Specialist: Hacking and Securing Cloud

# Answer

## Part 4

https://t.me/learningnets

# Contents

**Defence** ..... **2**
K8s: Defence 1 ..... 2
Solution ..... 2
K8s: Defence 2 ..... 12
Solution ..... 12
K8s: Defence 3 ..... 22
Solution ..... 22
**Identify:** ..... **28**
Solution ..... 28
**Identify:** ..... **39**
Audit 1: ..... 39
Solution ..... 39
Audit 2: ..... 44
Solution ..... 44
Audit 3: ..... 50
Solution ..... 50
Audit 4: ..... 52
Solution ..... 52


© Claranet Cyber Security. All rights reserved
https://t.me/learningnets

# Defence

## K8s: Defence 1

Run a local kubernetes cluster with kyverno, policy-reporter and Kube-prometheus

- Use kyverno to enforce pod security
- Use policy reporter to send policy violation reports to prometheus
- Use Grafana to view graphical reports policy violation.

### Solution

Policy Management in Kubernetes

Kubernetes inherently lacks any process to restrict resources based on policies. So to enable support for policy based restrictions 'webhooks' are used. There are multiple projects utilize this technique to enable policy-based validation of resources in kubernetes.

Some of the most well-known projects are

- Gatekeeper
- Kubewarden
- Kyverno

Gatekeeper is one of the popular policy engine in kubernetes community which uses webhooks to impose policies in kubernetes cluster, it is capable of creating complex policies with the use of policy language called 'rego'. So, to write policies in Gatekeeper knowledge of 'rego' is mandatory.

Kubewarden is also uses webhooks to impose policies in the kubernetes cluster, but it is not dependent on any particular language for policy creation as all the policy code is converted to web assembly and then implemented by the kubewarden. so to create policies in kubewarden you don't need any particular language but we have to write code that could be complied in web assembly binary for execution.

Kyverno is Kubernetes native policy engine which uses webhooks to impose policies in the Kubernetes cluster and the process of policy creation is similar to yaml files used by other resources in the cluster. so the prerequisites of using Kyverno is to have familiarity with yaml files used in kubernetes. Hence, we will be using Kyverno in our course for understanding policy engine.

Setting up Kyverno policy engine with Prometheus and Policy reporter

Navigate to the folder where we have installed our scripts.

**Command:**



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

```
cd /home/pentester/k8sScript/kyverno
```

Run the script to setup the cluster.

### Command:

```
./kyverno_policy-reporter_promethous.sh
```

Verify the cluster

### Command:

```
kubectl get pods -A
```

### Output:

```
NAMESPACE            NAME                                                    READY
STATUS    RESTARTS   AGE
kube-system          coredns-558bd4d5db-svrs8                                1/1
Running   0          6m21s
kube-system          coredns-558bd4d5db-vhs7d                                1/1
Running   0          6m20s
kube-system          etcd-kind-control-plane                                 1/1
Running   0          6m30s
kube-system          kindnet-clvh2                                           1/1
Running   0          6m9s
kube-system          kindnet-xk9bp                                           1/1
Running   0          6m21s
kube-system          kube-apiserver-kind-control-plane                       1/1
Running   0          6m30s
kube-system          kube-controller-manager-kind-control-plane              1/1
Running   0          6m30s
kube-system          kube-proxy-2xtfl                                        1/1
Running   0          6m9s
kube-system          kube-proxy-chkkb                                        1/1
Running   0          6m21s
kube-system          kube-scheduler-kind-control-plane                       1/1
Running   0          6m30s
kyverno              kyverno-6bbc7c45c8-8kzrs                                1/1
Running   0          4m4s
local-path-storage   local-path-provisioner-547f784dff-bnj2v                 1/1
Running   0          6m20s
monitoring           alertmanager-prometheus-kube-prometheus-alertmanager-0  2/2
Running   0          5m13s
monitoring           prometheus-grafana-65b46787f7-wlxfq                     3/3
Running   0          5m39s
monitoring           prometheus-kube-prometheus-operator-f4bcfb987-kvsdg     1/1
Running   0          5m39s
monitoring           prometheus-kube-state-metrics-57c988498f-dj579          1/1
Running   0          5m39s
monitoring           prometheus-prometheus-kube-prometheus-prometheus-0      2/2
Running   0          5m13s
monitoring           prometheus-prometheus-node-exporter-9w9bj               1/1
Running   0          5m39s
monitoring           prometheus-prometheus-node-exporter-zl7lt               1/1
Running   0          5m39s
policy-reporter      policy-reporter-fcd57bfc4-wpxhh                         1/1
Running   0          3m30s
policy-reporter      policy-reporter-kyverno-plugin-548b548b56-8mhfr         1/1
Running   0          3m30s
policy-reporter      policy-reporter-ui-847fc4bcdc-mrk6r                     1/1
Running   0          3m30s
```

### Validating Policy



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Let's use a prebuilt policy to notify when a privileged containers is created in the cluster called Disallow Privileged Containers.

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-privileged-containers
  annotations:
    policies.kyverno.io/title: Disallow Privileged Containers
    policies.kyverno.io/category: Pod Security Standards (Baseline)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    kyverno.io/kyverno-version: 1.6.0
    kyverno.io/kubernetes-version: "1.22-1.23"
    policies.kyverno.io/description: >-
      Privileged mode disables most security mechanisms and must not be allowed. This
policy
      ensures Pods do not call for privileged mode.
spec:
  validationFailureAction: audit
  background: true
  rules:
    - name: privileged-containers
      match:
        any:
        - resources:
            kinds:
              - Pod
      validate:
        message: >-
          Privileged mode is disallowed. The fields
spec.containers[*].securityContext.privileged
          and spec.initContainers[*].securityContext.privileged must be unset or set to
`false`.
        pattern:
          spec:
            =(ephemeralContainers):
              - =(securityContext):
                  =(privileged): "false"
            =(initContainers):
              - =(securityContext):
                  =(privileged): "false"
            containers:
              - =(securityContext):
                  =(privileged): "false"
```

This policy checks for patterns in resource definition file and restricts/warns any resource violating this policy.

***(Note: Kyverno will warn when "validationFailureAction" parameter is set to 'audit' and restrict when its set to 'enforce')***

Let's apply the policy in our cluster.

**Command:**

```
kubectl apply -f disallow-privileged-containers.yaml
```

**Output:**

```
clusterpolicy.kyverno.io/disallow-privileged-containers created
```



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Now let's create a pod which contains a privileged container using a prebuild pod definition file **priv-exec-pod.yaml**

```
# cat priv-exec-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: priv-exec-pod
  labels:
    app: pentest
spec:
  containers:
  - name: priv-pod
    image: ubuntu
    securityContext:
      privileged: true
    command: [ "/bin/sh", "-c", "--" ]
    args: [ "while true; do sleep 30; done;" ]
  #nodeName: k8s-control-plane-node # Force your pod to run on the control-plane node by
uncommenting this line and changing to a control-plane node name
```

Create pod using the above YAML file.

**Command:**

```
kubectl apply -f priv-exec-pod.yaml
```

**Output:**

```
pod/priv-exec-pod created
```

As our policy was in audit mode privileged pod was allowed to be created but lets check the policy report.

**Command:**

```
kubectl get polr -A
```

**Output:**

```
NAMESPACE            NAME                           PASS   FAIL   WARN   ERROR   SKIP   AGE
default              polr-ns-default                0      1      0      0       0      5m8s
local-path-storage   polr-ns-local-path-storage     1      0      0      0       0      12m
monitoring           polr-ns-monitoring             6      0      0      0       0      12m
policy-reporter      polr-ns-policy-reporter        3      0      0      0       0      12m
```

Let's check more details of failed policy.

**Command:**

```
kubectl describe polr  polr-ns-default
```

**Output:**

```
Name:         polr-ns-default
Namespace:    default
Labels:       managed-by=kyverno
Annotations:  <none>
API Version:  wgpolicyk8s.io/v1alpha2
Kind:         PolicyReport
Metadata:
  Creation Timestamp:  2022-02-22T13:01:32Z
  Generation:          1
```



NotSoSecure part of claranet cyber security

© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

```
  Managed Fields:
    API Version:  wgpolicyk8s.io/v1alpha2
    Fields Type:  FieldsV1
    fieldsV1:
      f:metadata:
        f:labels:
          .:
          f:managed-by:
        f:ownerReferences:
          .:
          k:{"uid":"7880560b-3042-47b9-9ab1-312d7710c006"}:
            .:
            f:apiVersion:
            f:blockOwnerDeletion:
            f:controller:
            f:kind:
            f:name:
            f:uid:
      f:results:
      f:summary:
        .:
        f:error:
        f:fail:
        f:pass:
        f:skip:
        f:warn:
    Manager:    kyverno
    Operation:  Update
    Time:       2022-02-22T13:01:32Z
  Owner References:
    API Version:           v1
    Block Owner Deletion:  true
    Controller:            true
    Kind:                  Namespace
    Name:                  default
    UID:                   7880560b-3042-47b9-9ab1-312d7710c006
  Resource Version:        23514
  UID:                     4402cfa3-8ebb-4fdd-95e6-3677b9f9edcf
Results:
  Category:  Pod Security Standards (Baseline)
  Message:   validation error: Privileged mode is disallowed. The fields
spec.containers[*].securityContext.privileged and
spec.initContainers[*].securityContext.privileged must be unset or set to `false`. Rule
privileged-containers failed at path /spec/containers/0/securityContext/privileged/
  Policy:    disallow-privileged-containers
  Resources:
    API Version:  v1
    Kind:         Pod
    Name:         priv-exec-pod
    Namespace:    default
    UID:          20748406-9215-4b18-a5b1-d328a4225862
  Result:         fail
  Rule:           privileged-containers
  Scored:         true
  Severity:       medium
  Source:         Kyverno
  Timestamp:
    Nanos:    0
    Seconds:  1645534890
Summary:
  Error:  0
  Fail:   1
  Pass:   0
  Skip:   0
  Warn:   0
Events:   <none>
```



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

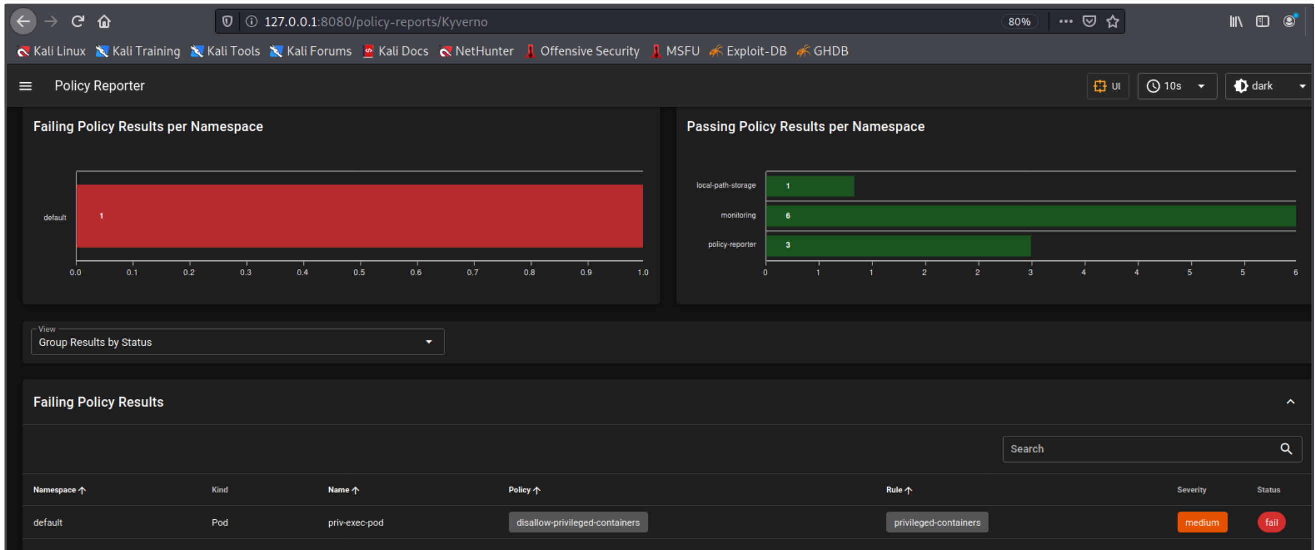
We can check this instance in policy-reporter by port-fowrding policy-reporter-ui service to localhost.

**Command:**

```
kubectl port-forward -n policy-reporter svc/policy-reporter-ui 8080
```

**Output:**

```
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```




This can also be viewed in grafana-dashboard by port-forwarding grafana server to localhost.

**Command:**

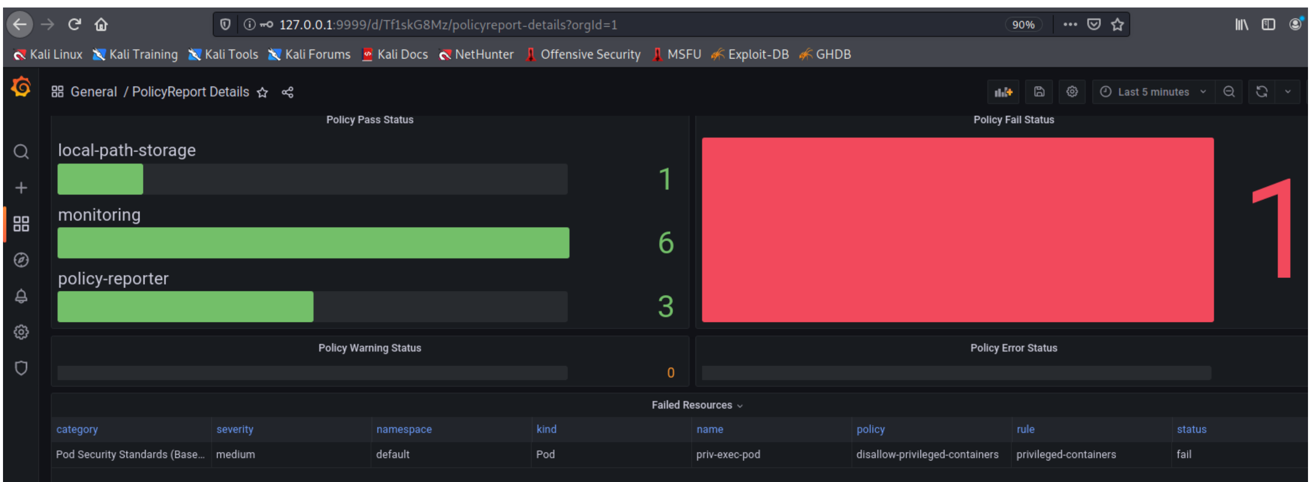
```
kubectl port-forward -n monitoring svc/prometheus-grafana 9999:80
```

**Output:**

```
Forwarding from 127.0.0.1:9999 -> 3000
Forwarding from [::1]:9999 -> 3000
```

Credentials for Grafana dashboard are as follow:

username: **admin** and password: **prom-operator**






NotSoSecure part of claranet cyber security

© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Let's remove the privileged pod from the cluster.

### Command:

```
kubectl get pod
```

### Output:

```
NAME            READY   STATUS    RESTARTS   AGE
priv-exec-pod   1/1     Running   0          74m
# kubectl delete pod priv-exec-pod
pod "priv-exec-pod" deleted
```

Now that we have visibility via policy report, lets update this policy to enforce mode.

### Command:

```
nano disallow-privileged-containers.yaml
```

### Output:

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-privileged-containers
  annotations:
    policies.kyverno.io/title: Disallow Privileged Containers
    policies.kyverno.io/category: Pod Security Standards (Baseline)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    kyverno.io/kyverno-version: 1.6.0
    kyverno.io/kubernetes-version: "1.22-1.23"
    policies.kyverno.io/description: >-
      Privileged mode disables most security mechanisms and must not be allowed. This
policy
      ensures Pods do not call for privileged mode.
spec:
  validationFailureAction: enforce
  background: true
  rules:
    - name: privileged-containers
      match:
        any:
        - resources:
            kinds:
              - Pod
      validate:
        message: >-
          Privileged mode is disallowed. The fields
spec.containers[*].securityContext.privileged
          and spec.initContainers[*].securityContext.privileged must be unset or set to
`false`.
        pattern:
          spec:
            =(ephemeralContainers):
              - =(securityContext):
                  =(privileged): "false"
            =(initContainers):
              - =(securityContext):
                  =(privileged): "false"
            containers:
              - =(securityContext):
                  =(privileged): "false"
```



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Now apply the policy in our cluster.

**Command:**

```
kubectl apply -f disallow-privileged-containers.yaml
```

**Output:**

```
clusterpolicy.kyverno.io/disallow-privileged-containers configured
```

Let's again try to create a privileged pod using same configuration as earlier.

**Command:**

```
kubectl apply -f priv-exec-pod.yaml
# Note: Ensure that you are in the same folder from where you have created the YAML file
in earlier steps.
```

**Output:**

As our policy was in audit mode privileged pod was allowed to be created but lets check the policy report.

