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
```

```
alt ~ # rpm -q ima-evm-utils
ima-evm-utils-1.3.2-alt1.x86_64
alt ~ # rpm -q keyutils
keyutils-1.6.3-alt1.x86_64
alt ~ # rpm -q openssl
openssl-1.1.1w-alt0.p10.1.x86_64
```

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.

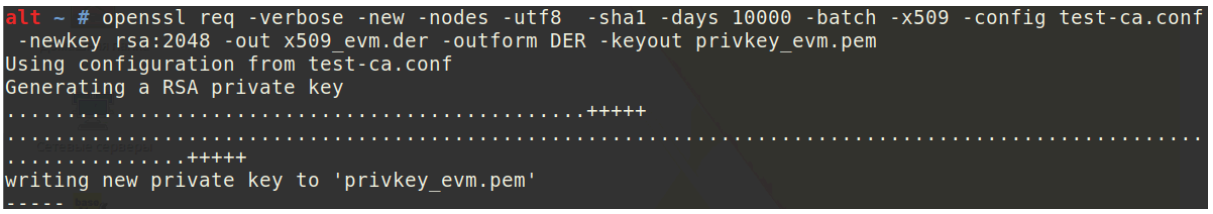
Секция *[req]* определяет параметры для создания запроса на сертификат.

Секция *[req_distinguished_name]* задает значения для Distinguished Name (DN), представленного в сертификате.

Секция *[v3_ca]* определяет расширения, которые будут добавлены к создаваемому сертификату.

2. 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

```
alt ~ # mv x509_evm.der /etc/keys/
alt ~ # mv privkey_evm.pem /etc/keys/
```

Создадим файл *file.txt* и подпишем его сгенерированным ключом:

echo “Проверка подсистемы ядра IMA” > file.txt

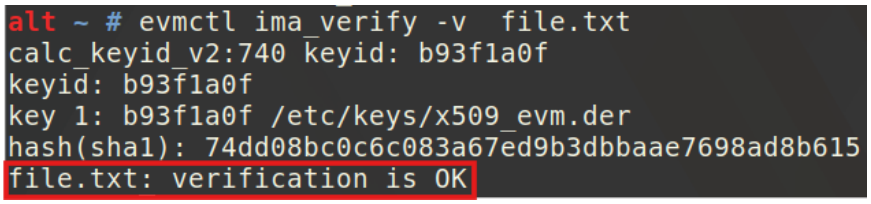
evmctl sign --imasign file.txt

```
alt ~ # evmctl sign --imasig file.txt
hash(sha1): 74dd08bc0c6c083a67ed9b3dbbaae7698ad8b615
evm/ima signature: 264 bytes
030202b93f1a0f01003bdfb8470d6b174b5f8ca1a035c9352aefd68916190ea748090c07cf5997
e51e6e3d60821e7f6733859fa7347665a7ee18b84a9455820e03b2507a1b335f51713494150ec2
9a1093074661aa3004092c659aa970c43b350c753a095d8054b6960df9649b5c71ec1bd47f647e
1c0d0641e08343ac31ca64ba33d4ae8dbbb2ce859ad0075e617076e38d65dde9d50f485802b315
255d18cf52a18c783b00db175a868679f108b2ebf02b2d31d2f60ea11cedd384ba4e7ddc655cac
159f00e6a4343dc66673e620bfad675247aab7128ca2d544442645e47cb46ad4031a348f60ffab
f57e9f75e5827ef64c7011e1ab9e75cc423e8eb8cbebea27d1298981218289
generation: 2468087573
no xattr: security.selinux
no xattr: security.SMACK64
no xattr: security.apparmor
name: security.ima, size: 265
no xattr: security.capability
uuid: 73b9d6b9268442ccb2ff9943998c41ab
hash(sha1): a4bdf1a939c8455cae1a6baa461a334778e486c2
evm/ima signature: 264 bytes
030202b93f1a0f0100173259e278d4bfea141f5a351bed0bc6fc8a471b83e13df01c0c0270dc98
83061d87f37da43c0297fa5d4be9eceeedd903421b6cdd4f76eedd735f57fd55b2b3648e96df65
3509961aa0178e1c831427a3fc39a9925ad03618c7d7ca10fb3b0f33e9c06a9d99faf99b7f768f
fe0319c204f6c183e4ddfd4f01ae4fd040f2d5b6de58d8814810146a8dba9df489a0d679c3a972
b5ca2847ddb4e570fa63e5788bfa61e452ea4ee3acb37b2aec9a7fee4b0433a91e0e2578baca71
258840c116c41c81b5aaac9a44135f9b721eb95506c1d3a0c45d3ed094d2f3e6263a8fa5ac6627
57f9bfaf2345627d7d897595b9cb8909f797444704b253db9b4e05ad09eadf
```

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

2. 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

