Kubernetes Pentest All-in-one: The Ultimate Toolkit

Sergey Kanibor

Luntry

@r0binak

whoami

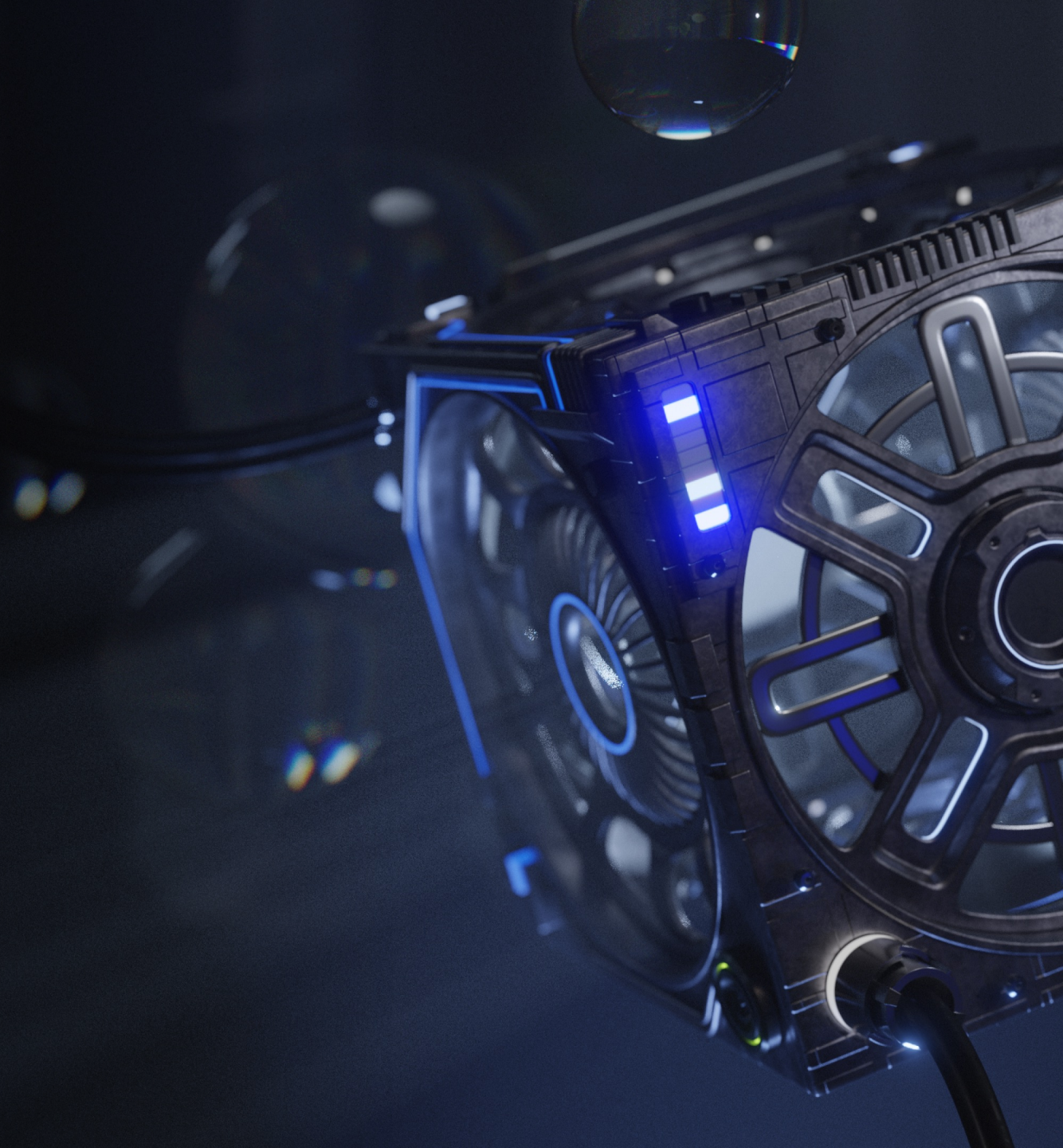
- R&D, Container Security at Luntry
- Focused on containers & Kubernetes security
- Talks at PHDays, HackConf, CyberCamp, VolgaCTF, БЕКОН
- Editor Telegram channel @k8secuirty

Agenda

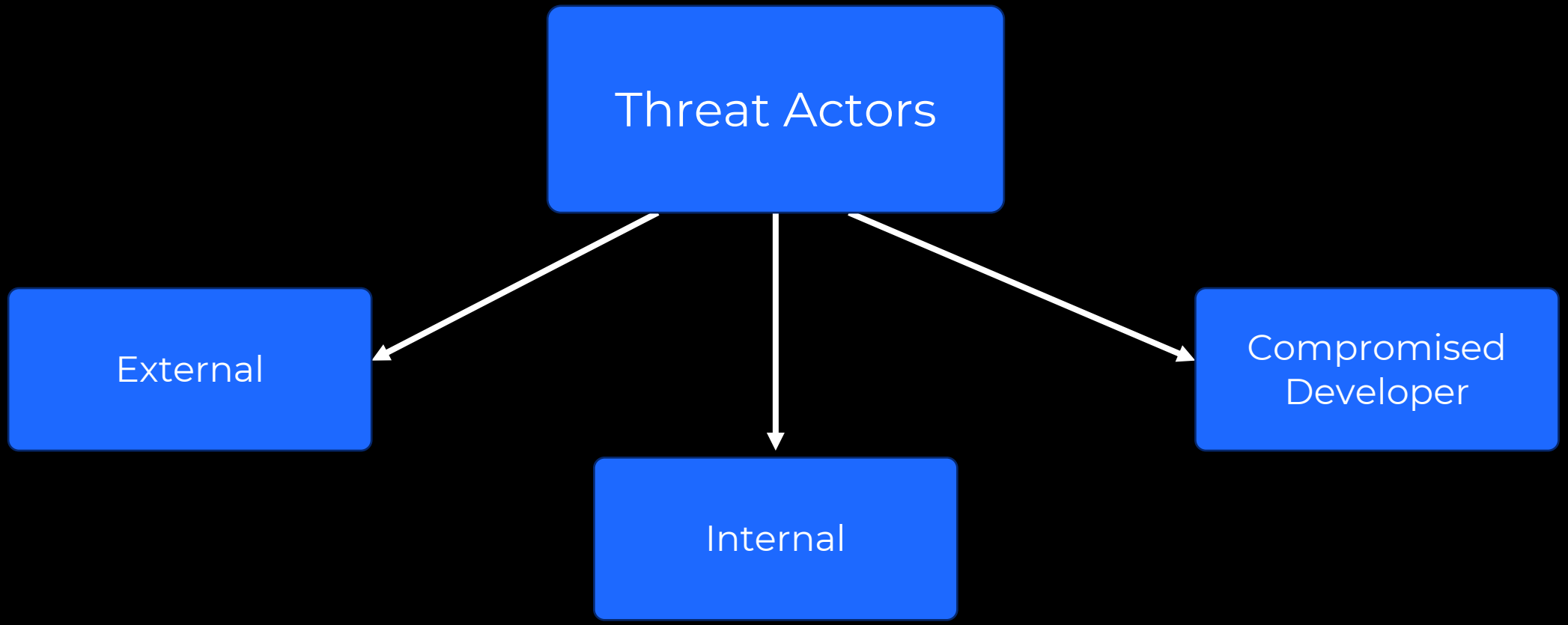
- Typical Kubernetes Pentest
- Target Environment
- MTKPI
- Conclusions

