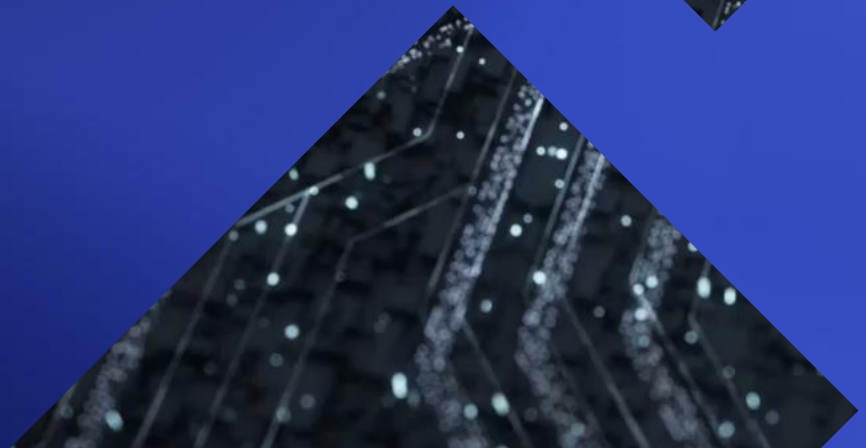
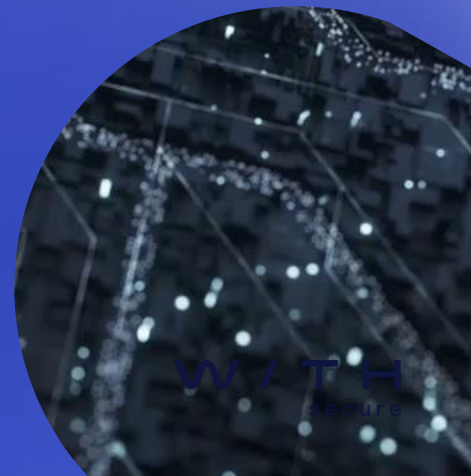
Arbitrary Code & File Execution in R/O FS – Am I Write?





Golan Myers

- Security Consultant @ WithSecure
- Focus on low-level aspect of container security
- Enthusiastic about all things containers
- Containers, containers, containers, containers!!!!!!!!

W / T H
secure

Agenda

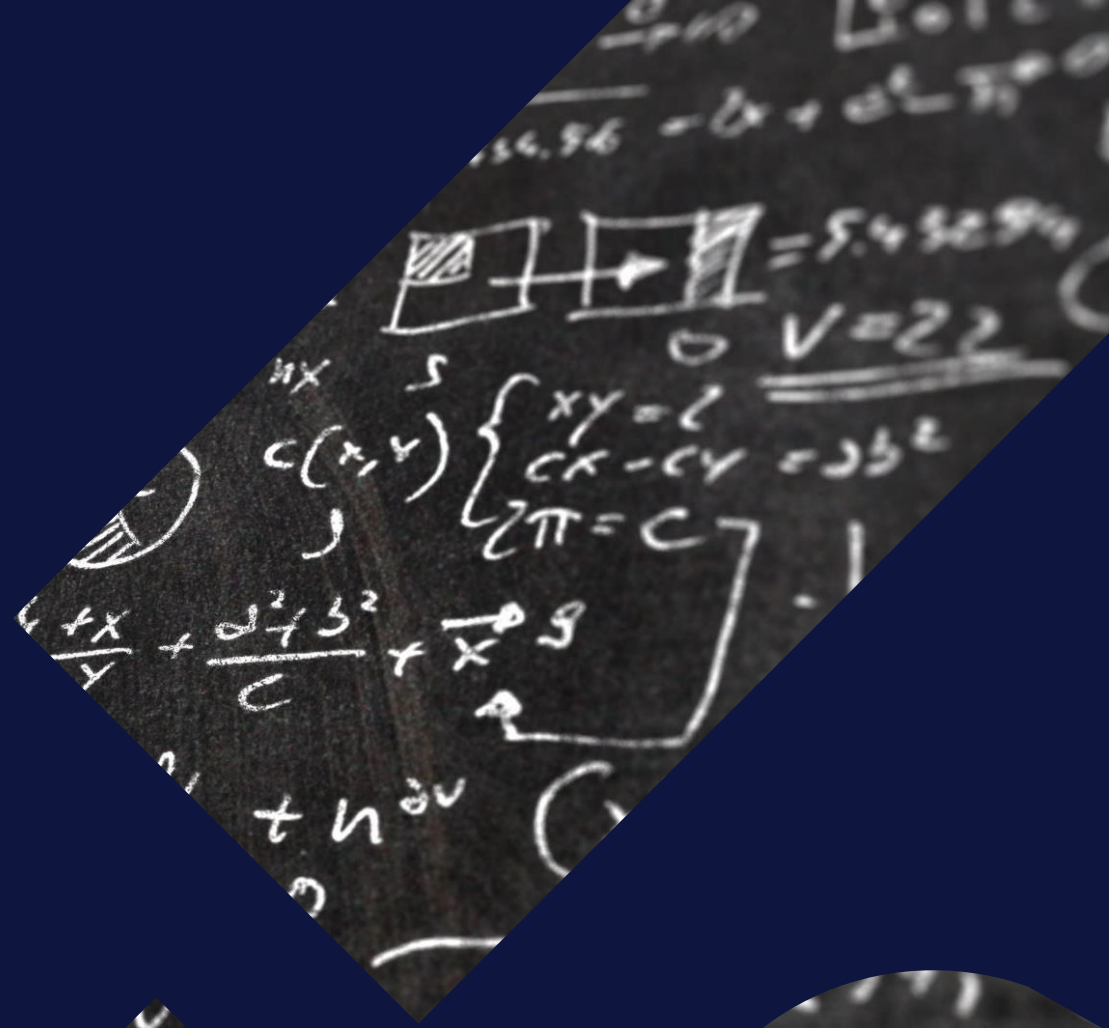
- What are RO/FS
- Why are they used
- Kubernetes and RO/FS
- An attacker’s perspective
- 3 methods to bypass R/O restrictions
- Remediation and mitigation
- Final thoughts and conclusions

