condignum

Monitoring, attack detection and analysis options on container environments at runtime

Johannes Bär

Table of contents

- About us / about me
- Containers and their advantages for runtime analysis
- Container monitoring tools and their features
- Protection goals
- Critical reflection on tools and their potential
- Interesting use / What is recommended
- From event to evaluation / usage example
- Edge Cases / Known Issues / Limitations

condignum at a glance

Founded in Austria in 2019.

Focus on innovative IT security services..

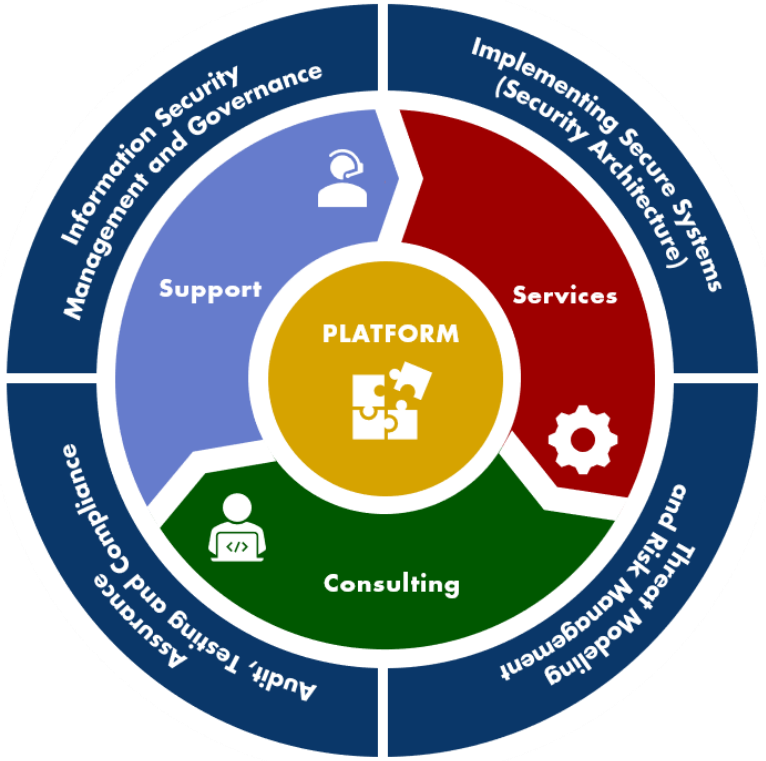
- Cooperation
  - With corporations in the DACH-region
  - Varieties of industry sectors
- Cooperation with research facilites

Numbers

- 20+ team members
- Represented in Austria and Germany

Products and services

- condignum SaaS platform, condignum PentestVM
- IT-Security Consulting
- Managed Security Services



Über mich

- Johannes Bär
- Team Lead Professional Services @condignum GmbH
- Security analyzes and penetration tests

