# OWASP Docker(/Container) Top 10

Dr. Dirk Wetter

@drwetter

https://de.wikipedia.org/wiki/Datei:Container_ship_MSC_Zoe_on_the_river_Elbe_in_front_of_Blankenese.jpg by Hummelhummel, CC BY-SA 3.0

# about:me

## Independent Consultant - Information Security

(self-employed)

OWASP

- Organized + chaired AppSec Europe 2013 in Hamburg
- Involved in few following European conferences

Open Source

- Old „fart“: First publication 1995 about Linux (heise)
- >= 60 publications in magazines
- Co-authored Linux book ages ago
- TLS-Checker testssl.sh

- PhD in natural science

- 20+ years paid profession in infosec
- Pentests, consulting, training

- Application, system, network security
- Information security management

# What awaits you

- **Introducing Docker Top 10**
  - Motivation
  - Idea
  - Status

# Motivation

- **Prerequisite: Understand what you’re doing**

https://twitter.com/AwardsDarwin/status/1174986709794197504

# Motivation

- **Prerequisite: Understand what you’re doing**
  - Underestimation of complexity
    - Building a new network with new systems
  - Managers not knowing required skills well enough
    - Devs are no system / network architects
    - An average admin (Ops guy) isn’t either

# Idea

- **Docker/container security**

→ is about **system and network security**.

→ *Project is suggesting controls to minimize attack surfaces*

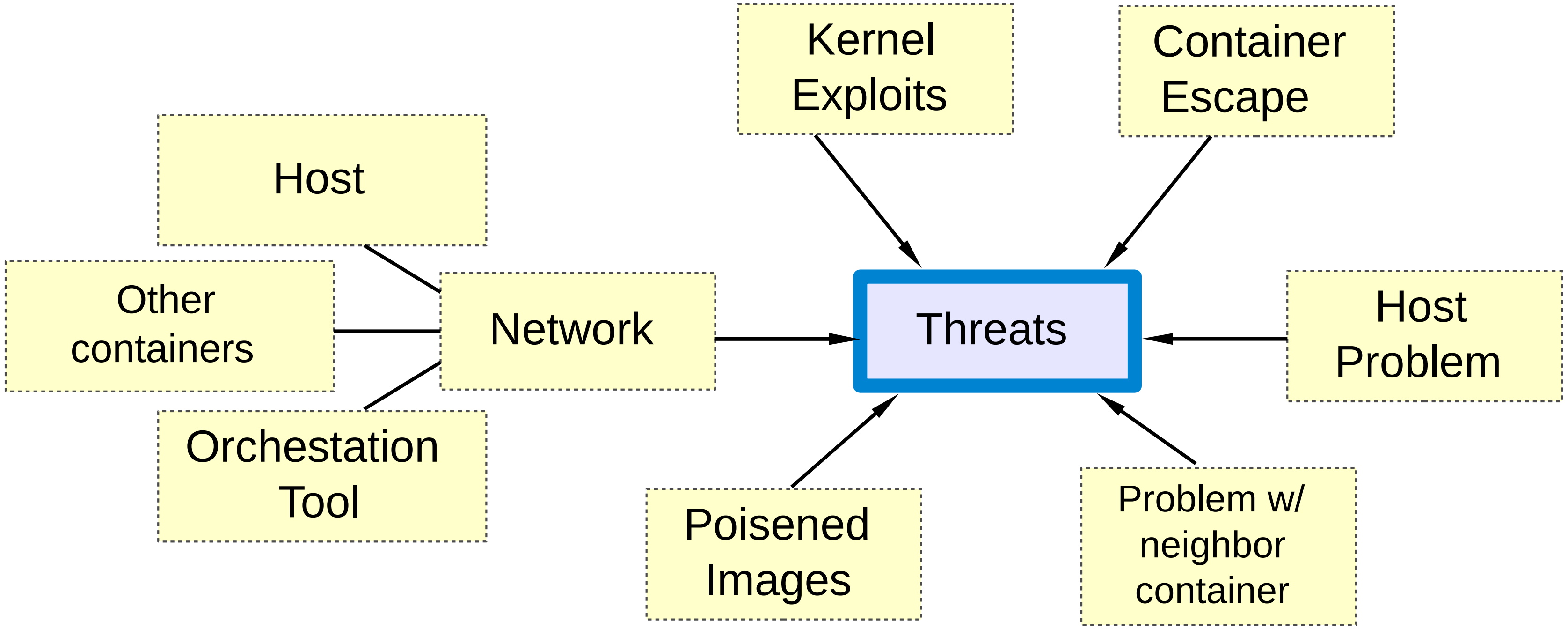
# Threat modeling

- **Threats to my containers?**

→ **Enumerate!**

https://imgur.com/gallery/ZdEQDwh

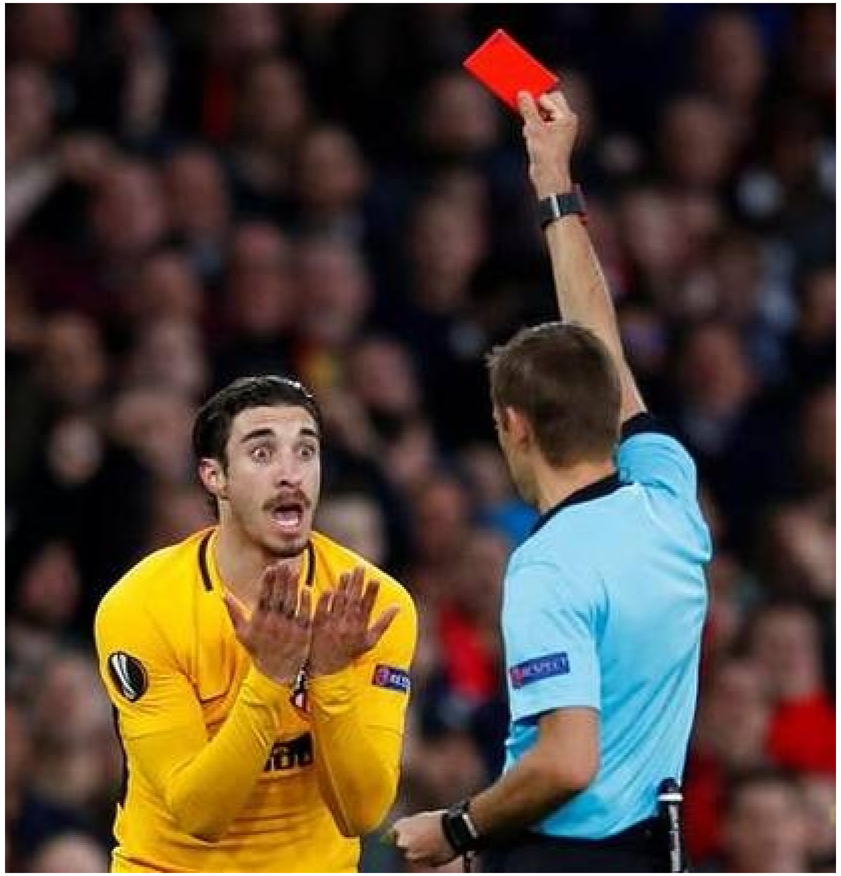
# Threat modeling



# Threat modeling

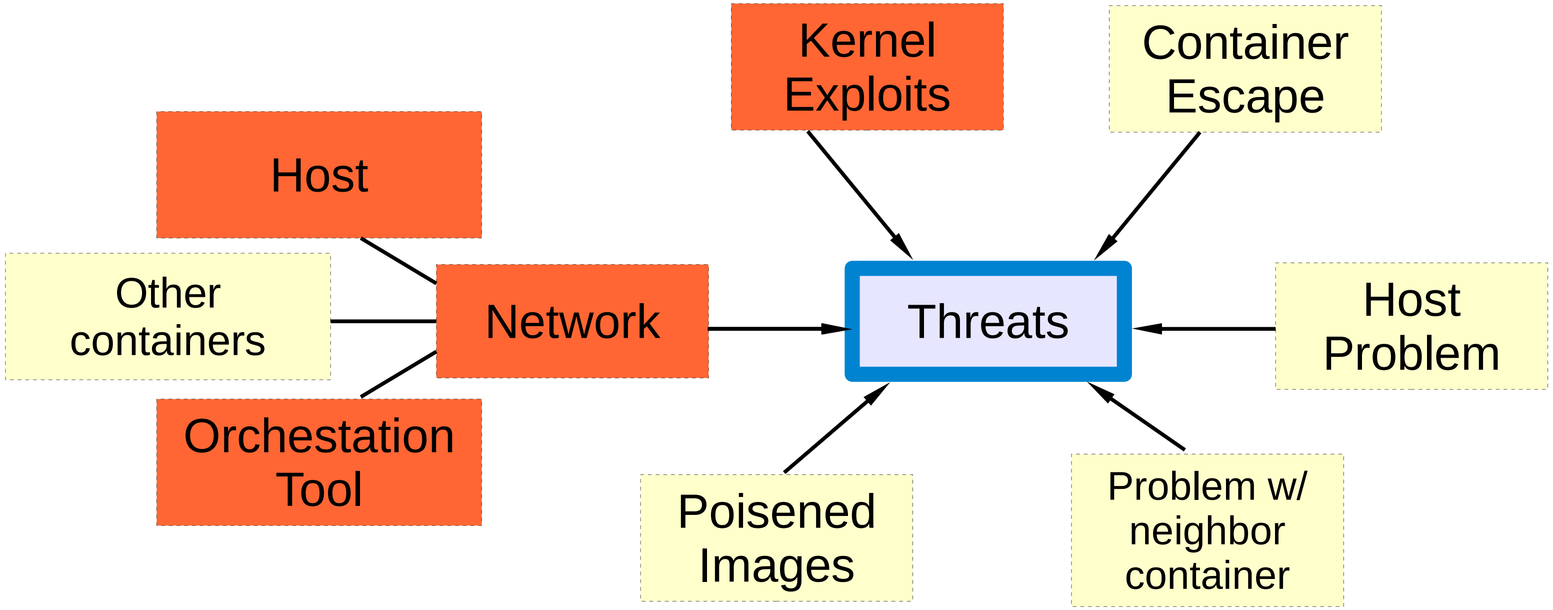
- **Biggest Threats a.k.a. game over**
  - Attack to host via
    - Network services (or just protocol flaw)
    - Kernel exploit
  - Attack to orchestration
    - Via network

      Your management backplane!



Reuters, see https://www.thehindu.com/sport/football/10-man-atletico-and-arsenal-end-1-1-in-europa-semifinal/article23697997.ece