```
Error from server: error when creating "priv-exec-pod.yaml": admission webhook
"validate.kyverno.svc-fail" denied the request:

resource Pod/default/priv-exec-pod was blocked due to the following policies

disallow-privileged-containers:
  privileged-containers: 'validation error: Privileged mode is disallowed. The fields
    spec.containers[*].securityContext.privileged and
spec.initContainers[*].securityContext.privileged
    must be unset or set to `false`. Rule privileged-containers failed at path
/spec/containers/0/securityContext/privileged/'
```

Our request gets denied due to policy violation (Note: Kyverno do not provide logs for failed/restricted resources)

**Mutating Policy**

Kyverno is also capable of creating policies that can intercept the resource request and modify them before sending it to validation module.

in this senario we will use a policy that mutates the pod creation request and adds default security context before sending it to validation module.

We will be using add-default-securitycontext policy. This will ensure that containers created by the pods are not running as root user.

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: add-default-securitycontext
  annotations:
    policies.kyverno.io/title: Add Default securityContext
    policies.kyverno.io/category: Sample
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
```



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

```
      A Pod securityContext entry defines fields such as the user and group which should
be used to run the Pod.
      Sometimes choosing default values for users rather than blocking is a better
alternative to not impede
      such Pod definitions. This policy will mutate a Pod to set `runAsNonRoot`,
runAsUser`, `runAsGroup`, and `fsGroup` fields
      within the Pod securityContext if they are not already set.
spec:
  rules:
  - name: add-default-securitycontext
    match:
      resources:
        kinds:
        - Pod
    mutate:
      patchStrategicMerge:
        spec:
          securityContext:
            +(runAsNonRoot): true
            +(runAsUser): 1000
            +(runAsGroup): 3000
            +(fsGroup): 2000
```

Let's apply this policy in our cluster.

**Command:**

```
kubectl apply -f add-default-securitycontext.yaml
```

**Output:**

```
clusterpolicy.kyverno.io/add-default-securitycontext created
```

Now let's try to create a ubuntu pod without specifying any security context as shown below.

```
# cat ubuntu.yaml
apiVersion: v1
kind: Pod
metadata:
  name: ubuntu
spec:
  containers:
  - name: ubuntu
    image: ubuntu
    command: ["sleep"]
    args: ["99d"]
```

Let's create the ubuntu pod.

**Command:**

```
kubectl apply -f ubuntu.yaml
```

**Output:**

```
pod/ubuntu created
```

Now let's check if security context is added to ubuntu pod.

**Command:**

```
kubectl get pod ubuntu -o yaml
```

**Output:**



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

```
apiVersion: v1
kind: Pod
metadata:
---------------------------------snipped-------------------------------------------
  name: ubuntu
  namespace: default
  resourceVersion: "54938"
  uid: b997c7c5-b705-4eb0-85fc-ba0662080849
spec:
  containers:
  - args:
    - 99d
    command:
    - sleep
    image: ubuntu
    imagePullPolicy: Always
    name: ubuntu
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: kube-api-access-6mcqg
      readOnly: true
  dnsPolicy: ClusterFirst
  enableServiceLinks: true
  nodeName: kind-worker
  preemptionPolicy: PreemptLowerPriority
  priority: 0
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext:
    fsGroup: 2000
    runAsGroup: 3000
    runAsNonRoot: true
    runAsUser: 1000
---------------------------------------------snipped--------------------------------------
  hostIP: 172.18.0.2
  phase: Running
  podIP: 10.244.1.15
  podIPs:
  - ip: 10.244.1.15
  qosClass: BestEffort
  startTime: "2022-02-22T16:04:40Z"
```


So, we can observed that the security context is added to the ubuntu pod. let verify same via shell access in pod

```
# kubectl exec -it ubuntu -- bash
groups: cannot find name for group ID 3000
groups: cannot find name for group ID 2000
I have no name!@ubuntu:/$ id
uid=1000 gid=3000 groups=3000,2000
```

Let's destroy the cluster.

**Command:**

```
./destroy-kyverno_policy-reporter_promethous.sh
```

To verify if cluster is deleted, use following command

```
# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

you should see no docker containers running.

© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

# K8s: Defence 2

Run a local kubernetes cluster with Cilium CNI along with Hubble plugin enabled

- Observe the traffic flow in the cluster with Hubble CLI and GUI
- Apply Layer 3 and layer 7 policies in the kubernetes cluster to manage traffic in the cluster

## Solution

In this exercise we will monitor network traffic of kubernetes cluster and learn how to generate and apply network policies

Components that we will be using in this exercise are:

- kind is a tool for running local Kubernetes clusters using Docker container 'nodes'.
- Cilium is a **eBPF** based CNI that could be used to observe network traffic in the kubernetes cluster and implement network policies in it.
- Hubble is a part of **Cilium** CNI which has CLI and GUI based interface to observe network traffic.

Navigate to the folder where we have installed our scripts.

**Command:**

```
cd /home/pentester/k8sScript/cilium
```

Run the script to setup the cluster.

**Command:**

```
cilium-hubble-cluster.sh
```

**Command:**

```
kubectl get pod -A
```

**Output:**

```
NAMESPACE           NAME                                                          READY
STATUS    RESTARTS  AGE
default             deathstar-7fb98564d4-4xwm8                                    1/1
Running   0         159m
default             deathstar-7fb98564d4-hj6rl                                    1/1
Running   0         159m
default             tiefighter                                                    1/1
Running   0         159m
default             xwing                                                         1/1
Running   0         159m
kube-system         cilium-n7jmq                                                  1/1
Running   0         160m
kube-system         cilium-operator-6bbdb895b5-2fsvm                              1/1
Running   0         162m
kube-system         cilium-x9b8x                                                  1/1
Running   0         160m
kube-system         coredns-558bd4d5db-pdbqt                                      1/1
Running   0         163m
kube-system         coredns-558bd4d5db-scg2n                                      1/1
```



NotSoSecure part of claranet cyber security

© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

```
Running   0          163m
kube-system          etcd-cilium-hubble-cluster-control-plane                     1/1
Running   0          163m
kube-system          hubble-relay-84999fcb48-hsxkw                                1/1
Running   0          161m
kube-system          hubble-ui-9b6d87f-ljbkk                                      3/3
Running   0          160m
kube-system          kube-apiserver-cilium-hubble-cluster-control-plane           1/1
Running   0          163m
kube-system          kube-controller-manager-cilium-hubble-cluster-control-plane  1/1
Running   0          163m
kube-system          kube-proxy-vdd7r                                             1/1
Running   0          162m
kube-system          kube-proxy-vtxz5                                             1/1
Running   0          163m
kube-system          kube-scheduler-cilium-hubble-cluster-control-plane           1/1
Running   0          163m
local-path-storage   local-path-provisioner-547f784dff-zcthx                      1/1
Running   0          163m
```

This cluster has 4 pods running in the default namespace. these pods have following labels

| Pods | Labels |
|---|---|
| deathstar-* | app=deathstar,org=empire |
| tiefighter | app=tiefighter,org=empire |
| xwing | app=xwing,org=alliance |

Above information can be identified using command mentioned below.

**Command:**

```
kubectl get pods --show-labels
```

**Output:**

```
NAME                         READY   STATUS    RESTARTS   AGE     LABELS
deathstar-7fb98564d4-4xwm8   1/1     Running   0          5h52m   app=deathstar,org=empire
,pod-template-hash=7fb98564d4
deathstar-7fb98564d4-hj6rl   1/1     Running   0          5h52m   app=deathstar,org=empire
,pod-template-hash=7fb98564d4
tiefighter                   1/1     Running   0          5h52m   app=tiefighter,org=empir
e
xwing                        1/1     Running   0          5h52m   app=xwing,org=alliance
```

**Monitoring Network Traffic with Hubble CLI and GUI**

To create some network traffic in our cluster for pods ‘tiefighter’ and ‘xwing’, both these pods will try to connect to service ‘deathstar’.

**Command:**

```
kubectl get svc
```

**Output:**

```
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
deathstar    ClusterIP   10.96.184.0    <none>        80/TCP    6h19m
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP   6h22m
```



NotSoSecure part of claranet cyber security

© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Let us generate traffic in our cluster using commands mentioned below.

**Command:**

```
# kubectl exec xwing -- curl -s -XPOST deathstar.default.svc.cluster.local/v1/request-land
ing

Ship landed

# kubectl exec tiefighter -- curl -s -XPOST deathstar.default.svc.cluster.local/v1/request
-landing

Ship landed
```

To monitor the network traffic via CLI we can use below mentioned command:

```
hubble observe -f
```

**Output:**

```
Feb 25 21:28:41.385: default/xwing:45628 -> default/deathstar-7fb98564d4-5f524:80 to-endpo
int FORWARDED (TCP Flags: SYN)
Feb 25 21:28:41.385: default/xwing:45628 <- default/deathstar-7fb98564d4-5f524:80 to-endpo
int FORWARDED (TCP Flags: SYN, ACK)
-----------------------------------snipped-------------------------------------------------
-
Feb 25 21:28:43.601: default/tiefighter:34768 -> default/deathstar-7fb98564d4-4tft5:80 to-
endpoint FORWARDED (TCP Flags: ACK)
Feb 25 21:28:43.601: default/tiefighter:34768 -> default/deathstar-7fb98564d4-4tft5:80 to-
endpoint FORWARDED (TCP Flags: ACK, FIN)
```

We can also see flow of traffic in Hubble GUI using command mentioned below:

```
cilium hubble ui
```

**Output:**

```
i  Opening "http://localhost:12000" in your browser...
Running Firefox as root in a regular user's session is not supported.  ($XAUTHORITY is /ru
n/user/1000/gdm/Xauthority which is owned by osboxes.)
Running Firefox as root in a regular user's session is not supported.  ($XAUTHORITY is /ru
n/user/1000/gdm/Xauthority which is owned by osboxes.)
Running Firefox as root in a regular user's session is not supported.  ($XAUTHORITY is /ru
n/user/1000/gdm/Xauthority which is owned by osboxes.)
/usr/bin/xdg-open: 869: iceweasel: not found
/usr/bin/xdg-open: 869: seamonkey: not found
/usr/bin/xdg-open: 869: mozilla: not found
/usr/bin/xdg-open: 869: epiphany: not found
/usr/bin/xdg-open: 869: konqueror: not found
/usr/bin/xdg-open: 869: chromium: not found
/usr/bin/xdg-open: 869: chromium-browser: not found
/usr/bin/xdg-open: 869: google-chrome: not found
/usr/bin/xdg-open: 869: www-browser: not found
/usr/bin/xdg-open: 869: links2: not found
/usr/bin/xdg-open: 869: elinks: not found
/usr/bin/xdg-open: 869: links: not found
/usr/bin/xdg-open: 869: lynx: not found
/usr/bin/xdg-open: 869: w3m: not found
xdg-open: no method available for opening 'http://localhost:12000'
```

Let’s navigate to URL **http://localhost:12000** in the browser and click on default namespace



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

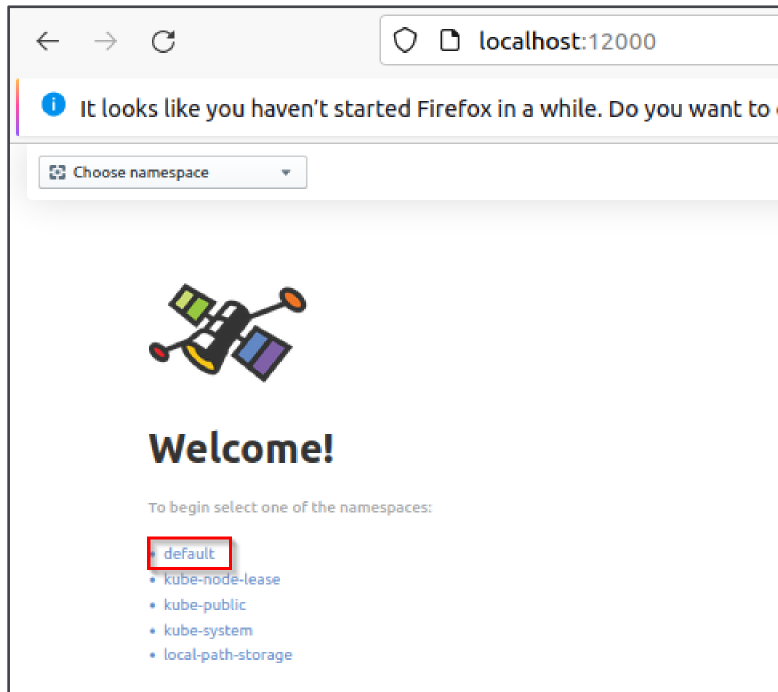
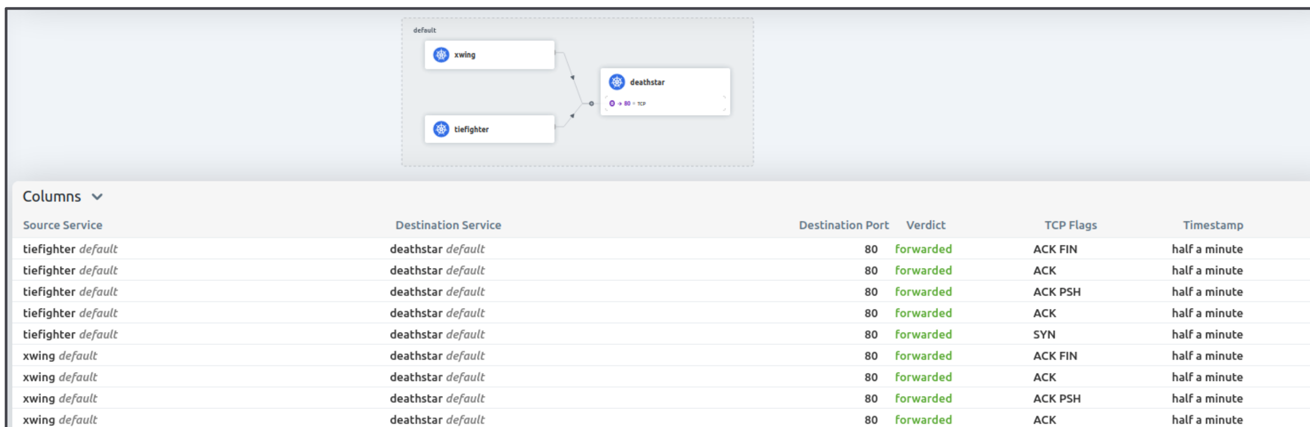



Figure 1:

Now we should be able to visualize the traffic flow as shown below:




| Source Service | Destination Service | Destination Port | Verdict | TCP Flags | Timestamp |
|---|---|---|---|---|---|
| tiefighter *default* | deathstar *default* | 80 | forwarded | ACK FIN | half a minute |
| tiefighter *default* | deathstar *default* | 80 | forwarded | ACK | half a minute |
| tiefighter *default* | deathstar *default* | 80 | forwarded | ACK PSH | half a minute |
| tiefighter *default* | deathstar *default* | 80 | forwarded | ACK | half a minute |
| tiefighter *default* | deathstar *default* | 80 | forwarded | SYN | half a minute |
| xwing *default* | deathstar *default* | 80 | forwarded | ACK FIN | half a minute |
| xwing *default* | deathstar *default* | 80 | forwarded | ACK | half a minute |
| xwing *default* | deathstar *default* | 80 | forwarded | ACK PSH | half a minute |
| xwing *default* | deathstar *default* | 80 | forwarded | ACK | half a minute |

Figure 2:

**Network Policies**

Cilium CNI supports network access control policies which allows administrators to manage network traffic in the cluster. Cilium network policies can be implemented on network or application layer.

CiliumNetworkPolicies match on pod labels using an “endpointSelector” to identify the sources and destinations to which the policy applies.

Lets check a network policy that allowed ingress traffic from any pod with label ‘org=empire’ and destination pod should contain labels ‘org=empire’ and ‘app=deathstar’



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "rule1"
spec:
  description: "L3-L4 policy to restrict deathstar access to empire ships only"
  endpointSelector:
    matchLabels:
      org: empire
      app: deathstar
  ingress:
  - fromEndpoints:
    - matchLabels:
        org: empire
    toPorts:
    - ports:
      - port: "80"
        protocol: TCP
```

