**Lab5**

Ziyang Lin zlin32@jhu.edu

**Using cgroups to deliver the exploit**

This example comes from Trail of Bits Blog.[[1]](#footnote-1) This exploit sets notify\_no\_release falg to 1 and run command in the release\_agent file. This command is run as a fully privileged root on the host. They specified the requirements for this exploit as follows:

1. We must be running as root inside the container
2. The container must be run with the SYS\_ADMIN Linux capability
3. The container must lack an AppArmor profile, or otherwise allow the mount syscall
4. The cgroup v1 virtual filesystem must be mounted read-write inside the container

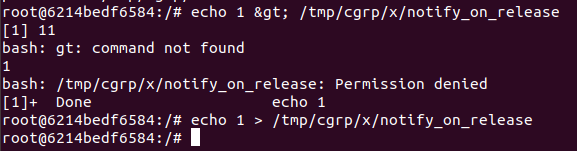
Thus, I run our docker as mentioned in blog.



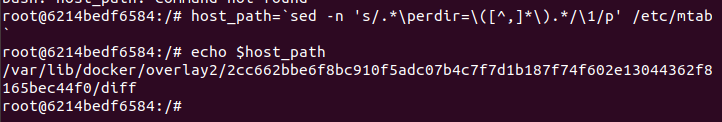
In the container, I create the cgroup’s release\_agent file which will execute script after all cgroup tasks are killed. Then, mounting the RDMA cgroup and creating a child cgroup, named x.



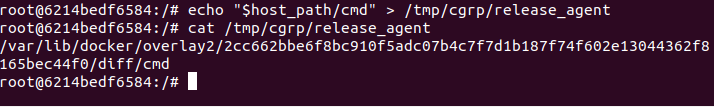
Then, I set the notify\_on\_release flag to 1.



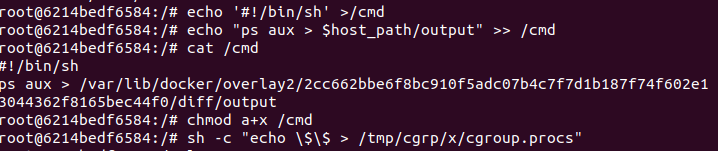
This can make system finding the container’s directory as specified in the /etc/mtab file on the host system.



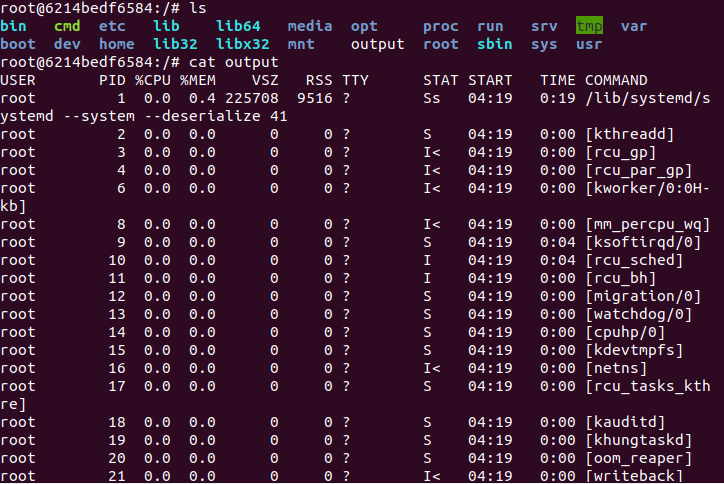
And placing the full path in the release\_agent file inside the container.



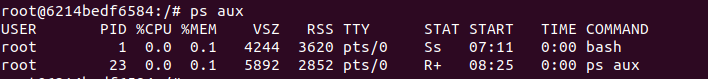
I write our /cmd script and execute by killing the cgroup’s tasks. it will execute the ps aux command and save its output into /output on the container



I can see the output file which lists the host system’s processes.



Using ps aux in docker directly will be like this.



1. Trail of Bits. 2019. Understanding Docker Container Escapes. Trail of Bits. Accessed Oct. 20, 2020. https://blog.trailofbits.com/2019/07/19/understanding-docker-container-escapes/ [↑](#footnote-ref-1)