# Threat modeling



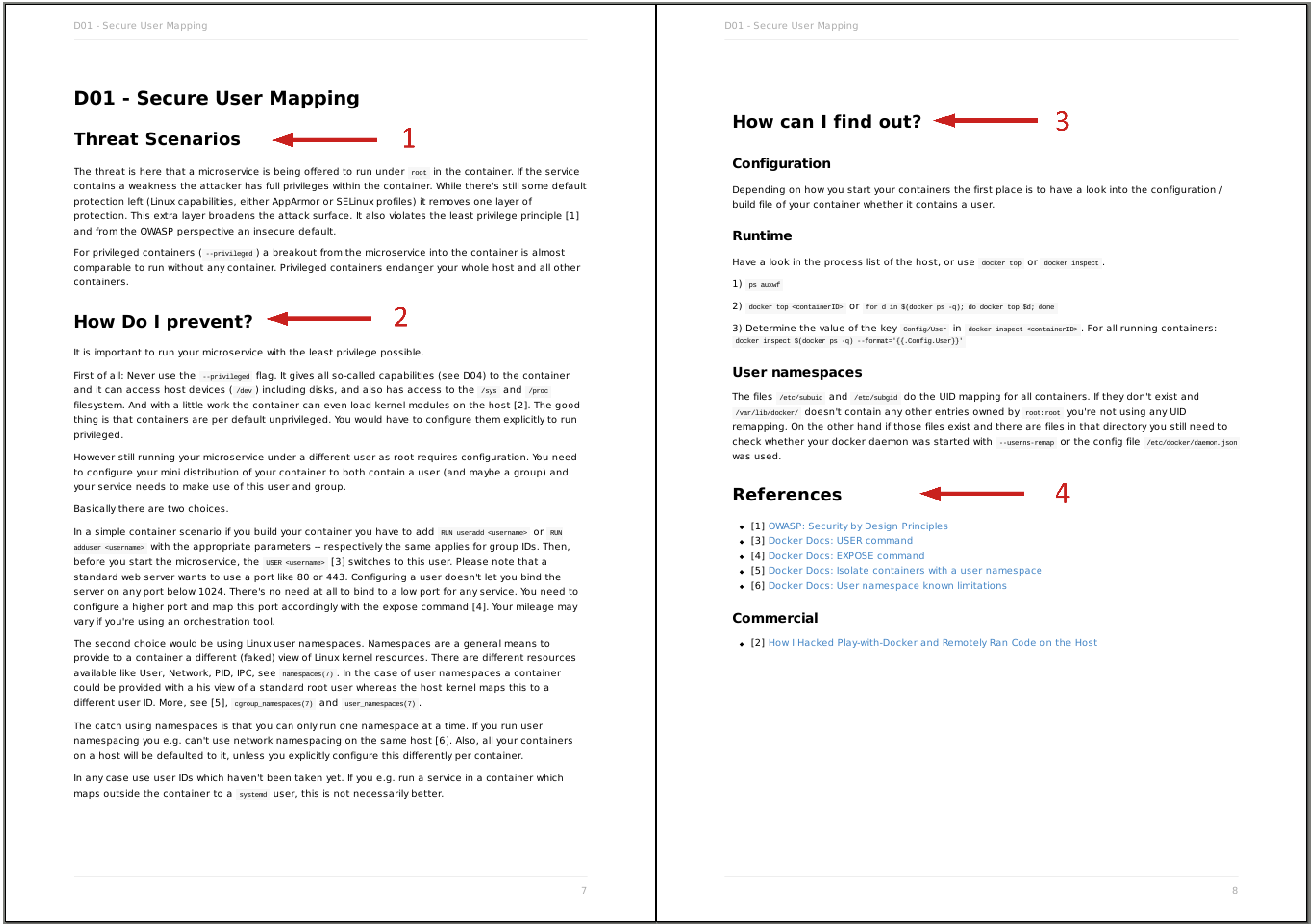
① Introduction

② Threats

③ Overview

| ④ Top# | Title |
|---|---|
| D01 | Secure User Mapping |
| D02 | Patch Management Policy |
| D03 | Network Segmentation |
| D04 | Secure Defaults and Hardening |
| D05 | Maintain Security Contexts |
| D06 | Protect Secrets |
| D07 | Ressource Protection |
| D08 | Container Image Integrity and Origin |
| D09 | Follow Immutable Paradigm |
| D10 | Logging |

⑤ What‘s next for …



# D01 - Secure User Mapping

## Threat Scenarios

The threat is here that a microservice is being offered to run under `root` in the container. If the service contains a weakness the attacker has full privileges within the container. While there's still some default protection left (Linux capabilities, either AppArmor or SELinux profiles) it removes one layer of protection. This extra layer broadens the attack surface. It also violates the least privilege principle [1] and from the OWASP perspective an insecure default.

For privileged containers ( `--privileged` ) a breakout from the microservice into the container is almost comparable to run without any container. Privileged containers endanger your whole host and all other containers.

## How Do I prevent?

It is important to run your microservice with the least privilege possible.

First of all: Never use the `--privileged` flag. It gives all so-called capabilities (see D04) to the container and it can access host devices ( `/dev` ) including disks, and also has access to the `/sys` and `/proc` filesystem. And with a little work the container can even load kernel modules on the host [2]. The good thing is that containers are per default unprivileged. You would have to configure them explicitly to run privileged.

However still running your microservice under a different user as root requires configuration. You need to configure your mini distribution of your container to both contain a user (and maybe a group) and your service needs to make use of this user and group.

Basically there are two choices.

In a simple container scenario if you build your container you have to add `RUN useradd <username>` or `RUN adduser <username>` with the appropriate parameters -- respectively the same applies for group IDs. Then, before you start the microservice, the `USER <username>` [3] switches to this user. Please note that a standard web server wants to use a port like 80 or 443. Configuring a user doesn't let you bind the server on any port below 1024. There's no need at all to bind to a low port for any service. You need to configure a higher port and map this port accordingly with the expose command [4]. Your mileage may vary if you're using an orchestration tool.

The second choice would be using Linux user namespaces. Namespaces are a general means to provide to a container a different (faked) view of Linux kernel resources. There are different resources available like User, Network, PID, IPC, see `namespaces(7)` . In the case of user namespaces a container could be provided with a his view of a standard root user whereas the host kernel maps this to a different user ID. More, see [5], `cgroup_namespaces(7)` and `user_namespaces(7)` .

The catch using namespaces is that you can only run one namespace at a time. If you run user namespacing you e.g. can't use network namespacing on the same host [6]. Also, all your containers on a host will be defaulted to it, unless you explicitly configure this differently per container.

In any case use user IDs which haven't been taken yet. If you e.g. run a service in a container which maps outside the container to a `systemd` user, this is not necessarily better.

## How can I find out?

### Configuration

Depending on how you start your containers the first place is to have a look into the configuration / build file of your container whether it contains a user.

### Runtime

Have a look in the process list of the host, or use `docker top` or `docker inspect` .

1) `ps auxwf`

2) `docker top <containerID>` or `for d in $(docker ps -q); do docker top $d; done`

3) Determine the value of the key `Config/User` in `docker inspect <containerID>` . For all running containers: `docker inspect $(docker ps -q) --format='{{.Config.User}}'`

### User namespaces

The files `/etc/subuid` and `/etc/subgid` do the UID mapping for all containers. If they don't exist and `/var/lib/docker/` doesn't contain any other entries owned by `root:root` you're not using any UID remapping. On the other hand if those files exist and there are files in that directory you still need to check whether your docker daemon was started with `--userns-remap` or the config file `/etc/docker/daemon.json` was used.

## References

- [1] OWASP: Security by Design Principles
- [3] Docker Docs: USER command
- [4] Docker Docs: EXPOSE command
- [5] Docker Docs: Isolate containers with a user namespace
- [6] Docker Docs: User namespace known limitations

### Commercial

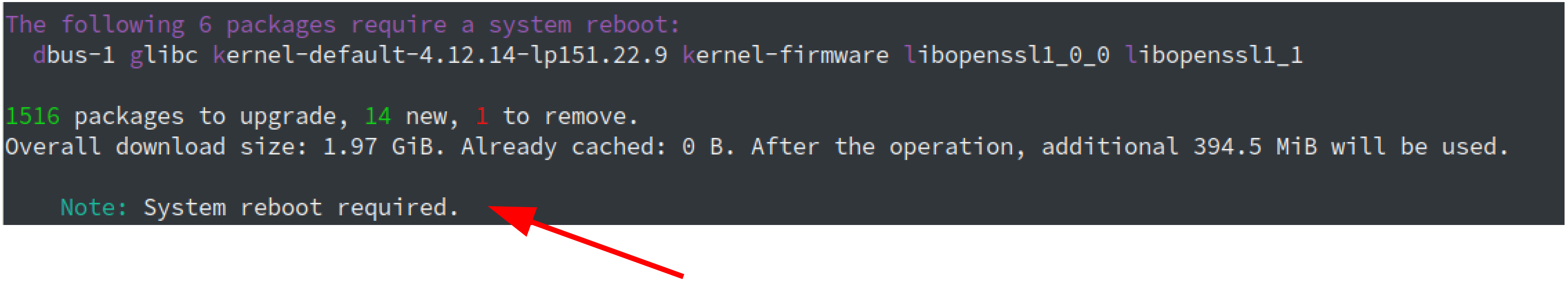
- [2] How I Hacked Play-with-Docker and Remotely Ran Code on the Host

- **D02 – Patch Management Policy**

  → **A9 in OWASP Top 10**
  *Using Components with Known Vulnerabilities*

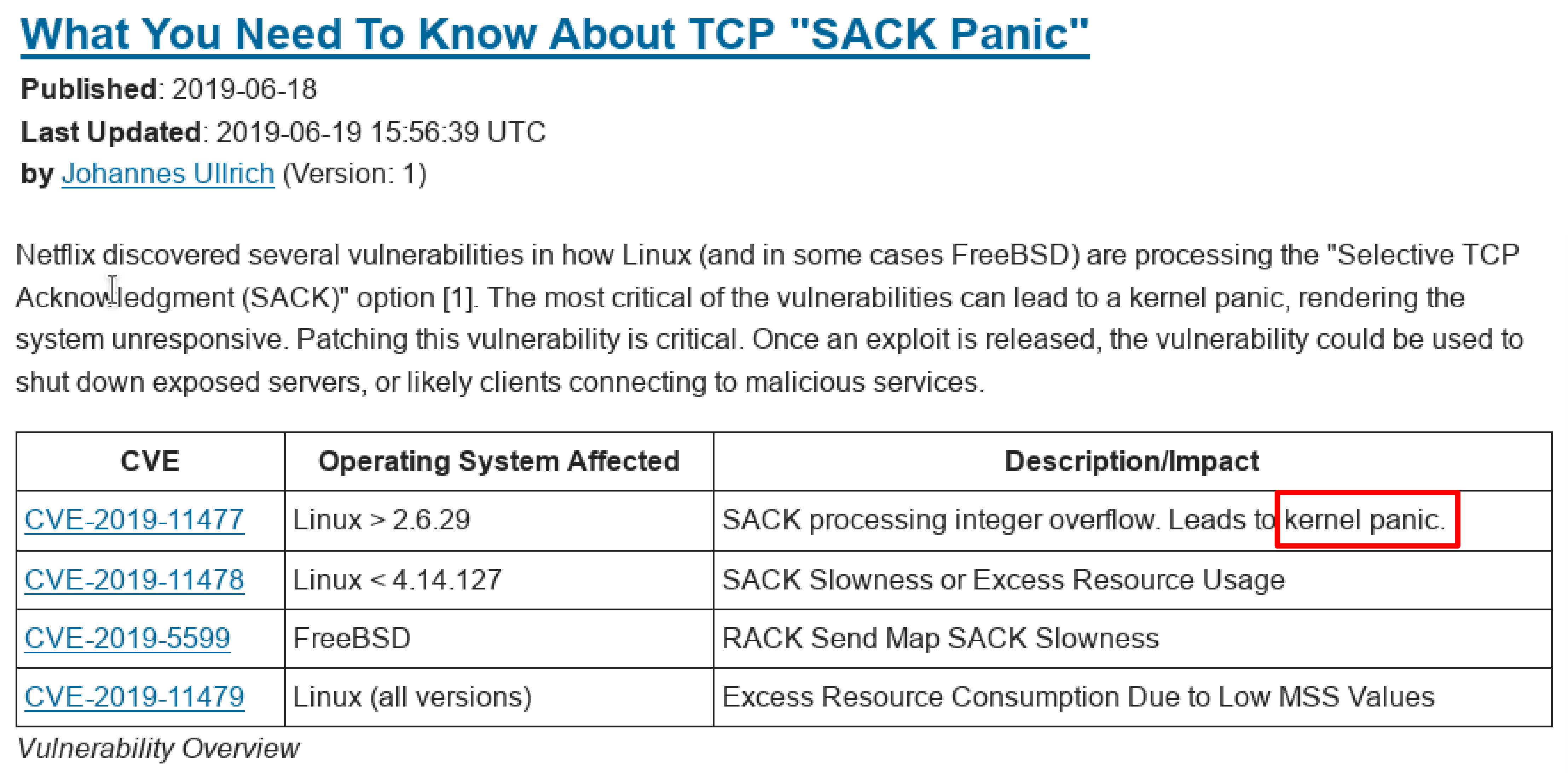
  - Host
  - Container Orchestration
  - Container Images
  - (Container Software)

- **D02 – Patch Management Policy**
  - **Host**
    - Kernel-Syscalls
      - Window for privilege escalation!
    - Hopefully nothing is exposed, see D04



```
The following 6 packages require a system reboot:
  dbus-1 glibc kernel-default-4.12.14-lp151.22.9 kernel-firmware libopenssl1_0_0 libopenssl1_1

1516 packages to upgrade, 14 new, 1 to remove.
Overall download size: 1.97 GiB. Already cached: 0 B. After the operation, additional 394.5 MiB will be used.

    Note: System reboot required.
```