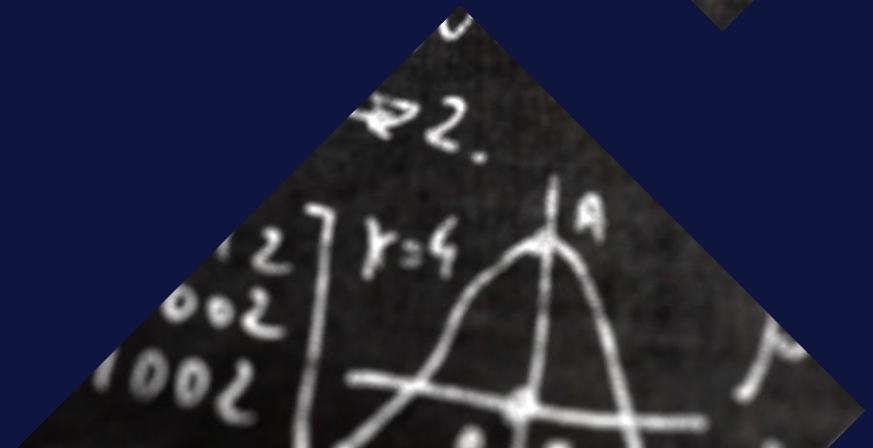
WITH secure

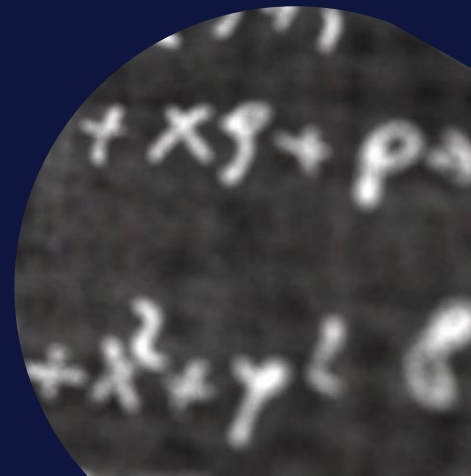
What Is A R/O FS Why Should We Use It?

- **Generally** 3 main “actions” – Read/Write/Execute
- **R/O FS** - Read/Execute
- **Security-wise** – better control/mgmt of containerised applications

```
apiVersion: v1
kind: Pod
metdata:
    name: pod
spec:
    containers:
    - name: container
      image: alpine
      securityContext:
          readOnlyRootFileSystem: true
```







I'm in, but it's R/O

What do I do now?