We can visualize this policy by uploading it to **NetworkPolicyEditor**

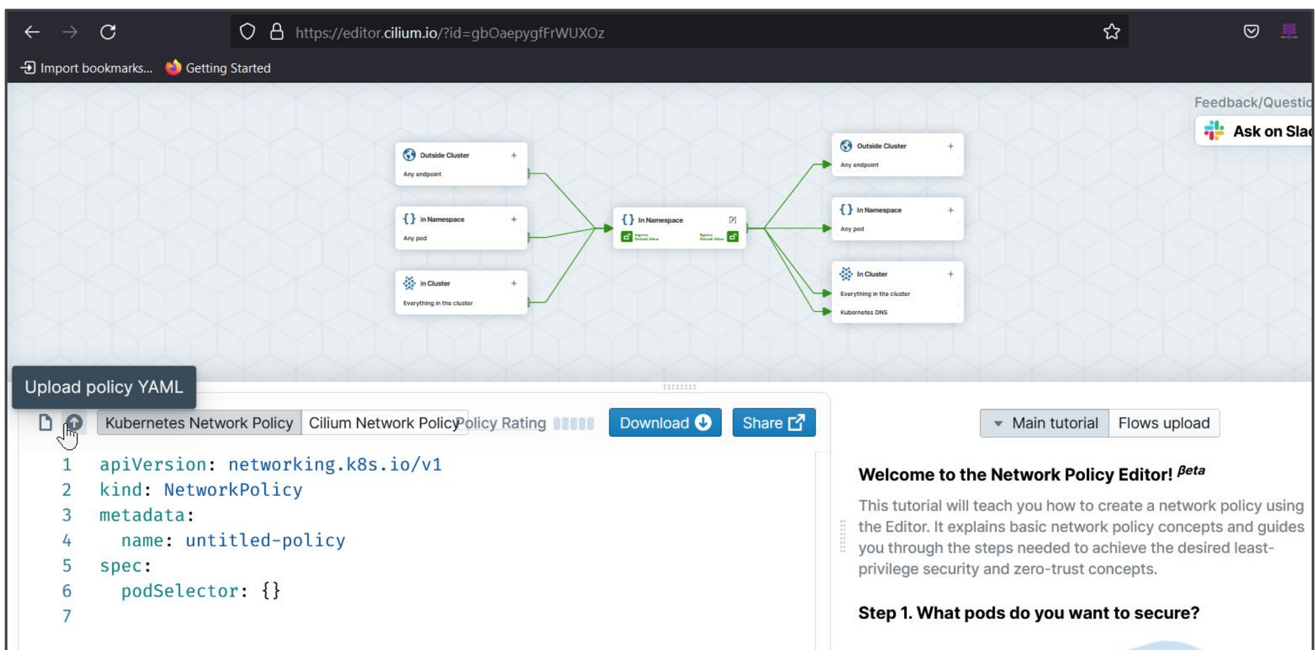
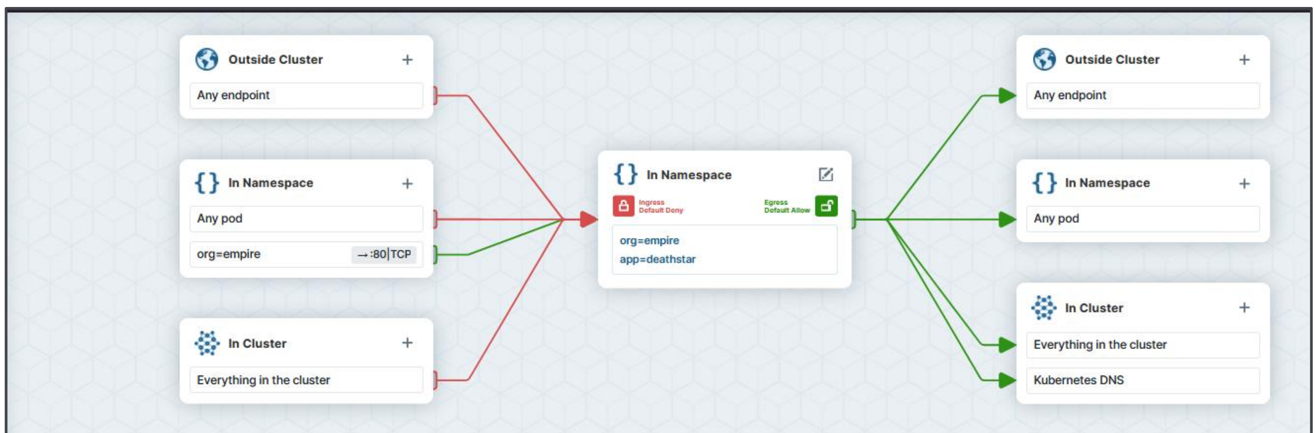



Figure 3:

Once the policy is uploaded, diagram gets updated as shown below:






© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Figure 4:

By closely observing this diagram we can deduce that destination pods should contain labels 'org=empire' and 'app=deathstar' as shown in screenshot below:

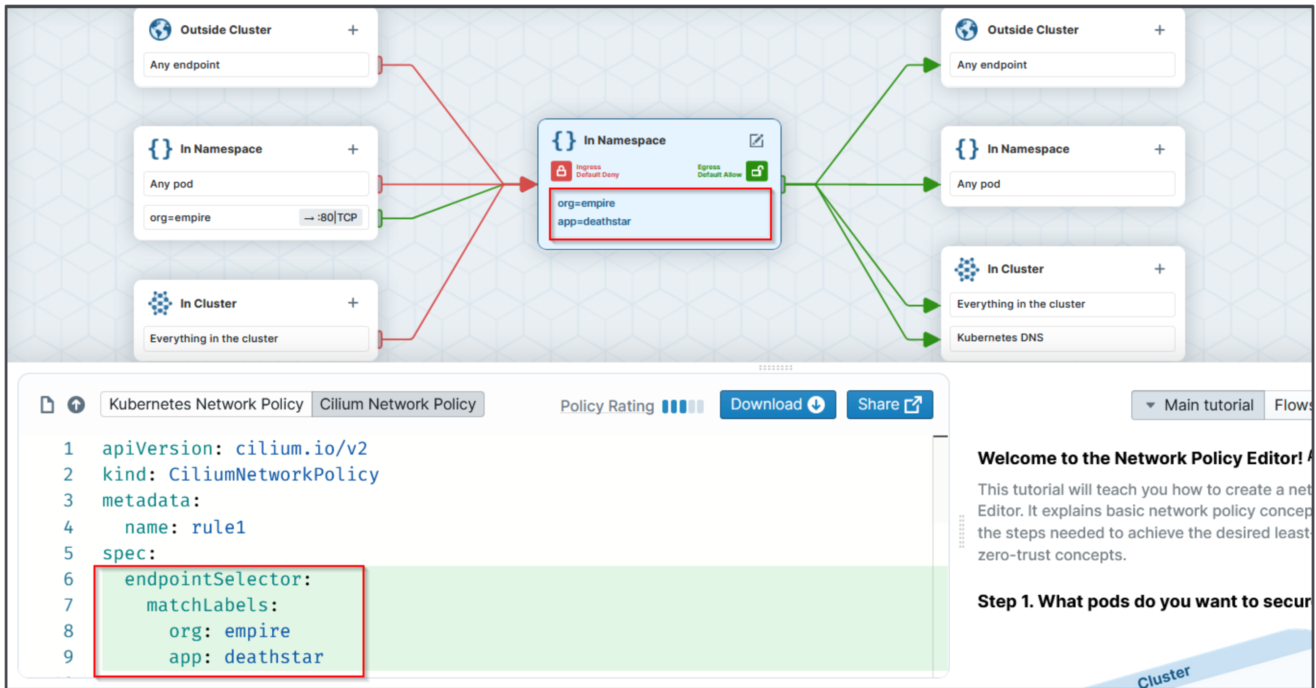



Figure 5:

As per the diagram ingress traffic is only allowed from any pods with label 'org=empire' in the same namespace to port 80 (TCP) of the destination pods as shown below:



NotSoSecure part of claranet cyber security

© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

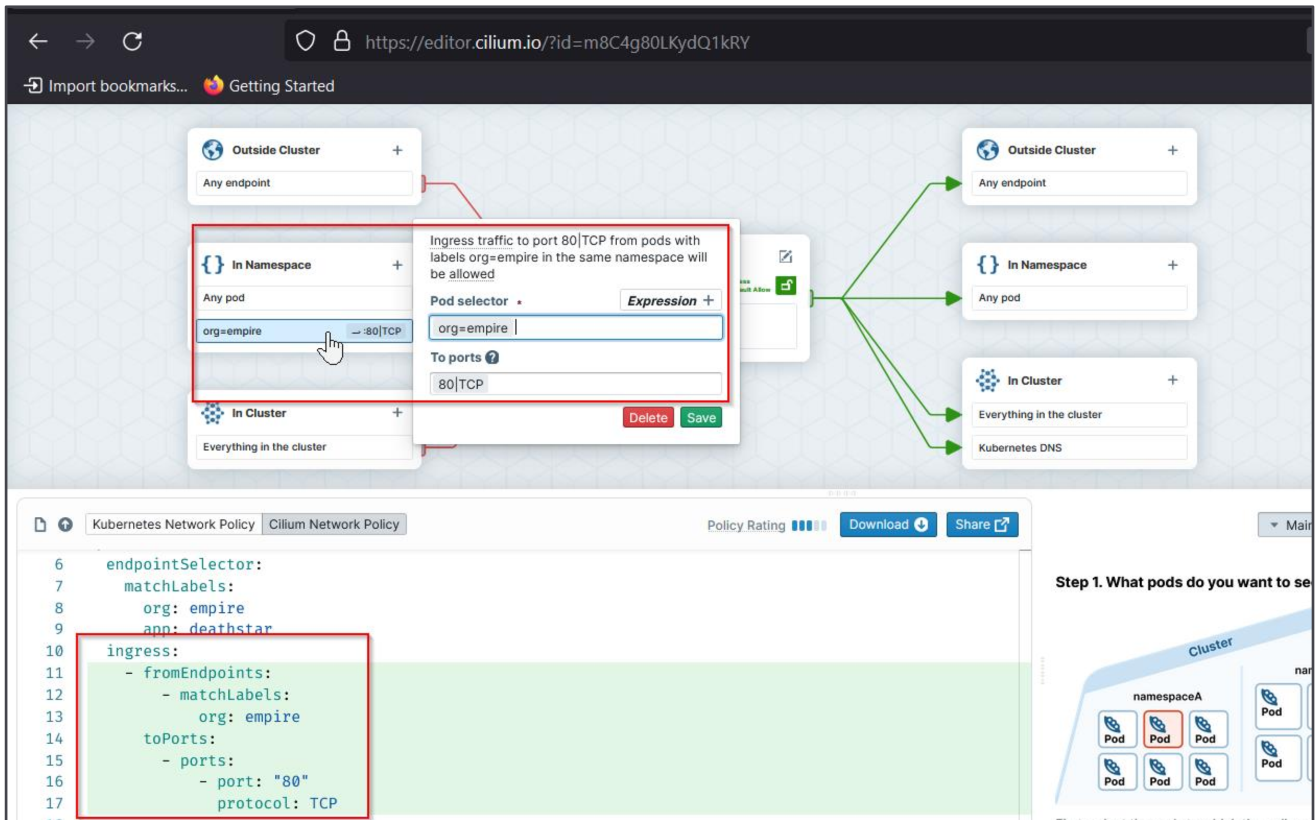


Figure 6:

Validate the yaml file.

**Command:**

```
cat l3-cilium.yaml
```

**Output:**

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "rule1"
spec:
  description: "L3-L4 policy to restrict deathstar access to empire ships only"
  endpointSelector:
    matchLabels:
      org: empire
      app: deathstar
  ingress:
  - fromEndpoints:
    - matchLabels:
        org: empire
    toPorts:
    - ports:
      - port: "80"
        protocol: TCP
```

Lets apply this policy in our cluster using command mentioned below:

**Command:**

```
kubectl apply -f l3-cilium.yaml
```



NotSoSecure part of claranet cyber security

© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

**Output:**

```
ciliumnetworkpolicy.cilium.io/rule1 created
```

Now if we try to access 'deathstar' pods from 'xwing' pod using command mentioned below:

**Command:**

```
kubectl exec xwing -- curl -s -XPOST deathstar.default.svc.cluster.local/v1/request-landin
g
```

We do not get any response, lets investigate this in Hubble GUI and we can observe that TCP from 'xwing' pod intended towards 'deathstar' pod starts dropping.

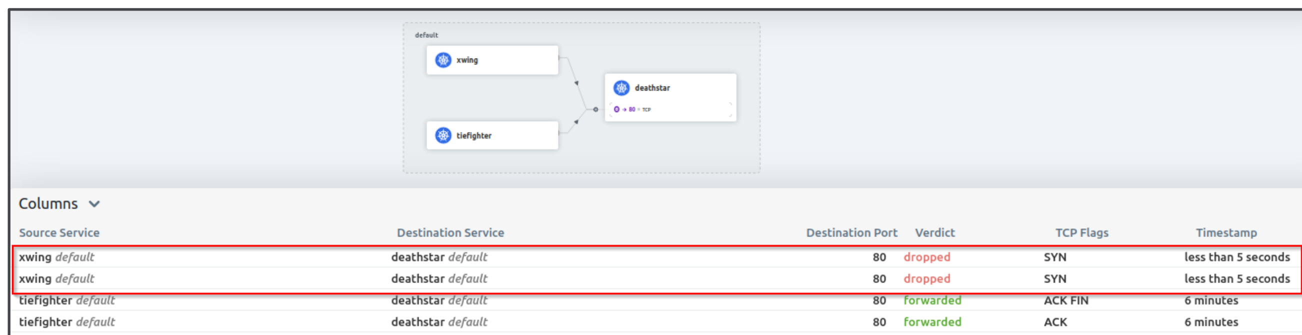



Figure 7:

we can check same in Hubble CLI using command mentioned below:

**Command**:

```
hubble observe --pod xwing -f
```

**Output:**

```
Feb 25 23:08:57.882: default/xwing:34750 <> default/deathstar-7fb98564d4-4tft5:80 Policy d
enied DROPPED (TCP Flags: SYN)
Feb 25 23:08:57.882: default/xwing:34750 <> default/deathstar-7fb98564d4-4tft5:80 Policy d
enied DROPPED (TCP Flags: SYN)
```

So we are able to restrict traffic at a network layer so no TCP traffic is forwarded towards 'deathstar' pod from 'xwing' pod.

Before we check out application layer policy let's try to access 'deathstar' pod from 'tiefighter' pod using PUT HTTP method on specified URL.

**Command:**

```
kubectl exec tiefighter -- curl -s -XPUT deathstar.default.svc.cluster.local/v1/exhaust-po
rt
```

**Output:**

```
Panic: deathstar exploded

goroutine 1 [running]:
main.HandleGarbage(0x2080c3f50, 0x2, 0x4, 0x425c0, 0x5, 0xa)
        /code/src/github.com/empire/deathstar/
        temp/main.go:9 +0x64
main.main()
```



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

```
        /code/src/github.com/empire/deathstar/
        temp/main.go:5 +0x85
```

As we can see program running on 'deathstar' pod explodes if tiefighter access '/v1/exhaust-port' path via PUT HTTP method.

The application layer policy mentioned below only allows HTTP POST method along with specific path '/v1/request-landing' from 'tiefighter' pod to 'deathstar' pod

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "rule1"
spec:
  description: "L7 policy to restrict access to specific HTTP call"
  endpointSelector:
    matchLabels:
      org: empire
      app: deathstar
  ingress:
  - fromEndpoints:
    - matchLabels:
        org: empire
    toPorts:
    - ports:
      - port: "80"
        protocol: TCP
      rules:
        http:
        - method: "POST"
          path: "/v1/request-landing"
```

Let's apply this policy in our cluster.

**Command:**

```
kubectl apply -f l7-cilium.yaml
```

**Output:**

```
ciliumnetworkpolicy.cilium.io/rule1 configured
```

Now let's try again to access 'deathstar' pod from 'tiefighter' pod using PUT HTTP method on specified path '/v1/request-landing'.

**Command**:

```
kubectl exec tiefighter -- curl -s -XPUT deathstar.default.svc.cluster.local/v1/exhaust-po
rt
```

**Output:**

```
Access denied
```

We get an error access denied, lets also check hubble CLI to inspect traffic using command mentioned below:

**Command**:

```
hubble observe --pod tiefighter -f
```

**Output:**



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

```
Feb 25 23:42:38.606: default/tiefighter:47975 <- kube-system/coredns-558bd4d5db-hn9rz:53 t
o-endpoint FORWARDED (UDP)
Feb 25 23:42:38.606: default/tiefighter:47975 <- kube-system/coredns-558bd4d5db-hn9rz:53 t
o-endpoint FORWARDED (UDP)
---------------------------------------------snipped--------------------------------
Feb 25 23:42:38.606: default/tiefighter:34068 <- default/deathstar-7fb98564d4-4tft5:80 to-
endpoint FORWARDED (TCP Flags: ACK, FIN)
Feb 25 23:42:38.606: default/tiefighter:34068 -> default/deathstar-7fb98564d4-4tft5:80 htt
p-request DROPPED (HTTP/1.1 PUT http://deathstar.default.svc.cluster.local/v1/exhaust-port
)
```

We can clearly observe that HTTP PUT request from ‘tiefighter’ pod to ‘deathstar’ is dropped.