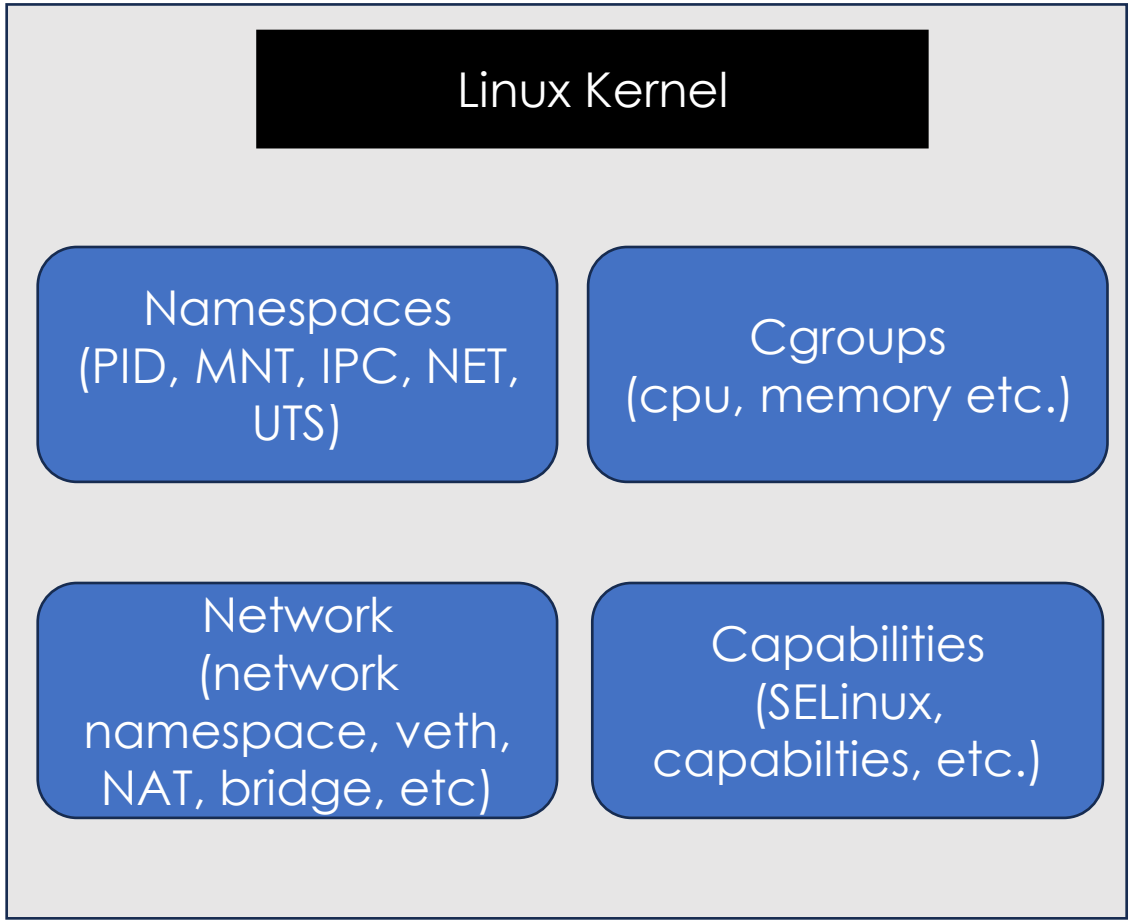
- Kubernetes research since 2019
- Supporting customers in the secure construction and operation of container environments
- Current: Analyzes and tests of Kubernetes “add-ons”

Container und deren Vorteile für Laufzeitanalysen

- Linux containers are **an isolation approach**
- Isolation is achieved using, among other things, the **following properties**:

In short:

- Namespaces e.g. for **PID, IPC, NET, cgroup, mount**
- **Linux Capabilties (SELinux, dropping of capabilites)**
- Chroot -> Copy-on-write file system
- And some other behaviours…



Container-Namespaces

```
vagrant@vagrant:~/vuln_app/vuln_app/hello-python/kubernetes$ kubectl apply -f deployment.yaml
service/super-sichere-flask-app-service created
deployment.apps/super-sichere-flask-app created
vagrant@vagrant:~/vuln_app/vuln_app/hello-python/kubernetes$ kubectl get pods
NAME                                      READY   STATUS    RESTARTS   AGE
super-sichere-flask-app-76ddc6bc67-l5sn2   1/1     Running   0          14s
vagrant@vagrant:~/vuln_app/vuln_app/hello-python/kubernetes$ ps aux | grep -i main.py
root      136327  3.3  0.3  34804 29540 ?        Ss   18:20   0:00 python /app/main.py
vagrant   136560  0.0  0.0   6432   656 pts/0    R+   18:20   0:00 grep --color=auto -i main.py
vagrant@vagrant:~/vuln_app/vuln_app/hello-python/kubernetes$
```


