1. Министерство науки и высшего образования Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 3**

1. **«Виртуализация на уровне ОС»**
2. по дисциплине «Безопасность операционных систем»

Вариант 2

1. Выполнил
2. студент гр. 4851003/70801 Гасанов Э.А.

<*подпись*>

1. Ассистент
2. преподавателя Ахтямов Д.Р.

<*подпись*>

1. Санкт-Петербург
2. 2020

**Цели**

Изучение принципов работы виртуализации на уровне ОС на примере контейнеров Docker.

**Задачи**

1. Разработать Dockerfile vsftpd сервиса в соответствии с вариантом 2.
2. Определить минимальный набор привилегий, необходимых для работы сервиса.
3. Настроить необходимые привилегии с помощью Docker.
4. Разработать программу, использующую библиотеку libcap-ng, для настройки привилегий аналогично предыдущему пункту.
5. Определить минимальный набор системных вызовов, необходимых для работы сервиса, и написать соответствующий профиль seccomp.

**Ход работы**

В данной лабораторной работы сборка и запуск Docker производились на Ubuntu x64. После установки Docker командой *sudo apt install* docker.io и его запуска командами *systemctl start docker* и *systemctl enable docker*, провреим работоспособность Docker командой *systemctl status docker*:

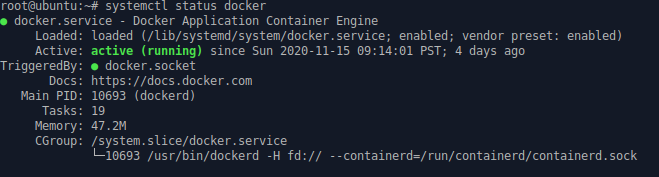


Рисунок 1 – проверка работоспособности Docker.

Согласно полученному варианту, создадим vsftpd сервис. Для этого создадим Dockerfile, где по мимо всего прочего установим с помощью пакетного менеджера (apt) нужную версию vsftpd и supervisord. Последний является клиент-серверной системой.

Далее необходимо создать доступ виртуальным пользователям к ftp-серверу. Для этого используются скрипт vsftpd\_vuser.sh, который добавляется в папку conf с помощью ADD и затем запускается RUN chmod a+x /bin/vsftpd\_vuser.sh. Построим образ следующей командой:

|  |
| --- |
| root@ubuntu:/opt2/docker-vsftpd# docker build --rm -t vsftpd:3.0.2 . |

И запустим контейнер из образа командой:

|  |
| --- |
| root@ubuntu:/opt2/docker-vsftpd# docker run -d --name="vsftpd" -h "vsftpd" -p 20:20 -p 21:21 -v /home:/home -v /tmp:/tmp vsftpd:3.0.2 |

Создадим виртуального пользователя командами:

|  |
| --- |
| root@ubuntu:/opt2/docker-vsftpd# mkdir -p /data/ftpdata/elvin  root@ubuntu:/opt2/docker-vsftpd# docker exec vsftpd vsftpd\_vuser.sh elvin 123366 |

Создадим контейнер и подключимся к созданному ftp-серверу с помощью FileZilla:

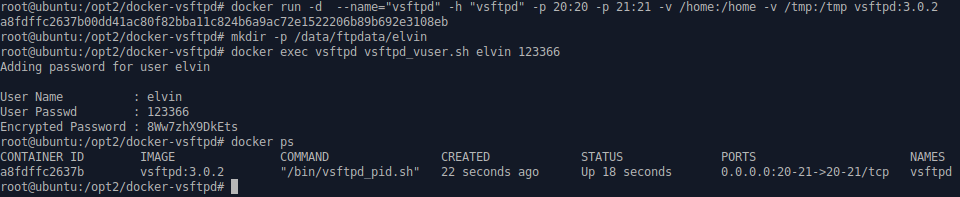


Рисунок 2 - Первый запуск сервера.

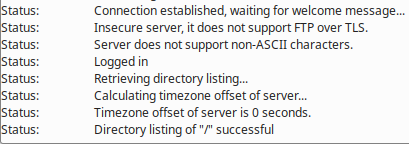


Рисунок 3 - Подключение с помощью FileZilla.

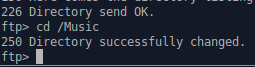


Рисунок 4 – Подключение по ftp в терминале.

