Network filtering in a distributed app

Brian Cheung and Sam Wang

ABSTRACT

We will examine the vulnerabilities of a distributed application without using any layers of security. Then we will implement Pod Security Policies and Data Plane Filters to imporve the security of the application itself and filter out suspicious network traffic. The filtering of certain packets based on information such as the source/destination IP addresses and protocols can further reduce the attack surface and strengthen current security practices for applications. Through our research, we hope that the implementation of filtering out network traffic will lead to an increase in security performance.

1. INTRODUCTION

There exists many vulnerabilities within networks and distributed applications, and the implementation of network filtering as well as strong pod security policies can help mitigate these threats.

Figuring out how to effectively utilize network filtering and pod security policies to work together will hopefully result in a strong security infrastructure that can successfully defend against all attacks and will be adopted by all applications.

We will be researching how network filtering and pod security policies function and how they can be applied to improving the security of an application. By understanding how each work, we will be able to understand how to effectively filter out network traffic that seems suspicious, thus reducing the amount of potential attacks from the start, and we will understand how to write a concrete and complete pod security policy to defend against the attacks that slip through the filter.

Pod Security Policies (PSP) provides a framework that will layout the rules of how a pod can operate and ensures that they run with the correct privileges and access. Furthermore, PSPs are used so that those operating Kubernetes clusters can control pod creations and limit what pods can access. When a pod is deployed, the PSP acts as a gatekeeper that will compare the pod security configuration to what is defined in the PSP. Some examples of how PSPs can limit pod behaviors include, preventing privileged pods from starting and controlling its privilege, restricting pods from accessing host namespaces, filesystem, and networks, restrict the amount of user/groups that a pod can run, and more. Network filtering is the practice of monitoring the inflow/outflow of packets in a network. There are two types of filtering, ingress and egress filtering. Ingress filtering is the technique of monitoring incoming packet data. It is considered the first-line of defense in a network because it blocks out unwanted inflow traffic to the network. While this isn't a robust and complete form of defense, it's beneficial because it can greatly reduce the load on some proxy or firewall, and it's an effective for getting rid of the majority of unwanted traffic. Egress filtering is the technique of monitoring the flow of outbound network traffic and prevents any outbound connections to potential threats/unwanted hosts. Egress filtering can be used to disrupt malware, block unwanted services, and gives greater awareness of network traffic. First, we will setup a simple application and document the vulnerabilities. Next, we will implement pod security policies and document the results. Then we will implement network filtering on the base application without pod security policies and document the results. Finally, we will combine both layers of security and document the results. Hopefully, the final application will be free of vulnerabilities.

2. MOTIVATION

It is important to continue researching how network filtering can lead to better application security, especially with a lot of technologies and applications all moving into the cloud. A strong initial filter that effectively identifies a majority of attacks dramatically decreases the chance of a successful attack. If most of the attempted attacks are initially thwarted, there are few attacks that can get through the security in place. A good network filter means less attacks an application has to be wary of, which should lead to a higher defense rate.

3. OUR ARCHITECTURE

For this project, we have two Dockerfiles that create docker images that use the Debian Linux distribution as the base. The `root-debian` image runs as `root` with root privelages by default, while the `non-root-debian` image runs as `appuser` without root privileges. These two were built using the `docker-compose` in the `app` directory.

We also created Pod Security Policies to enforce specific rules in the Kubernetes cluster.

4. EXPERIMENTAL RESULTS

First, we will improve the base security of the application through Pod Security Policies. Then we will add another layer of security through Data Plane filters in order to reduce the attack surface and prevent suspicious activity. These extra layers of security should hopefully eliminate the vulnerabilities of the application.

4.1 Without Pod Security Policies

Both pods successfully deploy and run in the cluster **??**. On the `non-root-debian` pod, the user can escalate privilages to run as the `root` user **??**.

4.2 With Pod Security Policies

With a restrictive PSP that requires the user to run as a non-root user, the `non-root-debian` pod successfully runs while the `root-debian` pod is prevented from running with a `CreateContainerConfigError` as shown in Figure 3 and Figure 4. The PSP prevents the `root-debian` pod from running because the policies does not allow the pod to run as the `root` user.

