«СА» – 20 лет: наши темы, наши авторы

ISSN 1813-5579 22009

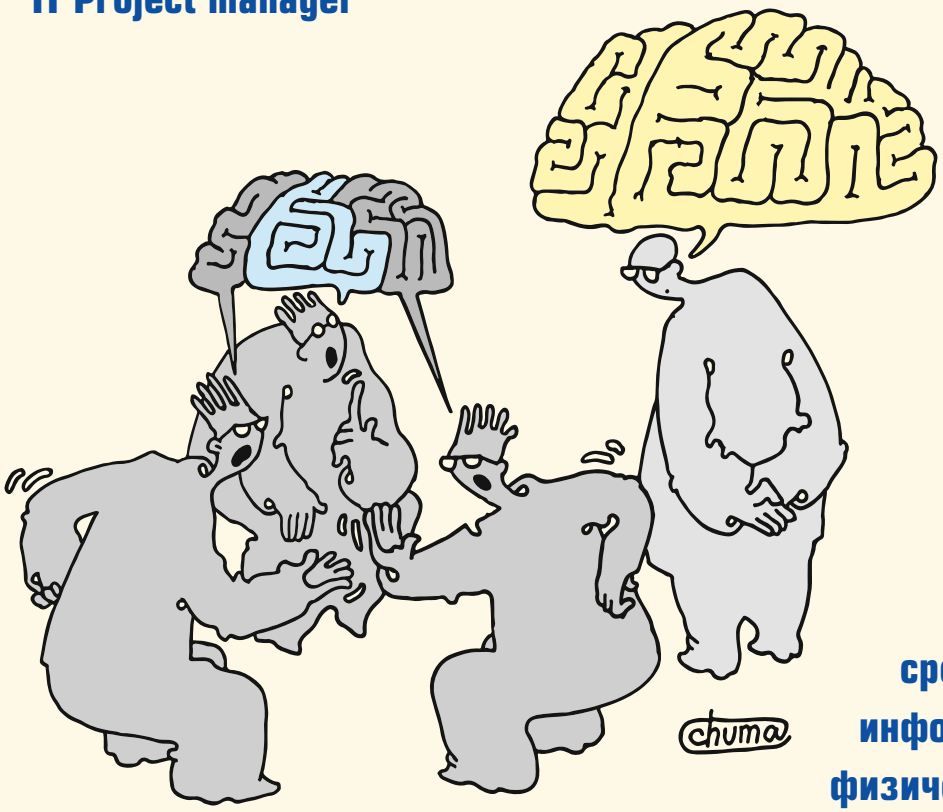
9 771813 557005

Системный администратор

ежемесячный журнал www.samag.ru

№9(238)
2022

Вакансия:
Руководитель ИТ-проектов/
IT Project manager



SELinux –
система принудительного контроля доступа

Резервная копия Asterisk FreePBX 15
и создание Ring Group

Умные вещи века

Программирование
средства поддержки расчетов информационных характеристик физических систем на Python и Qt

Образование без Майкрософта?
Заочный круглый стол

Наука и технологии | Наука и технологии | Наука и технологии 16+

Автоматизированный анализ лексического состава художественных текстов с использованием стилометрии

Варианты построения цифровых моделей местности на базе различных геоинформационных систем и технологий

Электронный паспорт как элемент цифрового профиля гражданина

Визитка

СЕРГЕЙ ГОЛОВАШОВ,
руководитель центра компетенций DevOPS/DevSecOPS

Визитка

ИВАН АГАТИЙ, кандидат психологических наук, магистр юриспруденции, главный специалист-эксперт ФСТЭК России

SELinux – система принудительного контроля доступа

В статье рассказано про возможности работы операционной системы SE Linux в режимах полной защищенности, про мандатные доступы и систему контроля на базе ядра, используемую как в западных, так и в российских дистрибутивах, таких как – Ред ОС и Астра Линукс

Security-Enhanced Linux (SELinux) – это новый метод контроля доступа в Linux на основе модуля ядраLinux Security (LSM). SELinux включен по умолчанию во многих дистрибутивах на основе Red Hat, использующих пакетную базу rpm, например, Fedora, CentOS и т д.

LSM (англ. Linux Security Modules – модули безопасности Linux) представляют собой реализацию в виде подгружаемых модулей ядра. В первую очередь, LSM применяются для поддержки контроля доступа. Сами по себе LSM не обеспечивают систему какой-то дополнительной безопасностью, а лишь являются неким интерфейсом для её поддержки. Система LSM обеспечивает реализацию функций перехватчиков, которые хранятся в структуре политик безопасности, охватывающей основные операции, защиту которых необходимо обеспечить. Контроль доступа в систему осуществляется благодаря настроенным политикам.

Мы рассмотрим настройки SELinux, а также постараемся подойти к системе с другой стороны, посмотреть, чем она может быть полезна обычному пользователю Linux, рассмотрим основы её работы, включение, отключение и изменение состояний, а также рассмотрим методы и подходы в создании своих политик. В качестве системы для выполнения примеров использовалась CentOS 8.

В большинстве операционных систем имеются средства управления доступом, которые определяют, может ли определенный объект (пользователь или программа) получить доступ к определенному ресурсу. В системах UNIX применяется разграничительный контроль доступа (discretionary access control, DAC). Этот метод позволяет ограничить доступ к объектам на основе групп, к которым они принадлежат.

