SANS DFIR Summit 2023

EKS Incident Response and Forensic Analysis

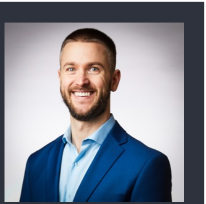
> Jonathon Poling

Principal Consultant

```
kubectl apply -f presentation.yaml
```

0 0 1 0 1   0 1 0 1 1   1 0 0 1 1   0 0 1 0 0   0 0 1 1 0   0 1 0 0 1   1 0 0 1 0

# About Me



- Spent my entire career (16+ years now) focused on DFIR
- SME in all major operating systems (Windows/Linux/Mac) + Cloud (AWS)
- Worked across all types of industry (Gov't, private, public) honing my skills
- Have helped all types of customers from "mom and pop" to Fortune 10 build, protect, and defend their org's from threats
- Driven to tackling the hardest, most challenging, and/or unaddressed problems (hence, EKS DFIR)
- Always interested in learning, teaching, and making tangible differences in the security/DFIR landscape

# Agenda

{01}

Introduction

{02}

EKS Data / Artifacts

{03}

EKS Incident Response and Investigation

{04}

Conclusion

0 0 1 0 1 0 1 0 1 1 1 0 0 1 1 0 0 1 0 0 0 0 1 1 0 0 1 0 0 1 1 0 0 1 0

# #

Introduction

01

0 0 1 0 1 0 1 0 1 1 1 0 0 1 1 0 0 1 0 0 0 0 1 1 0 0 1 0 0 1 1 0 0 1 0

# Intro [Current Problem]

- ✓ AWS Elastic Kubernetes Service (EKS) is becoming increasingly popular
- ✓ (Ergo…) Effectively securing and investigating EKS incidents is becoming increasingly important
- ✓ Numerous higher-level blog posts and articles on “how to respond” to EKS incidents
- ✓ Very few (if any) low-level walkthroughs with prescriptive guidance on *how to comprehensively investigate* Kubernetes incidents

Today we change this.

Obligatory Stock Photo

[Pretty cool, yeah?]

0 0 1 0 1 0 1 0 1 1 1 0 0 1 1 0 0 1 0 0 0 0 1 1 0 0 1 0 0 1 1 0 0 1 0

#

EKS Data / Artifacts

02

0 0 1 0 1 0 1 0 1 1 1 0 0 1 1 0 0 1 0 0 0 0 1 1 0 0 1 0 0 1 1 0 0 1 0

EKS Data / Artifacts [Disk]

- EKS leverages Docker (or Containerd) under the hood to run its containers
- Docker uses the OverlayFS filesystem/drivers to run containers
- All containers (and data) located under /var/lib/docker/* on the local (Linux) filesystem
- Container info (Image ID, Mount Points, etc.) located in /var/lib/docker/containers/<Container-ID>/config.json (or config.v2.json)
- Each image layer (filesystem for use by containers) has its own directory under /var/lib/docker/overlay/* or /var/lib/docker/overlay2/* (latter is latest for best performance)
- Images are stored by ID (e.g., /var/lib/docker/overlay2/<Image-ID>/*)
- It’s possible to do live interrogation/investigation of Docker containers, BUT ... often best to collect a full image of the entire system to perform comprehensive investigation

EKS Data / Artifacts [Memory]

- Each Container runs in a separate/dedicated process on the system
- On EKS, this means each Container will run as a child process of containerd-shim (latest EKS versions use containerd runtime instead of docker)
- Each Pod runs in a dedicated sub-process of the parent Container process (containerd-shim)
- For example, a system (Node) running 5 containers would have 5 container-shim processes running associated Pods and sub-processes
- It’s possible to attempt to collect from a singular Container process, BUT ...
- Best practice is to collect memory from the entire system (Node) to ensure you have a full comprehensive view of the system for analysis (i.e., what if the entire Node is compromised and there are other compromised Pods running?)