**Сброс привилегий (capabilities) до запуска контейнера**

Если перед запуском контейнера сбросить привилегии (capabilities), то процесс внутри контейнера никогда не сможет получить их обратно. Это сделано в целях безопасности. По умолчанию привилегированный процесс в контейнере имеет следующие привилегии (capabilities):

|  |
| --- |
| cap\_chown,cap\_dac\_override,cap\_fowner,cap\_fsetid,cap\_kill,cap\_setgid,cap\_setuid,cap\_setpcap,cap\_net\_bind\_service,cap\_net\_raw,cap\_sys\_chroot,cap\_mknod,cap\_audit\_write,cap\_setfcap. |

Этот перечень был получен путём вызова функции

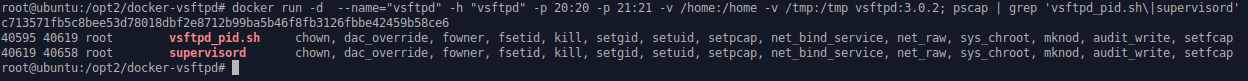


Рисунок 5 – Запуск контейнера и вывод привилегий с помощью pscap.

Так же имеется альтернативный способ получения привилегий (capabilities) c помощью функций:

* Getcaps возвращает привилегии (capabilities) процесса, номер которого ей передан
* С помощью вызова # cat /proc/<PID>/status | grep Cap можно получить битовые карты с наборами разрешений
* Для того чтобы декодировать, используется capsh --decode=00000000a80425fb

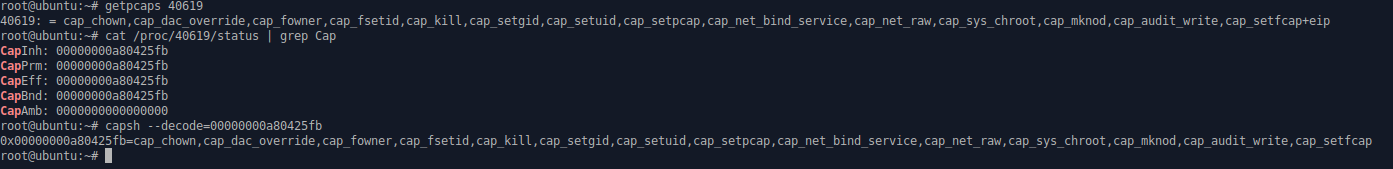


Рисунок 6 – Вывод привилегий

Опишем имеющиеся привилегии (capabilities):

* chown позволяет root изменять владельца или группу любого объекта файловой системы.
* dac\_override позволяет обходить проверки доступа на чтение, запись, выполнение, даже если поля разрешения и владения не позволяют этого делать. Стив Грабб, эксперт по стандартам безопасности в Red Hat, считает, что эта capability не должна использоваться в контейнерах, потому что она используется только в редких случаях в дистрибутивах ОС.
* Fowner позволяет обходить проверки разрешений для операций, которые требуют, чтобы UID файловой системы процесса соответствовал UID файла. В основном это необходимо для инструментов установки программного обеспечения, поэтому использовать этот capability в контейнере не нужно.
* Fsetid позволяет не очищать биты set-user-ID и set-group-ID при изменении файла; устанавливать бит set-group-ID на файл, у которого GID не совпадает с битом файловой системы. В контейнере можно не использовать.
* Kill позволяет root процессу отправлять сигналы уничтожения не root процессам. Если в контейнере все процессы запущены от root, то capability kill не нужна.
* Setgid позволяет изменить GID на любой другой GID. Это обеспечивает полный групповой доступ ко всем файлам в системе. Поэтому она необходима в контейнере.
* Setuid даёт возможность изменить UID на любой другой UID.Это обеспечивает полный доступ ко всем файлам в системе. Поэтому она необходима в контейнере.
* Setpcap позволяет процессу отбросить capabilities или их добавить, но он ограничен ограничивающим набором (Bounding set).
* net\_bind\_service позволяет привязываться к привилегированным портам, то есть портам меньше 1024. Необходимо использовать в контейнере, так как иначе не получиться по ftp.
* net\_raw позволяет следить за пакетами в локальной сети, позволяет использовать сокеты RAW и Packet.
* sys\_chroot позволяет использовать chroot (), то есть позволяет процессам подключаться к другому rootfs.
* mknod позволяет процессу создавать узлы устройств. Контейнеры уже имеют все нужные узлы устройств, поэтому этот capability можно не использовать
* audit\_write позволяет записывать сообщения в журнал аудита ядра. Этим пользуется, например, sudo, то есть доверенное приложение, а приложение, что находится в контейнере может быть не доверенным, поэтому эту capability не нужно использовать.
* Setfcap позволяет устанавливать или удалять любые capability файлов, ограничиваясь ограничивающим набором (Bounding set)