A Threat Actor’s Approach To a R/O Pod

- Foothold in the environment
- Enumeration from an internal perspective
- Intermediary

Attack Scenario

- Adversary with a foothold on an application pod with a R/O FS
- Execution of arbitrary code and executables vital to further the attack
- Low Privileged

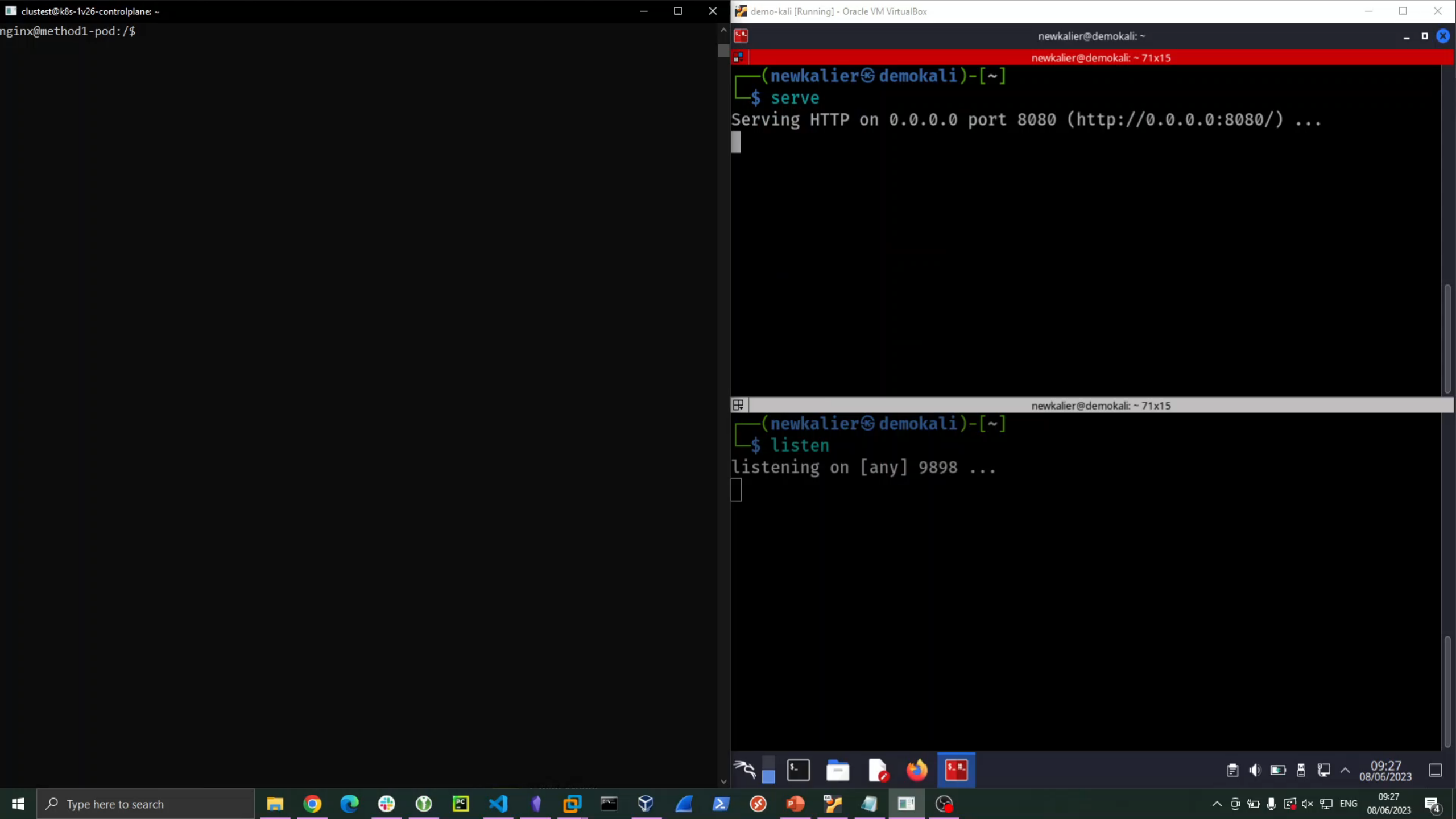
```yaml
apiVersion: v1
kind: Pod
metdata:
    name: method1-pod
spec:
    containers:
    - name: nginx
      image: nginx:latest
      securityContext:
          readOnlyRootFileSystem: true
          runAsUser: 101
      ports:
          - containerPort: 80
      volumeMounts:
         - mountPath: /var/run
           name: run
         - mountPath: /var/cache/nginx
           name: nginx-cache
    securityContext:
      seccompProfile:
          type: RuntimeDefault
    volumes:
      - name: run
        emptyDir: {}
      - name: nginx-cache
        emptyDir: {}
```