EKS Data / Artifacts [Logs]

- The following control plane / audit logs exist for EKS:

API server (api) – API related logs, details, and errors
Audit (audit) – actions/activities performed on cluster
Authenticator (authenticator) – IAM/RBAC authentication logs
Controller manager (controllerManager) – Node/Pod operations on cluster
Scheduler (scheduler) – when/where Pods are assigned and run

- CloudTrail logs available for AWS Control Plane actions performed (Creating/Deleting/Managing Clusters)
- Logs produced by each Pod on the local Node (need fetched/exported)
- Ensure you enable ALL the above logs for effective monitoring and investigation

#

EKS Incident Response and Investigation

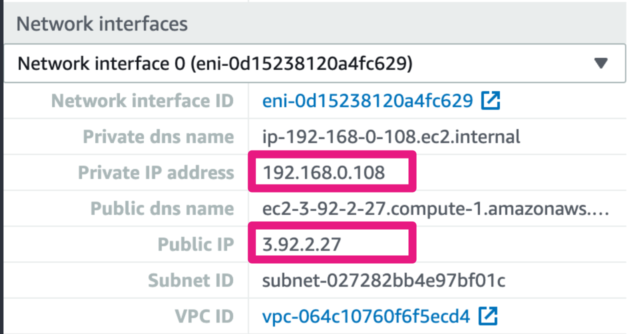
03

0 0 1 0 1 0 1 0 1 1 1 0 0 1 1 0 0 1 0 0 0 0 1 1 0 0 1 0 0 1 1 0 0 1 0

Identification

# Via GuardDuty

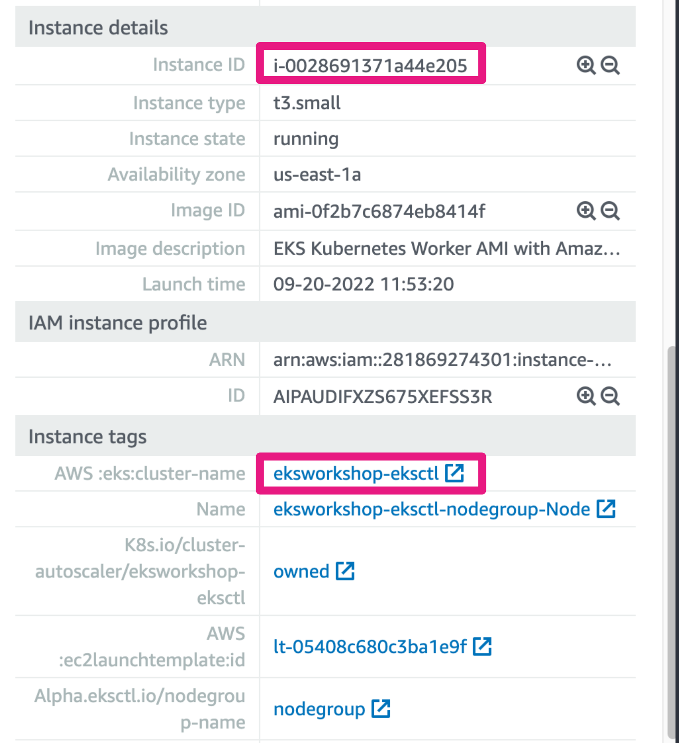
- Identify Cluster Info
- Identify Instance ID
- Identify Private and Public IP's



Network interfaces

Network interface 0 (eni-0d15238120a4fc629) ▼

Network interface ID	eni-0d15238120a4fc629
Private dns name	ip-192-168-0-108.ec2.internal
Private IP address	192.168.0.108
Public dns name	ec2-3-92-2-27.compute-1.amazonaws....
Public IP	3.92.2.27
Subnet ID	subnet-027282bb4e97bf01c
VPC ID	vpc-064c10760f6f5ecd4



Instance details

Instance ID	i-0028691371a44e205
Instance type	t3.small
Instance state	running
Availability zone	us-east-1a
Image ID	ami-0f2b7c6874eb8414f
Image description	EKS Kubernetes Worker AMI with Amaz...
Launch time	09-20-2022 11:53:20