```
CapInh: 0000000000000000
CapPrm: 000001ffffffffff
CapEff: 000001ffffffffff
CapBnd: 000001ffffffffff
CapAmb: 0000000000000000
```

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

```
alt ~ # capsh --print
Current: = cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid
,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_se
rvice,cap_net_broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,ca
p_sys_module,cap_sys_rawio,cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_
admin,cap_sys_boot,cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_confi
g,cap_mknod,cap_lease,cap_audit_write,cap_audit_control,cap_setfcap,cap_mac_ove
rride,cap_mac_admin,cap_syslog,cap_wake_alarm,cap_block_suspend,cap_audit_read,
38,39,40+ep
Bounding set =cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fse
tid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind
_service,cap_net_broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner
,cap_sys_module,cap_sys_rawio,cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_s
ys_admin,cap_sys_boot,cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_co
nfig,cap_mknod,cap_lease,cap_audit_write,cap_audit_control,cap_setfcap,cap_mac_
override,cap_mac_admin,cap_syslog,cap_wake_alarm,cap_block_suspend,cap_audit_re
ad,38,39,40
Ambient set =
Securebits: 00/0x0/1'b0
 secure-noroot: no (unlocked)
 secure-no-suid-fixup: no (unlocked)
 secure-keep-caps: no (unlocked)
 secure-no-ambient-raise: no (unlocked)
uid=0(root)
gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),19(proc)
```

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

```
alt ~ # rpm -q libcap
libcap-2.27.0.2.ac1e-alt3.x86_64
```

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

```
alt ~ # getcap -r / 2>/dev/null
/usr/libexec/gstreamer-1.0/gst-ptp-helper = cap_net_bind_service,cap_net_admin+ep
/usr/libexec/gvfs/gvfsd-nfs = cap_net_bind_service+ep
/usr/bin/gnome-keyring-daemon = cap_ipc_lock+ep
```

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

```
alt ~ # getcap file.txt
file.txt = cap_dac_override+ep
```

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

setcap -r file.txt

Еще раз проверим установленные привилегии:

getcap file.txt

3. 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.

2. Тестирование программ.

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

3. Восстановление системы.

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

4. 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

```
alt / # cp -R /bin /lib /lib64 /usr /etc /env/
alt / # chroot /env /bin/ls
bin  etc  lib  lib64  usr
```

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

```
alt / # chroot --userspec=user:user env
bash-4.4$ ls
bin  etc  lib  lib64  usr
bash-4.4$ exit
exit
```

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_p7i7iuGFF31sk5oq0o1_400.png -O capybara

```
bash-4.4# wget https://66.media.tumblr.com/e2ccf411349c816a295c7fe11974059b/tumblr_p7i7iuGFF31sk5oq0o1_400.png -O capybara
--2024-09-15 02:13:39--  https://66.media.tumblr.com/e2ccf411349c816a295c7fe11974059b/tumblr_p7i7iuGFF31sk5oq0o1_400.png
Распознаётся 66.media.tumblr.com (66.media.tumblr.com)… 192.0.77.3
Подключение к 66.media.tumblr.com (66.media.tumblr.com)|192.0.77.3|:443... соединение установлено.
HTTP-запрос отправлен. Ожидание ответа… 200 OK
Длина: 45039 (44K) [image/png]
Сохранение в: «capybara»

capybara            100%[=================>]  43,98K   225KB/s    за 0,2s

                    (225 KB/s) - «capybara» сохранён [45039/45039]
```

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

img2txt capybara