Method #1

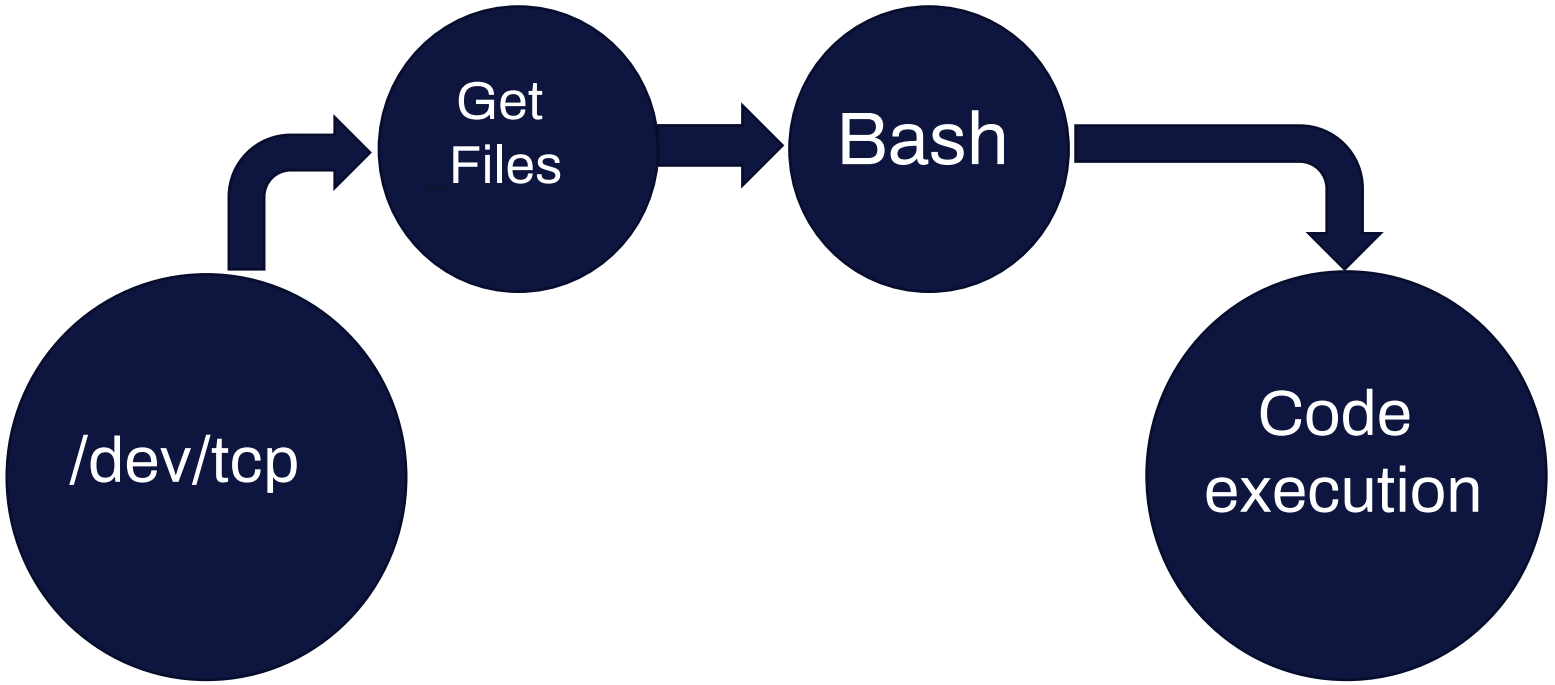
- R/O FS
- Nginx image
- Contains Bash
- No standard network tools to retrieve data (wget, curl, etc..)