NO FF ONE 2023

Typical Kubernetes Pentest



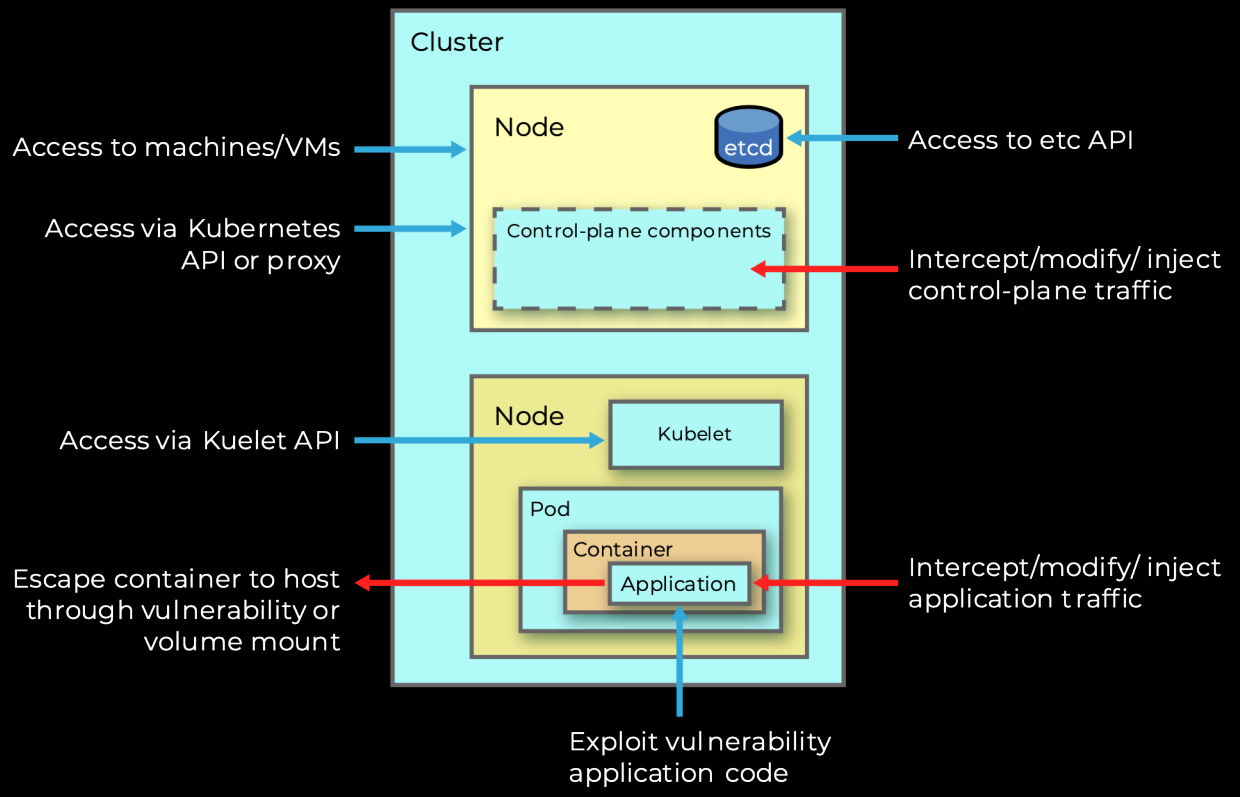
Threat actors



External

See the Kubernetes network outside the cluster

- Vulnerable K8S components

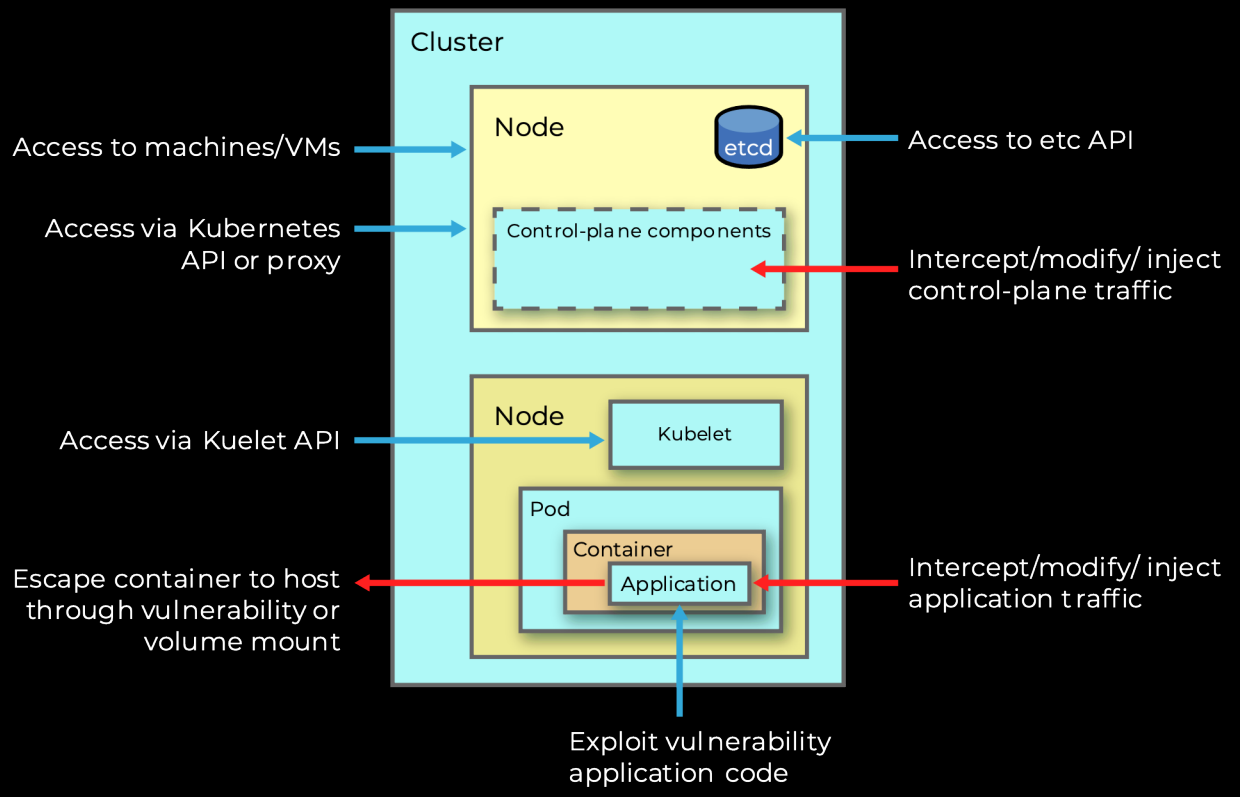


External actor ways

External

See the Kubernetes network outside the cluster

- Vulnerable K8S components
- Vulnerable applications

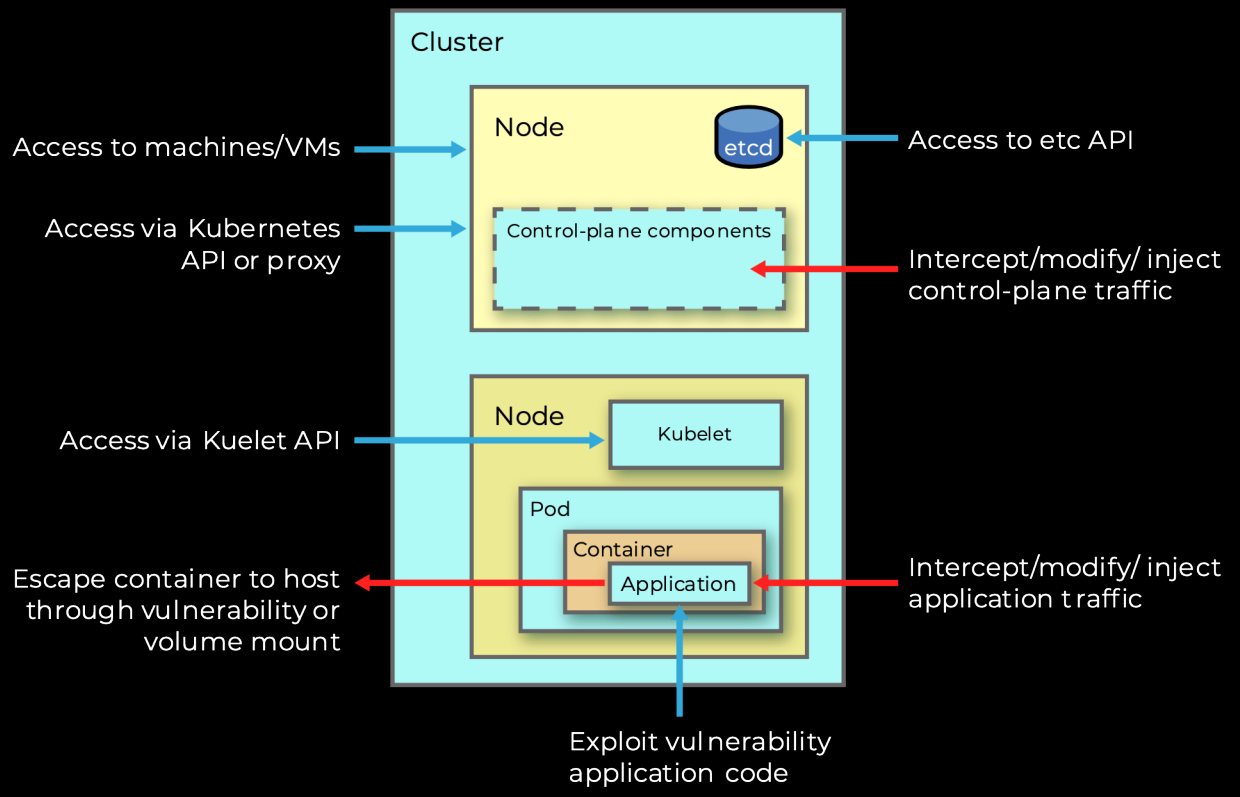


External actor ways

External

See the Kubernetes network outside the cluster

- Vulnerable K8S components
- Vulnerable applications
- Misconfigured Kubernetes components (API Server, Kubelet, etcd)