Например, в GNU/Linux для каждого файла определены владелец, группа, а также указаны права доступа к этому файлу. Правами доступа определяется, кто может получить доступ к файлу, кто может открыть его для чтения, кто может внести в него изменения, кто может запустить этот файл на выполнение. Права доступа определены для трех категорий: пользователь (владелец файла), группа (все пользователи, которые являются членами группы) и другие (все пользователи, которые не являются ни владельцем файла, ни членами группы).

Такое разграничение прав доступа может привести к возникновению ряда проблем из-за того, что программа, в которой может быть обнаружена уязвимость, наследует все права доступа пользователя. Следовательно, она может выполнять действия с тем же уровнем привилегий, какой есть у пользователя (что нежелательно).

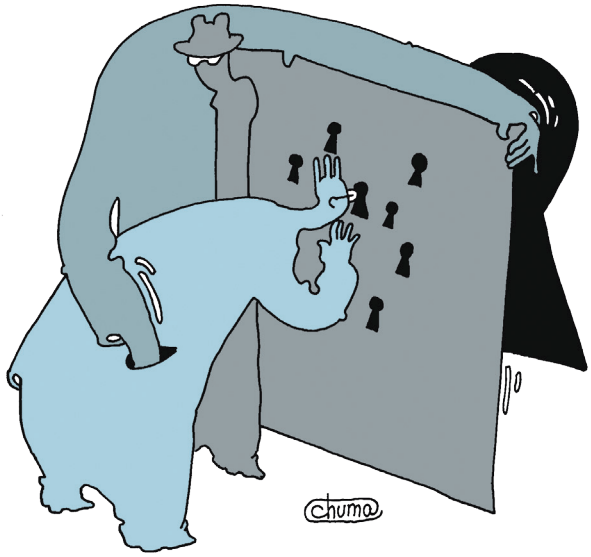
Вместо того, чтобы определять ограничения подобным образом, более безопасно использовать принцип наименьшего уровня привилегий (principle of least privilege), согласно которому программы могут делать только то, что им необходимо для выполнения своих задач, и не более того. Таким образом, даже если в программе будет обнаружена уязвимость, то возможности доступа данной программы будут жестко ограничены. Такой тип контроля называется принудительным управлением доступом (mandatory access control, MAC).

SELinux представляет собой систему маркировки, каждый процесс имеет метку. Каждый файл, каталог или даже пользователь в системе имеет метку. Даже портам, устройствам и именам хостов в системе присвоены метки. SELinux определяет правила доступа процесса к объектам с определенными метками. Это и называется политикой либо обязательным контролем доступа (Mandatory Access Control, MAC).

Владелец файла не имеет полной свободы действий над атрибутами безопасности. Стандартные атрибуты контроля доступа, такие как группа и владелец, ничего не значат для SELinux. Полностью все управляется метками. Значения атрибутов могут быть установлены и без прав root, но на это нужно иметь специальные полномочия SELinux.

Внутренняя архитектура SELinux – чуть подробнее

В самом начале своего появления SELinux была реализована в виде патча. В данном случае было непросто настраивать политику безопасности. С появлением механизмов



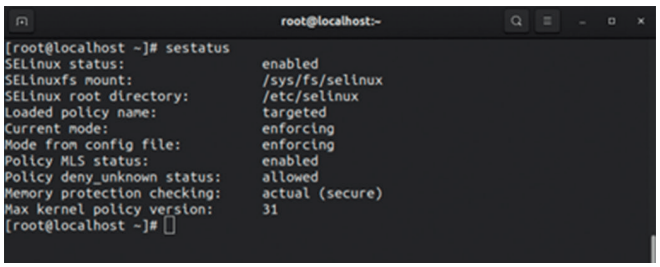
Разграничение прав доступа может привести к возникновению ряда проблем из-за того, что программа, в которой может быть обнаружена уязвимость, наследует все права доступа пользователя

LSM, настройка и управление безопасностью значительно упростились (политика и механизмы усиления безопасности были разделены), SELinux была реализована в виде подгружаемых модулей ядра. Перед доступом к внутренним объектам операционной системы производится изменение кода ядра. Это реализуется при помощи специальных функций (перехватчиков системных вызовов), так называемых функций «хуков» (англ. ***hook functions***). Функции-перехватчики хранятся в некоторой структуре данных, их целью является выполнение определенных действий по обеспечению безопасности, основанных на заранее установленной политике. Сам модуль включает в себя шесть главных компонентов: сервер безопасности; кэш вектора доступа (англ. ***Access Vector Cache, AVC***); таблицы сетевых интерфейсов; код сигнала сетевого уведомления; свою виртуальную файловую систему (selinuxfs) и реализацию функций-перехватчиков.

