Лабораторная работа №2. Часть 2 Углубленное исследование безопасности ядра и

среды выполнения в ОС Linux

*Цель работы:* Изучение и настройка различных аспектов безопасности ядра Linux, включая интеграцию Integrity Measurement Architecture (IMA), управление привилегиями и настройку chroot.

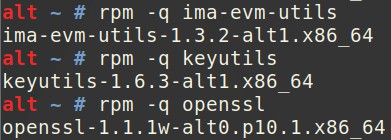
1. Подсистема IMA/EVM

Integrity Measurement Architecture (IMA) — подсистема ядра Linux, позволяющая осуществлять контроль целостности файловой системы. IMA включает в себя две подсистемы — IMA-measurement (измерение) и IMA-appraisal (оценка). Первая собирает хеш-образы файлов, вторая сравнивает собранный хеш с сохраненным хешем и запрещает доступ в случае несоответствия. Собранные хеш-образы хранятся в расширенных атрибутах файловой системы.

Подсистема целостности Linux позволяет использовать подписи IMA. Подпись IMA защищает содержимое файла.

Утилита **evmctl** создает и проверяет цифровые подписи, которые используются подсистемой IMA, а также добавляет ключи в набор ключей ядра.

В ОС “Альт Рабочая станция” пакет *ima-evm-utils* установлен по умолчанию. Проверим наличие пакета *ima-evm-utils* с помощью утилиты *rpm*. Также для практики нам потребуются пакеты *keyutils* и *openssl*:

# rpm -q ima-evm-utils

Для IMA-подписи произвольного файла необходимо предварительно сгенерировать ключ:

1. Чтобы сгенерировать 2048-битный RSA-ключ и x509 сертификат, создадим конфигурационный файл test-ca.conf в любом каталоге и запишем туда следующую информацию:

[ req ]

distinguished\_name = req\_distinguished\_name prompt = no

string\_mask = utf8only x509\_extensions = v3\_ca

[ req\_distinguished\_name ] O = IMA

CN = IMA/EVM certificate signing key emailAddress = ca@ima-ca

[ v3\_ca ] basicConstraints=CA:TRUE subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid:always,issuer

Файл test-ca является шаблоном для генерации сертификата, который в свою очередь будет использоваться в системе для создания IMA-подписей.

Секция *[ req ]* определяет параметры для создания запроса на сертификат.

Секция *[ req\_distinguished\_name ]* задает значения для Distinguished Name (DN), представленного в сертификате.

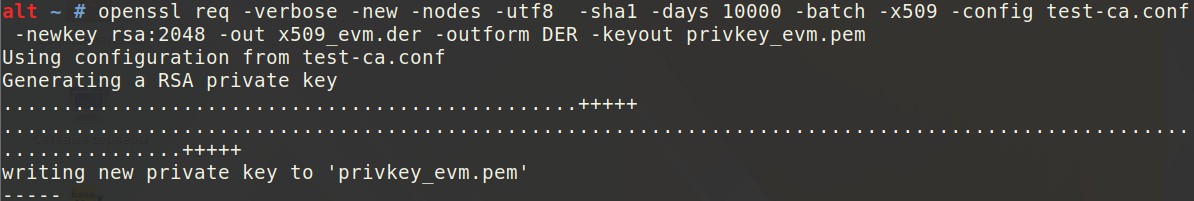
Секция *[ v3\_ca ]* определяет расширения, которые будут добавлены к создаваемому сертификату.