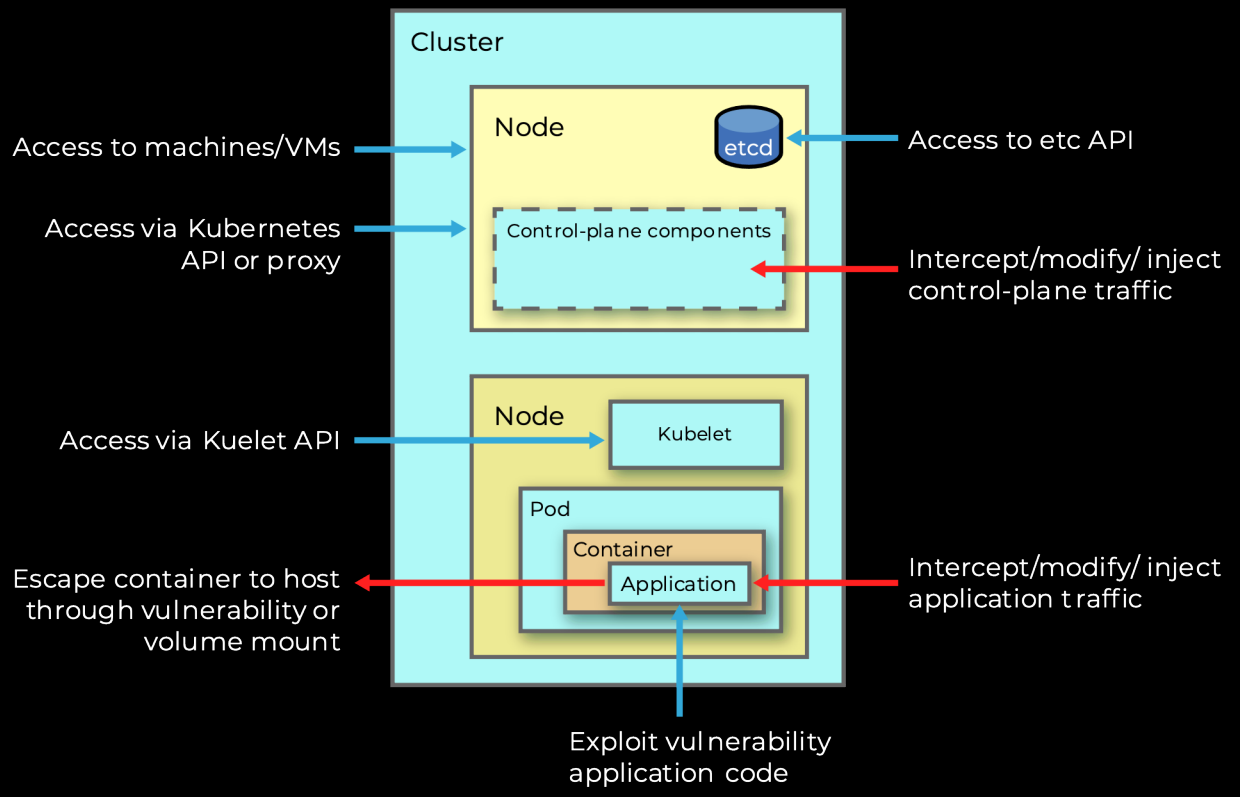


External actor ways

External

See the Kubernetes network outside the cluster

- Vulnerable K8S components
- Vulnerable applications
- Misconfigured Kubernetes components (API Server, Kubelet, etcd)
- Misconfigured Docker/Containerd/CRIO daemon

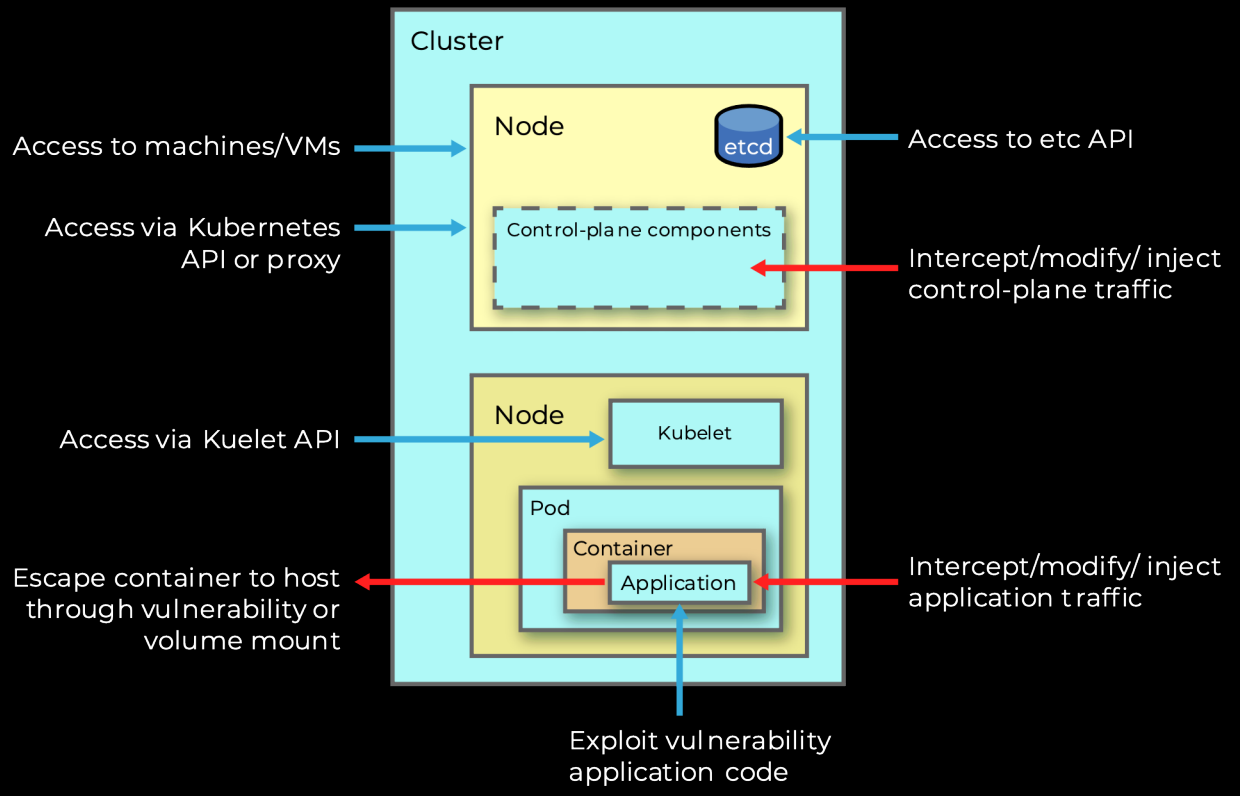


External actor ways

External

See the Kubernetes network outside the cluster

- Vulnerable K8S components
- Vulnerable applications
- Misconfigured Kubernetes components (API Server, Kubelet, etcd)
- Misconfigured Docker/Containerd/CRIO daemon
- Misconfigured dashboards (Weave Scope, Kubernetes Dashboard, Octant)