- Кэш вектора доступа используется для кэширования вычисленных результатов (хранения готовых решений управления доступом), полученных из сервера безопасности. Нужно это для того, чтобы минимизировать накладные расходы на производительность со стороны механизмов безопасности SELinux. Интерфейс кэша вектора доступа для сервера безопасности определен в заголовочном файле include/avc_ss.h.
- Таблица сетевых интерфейсов отображает сетевые устройства в контексты безопасности. Поддержка отдельных таблиц необходима ввиду того, что поле безопасности сетевых устройств было исключено из проекта. Сетевые устройства добавляются в таблицу, когда впервые оказываются в поле видимости функций-перехватчиков (засекаются ими) и удаляются из неё, когда устройство настраивается или перезагружается политика безопасности в связи с перенастройкой. Это возможно благодаря callback-функциям, которые регистрируют изменения конфигурации устройства и перезагрузку политик безопасности. Код таблицы сетевых интерфейсов можно найти в файле netif.c.
- Код события сетевого уведомления позволяет модулю SELinux уведомлять процессы операционной системы в случае, когда политика безопасности была перезагружена. Эти уведомления используются пространством пользователя для поддержки совместимости с ядром. Этот механизм возможен благодаря библиотеке SELinux. Код события сетевого уведомления можно найти в файле netlink.c.
- Псевдофайловая система SELinux экспортирует API-функции политики сервера безопасности в процессы операционной системы. Изначально API-функции ядра SELinux были разделены на три компонента (атрибуты процесса, файловые атрибуты, политика API) как часть изменений для внесения в версию ядра Linux 2.6. Также SELinux предоставляет поддержку политики API-вызовов. Все три компонента API нового ядра инкапсулированы в библиотеку API SELinux (libselinux). Код псевдофайловой системы можно найти в файле selinuxfs.c.
- Реализация функций перехватчиков управляет информационной безопасностью, связанной с внутренними объектами ядра и реализацией контроля доступа SELinux для каждой операции внутри ядра. Функции перехватчики вызывают сервер безопасности и кэш вектора доступа для получения решений политики безопасности и применения этих решений для маркировки по каким-либо признакам и контроля объектов ядра. Код этих функций перехватчиков расположен в файле hooks.c, а структуры данных, связанные с внутренними объектами, определены в файле include/objsec.h.

Теперь поговорим немного о политиках. Мы определяем метку для процессов определенного типа, а также на объекты файловой системы тоже определенного типа. Вот представьте себе систему, в которой объекты (процессы) это кошки и собаки. Это типы процессов. И у нас есть объекты, к которым они хотят иметь доступ – еда. Но еда у них разная: еда котов и еда собак. Нужно, чтобы объекты имели доступ только к своей еде.

После перезагрузки вы можете посмотреть состояние SELinux: $ sestatus



Здесь мы видим, что система включена: SELinuxstatus: enabled

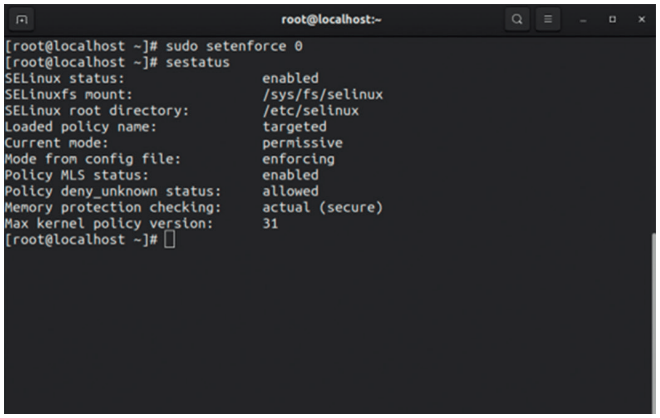
Текущий режим Current mode-enforcing, то есть система будет блокировать неразрешенные действия.

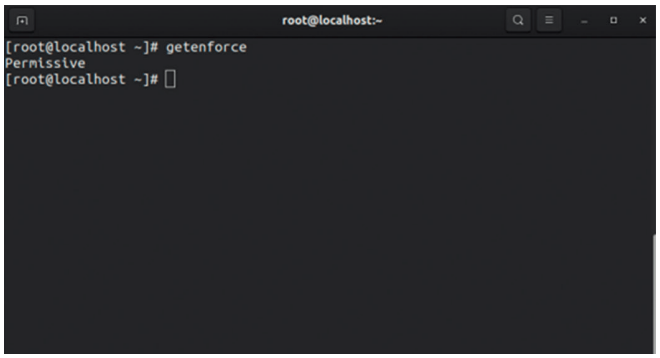
Используемая сейчас политика – targeted. Эта политика используется для того, чтобы правила SELinux распространялись только на определённые службы.

Сейчас давайте включим активный режим: sudo setenforce1

Отключить активный режим можно, передав ту же команду: sudo setenforce0

Посмотреть используемый сейчас режим тоже можно подобной командой: getenforce





Здесь мы видим, что система включена: SELinuxstatus: enabled

Текущий режим Current mode-enforcing, то есть система будет блокировать неразрешенные действия.

Используемая сейчас политика – targeted. Эта политика используется для того, чтобы правила SELinux распространялись только на определённые службы.