1. Генерируем ключ с использованием созданного конфигурационного файла:

# openssl req -verbose -new -nodes -utf8 -sha1 -days 100000 -batch -x509 \

-config test-ca.conf -newkey rsa:2048 -out x509\_evm.der -outform DER \

-keyout privkey\_evm.pem



Параметры команды:

req запускает создание запроса на сертификат или самого сертификата;

-verbose детализированного выводит информацию в консоль во время выполнения команды;

-new указывает на создание нового запроса на сертификат или самого сертификата;

-nodes указывает, что закрытый ключ не должен быть зашифрован;

-utf8 указывает на кодировку UTF-8 для строковых значений;

-sha1 указывает, что при создании сертификата будет использоваться алгоритм хеширования SHA-1;

-days задает срок действия сертификата в днях;

-batch запускает команду в пакетном режиме, т.е. без запроса ввода данных от пользователя;

-config указывает, что настройки сертификата есть в файле *test-ca.conf*;

-newkey указывает создать новый RSA-ключ длиной 2048 бит;

-out указывает файл *x509\_evm.der*, в котором будет содержимое сертификата;

-outform указывает формат DER для файла *x509\_evm.der*;

-keyout указывает файл *privkey\_evm.pem*, в котором будет сохранен закрытый ключ.

Важно! Приватный ключ *privkey\_evm.pem* и публичный сертификат

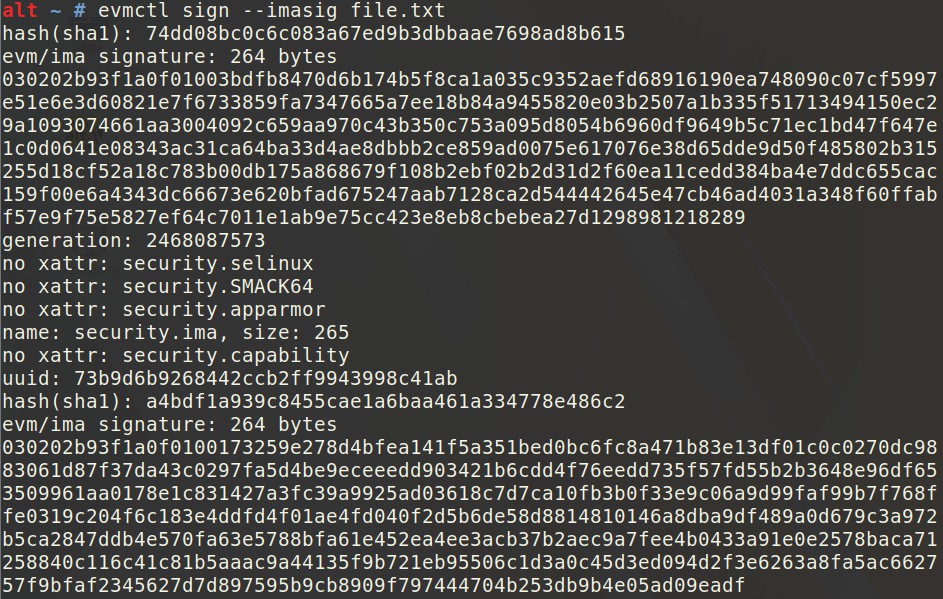
*x509\_evm.der* следует расположить в каталоге */etc/keys*: # mv x509\_evm.der /etc/keys

# mv privkey\_evm.pem



Создадим файл *file.txt* и подпишем его сгенерированным ключом: # echo “Проверка подсистемы ядра IMA” > file.txt

# evmctl sign --imasign file.txt

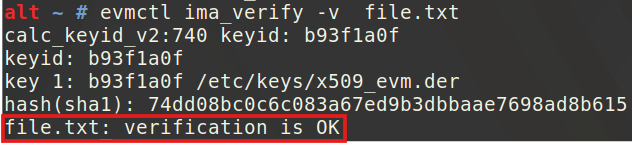


Примечание: Предполагается, что ключ хранится в каталоге */etc/keys*. Если расположение находится в другом каталоге, то следует добавить в команду опцию --key с указанием пути до ключа.

Проверим IMA-подпись по публичному сертификату

*/etc/keys/x509\_evm.der*.