IAM instance profile

ARN	arn:aws:iam::281869274301:instance-...
ID	AIPAUDIFXZS675XEFSS3R

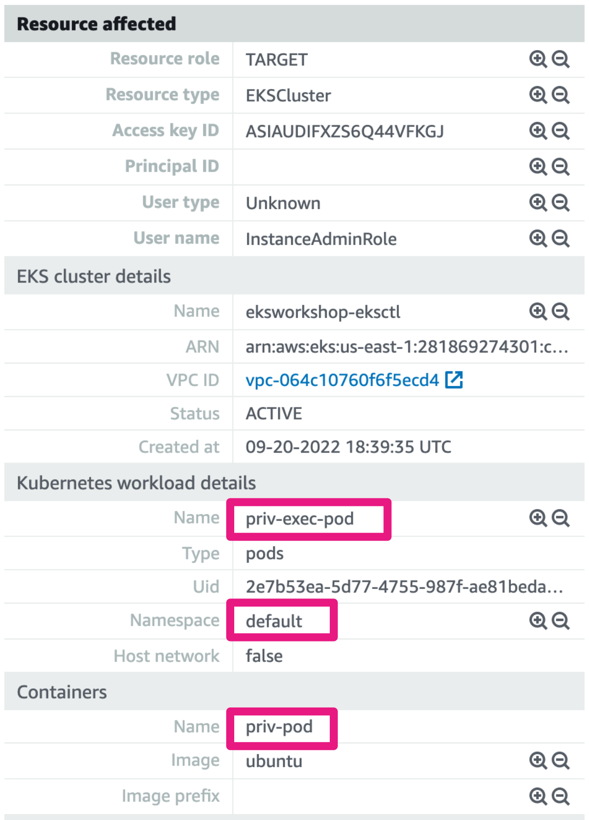
Instance tags

AWS :eks:cluster-name	eksworkshop-eksctl
Name	eksworkshop-eksctl-nodegroup-Node
K8s.io/cluster-autoscaler/eksworkshop-eksctl	owned
AWS :ec2launchtemplate:id	lt-05408c680c3ba1e9f
Alpha.eksctl.io/nodegroup-name	nodegroup

Identification

# Via GuardDuty

- Identify Cluster
- Identify Workload Name
- Identify Namespace
- Identify Container / Image Info



Resource affected	
Resource role	TARGET
Resource type	EKSCluster
Access key ID	ASIAUDIFXZS6Q44VFKGJ
Principal ID	
User type	Unknown
User name	InstanceAdminRole
EKS cluster details	
Name	eksworkshop-eksctl
ARN	arn:aws:eks:us-east-1:281869274301:c...
VPC ID	vpc-064c10760f6f5ecd4
Status	ACTIVE
Created at	09-20-2022 18:39:35 UTC
Kubernetes workload details	
Name	priv-exec-pod
Type	pods
Uid	2e7b53ea-5d77-4755-987f-ae81beda...
Namespace	default
Host network	false
Containers	
Name	priv-pod
Image	ubuntu
Image prefix	

Identification

# Via Kubectl (for compromised Node)

Get Node Information based on IP

```
$ kubectl get nodes -o wide | grep <PrivateIP>
```

Identify Instance ID of Node

```
$ kubectl get nodes <nodename> -n <namespace> -o custom-columns=NAME:.metadata.name,INSTANCEID:.spec.providerID
  -- OR --
$ kubectl get nodes <nodename> -n <namespace> -o custom-columns=NAME:.metadata.name,INSTANCEID:.spec.providerID | sed -e 's/aws:.*\///g'
```

Label the Node

```
$ kubectl label node <nodename> phase=QUARANTINE
```

Identification

# Via Kubectl (for compromised Pod)

Identify Node associated with Pod and Namespace

```
$ kubectl get pods <podname> -n <namespace> -o wide –show-labels
```