Container-Namespaces

```
vagrant@vagrant:~/vuln_app/vuln_app/hello-python/kubernetes$ ps aux | grep -i main.py
root      136327  3.3  0.3  34804 29540 ?        Ss   18:20   0:00 python /app/main.py
vagrant   136560  0.0  0.0   6432   656 pts/0    R+   18:20   0:00 grep --color=auto -i main.py
(failed reverse-i-search)`': ^C aux | grep -i main.py
vagrant@vagrant:~/vuln_app/vuln_app/hello-python/kubernetes$ ls -la /proc/136327/ns/
ls: cannot open directory '/proc/136327/ns/': Permission denied
vagrant@vagrant:~/vuln_app/vuln_app/hello-python/kubernetes$ sudo !!
sudo ls -la /proc/136327/ns/
total 0
dr-x--x--x 2 root root 0 Sep 11 18:22 .
dr-xr-xr-x 9 root root 0 Sep 11 18:20 ..
lrwxrwxrwx 1 root root 0 Sep 11 18:22 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Sep 11 18:22 ipc -> 'ipc:[4026533097]'
lrwxrwxrwx 1 root root 0 Sep 11 18:22 mnt -> 'mnt:[4026533103]'
lrwxrwxrwx 1 root root 0 Sep 11 18:22 net -> 'net:[4026532578]'
lrwxrwxrwx 1 root root 0 Sep 11 18:22 pid -> 'pid:[4026533104]'
lrwxrwxrwx 1 root root 0 Sep 11 18:22 pid_for_children -> 'pid:[4026533104]'
lrwxrwxrwx 1 root root 0 Sep 11 18:22 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Sep 11 18:22 uts -> 'uts:[4026533096]'
vagrant@vagrant:~/vuln_app/vuln_app/hello-python/kubernetes$
```


Container-Namespaces

```
vagrant@vagrant:~/vuln_app/vuln_app/hello-python/kubernetes$ kubectl exec --stdin --tty super-sichere-flask-app-76ddc6bc
67-l5sn2 -- /bin/bash
root@super-sichere-flask-app-76ddc6bc67-l5sn2:/app# id
uid=0(root) gid=0(root) groups=0(root)
root@super-sichere-flask-app-76ddc6bc67-l5sn2:/app# whoami
root
root@super-sichere-flask-app-76ddc6bc67-l5sn2:/app# ls -la
total 20
drwxr-xr-x 1 root root 4096 Sep  7 12:40 .
drwxr-xr-x 1 root root 4096 Sep 11 18:20 ..
-rw-r--r-- 1 root root  137 Sep 21  2022 Dockerfile
-rw-r--r-- 1 root root  647 Sep 21  2022 main.py
-rw-r--r-- 1 root root    5 Sep 21  2022 requirements.txt
root@super-sichere-flask-app-76ddc6bc67-l5sn2:/app#
```


Container-Namespaces

```
vagrant@vagrant:~$ ps aux | grep -i main.py
root      136327  0.1  0.3  34804 29540 ?        Ss   18:20   0:00 python /app/main.py
vagrant   140532  0.0  0.0   6432   656 pts/2    S+   18:26   0:00 grep --color=auto -i main.py
vagrant@vagrant:~$ ps aux | grep -i bash
vagrant     6149  0.0  0.0   9180  5836 pts/0    Ss   13:13   0:00 -bash
vagrant     6155  0.0  0.0   8916  5208 pts/1    Ss+  13:13   0:00 -bash
vagrant    13251  0.0  0.0   8916  5800 pts/2    Ss   13:19   0:00 -bash
vagrant   139215  0.7  0.5 4956000 45156 pts/0   Sl+  18:24   0:00 kubectl exec --stdin --tty super-sichere-flask-app-76
ddc6bc67-l5sn2 -- /bin/bash
root      139250  0.0  0.0   4600  3620 pts/0    Ss+  18:24   0:00 /bin/bash
vagrant   140585  0.0  0.0   6432   720 pts/2    S+   18:26   0:00 grep --color=auto -i bash
vagrant@vagrant:~$
```