Таблица ниже показывает, какие минимальные привилегии (capabilities) нужно выбрать. Они были выбраны путём проб и ошибок запуска сервера, то есть удалялся параметр и производилось подключение, если ftp работал без проблем, то этот параметр не является критическим и его можно отбросить.

Ниже после таблицы скриншоты, показывающие без каких параметров ftp-сервер, работал некорректно.

|  |  |  |
| --- | --- | --- |
| Capability | Отбросить | Использовать |
| chown | + |  |
| dac\_override | + |  |
| Fowner | + |  |
| Fsetid | + |  |
| Kill | + |  |
| Setgid |  | + |
| Setuid |  | + |
| Setpcap | + |  |
| net\_bind\_service |  | + |
| net\_raw | + |  |
| sys\_chroot |  | + |
| mknod | + |  |
| audit\_write | + |  |
| Setfcap | + |  |

Таблица 1, выбор параметров.

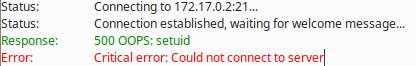


Рисунок 7 – Отбросить setuid не удастся.

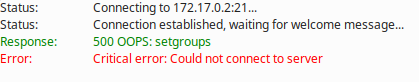


Рисунок 8 – Отбросить setgroups нельзя.

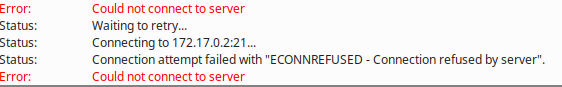


Рисунок 9 - net\_bind\_service отбросить нельзя, так как он слушает порты.

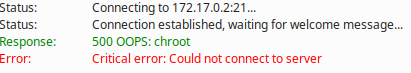


Рисунок 10 - sys\_chroot отбросить нельзя.

Тогда команда запуска контейнера из образа принимает вид:

|  |
| --- |
| root@ubuntu:/opt2/docker-vsftpd# docker run -d --cap-drop=all --cap-add=setgid --cap-add=setuid --cap-add=net\_bind\_service --cap-add=sys\_chroot --name="vsftpd" -h "vsftpd" -p 20:20 -p 21:21 -v /home:/home -v /tmp:/tmp vsftpd:3.0.2; pscap |

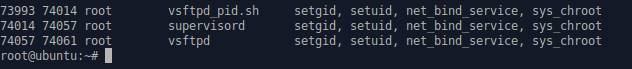


Рисунок 11 – минимальные привилегии(capabilities)

Таким образом, был получен работоспособный минимум привилегий (capabilities) и контейнер был запущен с минимальными привилегиями.

**Seccomp**

Прежде чем ограничивать набор системных вызовов, нужно получить этот набор. Для того чтобы это сделать, можно использовать утилиту strace. Изначально Docker использует профиль по умолчанию. Его вывод можно посмотреть вызовом strace ls.

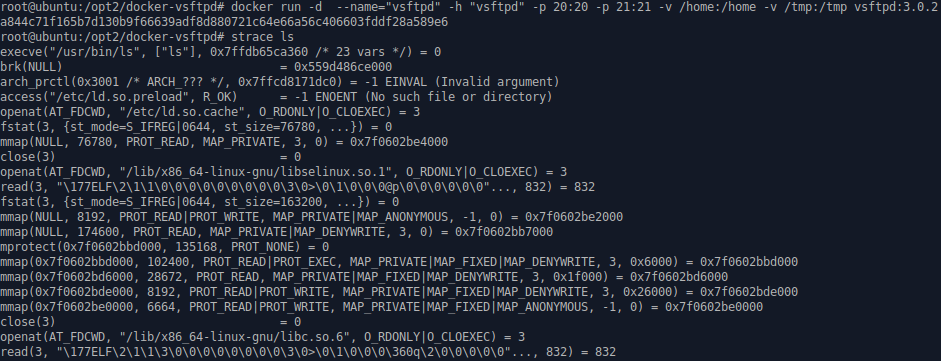


Рисунок12 – Дефолтный профиль.