Identify Instance ID of Node where Pod is running

```
$ kubectl get nodes <nodename> -n <namespace> --show-labels -o wide
  -- OR --
$ kubectl get nodes <nodename> -n <namespace> -o custom-columns=NAME:.metadata.name,INSTANCEID:.spec.providerID | sed -e 's/aws:.*\///g'
```

Label the Pod

```
$ kubectl label pod <podname> phase=QUARANTINE
```

Data Acquisition

# Disk / Memory Acquisition

- Enable Termination Protection on the Instance
- Ensure Instance Shutdown behavior is set to “Stop”
- Tag the Instance (according to your needs/policies)
- Identify Volumes attached to the Instance
- Disable “Delete on Termination” setting for each Volume
- Acquire Snapshot of each Volume
- Acquire memory from Instance (your choice of method)

Data Acquisition

# Control Plane Logs

- Ideally, both CloudTrail and EKS audit logs were enabled a long time ago and reside in accessible storage
- (Optional) You can acquire/query specific Pod(s) logs via kubectl

Fetch logs for an active Pod/Container

```
$ kubectl logs <podname> [-c <containername>]
```

Fetch logs for a previously running Pod

```
$ kubectl logs –p <podname>
```

Initial Containment

# Node Containment

Isolate the Node

```
$ kubectl cordon <nodename>
```

# Pod Containment

- Develop a default-deny policy for the associated Pod (update the policy with the appropriate tags before running)

Isolate the Pod

```
$ kubectl apply -f pod-isolation-default-deny.yaml
```

Initial Containment

# Instance Containment

- Leverage appropriate Security Group, NACL, Firewall, etc. mechanisms to effectively isolate the Instance
- Remove or update (with appropriately scoped) Instance Profile
- Revoke existing temporary (STS) credentials by applying an appropriate revocation policy to the Instance’s associated Role

Note: Ensure you update the *aws:TokenIssueTime* value within the policy to an appropriate time based on the situation and incident

Disk Analysis

# Docker Forensics Toolkit (Initial Setup)

- Instrument a dedicated forensic analysis Instance
- Create new Volume(s) from previously acquired Volume Snapshot(s)
- Attach new Volume(s) to the analysis Instance
- Mount Volume(s) READ-ONLY

```
$ sudo mount -o ro,nouuid,norecovery,offset=<offset> /dev/<device> /mnt/point
```

- Instrument Docker Forensics Toolkit

```
$ git clone https://github.com/docker-forensics-toolkit/toolkit.git
$ pyinstaller dof.spec
```

Disk Analysis

# Docker Forensics Toolkit (Analysis)

Get Docker environment information

```
$ sudo dof status /mnt/point/
```

Identify Containers/Pods on system

```
$ sudo dof list-containers /mnt/point/
```

List all images running on system

```
$ sudo dof list-images /mnt/point/
```

Note: Images that don't belong to a Repository were not pulled from Docker Hub or a private Registry, but likely built on this system. Images without a corresponding container instance may indicate a deleted container.

Disk Analysis

# Docker Forensics Toolkit (Analysis)

Identify specific Container/Pod information

```
$ sudo dof list-containers /mnt/point/ | grep <podname>
```

Show image build history

```
$ sudo dof show-image-history --image <image> /mnt/point/
```

Note: Identify any possibly malicious commands involved in the image build

Show all logs for a given Container/Pod

```
$ dof show-container-log --container <container-name> /mnt/point/
```

Disk Analysis

# Docker Forensics Toolkit (Analysis)

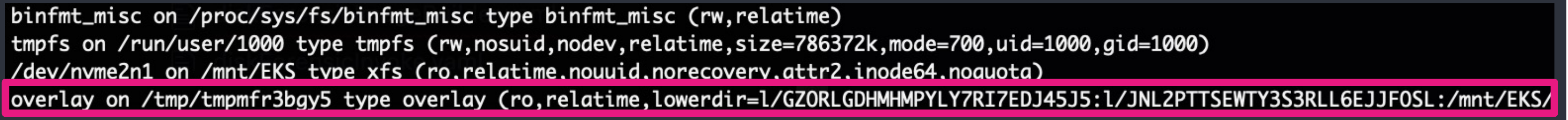
Mount Container/Pod filesystem for analysis

```
$ dof mount-container --container <container-name> /mnt/point/
```

Note: You may receive an “Failed to execute script 'main' due to unhandled exception!” error, even though the filesystem has successfully mounted.