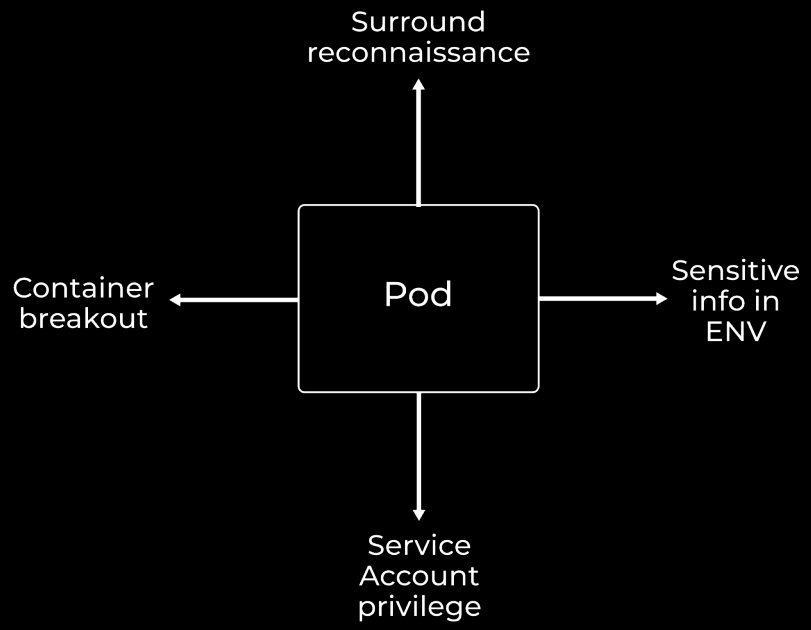


External actor ways

Internal – in Pod

Inside the Pod

- Bad Pods

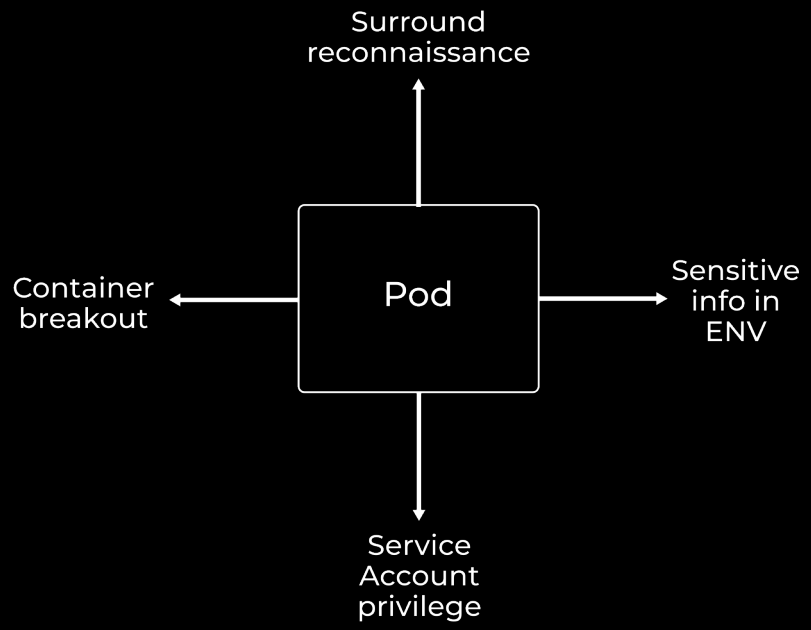


Internal actor ways

Internal – in Pod

Inside the Pod

- Bad Pods
- Private keys, tokens & creds in ENV

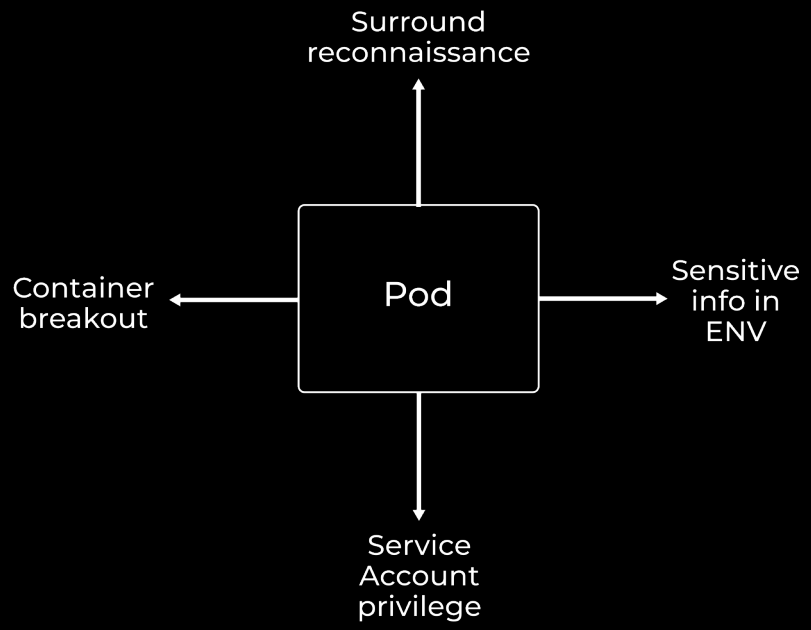


Internal actor ways

Internal – in Pod

Inside the Pod

- Bad Pods
- Private keys, tokens & creds in ENV
- Other services, cluster components

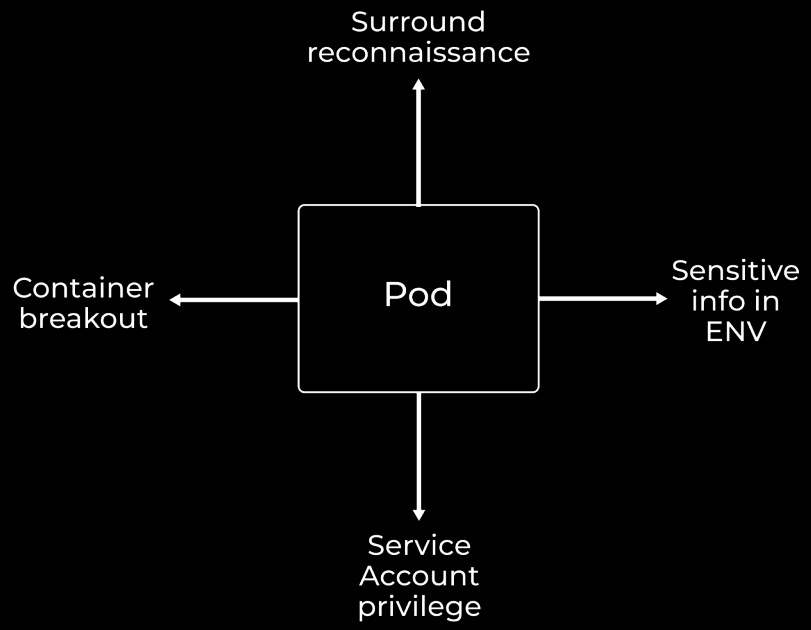


Internal actor ways

Internal – in Pod

Inside the Pod

- Bad Pods
- Private keys, tokens & creds in ENV
- Other services, cluster components
- Service account permissions

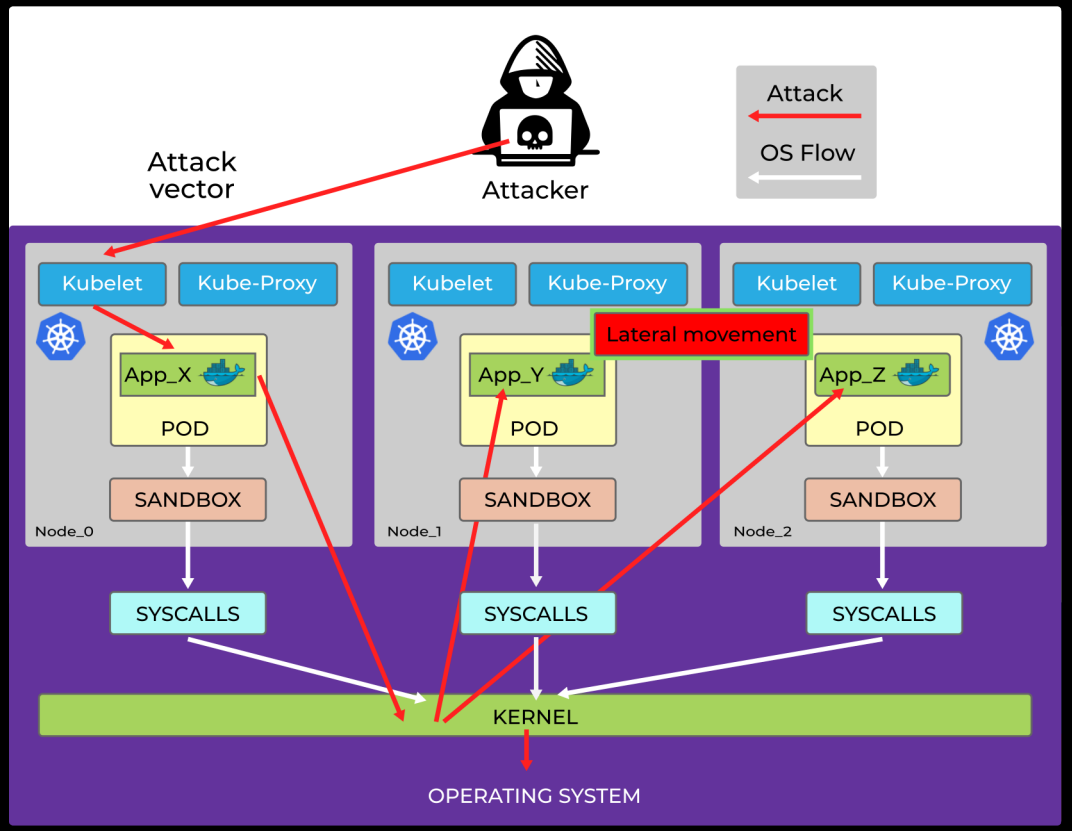


Internal actor ways

Internal – on Node

Inside the Node

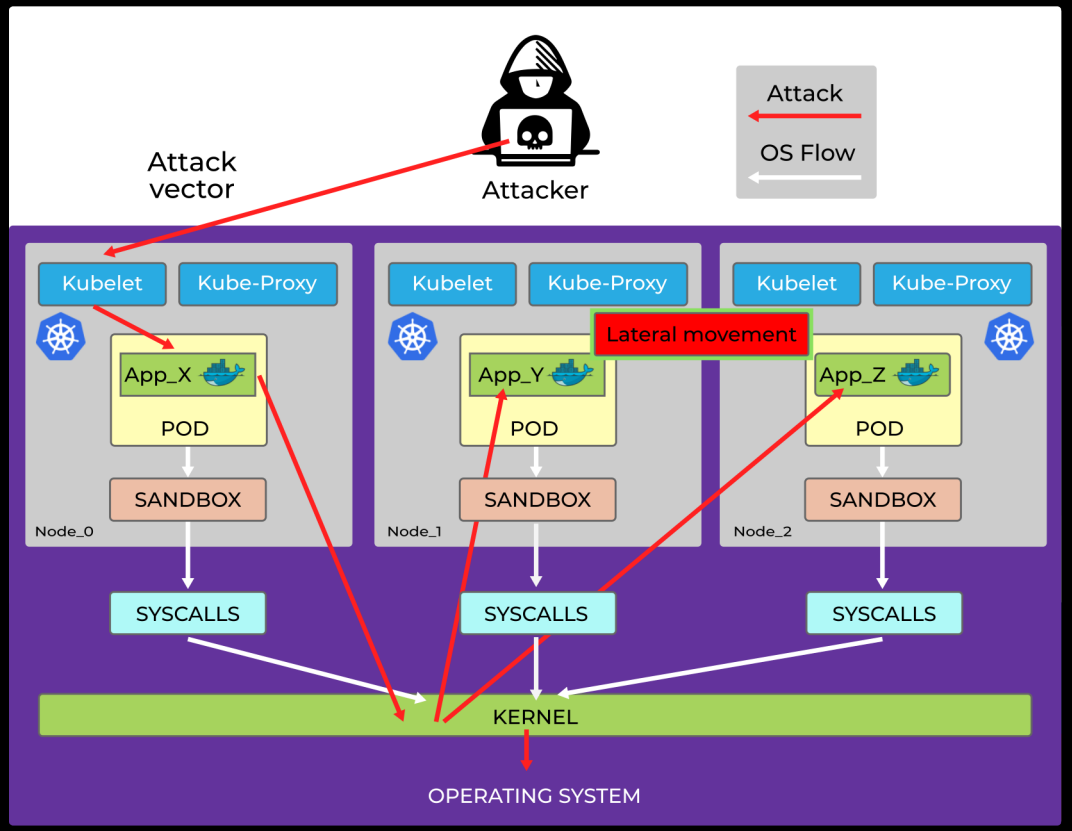
- Keys, SA tokens, configs, certs
  - Kubernetes Privilege Escalation: Container Escape == Cluster Admin? (Yuval Avrahami & Shaul Ben Hai, Palo Alto Networks. BlackHat USA 2022)