D02



# What You Need To Know About TCP "SACK Panic"

**Published**: 2019-06-18

**Last Updated**: 2019-06-19 15:56:39 UTC

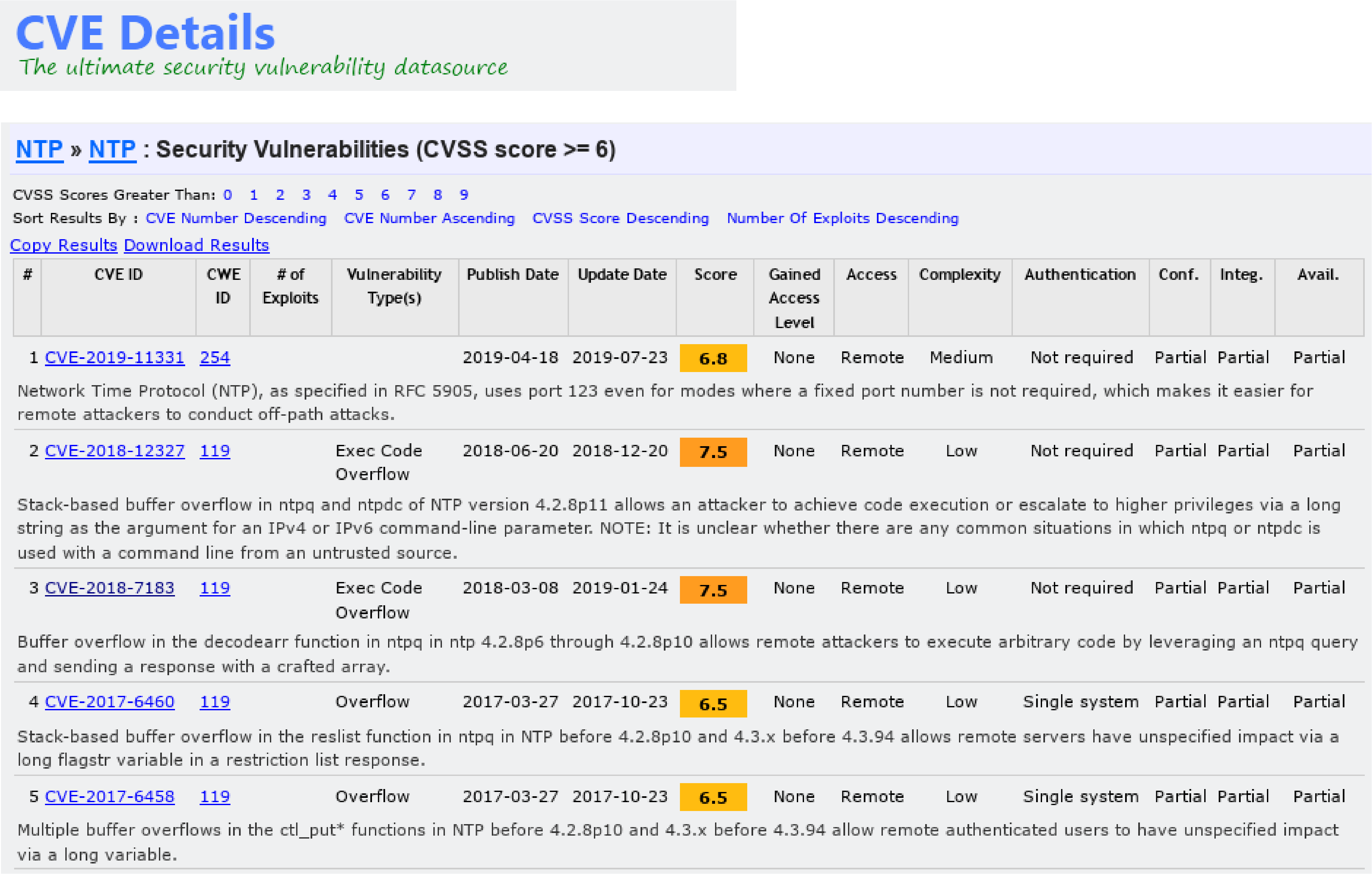
**by** Johannes Ullrich (Version: 1)

Netflix discovered several vulnerabilities in how Linux (and in some cases FreeBSD) are processing the "Selective TCP Acknowledgment (SACK)" option [1]. The most critical of the vulnerabilities can lead to a kernel panic, rendering the system unresponsive. Patching this vulnerability is critical. Once an exploit is released, the vulnerability could be used to shut down exposed servers, or likely clients connecting to malicious services.

| CVE | Operating System Affected | Description/Impact |
|---|---|---|
| CVE-2019-11477 | Linux > 2.6.29 | SACK processing integer overflow. Leads to kernel panic. |
| CVE-2019-11478 | Linux < 4.14.127 | SACK Slowness or Excess Resource Usage |
| CVE-2019-5599 | FreeBSD | RACK Send Map SACK Slowness |
| CVE-2019-11479 | Linux (all versions) | Excess Resource Consumption Due to Low MSS Values |

*Vulnerability Overview*

https://isc.sans.edu/forums/diary/What+You+Need+To+Know+About+TCP+SACK+Panic/25046/



# CVE Details

*The ultimate security vulnerability datasource*

## NTP » NTP : Security Vulnerabilities (CVSS score >= 6)

CVSS Scores Greater Than: 0 1 2 3 4 5 6 7 8 9

Sort Results By : CVE Number Descending CVE Number Ascending CVSS Score Descending Number Of Exploits Descending

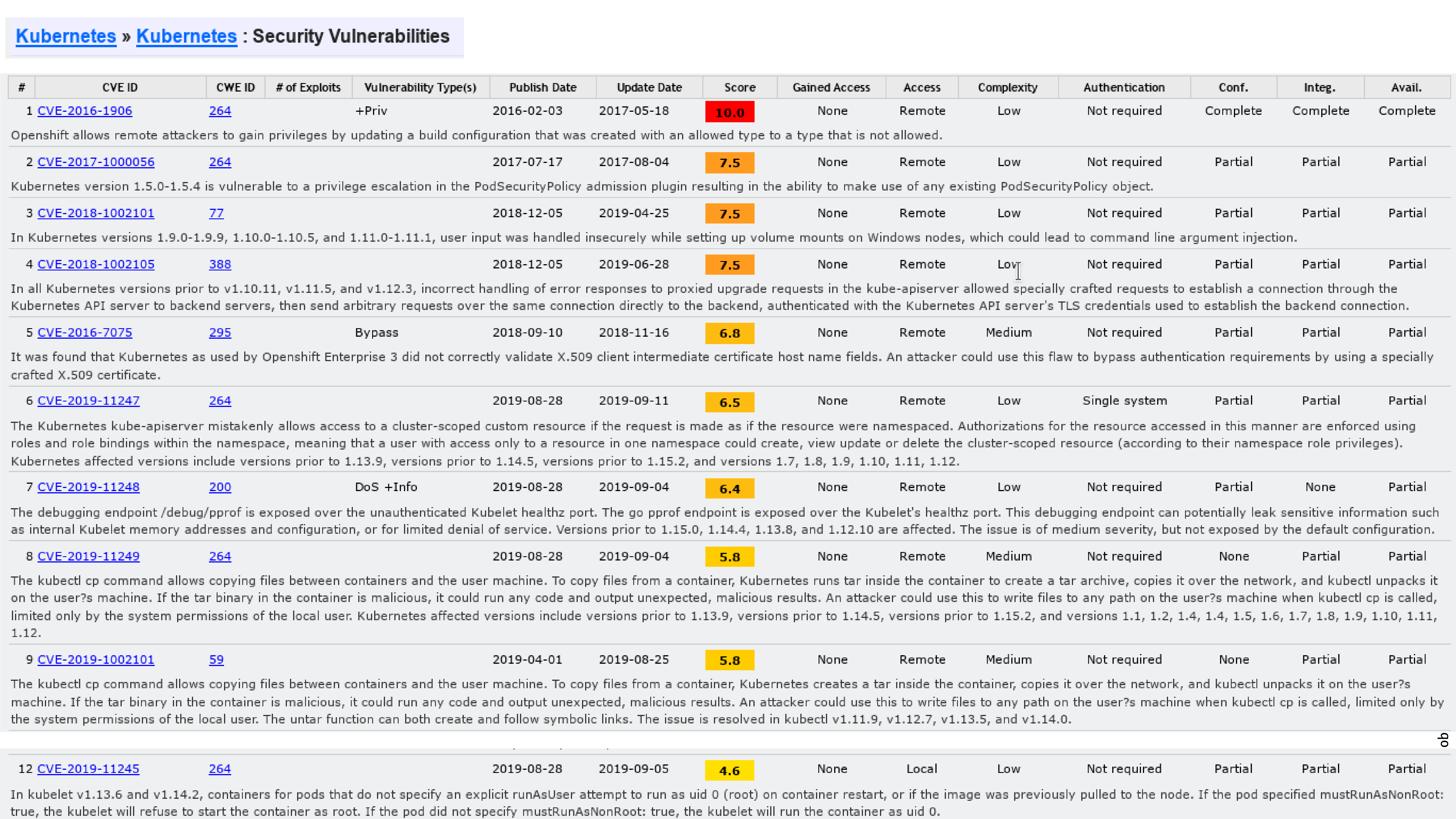
Copy Results Download Results

<table>
<tr><th>#</th><th>CVE ID</th><th>CWE ID</th><th># of Exploits</th><th>Vulnerability Type(s)</th><th>Publish Date</th><th>Update Date</th><th>Score</th><th>Gained Access Level</th><th>Access</th><th>Complexity</th><th>Authentication</th><th>Conf.</th><th>Integ.</th><th>Avail.</th></tr>
<tr><td>1</td><td>CVE-2019-11331</td><td>254</td><td></td><td></td><td>2019-04-18</td><td>2019-07-23</td><td>6.8</td><td>None</td><td>Remote</td><td>Medium</td><td>Not required</td><td>Partial</td><td>Partial</td><td>Partial</td></tr>
<tr><td colspan="15">Network Time Protocol (NTP), as specified in RFC 5905, uses port 123 even for modes where a fixed port number is not required, which makes it easier for remote attackers to conduct off-path attacks.</td></tr>
<tr><td>2</td><td>CVE-2018-12327</td><td>119</td><td></td><td>Exec Code Overflow</td><td>2018-06-20</td><td>2018-12-20</td><td>7.5</td><td>None</td><td>Remote</td><td>Low</td><td>Not required</td><td>Partial</td><td>Partial</td><td>Partial</td></tr>
<tr><td colspan="15">Stack-based buffer overflow in ntpq and ntpdc of NTP version 4.2.8p11 allows an attacker to achieve code execution or escalate to higher privileges via a long string as the argument for an IPv4 or IPv6 command-line parameter. NOTE: It is unclear whether there are any common situations in which ntpq or ntpdc is used with a command line from an untrusted source.</td></tr>
<tr><td>3</td><td>CVE-2018-7183</td><td>119</td><td></td><td>Exec Code Overflow</td><td>2018-03-08</td><td>2019-01-24</td><td>7.5</td><td>None</td><td>Remote</td><td>Low</td><td>Not required</td><td>Partial</td><td>Partial</td><td>Partial</td></tr>
<tr><td colspan="15">Buffer overflow in the decodearr function in ntpq in ntp 4.2.8p6 through 4.2.8p10 allows remote attackers to execute arbitrary code by leveraging an ntpq query and sending a response with a crafted array.</td></tr>
<tr><td>4</td><td>CVE-2017-6460</td><td>119</td><td></td><td>Overflow</td><td>2017-03-27</td><td>2017-10-23</td><td>6.5</td><td>None</td><td>Remote</td><td>Low</td><td>Single system</td><td>Partial</td><td>Partial</td><td>Partial</td></tr>
<tr><td colspan="15">Stack-based buffer overflow in the reslist function in ntpq in NTP before 4.2.8p10 and 4.3.x before 4.3.94 allows remote servers have unspecified impact via a long flagstr variable in a restriction list response.</td></tr>
<tr><td>5</td><td>CVE-2017-6458</td><td>119</td><td></td><td>Overflow</td><td>2017-03-27</td><td>2017-10-23</td><td>6.5</td><td>None</td><td>Remote</td><td>Low</td><td>Single system</td><td>Partial</td><td>Partial</td><td>Partial</td></tr>
<tr><td colspan="15">Multiple buffer overflows in the ctl_put* functions in NTP before 4.2.8p10 and 4.3.x before 4.3.94 allow remote authenticated users to have unspecified impact via a long variable.</td></tr>
</table>

- **Top 2: Patch Management Policy**
  - **Host**
    - Auto-updates to the rescue!
      - `unattended-upgrade(8)` and friends
      - monitor: `apt-listchanges(1)`