Verify successful mount

```
$ mount
```



```
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,relatime)
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,size=786372k,mode=700,uid=1000,gid=1000)
/dev/nvme2n1 on /mnt/EKS type xfs (ro,relatime,nouuid,norecovery,attr2,inode64,noquota)
overlay on /tmp/tmpmfr3bgy5 type overlay (ro,relatime,lowerdir=l/GZORLGDHMHMPYLY7RI7EDJ45J5:l/JNL2PTTSEWTY3S3RLL6EJJFOSL:/mnt/EKS/
```

Disk Analysis

```
# Docker Forensics Toolkit (Analysis)

Examine Container/Pod filesystem
$ sudo ls -la </tmp/mountpoint>
$ sudo log2timeline ...
... standard linux filesystem analysis ...

Dismount Container/Pod filesystem
$ sudo umount </tmp/mountpoint>
```

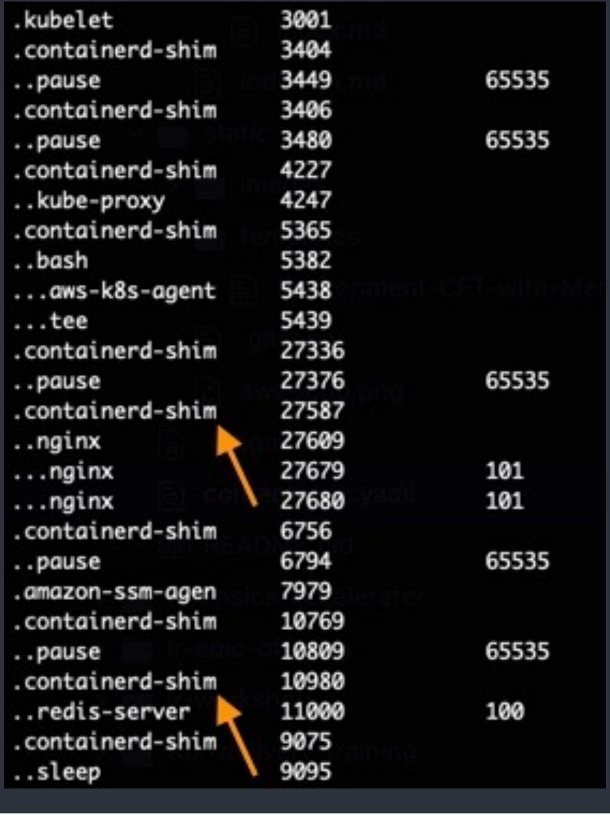
Memory Analysis

# Volatility

Acquire process tree listing of running processes on the Node

```
$ python ./volatility/vol.py -f <data.lime> --profile=<Vol-Profile> linux_pstree
```

Note: Each container on EKS (running containerd) will have a parent process name of containerd-shim *as seen below*



```
.kubelet             3001
.containerd-shim     3404
..pause              3449              65535
.containerd-shim     3406
..pause              3480              65535
.containerd-shim     4227
..kube-proxy         4247
.containerd-shim     5365
..bash               5382
...aws-k8s-agent     5438
...tee               5439
.containerd-shim     27336
..pause              27376             65535
.containerd-shim     27587
..nginx              27609
...nginx             27679             101
...nginx             27680             101
.containerd-shim     6756
..pause              6794              65535
.amazon-ssm-agen     7979
.containerd-shim     10769
..pause              10809             65535
.containerd-shim     10980
..redis-server       11000             100
.containerd-shim     9075
..sleep              9095
```