Internal – on Node

Inside the Node

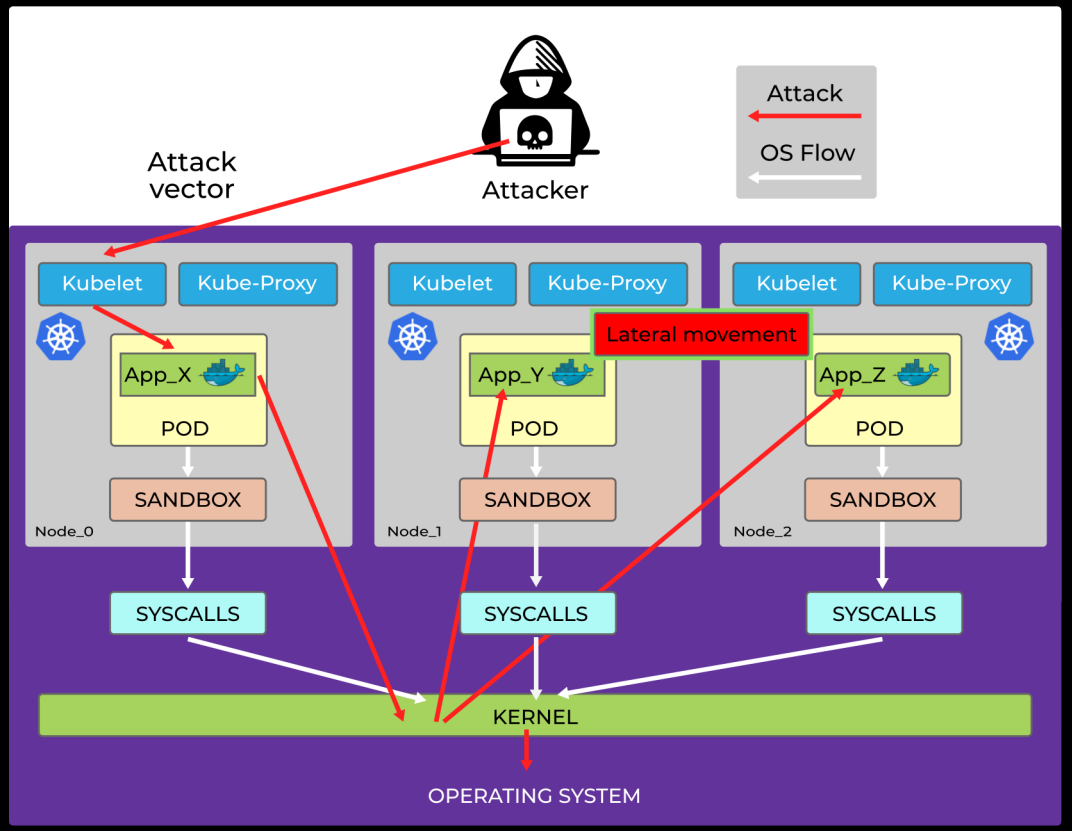
- Keys, SA tokens, configs, certs
  - Kubernetes Privilege Escalation: Container Escape == Cluster Admin? (Yuval Avrahami & Shaul Ben Hai, Palo Alto Networks. BlackHat USA 2022)
- Old dumps, sensitive logs



Internal – on Node

Inside the Node

- Keys, SA tokens, configs, certs
  - Kubernetes Privilege Escalation: Container Escape == Cluster Admin? (Yuval Avrahami & Shaul Ben Hai, Palo Alto Networks. BlackHat USA 2022)
- Old dumps, sensitive logs
- Third-party instances



Compromised Developer

Compromised Developer

Compromised Developer

- Can push its own images in registry

Compromised Developer

Compromised Developer

- Can push its own images in registry
- There is access to internal services

Compromised Developer

Pentesting Kubernetes: From Zero to Hero

Link

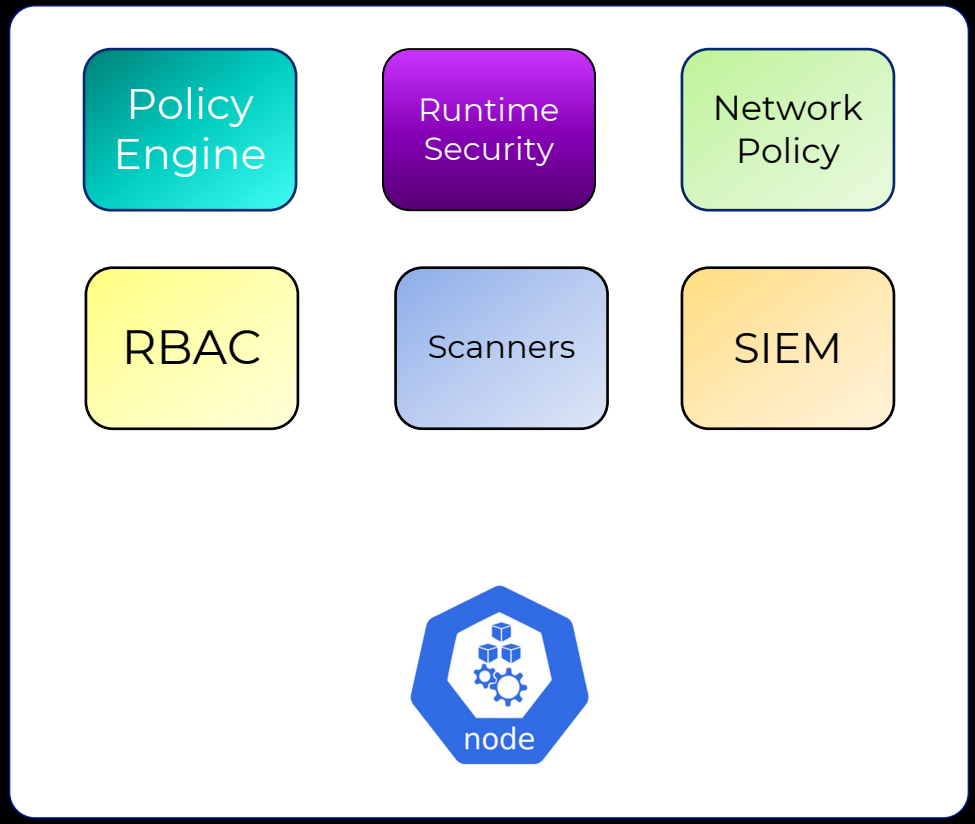
Get more info about Kubernetes pentest for newbies:

ON FF ONE 2023

Target Environment

Environment

- SOC, DevSecOps, AppSec
- Policy Engine
- Runtime Security
- Network Policy
- RBAC



NOFF ONE 2023

MTKPI

MTKPI – Multi Tool Kubernetes Pentest Image

Inspired by:

- botty
- Hacker container
- netshoot
- alpine-containertools

Whatever you need

- Reconnaissance
  - kdigger
  - botb
  - deepce

Whatever you need

- Reconnaissance
  - kdigger
  - botb
  - deepce
- Privilege Escalation
  - Traitor
  - linuxprivchecker

Whatever you need

- Reconnaissance
  - kdigger
  - botb
  - deepce
- Privilege Escalation
  - Traitor
  - linuxprivchecker
- Network
  - curl
  - dig
  - nc

Whatever you need

- Reconnaissance
  - kdigger
  - botb
  - deepce
- Privilege Escalation
  - Traitor
  - linuxprivchecker
- Network
  - curl
  - dig
  - nc
- Combine tools
  - kubeletctl
  - kubesploit C2 agent
  - CDK
  - peirates
  - ctrsploit
  - kube-hunter
  - kubectl

Microsoft – Threat Matrix for Kubernetes

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Impact
Using cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access Kubernetes API server	Access cloud resources	Images from a private registry	Data destruction
Compromised image In registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account	Collecting data from pod	Resource hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Container service account	Network mapping	Cluster internal networking		Denial of service
Application vulnerability	Application exploit (RCE)	Malicious admission controller	Access cloud resources	Connect from proxy server	Application credentials in configuration files	Exposed sensitive interfaces	Application credentials in configuration files		
Exposed sensitive interfaces	SSH server running inside container	Container service account			Access managed identity credentials	Instance Metadata API	Writable hostPath mount		
	Sidecar injection	Static pods			Malicious admission controller		CoreDNS poisoning		
							ARP poisoning and IP spoofing		

Microsoft – Threat Matrix for Kubernetes

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Impact
Using cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access Kubernetes API server	Access cloud resources	Images from a private registry	Data destruction
Compromised image In registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account	Collecting data from pod	Resource hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Container service account	Network mapping	Cluster internal networking		Denial of service
Application vulnerability	Application exploit (RCE)	Malicious admission controller	Access cloud resources	Connect from proxy server	Application credentials in configuration files	Exposed sensitive interfaces	Application credentials in configuration files		
Exposed sensitive interfaces	SSH server running inside container	Container service account			Access managed identity credentials	Instance Metadata API	Writable hostPath mount		
	Sidecar injection	Static pods			Malicious admission controller		CoreDNS poisoning		
							ARP poisoning and IP spoofing		