Let’s destroy the cluster.

```
./destroy-cilium-hubble-cluster.sh
```

To verify if cluster is deleted, use following command

```
# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

you should see no docker containers running.



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

# K8s: Defence 3

Run a local kubernetes cluster with falco and Kube-Prometheus enabled

- Create a pod with privileged container and access it via terminal shell
- Identify these actions in Grafana Dashboard using Prometheus monitoring

## Solution

**Introduction**

In this exercise we will see how Falco could be used to monitor kubernetes cluster and detect malicious activities via kernel event in the cluster.

Components that we will be using in this Exercise are:

**kind** is a tool for running local Kubernetes clusters using Docker container 'nodes'.

**Helm** is a package manager for kubernetes which is used to install monitoring components in this cluster

**kube-prometheus-stack** is a chart that could be deployed using Helm. This chart contains all the componentes that are required to monitor kubernetes cluster via Promethous. some of major components are mentioned below:

- **Promethous** is a systems and service monitoring system. It collects metrics from configured targets
- **Grafana Dashboard** allows you to query, visualize, alert on and understand your metrics.
- **Promethous operator** provides Kubernetes native deployment and management of Prometheus and related monitoring components.

**Falco** is kernel events monitoring tool which is used to observe event generated by the kernel and generate alerts for potentially malicious processes spawned in the Kubernetes cluster.

Navigate to the folder where we have installed our scripts.

**Command:**

```
cd /home/pentester/k8sScript/falco
```

Run the script to setup a cluster.

**Command:**

```
./falco-prometheus-cluster-setup.sh
```

Once our local cluster is ready, we should see following pods in running state in our cluster.

**Command:**

```
kubectl get pod -A
```



NotSoSecure part of claranet cyber security

© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

**Output:**

```
NAMESPACE              NAME                                                     READ
Y   STATUS     RESTARTS   AGE
falco                  falco-exporter-4rrvr                                     1/1
Running   0            118s
falco                  falco-exporter-xf6cw                                     1/1
Running   0            118s
falco                  falco-fg745                                              1/1
Running   0            4m14s
falco                  falco-p4cnv                                              1/1
Running   0            4m14s
kube-system            coredns-558bd4d5db-89w8r                                 1/1
Running   0            6m57s
kube-system            coredns-558bd4d5db-kzp8v                                 1/1
Running   0            6m57s
kube-system            etcd-falco-prometheus-cluster-control-plane              1/1
Running   0            7m2s
kube-system            kindnet-g7145                                            1/1
Running   0            6m42s
kube-system            kindnet-sx5nt                                            1/1
Running   0            6m58s
kube-system            kube-apiserver-falco-prometheus-cluster-control-plane    1/1
Running   0            7m2s
kube-system            kube-controller-manager-falco-prometheus-cluster-control-plane   1/1
Running   0            7m2s
kube-system            kube-proxy-6zz6n                                         1/1
Running   0            6m58s
kube-system            kube-proxy-b7pwz                                         1/1
Running   0            6m42s
kube-system            kube-scheduler-falco-prometheus-cluster-control-plane    1/1
Running   0            7m2s
local-path-storage     local-path-provisioner-547f784dff-gtk26                  1/1
Running   0            6m57s
monitoring             alertmanager-prometheus-kube-prometheus-alertmanager-0   2/2
Running   0            6m7s
monitoring             prometheus-grafana-65b46787f7-qvfk6                      3/3
Running   0            6m23s
monitoring             prometheus-kube-prometheus-operator-56b9f9fb67-jsnq2     1/1
Running   0            6m23s
monitoring             prometheus-kube-state-metrics-57c988498f-8mhw8           1/1
Running   0            6m23s
monitoring             prometheus-prometheus-kube-prometheus-prometheus-0       2/2
Running   0            6m7s
monitoring             prometheus-prometheus-node-exporter-2sfwg                1/1
Running   0            6m23s
monitoring             prometheus-prometheus-node-exporter-6rhl7                1/1
Running   0            6m23s
```

### Monitoring Falco alerts in Grafana dashboard

Now, let’s create new privileged pod in our cluster using command mentioned below:

**Command:**

```
kubectl run --privileged priv-pod --image ubuntu sleep 99d
```

This command will spin up an Ubuntu pod with name ‘priv-pod’ and will stay alive till 99days.

Now let’s check the services running in our cluster using below mentioned command.

**Command:**

```
kubectl get svc -A
```



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

**Output:**

```
NAMESPACE      NAME                                                 TYPE        CLUSTER-IP
EXTERNAL-IP    PORT(S)                        AGE
default        kubernetes                                           ClusterIP   10.96.0.1
<none>         443/TCP                        8m34s
falco          falco-exporter                                       ClusterIP   None
<none>         9376/TCP                       3m17s
kube-system    kube-dns                                             ClusterIP   10.96.0.10
<none>         53/UDP,53/TCP,9153/TCP         8m33s
kube-system    prometheus-kube-prometheus-coredns                   ClusterIP   None
<none>         9153/TCP                       7m42s
kube-system    prometheus-kube-prometheus-kube-controller-manager   ClusterIP   None
<none>         10252/TCP                      7m42s
kube-system    prometheus-kube-prometheus-kube-etcd                 ClusterIP   None
<none>         2379/TCP                       7m42s
kube-system    prometheus-kube-prometheus-kube-proxy                ClusterIP   None
<none>         10249/TCP                      7m42s
kube-system    prometheus-kube-prometheus-kube-scheduler            ClusterIP   None
<none>         10251/TCP                      7m42s
kube-system    prometheus-kube-prometheus-kubelet                   ClusterIP   None
<none>         10250/TCP,10255/TCP,4194/TCP   7m26s
monitoring     alertmanager-operated                                ClusterIP   None
<none>         9093/TCP,9094/TCP,9094/UDP     7m26s
monitoring     prometheus-grafana                                   ClusterIP   10.96.65.24
3     <none>        80/TCP                         7m42s
monitoring     prometheus-kube-prometheus-alertmanager              ClusterIP   10.96.114.4
1     <none>        9093/TCP                       7m42s
monitoring     prometheus-kube-prometheus-operator                  ClusterIP   10.96.151.1
4     <none>        443/TCP                        7m42s
monitoring     prometheus-kube-prometheus-prometheus                ClusterIP   10.96.154.1
26    <none>        9090/TCP                       7m42s
monitoring     prometheus-kube-state-metrics                        ClusterIP   10.96.6.162
<none>         8080/TCP                       7m42s
monitoring     prometheus-operated                                  ClusterIP   None
<none>         9090/TCP                       7m26s
monitoring     prometheus-prometheus-node-exporter                  ClusterIP   10.96.148.1
57    <none>        9100/TCP                       7m42s
```

let’s port-forward ‘prometheus-grafana’ service to localhost using command mentioned below:

**Command:**

```
kubectl port-forward -n monitoring svc/prometheus-grafana 80
```

**Output:**

```
Forwarding from 127.0.0.1:80 -> 3000
Forwarding from [::1]:80 -> 3000
```

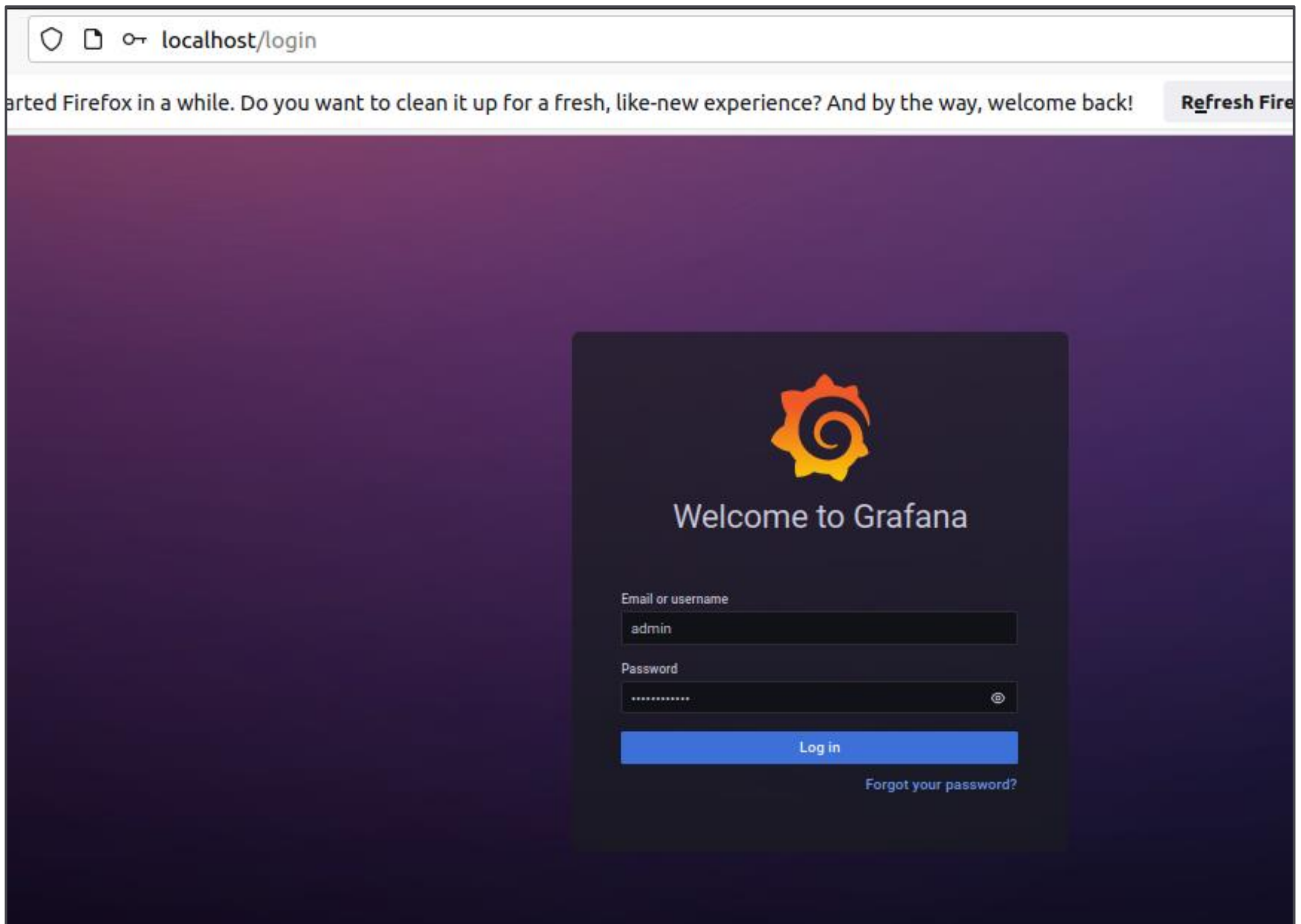
Let’s navigate to **http://127.0.0.1:80** to access Grafana-dashboard. Credentials for this dashboard are

- Username: admin
- Password: prom-operator

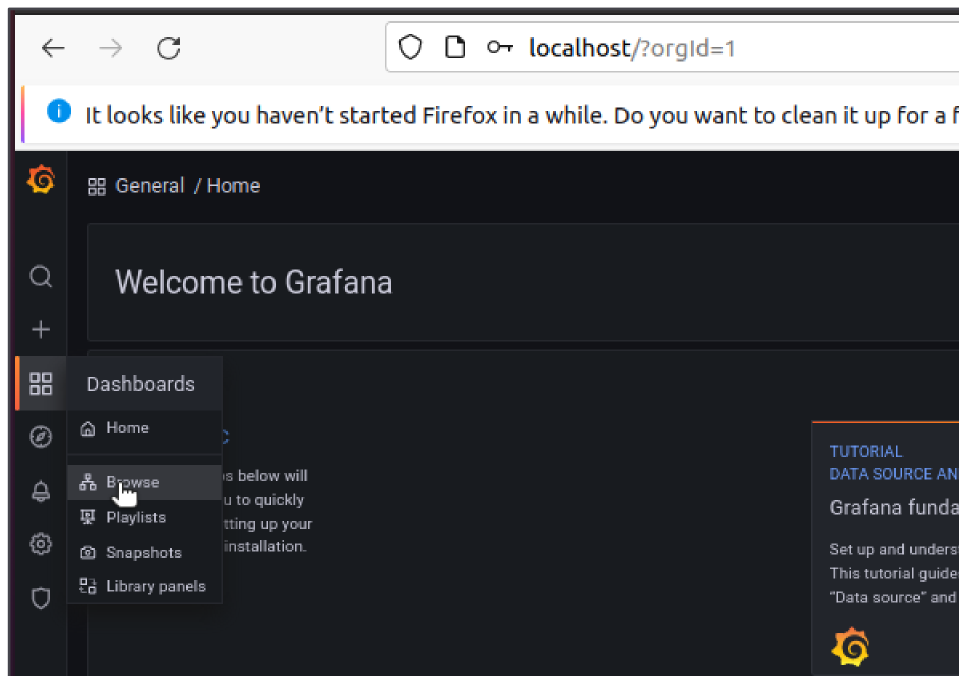


© Claranet Cyber Security. All rights reserved

https://t.me/learningnets




Once you have logged in, access the Manage option in Dashboard




Now access Falco-dashboard to see the privileged container alert generated by Falco



NotSoSecure part of claranet cyber security

© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Now let’s spawn a shell inside the privileged pod.

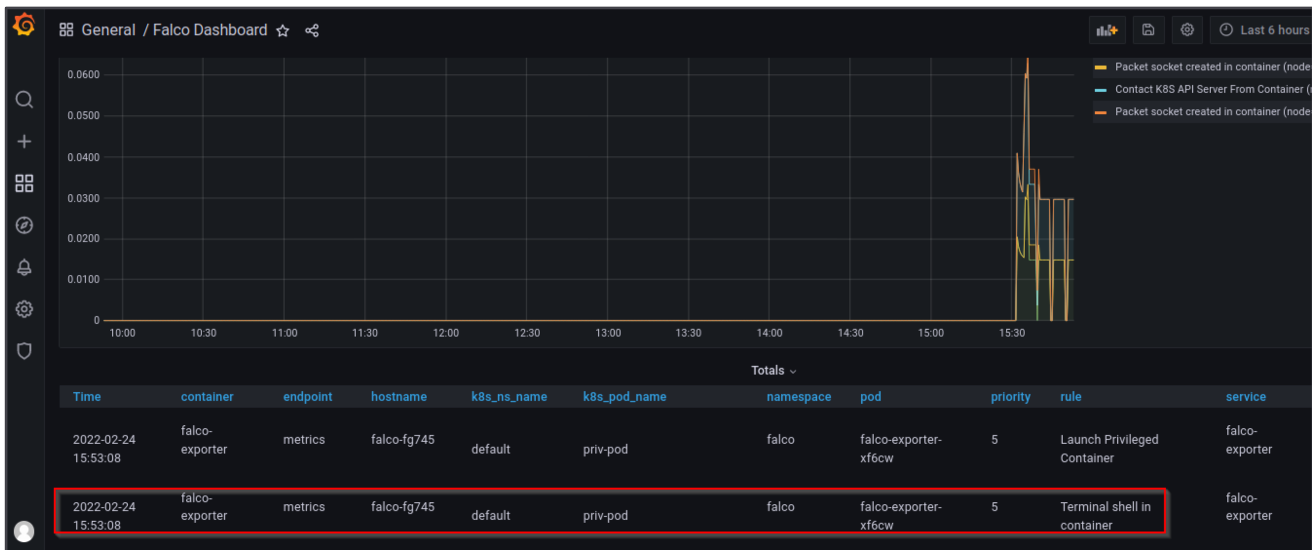
### Command:

```
kubectl exec -it priv-pod -- bash
```

### Output:

```
root@priv-pod:/# id
uid=0(root) gid=0(root) groups=0(root)
```

We can observe in our grafana dashboard an alert is raised called ‘Terminal shell in ’container’.




A shell spawned in production environment should raise alerts, cause as per the best practice no one should be able to execute commands in a running container.

### Destroy this local cluster using below mentioned command

```
# ./destroy-falco-prometheus-cluster.sh
Deleting cluster "falco-prometheus-cluster" ...
```

To verify if cluster is deleted, use following command


© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

```
# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

you should see no docker containers running.



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

# Identify:

We have provided auditor level access for all 3 vendors

Using the access provided list various assets:

- VM's
- Buckets
- Functions

## Solution

In this exercise we will login to the cli of all 3 vendors i.e. AWS, Azure and GCP using the provided credentials and enumerate the resources accessible to these accounts.