00101 01011 10011 00100 00110 01001 10010

Memory Analysis

# Volatility

Examine a specific Container/Pod

- Identify the Container/Pod by name
- Identify PID(s) associated with the Container/Pod
- Examine specific PID(s) memory space for linked/referenced/open files, etc.

```
$ python ./volatility/vol.py -f <data.lime> --profile=<Vol-Profile>
linux_proc_maps -p <PID>
```

- Dump specific memory space from the Pod (by PID) for analysis

```
$ python ./volatility/vol.py -f <data.lime> --profile=<Vol-Profile>
linux_dump_map -p <PID> -s <0xMEM> -D . --output-file=<name>
```

- ... Whatever other memory analysis is needed

Log Analysis

# Control Plane Logs (CloudTrail)

- Some Athena (SQL) samples to get you started...

Identify top EKS events/actions

```sql
SELECT region_partition, eventname, count(*) as eventcount FROM cloudtrail
WHERE eventsource = 'eks.amazonaws.com'
AND date_partition >= '2021/07/01'
AND date_partition <= '2021/07/31'
AND account_partition = '111122223333'
AND region_partition in ('us-east-1','us-east-2','us-west-2', 'us-west-2')
GROUP BY region_partition, eventname
ORDER BY region_partition, eventcount DESC
```

0 0 1 0 1   0 1 0 1 1   1 0 0 1 1   0 0 1 0 0   0 0 1 1 0   0 1 0 0 1   1 0 0 1 0

Log Analysis

# Control Plane Logs (CloudTrail)

Identify all Create* EKS events/actions

```
SELECT region_partition, eventname, count(*) as eventcount FROM
cloudtrail
WHERE eventsource = 'eks.amazonaws.com'
AND eventname LIKE 'Create%'
AND date_partition >= '2021/07/01'
AND date_partition <= '2021/07/31'
AND account_partition = '111122223333'
AND region_partition in ('us-east-1','us-east-2','us-west-2', 'us-
west-2')
GROUP BY region_partition, eventname
ORDER BY region_partition, eventcount DESC
```

Log Analysis

# Control Plane Logs (CloudTrail)

Identify all Delete* EKS events/actions

```sql
SELECT region_partition, eventname, count(*) as eventcount FROM cloudtrail
WHERE eventsource = 'eks.amazonaws.com'
AND eventname LIKE 'Delete%'
AND date_partition >= '2021/07/01'
AND date_partition <= '2021/07/31'
AND account_partition = '111122223333'
AND region_partition in ('us-east-1','us-east-2','us-west-2', 'us-west-2')
GROUP BY region_partition, eventname
ORDER BY region_partition, eventcount DESC
```

0 0 1 0 1   0 1 0 1 1   1 0 0 1 1   0 0 1 0 0   0 0 1 1 0   0 1 0 0 1   1 0 0 1 0

Log Analysis

# Audit Logs

- Leveraging CloudWatch Logs Insights ...

Identify all actions associated with a Node (Instance)

```
fields @timestamp, @message
| filter @message like "<nodename>" or @message like "<PrivateIP>"
| filter @timestamp >= toMillis("YYYY-MM-DDT12:34:56.123-07:00")
| filter @timestamp <= toMillis("YYYY-MM-DDT12:34:56.123-07:00")
| sort @timestamp asc
```

Note: Adjust timestamp filter to the appropriate time range within the console or within the search query

Log Analysis

# Audit Logs

Identify all API Audit logs with “create” events for the Node (Instance)

```
fields @timestamp, @message
| filter verb == "create"
| filter @message like "<PrivateIP>" or @message like "<nodename>"
| sort @timestamp asc
```

Log Analysis

# Audit Logs

Identify who created a Node and when, along with Instance metadata

```
fields @timestamp, requestReceivedTimestamp, objectRef.name,
objectRef.resource, verb, stage, responseObject.kind,
responseStatus.code, user.extra.accessKeyId.0, user.extra.arn.0,
user.username, sourceIPs.0, userAgent
| filter verb == "create"
| filter @message like "<PrivateIP>"
| sort requestReceivedTimestamp asc
```

