Aqua Blog

Kubernetes Pod Escape Using Log Mounts

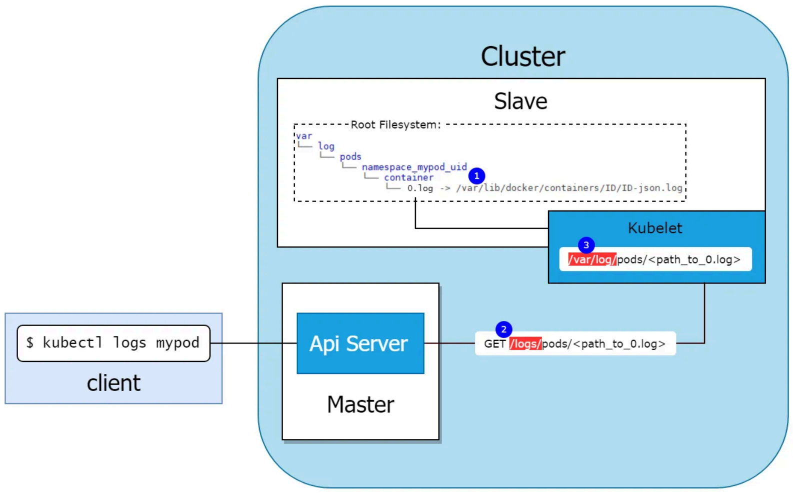
Aqua Research Team August 1, 2019

Kubernetes has many moving parts, and sometimes combining them in certain ways can create unexpected security flaws. In this post you'll see how a pod running as root and with a mount point to the node's `/var/log` directory can **expose the entire contents of its host filesystem** to any user who has access to its logs. We'll also talk about your options to mitigate this issue in your cluster.

Understanding how kubernetes sees logs

Have you ever wondered how `kubectl logs <pod_name>` retrieves logs from your pod? Who's

The following diagram illustrates how this works:



The kubelet creates a directory structure inside the `/var/log` directory on the host, representing pods on the node. Inside the directory for our pod we see the 0.log file (**1**), but it's really a symlink (Symbolic Link) to the container log file that lives within the `/var/lib/docker/containers` directory. This is all from the host's perspective.

The kubelet exposes a `/logs/` endpoint (**2**) that simply operates an HTTP file server in the `/var/log` directory (**3**), making the log files accessible to requests that come from the API Server.

Now, imagine if we were to deploy a pod with a `hostPath` mount to `/var/log`. The pod will have access to all pod log files on that host. While that could constitute a problem on its own, we can take the next logical step. What if we replace the `0.log` file with a symlink to… let's say, `/etc/shadow`

```
|
├── var
|  ├── logs
|  |  ├── pods
|  |  |  ├── default_mypod_e7869b14-abca-
11e8-9888-42010a8e020e
|  |  |  |  ├── mypod
```

`kubectl logs` on our client machine

```
$ kubectl logs mypod
failed to get parse function: unsupported
log format: "root:*:18033:0:99999:7:::n"
```

The kubelet follows the symlink and reads the content of whatever file it points to, which could be any file on the node.

kubectl failed after the first line because it expected a json format, but we could easily read specific lines of the shadow file, by passing the `--tail=-<line_number>` flag.

That's amazing. Because the symlink is followed by the kubelet, we can exploit the kubelet's root permissions to read any file on the node, just by creating a symlink within the pod.

Pod Escape

Let's take this a step further. We know that by running as a pod in Kubernetes, a service account token gets installed, so if the service account permits log access, we could directly access the kubelet and elevate ourselves to the root on the node.

I've created a proof of concept (POC) to demonstrate the attack vector:

- Deploying a pod with a mount point to `/var/log`
- Creating a symbolic link to the root folder on the host
- Reading an ssh private key of a user on the host

The following video shows two custom commands running from within a pod:

All the files used in this POC can be found under this GitHub repository. In the same repository there is another POC script that automatically extracts private keys, and Kubernetes service–account tokens from the host filesystem.

Mounting directories can be dangerous

So, is this a vulnerability in Kubernetes or just bad practice?

Deploying a pod with a writeable `hostPath` to `/var/log` is rare (and there are other ways through which mounting sensitive host directories into a pod can be abused). But even if you knew that mounting /var/log was likely to be sensitive, you probably didn't expect that it could lead to a node takeover so easily.

We raised this issue with the Kubernetes security team before publishing this to check whether they consider it a vulnerability. They concluded that it's just the unfortunate consequence of mounting a writable sensitive host directory – and the risks of that are documented. That said, this one is relatively easy to abuse. There are currently many projects that use this mount. If you use one of

This method was tested on Kubernetes 1.15 and 1.13 but is likely to affect other versions as well.

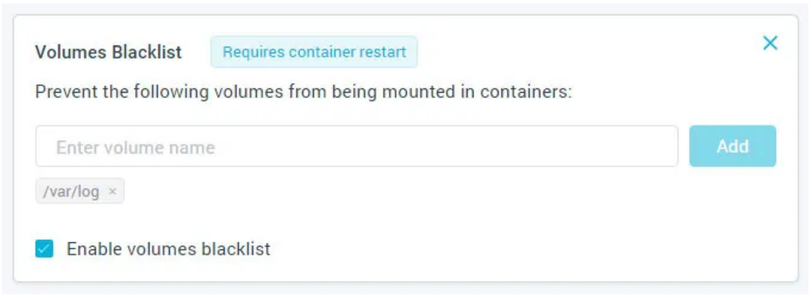
Mitigation

This escape is only possible if the pod is running as root. In general this is to be avoided, and it's easy to set up a policy in Aqua CSP to prevent containers running as root, or to approve only a certain group of images for which root is a requirement.

Another course of action is simply not to deploy pods with a writeable `hostPath` to `/var/log`. It isn't a default setting or common practice, so someone needs to deliberate define it this way, but it's still a possibility. But how do you check it?

We've added a new hunter for kube-hunter, our open source pen–testing tool for Kubernetes, that checks your cluster for the existence of a pod with this dangerous mount.

Aqua customers can also prevent this type of risk by using our runtime policy to blocklist volume mounts from the `/var/log` path:


Volumes Blacklist Requires container restart

Prevent the following volumes from being mounted in containers:

Enter volume name Add

/var/log ×

✓ Enable volumes blacklist

Summary

Kubernetes is a complex
system with security dependencies that aren't always evident to the average, or even expert, user. I showed how under certain circumstances innocent logging could lead to potential exploits. Under most conditions, this will not be possible, but Kubernetes

such mistakes.

Published under: RESEARCH
Tags: Kubernetes Security, Security Threats

Aqua Research Team

Team Nautilus focuses on cybersecurity research of the cloud native stack. Its mission is to uncover new vulnerabilities, threats and attacks that target containers, Kubernetes, serverless, and public cloud infrastructure — enabling new methods and tools to address them.