D02

- **Top 2: Patch Management Policy**
  - **Container Orchestration**
    - Don’t forget to patch the management as needed ;-)



## Kubernetes » Kubernetes : Security Vulnerabilities

| # | CVE ID | CWE ID | # of Exploits | Vulnerability Type(s) | Publish Date | Update Date | Score | Gained Access | Access | Complexity | Authentication | Conf. | Integ. | Avail. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | CVE-2016-1906 | 264 | | +Priv | 2016-02-03 | 2017-05-18 | 10.0 | None | Remote | Low | Not required | Complete | Complete | Complete |
| Openshift allows remote attackers to gain privileges by updating a build configuration that was created with an allowed type to a type that is not allowed. | | | | | | | | | | | | | | |
| 2 | CVE-2017-1000056 | 264 | | | 2017-07-17 | 2017-08-04 | 7.5 | None | Remote | Low | Not required | Partial | Partial | Partial |
| Kubernetes version 1.5.0-1.5.4 is vulnerable to a privilege escalation in the PodSecurityPolicy admission plugin resulting in the ability to make use of any existing PodSecurityPolicy object. | | | | | | | | | | | | | | |
| 3 | CVE-2018-1002101 | 77 | | | 2018-12-05 | 2019-04-25 | 7.5 | None | Remote | Low | Not required | Partial | Partial | Partial |
| In Kubernetes versions 1.9.0-1.9.9, 1.10.0-1.10.5, and 1.11.0-1.11.1, user input was handled insecurely while setting up volume mounts on Windows nodes, which could lead to command line argument injection. | | | | | | | | | | | | | | |
| 4 | CVE-2018-1002105 | 388 | | | 2018-12-05 | 2019-06-28 | 7.5 | None | Remote | Low | Not required | Partial | Partial | Partial |
| In all Kubernetes versions prior to v1.10.11, v1.11.5, and v1.12.3, incorrect handling of error responses to proxied upgrade requests in the kube-apiserver allowed specially crafted requests to establish a connection through the Kubernetes API server to backend servers, then send arbitrary requests over the same connection directly to the backend, authenticated with the Kubernetes API server's TLS credentials used to establish the backend connection. | | | | | | | | | | | | | | |
| 5 | CVE-2016-7075 | 295 | | Bypass | 2018-09-10 | 2018-11-16 | 6.8 | None | Remote | Medium | Not required | Partial | Partial | Partial |
| It was found that Kubernetes as used by Openshift Enterprise 3 did not correctly validate X.509 client intermediate certificate host name fields. An attacker could use this flaw to bypass authentication requirements by using a specially crafted X.509 certificate. | | | | | | | | | | | | | | |
| 6 | CVE-2019-11247 | 264 | | | 2019-08-28 | 2019-09-11 | 6.5 | None | Remote | Low | Single system | Partial | Partial | Partial |
| The Kubernetes kube-apiserver mistakenly allows access to a cluster-scoped custom resource if the request is made as if the resource were namespaced. Authorizations for the resource accessed in this manner are enforced using roles and role bindings within the namespace, meaning that a user with access only to a resource in one namespace could create, view update or delete the cluster-scoped resource (according to their namespace role privileges). Kubernetes affected versions include versions prior to 1.13.9, versions prior to 1.14.5, versions prior to 1.15.2, and versions 1.7, 1.8, 1.9, 1.10, 1.11, 1.12. | | | | | | | | | | | | | | |
| 7 | CVE-2019-11248 | 200 | | DoS +Info | 2019-08-28 | 2019-09-04 | 6.4 | None | Remote | Low | Not required | Partial | None | Partial |
| The debugging endpoint /debug/pprof is exposed over the unauthenticated Kubelet healthz port. The go pprof endpoint is exposed over the Kubelet's healthz port. This debugging endpoint can potentially leak sensitive information such as internal Kubelet memory addresses and configuration, or for limited denial of service. Versions prior to 1.15.0, 1.14.4, 1.13.8, and 1.12.10 are affected. The issue is of medium severity, but not exposed by the default configuration. | | | | | | | | | | | | | | |
| 8 | CVE-2019-11249 | 264 | | | 2019-08-28 | 2019-09-04 | 5.8 | None | Remote | Medium | Not required | None | Partial | Partial |
| The kubectl cp command allows copying files between containers and the user machine. To copy files from a container, Kubernetes runs tar inside the container to create a tar archive, copies it over the network, and kubectl unpacks it on the user?s machine. If the tar binary in the container is malicious, it could run any code and output unexpected, malicious results. An attacker could use this to write files to any path on the user?s machine when kubectl cp is called, limited only by the system permissions of the local user. Kubernetes affected versions include versions prior to 1.13.9, versions prior to 1.14.5, versions prior to 1.15.2, and versions 1.1, 1.2, 1.4, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12. | | | | | | | | | | | | | | |
| 9 | CVE-2019-1002101 | 59 | | | 2019-04-01 | 2019-08-25 | 5.8 | None | Remote | Medium | Not required | None | Partial | Partial |
| The kubectl cp command allows copying files between containers and the user machine. To copy files from a container, Kubernetes creates a tar inside the container, copies it over the network, and kubectl unpacks it on the user?s machine. If the tar binary in the container is malicious, it could run any code and output unexpected, malicious results. An attacker could use this to write files to any path on the user?s machine when kubectl cp is called, limited only by the system permissions of the local user. The untar function can both create and follow symbolic links. The issue is resolved in kubectl v1.11.9, v1.12.7, v1.13.5, and v1.14.0. | | | | | | | | | | | | | | |
| 12 | CVE-2019-11245 | 264 | | | 2019-08-28 | 2019-09-05 | 4.6 | None | Local | Low | Not required | Partial | Partial | Partial |
| In kubelet v1.13.6 and v1.14.2, containers for pods that do not specify an explicit runAsUser attempt to run as uid 0 (root) on container restart, or if the image was previously pulled to the node. If the pod specified mustRunAsNonRoot: true, the kubelet will refuse to start the container as root. If the pod did not specify mustRunAsNonRoot: true, the kubelet will run the container as uid 0. | | | | | | | | | | | | | | |

(D02)

Cloud Native Computing Foundation

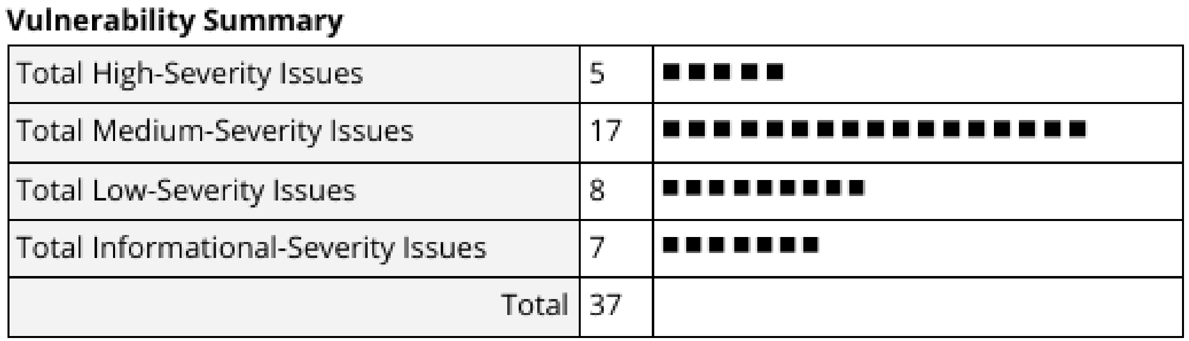
– Open Sourcing the Kubernetes Security Audit (github)

- *...managed the audit over a **four month time span**…*
- *… to complete a security assessment against Kubernetes, bearing in mind the **high complexity** and wide scope of the project*
- *… **significant room for improvement**. The **codebase is large and complex**, with large sections of code containing minimal documentation and numerous dependencies, including systems external to Kubernetes. There are **many cases of logic re-implementation** within the codebase ...*
- *… **selected** eight components …*

(D02)

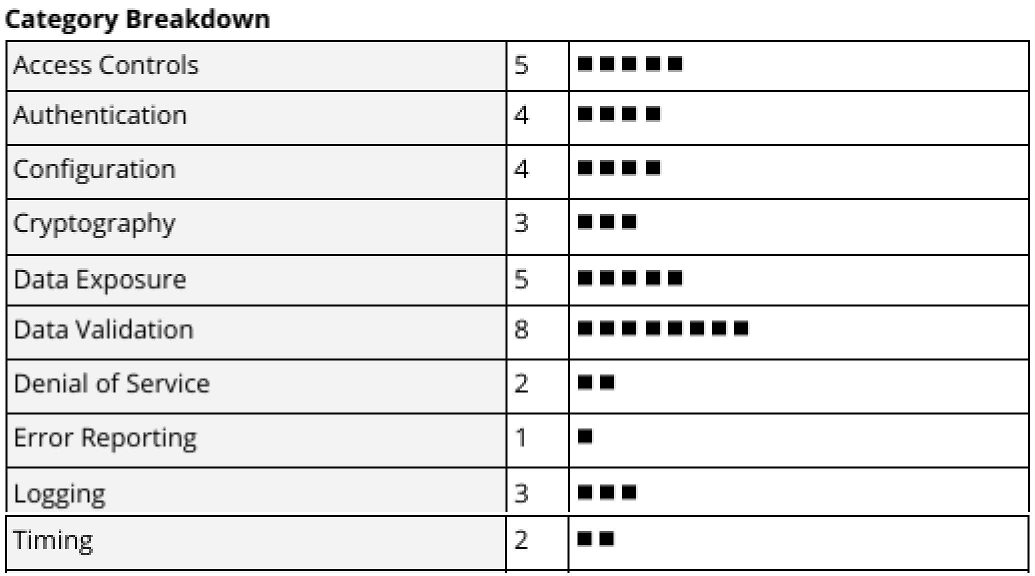
# Cloud Native Computing Foundation

- Open Sourcing the Kubernetes Security Audit (github)



**Vulnerability Summary**

| | | |
|---|---|---|
| Total High-Severity Issues | 5 | ■■■■■ |
| Total Medium-Severity Issues | 17 | ■■■■■■■■■■■■■■■■■ |
| Total Low-Severity Issues | 8 | ■■■■■■■■■ |
| Total Informational-Severity Issues | 7 | ■■■■■■■ |
| Total | 37 | |



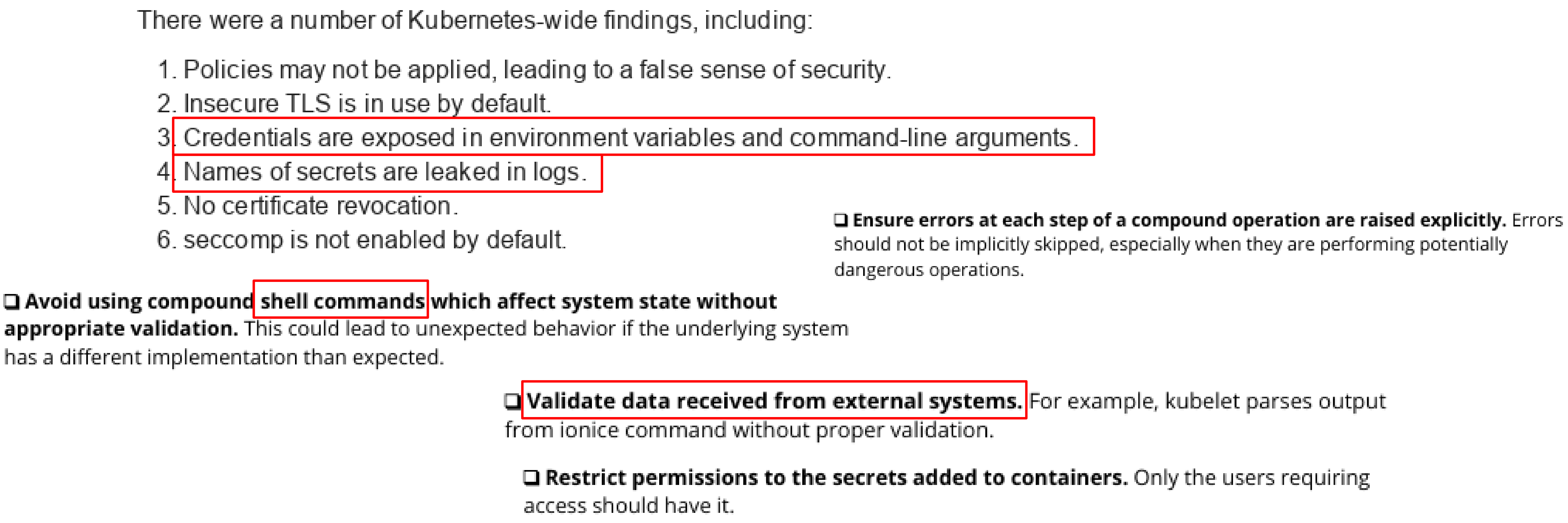
**Category Breakdown**

| | | |
|---|---|---|
| Access Controls | 5 | ■■■■■ |
| Authentication | 4 | ■■■■ |
| Configuration | 4 | ■■■■ |
| Cryptography | 3 | ■■■ |
| Data Exposure | 5 | ■■■■■ |
| Data Validation | 8 | ■■■■■■■■ |
| Denial of Service | 2 | ■■ |
| Error Reporting | 1 | ■ |
| Logging | 3 | ■■■ |
| Timing | 2 | ■■ |

(D02)

# Cloud Native Computing Foundation

– Open Sourcing the Kubernetes Security Audit (github)



There were a number of Kubernetes-wide findings, including:

1. Policies may not be applied, leading to a false sense of security.
2. Insecure TLS is in use by default.
3. Credentials are exposed in environment variables and command-line arguments.
4. Names of secrets are leaked in logs.
5. No certificate revocation.
6. seccomp is not enabled by default.