Log Analysis

# Audit Logs

Identify when Node (Instance) infrastructure was created/launched

```
fields @timestamp, @message
| filter @logStream like /cloud-controller-manager/
| filter @message like "<nodename>"
| filter @message like "Added" or @message like "process"
| sort @timestampe asc
```

Log Analysis

# Audit Logs

Identify all scheduling activity on a Node (Instance)

```
fields @timestamp, @message
| filter @logStream like /kube-scheduler/
| filter @message like "<nodename>"
| sort @timestamp asc
```

Log Analysis

# Audit Logs

Identify all actions associated with a Container/Pod

```
fields @timestamp, @message
| filter objectRef.name == "<pod-name>"
| filter @timestamp >= toMillis("YYYY-MM-DDT12:34:56.123-07:00")
| filter @timestamp <= toMillis("YYYY-MM-DDT12:34:56.123-07:00")
| sort @timestamp asc
```

Note: Adjust timestamp filter to the appropriate time range within the console or within the search query

Log Analysis

# Audit Logs

Identify who created a Container/Pod and when

```
fields @timestamp, requestReceivedTimestamp, objectRef.name,
objectRef.namespace, objectRef.resource, verb, stage,
responseObject.kind, responseObject.status.phase, responseStatus.code,
responseObject.status, responseObject.reason, responseObject.message,
user.extra.accessKeyId.0, user.extra.arn.0, user.username,
sourceIPs.0, userAgent
| filter objectRef.name == "<pod-name>"
| filter verb == "create"
| filter responseObject.kind in ["Pod","Status"]
| sort requestReceivedTimestamp asc
```

Log Analysis

# Audit Logs

Identify the Node (Instance) where the Container/Pod was scheduled (run)

```
fields @timestamp, @message
| filter @logStream like /kube-scheduler/
| filter @message like "<pod-name>"
| parse @message 'pod="*" node="*"' as pod, node
```

Log Analysis

# Audit Logs

Identify the Instance ID of the Node

```
fields @timestamp, @message
| filter @logStream like /cloud-controller-manager/
| filter @message like "<nodename>" and @message like "Instance ID"
| parse @message '] is *' instance_id
```

Log Analysis

# Audit Logs

Identify commands executed against/on the Container/Pod through kubectl

```
fields @timestamp, requestReceivedTimestamp, objectRef.name,
objectRef.namespace, objectRef.resource, verb, stage,
responseStatus.code, user.extra.accessKeyId.0, user.extra.arn.0,
user.username, sourceIPs.0, userAgent, requestURI
| filter objectRef.name == "<pod-name>"
| filter requestURI like /exec\?command=/
| parse @message /(exec\?command=?)(?<command>([a-zA-Z0-9-_.]+))/
| sort requestReceivedTimestamp asc
```

Conclusion

04

0 0 1 0 1 0 1 0 1 1 1 0 0 1 1 0 0 1 0 0 0 0 1 1 0 0 1 0 0 1 1 0 0 1 0

Conclusion

# There’s a lot more to EKS Incident Response and Forensic Analysis than high-level “isolate the Node” (in what order/manner?), “determine who created the Pod” (ok, but how?), ...

# Understanding the container filesystem and memory structure is key to effective and comprehensive investigation

# There are a variety of tools/mechanisms to effectively search EKS data and artifacts (this presentation is just a sampling)

# Live response is an option, but data collection for offline analysis is better practice and relatively easy leveraging cloud native mechanisms

# I recommend acquiring data from the entire Node/Instance versus a singular Container/Pod for thoroughness and the ability to perform more comprehensive investigation (what if more than a singular Container/Pod is compromised?)

# Understanding the control plane logs and their contents/value is key to effectively searching and identifying artifacts/evidence

# Thanks!

Twitter: @JPoForenso

Website: https://www.ponderthebits.com