Container-Namespaces

```
vagrant@vagrant:~$ ps aux | grep -i main.py
root      136327  0.1  0.3  34804 29540 ?        Ss   18:20   0:00 python /app/main.py
vagrant   140532  0.0  0.0   6432   656 pts/2    S+   18:26   0:00 grep --color=auto -i main.py
vagrant@vagrant:~$ ps aux | grep -i bash
vagrant     6149  0.0  0.0   9180  5836 pts/0    Ss   13:13   0:00 -bash
vagrant     6155  0.0  0.0   8916  5208 pts/1    Ss+  13:13   0:00 -bash
vagrant    13251  0.0  0.0   8916  5800 pts/2    Ss   13:19   0:00 -bash
vagrant   139215  0.7  0.5 4956000 45156 pts/0   Sl+  18:24   0:00 kubectl exec --stdin --tty super-sichere-flask-app-76dd
c6bc67-l5sn2 -- /bin/bash
root      139250  0.0  0.0   4600  3620 pts/0    Ss+  18:24   0:00 /bin/bash
vagrant   140585  0.0  0.0   6432   720 pts/2    S+   18:26   0:00 grep --color=auto -i bash
vagrant@vagrant:~$ sudo ls -la /proc/136327/ns/
total 0
dr-x--x--x 2 root root 0 Sep 11 18:22 .
dr-xr-xr-x 9 root root 0 Sep 11 18:20 ..
lrwxrwxrwx 1 root root 0 Sep 11 18:22 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Sep 11 18:22 ipc -> 'ipc:[4026533097]'
lrwxrwxrwx 1 root root 0 Sep 11 18:22 mnt -> 'mnt:[4026533103]'
lrwxrwxrwx 1 root root 0 Sep 11 18:22 net -> 'net:[4026532578]'
lrwxrwxrwx 1 root root 0 Sep 11 18:22 pid -> 'pid:[4026533104]'
lrwxrwxrwx 1 root root 0 Sep 11 18:22 pid_for_children -> 'pid:[4026533104]'
lrwxrwxrwx 1 root root 0 Sep 11 18:22 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Sep 11 18:22 uts -> 'uts:[4026533096]'
vagrant@vagrant:~$ sudo ls -la /proc/139250/ns/
total 0
dr-x--x--x 2 root root 0 Sep 11 18:28 .
dr-xr-xr-x 9 root root 0 Sep 11 18:24 ..
lrwxrwxrwx 1 root root 0 Sep 11 18:28 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Sep 11 18:28 ipc -> 'ipc:[4026533097]'
lrwxrwxrwx 1 root root 0 Sep 11 18:28 mnt -> 'mnt:[4026533103]'
lrwxrwxrwx 1 root root 0 Sep 11 18:28 net -> 'net:[4026532578]'
lrwxrwxrwx 1 root root 0 Sep 11 18:28 pid -> 'pid:[4026533104]'
lrwxrwxrwx 1 root root 0 Sep 11 18:28 pid_for_children -> 'pid:[4026533104]'
lrwxrwxrwx 1 root root 0 Sep 11 18:28 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Sep 11 18:28 uts -> 'uts:[4026533096]'
vagrant@vagrant:~$ 139250
```


Containers and their advantages for runtime analysis

✓ Application in the containers run isolated

✓ Easier and more efficient determination of container behavior (processes, file system accesses, network connections)

vs.

X Non-containerized processes are difficult to view in isolation - unusual behavior is harder to detect

Container monitoring tools

Interim conclusion:

- **Advantages** and new possibilities for monitoring through containerization **are now well known and widespread**

- Many manufacturers offer tools with **comparable monitoring/runtime analysis capabilities:**
  - Palo Alto with Prisma Cloud (former Twistlock)
  - Sysdig Secure
  - Aqua Security
  - Wiz
  - SuSE with NeuVector

The following will be more about the **idea and the process** than about the tools