Just Dockerfile

```
FROM tsl0922/ttyd:latest
LABEL maintainer="r0binak"

#Base image with web shell

EXPOSE 7681

WORKDIR /var/run

RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -y \
        curl \
        iputils-ping \
        nano \
        python3-pip \
        dnsutils \
        apt-file \
        net-tools \
        nmap \
        stow \
        git-core \
        sudo \
        util-linux\
        p7zip-full \
        jq \
        ssh \
        python \
        python3 \
        upx \
        && rm -rf /var/lib/apt/lists/*

# Install botb
RUN curl -LO https://github.com/brompwnie/botb/releases/latest/download/botb-linux-amd64 \
    && install botb-linux-amd64 /usr/local/bin/botb \
    && rm -rf botb-linux-amd64

# Install traitor
RUN curl -LO https://github.com/liamg/traitor/releases/latest/download/traitor-amd64 \
    && install traitor-amd64 /usr/local/bin/traitor \
    && rm -rf traitor-amd64

# Install kubeletctl
RUN curl -LO https://github.com/cyberark/kubeletctl/releases/latest/download/kubeletctl_linux_amd64 \
    && install kubeletctl_linux_amd64 /usr/local/bin/kubeletctl \
    && rm -rf kubeletctl_linux_amd64

# Install kubesploit C2 agent
RUN curl -LO https://github.com/cyberark/kubesploit/releases/latest/download/kubesploitAgent-Linux-x64.7z \
    && 7z x kubesploitAgent-Linux-x64.7z -r kubesploitAgent-Linux-x64 -pkubesploit \
    && install kubesploitAgent-Linux-x64 /usr/local/bin/kubesploit \
```

Just Dockerfile

```
FROM tsl0922/ttyd:latest
LABEL maintainer="r0binak"

#Base image with web shell

EXPOSE 7681

WORKDIR /var/run

RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -y \
        curl \
        iputils-ping \
        nano \
        python3-pip \
        dnsutils \
        apt-file \
        net-tools \
        nmap \
        stow \
        git-core \
        sudo \
        util-linux\
        p7zip-full \
        jq \
        ssh \
        python \
        python3 \
        upx \
        && rm -rf /var/lib/apt/lists/*

# Install botb
RUN curl -LO https://github.com/brompwnie/botb/releases/latest/download/botb-linux-amd64 \
    && install botb-linux-amd64 /usr/local/bin/botb \
    && rm -rf botb-linux-amd64

# Install traitor
RUN curl -LO https://github.com/liamg/traitor/releases/latest/download/traitor-amd64 \
    && install traitor-amd64 /usr/local/bin/traitor \
    && rm -rf traitor-amd64

# Install kubeletctl
RUN curl -LO https://github.com/cyberark/kubeletctl/releases/latest/download/kubeletctl_linux_amd64 \
    && install kubeletctl_linux_amd64 /usr/local/bin/kubeletctl \
    && rm -rf kubeletctl_linux_amd64

# Install kubesploit C2 agent
RUN curl -LO https://github.com/cyberark/kubesploit/releases/latest/download/kubesploitAgent-Linux-x64.7z \
    && 7z x kubesploitAgent-Linux-x64.7z -r kubesploitAgent-Linux-x64 -pkubesploit \
    && install kubesploitAgent-Linux-x64 /usr/local/bin/kubesploit \
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mtkpi-pod
  labels:
    app: mtkpi
spec:
  containers:
  - name: mtkpi-pod
    image: r0binak/mtkpi:v1
    ports:
    - containerPort: 7681
  securityContext:
      capabilities:
        drop:
          - all
      readOnlyRootFilesystem: true
```

ON FF ONE 2023

MTKPI in action

Bypassing Vulnerability Scanners

- There are a number of files and directories that contain information used by vulnerability scanners and SBOM tools

Bypassing Vulnerability Scanners

- There are a number of files and directories that contain information used by vulnerability scanners and SBOM tools
- No feeds == no vulns ☺

Bypassing Vulnerability Scanners – techniques

- Modified /etc/os/release
- Deleted APK metadata
- Symlinked Language Dependency Files
- UPX packed binaries
- Multi-stage build with all techniques

Talks, Slides, Repo, Blog

Get shell in Pod

- We don't have pods/exec rights

Get shell in Pod

- We don't have pods/exec rights

- We need access to the shell to execute the command

Get shell in Pod

- We don't have pods/exec rights

- We need access to the shell to execute the command

- We can proxy the traffic to ourselves on localhost

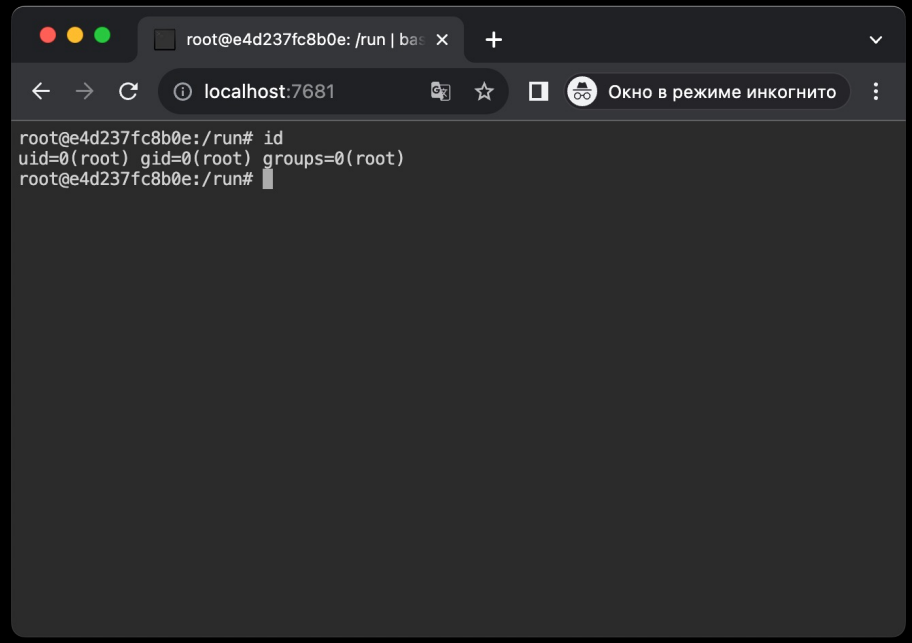
Get shell in Pod

- We don't have pods/exec rights

- We need access to the shell to execute the command
- We can proxy the traffic to ourselves on localhost
- Share the terminal over the web

Get shell in Pod

- We don't have pods/exec rights
- We need access to the shell to execute the command
- We can proxy the traffic to ourselves on localhost
- Share the terminal over the web



kubectl port-forward <pod-name> 28015:27017

LoTL attack

- Attacks in which the attacker uses legitimate utilities to perform malicious actions

LoTL attack

- Attacks in which the attacker uses legitimate utilities to perform malicious actions
- There are usually quite a few of these utilities in the container

LoTL attack

- Attacks in which the attacker uses legitimate utilities to perform malicious actions
- There are usually quite a few of these utilities in the container
- GTFObins

LoTL attack

- Attacks in which the attacker uses legitimate utilities to perform malicious actions
- There are usually quite a few of these utilities in the container
- GTFObins
- Use traitor to automate this

Getting root back in non-root envs

```
FROM ubuntu:22.04
RUN cp /bin/bash /bin/setuidbash && chmod 4755 /bin/setuidbash
RUN adduser tester
USER tester
CMD ["/bin/bash"]
```

RWX in readOnly

- Use ddexec.sh for fileless attack

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: alpine-ro
  namespace: default
spec:
  containers:
  - name: alpine
    image: alpine
    command: ["/bin/sh"]
    args: ["-c", "sleep 100000"]
    securityContext:
      readOnlyRootFilesystem: True
```

RWX in readOnly

- Use ddexec.sh for fileless attack
- /dev/shm for RW only

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: alpine-ro
  namespace: default
spec:
  containers:
  - name: alpine
    image: alpine
    command: ["/bin/sh"]
    args: ["-c", "sleep 100000"]
    securityContext:
      readOnlyRootFilesystem: True
```

RWX in readOnly

- Use ddexec.sh for fileless attack
- /dev/shm for RW only
- You have RWX permissions at /etc/hosts, /dev/termination-log