# evmctl ima\_verify -v file.txt



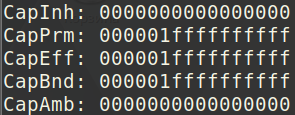
В случае соответствия подписи будет выведено сообщение: file.txt: verification is OK

1. Capabilities

*Capabilities* — атрибуты ядра, которые предоставляют некоторые root-привилегии процессам или исполняемым файлам.

Разрешения процессов (process capabilities)

Каждый процесс имеет пять 64-битных чисел (наборов), содержащих биты разрешений, которые можно посмотреть в файлах */proc/<pid>/status*.

Посмотрим разрешения для процесса *systemd* (*PID = 1*): # cat /etc/1/status

Представленные в шестнадцатеричной системе счисления числа являются битовыми картами с наборами разрешений:

*Inheritable* разрешения могут наследовать дочерние процессы; *Permitted* разрешения могут использоваться самим процессом; *Effective* указывают на текущие действующие разрешения;

*Bounding* представляет ограничивающий набор для каждого процесса;

*Ambient* предоставляют разрешения не-root пользователю, без использования setuid или файловых разрешений.

Например, если задача запрашивает выполнение привилегированной операции (например, привязку к определенным портам), то ядро проверяет действующий ограничивающий набор на наличие CAP\_NET\_BIND\_SERVICE. Если он установлен, то операция продолжается. В противном случае операция отклоняется с EPERM (операция не разрешена). CAP\_ определены в исходном коде ядра и нумеруются последовательно.

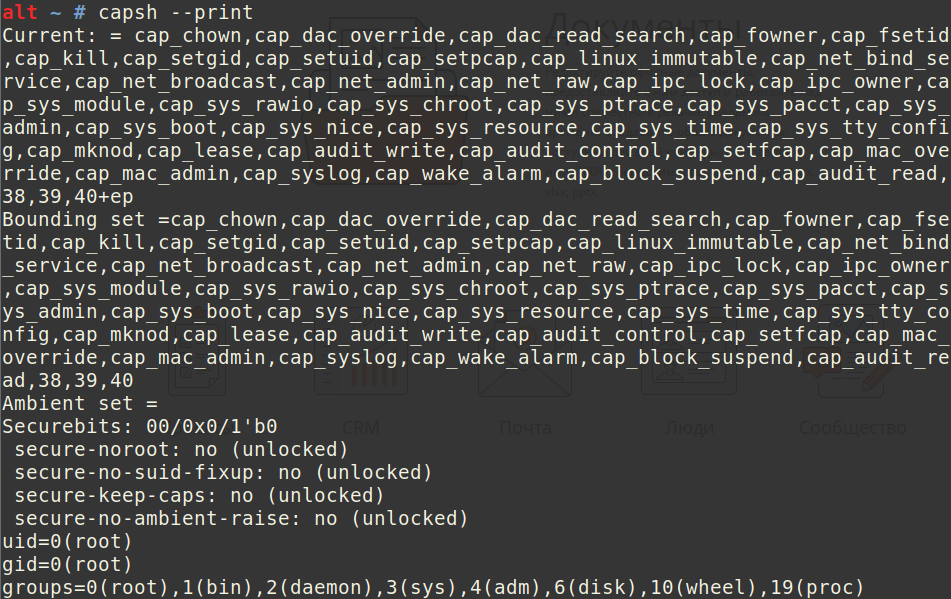
Полный человекочитаемый список полномочий можно найти в справочном руководстве *capabilities(7)*:

# man capabilities

Чтобы выйти из справочного руководства, нажмите клавишу q.

Чтобы упростить управление и проверку полномочий, можно воспользоваться библиотекой *libcap*. В данную библиотеку входит утилита **capsh**, которая, помимо прочего, позволяет показать свои полномочия:

# capsh --print



Основные параметры:

*Current* отображает эффективные, наследуемые и доступные привилегии процесса *capsh* в формате *cap\_to\_text*. В этом формате права перечислены как группы разрешений *“capability[,capability…]+(e|i|p)”*, где e — эффективные, i — наследуемые, p — доступные привилегии.