WITH secure

WITH secure


clustest@k8s-1v26-controlplane: ~
nginx@method1-pod:/$
demo-kali [Running] - Oracle VM VirtualBox
newkalier@demokali: ~
newkalier@demokali: ~ 71x15
(newkalier㉿demokali)-[~]
$ serve
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
newkalier@demokali: ~ 71x15
(newkalier㉿demokali)-[~]
$ listen
listening on [any] 9898 ...
09:27
08/06/2023
Type here to search
ENG
09:27
08/06/2023

What we saw



```
apiVersion: v1
kind: Pod
metdata:
    name: method2-pod
spec:
    containers:
    - args:
      - alpine
      name: alpine
      image: alpine
      command:
          - “sleep”
          - “3600”
      securityContext:
          readOnlyRootFileSystem: true
          runAsUser: 65534
    securityContext:
      seccompProfile:
          type: RuntimeDefault
```

Method #2

- R/O FS
- Alpine image
- Contains sh
- No standard network tools to retrieve data (wget*, curl, etc..)

WITH secure


DEMO
PgUp

clustest@k8s-1v26-controlplane: ~

~ $

demo-kali [Running] - Oracle VM VirtualBox

newkalier@demokali: ~

newkalier@demokali: ~ 66x30

```
┌──(newkalier㉿demokali)-[~]
└─$ listen
listening on [any] 9898 ...
```