Tool example: NeuVector

- **Open Source (Apache 2.0)/** /bought by SuSE
- Written in **Go**
- Tightly coupled with **Kubernetes**
- privileged container running in every kubernetes node
- Injects a „probe“ (runs **eBPF programs**), **kernelspace**
- Receives **metadata** to running container/pods using the kubernetes API
- Uses this information to obtain **all runtime information of the application in the containers**

Tool example: Falco

- **Open Source (Apache 2.0)/** Sysdig
- Developed in C++ (performance reasons)
- **eBPF-Probe** or **kernel module** monitor containers (Syscall based monitoring)
- **Monitoring** rules are also contributed **by the community**
- Rules categorized according to the **MITRE Attack Framework**, among others
- Broad range of possible uses:
  - Container Standalone / Kubernetes
  - Community extensions

Relevanteste Features beider Tools

- ✓ **Container-aware/Kubernetes**
- ✓ Benefit from the **advantages of containerized application deployments**
- ✓ Offer various options for detecting
  - **Process anomalies**
  - **Network anomalies**
  - **File system access anomalies**

X These features are **no longer secrets** - most vendors/open source tools use **similar techniques/approaches** to detect anomalies!

Other features

- Web Application Firewalls
- Intrusion Detection/Prevention
- Network Monitoring
- Container Scanning (Vulnerabilities)
- Compliance (CIS Benchmarks, Hardening Guides
- etc.

- **This talk will be less about them**

Protection goals

What can be achieved with these tools, among other things?

✓ **Detect attacker movements after a vulnerability has been successfully exploited (for whatever reason)**

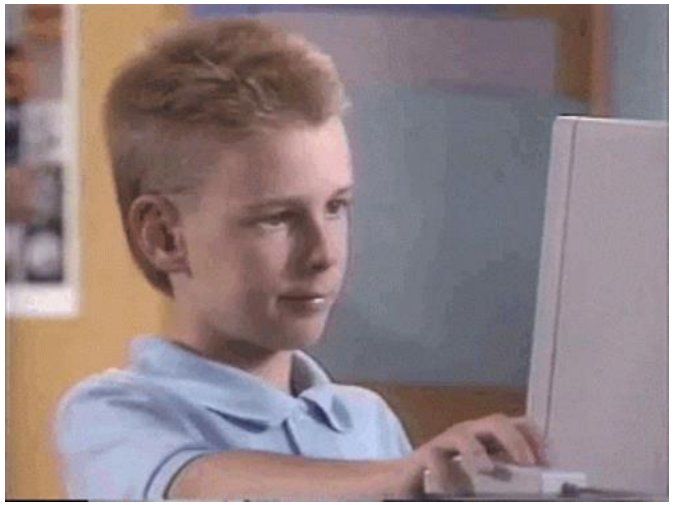
X Threat scenarios:

- 0-DaysPatch
- installation forgotten/overlooked
- Vulnerability somewhere in the dependencies
- Known/unknown vulnerability in proprietary applications

Demo

Was ist zu empfehlen?

- ✓ Start **suspicious processes** (self-defined or template)
- ✓ Dropping **new executable programs**
- ✓ Attempt to **manipulate executable files or libraries**
- ✓ Starting **reverse shells**
- ✓ Other misuse of **on-board tools**



Was ist zu empfehlen?

✓ Erstellen eines Prozessprofils

- **Welche Prozesse werden** in meinem Container **legitimerweise ausgeführt?**
- Erlaubt Abweichungen (in Form unbekannter Prozesse) zu erkennen
- **Benötigt gute Lernphasen** über Zeitraum X
  - Übergang in ein Reporting oder Block-Modus im Produktivbetrieb
- Benötigt **häufig Kooperation mit den Devs/DevOps-Teams**

Was ist zu empfehlen?

✓ Erstellen eines Netzwerkprofils:

- In der **Theorie machbar**
- **In der Praxis häufig schwer umsetzbar**
  - Netzwerkverbindungen werden häufiger und unberechenbarer geöffnet als Prozesse
  - **Sauberes Anwendungsverhalten** von Nöten
- Muss **individuell verprobt** werden!