### AWS:

```
export AWS_ACCESS_KEY_ID=AKIA2QOCAU2PEIHDJSLK
export AWS_SECRET_ACCESS_KEY=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
export AWS_DEFAULT_REGION=us-east-1
```

Let’s login to the AWS cli using below mentioned command:

### Command

```
aws sts get-caller-identity
```

### Output :

```
{
    "UserId": "AIDA2QOCAU2PCKQIPZEI6",
    "Account": "722498266782",
    "Arn": "arn:aws:iam::722498266782:user/secauditor"
}
```

Now that we have valid account let’s try to list of EC2 VMs present in this account using below mentioned command:

### Command

```
for i in $(aws ec2 describe-regions --query "Regions[*].[RegionName]" --output text); do a
ws ec2 describe-instances --region $i; done
```

### Output :

```
ap-southeast-2
{
    "Reservations": [
        {
            "Groups": [],
            "Instances": [
                {
                    "AmiLaunchIndex": 0,
                    "ImageId": "ami-0fb7513bcdc525c3b",
                    "InstanceId": "i-06256842554e5ce72",
                    "InstanceType": "t2.micro",
                    "KeyName": "aws_sydney",
                    "LaunchTime": "2019-05-26T09:13:45.000Z",
                    "Monitoring": {
```



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

```
                    "State": "disabled"
                },
                "Placement": {
                    "AvailabilityZone": "ap-southeast-2a",
                    "GroupName": "",
                    "Tenancy": "default"
                },
                "PrivateDnsName": "ip-172-31-3-238.ap-southeast-2.compute.internal",
                "PrivateIpAddress": "172.31.3.238",
                "ProductCodes": [],
                "PublicDnsName": "ec2-13-211-222-159.ap-southeast-2.compute.amazonaws.
com",
                "PublicIpAddress": "13.211.222.159",
                "State": {
                    "Code": 16,
                    "Name": "running"
                },
                "StateTransitionReason": "",
                "SubnetId": "subnet-c1dad2a6",
                "VpcId": "vpc-7df4d41a",
                "Architecture": "x86_64",
                "BlockDeviceMappings": [
                    {
                        "DeviceName": "/dev/xvda",
                        "Ebs": {
                            "AttachTime": "2019-05-26T09:13:45.000Z",
                            "DeleteOnTermination": true,
                            "Status": "attached",
                            "VolumeId": "vol-042ee68561415988f"
                        }
                    }
                ],
                "ClientToken": "",
                "EbsOptimized": false,
                "EnaSupport": true,
                "Hypervisor": "xen",
                "NetworkInterfaces": [
                    {
                        "Association": {
                            "IpOwnerId": "amazon",
                            "PublicDnsName": "ec2-13-211-222-159.ap-southeast-2.comput
e.amazonaws.com",
                            "PublicIp": "13.211.222.159"
                        },
                        "Attachment": {
                            "AttachTime": "2019-05-26T09:13:45.000Z",
                            "AttachmentId": "eni-attach-0ec98b7c63cba0e70",
                            "DeleteOnTermination": true,
                            "DeviceIndex": 0,
                            "Status": "attached"
                        },
                        "Description": "",
                        "Groups": [
                            {
                                "GroupName": "launch-wizard-1",
                                "GroupId": "sg-06ea33cd5d4dc4c9a"
                            }
                        ],
                        "Ipv6Addresses": [],
                        "MacAddress": "02:1f:4c:bc:3f:f4",
                        "NetworkInterfaceId": "eni-0f1d68d51eb1f387c",
                        "OwnerId": "722498266782",
                        "PrivateDnsName": "ip-172-31-3-238.ap-southeast-2.compute.inte
rnal",
                        "PrivateIpAddress": "172.31.3.238",
                        "PrivateIpAddresses": [
                            {
```




© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Technet24

```
                                    "Association": {
                                        "IpOwnerId": "amazon",
                                        "PublicDnsName": "ec2-13-211-222-159.ap-southeast-
2.compute.amazonaws.com",

                                        "PublicIp": "13.211.222.159"
                                    },
                                    "Primary": true,
                                    "PrivateDnsName": "ip-172-31-3-238.ap-southeast-2.comp
ute.internal",

                                    "PrivateIpAddress": "172.31.3.238"
                                }
                            ],
                            "SourceDestCheck": true,
                            "Status": "in-use",
                            "SubnetId": "subnet-c1dad2a6",
                            "VpcId": "vpc-7df4d41a",
                            "InterfaceType": "interface"
                        }
                    ],
                    "RootDeviceName": "/dev/xvda",
                    "RootDeviceType": "ebs",
                    "SecurityGroups": [
                        {
                            "GroupName": "launch-wizard-1",
                            "GroupId": "sg-06ea33cd5d4dc4c9a"
                        }
                    ],
                    "SourceDestCheck": true,
                    "VirtualizationType": "hvm",
                    "CpuOptions": {
                        "CoreCount": 1,
                        "ThreadsPerCore": 1
                    },
                    "CapacityReservationSpecification": {
                        "CapacityReservationPreference": "open"
                    },
                    "HibernationOptions": {
                        "Configured": false
                    },
                    "MetadataOptions": {
                        "State": "applied",
                        "HttpTokens": "optional",
                        "HttpPutResponseHopLimit": 1,
                        "HttpEndpoint": "enabled"
                    }
                }
            ],
            "OwnerId": "722498266782",
            "ReservationId": "r-06305d8bf6a10a2f1"
        }
    ]
}
```


Ok, so we have one ec2 instances running for this account, let's try to list buckets present in this account:

### Command

```
aws s3api list-buckets
```

### Output :


```
{
    "Buckets": [
        {
            "Name": "cf-templates-1vvffrylpsjsd-us-east-2",
```




© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

```
            "CreationDate": "2020-07-27T02:24:32.000Z"
        },
        {
            "Name": "victimauth",
            "CreationDate": "2019-05-21T09:30:56.000Z"
        },
        {
            "Name": "victimcloudelk",
            "CreationDate": "2020-07-27T02:29:35.000Z"
        },
        {
            "Name": "victimcloudelkcloudtraillogs",
            "CreationDate": "2020-07-27T02:40:40.000Z"
        },
        {
            "Name": "victimcloudelkconfigbucket",
            "CreationDate": "2020-07-27T02:40:40.000Z"
        },
        {
            "Name": "victimprivate",
            "CreationDate": "2019-05-21T09:30:56.000Z"
        },
        {
            "Name": "victimpublic",
            "CreationDate": "2019-05-21T09:30:56.000Z"
        },
        {
            "Name": "voffice.victim.cloud",
            "CreationDate": "2020-07-08T22:45:36.000Z"
        }
    ],
    "Owner": {
        "DisplayName": "cloudhd",
        "ID": "376dacdb7eb6f55abf298fa77d6bcf54e292819ba4e4054d099450060ff14435"
    }
}
```


Ok, so complete the first part of this challenge lets list all the lambda functions present in this account using below mentioned command:

### Command

```
for i in $(aws ec2 describe-regions --query "Regions[*].[RegionName]" --output text); do e
cho $i; aws lambda list-functions --region $i; done
```

### Output :


```
{
    "Functions": [
        {
            "FunctionName": "awsproxy",
            "FunctionArn": "arn:aws:lambda:us-east-2:722498266782:function:awsproxy",
            "Runtime": "nodejs10.x",
            "Role": "arn:aws:iam::722498266782:role/prodaccess",
            "Handler": "index.handler",
            "CodeSize": 1315348,
            "Description": "",
            "Timeout": 3,
            "MemorySize": 128,
            "LastModified": "2020-06-22T02:04:43.222+0000",
            "CodeSha256": "OVEqq460CaA6cOpHD7uaWuJZk7BXZ7i46bKZ5HmEqKY=",
            "Version": "$LATEST",
            "Environment": {
                "Variables": {
                    "SuperSecretEnvironmentKey": "aa3e3c2e6d9c088564f8ba25487dbbc421aa35ee
835a3689c02c4090bb0ca44c"
```




NotSoSecure part of claranet cyber security

© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Technet24

```
                }
            },
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "RevisionId": "38914857-cca3-40c8-81ca-d136d4782a7f"
        }
    ]
}
```


### Azure:

We have to login to Azure cli for enumerating resources present in this account using command mentioned below:

### Command

```
az login -u azureauditaccount@victim.cloud -p '39Solutions'
```

### Output :


```
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "dfb882fd-afcd-4f7f-b67f-45bff25daf24",
    "id": "b85c3133-5cb6-4cdf-942d-28a88262ebf1",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Free Trial",
    "state": "Enabled",
    "tenantId": "dfb882fd-afcd-4f7f-b67f-45bff25daf24",
    "user": {
      "name": "azureauditaccount@victim.cloud",
      "type": "user"
    }
  }
]
```


Now let's try to enumerate VMs present in this account using command mentioned below:

### Command

```
az vm list
```

### Output :


```
[
  {
    "additionalCapabilities": null,
    "availabilitySet": null,
    "billingProfile": null,
    "diagnosticsProfile": {
      "bootDiagnostics": {
        "enabled": true,
        "storageUri": "https://hastcdiag.blob.core.windows.net/"
      }
    },
    "evictionPolicy": null,
    "extensionsTimeBudget": null,
    "hardwareProfile": {
      "vmSize": "Standard_B1s"
    },
    "host": null,
    "hostGroup": null,
    "id": "/subscriptions/b85c3133-5cb6-4cdf-942d-28a88262ebf1/resourceGroups/HASTC/provid
```




NotSoSecure part of claranet cyber security

© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

```
ers/Microsoft.Compute/virtualMachines/dev-box",
    "identity": null,
    "instanceView": null,
    "licenseType": null,
    "location": "eastus",
    "name": "dev-box",
    "networkProfile": {
      "networkInterfaces": [
        {
          "id": "/subscriptions/b85c3133-5cb6-4cdf-942d-28a88262ebf1/resourceGroups/HaStC/
providers/Microsoft.Network/networkInterfaces/dev-box621",
          "primary": null,
          "resourceGroup": "HaStC"
        }
      ]
    },
    "osProfile": {
      "adminPassword": null,
      "adminUsername": "AzureUser",
      "allowExtensionOperations": true,
      "computerName": "dev-box",
      "customData": null,
      "linuxConfiguration": {
        "disablePasswordAuthentication": false,
        "provisionVmAgent": true,
        "ssh": null
      },
      "requireGuestProvisionSignal": true,
      "secrets": [],
      "windowsConfiguration": null
    },
    "plan": null,
    "priority": null,
    "provisioningState": "Succeeded",
    "proximityPlacementGroup": null,
    "resourceGroup": "HASTC",
    "resources": null,
    "securityProfile": null,
    "storageProfile": {
      "dataDisks": [],
      "imageReference": {
        "exactVersion": "18.04.202006101",
        "id": null,
        "offer": "UbuntuServer",
        "publisher": "Canonical",
        "sku": "18.04-LTS",
        "version": "latest"
      },
      "osDisk": {
        "caching": "ReadWrite",
        "createOption": "FromImage",
        "diffDiskSettings": null,
        "diskSizeGb": 30,
        "encryptionSettings": null,
        "image": null,
        "managedDisk": {
          "diskEncryptionSet": null,
          "id": "/subscriptions/b85c3133-5cb6-4cdf-942d-28a88262ebf1/resourceGroups/HASTC/
providers/Microsoft.Compute/disks/dev-box_disk1_700991dbae2749caabb9b8d924465af2",
          "resourceGroup": "HASTC",
          "storageAccountType": "Standard_LRS"
        },
        "name": "dev-box_disk1_700991dbae2749caabb9b8d924465af2",
        "osType": "Linux",
        "vhd": null,
        "writeAcceleratorEnabled": null
      }
```




© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Technet24

```
    },
    "tags": null,
    "type": "Microsoft.Compute/virtualMachines",
    "virtualMachineScaleSet": null,
    "vmId": "2cdf5841-11b5-43eb-9e09-70dc6a326a7c",
    "zones": null
  }
]
```


Let’s check storage resources present in this account:

## Command

```
az storage account list
```

## Output :


```
[
  {
    "accessTier": null,
    "allowBlobPublicAccess": null,
    "azureFilesIdentityBasedAuthentication": null,
    "blobRestoreStatus": null,
    "creationTime": "2020-07-03T02:25:27.814157+00:00",
    "customDomain": null,
    "enableHttpsTrafficOnly": true,
    "encryption": {
      "keySource": "Microsoft.Storage",
      "keyVaultProperties": null,
      "requireInfrastructureEncryption": null,
      "services": {
        "blob": {
          "enabled": true,
          "keyType": "Account",
          "lastEnabledTime": "2020-07-03T02:25:27.892251+00:00"
        },
        "file": {
          "enabled": true,
          "keyType": "Account",
          "lastEnabledTime": "2020-07-03T02:25:27.892251+00:00"
        },
        "queue": null,
        "table": null
      }
    },
    "failoverInProgress": null,
    "geoReplicationStats": null,
    "id": "/subscriptions/b85c3133-5cb6-4cdf-942d-28a88262ebf1/resourceGroups/HaStC/provid
ers/Microsoft.Storage/storageAccounts/hastcdiag",
    "identity": null,
    "isHnsEnabled": null,
    "kind": "Storage",
    "largeFileSharesState": null,
    "lastGeoFailoverTime": null,
    "location": "eastus",
    "minimumTlsVersion": null,
    "name": "hastcdiag",
    "networkRuleSet": {
      "bypass": "AzureServices",
      "defaultAction": "Allow",
      "ipRules": [],
      "virtualNetworkRules": []
    },
    "primaryEndpoints": {
      "blob": "https://hastcdiag.blob.core.windows.net/",
      "dfs": null,
```




© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

```
      "file": "https://hastcdiag.file.core.windows.net/",
      "internetEndpoints": null,
      "microsoftEndpoints": null,
      "queue": "https://hastcdiag.queue.core.windows.net/",
      "table": "https://hastcdiag.table.core.windows.net/",
      "web": null
    },
    "primaryLocation": "eastus",
    "privateEndpointConnections": [],
    "provisioningState": "Succeeded",
    "resourceGroup": "HaStC",
    "routingPreference": null,
    "secondaryEndpoints": null,
    "secondaryLocation": null,
    "sku": {
      "name": "Standard_LRS",
      "tier": "Standard"
    },
    "statusOfPrimary": "available",
    "statusOfSecondary": null,
    "tags": {},
    "type": "Microsoft.Storage/storageAccounts"
  },
  {
    "accessTier": "Cool",
    "allowBlobPublicAccess": true,
    "azureFilesIdentityBasedAuthentication": null,
    "blobRestoreStatus": null,
    "creationTime": "2020-07-25T22:36:23.692276+00:00",
    "customDomain": null,
    "enableHttpsTrafficOnly": true,
    "encryption": {
      "keySource": "Microsoft.Storage",
      "keyVaultProperties": null,
      "requireInfrastructureEncryption": null,
      "services": {
        "blob": {
          "enabled": true,
          "keyType": "Account",
          "lastEnabledTime": "2020-07-25T22:36:23.754753+00:00"
        },
        "file": {
          "enabled": true,
          "keyType": "Account",
          "lastEnabledTime": "2020-07-25T22:36:23.754753+00:00"
        },
        "queue": null,
        "table": null
      }
    },
    "failoverInProgress": null,
    "geoReplicationStats": null,
    "id": "/subscriptions/b85c3133-5cb6-4cdf-942d-28a88262ebf1/resourceGroups/HaStC/provid
ers/Microsoft.Storage/storageAccounts/victimassets",
    "identity": null,
    "isHnsEnabled": null,
    "kind": "BlobStorage",
    "largeFileSharesState": null,
    "lastGeoFailoverTime": null,
    "location": "eastus",
    "minimumTlsVersion": "TLS1_0",
    "name": "victimassets",
    "networkRuleSet": {
      "bypass": "AzureServices",
      "defaultAction": "Allow",
      "ipRules": [],
      "virtualNetworkRules": []
```




© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Technet24

```
    },
    "primaryEndpoints": {
      "blob": "https://victimassets.blob.core.windows.net/",
      "dfs": "https://victimassets.dfs.core.windows.net/",
      "file": null,
      "internetEndpoints": null,
      "microsoftEndpoints": null,
      "queue": null,
      "table": "https://victimassets.table.core.windows.net/",
      "web": null
    },
    "primaryLocation": "eastus",
    "privateEndpointConnections": [],
    "provisioningState": "Succeeded",
    "resourceGroup": "HaStC",
    "routingPreference": null,
    "secondaryEndpoints": null,
    "secondaryLocation": null,
    "sku": {
      "name": "Standard_LRS",
      "tier": "Standard"
    },
    "statusOfPrimary": "available",
    "statusOfSecondary": null,
    "tags": {},
    "type": "Microsoft.Storage/storageAccounts"
  }
]
```


To complete the second part of this challenge we have list functionapps present in this account using below mentioned command:

**Command**

```
az functionapp list
```

**Output :**

```
[]
```

So, we can conclude here that no functionapps are present in this account.