❑ **Ensure errors at each step of a compound operation are raised explicitly.** Errors should not be implicitly skipped, especially when they are performing potentially dangerous operations.

❑ **Avoid using compound shell commands which affect system state without appropriate validation.** This could lead to unexpected behavior if the underlying system has a different implementation than expected.

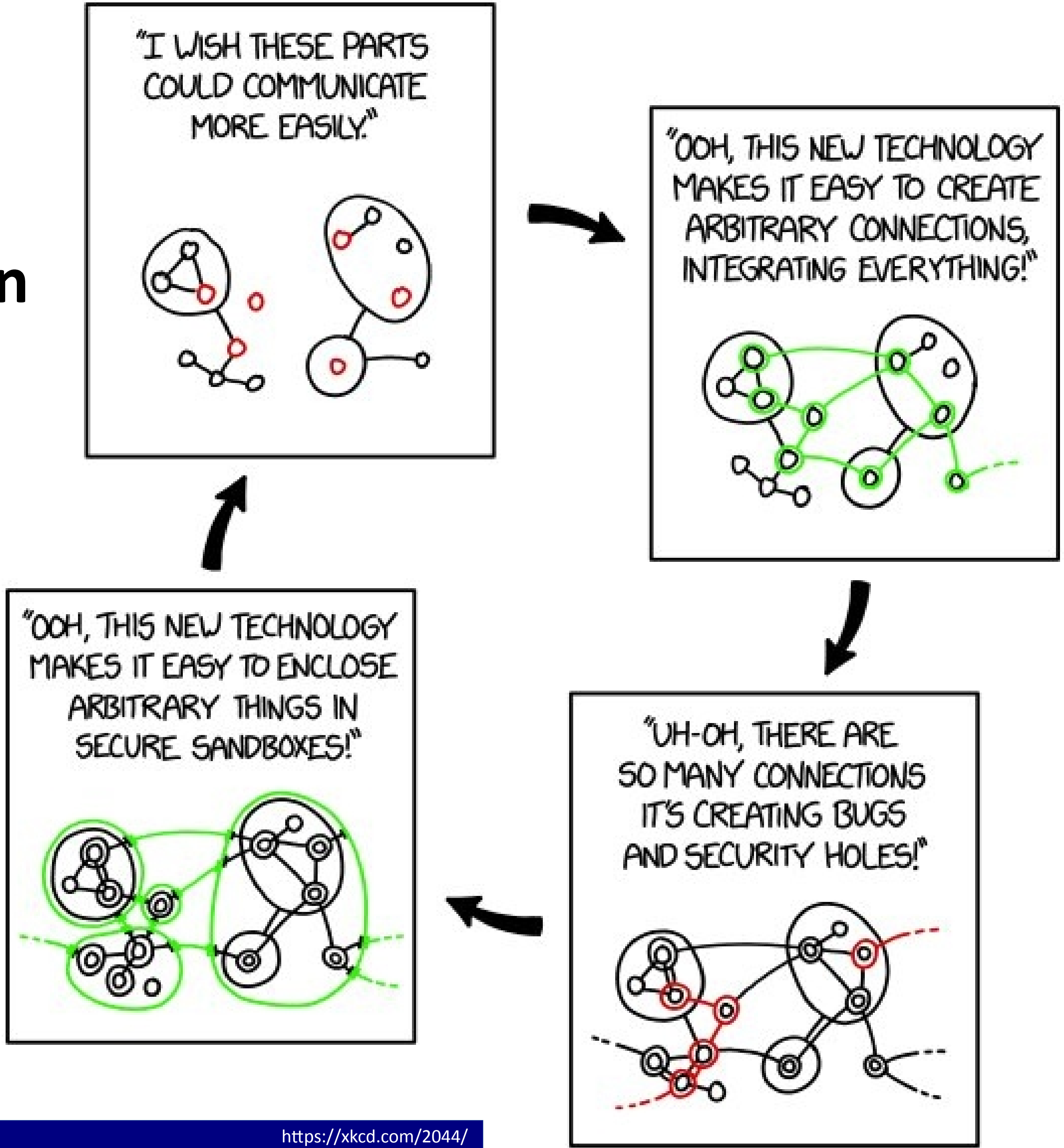
❑ **Validate data received from external systems.** For example, kubelet parses output from ionice command without proper validation.

❑ **Restrict permissions to the secrets added to containers.** Only the users requiring access should have it.

- **D02 – Patch Management Policy**
  - **Mini Distro Images**
    - Do often: Tear down & freshly deploy
    - (Best: Unit/integration testing before)

- **D02 – Patch Management Policy**
  - **Docker / Container  Software**
    - dockerd ,  docker-containerd-shim
    - libs, ...

- **D03 – Network Segmentation**
  - Basic DMZ techniques
    - Part I: Building the network


"I WISH THESE PARTS COULD COMMUNICATE MORE EASILY."
"OOH, THIS NEW TECHNOLOGY MAKES IT EASY TO CREATE ARBITRARY CONNECTIONS, INTEGRATING EVERYTHING!"
"UH-OH, THERE ARE SO MANY CONNECTIONS IT'S CREATING BUGS AND SECURITY HOLES!"
"OOH, THIS NEW TECHNOLOGY MAKES IT EASY TO ENCLOSE ARBITRARY THINGS IN SECURE SANDBOXES!"

https://xkcd.com/2044/

D03

- **D03 – Network Segmentation**

  - Depends on Network driver
    - Bridge:
      - **use different bridges / networks for segmentation**
      - **DON’T put every container into one /24**

    - **Different Tenants: never ever in one network.**
      - More later

- **D04 – Secure Defaults and Hardening**

  - 3 domains
    - Orchestration tool
    - Host
    - Container image

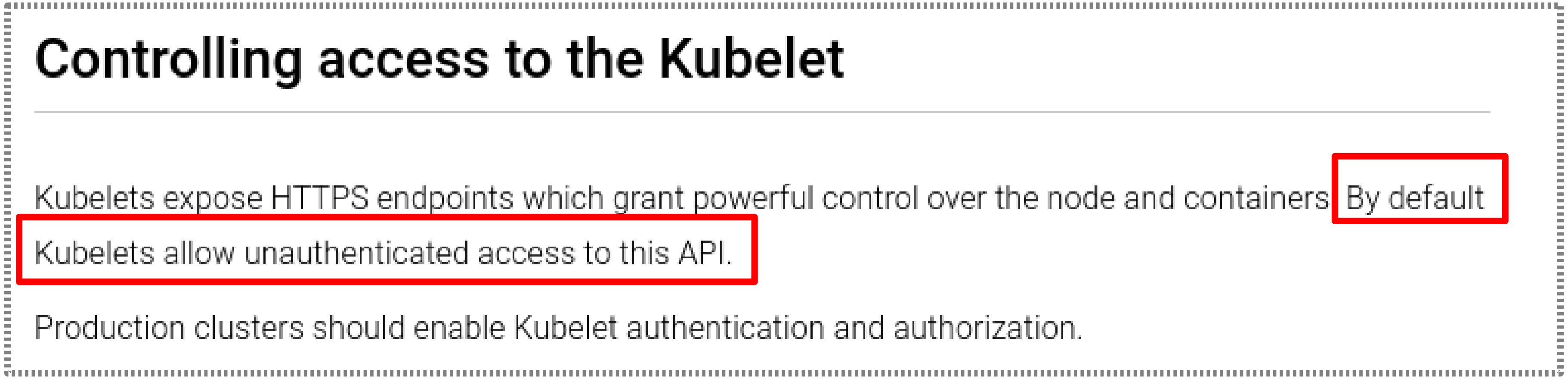
- **D04 – Secure Defaults and Hardening**

  - **Orchestration** tool’s management interfaces
    - Lock down
      - Network access
      - Interface with AuthN

    - Question secure defaults!

k8s:

- Insecure kubelet @ tcp/10250 (HTTPS) + 10255 (HTTP)



## Controlling access to the Kubelet

Kubelets expose HTTPS endpoints which grant powerful control over the node and containers. By default Kubelets allow unauthenticated access to this API.

Production clusters should enable Kubelet authentication and authorization.

- Default still open? Fixes complete?

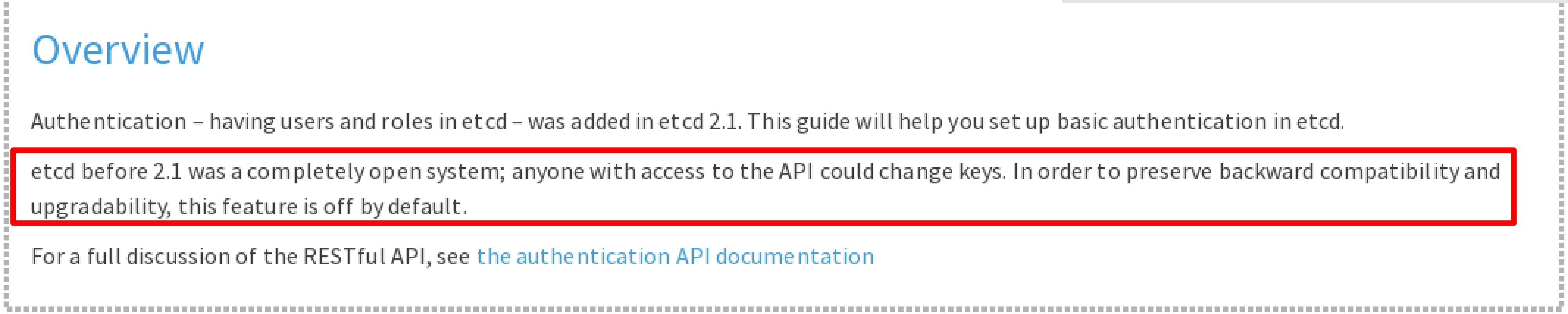
D04

# CoreOS:

- etcd @ tcp/2379

The security footgun in etcd

March 16, 2018



## Authentication Guide

### Overview

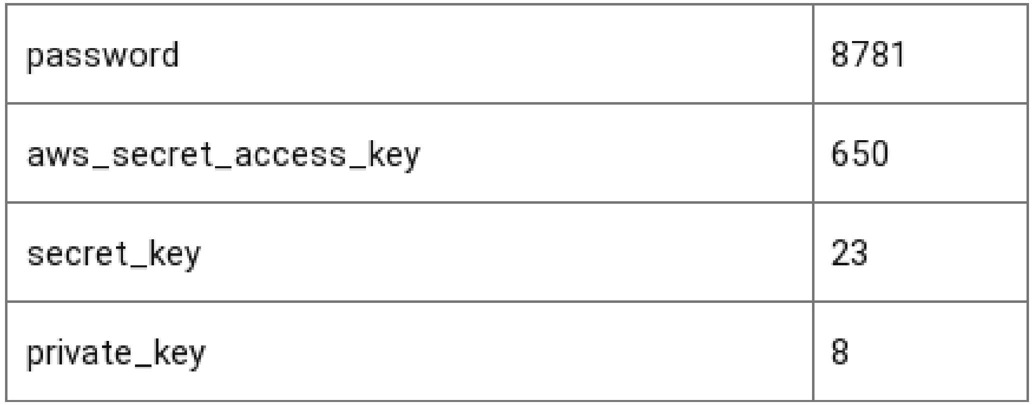
Authentication – having users and roles in etcd – was added in etcd 2.1. This guide will help you set up basic authentication in etcd.

etcd before 2.1 was a completely open system; anyone with access to the API could change keys. In order to preserve backward compatibility and upgradability, this feature is off by default.

