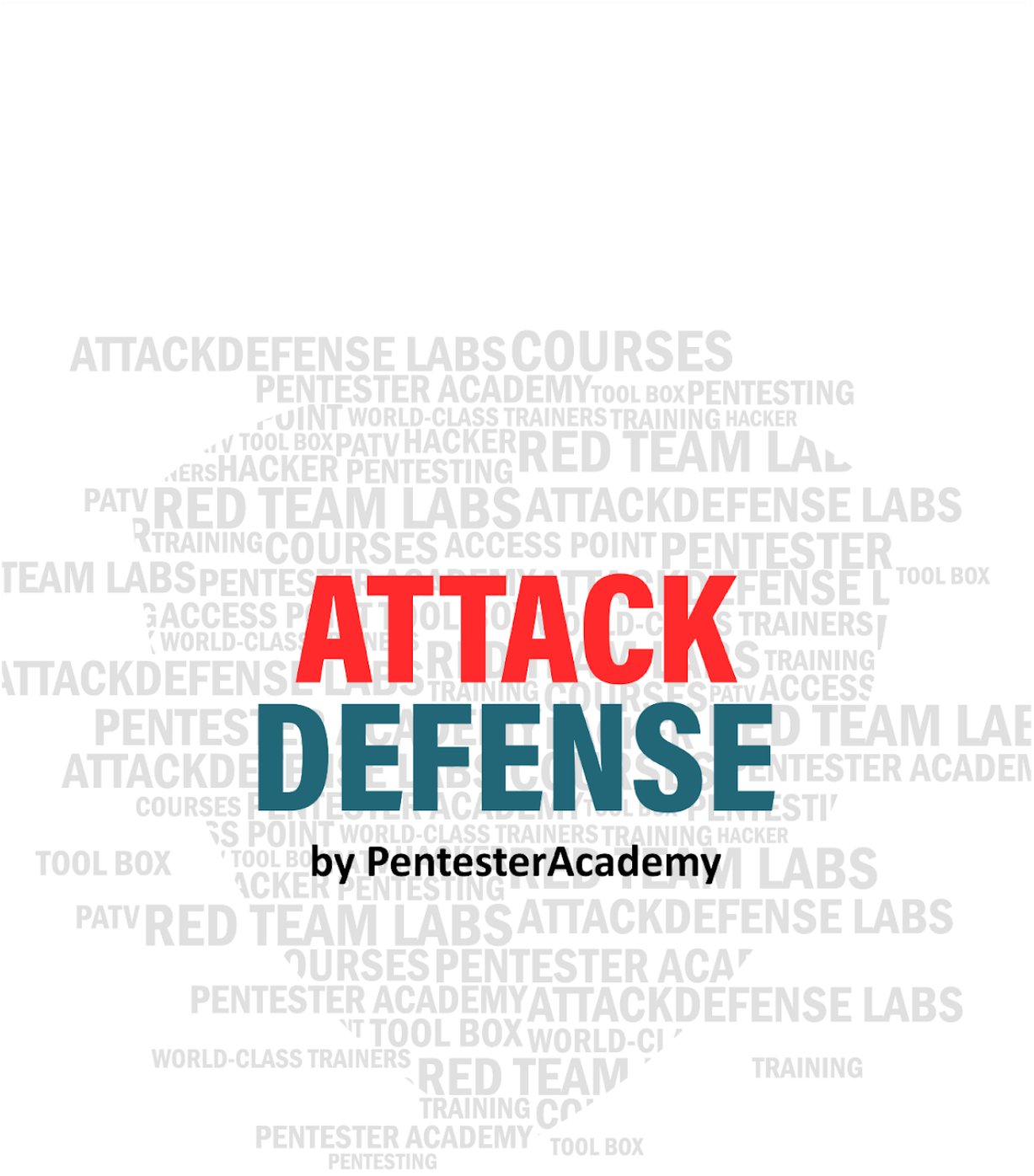

ATTACK
DEFENSE
by PentesterAcademy

Name	Docker Seccomp Profile
URL	https://attackdefense.com/challengedetails?cid=1827
Type	Privilege Escalation : Seccomp

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Objective: Learn how to use Seccomp profiles with Docker by editing the default docker seccomp profile file to achieve the following:

a. **Block the usage of uname command**
b. **Check the syscalls being made by a binary/command**
c. **Block a process from listening on a socket**

Solution:

Step 1: First we need to stop the AppArmor service so it doesn't interfere.

Command: /etc/init.d/apparmor stop

```
root@localhost:~#
root@localhost:~# /etc/init.d/apparmor stop
[ ok ] Stopping apparmor (via systemctl): apparmor.service.
root@localhost:~#
```

Step 2: Also drop all AppArmor profiles and check the status to verify the action.

