Security in a Distributed Application

Brian Cheung and Sam Wang

Electrical and Computer Engineering, Cockrell, University of Texas at Austin

Abstract

Container orchestration technologies like Kubernetes provide a simple way to deploy, scale, and manage containerized applications [2]. However, as with any production-level system, these distributed systems must be properly secured at each appropriate level. In this project, we examine common vulnerabilities of distributed applications and demonstrate how we can help mitigate them through Pod Security Policies and network proxies like Envoy. We first implement Pod Security Policies to improve the base security of the application itself by limiting the pod's capabilities and only allowing appropriate privileges. Then we implement Envoy with Open Policy Agent to reinforce the network security by only accepting specific requests based on client privileges. As shown through our examples, these additional layers of security can help mitigate common vulnerabilities.

Motivation

With the increasing popularity of distributed applications deployed with container orchestration technologies like Kubernetes, developers must understand how to reinforce the security of these distributed applications. Additionally, we must consider additional layers of security, which depend on the type of application and its exposure to other services. Often times, services communicate with other services through a networks, so a network proxy that filters incoming and outgoing requests can help protect the service itself along with the clients who use it. Securing an application at multiple levels allows developers to reduce the overall attack surface by making it more difficult for attackers to find vulnerabilities.

Common Vulnerabilities and Threats

Data Breach/Loss: Attackers will target sensitive/confidential/personal data that some application holds.
Denial of Service: This type of attack can appear in the form of a Denial of Service (DoS) or Distributed Denial of Service (DDoS). In a DoS attack, one machine will start the attack and prevents legitimate users from using some service/application. A DDoS attack is when an attacker will take control of multiple systems to use to take down some application.
Vulnerable Cloud API and Systems: By accidentally exposing an API to the public, attackers can exploit that API and can attack a system.
Malicious Insider(s): When someone attacks from within the application, like an employee, system administrator, business partner, etc. This type of attack can get access to sensitive information and can spread malware throughout the entire system or even to clients.
Account Hijacking: When an account's credentials are compromised by a hacker and used for purposes against the owner's will.
Lack of Due Diligence: Due diligence is the practice of knowing and applying the best security practices. The lack of a good foundation and implementation of best security practices leaves a service extremely vulnerable to attacks. [3]

Pod Security Policies (PSP)

Pod Security Policies (PSP) are policies that define rules/conditions that a pod must run with in order to function within the cluster of pods. This ensures that pods can and will run only if it maintains the correct privileges and resources defined in the PSPs. 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 privilege escalation, restricting pods from accessing host namespaces, filesystem, and networks, restrict the amount of user/groups that a pod can run, and more.

Examples

```
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
```

Fig. 1: Current Resources with restricitve PSP applied.

```
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
```

Fig. 2: Describing Pod with root as default user.

```
bcheung@bcheung:~/Documents/EE379K-Final-Project$ microk8s.kubectl exec -it non-root-debian-5b8bbd6c5-cjtl4 bash
appuser@non-root-debian-5b8bbd6c5-cjtl4:/$ 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-cjtl4:/$ cd root
bash: cd: root: Permission denied
appuser@non-root-debian-5b8bbd6c5-cjtl4:/$ su root
Password:
su: Authentication failure
```

Fig. 3: Attempting Privilege Escalation with PSP applied.

```
stanqinlee-VirtualBox:~$ curl -L -H "Authorization: Bearer "$BRIAN_TOKEN"" http://$SERVICE_URL/people
HTTP/1.1 200 OK
content-type: application/json
date: Sat, 14 Dec 2019 19:51:57 GMT
content-length: 151
x-envoy-upstream-service-time: 0
server: envoy

[{"id":"1","firstname":"John","lastname":"Doe"},{"id":"2","firstname":"Jane","lastname":"Doe"},{"id":"498081","firstname":"Charlie","lastname":"Opa"}]
stanqinlee-VirtualBox:~$ curl -L -H "Authorization: Bearer "$BRIAN_TOKEN"" -d '{"firstname":"Charlie", "lastname":"OPA"}' -H "Content-Type: application/json" -X POST http://$SERVICE_URL/people
HTTP/1.1 403 Forbidden
date: Sat, 14 Dec 2019 19:52:24 GMT
server: envoy
content-length: 0
```