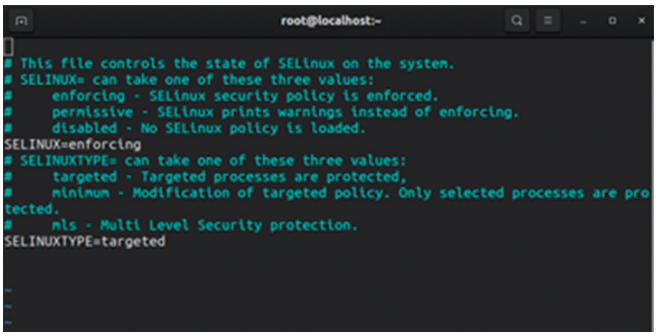
Вся основная настройка SELinux выполняется через файл /etc/selinux/config.8

Здесь можно как полностью отключить selinux, так и настроить используемую политику безопасности.

```
sudovi /etc/selinux/config
```

Этот параметр означает режим работы SELinux, вы можете указать здесь один из трех параметров enforce, permissive и disabled

```
SELINUX=enforcing
```



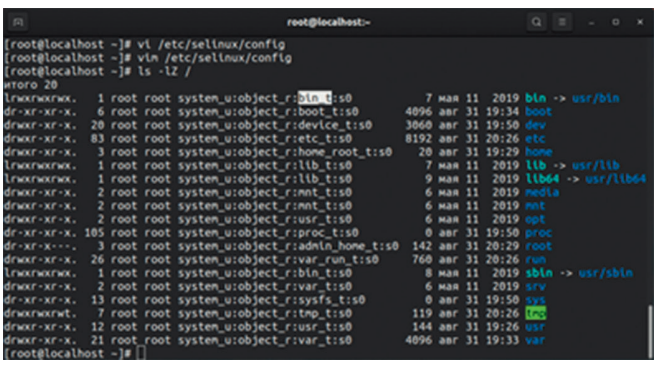
Настройка SELinux политик выполняется тоже в этом файле. За политику отвечает параметр SELINUXTYPE. Вы можете ссылаться на любую политику, расположенную в каталоге /etc/selinux.

Могут использоваться три основные политики:

- **targeted** – защищает основные системные сервисы, например, веб-сервер, DHCP, DNS, но не трогает все остальные программы.
- **strict** – самая строгая политика, управляет не только сетевыми службами, но и программами пользователя.
- **mls** – содержит не только правила, но и различные уровни безопасности. Она позволяет реализовать многоуровневую систему безопасности на основе SELinux.

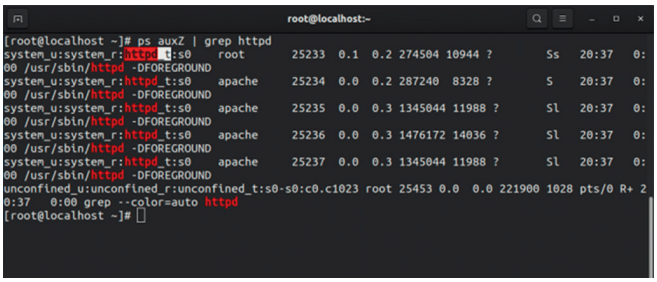
Также можно добавить свои политики. Для применения политики необходимо перезагрузить компьютер, и желательно чтобы SELinux во время этой перезагрузки был в режиме аудита (permissive). Также, чтобы система обновила все метки в файловой системе, возможно, придется создать пустой файл в корне: sudovi /.autolabel.

Каждый файл и каждый процесс имеет свою SELinux метку, которую принято называть контекстом. Какая метка будет присвоена тому или иному файлу или процессу, определяется политикой, в нашем случае – это targeted. Посмотреть контекст SELinux можно с помощью команды ls: ls -lZ/



Синтаксис строки контекста такой: имя_пользователя:имя_объекта:тип_или_домен:уровень_доступа

В политике targeted имя пользователя и имя объекта практически не используются, уровень доступа относится к MLS политиками, на него тоже можно внимания не обращать. А смотреть стоит на третье поле. Для файлов или папок оно называется типом, а для процессов – доменом. Например, у папки /bin тип bin_t. Чтобы посмотреть домен процесса используйте команду ps, например, для httpd: ps auxZl grep httpd



Как видите, здесь у сервиса httpd домен httpd_t, это значит, что ему будут доступны только те ресурсы, которые разрешено трогать этому домену, в данном случае это тип sys_httpd_content.

Все это, включая названия, определено в политике targeted. В то же время другие процессы, здесь, например, grep, запущены с доменом unconfined_t, это означает, что им будут доступны все без исключения ресурсы в системе, потому что именно такое поведение настроено в политике для этого домена.

Теперь давайте поговорим про изменение контекста.

По умолчанию папка веб-сервера находится по пути /var/www/html. У неё контекст такой, как мы обсудили ранее.

Если вы захотите перенести эту папку в другое место, то надо будет сменить контекст для новой папки. Для этого используется команда chcon, надо указать только тип: sudo chcon -Rv --type=httpd_sys_content_t/home/losst/htdocs

Однако, можно указать контекст полностью: sudo chcon -Rv system_u: object_r: httpd_sys_content_t: s0 /home/losst/htdocs

Это изменение сохранится после перезагрузки, но после обновления меток файловой системы оно будет стёрто. Чтобы этого избежать, надо добавить правило в политику.