Commands:
/etc/init.d/apparmor teardown
aa-status

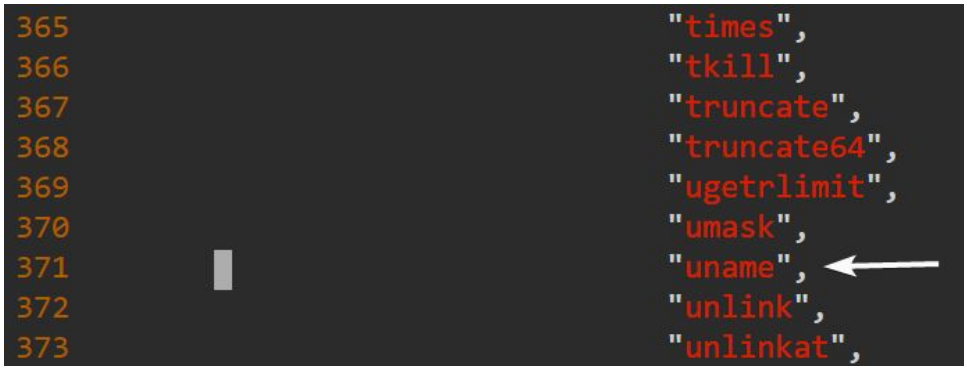
```
root@localhost:~# /etc/init.d/apparmor teardown
 * Unloading AppArmor profiles
root@localhost:~# aa-status
apparmor module is loaded.
0 profiles are loaded.
0 profiles are in enforce mode.
0 profiles are in complain mode.
0 processes have profiles defined.
0 processes are in enforce mode.
0 processes are in complain mode.
0 processes are unconfined but have a profile defined.
root@localhost:~#
```

Now AppArmor is down and the user can focus on seccomp.

Objective a: Blocking usage of uname command.

Step 1: The default seccomp profile is provided in the home directory of the root user (i.e. /root/default.json). Open the file and check the permitted system calls.

Command: vim default.json (use :set nu to number the lines)



On line 371, the permission to make uname syscall is there. Remove this line, so the file will look like:

And, save the file.

Step 2: Run ubuntu:18.04 docker image without specifying seccomp profile and try to run the uname command.

Commands:
docker run -it ubuntu:18.04 bash
uname -a

```
root@localhost:~# docker run -it ubuntu:18.04 bash
root@bde29c085985:/#
root@bde29c085985:/# uname -a
Linux bde29c085985 5.0.0-20-generic #21-Ubuntu SMP Mon Jun 24 09:32:09 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
root@bde29c085985:/#
root@bde29c085985:/#
```

This container is running with default seccomp profile and hence the command worked.

Step 3: Run ubuntu:18.04 docker image with modified default docker seccomp profile and try to run uname command.

Commands:
docker run -it --security-opt seccomp=default.json ubuntu:18.04 bash
uname -a

```
root@localhost:~# docker run -it --security-opt seccomp=default.json ubuntu:18.04 bash
root@??host??:/#
root@??host??:/# uname -a
uname: cannot get system name: Operation not permitted
root@??host??:/#
```

The permission for uname syscall was removed from the default profile. Hence, the command didn't work.

Objective b: Check the syscalls being made by a binary/command

Step 1: The strace command can be used to see all the system calls made by a binary/command. For example, netcat started in TCP listen mode.