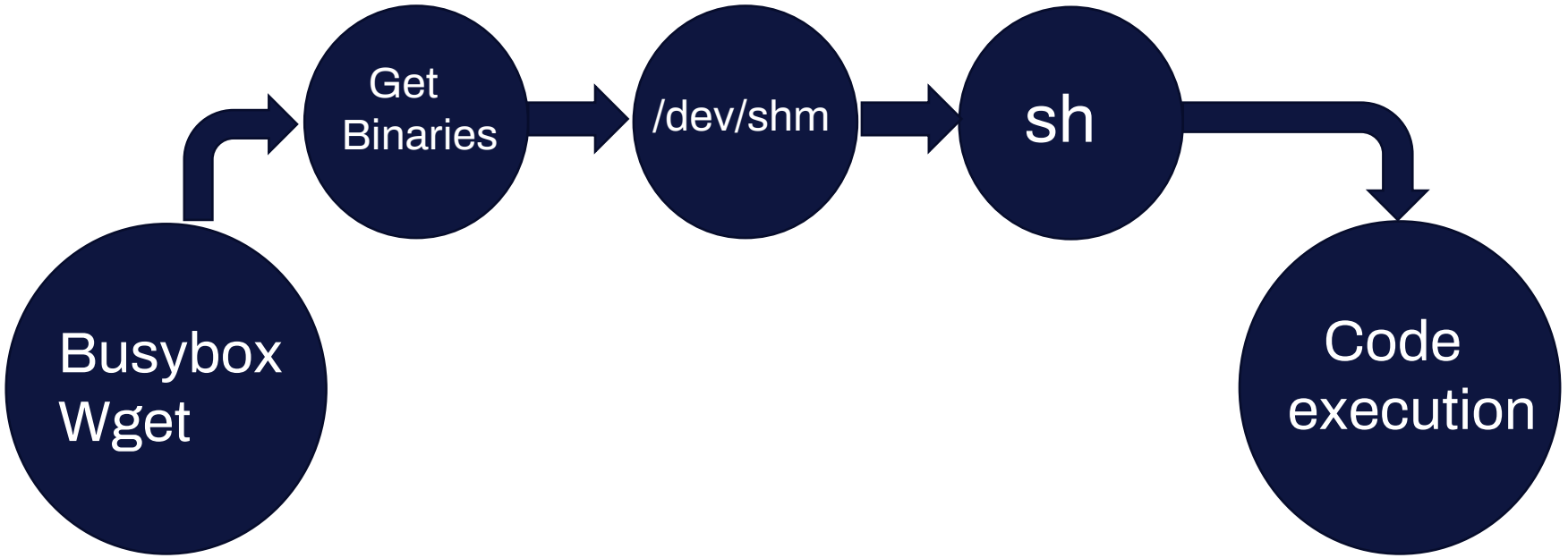
09:34
08/06/2023

Type here to search

ENG

09:34
08/06/2023

What we saw



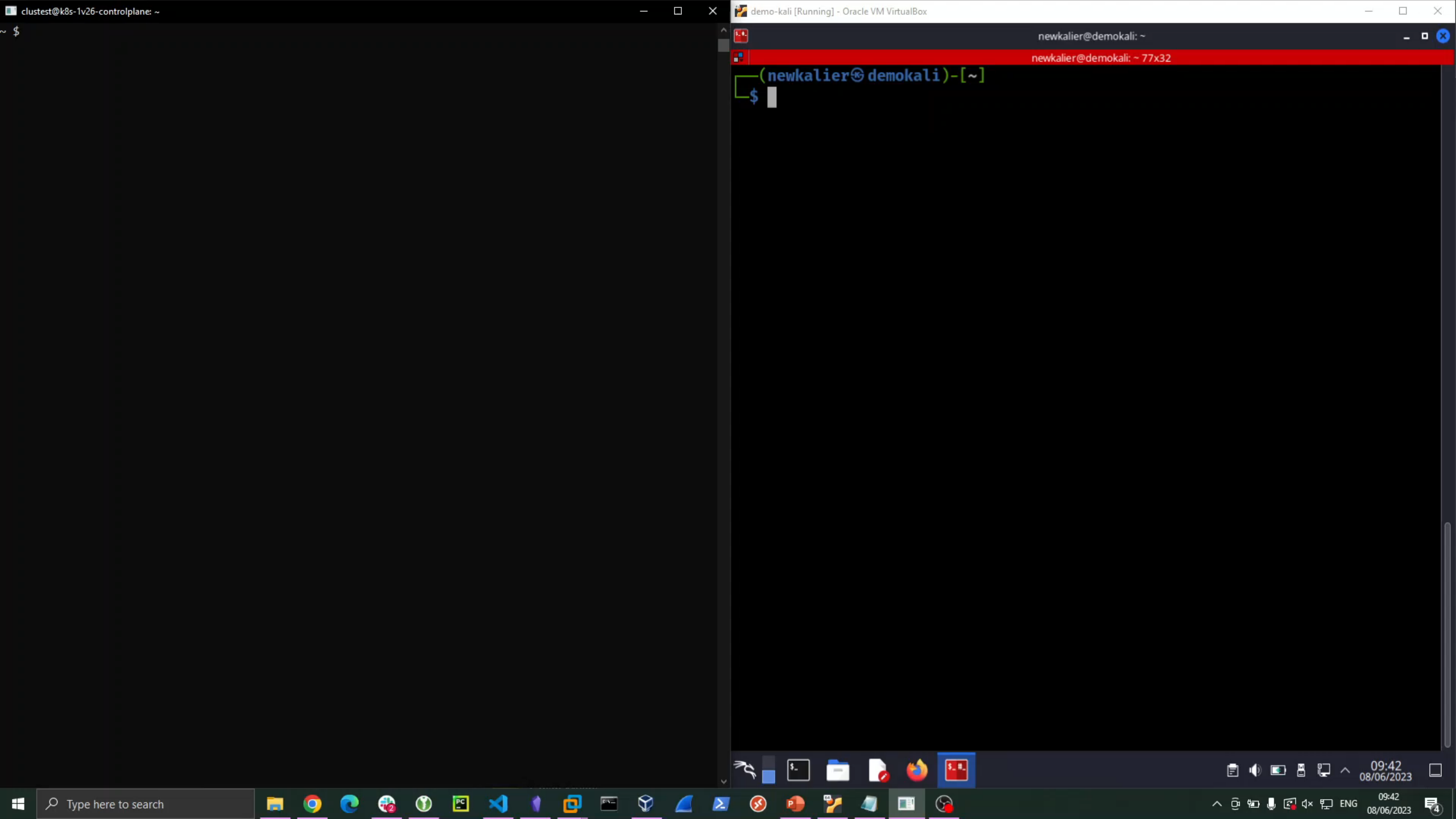
```
apiVersion: v1
kind: Pod
metdata:
    name: method2-pod
spec:
    containers:
    - args:
      - alpine
      name: alpine
      image: alpine
      command:
          - “sleep”
          - “3600”
      securityContext:
          readOnlyRootFileSystem: true
          runAsUser: 65534
    securityContext:
      seccompProfile:
          type: RuntimeDefault
```