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: alpine-ro
  namespace: default
spec:
  containers:
  - name: alpine
    image: alpine
    command: ["/bin/sh"]
    args: ["-c", "sleep 100000"]
    securityContext:
      readOnlyRootFilesystem: True
```

RWX in readOnly

- Use ddexec.sh for fileless attack
- /dev/shm for RW only
- You have RWX permissions at /etc/hosts, /dev/termination-log
- Executing Arbitrary Code & Executables in Read-Only FlieSystems

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: alpine-ro
  namespace: default
spec:
  containers:
  - name: alpine
    image: alpine
    command: ["/bin/sh"]
    args: ["-c", "sleep 100000"]
    securityContext:
      readOnlyRootFilesystem: True
```

hostNetwork gotcha

- Kubernetes usually runs your pods in their own isolated network

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hostnetwork-exec-pod
  labels:
    app: pentest
spec:
  hostNetwork: true
  containers:
  - name: hostnetwork-pod
    image: ubuntu
    command: [ "/bin/sh", "-c", "--" ]
    args: [ "while true; do sleep 30; done;" ]
```

hostNetwork gotcha

- Kubernetes usually runs your pods in their own isolated network
- hostNetwork: true

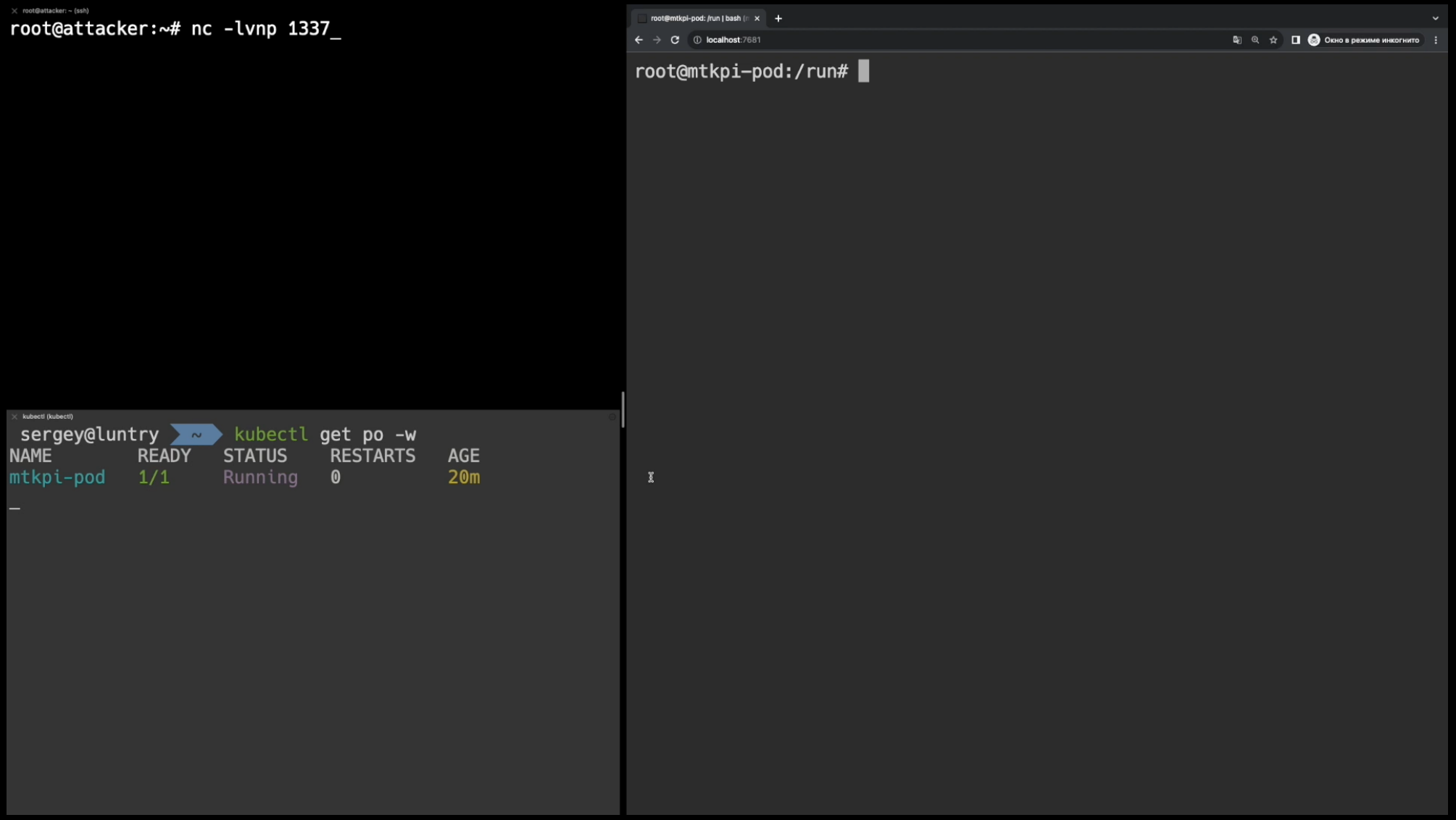
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hostnetwork-exec-pod
  labels:
    app: pentest
spec:
  hostNetwork: true
  containers:
  - name: hostnetwork-pod
    image: ubuntu
    command: [ "/bin/sh", "-c", "--" ]
    args: [ "while true; do sleep 30; done;" ]
```

hostNetwork gotcha

- Kubernetes usually runs your pods in their own isolated network
- hostNetwork: true
- **Doing this, bypasses network policies altogether.** Your pod will be able to communicate just like any other process that is running on the host itself.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hostnetwork-exec-pod
  labels:
    app: pentest
spec:
  hostNetwork: true
  containers:
  - name: hostnetwork-pod
    image: ubuntu
    command: [ "/bin/sh", "-c", "--" ]
    args: [ "while true; do sleep 30; done;" ]
```

Gain a reverse rootshell on a Node

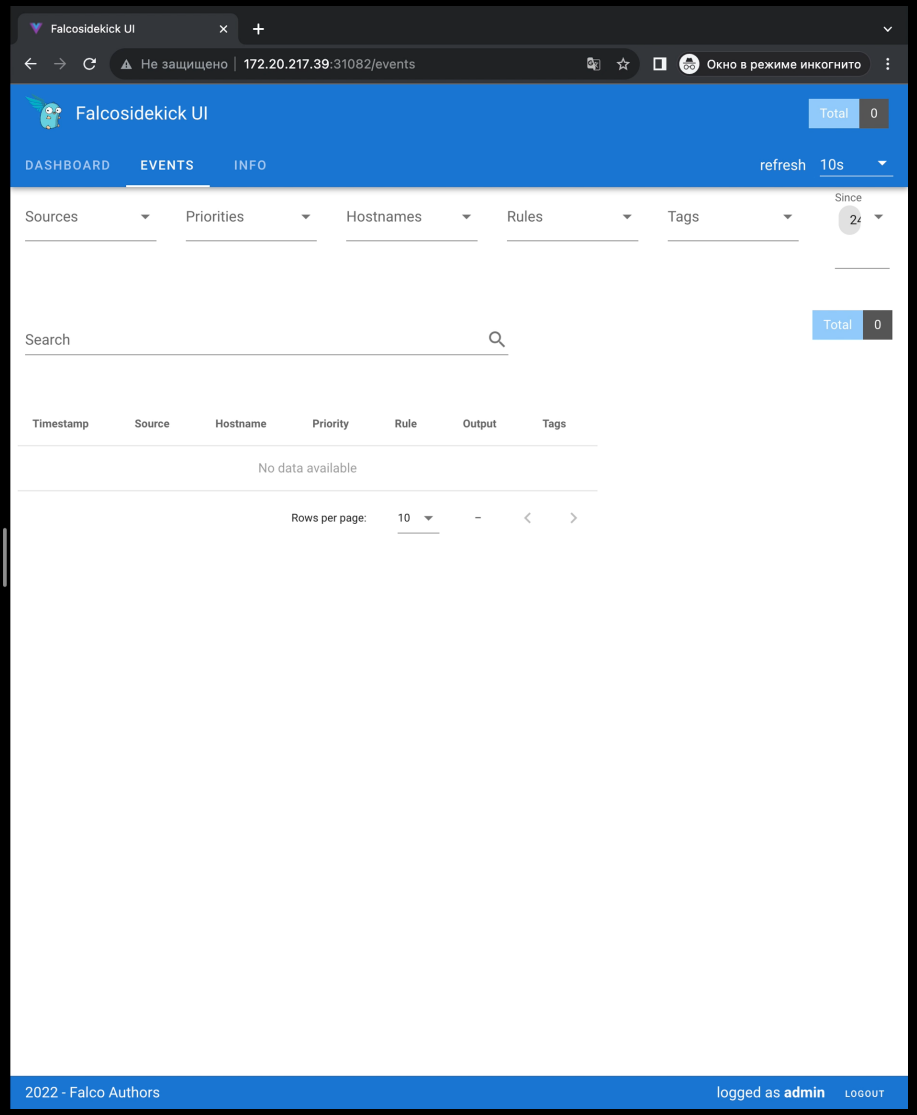


Signature / rule bypass (Falco – part 1)

```
- rule: Terminal shell in container
  desc: >
    A shell was used as the entrypoint/exec point into a container with an attached terminal. Parent
process may have
    legitimately already exited and be null (read container_entrypoint macro). Common when using
'kubectl exec' in Kubernetes.
    Correlate with k8saudit exec logs if possible to find user or serviceaccount token used (fuzzy
correlation by namespace and pod name).
    Rather than considering it a standalone rule, it may be best used when checking for other triggered
rules in this container/tty.
  condition: >
    spawned_process and container
    and shell_procs and proc.tty != 0
    and container_entrypoint
    and not user_expected_terminal_shell_in_container_conditions
  output: >
    A shell was spawned in a container with an attached terminal (user=%user.name uid=%user.uid
    user_loginuid=%user.loginuid container_info=%container.info shell=%proc.name parent=%proc.pname
    cmdline=%proc.cmdline pid=%proc.pid terminal=%proc.tty container_id=%container.id
    image=%container.image.repository namespace=%k8s.ns.name pod_name=%k8s.pod.name
exe_flags=%evt.arg.flags)
  priority: NOTICE
  tags: [maturity_stable, container, shell, mitre_execution, T1059]
```

Signature / rule bypass demo – part 1

sergey@luntry ~/Desktop _



Signature / rule bypass (Falco – part 2)

```
- macro: curl_download
  condition: proc.name = curl and
             (proc.cmdline contains " -o " or
             proc.cmdline contains " --output " or
             proc.cmdline contains " -O " or
             proc.cmdline contains " --remote-name ")