Поэтому в качестве исходного уменьшаемого профиля seccomp был взят профиль *Docker* по умолчанию.

Запустим контейнер с созданным профилем seccomp и минимальными привилегиями:



Рисунок 13 – Использование собственного профиля docker.json

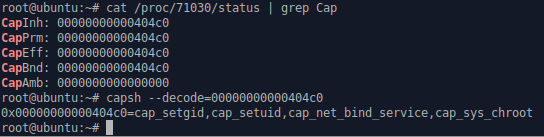


Рисунок 14 – Проверка привилегий (capabilities).

**Программа, работающая с привилегиями и библиотекой libcap-ng**

Для того чтобы программно изменить настройки привелегий, была разработана программа, использующая библиотеку libcap-ng, которая выполняется при запуске контейнера. В ней создаются процессы (execve) для активации демона(сервера). Если в качестве параметра ей передать 0, то она выполнит сброс привилегий и установку только необходимых, вызывая соответствующие функции libcap-ng.

Для работы программы необходимо установить в образ библиотеку libcap-ng, скопировать программу в контейнер и прописать её как ENTRYPOINT. То есть мы изменили первоначальный ENTRYPOINT (/bin/vsftpd\_pid.sh) на нашу программу, которая в свою очередь вызовет первоначальный ENTRYPOINT и запустит демона.

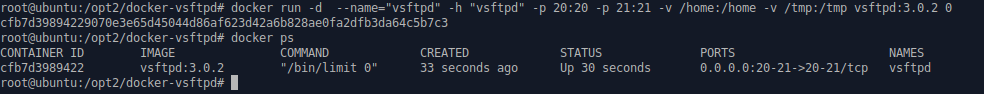


Рисунок 15 – Новый ENTRYPOINT для созданной программы, которая изменяет привелегии.

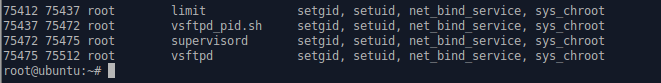


Рисунок 16 – Установлены минимальные привилегии с помощью созданной программы.

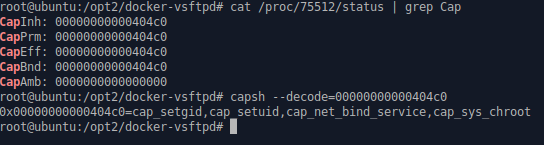


Рисунок 17 – Минимальные привилегии.

**Вывод**

В данной лабораторной работе мы познакомились с концепцией контейнеров. Создали свой собственный образ сервиса, научились ограничивать привелегии процессов в контейнере и познакомились с концепцией seccomp.

Листинг

#

# Dockerfile - vsftpd

#

#

FROM ubuntu:20.04

MAINTAINER Elvin Gasanov

# Last Package Update & Install

RUN apt-get update ; apt-get install -y vsftpd libpam-pwdfile apache2-utils supervisor libcap-ng-dev

RUN mkdir /etc/vsftpd \

&& mkdir -p /var/run/vsftpd/empty \

&& useradd --home /home --gid nogroup -m --shell /bin/false vsftpd

ADD conf/vsftpd.pam /etc/pam.d/vsftpd

ADD conf/vsftpd.conf /etc/vsftpd.conf

ADD conf/vsftpd\_vuser.sh /bin/vsftpd\_vuser.sh

RUN chmod a+x /bin/vsftpd\_vuser.sh

# Supervisor

RUN mkdir -p /var/log/supervisor

ADD conf/supervisord.conf /etc/supervisor/conf.d/supervisord.conf

ADD conf/vsftpd\_pid.sh /bin/vsftpd\_pid.sh

ADD conf/limit /bin/limit

RUN chmod a+x /bin/vsftpd\_pid.sh

# Port

EXPOSE 20 21

# Daemon

ENTRYPOINT ["/bin/limit"]

#ENTRYPOINT ["/bin/vsftpd\_pid.sh"]

#CMD ["/usr/bin/supervisord"]