**GCP :**

We can login to the GCP account using gcp_creds.json file which contains GCP credentials:

**Command**

```
cat gcp_creds.json
```

**Output :**


```
{
  "type": "service_account",
  "project_id": "cloud-hd-gcp",
  "private_key_id": "0fc8c7d94b11f95a4a886f39b9fa1a07895edeb9",
  "private_key": "-----BEGIN PRIVATE KEY-----\nMIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgE
AAoIBAQDbV7MWYRazTSjl\n+zY3kebx4SUchxbPn3Bnsm9XrxR7JBp/hmMhqhUN1gBRLie++5CYu2zPSp+ncKrG\ny
7LN4Y8R7uGIiqtIPK34zw4X5dn1wEQ6WcbEtxi/cBTjG2f+cPmE1bMTjcQa67F7\n3tnh1RS/UrD1F6Kubu1gWFV0Q
9Jh0FEwJVTp2DID/ZNRVuhAOwqgW5el2vxtTrNp\nZQWDx0q4BHdDwZu3wCP++hcd6v6vsg08IUqyEBMDbJMV3KaRZ
/2CoMjO2b1eBKyF\nytzgzDbpulIhvGP643LcUWFM1Vv4yNpthGOZnbgo8S/h/IRuqlrHxPitCAJfqn9c\ndubxpRQ
3AgMBAAECggEAAQzj9iej3gWbZwfeGW5bmUKXP2eO3pKoYqQSRWy4tThA\nijt2A7tr8Ljayoi6jFRpu67WFWNe7Vn
W74Rv3Zgxa7K6+2uSk2d6+8szXujtCMYR\nbZLFfBFWZIBLEVADHk74hZV9DzW+v9RAxtOT3obQhnQFFeoK5OPyGLk
uO5QhkJyr\nqbfps/Yr7bS9pIBJsWBSF5pA1nrQvMaaXz1lew9+F4U3u7U29KUbAG4uiRmcsjEY\nZtIc6Exw2fkRu
```




NotSoSecure part of claranet cyber security

© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

```
gzbABQac7DgRLNbXSYDldJ4xcQ1I6FVnSRVP10qg13uCeVPq8i8\nC4viUOBcSs9gF0gz0ReBBAbRDvQ0brFJhBV+I
mOa4QKBgQDyXuE9HXOw3CmWjSCD\nG6dvymN/YBzSR+09ZZAiFmqlGLTBA/qb+Tt/L/S5wdxOOVx2iCjGtiOr1jr7V
GiF\nPlImaVSOU6PINEp6iL2JpTFMV5G2sHxp0V18K6AyqUWwoWWiRAc2S8RXR0XoHB6U\nT2PZQRNsmZyUN1tZNXB
H7iNY1wKBgQDnrU/8KpMXE27TYBJYXYeqq0UXz3l3kKRd\niR5lKWg28iBUSZvEuBQXQIc6U180OZjx/wFNZy2slUM
9YqCLsZ/O1mnDdauj/pgE\nK4Bmetby3EZRXfzoDkRmAk+CQalPW/N4GlHrj+oZXe36bE5tOyf29v3iL3uzk84Y\nY
+0FEzfToQKBgDs5L/SNE83tnkPpbD5dLYbFf5aKV8CSTsgn3xZVP+3joZdgb4ZZ\nWk5z22rgkgKTRaK5Fq2nqAO/L
15me7vDRLOaHU1B5s1u6ZwTeaz+rpMm+rYXXe9F\nd4B18Ikwj5Tfe0QxN0nvLaTeB5j19grrc0yM+EAbMQWOmUWtJ
yiMahq1AoGANvaL\nD0wacVMO697S+1vVH8xaK/fw3UjXdcleCKn2K3Lt8JE1/0mcmpeYfWHiKEWx31ec\nO3zK/TK
8LQgFBp8xRqTfmpuECPgRoIuhfq6N6DgSBqPBqFwaJglOS5zASi+KZ2f3\nH2eZJkN+ffbL9GYgaRrXyJ1zP3tFXcv
3I0kX6YECgYAy6PyIbn1k1Nocqb6vWj3x\nVtjshuc31sOIGNYCpdUOX0PDhEw2G9WOyHToue742mwU1Cje745PlPS
5lBjuM4Xr\n7kh9DePETmB1+EZM3JRcD190fQjzSZ03tmJtxy7laS8iyy0gxgNw4K1VRo+xGSTK\n0Tw4nm8ycnBXA
j2vZW+5AQ==\n-----END PRIVATE KEY-----\n",
  "client_email": "auditor@cloud-hd-gcp.iam.gserviceaccount.com",
  "client_id": "109742499766490466140",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/auditor%40clo
ud-hd-gcp.iam.gserviceaccount.com"
}
```


Let's login to GCP account using below mentioned command:

**Command**

```
gcloud alpha auth activate-service-account --key-file=gcp_creds.json
```

**Output :**

```
Activated service account credentials for: [auditor@cloud-hd-gcp.iam.gserviceaccount.com]
```

Now we will have to configure project id for this account which is present in gcp_creds.json file using below mentioned command:

**Command**

```
gcloud config set project cloud-hd-gcp
```

**Output :**

```
Updated property [core/project].
```

Now that we are able to login to auditor's account lets list Compute Engine VMs present in this account using command mentioned below:

**Command**

```
gcloud compute instances list
```

**Output :**

```
NAME     ZONE           MACHINE_TYPE  PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP     STATUS
webapp1  us-central1-a  f1-micro                   10.128.0.3   35.208.245.198  RUNNING
```

Let's check list of buckets present in this account using below mentioned command:

**Command**

```
gsutil ls
```

**Output :**



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Technet24

```
gs://victimauth/
gs://victimprivate/
gs://victimpublic/
```

And to complete this challenge we have to list all the functions present in this account using command mentioned below:

### Command

```
gcloud functions list
```

### Output :

```
NAME          STATUS   TRIGGER       REGION
victimproxy   ACTIVE   HTTP Trigger  us-central1
```

In conclusion in this exercise, we have learnt how to identify assets present in our cloud account. Using this information, we can move forward to implementing defined strategies for these assets.



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

# Audit:

## Audit 1:

Perform cloud security audit over the following environments:

- AWS
- GCP

### Solution

**Auditing GCP environment with Scout Suite**

We can run the following command to scan the GCP environment where gcp_creds.json contains file based authentication keys for the environment. The credentials file uses a service account having only read permission set for audit purposes.

**Command**

```
scout gcp --service-account gcp_creds.json
```

**Note:** Multiple APIs should be enabled to successfully complete the scan. If you get an error while running the scan, it will show the specific API URLs that need to be enabled. Simply browse to the URL in your GCP console and enable them. It is to be noted that some APIs take time to populate the change once enabled for the first time.

```
pentester@NotSoSecure:~$ scout gcp --service-account gcp_cred.json
2022-03-25 18:49:06 NotSoSecure scout[16893] INFO Launching Scout
2022-03-25 18:49:06 NotSoSecure scout[16893] INFO Authenticating to cloud provider
2022-03-25 18:49:07 NotSoSecure scout[16893] INFO Gathering data from APIs
2022-03-25 18:49:07 NotSoSecure scout[16893] INFO Fetching resources for the Cloud SQL service
2022-03-25 18:49:09 NotSoSecure scout[16893] INFO Fetching resources for the Cloud Storage service
2022-03-25 18:49:09 NotSoSecure scout[16893] INFO Fetching resources for the Compute Engine service
2022-03-25 18:49:10 NotSoSecure scout[16893] INFO Fetching resources for the IAM service
2022-03-25 18:49:11 NotSoSecure scout[16893] INFO Fetching resources for the KMS service
2022-03-25 18:49:12 NotSoSecure scout[16893] INFO Fetching resources for the Stackdriver Logging service
2022-03-25 18:49:12 NotSoSecure scout[16893] INFO Fetching resources for the Stackdriver Monitoring service
2022-03-25 18:49:13 NotSoSecure scout[16893] INFO Fetching resources for the Kubernetes Engine service
```

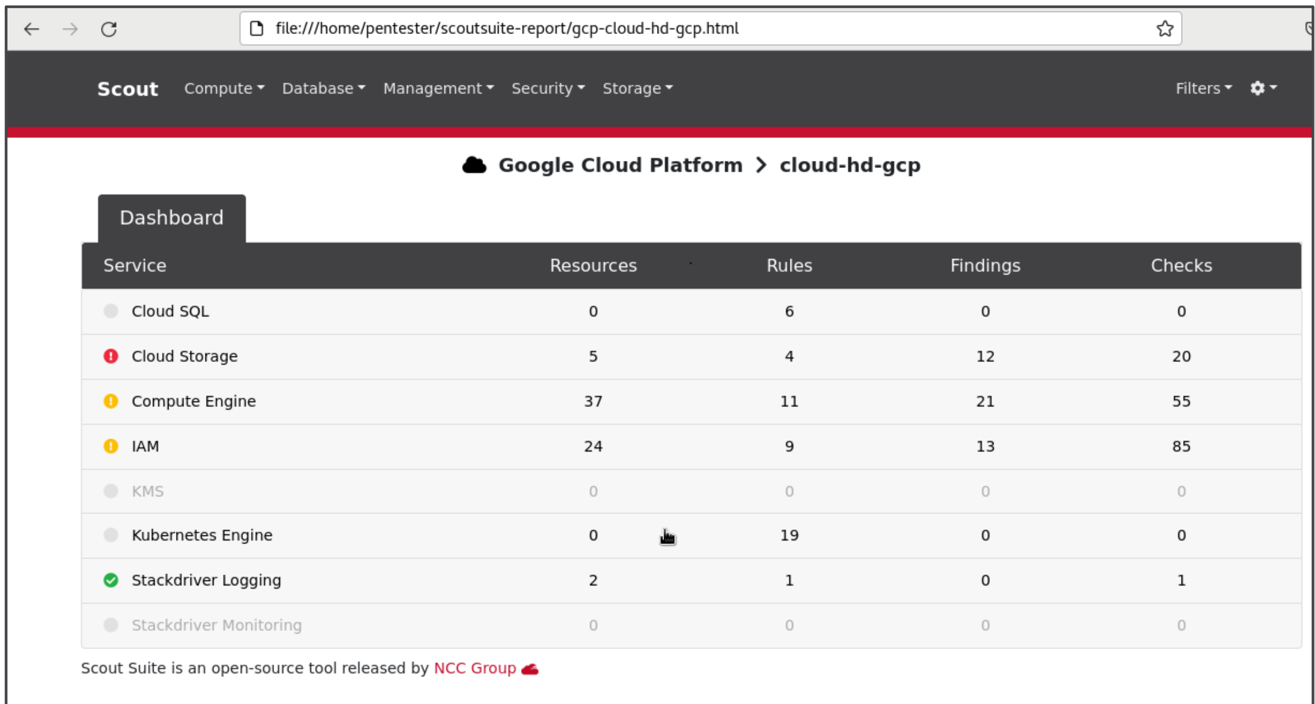
After completion it will create a report and automatically try to open it. We can also browse scoutsuite-report directory to view the report manually as shown in the below figure.



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Technet24




### Auditing Azure with Scout Suite

To run it in an Azure environment you should be logged in Azure SDK.

### Command

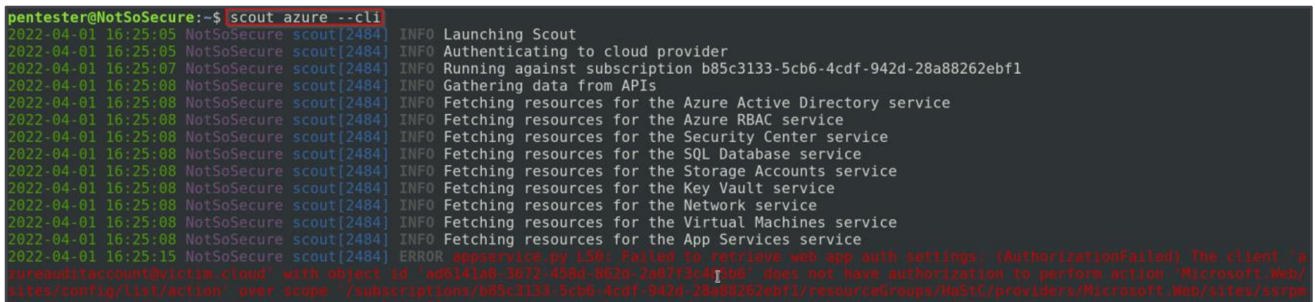
```
az login -u azureauditaccount@victim.cloud -p '39Solutions'
```

Once you are logged in, use the following command to run the tool. And same as GCP the account must have relevant read permission on the environment.

### Command

```
scout azure --cli
```

Similarly, after completion it will create a report and automatically try to open it as shown in the below figure.




Below is a sample report for Azure



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Scout Applications ▾ Compute ▾ Database ▾ Networking ▾ Security ▾ Storage ▾ Filters ▾

Microsoft Azure > dfb882fd-afcd-4f7f-b67f-45bff25daf24

Dashboard

| Service | Resources | Rules | Findings | Checks |
|---|---|---|---|---|
| Azure Active Directory | 17 | 1 | 0 | 4 |
| App Services | 1 | 10 | 3 | 10 |
| Key Vault | 1 | 0 | 0 | 0 |
| Network | 5 | 6 | 1 | 30 |
| Azure RBAC | 338 | 0 | 0 | 0 |
| Security Center | 14 | 7 | 14 | 15 |
| SQL Database | 1 | 14 | 10 | 14 |
| Storage Accounts | 2 | 5 | 4 | 11 |
| Virtual Machines | 2 | 2 | 1 | 2 |

### Auditing AWS with Scout Suite

Run the following command to execute Scout Suite on AWS.

**Note:** We may get access related errors while performing read operations on certain modules, this is because we have not given full read permissions to our account

### Command

```
scout aws -p userX
```

Below screenshot shows Scout Suite Console output.

```
pentester@NotSoSecure:~$ scout aws -p user10
2022-03-25 18:58:21 NotSoSecure scout[17276] INFO Launching Scout
2022-03-25 18:58:21 NotSoSecure scout[17276] INFO Authenticating to cloud provider
2022-03-25 18:58:26 NotSoSecure scout[17276] INFO Gathering data from APIs
2022-03-25 18:58:26 NotSoSecure scout[17276] INFO Fetching resources for the ACM service
2022-03-25 18:58:27 NotSoSecure scout[17276] INFO Fetching resources for the Lambda service
2022-03-25 18:58:28 NotSoSecure scout[17276] INFO Fetching resources for the CloudFormation service
2022-03-25 18:58:29 NotSoSecure scout[17276] INFO Fetching resources for the CloudTrail service
2022-03-25 18:58:30 NotSoSecure scout[17276] INFO Fetching resources for the CloudWatch service
2022-03-25 18:58:31 NotSoSecure scout[17276] INFO Fetching resources for the Config service
2022-03-25 18:58:32 NotSoSecure scout[17276] INFO Fetching resources for the Direct Connect service
2022-03-25 18:58:34 NotSoSecure scout[17276] INFO Fetching resources for the DynamoDB service
2022-03-25 18:58:35 NotSoSecure scout[17276] INFO Fetching resources for the EC2 service
2022-03-25 18:58:36 NotSoSecure scout[17276] INFO Fetching resources for the EFS service
2022-03-25 18:58:37 NotSoSecure scout[17276] INFO Fetching resources for the ElastiCache service
2022-03-25 18:58:38 NotSoSecure scout[17276] INFO Fetching resources for the ELB service
2022-03-25 18:58:39 NotSoSecure scout[17276] INFO Fetching resources for the ELBv2 service
2022-03-25 18:58:41 NotSoSecure scout[17276] INFO Fetching resources for the EMR service
2022-03-25 18:58:42 NotSoSecure scout[17276] INFO Fetching resources for the IAM service
2022-03-25 18:58:42 NotSoSecure scout[17276] INFO Fetching resources for the KMS service
2022-03-25 18:58:43 NotSoSecure scout[17276] INFO Fetching resources for the RDS service
2022-03-25 18:58:45 NotSoSecure scout[17276] INFO Fetching resources for the RedShift service
2022-03-25 18:58:46 NotSoSecure scout[17276] INFO Fetching resources for the Route53 service
2022-03-25 18:58:47 NotSoSecure scout[17276] INFO Fetching resources for the S3 service
2022-03-25 18:58:50 NotSoSecure scout[17276] INFO Fetching resources for the SES service
```

Similarly, after completion it will create a report and automatically try to open it as shown in the

© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Technet24

below figure.

Scout Analytics Compute Containers Database Management Messaging Network Security Storage Filters

Amazon Web Services > 143439767741

Dashboard

| Service | Resources | Rules | Findings | Checks |
|---|---|---|---|---|
| ACM | 0 | 2 | 0 | 0 |
| Lambda | 0 | 0 | 0 | 0 |
| CloudFormation | 0 | 1 | 0 | 0 |
| CloudTrail | 0 | 8 | 0 | 0 |
| CloudWatch | 0 | 1 | 0 | 0 |
| Config | 0 | 1 | 0 | 0 |
| Directconnect | 0 | 0 | 0 | 0 |
| DynamoDB | 0 | 0 | 0 | 0 |
| EC2 | 0 | 28 | 0 | 0 |
| EFS | 0 | 0 | 0 | 0 |
| ElastiCache | 0 | 0 | 0 | 0 |
| ELB | 0 | 3 | 0 | 0 |
| ELBV2 | 0 | 5 | 0 | 0 |
| EMR | 0 | 0 | 0 | 0 |
| IAM | 0 | 36 | 4 | 4 |

Below screenshot also depicts findings for one of the sections.

file:///home/pentester/scoutsuite-report/aws-user10.html#services.iam.findings

Scout Analytics Compute Containers Database Management Messaging Network Security Storage Filters

IAM Dashboard

Filter findings | Show All | Good | Warning | Danger

- Minimum Password Length Too Short
- Password Expiration Disabled
- Password Policy Allows the Reuse of Passwords
- Passwords Expire after 90 Days

The other tool we are going to use for our demonstration is Prowler, this tool is specifically for AWS environments.