Command: docker run -it modified-ubuntu strace nc -l 9000

```
root@localhost:~# docker run -it modified-ubuntu strace nc -l 9000
execve("/bin/nc", ["nc", "-l", "9000"], 0x7fff8969f590 /* 4 vars */) = 0
brk(NULL)                               = 0x56130d51c000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=18708, ...}) = 0
mmap(NULL, 18708, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f9dedeb3000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\34\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2030544, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f9dedeb1000
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f9ded8a0000
mprotect(0x7f9deda87000, 2097152, PROT_NONE) = 0
mmap(0x7f9dedc87000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f9dedc87000
mmap(0x7f9dedc8d000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f9dedc8d000
close(3)                                = 0
```

```
arch_prctl(ARCH_SET_FS, 0x7f9dedeb2540) = 0
mprotect(0x7f9dedc87000, 16384, PROT_READ) = 0
mprotect(0x56130ce62000, 4096, PROT_READ) = 0
mprotect(0x7f9dedeb8000, 4096, PROT_READ) = 0
munmap(0x7f9dedeb3000, 18708)           = 0
getpid()                                = 7
stat("/etc/resolv.conf", {st_mode=S_IFREG|0644, st_size=698, ...}) = 0
brk(NULL)                               = 0x56130d51c000
brk(0x56130d53d000)                     = 0x56130d53d000
openat(AT_FDCWD, "/etc/host.conf", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=92, ...}) = 0
read(3, "# The \"order\" line is only used "..., 4096) = 92
read(3, "", 4096)                       = 0
close(3)                                = 0
openat(AT_FDCWD, "/etc/resolv.conf", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=698, ...}) = 0
read(3, "# This file is managed by man:sy"..., 4096) = 698
read(3, "", 4096)                       = 0
close(3)                                = 0
uname({sysname="Linux", nodename="d36c3836fa92", ...}) = 0
getpid()                                = 7
```

```
rt_sigaction(SIGINT, {sa_handler=0x56130cc5eca0, sa_mask=[INT], sa_flags=SA_RESTORER|SA_RESTART, sa_restorer=0x7f9ded8def20}, {sa_handler=SIG_DF
L, sa_mask=[], sa_flags=0}, 8) = 0
rt_sigaction(SIGQUIT, {sa_handler=0x56130cc5eca0, sa_mask=[QUIT], sa_flags=SA_RESTORER|SA_RESTART, sa_restorer=0x7f9ded8def20}, {sa_handler=SIG_
DFL, sa_mask=[], sa_flags=0}, 8) = 0
rt_sigaction(SIGTERM, {sa_handler=0x56130cc5eca0, sa_mask=[TERM], sa_flags=SA_RESTORER|SA_RESTART, sa_restorer=0x7f9ded8def20}, {sa_handler=SIG_
DFL, sa_mask=[], sa_flags=0}, 8) = 0
rt_sigaction(SIGURG, {sa_handler=SIG_IGN, sa_mask=[URG], sa_flags=SA_RESTORER|SA_RESTART, sa_restorer=0x7f9ded8def20}, {sa_handler=SIG_DFL, sa_m
ask=[], sa_flags=0}, 8) = 0
rt_sigaction(SIGPIPE, {sa_handler=SIG_IGN, sa_mask=[PIPE], sa_flags=SA_RESTORER|SA_RESTART, sa_restorer=0x7f9ded8def20}, {sa_handler=SIG_DFL, sa
_mask=[], sa_flags=0}, 8) = 0
```

```
socket(AF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
setsockopt(3, SOL_SOCKET, SO_REUSEPORT, [1], 4) = 0
listen(3, 1)                            = 0
rt_sigaction(SIGALRM, {sa_handler=SIG_IGN, sa_mask=[ALRM], sa_flags=SA_RESTORER|SA_RESTART, sa_restorer=0x7f9ded8def20}, {sa_handler=SIG_DFL, sa
_mask=[], sa_flags=0}, 8) = 0
alarm(0)                                = 0
rt_sigprocmask(SIG_BLOCK, NULL, [], 8)  = 0
accept(3, 0x56130d51c630, [16])         = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
```

In this manner, system calls can be checked for other binaries/executables.

Objective c: Block a process from listening on a socket.

Step 1: From above steps, it is clear that listen() call is used to open a listening socket. The default seccomp profile is provided in the home directory of the root user (i.e. /root/default.json). Open the file and check the permitted system calls. Also, please add the “uname“ syscall at line number 371 (deleted in objective a).

Command: vim default.json (use :set nu to number the lines)

On line 371, add the permission to perform uname syscall.

Remove the listen syscall from line 185

After removing the line:

Save the file.

Step 2: Run ubuntu:18.04 docker image without specifying seccomp profile and try to run the netcat listen command.

Command: docker run -it modified-ubuntu nc -l 9000

```
root@localhost:~# docker run -it modified-ubuntu nc -l 9000
```

This container is running with default seccomp profile and hence the command worked.

Step 3: Run ubuntu:18.04 docker image with modified default docker seccomp profile and and try to run the netcat listen command.

Command: docker run -it --security-opt seccomp=default.json ubuntu:18.04 nc -l 9000

```
root@localhost:~# docker run -it --security-opt seccomp=default.json modified-ubuntu nc -l 9000
local listen fuxored : Operation not permitted
root@localhost:~#
```

The permission for listen syscall was removed from the default profile. Hence, the command didn't work.

If the same command is run with strace, access denied to listen() syscall will be there.

Command: docker run -it --security-opt seccomp=default.json modified-ubuntu strace nc -l 9000

```
socket(AF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
setsockopt(3, SOL_SOCKET, SO_REUSEPORT, [1], 4) = 0
listen(3, 1)                            = -1 EPERM (Operation not permitted)
write(2, "local listen fuxored", 20local listen fuxored)    = 20
write(2, " : Operation not permitted\n", 27 : Operation not permitted
) = 27
close(-1)                               = -1 EBADF (Bad file descriptor)
exit_group(1)                           = ?
+++ exited with 1 +++
root@localhost:~#
```

References:

- Seccomp (http://man7.org/linux/man-pages/man2/seccomp.2.html)
- Difference between seccomp and linux capabilities (https://security.stackexchange.com/questions/210400/difference-between-linux-capabilities-and-seccomp)
- How Sandboxing is implemented (https://security.stackexchange.com/questions/168452/how-is-sandboxing-implemented/175373#175373)