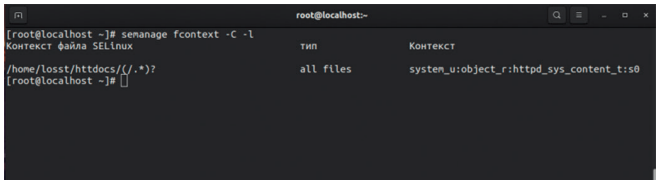
Модификация политики

Вы можете добавить дополнительные правила присвоения меток файлам в политику с помощью утилиты semanage. Чтобы установить тип httpd_sys_content_t для директории /home/losst/httdocs и, все файлы в ней надо выполнить: semanage fcontext -a -t httpd_sys_content_t “/home/losst/httdocs(/.*)?”

Если надо поменять контекст только для одного файла, то маску использовать не обязательно: semanage fcontext -a -t httpd_sys_content_t “/home/losst/httdocs/index.html”

Сама собой эта команда в файловой системе ничего не меняет, надо создать файл /.autolabel и перезагрузить компьютер или выполнить команду restorecon для нужной папки: restorecon -R -v /home/losst/httdocs

Посмотреть все добавленные таким образом правила можно с помощью такой команды: semanage fcontext -C -l



Несмотря на режим, в котором работает SELinux, все сообщения о нарушениях пишутся в лог файл /var/log/audit/audit.log. Вы можете посмотреть его вручную: less /var/log/audit/audit.log

Или для более удобного просмотра можно использовать утилиту sealert. Для её установки выполните: sudo yum install setroubleshoot. Затем можно смотреть: sealert-a /var/log/audit/audit.log

Утилиты

Утилита audit2allow

Утилита audit2allow создает разрешающие правила политики SELinux из файлов журналов, содержащих сообщения о запрете операций.

Эта утилита сканирует журналы в поиске сообщений, появляющихся, когда система не дает разрешения на операцию. Далее утилита генерирует ряд правил, которые, будучи загруженными в политику, могли бы разрешить эти операции. Однако, данная утилита генерирует только разрешающие правила Type Enforcement (TE). Некоторые отказы в использовании разрешений могут потребовать других изменений политики. Например, добавление атрибута в определение типа, для разрешения существующего ограничения (constraint), добавления разрешающего правила для роли или модификации ограничения (constraint). В случае сомнений для диагностики можно попробовать использовать утилиту audit2why.

Следует с осторожностью работать с выводом данной утилиты, убедившись, что разрешаемые операции не представляют угрозы безопасности. Обычно бывает лучше определить новый домен и/или тип или произвести другие структурные изменения. Лучше избирательно разрешить оптимальный набор операций вместо того, чтобы вслепую применить иногда слишком широкие разрешения, рекомендованные этой утилитой. Некоторые запреты на использование разрешений бывают не принципиальны для приложения. В таких случаях вместо использования разрешительного правила («allow» rule) лучше просто подавить журналирование этих запретов при помощи правила «dontaudit».

Синтаксис:

```
audit2allow
```

Опция	Значение опции
-a l --all	Прочесть входную информацию из журналов message и audit. Не используется вместе с опцией -i
-d l --dmesg	Прочесть входную информацию из вывода команды /bin/dmesg. Обратите внимание, что, когда работает auditd, не все сообщения аудита доступны через dmesg. Вместо этого используйте "ausearch -m avc l audit2allow " или "-a".

Опция	Значение опции
-f I --fcfile <File Context File>	Добавить Файл Контекстов в генерируемый пакет модуля. Требует опцию -M.
-h I --help	Вывести краткую справку по использованию
-i <inputfile> I --input <inputfile>	Прочесть входную информацию из <inputfile>
-l I --lastreload	Прочесть только часть входной информации, начиная с момента последней перезагрузки политики
-m <modulename> I --module <modulename>	Генерировать модуль. Требуется вывод <modulename>
-M <modulename>	Генерировать загружаемый пакет модуля. Опция конфликтует с -o
-o <outputfile> I --output <outputfile>	Дописать вывод в <outputfile>
-r I --requires	Генерировать вывод в синтаксисе загружаемого модуля.
-R I --reference	Генерировать эталонную политику (reference policy), используя установленные макросы. Требуется пакет selinux-policy-devel.
-t I --tefile	Указывает, что выходной файл является te-файлом (type enforcement). Может использоваться для конвертации старого формата политик в новый формат.
-v I --verbose	Включить подробный вывод

Пример: Этот пример подходит для системы, использующей пакет audit. Если вы не используете пакет audit, сообщения AVC будут появляться в /var/log/messages. В этом случае замените /var/log/audit/audit.log, используемый в примере, на /var/log/messages.

Использование audit2allow для генерации монолитной (не модульной) политики.

```
cd /etc/selinux/$ SELINUXTYPE/src/policycat /var/log/audit/
audit.log | audit2allow >> domains/misc/local.tecat domains/
misc/local.teallow cupsd_config_t unconfined_t:fifo_file
{ getattr ioctl };
```

Просмотрите domains/misc/local.te и измените его.

```
make load
```

Использование audit2allow для генерации модульной политики.

```
cat /var/log/audit/audit.log | audit2allow -m local > local.te
cat local.temodule local 1.0; require { role system_r; class
fifo_file { getattr ioctl }; type cupsd_config_t; type
unconfined_t;}; allow cupsd_config_t unconfined_t:fifo_file
{ getattr ioctl };
```

Просмотрите local.te и измените его.