For a full discussion of the RESTful API, see the authentication API documentation

## CoreOS:

- etcd @ tcp/2379



|  |  |
|---|---|
| password | 8781 |
| aws_secret_access_key | 650 |
| secret_key | 23 |
| private_key | 8 |

*I did a simple search on shodan and came up with 2,284 etcd servers on the open internet. So I clicked a few and on the third try I saw what I was hoping not to see. CREDENTIALS, a lot of CREDENTIALS. Credentials for things like cms_admin, mysql_root, postgres, etc.*

*[..] I wrote a very simple script that basically called the etcd API and requested all keys. That's basically equivalent to doing a database dump but over their very nice REST API.*

```
GET http://<ip address>:2379/v2/keys/?recursive=true
```

*This will return all the keys stored on the servers in JSON format. So my script basically went down the list and created a file for each IP (127-0-0-1.json) with the contents of etcd. I stopped the script at about 750 MB of data and 1,485 of the original IP list.*

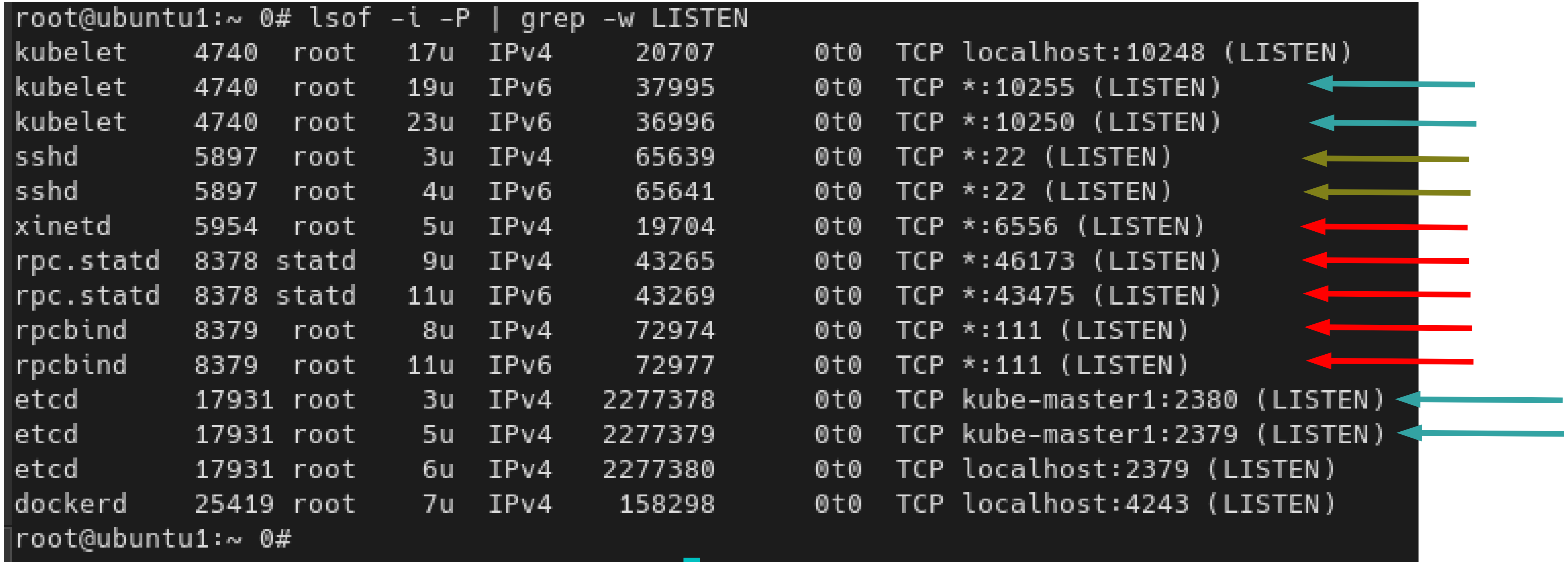
From*:* https://gcollazo.com/the-security-footgun-in-etcd/

D04

- **D04 – Secure Defaults and Hardening**
  - **Host: OS**
    - A standard Debian / Ubuntu … is a standard Debian / Ubuntu
    - Specialized container OS like
      - CoreOS (RH)
      - RancherOS
      - VMWare Photon (FLOSS!)
      - Snappy Ubuntu Core(?)
      - ...
    - Mind: Support time / EOL

- **D04 – Secure Defaults and Hardening**
  - **Host: Services**
    - Standard Distribution
      - Minimum principle, a.k.a.: Do not install useless junk
    - Also not needed:
      - Avahi
      - RPC services
      - CUPS
      - SMB / NFS

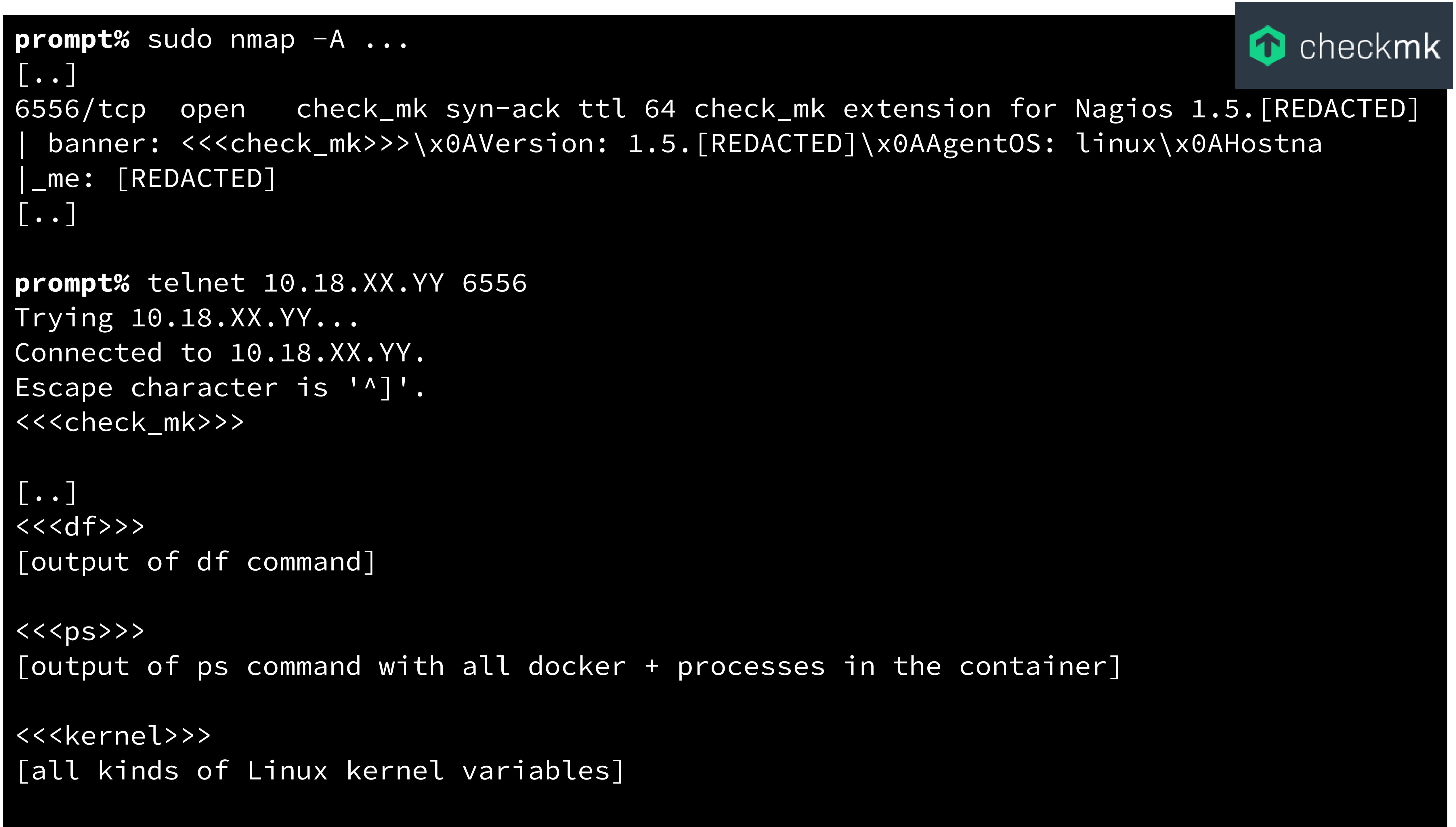
D04



```
root@ubuntu1:~ 0# lsof -i -P | grep -w LISTEN
kubelet     4740   root    17u  IPv4     20707       0t0  TCP localhost:10248 (LISTEN)
kubelet     4740   root    19u  IPv6     37995       0t0  TCP *:10255 (LISTEN)
kubelet     4740   root    23u  IPv6     36996       0t0  TCP *:10250 (LISTEN)
sshd        5897   root     3u  IPv4     65639       0t0  TCP *:22 (LISTEN)
sshd        5897   root     4u  IPv6     65641       0t0  TCP *:22 (LISTEN)
xinetd      5954   root     5u  IPv4     19704       0t0  TCP *:6556 (LISTEN)
rpc.statd   8378  statd     9u  IPv4     43265       0t0  TCP *:46173 (LISTEN)
rpc.statd   8378  statd    11u  IPv6     43269       0t0  TCP *:43475 (LISTEN)
rpcbind     8379   root     8u  IPv4     72974       0t0  TCP *:111 (LISTEN)
rpcbind     8379   root    11u  IPv6     72977       0t0  TCP *:111 (LISTEN)
etcd        17931 root      3u  IPv4   2277378       0t0  TCP kube-master1:2380 (LISTEN)
etcd        17931 root      5u  IPv4   2277379       0t0  TCP kube-master1:2379 (LISTEN)
etcd        17931 root      6u  IPv4   2277380       0t0  TCP localhost:2379 (LISTEN)
dockerd     25419 root      7u  IPv4    158298       0t0  TCP localhost:4243 (LISTEN)
root@ubuntu1:~ 0#
```

- **D04 – Secure Defaults and Hardening**
  - **Host**
    - Apply some custom hardening
      - lynis
      - CIS
    - Put all changes into your config management system!

checkmk



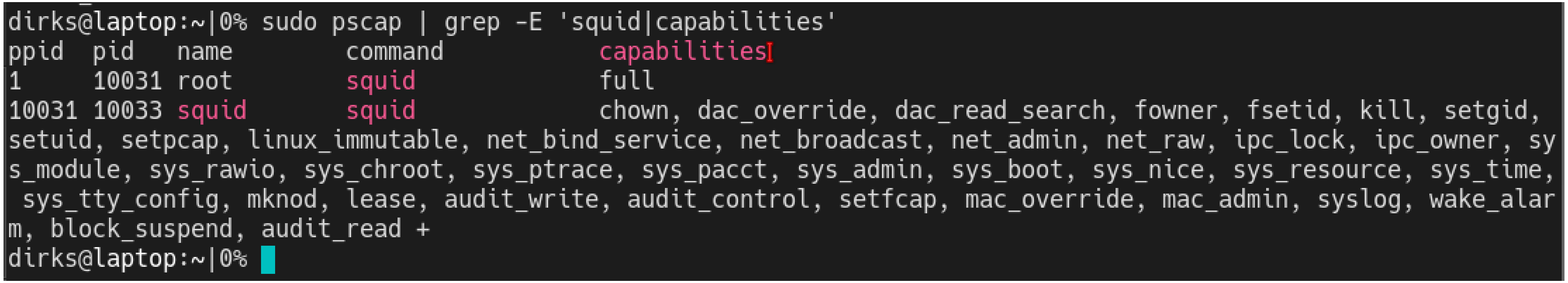
```
prompt% sudo nmap -A ...
[..]
6556/tcp  open   check_mk syn-ack ttl 64 check_mk extension for Nagios 1.5.[REDACTED]
| banner: <<<check_mk>>>\x0AVersion: 1.5.[REDACTED]\x0AAgentOS: linux\x0AHostna
|_me: [REDACTED]
[..]

prompt% telnet 10.18.XX.YY 6556
Trying 10.18.XX.YY...
Connected to 10.18.XX.YY.
Escape character is '^]'.
<<<check_mk>>>

[..]
<<<df>>>
[output of df command]

<<<ps>>>
[output of ps command with all docker + processes in the container]

<<<kernel>>>
[all kinds of Linux kernel variables]
```