As we are already authenticated to AWS Cloud environment, we can just initiate the scan using the following command.



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

### Command

```
cd ~/tools/prowler/
```

Let's generate a HTML report using following command.

```
./prowler -p userX -M html
```

### Output:

Following snippet shows the output of CLI.

```
Color code for results:
 -  INFO (Information)
 -  PASS (Recommended value)
 -  WARNING (Ignored by whitelist)
 -  FAIL (Fix required)

 This report is being generated using credentials below:

 AWS-CLI Profile: [userX] AWS API Region: [us-east-1] AWS Filter Region: [all]
 AWS Account: [143439767741] UserId: [AIDASCZNNFC6Y6HHMJL7X]
 Caller Identity ARN: [arn:aws:iam::143439767741:user/userX]

1.0 Identity and Access Management - CIS only - [group1] *********** -  []
 Generating AWS IAM Credential Report... -  []
1.1 [check11] Avoid the use of the root account - iam [High]
       PASS! us-east-1: Root user in the account wasn't accessed in the last 1 days
1.2 [check12] Ensure multi-factor authentication (MFA) is enabled for all IAM users that h
ave a console password - iam [High]
       FAIL! us-east-1: User user10 has Password enabled but MFA disabled
       FAIL! us-east-1: User user11 has Password enabled but MFA disabled
```

HTML Report:



NotSoSecure part of claranet cyber security

© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Technet24

# Audit 2:

- Based on your experience with full audit, perform limited s3 only audit on new accounts provided to you.
- Apply Fix to the identified mis config (s3) and perform the audit again to ensure its fixed.

## Solution

In this exercise we will perform a s3 only audit on the newly provided AWS account and use the audit report to identify the misconfiguration implemented on the s3 bucket.

Once the misconfiguration is identified, implement the fix on the s3 bucket and reperform the AWS s3 only audit to verify the fix.

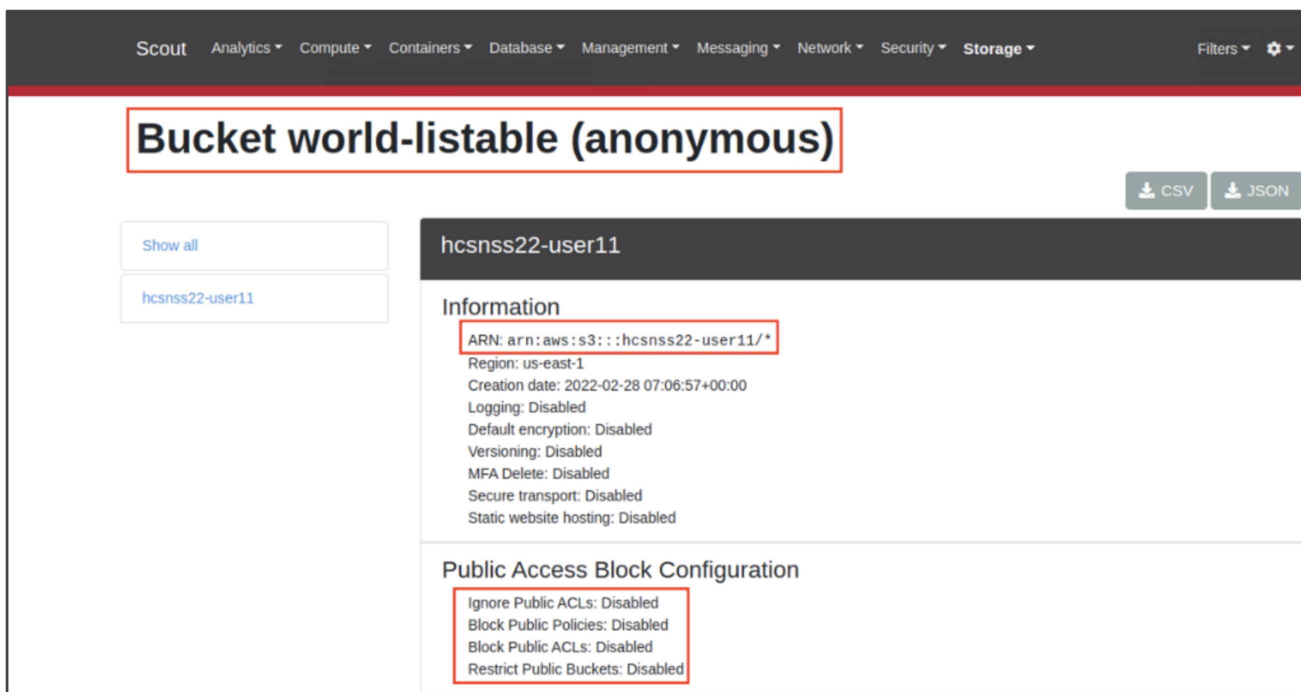
Run the following command to audit the S3 bucket.

**Command:**

```
scout aws --services s3 -p userX
```

```
pentester@NotSoSecure:~$ scout  aws --services s3 -p user11
2022-04-01 14:14:14 NotSoSecure scout[3918] INFO Launching Scout
2022-04-01 14:14:14 NotSoSecure scout[3918] INFO Authenticating to cloud provider
2022-04-01 14:14:19 NotSoSecure scout[3918] INFO Gathering data from APIs
2022-04-01 14:14:19 NotSoSecure scout[3918] INFO Fetching resources for the S3 service
2022-04-01 14:14:25 NotSoSecure scout[3918] ERROR s3.py L62: Failed to get bucket location for cih-attacker-serverless-app: An error occurred (AccessDenied) when calling the GetBucketLocation operation: Access Denied
2022-04-01 14:14:25 NotSoSecure scout[3918] ERROR s3.py L62: Failed to get bucket location for hcsnss22-user10: An error occurred (AccessDenied) when calling the GetBucketLocation operation: Access Denied
```

Once Scout Suite scan is completed, we can review the audit report, in this audit report we can observe that s3 bucket in our account is misconfigured to allow public access.



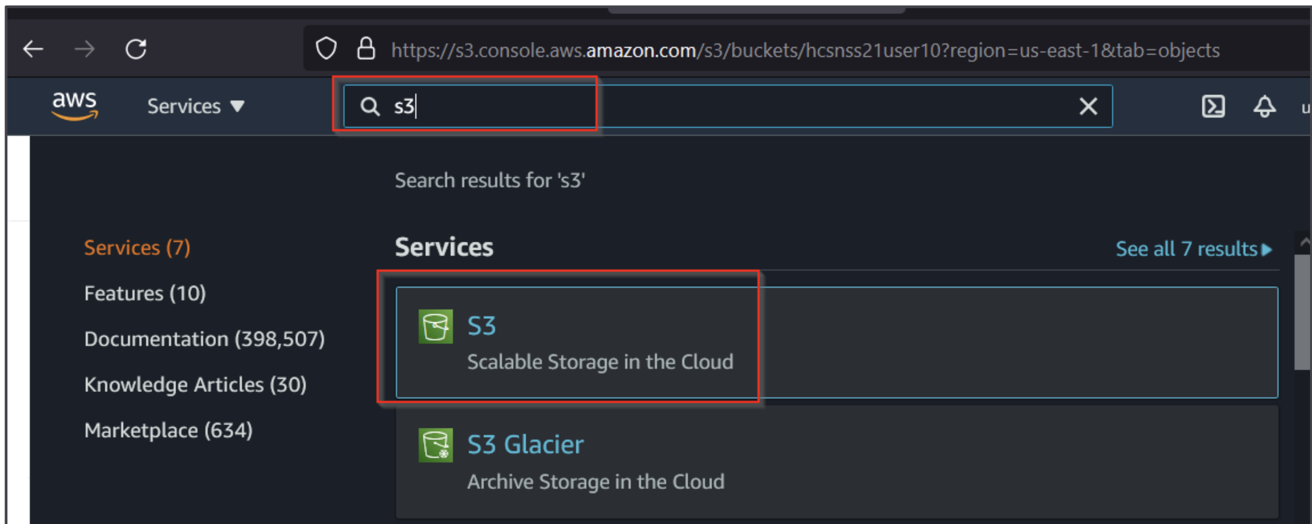

Let's find this misconfigured bucket by navigating to s3 service as shown below:



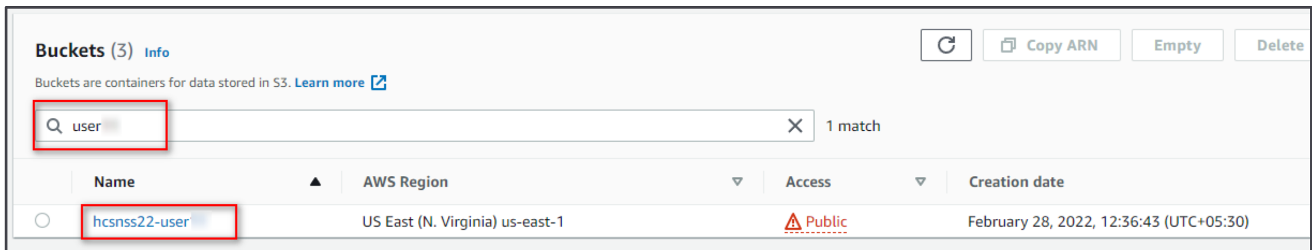
NotSoSecure part of claranet cyber security

© Claranet Cyber Security. All rights reserved

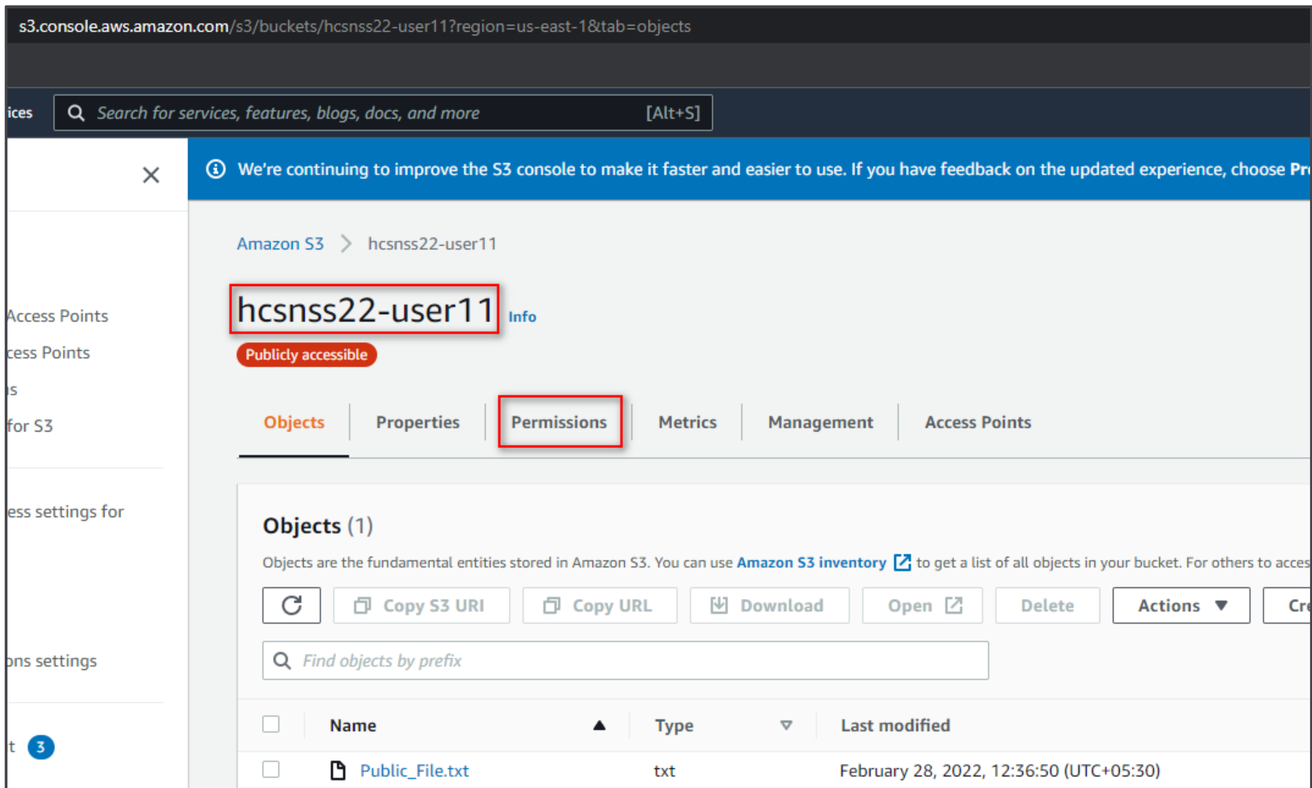
https://t.me/learningnets




Click on s3 to access the s3 service module. In this module search for bucket name assigned to your user account as shown below:




Click on the identified bucket name and access the permissions tab for this bucket.






© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Technet24

In the Permissions tab we can observe that 'Block public access (bucket settings)' is turned off as shown below:

hcsnss22-user11 Info

Publicly accessible

Objects | Properties | Permissions | Metrics | Management | Access Points

**Permissions overview**

Access

Public

**Block public access (bucket settings)**

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to all your S3 buckets and objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to your buckets or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more

Edit

Block *all* public access

Off

▼ Individual Block Public Access settings for this bucket

- Block public access to buckets and objects granted through *new* access control lists (ACLs)
  S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- Block public access to buckets and objects granted through *any* access control lists (ACLs)
  S3 will ignore all ACLs that grant public access to buckets and objects.
- Block public access to buckets and objects granted through *new* public bucket or access point policies
  S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
- Block public and cross-account access to buckets and objects through *any* public bucket or access point policies
  S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

To fix the public bucket access issue identified during audit phase, click on edit in 'Block public access (bucket settings)' and check on block all access checkbox then save the changes as shown below



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Amazon S3 > hcsnss22-user11 > Edit Block public access (bucket settings)

Edit Block public access (bucket settings) Info

Block public access (bucket settings)

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to all your S3 buckets and objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to your buckets or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more

Block *all* public access
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

Block public access to buckets and objects granted through *new* access control lists (ACLs)
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

Block public access to buckets and objects granted through *any* access control lists (ACLs)
S3 will ignore all ACLs that grant public access to buckets and objects.

Block public access to buckets and objects granted through *new* public bucket or access point policies
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

Block public and cross-account access to buckets and objects through *any* public bucket or access point policies
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

Cancel Save changes

Confirm the changes to 'Block public access (bucket settings)' as shown below:

Edit Block public access (bucket settings)

This will result in public access being blocked for this bucket and all objects in the bucket.

To confirm the settings, enter *confirm* in the field.

confirm

Cancel Confirm

Once our changes are applied, we should see 'Block public access (bucket settings)' turned On as shown below:



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Technet24

Successfully edited Block Public Access settings for this bucket.

hcsnss22-user11 Info

Objects | Properties | Permissions | Metrics | Management | Access Points

Permissions overview

Access

Bucket and objects not public

Block public access (bucket settings)

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to all your S3 buckets and objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to your buckets or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more

Edit

Block *all* public access

On

Individual Block Public Access settings for this bucket

Block public access to buckets and objects granted through *new* access control lists (ACLs)

S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

Block public access to buckets and objects granted through *any* access control lists (ACLs)

S3 will ignore all ACLs that grant public access to buckets and objects.

Block public access to buckets and objects granted through *new* public bucket or access point policies

S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

Block public and cross-account access to buckets and objects through *any* public bucket or access point policies

S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

Now, let's rerun the Scout Suite and check if this misconfiguration is fixed or not.

```
scout aws --services s3 -p userX
```

Scout Suite report for s3 post implementation of fix for public bucket access as shown below:

Scout Analytics Compute Containers Database Management Messaging Network Security Storage Filters

S3 Dashboard

Filter findings | Show All | Good | Warning | Danger

- Bucket Access Logging Disabled
- Bucket Allowing Clear Text (HTTP) Communication
- Bucket without Default Encryption Enabled
- Bucket without MFA Delete
- Bucket without Versioning
- All Actions Authorized to All Principals
- Bucket world-listable

Description

An S3 bucket has been found that was accessible for AuthenticatedUsers. If sensitive information is stored in this bucket, it could be at risk of being accessed by potential attackers.