**Json**

{

"defaultAction": "SCMP\_ACT\_ERRNO",

"archMap": [

{

"architecture": "SCMP\_ARCH\_X86\_64",

"subArchitectures": [

"SCMP\_ARCH\_X86",

"SCMP\_ARCH\_X32"

]

},

{

"architecture": "SCMP\_ARCH\_AARCH64",

"subArchitectures": [

"SCMP\_ARCH\_ARM"

]

},

{

"architecture": "SCMP\_ARCH\_MIPS64",

"subArchitectures": [

"SCMP\_ARCH\_MIPS",

"SCMP\_ARCH\_MIPS64N32"

]

},

{

"architecture": "SCMP\_ARCH\_MIPS64N32",

"subArchitectures": [

"SCMP\_ARCH\_MIPS",

"SCMP\_ARCH\_MIPS64"

]

},

{

"architecture": "SCMP\_ARCH\_MIPSEL64",

"subArchitectures": [

"SCMP\_ARCH\_MIPSEL",

"SCMP\_ARCH\_MIPSEL64N32"

]

},

{

"architecture": "SCMP\_ARCH\_MIPSEL64N32",

"subArchitectures": [

"SCMP\_ARCH\_MIPSEL",

"SCMP\_ARCH\_MIPSEL64"

]

},

{

"architecture": "SCMP\_ARCH\_S390X",

"subArchitectures": [

"SCMP\_ARCH\_S390"

]

}

],

"syscalls": [

{

"names": [

"accept",

"bind",

"brk",

"capget",

"capset",

"chdir",

"close",

"connect",

"dup",

"dup2",

"epoll\_pwait",

"execve",

"exit\_group",

"fchmod",

"fcntl",

"fork",

"fstat",

"fstatfs",

"futex",

"getcwd",

"getdents64",

"getegid",

"geteuid",

"getgid",

"getpeername",

"getpgid",

"getpid",

"getppid",

"getrandom",

"getsockname",

"gettid",

"getuid",

"ioctl",

"listen",

"lseek",

"mmap",

"mprotect",

"newfstatat",

"open",

"openat",

"pipe",

"poll",

"prctl",

"prlimit64",

"read",

"readlink",

"rename",

"rt\_sigaction",

"rt\_sigprocmask",

"rt\_sigreturn",

"select",

"sendto",

"setgid",

"setgroups",

"setresgid",

"setresuid",

"setsid",

"setsockopt",

"set\_tid\_address",

"setuid",

"socket",

"socketpair",

"stat",

"umask",

"uname",

"wait4",

"write",

"writev"

],

"action": "SCMP\_ACT\_ALLOW",

"args": [],

"comment": "",

"includes": {},

"excludes": {}

},

{

"names": [

"arch\_prctl"

],

"action": "SCMP\_ACT\_ALLOW",

"args": [],

"comment": "",

"includes": {

"arches": [

"amd64",

"x32"

]

},

"excludes": {}

},

{

"names": [

"chroot"

],

"action": "SCMP\_ACT\_ALLOW",

"args": [],

"comment": "",

"includes": {

"caps": [

"CAP\_SYS\_CHROOT"

]

},

"excludes": {}

}

]}

**Limit.c**

#include <cap-ng.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main(int argc, char \*argv[], char \*envp[])

{

if (2 != argc || 1 != strlen(argv[1]) || '0' != argv[1][0])

{

}

else

{

capng\_clear(CAPNG\_SELECT\_BOTH);

capng\_updatev(CAPNG\_ADD, (CAPNG\_EFFECTIVE | CAPNG\_PERMITTED | CAPNG\_INHERITABLE | CAPNG\_BOUNDING\_SET), CAP\_SETGID , CAP\_NET\_BIND\_SERVICE , CAP\_SYS\_CHROOT , CAP\_SETUID, -1);

capng\_apply(CAPNG\_SELECT\_BOTH);

}

char \*const argv\_1[] = { "/bin/vsftpd\_pid.sh", NULL };

char \*const argv\_2[] = { "/bin/vsftpd\_pid.sh", NULL };

pid\_t childPID = fork();

if (!childPID)

{

execve("/bin/vsftpd\_pid.sh", argv\_1, envp);

}

else if (0 < childPID)

{

wait(NULL);

execve("/bin/vsftpd\_pid.sh", argv\_2, envp);

}

return 0;

}