- rule: Launch Ingress Remote File Copy Tools in Container
  desc: Detect ingress remote file copy tools launched in container
  condition: >
    spawned_process and
    container and
    (ingress_remote_file_copy_procs or curl_download) and
    not user_known_ingress_remote_file_copy_activities
  output: Ingress remote file copy tool launched in container (evt_type=%evt.type user=%user.name
user_uid=%user.uid user_loginuid=%user.loginuid process=%proc.name proc_exepath=%proc.exepath
parent=%proc.pname command=%proc.cmdline terminal=%proc.tty exe_flags=%evt.arg.flags %contaner.info)
  priority: NOTICE
  tags: [maturity_sandbox, container, network, process, mitre_command_and_control, TA0011]
```

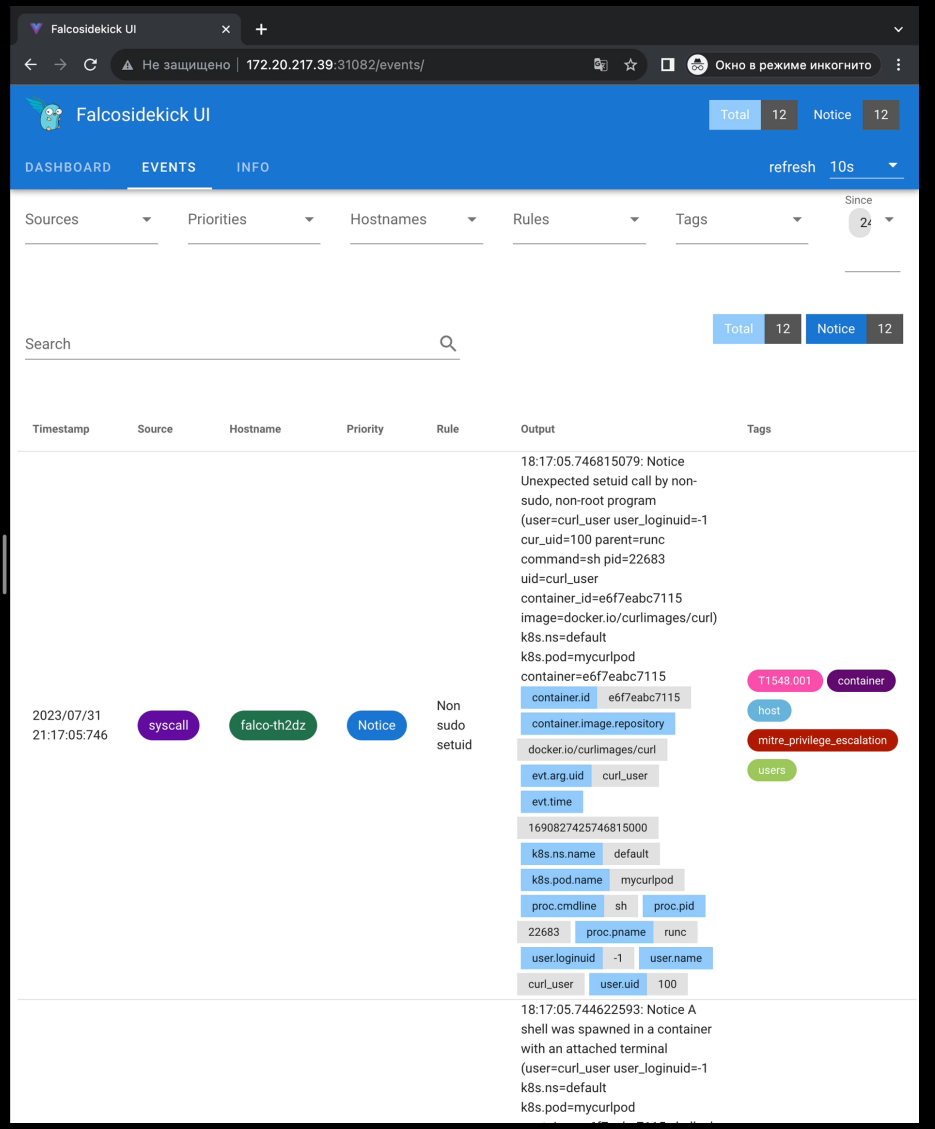
Bypass Falco signature – in Dockerfile

```
RUN mv /usr/bin/python3 /usr/bin/pton3 \
    && mv /usr/bin/curl /usr/bin/kurl \
    && mv /usr/bin/wget /usr/bin/vget \
```

Signature / rule bypass demo – part 2

```
 sergey@luntry  ~/Desktop  kubectl get po
NAME                                          READY   STATUS    RESTA
RTS      AGE
alpine                                        1/1     Running   0
         37m
falco-8bm6w                                   2/2     Running   0
         13m
falco-falcosidekick-84975cd8f4-q9rxs          1/1     Running   0
         13m
falco-falcosidekick-84975cd8f4-wfkfh          1/1     Running   0
         13m
falco-falcosidekick-ui-5854bc5d66-2crwz       1/1     Running   0
         13m
falco-falcosidekick-ui-5854bc5d66-g87tj       1/1     Running   0
         13m
falco-falcosidekick-ui-redis-0                1/1     Running   0
         13m
falco-hzbwb                                   2/2     Running   0
         13m
falco-rs4j4                                   2/2     Running   0
         13m
falco-th2dz                                   2/2     Running   0
         13m
falco-zgdtd                                   2/2     Running   0
         13m
mtkpi-pod                                     1/1     Running   0
         39m
mycurlpod                                     1/1     Running   1 (21
s ago)   2m59s
 sergey@luntry  ~/Desktop  kubectl exec mycurlpod -ti -- sh
~ $ curl goo^C

~ $ exit
command terminated with exit code 130
 ✘ sergey@luntry  ~/Desktop  _
```



Falco bypasses by default ☺

This is not the first work on Falco bypasses. There were several projects before that focused on different bypass vectors:

- Sep 2019 - by NCC Group - focused on image name manipulations to leverage Falco rules allow-lists.
- August 2020 - by Brad Geesaman - similar to previous work, exploited weak image name comparison logic to leverage Falco rules allow-lists.
- Nov 2020 - by Leonardo Di Donato - exploited twin syscalls that Falco missed, suggested other ideas used in this report.
- June 2019 and ongoing - by maintainers - ongoing issue handling the missing sister calls

- https://github.com/blackberry/Falco-bypasses
- https://www.antitree.com/2019/09/container-runtime-security-bypasses-on-falco/
- https://www.youtube.com/watch?v=nGqWskXRSmo
- https://github.com/falcosecurity/falco/security/advisories/GHSA-rfgw-vmxp-hp5g

Bypass Tetragon – rule

```yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: "sys-write"
spec:
  kprobes:
  - call: "sys_write"
    syscall: true
    args:
    - index: 0
      type: "int"
    - index: 1
      type: "char_buf"
      sizeArgIndex: 3
    - index: 2
      type: "size_t"
    # follow any non-init pids stdout e.g. exec into container
    selectors:
    - matchPIDs:
      - operator: NotIn
        followForks: true
        isNamespacePID: true
        values:
        - 1
      matchArgs:
      - index: 0
        operator: "Equal"
        values:
        - "1"
```

Bypass Tetragon – rule

```yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: "sys-write"
spec:
  kprobes:
  - call: "sys_write"
    syscall: true
    args:
    - index: 0
      type: "int"
    - index: 1
      type: "char_buf"
      sizeArgIndex: 3
    - index: 2
      type: "size_t"
    # follow any non-init pids stdout e.g. exec into container
    selectors:
    - matchPIDs:
      - operator: NotIn
        followForks: true
        isNamespacePID: true
        values:
        - 1
      matchArgs:
      - index: 0
        operator: "Equal"
        values:
        - "1"
```

```c
#include <sys/uio.h>

int main()
{
    struct iovec vecs;
    vecs.iov_base = "Writing using writev()!\n";
    vecs.iov_len = 25;

    writev(1, &vecs, 1);
}
```

[Bypassing eBPF-based Security Enforcement Tools]

NOFF ONE 2023

Conclusions

Conclusions

- With proper preparation, even secure environments aren't so secure after all

Conclusions

- With proper preparation, even secure environments aren't so secure after all

- Classic security methods (vulnerability compliance, signatures, ...) are bypassed quite easily in containerized environments (especially if you can use a custom image)

Conclusions

- With proper preparation, even secure environments aren't so secure after all
- Classic security methods (vulnerability compliance, signatures, ...) are bypassed quite easily in containerized environments (especially if you can use a custom image)
- To ensure a high level of security in Kubernetes, strive for the ZeroTrust model (NetworkPolicy, behavior models, AppArmor, Policy Engines, ...)

Conclusions

- With proper preparation, even secure environments aren't so secure after all
- Classic security methods (vulnerability compliance, signatures, ...) are bypassed quite easily in containerized environments (especially if you can use a custom image)
- To ensure a high level of security in Kubernetes, strive for the ZeroTrust model (NetworkPolicy, behavior models, AppArmor, Policy Engines, ...)
- MTKPI will both simplify pentesting and verify your actual level of security in containerized environments

Future plans

- Reduce image size
- Add new tools
- Integrate with BUS tools

Useful links

- Kubernetes Pentesting (HackCloud)
- Kubernetes Pentest Methodology
- Container Security Site

Thank you!

MTKPI

@k8security