Создание модуля политики вручную:

Скомпилируйте модуль:

```
checkmodule -M -m -o local.mod local.te
```

Создайте пакет:

```
semodule_package -o local.pp -m local.mod
```

Загрузите модуль в ядро.

```
semodule -i local.pp
```

Использование audit2allow для генерации и создания модуля политики.

```
cat /var/log/audit/audit.log | audit2allow -M local
```

Создается новый type enforcment файл: local.te.

Компиляция политики:

```
checkmodule -M -m -o local.mod local.te
```

Создание пакета.

```
semodule_package -o local.pp -m local.mod
```

Для того, чтобы загрузить только что созданный пакет политики в ядро, вам необходимо выполнить команду.

```
semodule -i local.pp.
```

Утилита secon

Утилита secon – просмотреть контекст SELinux для файла, программы или ввода пользователя.

Просматривает часть контекста. Контекст берется из файла, идентификатора процесса, ввода пользователя или контекста, в котором была запущена утилита secon.

Синтаксис:

```
secon [-hVurtscmPRfLp] [CONTEXT] [--file] FILE [--link] FILE
[--pid] PID
```

Опции:

Опция	Значение опции
-V, --version	Посмотреть текущую версию secon
-h, --help	Вывести информацию по использованию secon
-P, --prompt	Вывести данные в формате, подходящем для подсказки
-u, --user	Показать пользователя контекста безопасности
-r, --role	Показать роль контекста безопасности
-t, --type	Показать тип контекста безопасности
-s, --sensitivity	Показать уровень чувствительности (sensitivity level) контекста безопасности
-c, --clearance	Показать уровень допуска (clearance level) контекста безопасности
-m, --mls-range	Показать для контекста безопасности в виде диапазона уровень чувствительности (sensitivity level) и уровень допуска (clearance)
-R, --raw	Вывести уровень чувствительности (sensitivity level) и уровень допуска (clearance) в формате без трансляции
-f, --file	Получить контекст заданного файла FILE
-L, --link	Получить контекст заданного файла FILE (не следовать по символическим ссылкам)
-p, --pid	Получить контекст заданного процесса по идентификатору PID

Опция	Значение опции
--pid-exec	Получить exec контекст заданного процесса по идентификатору PID
--pid-fs	Получить fscreate контекст заданного процесса по идентификатору PID
--current, --self	Получить контекст из текущего процесса
--current-exec, --self-exec	Получить exec контекст из текущего процесса
--current-fs, --self-fs	Получить fscreate контекст из текущего процесса
--parent	Получить контекст из родительского процесса текущего процесса
--parent-exec	Получить exec контекст из родительского процесса текущего процесса
--parent-fs	Получить fscreate контекст из родительского процесса текущего процесса

Если не было задано опций, то для того, чтобы secon получил контекст из другого источника, может быть задан дополнительный аргумент CONTEXT. Если этим аргументом является знак дефиса (-), то контекст будет прочитан из стандартного ввода. Если аргументов не было задано, то secon будет пытаться прочитать контекст со стандартного ввода, но только в том случае, если стандартный ввод не терминал (tty). В этом случае secon будет вести себя как будто была передана опция --self .

Если не задана ни одна опция из --user, --role, --type, --level или --mls-range, то все они будут использованы.

Утилита audit2why

Утилита audit2why – определит из сообщения аудита SELinux причину запрета доступа.

Эта утилита обрабатывает сообщения аудита SELinux, принятые со стандартного ввода, и сообщает, какой компонент политики вызвал каждый из запретов. Если задана опция -p, то используется указанная этой опцией политика, в противном случае используется активная политика. Есть три возможные причины запрета доступа:

- отсутствует или отключено разрешительное TE правило (TE allow rule);
- нарушение ограничения (a constraint violation);
- отсутствует разрешительное правило для роли (role allow rule).

В первом случае разрешительное TE правило может присутствовать в политике, но было отключено через булевы атрибуты. Если же разрешительного правила не было, то оно может быть создано при помощи утилиты audit2allow.

Во втором случае могло произойти нарушение ограничения. Просмотрите policy/constraints или policy/mls для того, чтобы определить искомое ограничение. Обычно, проблема может быть решена добавлением атрибута типа к домену.

В третьем случае была произведена попытка сменить роль, при том, что не существует разрешительного правила для участвующей при этом пары ролей. Проблема может быть решена добавлением в политику разрешительного правила для этой пары ролей.

Синтаксис:

```
audit2why
```

Опция	Значение опции
--help	Вывести короткую подсказку
-p <policyfile>	Указать альтернативный файл политики.

Пример:

```
/usr/sbin/audit2why < /var/log/audit/audit.logtype=KERNEL
msg=audit(1115316408.926:336418): avc: denied
{ getattr } for path=/home/ sds   dev=hda5   ino=1175041
  scontext=root:secadm_r: secadm_t:s0-s9:c0.c127
tcontext=user_u:object_r:user_home_dir_t:s0 tclass=dir
```

Причина: Отсутствует или отключено разрешительное TE правило. Разрешительное TE правило может присутствовать в политике, но было отключено через булевы переключатели; проверьте булевы переключатели. Вы можете посмотреть необходимые разрешительные правила, запустив audit2allow и подав это сообщение аудита на ввод.

```
type=KERNEL msg=audit(1115320071.648:606858): avc: denied
{ append } for name=.bash_history dev=hda5 ino=1175047
scontext=user_u:user_r:user_t:s1-s9:c0.c127
tcontext=user_u:object_r:user_home_t:s0 tclass=file
```

Причина: Нарушение ограничения. Проверьте policy/constraints. Обычно, для разрешения ограничения необходимо добавить в домен атрибут типа.

