Kubernetes Security Lab — Week 2

Runtime Detection using Custom Python Scripts

Michael O'Malley

1 Abstract

This lab focused on implementing container runtime monitoring in a lightweight Kubernetes cluster on a Raspberry Pi using custom Python scripts. The environment simulated malicious activity such as shell spawning, unauthorized process execution, and suspicious file creation. Python-based monitoring agents were deployed inside pods to detect these activities in real time. The lab highlights practical approaches to runtime detection in constrained environments where kernel-level tools are unavailable.

2 Lab Objective and Threat Model

2.1 Objective

- Deploy runtime monitoring inside Kubernetes pods
- Detect suspicious container activity using Python scripts
- Simulate attack behaviors including shell spawn and file tampering
- Demonstrate custom detection rules without root privileges

2.2 Threat Model

- Malicious process execution within a container
- Unauthorized file creation or modification
- Pod-level privilege escalation attempts
- Constrained host environment without kernel-level monitoring

3 Environment Setup

3.1 Cluster Configuration

- Kubernetes Distribution: K3s
- Node Role: Single-node control plane on Raspberry Pi
- Container Runtime: containerd
- Networking: Flannel CNI
- Host Platform: ARM-based lab environment

3.2 Monitoring Agent Setup

- Pod: ubuntu-baseline deployed in namespace `rt-lab`
- Python 3 installed with `psutil` library

- Monitoring script `monitor.py` for process and file activity
- No root privileges required for runtime detection

4 Runtime Monitoring Implementation

4.1 Process Monitoring

The Python script continuously enumerates active processes within the pod. Newly spawned processes are detected in real time and logged to stdout with process ID and name.

```
import psutil

known = set(p.info['pid'] for p in psutil.process_iter(['pid', 'name'])
   )

while True:
    current = set(p.info['pid'] for p in psutil.process_iter(['pid', '
       name']))
    new = current - known
    if new:
        for p in psutil.process_iter(['pid', 'name']):
            if p.info['pid'] in new:
                print(f"[ALERT] New process detected: PID={p.info['pid
                   ']} NAME={p.info['name']}")
    known = current
```

4.2 File Monitoring

The script also tracks a target directory (`/tmp`) for new file creation. Alerts are triggered whenever a new file appears.

```
import os

watched_dir = "/tmp"
known_files = set(os.listdir(watched_dir))

while True:
    current_files = set(os.listdir(watched_dir))
    new_files = current_files - known_files
    if new_files:
        for f in new_files:
            print(f"[ALERT] New file detected: {f}")
    known_files = current_files
```

4.3 Alerting and Logging

Alerts are printed in real time to the container's stdout. Additional logging can be configured to write to files or external monitoring systems such as ELK, Loki/Grafana, or Slack webhooks.

5 Simulated Attacks and Detection

5.1 Shell Spawn

A new shell started inside the pod is detected immediately:

```
kubectl exec -it ubuntu-baseline -n rt-lab -- bash
```

5.2 File Creation

Suspicious file writes to the monitored directory trigger alerts:

```
touch /tmp/testfile
echo "malicious" > /tmp/malicious.txt
```

5.3 Unauthorized Process Execution

Any additional commands executed in the pod are detected and logged by the monitoring agent.

6 Security Assessment Tools

- Python 3 with `psutil` for process enumeration
- Standard Linux utilities (`ls`, `ps`) for verification
- Kubernetes `kubectl` for pod management and test attack simulation

7 Impact Analysis

- Real-time detection of unauthorized process execution
- Alerts on suspicious file creation
- Visibility into pod-level activity without kernel access
- Demonstrates runtime threat detection in constrained environments

8 Key Takeaways

- Python-based monitoring is effective for runtime detection in pods
- Real-time alerts help detect shell spawns, file tampering, and process anomalies
- Custom detection scripts can replace kernel-level tools on constrained hosts
- Runtime security monitoring is feasible on lightweight Kubernetes clusters such as K3s on Raspberry Pi

9 Conclusion

This lab demonstrated the implementation of runtime detection using custom Python scripts in Kubernetes pods. By monitoring process creation and file activity, the environment provided visibility into potentially malicious behavior in real time. The project illustrates a practical approach to runtime security in constrained environments where traditional kernel-level solutions like Falco are unavailable.

10 References

- Python Official Documentation — https://www.python.org/doc/
- psutil Library Documentation — https://psutil.readthedocs.io/
- Kubernetes Official Documentation — https://kubernetes.io/docs/
- Rancher K3s Documentation — https://rancher.com/docs/k3s/latest/en/