The PSP also prevents privilege escalation as well. The `non-root-debian` pod successfully runs as a non-root user. The same privelege escalation method is tried with this pod after applying PSP, and as a result, the privilege escalation fails as shown in Figure 5.

```
NAME                                 READY   STATUS    RESTARTS   AGE   IP            NODE      NOMINATED NODE   READINESS GATES
pod/non-root-debian-5b8bbd6c5-xx966   1/1     Running   0          15s   10.1.56.99    bcheung   <none>           <none>
pod/root-debian-69dcfd6f4c-7sv4g      1/1     Running   0          14s   10.1.56.100   bcheung   <none>           <none>

NAME                          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE     SELECTOR
service/kubernetes            ClusterIP   10.152.183.1    <none>        443/TCP        3d20h   <none>
service/non-root-debian-svc   NodePort    10.152.183.37   <none>        80:30000/TCP   14s     app=non-root-debian
service/root-debian-svc       NodePort    10.152.183.70   <none>        80:30001/TCP   15s     app=root-debian

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS        IMAGES                                   SELECTOR
deployment.apps/non-root-debian   1/1     1            1           15s   non-root-debian   localhost:32000/app_non_root_debian:k8s   app=non-root-debian
deployment.apps/root-debian       1/1     1            1           14s   root-debian       localhost:32000/app_root_debian:k8s       app=root-debian

NAME                                        DESIRED   CURRENT   READY   AGE   CONTAINERS        IMAGES                                   SELECTOR
replicaset.apps/non-root-debian-5b8bbd6c5   1         1         1       15s   non-root-debian   localhost:32000/app_non_root_debian:k8s   app=non-root-debian,pod-template-hash=5b8bbd6c5
replicaset.apps/root-debian-69dcfd6f4c      1         1         1       14s   root-debian       localhost:32000/app_root_debian:k8s       app=root-debian,pod-template-hash=69dcfd6f4c
```

Figure 1: Screenshot of the Kubernetes resources after deploying the pods without Pod Security Policies

```
bcheung@bcheung:~/Documents/EE379K-Final-Project$ microk8s.kubectl exec -it non-root-debian-5b8bbd6c5-xx966 bash
appuser@non-root-debian-5b8bbd6c5-xx966:/$ ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
appuser@non-root-debian-5b8bbd6c5-xx966:/$ cd root
bash: cd: root: Permission denied
appuser@non-root-debian-5b8bbd6c5-xx966:/$ su root
Password:
root@non-root-debian-5b8bbd6c5-xx966:/# cd root
root@non-root-debian-5b8bbd6c5-xx966:~# cat secrets.txt
This is a secret that only root has access to.
```

Figure 2: Screenshot of the privilege escalation on the non-root-debian pod

```
NAME                                 READY   STATUS                       RESTARTS   AGE     IP            NODE      NOMINATED NODE   READINESS GATES
pod/non-root-debian-5b8bbd6c5-cjtl4   1/1     Running                      0          2m18s   10.1.56.103   bcheung   <none>           <none>
pod/root-debian-69dcfd6f4c-b8kgz      0/1     CreateContainerConfigError   0          2m17s   10.1.56.104   bcheung   <none>           <none>

NAME                          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE     SELECTOR
service/kubernetes            ClusterIP   10.152.183.1    <none>        443/TCP        4d1h    <none>
service/non-root-debian-svc   NodePort    10.152.183.21   <none>        80:30000/TCP   2m17s   app=non-root-debian
service/root-debian-svc       NodePort    10.152.183.31   <none>        80:30001/TCP   2m18s   app=root-debian

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE     CONTAINERS        IMAGES                                   SELECTOR
deployment.apps/non-root-debian   1/1     1            1           2m18s   non-root-debian   localhost:32000/app_non_root_debian:k8s   app=non-root-debian
deployment.apps/root-debian       0/1     1            0           2m17s   root-debian       localhost:32000/app_root_debian:k8s       app=root-debian

NAME                                        DESIRED   CURRENT   READY   AGE     CONTAINERS        IMAGES                                   SELECTOR
replicaset.apps/non-root-debian-5b8bbd6c5   1         1         1       2m18s   non-root-debian   localhost:32000/app_non_root_debian:k8s   app=non-root-debian,pod-template-hash=5b8bbd6c5
replicaset.apps/root-debian-69dcfd6f4c      1         1         0       2m17s   root-debian       localhost:32000/app_root_debian:k8s       app=root-debian,pod-template-hash=69dcfd6f4c
```