- **D04 – Secure Defaults and Hardening**
  - **Container from kernel perspective (I)**
    - Controlling system calls
      - `syscalls(2)`, `syscall(2)`
      - `/usr/include/bits/syscall.h`
    - seccomp
      - **`--security-opt seccomp=`*`yourprofile.json`***

- **D04 – Secure Defaults and Hardening**

  – **Container from kernel perspective (II)**

    - Using capabilities
      - `capabilities(7)`
      - `/usr/include/linux/capability.h`

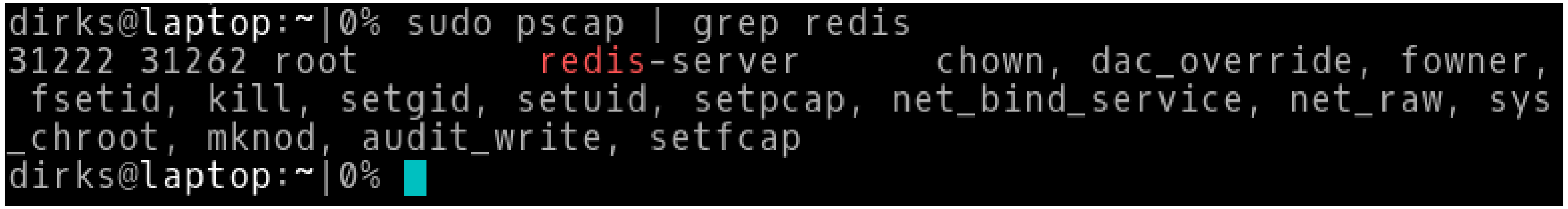


```
dirks@laptop:~|0% sudo pscap | grep -E 'squid|capabilities'
ppid  pid    name         command           capabilities
1     10031 root         squid             full
10031 10033 squid        squid             chown, dac_override, dac_read_search, fowner, fsetid, kill, setgid,
setuid, setpcap, linux_immutable, net_bind_service, net_broadcast, net_admin, net_raw, ipc_lock, ipc_owner, sy
s_module, sys_rawio, sys_chroot, sys_ptrace, sys_pacct, sys_admin, sys_boot, sys_nice, sys_resource, sys_time,
 sys_tty_config, mknod, lease, audit_write, audit_control, setfcap, mac_override, mac_admin, syslog, wake_alar
m, block_suspend, audit_read +
dirks@laptop:~|0%
```

- **D04 – Secure Defaults and Hardening**

  - **Container from kernel perspective (II)**
    - Using capabilities

      `--cap-drop`



```
dirks@laptop:~|0% sudo pscap | grep redis
31222 31262 root        redis-server       chown, dac_override, fowner,
 fsetid, kill, setgid, setuid, setpcap, net_bind_service, net_raw, sys
_chroot, mknod, audit_write, setfcap
dirks@laptop:~|0%
```

- **D04 – Secure Defaults and Hardening**
  - **Container**
    - Minimum principle
    - ~one microservice per container (but: see networking)

    - Debian / Ubuntu, comes with too much 💩
    - Better: Alpine
      - Busybox
      - But: wget / netcat “Hacker’s friends” (less 💩 )
    - Best:
      - Distroless, multistage

- **D04 – Secure Defaults and Hardening**
  - **Firewall**
    a) Last resort to protect services
    b) Good means for network boundaries

D04

- **D04 – Secure Defaults and Hardening**
  - **Firewall**
    a) Last resort to protect services

```
prompt% telnet 10.18.XX.YY 6556
Trying 10.18.XX.YY...
Connected to 10.18.XX.YY.
Escape character is '^]'.

(all dirty details follow)
```

- **D04 – Secure Defaults and Hardening**

  – **Firewall**

  a) Last resort (or additional protection) for network services

```
iptables -A INPUT -s <mgmt_IP> -d <myCHKMY_IP> -m tcp --dport 6556 -j ACCEPT
iptables -A INPUT -d <CHKMY_IP> -m tcp --dport 6556 -j LOG
iptables -A INPUT -d <CHKMY_IP> -m tcp --dport 6556 -j DROP
```

- **D04 – Secure Defaults and Hardening**

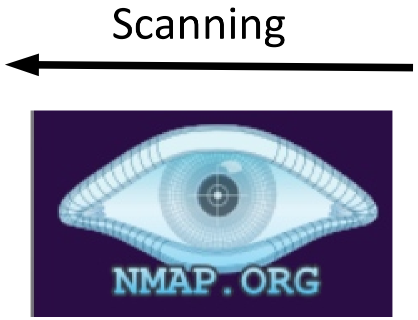
  - **Firewall**

    b) Good means for network boundaries
    - Whitelist what’s needed
    - Log everything which violates the whitelist
    - Block the rest

## D04 – Secure Defaults and Hardening

- **Verify:**
  - Did I miss any service?
  - Firewall settings

- <u>What (Baseline):</u>
  - Host
  - Orchestration

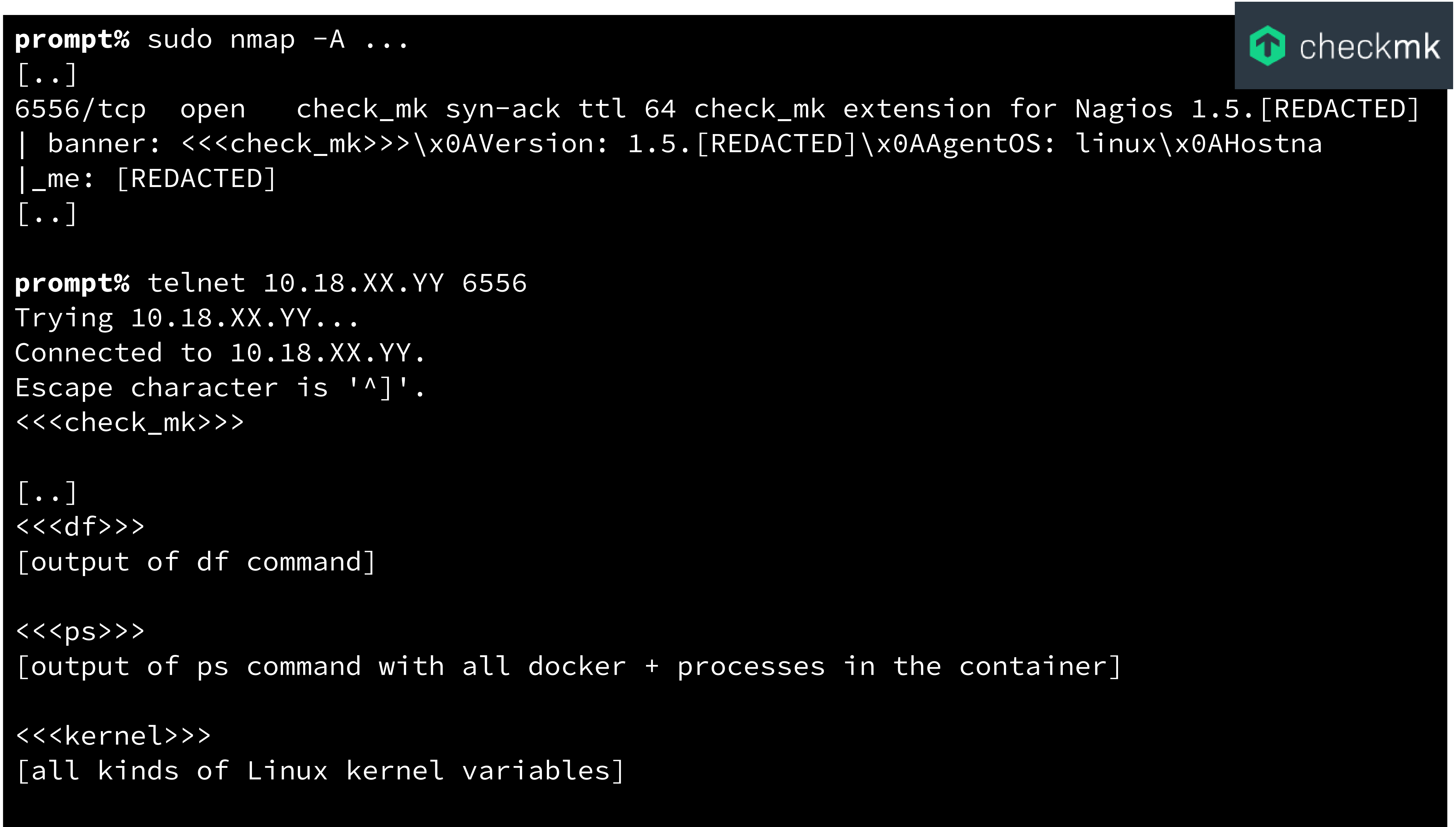


- <u>From where:</u>
  - WAN
  - Container Network
  - LAN

- **D06 – Protect Secrets**
  - Whereto: Keys, certificates, credentials, etc ???
    - Image ??
    - Env variables?

      -
        ```
        docker run -e SECRET=myprrecious <containerID>
        ```
      - **Careful!**

      - All processes in this container inherit `$SECRET` && know *`myprrecious`*



```
prompt% sudo nmap -A ...
[..]
6556/tcp  open   check_mk syn-ack ttl 64 check_mk extension for Nagios 1.5.[REDACTED]
| banner: <<<check_mk>>>\x0AVersion: 1.5.[REDACTED]\x0AAgentOS: linux\x0AHostna
|_me: [REDACTED]
[..]

prompt% telnet 10.18.XX.YY 6556
Trying 10.18.XX.YY...
Connected to 10.18.XX.YY.
Escape character is '^]'.
<<<check_mk>>>

[..]
<<<df>>>
[output of df command]

<<<ps>>>
[output of ps command with all docker + processes in the container]

<<<kernel>>>
[all kinds of Linux kernel variables]
```

```
<<<docker_containers:sep(XX)>>>
(more detailed info about containers and their processes)

<<<docker_node_images>>>
[[[images]]]

[[[image_inspect]]]
[
    {
        "Id": "sha256:      7d788a125269edce5e71f643….
[..]
     "Env": [
                "PATH=/usr/local/bin:/usr/bin:/sbin:/bin",
                "SLAPD_SUFFIX=dc=******,dc=***",
                "SLAPD_PASSWORD=********",
                "SLAPD_CONFIG_PASSWORD=*******"
```

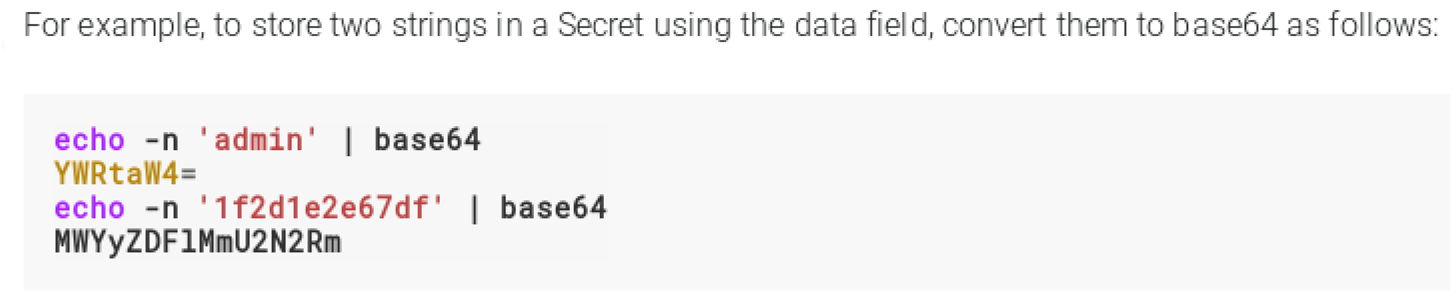
- **D06 – Protect Secrets**
  - Whereto: Keys, certificates, credentials, etc ???
    - Image ??
    - Env variables?
      - `docker run -e SECRET=`*`myprrecious`*` ID`
      - **Careful!  check_mk example + grepping equals to**

```
for c in $(docker ps -q); do
   docker inspect $c | grep PASS
done
```

→ LDAP_PASSWORD, SLAPD_PASSWORD,
→ MONGO_PASSWORD*, POSTGRESQL_PASS*
→ FTP_PASSWORD,
→ SPRING_PASS*,
→ JWT_HMAC*
→ ...