*Bounding set/Ambient set* содержат только список разрешений, которые находятся в специальным наборах.

*Securebits* отображают специальные флаги безопасности процесса.

Флаги показывают дополнительные настройки безопасности.

Данной утилите не хватает информации о флаге *NoNewPrivs*, которую можно посмотреть в */proc/<pid>/status*. Когда флаг *NoNewPrivs* активирован, процесс не может увеличить свои привилегии с помощью вызова, такого как *execve* (исполнение нового бинарного файла). Т.е. даже если процесс попытается выполнить программу, имеющую повышенные привилегии или права, он не сможет унаследовать эти привилегии.

Разрешения файлов (file capabilities)

Иногда пользователю с ограниченным набором прав необходимо запустить файл, который требует больше полномочий. Одним из вариантов решения данного вопроса — установка бита SUID. Однако такой файл будет иметь полные root-права при выполнении любым пользователем.

Другой вариант — установка файловых разрешений. Поскольку файловые разрешения хранятся в виде расширенного атрибута файла, необходима файловая система с поддержкой расширенных атрибутов (ext\*, XFS, …). Для изменения этого атрибута необходимо разрешение CAP\_SETFCAP.

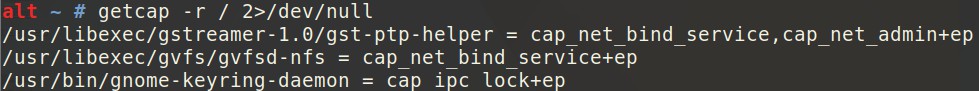
Для управления привилегиями необходим пакет *libcap*. Проверим наличие пакета с помощью утилиты *rpm*:

# rpm -q libcap



Выведем привилегии файла *file.txt* с помощью утилиты *getcap*: # getcap file.txt

Отсутствие вывода означает, что файл не содержит привилегий.

Выведем установленные во всей системе привилегии для файлов: # getcap -r / 2>/dev/null

Файл */usr/bin/gnome-keyring-daemon* является демоном ключей gnome и у него есть расширенная возможность *cap\_ipc\_lock*, что позволяет процессу блокировать IPC (межпроцессное взаимодействие). Дополнительно, присутствует бит выполнения (+ep), указывающий, что процесс с этим файлом как исполняемым запускается со своими возможностями.

Файл */usr/libexec/gstreamer-1.0/gst-ptp-helper* относится к подсистеме графики или мультимедиа (gstreamer) и имеет расширенные возможности *cap\_net\_bind\_service* и *cap\_net\_admin*. Привилегия *cap\_net\_bind\_service* разрешает процессу привязываться к любому сетевому интерфейсу для прослушивания соединений. Привилегия *cap\_net\_admin* позволяет процессу управлять административными возможностями сети.

Файл */usr/libexec/gvfs/gvfsd-nfs* связан с сетевой файловой системой (GVFS) и имеет расширенную возможность *cap\_net\_bind\_service*, которая разрешает процессу привязываться к любому сетевому интерфейсу.

Установим привилегию *cap\_dac\_override* для файла *file.txt*. Привилегия *cap\_dac\_override* игнорирует проверку прав доступа, что позволяет исполняемому файлу получать доступ к любым файлам независимо от их прав:

# setcap ‘cap\_dac\_override=ep’ file.txt Проверим установленные привилегии:

# getcap file.txt



Удалить привилегии можно также с помощью утилиты *setcap* с опцией *-r*.

# setcap -r file.txt

Еще раз проверим установленные привилегии:

# getcap file.txt

1. Среда Chroot

Среда *chroot* (Change root) изменяет корневую директорию для текущего процесса и его дочерних процессов.

Основные особенности chroot:

1. Изоляция процессов.

*Chroot* может запускать процессы в изолированном пространстве имен файловой системы, чтобы предотвратить доступ к конфиденциальным данным и ресурсам.

