было сказано ранее они не являются достаточно гибкими. Мы хотим выставлять права для каждого пользователя, например для user1 wr- , для user2 w-x и для group1 --x. Так что нам необходимы списки контроля доступа - access control lists (ACL). Для этого и был добавлен модуль ACL, расширяющий стандартные unix-права. В качестве прав можно назначать:

- ADD_TO_KERNEL – для модулей ядра загружать в ядро
- ALTER – изменять контрольную информацию для IPC
- APPEND_OPEN – открывать для добавления
- CHANGE_GROUP – изменять группу
- CHANGE_OWNER – изменять владельца
- CHDIR – сменить директорию
- CLONE – вызывать clone() или fork() (создать процесс/тред)
- CLOSE - закрывать
- CREATE - создавать
- DELETE - удалять
- EXECUTE – выполнять (запускать)
- GET_PERMISSIONS_DATA – прочитать unix-права

- GET_STATUS_DATA – выполнять stat() (получать информацию о стандартных атрибутах файла)
- LINK_HARD – делать жесткую ссылку
- MODIFY_ACCESS_DATA – изменять информацию о времени последнего доступа к файлу
- MODIFY_ATTRIBUTE – изменять RSBAC атрибуты
- MODIFY_PERMISSIONS_DATA – изменять unix-права
- MODIFY_SYSTEM_DATA – изменять системные данные.
- MOUNT – совершить монтирование
- READ - читать
- READ_ATTRIBUTE – прочитать RSBAC атрибуты
- READ_OPEN – открывать файл для чтения
- READ_WRITE_OPEN – открывать файл для записи
- REMOVE_FROM_KERNEL – выгружать модуль из ядра
- RENAME - переименовывать файл
- SEARCH – производить поиск
- SEND_SIGNAL – посылать сигнал другому процессу
- SHUTDOWN – выключать/перезагружать систему
- SWITCH_LOG – изменять параметры журналирования RSBAC
- SWITCH_MODULE включать/выключать RSBAC модули
- TERMINATE – завершать процесс
- TRACE – вызывать ptrace() (трассировать(дебагать) процесс)
- TRUNCATE – усекать файл
- UMOUNT – выполнять размонтирование
- WRITE – писать
- WRITE_OPEN – открывать файл для записи

Если нет ACL вхождения для субъекта к объекту, то права наследуются от родительского объекта. Наследственность может быть ограничена масками наследственности. Наверху иерархии наследования находится значение ACL по умолчанию для каждого типа объекта. Для изменения значения ACL объекта нам необходимы специальные ACL права для этого объекта. Специальные права Forward позволяют дать права, которыми мы обладаем, кому-нибудь еще, но мы не сможете их в последствии отменить. Специальные права Supervisor включают в себя все остальные права и никогда не могут быть замаскированы и могут быть выставлены только пользователями, которые уже их имеют. Эти права выставлены по умолчанию для офицера безопасности. Управление группами позволяет каждому пользователю определять глобальные или частные группы без ограничений. Глобальные группы могут быть использованы любыми пользователями, а частные только владельцем группы. Также владелец группы единственный, кто может добавлять или удалять пользователей из группы или менять ее тип и владельца.

Модуль MAC (Mandatory Access Controls) реализует модель предложенную Беллом и Ла Падулом для ОС Multics. Немного теории: Существуют два множества: множество субъектов (тех кто получает доступ, в линукс - процессы) и множество объектов (к которым получают доступ – файлы, директории, ipc, ...). Система безопасности проверяет права субъекта на доступ к данному объекту и разрешает или запрещает доступ. В MAC критерием для принятия решения о доступе является “мандатная метка”. Она состоит из уровня доступа (неотрицательно число) и набора категорий M (множество из 64 элементов, 64-битный вектор). По определению категория M1 доминирует над категорией M2, если множество M1 включает в себя

множество M2. Например для трехбитных множеств M1 {1,0,1} доминирует над M2 {1,0,0} и M3 {0,0,1} доминирует над M4 {0,0,0}. Метка Label1 {Li, Mi} доминирует над меткой {Lj, Mj} если Li>Lj и Mi доминирует над Mj. Для принятия решения о доступе имеется матрица доступа D, где Dij содержит права доступа субъекта Si к объекту Oj, в RSBAC для этой матрицы используются стандартные unix-права(wrx). Доступ (Si,Oj,x) предоставляется если:

1. ss-property: Si доминирует над Oj если x = r или x = w. То есть считается, что если мандатная метка субъекта доминирует над мандатной меткой объекта, то субъект имеет право на чтение или запись к объекту.
2. *-property: Si является доверенным либо

- Oj доминирует над Si, если режим = a
- уровень Oj равен Si, если режим = w
- Si доминирует над Oj, если режим = r

   То есть чтобы запретить перетекание данных из более секретных объектов в менее секретные доступ на добавление в объект дается если объект доминирует над субъектом, доступ на запись дается если уровни доступа объекта и субъекта равны, и доступ на чтение дается если субъект доминирует над объектом.

3. ds-property: x есть в ячейке Dij матрицы доступа D. То есть если в матрице доступа есть запрошенный доступ субъекта к объекту.

Из описания понятно, что эта модель сложно применима, но в некоторых случаях без нее не обойтись...

Коротко о назначении других модулей: UM (User Management) – дополняет или полностью заменяет подсистему управления пользователями и группами в Линукс. PAX (Pageexec) – содержит некоторые меры (разделение памяти на участки для кода и данных, рандомизация адреса стека и mmap адресов) против стандартных методов атак на программы (переполнение буфера, итд). DAZ (Dazuko) – защита от чтения или запуска известных вредоносных программ (троянов, руткитов). JAIL – усиление безопасности и увеличение функциональности chroot. RES (Linux Resources) – позволяет задавать лимиты на системные ресурсы (память, время процессора, итд).

### *Литература:*


1. Документация проекта RSBAC http://rsbac.org/documentation
2. The RSBAC introduction http://books.rsbac.org/unstable/book-0.html
3. RSBAC для начинающих http://linux.ru.net/index.php?module=library&action=show&docid=1&part=2
4. Serious Security for Linux http://www.linuxsecurity.com/content/view/117460/49/