- Bucket ACLs checked: 0
- Bucket ACLs flagged: 0

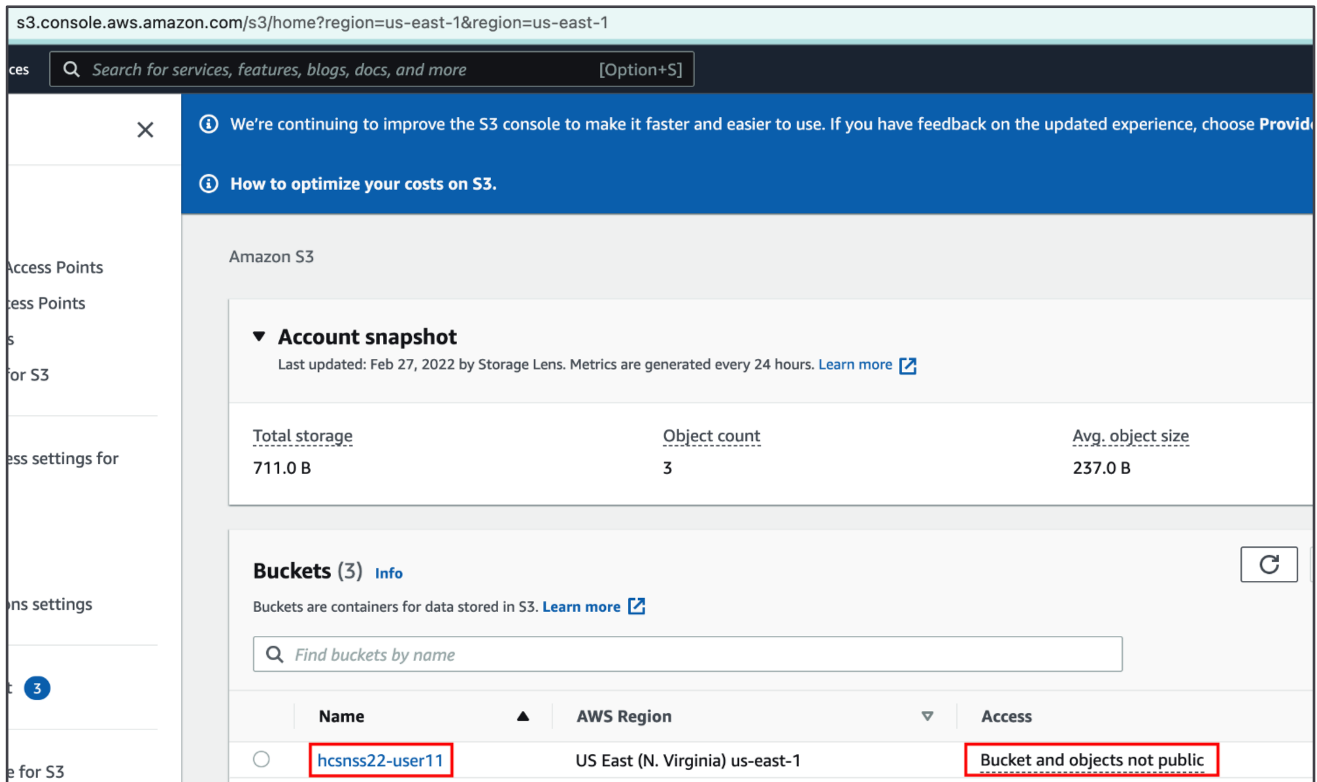
Additionally, we can see that the bucket and object are not public, as shown below.



NotSoSecure part of claranet cyber security

© Claranet Cyber Security. All rights reserved

https://t.me/learningnets




We can observe that s3 'Bucket world-listable (anonymous)' issue is fixed.

So, in this exercise we were able to identify the misconfigured s3 bucket which allowed public access by performing s3 only audit using Scout Suite on the provided AWS account, then we implemented fix for this issue by turning on 'Block public access (bucket settings)' and finally we confirmed that public access to our s3 bucket is blocked by reperforming the s3 only audit using Scout Suite.



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Technet24

# Audit 3:

- Enumerate the internal docker registries and identify various vulnerabilities in the images

## Solution

For demonstration purposes we will use a private registry hosted on GCR. We first authenticate to our Google cloud using file-based authentication using the following command.

### Command

```
gcloud alpha auth activate-service-account --key-file=gcp_creds.json
```

Once we are authenticated, let's try to list all the images hosted on our private registry under project 'cloud-hd-gcp' using the following command.

### Command

```
gcloud container images list --repository=gcr.io/cloud-hd-gcp
```

### Output :

```
NAME
gcr.io/cloud-hd-gcp/flag-image
gcr.io/cloud-hd-gcp/nginxnss
gcr.io/cloud-hd-gcp/vulhub_appweb
gcr.io/cloud-hd-gcp/vulnhub_jmeter_3.3
```

From the above output, we can observe that there are a total of 3 images present. Now submit the following command to list tags associated with either one of them.

### Command

```
gcloud container images list-tags gcr.io/cloud-hd-gcp/vulnhub_jmeter_3.3
```

### Output :

```
DIGEST        TAGS  TIMESTAMP
c1f0765cf101  tag1  2018-03-02T07:41:30
```

Now we have all the details required to run the scan that will identify vulnerabilities associated with them. But first, Run the following command to authenticate Docker with gcloud as credential helper.

### Command

```
docker login -u oauth2accesstoken -p "$(gcloud auth print-access-token)" https://gcr.io/v2
/
```

### Output :

```
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded
```

***Note:*** *There are multiple ways to authenticate to docker however this is easiest and almost always works hence we are using this method.*



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

```
sudo service docker start
```

Let’s run Trivy on docker image present in this account using below mentioned command:

### Command

```
trivy image gcr.io/cloud-hd-gcp/vulnhub_jmeter_3.3:tag1
```

### Output :

```
gcr.io/cloud-hd-gcp/vulnhub_jmeter_3.3:tag1 (debian 9.3)
========================================================
Total: 580 (UNKNOWN: 13, LOW: 208, MEDIUM: 114, HIGH: 184, CRITICAL: 61)

+-----------------------+-----------------+---------+-------------------------+--------------------------------+----------------------
--------------------------------------+
|        LIBRARY        | VULNERABILITY ID | SEVERITY |     INSTALLED VERSION      |          FIXED VERSION          |
     TITLE                             |
+-----------------------+-----------------+---------+-------------------------+--------------------------------+----------------------
--------------------------------------+
| apt                   | CVE-2019-3462    | HIGH     | 1.4.8                      | 1.4.9                           | Incorrect sanitation o
f                                     |
|                       |                  |          |                            |                                 | the 302 redirect field
 in                                   |
|                       |                  |          |                            |                                 | HTTP transport metho .
.....                                 |
|                       |                  |          |                            |                                 | -->avd.aquasec.com/nvd
/cve-2019-3462                        |
+                       +-----------------+---------+                            +--------------------------------+----------------------
--------------------------------------+
|                       | CVE-2020-27350   | MEDIUM   |                            | 1.4.11                          | apt: integer overflows
 and underflows                       |
|                       |                  |          |                            |                                 | while parsing .deb pac
kages                                 |
|                       |                  |          |                            |                                 | -->avd.aquasec.com/nvd
/cve-2020-27350                       |
```



NotSoSecure part of claranet cyber security

© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Technet24

# Audit 4:

In this exercise we will set a local cluster with multiple misconfigurations and potentially malicious resources.

Auditing a kubernetes environment can be divided into 2 parts:

- Auditing kubernetes cluster: in this stage we check for misconfiguration and malicious pods running the cluster
- Auditing kubernetes RBAC configurations: In this stage we try to identify over privileged users and service account tokens present in the cluster.

## Solution

Doing this manually may take lot of time depending on the size and resources present in the cluster. Hence we can use some of automated open-source tools to audit kubernetes environment.

It is also recommended to use multiple tools to ensure complete coverage of the audit

Some of the automated tools that we will using this exercise are as follows:

- **Kube-bench**: it checks if kubernetes environment is deployed securely
- **Kubeaudit**: This tools tries to identify malicious resources running inside the cluster
- **rback**: This tool can be used to visualize the rbac configuration applied in the cluster which could be used to identify over privileged service accounts and users
- **kubernetes-rbac-audit**: This tool tries to identify over privileged service accounts and users present in the cluster

Navigate to the folder where we have installed our scripts.

**Command:**

```
cd /home/pentester/k8sScript/audit
```

Once the error free execution of this script is complete we can confirm if following resources are running in the cluster or not.

**Command:**

```
kubectl get pod -A
```

**Output:**

```
NAMESPACE            NAME                                                   READY   STATUS
RESTARTS   AGE
default              everything-allowed-exec-deployment-6cd7685786-pvljc    1/1     Running
0          11m
default              hostipc-exec-statefulset-0                             1/1     Running
0          11m
default              priv-and-hostpid-exec-deployment-64dcfc557d-9v6qs      1/1     Running
0          11m
kube-system          coredns-558bd4d5db-d6m95                               1/1     Running
0          12m
kube-system          coredns-558bd4d5db-sp44p                               1/1     Running
0          12m
kube-system          etcd-audit-cluster-control-plane                       1/1     Running
0          12m
```



NotSoSecure part of claranet cyber security

© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

```
kube-system          kindnet-6r7pd                                          1/1     Running
0           12m
kube-system          kindnet-wvk7q                                          1/1     Running
0           11m
kube-system          kube-apiserver-audit-cluster-control-plane             1/1     Running
0           12m
kube-system          kube-controller-manager-audit-cluster-control-plane    1/1     Running
0           12m
kube-system          kube-proxy-mqjrm                                       1/1     Running
0           11m
kube-system          kube-proxy-trcxj                                       1/1     Running
0           12m
kube-system          kube-scheduler-audit-cluster-control-plane             1/1     Running
0           12m
local-path-storage   local-path-provisioner-547f784dff-7frxx                1/1     Running
0           12m
```

### Auditing kubernetes cluster

In this section we will check for misconfiguations along with potentially malicious resources present in kubernetes cluster

Navigate to the kube-bench folder and execute following command to run a job in the kubernetes cluster.

**Command:**

```
cd /home/pentester/tools/k8s-audit/kube-bench
```

This job will check for misconfiguations in the cluster.

**Command:**

```
kubectl apply -f job.yaml
```

**Output:**

```
job.batch/kube-bench created
```

Let's verfiy the pods.

**Command:**

```
kubectl get pods
```

**Output:**

```
NAME                                                  READY   STATUS      RESTARTS   AGE
everything-allowed-exec-deployment-6cd7685786-pvljc   1/1     Running     0          31m
hostipc-exec-statefulset-0                            1/1     Running     0          31m
kube-bench-rlj7h                                      0/1     Completed   0          15m
priv-and-hostpid-exec-deployment-64dcfc557d-9v6qs     1/1     Running     0          31m
```

Check the logs of the ## Please add the details of the below command.

**Command**:

```
kubectl logs <pod_id> # Pod ID of the newly created Kube-bench.
kubectl logs kube-bench-rlj7h
```

**Output:**

```
[INFO] 4 Worker Node Security Configuration
```

© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

Technet24

```
[INFO] 4.1 Worker Node Configuration Files
[PASS] 4.1.1 Ensure that the kubelet service file permissions are set to 644 or more restr
ictive (Automated)
[PASS] 4.1.2 Ensure that the kubelet service file ownership is set to root:root (Automated
)
--------------------------------Output Snipped---------------------------------

== Summary total ==
17 checks PASS
3 checks FAIL
29 checks WARN
0 checks INFO
```

Now that we have identified misconfiguration in the cluster,. lets remove the job created by kube-bench.

Note: Ensure that the you are in the directory (/home/pentester/tools/k8s-audit/kube-bench) before running the below directory.

**Command:**

```
kubectl delete -f job.yaml
```

**Output:**

```
job.batch "kube-bench" deleted
```

Let’s check for the potential resources present in the cluster using kubeaudit tool.

**Command:**

```
kubeaudit all
```

**Output:**

```
W0303 18:37:43.808982 1834340 warnings.go:70] batch/v1beta1 CronJob is deprecated in v1.21
+, unavailable in v1.25+; use batch/v1 CronJob

---------------- Results for ---------------

  apiVersion: apps/v1
  kind: Daemonset
  metadata:
    name: kindnet
    namespace: kube-system

--------------------------------------------

-- [error] AppArmorAnnotationMissing
   Message: AppArmor annotation missing. The annotation 'container.apparmor.security.beta.
kubernetes.io/kindnet-cni' should be added.
   Metadata:
      Container: kindnet-cni
      MissingAnnotation: container.apparmor.security.beta.kubernetes.io/kindnet-cni
```

### Auditing RBAC

To identify the over privileged and misconfigured account and users in the cluster we can use rback tool to visualize the RBAC configuration of the entire cluster.

To do so let’s navigate to the rback folder and Run the following commands.

**Command:**



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets

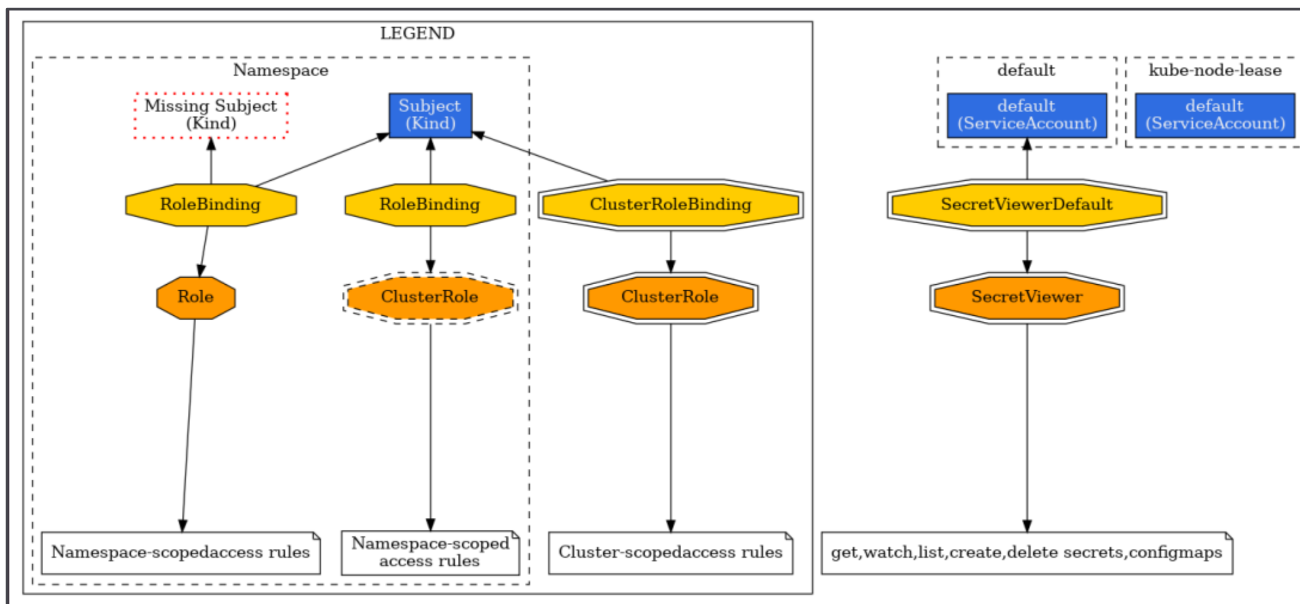
The following command will create the file result.dot in the folder.

```
kubectl get sa,roles,rolebindings,clusterroles,clusterrolebindings --all-namespaces -o js
on | rback > result.dot
```

Covert the .dot file to the PNG format or graphical view, using the following command:

```
dot -Tpng result.dot > rback.png
```

Now open the rback.png in the image viewer and you should be able to see visual representation of RBAC configuration of the cluster




We can also use kubernetes-rbac-audit to identify over privileged and misconfigured account and users in the cluster.

Just navigate to the kubernetes-rbac-audit folder.

**Command:**

```
cd /home/pentester/tools/k8s-audit/kubernetes-rbac-audit
```

Run following commands.

**Commands:**

```
kubectl get roles --all-namespaces -o json > Roles.json

kubectl get clusterroles -o json > clusterroles.json

kubectl get rolebindings --all-namespaces -o json > rolebindings.json

kubectl get clusterrolebindings -o json > clusterrolebindings.json
```

Verify the RBAC permission of the cluster.

**Command:**

```
python3 ExtensiveRoleCheck.py --clusterRole clusterroles.json  --role Roles.json --rolebin
dings rolebindings.json --cluseterolebindings clusterrolebindings.json
```

**Output:**


© Claranet Cyber Security. All rights reserved
https://t.me/learningnets
Technet24

```
[*] Started enumerating risky ClusterRoles:
[!][ClusterRole]→ SecretViewer Has permission to list secrets!
[!][ClusterRole]→ anon-user Has permission to create pods!
[!][ClusterRole]→ anon-user Has permission to use pod exec!
[!][ClusterRole]→ local-path-provisioner-role Has permission to access pods with any verb!
[*] Started enumerating risky Roles:
[*] Started enumerating risky ClusterRoleBinding:
[!][ClusterRoleBinding]→ SecretViewerDefault is binded to default ServiceAccount.
[!][ClusterRoleBinding]→ anon-to-pod is binded to the Group: system:unauthenticated!
[!][ClusterRoleBinding]→ local-path-provisioner-bind is binded to local-path-provisioner-s
ervice-account ServiceAccount.
[*] Started enumerating risky RoleRoleBindings:
```

And we were able to identify the high-privileged users and service accounts present in the cluster.

Destroy the cluster by running the following commands.

Navigate to the **k8sScript** folder.

**Command:**

```
cd /home/pentester/k8sScript/audit
```

**Command:**

```
./destroy-audit-cluster.sh
```

To verify if cluster is deleted, use following command

```
# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

you should see no docker containers running.



© Claranet Cyber Security. All rights reserved

https://t.me/learningnets