Method #3

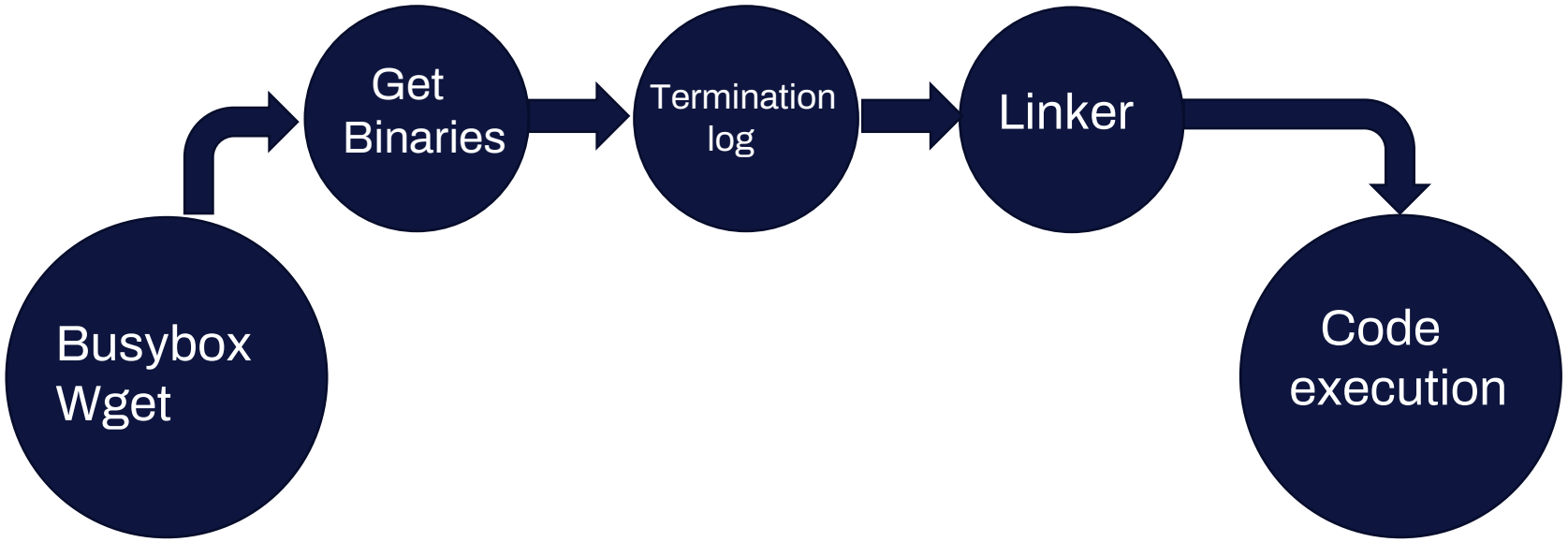
- R/O FS
- Alpine image
- Contains sh
- No standard network tools to retrieve data (wget*, curl, etc..)

WITH secure

WITH secure


clustest@k8s-1v26-controlplane: ~
~ $
demo-kali [Running] - Oracle VM VirtualBox
newkalier@demokali: ~
newkalier@demokali: ~ 77x32
(newkalier㉿demokali)-[~]
$
09:42
08/06/2023
Type here to search
ENG
09:42
08/06/2023