Figure 3: Screenshot of the Kubernetes resources after applying Pod Security Policies and deploying the pods

```
Conditions:
  Type              Status
  Initialized       True
  Ready             False
  ContainersReady   False
  PodScheduled      True
Volumes:
  default-token-6clvb:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-6clvb
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type     Reason     Age               From               Message
  ----     ------     ----              ----               -------
  Normal   Scheduled  <unknown>         default-scheduler  Successfully assigned default/root-debian-69dcfd6f4c-b8kgz to bcheung
  Normal   Pulled     7s (x2 over 7s)   kubelet, bcheung   Container image "localhost:32000/app_root_debian:k8s" already present on machine
  Warning  Failed     7s (x2 over 7s)   kubelet, bcheung   Error: container has runAsNonRoot and image will run as root
```

Figure 4: Screenshot of the root-debian pod's status after applying the restrictive PSP and deploying

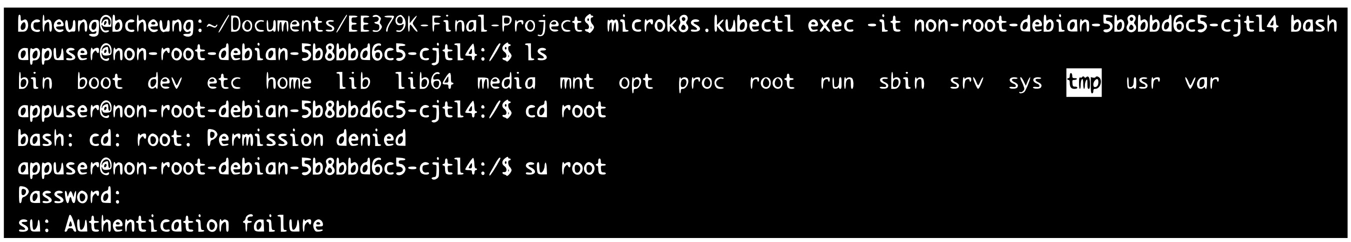

Figure 5: The non-root-debian pod fails to escalate privileges and login as the root user with the restrictive PSP

4.3 With Envoy

Envoy is to be used alongside the application that we are testing and will be used to monitor the inflow and outflow of traffic. With envoy, we will be able to see how well the implemented pod security policies are working.

5. RELATED WORK

[6] The article talks about how more and more enterprises are moving their workloads onto the cloud and while security has evolved over time, is still a major concern. The paper goes into details about the various forms of threats and vulnerabilities of the cloud, specifically listing and detailing 17 threats. This paper provides a good foundation for understanding common threats that exploit cloud vulnerabilities.

[4] This paper discusses the threats that networks face and the current network security practices to counteract these attacks. The paper begins by detailing security attacks, security measures, and security tools. The paper goes into great detail about different security methods, such as application gateways and packet filtering. The paper discusses different things that organizations can do to prepare for these attacks and the various technology options.

[2] This paper further discusses the vulnerabilities of cloud computing services. The paper details cloud service models and talks about the 3 layers of cloud computing: system layer (IaaS), platform layer (PaaS), and application layer (SaaS). The paper then analyzes the various security issues that each layer faces and talks about the threats that exploit those vulnerabilities.

[3] This paper discusses the advantages of using cloud services and also reveals the dangers and risks of those services.

[1] This is a known vulnerability, CVE-2019-5736 [1], that allows attackers to execute commands as root within two types of containers, a new container with an attack-controlled image and an already existing container that an attacker has had access to in the past.

[5] This article goes into detail about what packet filtering is and how it is used as a network security tool. The paper details the benefits of packet filters and gives a simple implementation of it and discusses the limitations of packet filtering.

6. CONCLUSIONS

7. REFERENCES

[1] Cve-2019-5736.

[2] T.-S. Chou. Security threats on cloud computing vulnerabilities.

[3] M. Kemal. Cloud security.

[4] S. Pandey. Modern network security: Issues and challenges.

[5] D. Strom. The packet filter: A basic network security tool.

[6] P. S. Suryateja. Threats and vulnerabilities of cloud computing: A review.