Модули

Политика targeted модульная. Она состоит из множества модулей для различных программ. Чтобы посмотреть все активные на данный момент модули, выполните:

> semodule -l

Для просмотра всех установленных модулей выполните:

> semodule –list-modules=full

В прошлом пункте было показано, как с помощью утилиты sealert посмотреть возможные решения проблемы доступа. Самое частое решение – создать свой модуль для политики на основе лога. Утилита audit2allow анализирует лог файл, находит там сведения об объектах, к которым нет доступа, а затем разрешает этот доступ в новом модуле. В предыдущем примере утилита sealert посоветовала такую команду: ausearch -c 'httpd' --raw | audit2allow -M my-httpd

После этого в текущей папке появится пакет модуля с расширением .pp, который можно уже установить с помощью утилиты semodule: sudo semodule -i my-httpd.pp

Флаги

Кроме того, поведение политики можно настраивать с помощью флагов. Обычно флаги, которые надо включить, рекомендует та же утилита sealert. Все флаги определены в модулях политики. Посмотреть, какие флаги доступны сейчас, и их состояние можно с помощью команды: getsebool -a. Чтобы изменить состояние флага, используйте команду setsebool. Например, чтобы разрешить модулям httpd подключаться к сети, выполните: setsebool -P httpd_can_network_connecton.

Создание своей политики SELinux

Описание процесса

Для компиляции готовой политики нам нужно создать 2 файла:

> .te (Type Enforcement) – содержит описание сущностей для политики.

- .fc (File Context) – файл для описания доступов к каталогам.

На основе созданных файлов мы получаем политику. pp, которую можно будет установить и включить при помощи команды semodule.

После включения политики необходимо проверить работоспособность приложения и вывод утилиты audit2allow, которая позволяет понять, каких прав не хватает нашему процессу.

Приступим.

Создание файла .te

Создадим каталог, в котором будем работать, и перейдем в него:

```
mkdir -p /opt/selinux/mypolicy
cd /opt/selinux/mypolicy
```

Создаем файл со следующим содержимым:

```
vi myapp.te
policy_module(myapp, 1.0.0)

type myapp_t;
type myapp_exec_t;
type myapp_opt_t;
type myapp_conf_t;
type myapp_log_t;
type myapp_cache_t;
type myapp_tmp_t;
type myapp_port_t;

files_config_file(myapp_conf_t)
files_type(myapp_cache_t)
files_type(myapp_opt_t)
logging_log_file(myapp_log_t)
files_tmp_file(myapp_tmp_t)
corenet_port(myapp_port_t)

application_domain(myapp_t, myapp_exec_t)
init_daemon_domain(myapp_t, myapp_exec_t)
corecmd_exec_bin(myapp_t)
libs_use_ld_so(myapp_t)
kernel_read_system_state(myapp_t)
files_rw_generic_tmp_dir(myapp_t)
sysnet_read_config(myapp_t)
dev_read_rand(myapp_t)
fs_getattr_xattr_fs(myapp_t)
sysnet_dns_name_resolve(myapp_t)
logging_search_logs(myapp_t)
logging_log_filetrans(myapp_t, myapp_log_t, file)
files_poly_member_tmp(myapp_t, myapp_tmp_t)
```

* содержимое файла разбито на блоки. Рассмотрим их подробнее.

Описание типов:

- myapp_t – это будет типом процесса для нашего приложения.
- myapp_exec_t – применяется для исполняемых файлов.
- myapp_conf_t – разрешения для конфигурационных файлов.
- myapp_opt_t – применим для разрешения доступа к файлам и папкам в каталоге /opt.
- myapp_log_t – для логирования.
- myapp_cache_t – для кэширования.
- myapp_tmp_t – для временных файлов.
- myapp_port_t – для сетевого порта, который использует наше приложение.

Макросы, с помощью которых мы дадим стандартные разрешения нашим типам:

- files_config_file – дает разрешения для использования конфигурационных файлов.
- files_type – для различных файлов. В нашем примере это кэш.
- logging_log_file – разрешения для log-файлов.
- files_tmp_file – для временных файлов.
- corenet_port – разрешение для сетевых портов.

Макросы, которые дают конкретные разрешения нашему процессу:

- application_domain – привязываем процесс myapp_t к типу myapp_exec_t. Последнему разрешается запуск исполняемых файлов.
- init_daemon_domain – разрешаем запуск через systemd.
- corecmd_exec_bin – разрешаем запуск бинарных файлов из каталогов /bin, /usr/bin.
- libs_use_ld_so – разрешаем подключение библиотек.
- kernel_read_system_state – чтение состояния системы, например, утилизации процессора и памяти.
- files_rw_generic_tmp_dir – разрешаем писать в каталог для временных файлов (например, /tmp).
- sysnet_read_config – разрешаем читать конфигурационные файлы сетевых настроек.
- dev_read_rand – получение случайных чисел.
- fs_getattr_xattr_fs – получение атрибутов файлов.
- sysnet_dns_name_resolve – разрешаем разрешение имен в IP-адреса с помощью DNS.
- logging_search_logs – доступ к каталогу с логами.
- logging_log_filetrans – указываем, что логи, создаваемые нашим процессом myapp_t, будут иметь тип myapp_log_t.
- files_poly_member_tmp – указываем, что временные файлы, создаваемые нашим процессом myapp_t, будут иметь тип myapp_tmp_t.