Besonders effektiv, wenn…

- **Minimalistisches Base-Image** im Einsatz (wenig Bypass-Möglichkeiten)
  - Kaum Onboard-Tools
  - Grundgehärtetes Linux
- **Geringstmögliche Privilegien** (erzwungen mittels Kubernetes)
  - **PSA/PSP geringprivilegiert**
  - **ReadOnlyRootFileSystem**
  - **Kein ServiceAccount-Token** gemountet per Default
  - etc.
- Saubere, gut getestetes **Anwendungsverhalten**
  - Je **weniger Prozess- und Netzwerkartefakte**, desto einfacher sind die Container zu überwachen.

Kritische Reflexion

Wie immer: **Tools sind keine „Silver Bullets“!**

Immer bedenken:

- Kommerzielle Container-Monitoringlösungen bieten **auf dem Papier sehr viele Features** (WAF, Intrusion Detection, Netzwerkmonitoring etc.)
- Ins Bewusstsein rufen, **was man erreichen will**/gegen **was man sich schützen will**
- Ins Bewusstsein rufen, **welche Features bereits anderweitig abgedeckt** werden (andere Security-Tools etc.)
- Welche Verwendung läuft auf **wieviel Kooperation zwischen Entwicklern und Security-Verantwortlichen** hinaus? Wie hoch ist der **Orga-Overhead bei der Verwendung des Tools**?

Vom Event zur Auswertung / Nutzungsbeispiel

- **Relevantesten Events festlegen**, die gemonitored werden sollen, z.B.:
  - **verdächtige Prozesse/Prozessverhalten**
  - **Illegitime Dateisystemzugriffe**

- Erstellen eines **Prozessprofils**
  - Herausfinden, wie lange die **Lernphase** sein muss, um die **False-Positive-Rate gering zu halten**

- Monitoring-Tool konfigurieren, **welche Events erfasst werden sollen** (wenn möglich)

Vom Event zur Auswertung / Nutzungsbeispiel

- **Testphase**, in der das **Verhalten der Deployments** beobachtet wird.
- **Feedback-Loop**, bis False Positives auf ein Minimum reduziert sind.
- Evtl. Events (gefiltert oder alle) an ein **zentrales System weiterleiten** (z.B. SIEM).

Vom Event zur Auswertung / Nutzungsbeispiel

- **Prozess definieren**, wie auf ein **Event reagiert werden soll.**

- **Beispiel** für ein eher **kritisches Event** (verdächtiger Prozess in Container gestartet):
  - **Alarmierung** über E-Mail/Ticket/etc.
  - **Kontakt zum betroffenen Team aufnehmen** (automatisiert/manuell)
  - **„Pre-Forensics“** (Snapshot des Containers über Kubernetes-API oder über Monitoring-Tool)
  - **Netzwerkmitschnitt** starten
  - **False Positive ausräumen** (was meistens der Fall sein wird)
  - **Falls kein false Positive: Quarantäne** des Containers/Fehlersuche/Forensik etc.

Edge Cases / bekannte Probleme / Grenzen

X Bypasses existieren:

- Missbrauch von Onboard-Tools (auf dem Image vorhanden)
- „**Living off the land**“-Ansatz kann **Erkennung erschweren** (die Summe aller Erkennungstechnologien erhöht Chancen auf Erkennung)
- Je **üblicher der Einsatz von Monitoring-Tools** werden, desto üblicher wird der Einsatz von Bypasses per Default (aktuell noch kaum sichtbar)

X Noch wenig „in the wild“ sichtbar, weil Container-Monitoringlösungen wenig auf dem Radar der Angreifer sind

- Fairerweise: aktuell weniger ausgeklügelte Angriffe auf Apps in Container-Umgebungen sichtbar als auf Windows-Umgebungen

X Häufig Performance-Penalties (5-20%, jedoch schwer vorherzusehen)

- Muss individuell verprobt werden!

Vielen Dank für Ihre Aufmerksamkeit!