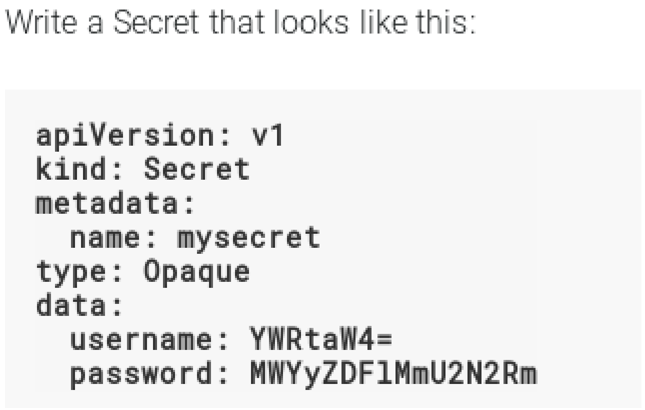
D06

- **D06 – Protect Secrets**
  - Whereto: Keys, certificates, credentials, etc ???
    - Image ??
    - Env variables?
      - `docker run -e SECRET=`*`myprrecious`*` ID`
      - Pointer
        - `docker run -env-file ./secretsfile.txt `*`ID`*
      - Kubernetes + YAML secrets: be careful too



For example, to store two strings in a Secret using the data field, convert them to base64 as follows:

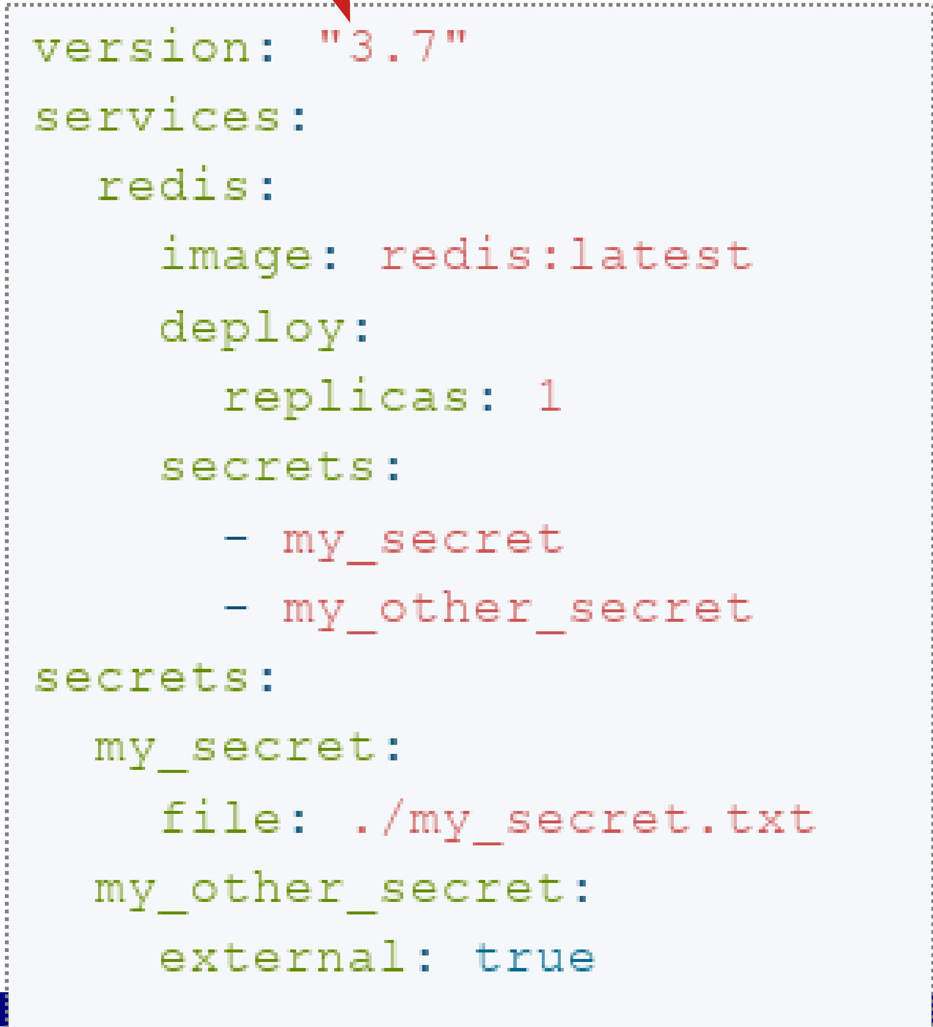
```
echo -n 'admin' | base64
YWRtaW4=
echo -n '1f2d1e2e67df' | base64
MWYyZDFlMmU2N2Rm
```



Write a Secret that looks like this:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
```

- **D06 – Protect Secrets**
  - Whereto: Keys, certificates, credentials, etc ???
    - Image ??
    - Env variables?
      - `docker run -e SECRET=`*`myprrecious ID`*
      - Pointer: as bad
      - Kubernetes + YAML secrets: be careful too
    - mounts
      - Secret mounts (formerly swarm only)
        - `/run/secrets`
        - similar k8



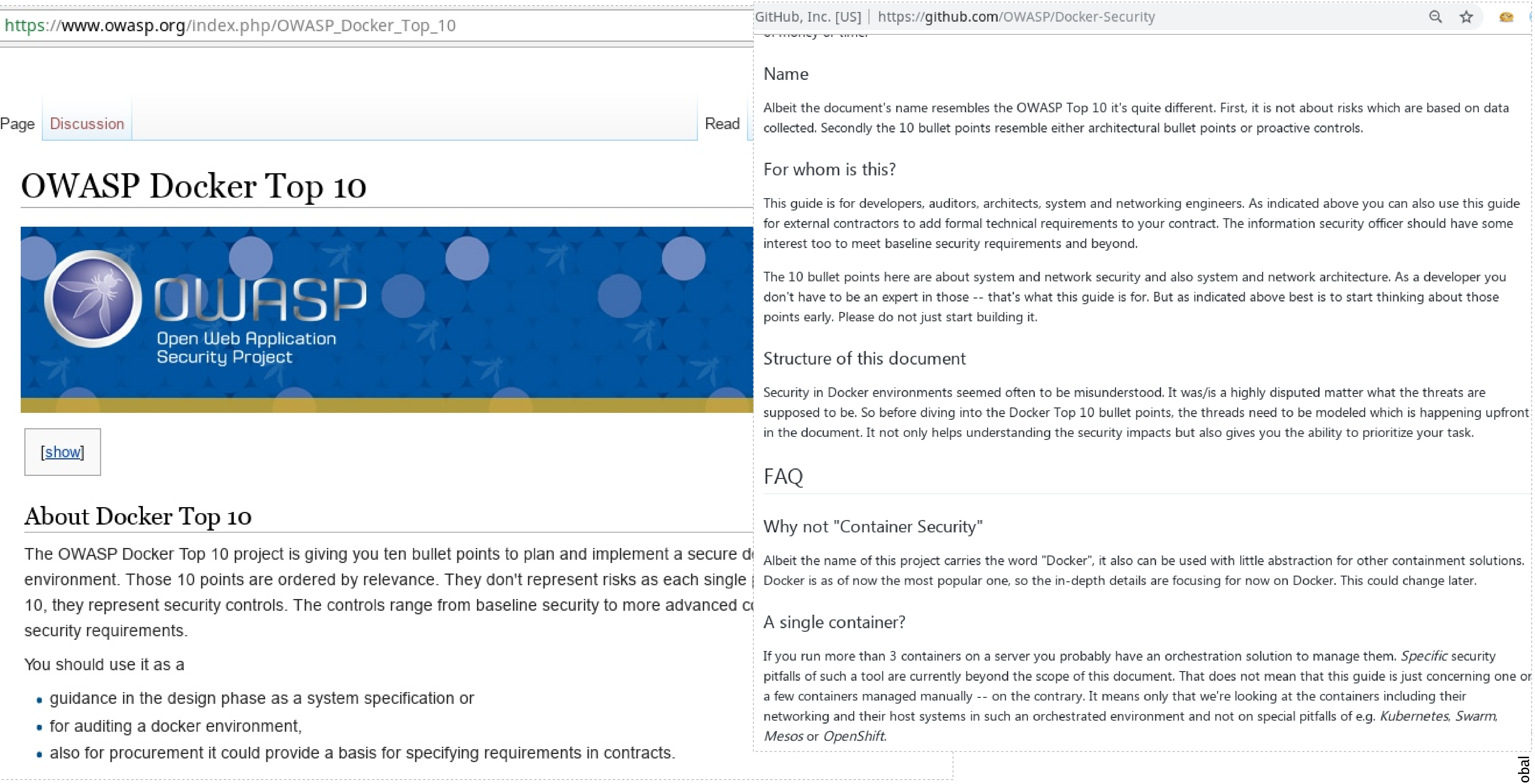
```
version: "3.7"
services:
  redis:
    image: redis:latest
    deploy:
      replicas: 1
    secrets:
      - my_secret
      - my_other_secret
secrets:
  my_secret:
    file: ./my_secret.txt
  my_other_secret:
    external: true
```

# What’s Next

- **Managers**
  - Ressources
    - Skills
      - Education needed?
    - Budget
      - External/internal Manpower needed?
  - CISO:
    - Patchmanagement / Monitoring of it
    - Network architecture?
    - Do I always have the security status? (scanners)

# What’s Next

- **Developers / Operation: Scan yourself**
  - Net: Nmapping
  - Host:
    - Lynis
    - Vuln. Scanner
    - Docker CIS benchmark
      - https://github.com/docker/docker-bench-security
    - `docker inspect / network inspect`
  - Images: Image Vulnerability Scanners



https://www.owasp.org/index.php/OWASP_Docker_Top_10

# OWASP Docker Top 10

## About Docker Top 10

The OWASP Docker Top 10 project is giving you ten bullet points to plan and implement a secure d
environment. Those 10 points are ordered by relevance. They don't represent risks as each single
10, they represent security controls. The controls range from baseline security to more advanced c
security requirements.

You should use it as a

- guidance in the design phase as a system specification or
- for auditing a docker environment,
- also for procurement it could provide a basis for specifying requirements in contracts.

GitHub, Inc. [US] | https://github.com/OWASP/Docker-Security

of money or time.

### Name

Albeit the document's name resembles the OWASP Top 10 it's quite different. First, it is not about risks which are based on data collected. Secondly the 10 bullet points resemble either architectural bullet points or proactive controls.

### For whom is this?

This guide is for developers, auditors, architects, system and networking engineers. As indicated above you can also use this guide for external contractors to add formal technical requirements to your contract. The information security officer should have some interest too to meet baseline security requirements and beyond.

The 10 bullet points here are about system and network security and also system and network architecture. As a developer you don't have to be an expert in those -- that's what this guide is for. But as indicated above best is to start thinking about those points early. Please do not just start building it.

### Structure of this document

Security in Docker environments seemed often to be misunderstood. It was/is a highly disputed matter what the threats are supposed to be. So before diving into the Docker Top 10 bullet points, the threads need to be modeled which is happening upfront in the document. It not only helps understanding the security impacts but also gives you the ability to prioritize your task.

## FAQ

### Why not "Container Security"

Albeit the name of this project carries the word "Docker", it also can be used with little abstraction for other containment solutions. Docker is as of now the most popular one, so the in-depth details are focusing for now on Docker. This could change later.

### A single container?

If you run more than 3 containers on a server you probably have an orchestration solution to manage them. *Specific* security pitfalls of such a tool are currently beyond the scope of this document. That does not mean that this guide is just concerning one or a few containers managed manually -- on the contrary. It means only that we're looking at the containers including their networking and their host systems in such an orchestrated environment and not on special pitfalls of e.g. *Kubernetes*, *Swarm*, *Mesos* or *OpenShift*.

# about:end

## Thank you!

@drwetter

dirk@owasp.org
mail@drwetter.eu

OWASP

### OWASP Docker Top 10

The Then Most Important Aspects To Build a Secure Containerized Environment.