Создание файла .fc

Создаем файл со следующим содержимым:

```
vi myapp.fc
/opt/myapp/myapp.sh        gen_
context(system_u:object_r:myapp_exec_t)
/opt/myapp(/.*)?           gen_
context(system_u:object_r:myapp_opt_t)
/etc/myapp/(.*)            gen_
context(system_u:object_r:myapp_conf_t)
/opt/myapp/cache(/.*)?     gen_
context(system_u:object_r:myapp_cache_t)
/opt/myapp/tmp(/.*)?       gen_
context(system_u:object_r:myapp_tmp_t)
/var/log/myapp(/.*)?       gen_
context(system_u:object_r:myapp_log_t)
```

* в данном примере мы каждому каталогу и его содержимому задаем политику установки контекста. Например, для каталога /opt/myapp/cache задается контекст myapp_cache_t, которому политиками в файле myapp.te мы разрешили писать кэш. И так далее.

Сборка политики

Для сборки политики нам нужен пакет:

```
yum install policycoreutils-devel
```

Саму сборку мы выполняем командой:

```
make -f /usr/share/selinux/devel/Makefile
```

Система найдет в текущем каталоге файлы .te и .fc и выполнит проверку со сборкой.

Мы должны увидеть что-то на подобие:

```
Compiling targeted myapp module
/usr/bin/checkmodule:  loading policy configuration from tmp/
myapp.tmp
/usr/bin/checkmodule:  policy configuration loaded
/usr/bin/checkmodule:  writing binary representation (version
19) to tmp/myapp.mod
Creating targeted myapp.pp policy package
rm tmp/myapp.mod.fc tmp/myapp.mod
```

Мы увидим в нашем каталоге /opt/selinux/mypolicy файл с расширением .pp. Это и есть наша политика. Ее переносим на конечный компьютер, где хотим ее запустить и проверить.

Работа с политикой

Прежде чем закончить работу с политикой, нам нужно проверить установку и отладить ее работу. Рассмотрим процессы по очереди.

Применение

На целевом компьютере, где должна работать наша политика, вводим:

```
semodule -i myapp.pp
```

* предполагается, что мы находимся в каталоге, где лежит файл myapp.pp, иначе, прописываем к нему полный путь.

В системе появится наша политика. Посмотреть можно командой:

```
semodule -l | grep myapp
```

Мы должны увидеть что-то на подобие:

```
myapp    1.0.0
```

Применить политику можно командой:

```
semodule -e myapp
```

Для всех путей, характерных нашему приложению, мы должны сбросить политики SELinux – тогда применятся те. что указаны в политике, например:

```
restorecon -Rv /opt/myapp
```

* в данном примере мы указываем команду для сброса политик каталога /opt/myapp.

** нам нужно повторить данную команду для всех каталогов и файлов, которые использует наше приложение.

Так как в нашем примере приложение использует сетевой порт, мы должны его назначить с помощью утилиты semanage. Для этого на целевой компьютер ставим:

```
yum install policycoreutils-python
```

После можем использовать команду:

```
semanage port -a -t myapp_port_t -p tcp 8888
```

* где myapp_port_t – созданный нами тип для сетевого порта; tcp – используемый приложением сетевой протокол; 8888 – порт, который используется для прослушивания сетевых запросов.

Отладка

Не факт, что мы учли все запросы приложения. Чтобы ничего не упустить, на тестовом целевом компьютере мы включаем SELinux (если он был выключен), ждем некоторое время, тестируя приложение и после выполняем команду:

```
audit2allow -b -r -t myapp
```

Данная команда покажет все запросы, для которых у нашего процесса нет доступа. Мы увидим готовые команды, наподобие:

```
require {
   type unconfined_t;
   type myapp_t;
   class dir relabelto;
}

#============= unconfined_t ==============

allow unconfined_t myapp_t:dir relabelto;
```

... которые нам нужно отдельно использовать в политике.

Однако решать, какие команды использовать, а какие нет, нужно самостоятельно, так как не все данные запросы действительно нужны нашему приложению.

Например, если мы решим добавить команду в политику, то откроем наш файл .te: vi myapp.te

Отредактируем версию, добавим require с типом и классом, а также команду, что предложила audit2allow: policy_module (myapp, 1.0.1)

```
...

require {
    type unconfined_t;
    class dir relabelto;
}

...

allow unconfined_t myapp_t:dir relabelto;
```

* обратите внимание на require. Данная директива определяет не создание нового типа и классов, а использование тех, что есть в системе. Это важно, так как если попытаться определить тип, который уже есть в системе, мы получим ошибку Re-declaration of type unconfined_t, Failed to create node.

Затем снова выполним сборку политики и повторную ее установку, и применение. EOF

Ключевые слова: Selinux- архитектура, режимы, политики, создание собственных политик