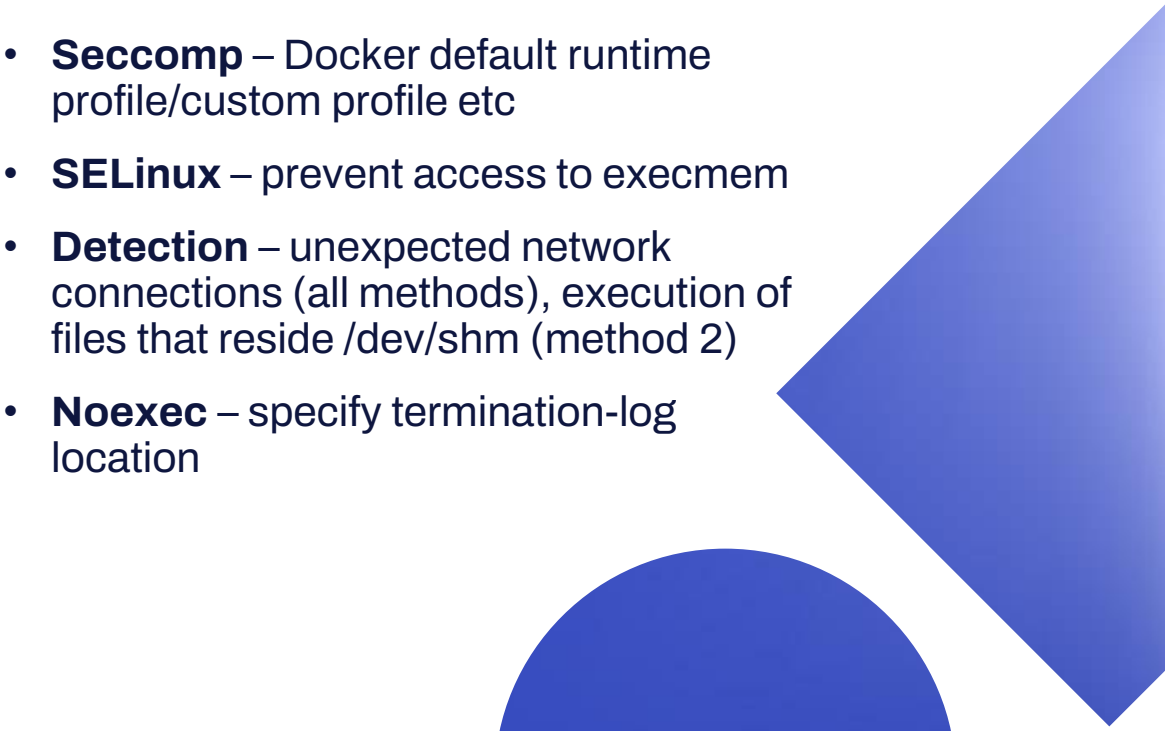
What we saw


Busybox
Wget
Get
Binaries
Termination
log
Linker
Code
execution

```
Detecting File Execution From /dev/shm
- rule: Execution from /dev/shm
  desc: This rule detects file execution from the /dev/shm
directory, a common tactic for threat actors to stash their
readable+writable+(sometimes)executable files.
  condition: >
    spawned_process and
    (proc.exe startswith "/dev/shm/" or
    (proc.cwd startswith "/dev/shm/" and proc.exe startswith "./" ) or
    (shell_procs and proc.args startswith "-c /dev/shm") or
    (shell_procs and proc.args startswith "-i /dev/shm") or
    (shell_procs and proc.args startswith "/dev/shm") or
    (proc.args contains "/dev/shm" or proc.cwd startswith
"/dev/shm") or
    (proc.cwd startswith "/dev/shm/" and proc.args startswith "./" ))
and
    not container.image.repository in (falco_privileged_images,
trusted_images)
  output: "File execution detected from /dev/shm
(proc.cmdline=%proc.cmdline connection=%fd.name
user.name=%user.name user.loginuid=%user.loginuid
container.id=%container.id evt.type=%evt.type evt.res=%evt.res
proc.pid=%proc.pid proc.cwd=%proc.cwd proc.ppid=%proc.ppid
proc.pcmdline=%proc.pcmdline proc.sid=%proc.sid
proc.exepath=%proc.exepath user.uid=%user.uid
user.loginname=%user.loginname group.gid=%group.gid
group.name=%group.name container.name=%container.name
image=%container.image.repository)"
  priority: WARNING
```

Defending against these attacks

- **Seccomp** – Docker default runtime profile/custom profile etc
- **SELinux** – prevent access to execmem
- **Detection** – unexpected network connections (all methods), execution of files that reside /dev/shm (method 2)
- **Noexec** – specify termination-log location



WITH secure

Rapping Up

- R/O FS for container IS recommended
- Just R/O FS ≠ Security

</dev/audience

Golan Myers:

https://labs.withsecure.com/publications/executing-arbitrary-code-executables-in-read-only-filesystems