1. Тестирование программ.

*Chroot* можно использовать для тестирования программ в безопасной среде, где они не могут повредить хост-систему.

1. Восстановление системы.

*Chroot* можно использовать для восстановления системы после сбоя или вредоносной атаки.

1. Управление разрешениями.

*Chroot* можно использовать для ограничения доступа пользователей к определенным файлам и каталогам на сервере.

Синтаксис утилиты **chroot**:

# **chroot** [options] newroot [command [arg] …]

Утилита *chroot* запускает *command* с корневым каталогом, установленным в *newroot*.

Изолируем служебный процесс *ls*. Для этого создаем каталог *env*, в котором будет новая среда:

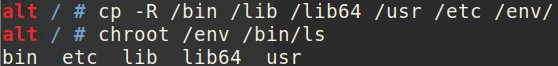
# cd / && mkdir env

Копируем все необходимые файлы и каталоги в данную папку: # cp -R /bin /lib /lib64 /usr /etc /var /env/

Примечание: Выполнение команды может занять некоторое время!

Теперь выполним следующую команду:

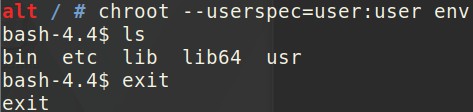
# chroot /env /bin/ls



Таким образом, команда *chroot* изменила корневой каталог для текущего процесса на */env*, а затем команда */bin/ls* вывела содержимое этого каталога.

Зададим имя пользователя *user* и группу *user*, от имени которых команда *ls* будет выполняться в изолированной среде. Затем выходим из окружения с помощью команды *exit*.

# chroot --userspec=user:user /env

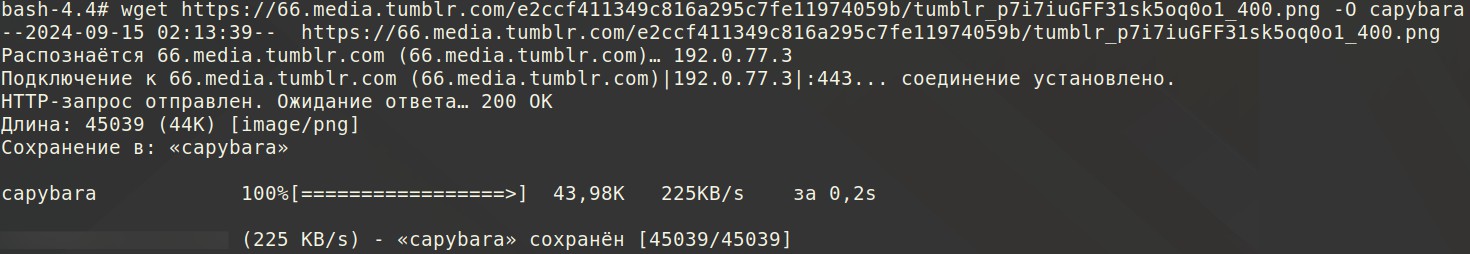


Снова зайдем в chroot-окружение:

# chroot /env /bin/bash

Установим пакет *caca-utils* для вывода изображений в текстовом виде: # apt-get install caca-utils

Далее скачиваем по ссылке изображение с помощью утилиты *wget*: # wget

[https://66.media.tumblr.com/e2ccf411349c816a295c7fe11974059b/tumblr\_p7i7iuGF](https://66.media.tumblr.com/e2ccf411349c816a295c7fe11974059b/tumblr_p7i7iuGFF31sk5oq0o1_400.png) [F31sk5oq0o1\_400.pn](https://66.media.tumblr.com/e2ccf411349c816a295c7fe11974059b/tumblr_p7i7iuGFF31sk5oq0o1_400.png)g -O capybara

С помощью команды *img2txt* выводим изображение в текстовом виде: # img2txt capybara