Fig. 4: Envoy with OPA Policy Guest.

```
stanqinlee-VirtualBox:~$ curl -L -H "Authorization: Bearer "$SAM_TOKEN"" http://$SERVICE_URL/people
HTTP/1.1 200 OK
content-type: application/json
date: Sat, 14 Dec 2019 19:52:33 GMT
content-length: 151
x-envoy-upstream-service-time: 0
server: envoy

[{"id":"1","firstname":"John","lastname":"Doe"},{"id":"2","firstname":"Jane","lastname":"Doe"},{"id":"498081","firstname":"Charlie","lastname":"Opa"}]
stanqinlee-VirtualBox:~$ curl -L -H "Authorization: Bearer "$SAM_TOKEN"" -d '{"firstname":"Charlie", "lastname":"Opa"}' -H "Content-Type: application/json" -X POST http://$SERVICE_URL/people
HTTP/1.1 200 OK
content-type: application/json
date: Sat, 14 Dec 2019 19:52:41 GMT
content-length: 53
x-envoy-upstream-service-time: 0
server: envoy

{"id":"727887","firstname":"Charlie","lastname":"Opa"}
```

Fig. 5: Envoy with OPA Policy Admin.

Envoy

Envoy is an open source network proxy that can be customized to support TCP/HTTP proxy, TLS authentication, load balancing, etc. Envoy aims to achieve network and application transparency and complete communication between the two. In our report, we demonstrated Envoy's ability to secure network traffic with HTTPS and SSL certificates as well as Envoy's external authorization filter functionality, specifically applying a policy written with Open Policy Agent [1].

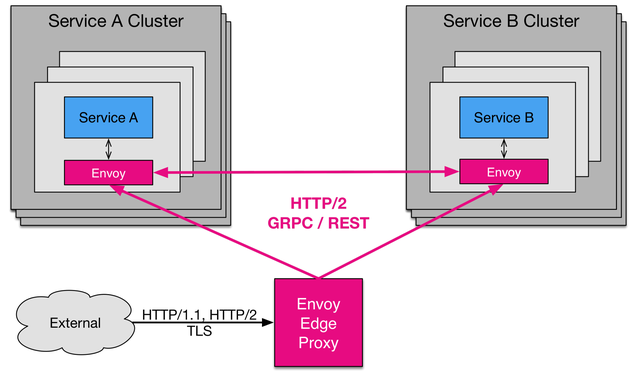

Fig. 6: Envoy Diagram.

Comparison

While Pod Security Policies and network proxies have similar goals of maintaining/strengthening the security of an application, they do have different use cases and have unique strengths when it comes to application/network security. Pod Security Policies secure the base application and the Kubernetes cluster that the application is hosted on. On the other hand, a network proxy is the middleman between the client and the network and thus is met with every network request. Furthermore, designing a network proxy that can effectively handle these network requests, whether that be filtering out malicious requests, or assigning security policies onto API requests is extremely important in protecting the overall security integrity of the application as well as reducing the load that the rest of the application has to deal with. Network proxies can be viewed as the outer line of defense, while Pod Security Policies act as the base-line defense that protects the system if a threat manages to bypass the network proxy. Both services add a layer of security that helps mitigate threats at different levels of the application, which ultimately minimizes the attack surface of the application.

References

[1] Envoy. "What is Envoy". https://www.envoyproxy.io/docs/envoy/latest/intro/what_is_envoy.

[2] Kubernetes. "Production-Grade Container Orchestration". https://kubernetes.io/.

[3] Pericherla Satya Suryateja. *Threats and Vulnerabilities of Cloud Computing: A Review.* https://www.researchgate.net/publication/324562008_Threats_and_Vulnerabilities_of_Cloud_Comput