Tesseract : a proposal for a container breakout tool

Théo PIRKL

University of Geneva

February 9, 2024

Introduction

Nowadays

Everyone is running some form of service, from web servers, VPNs, databases, and so on.

The constant rise of new services are encouraging the industry to leave monolithic applications for new technologies, where each module of a service is isolated from the other.

Main idea : transitioning from one big “blob” to bricks, each having a single purpose.

Technologies

Two main technologies are available :

- VMs
- Containers

VMs versus containers

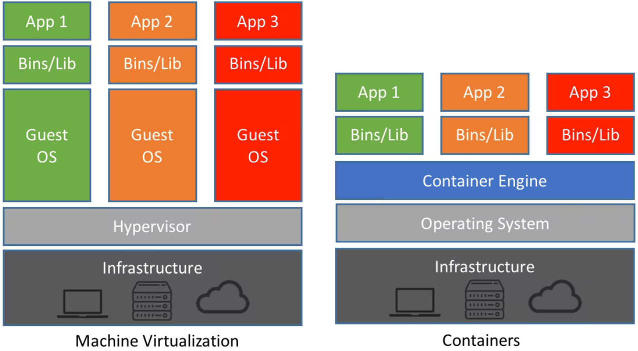
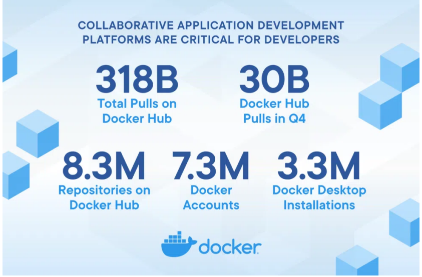


Figure 1: VMs and containers. References at the end

What are containers?

- A small isolated group of resources
- A light way of running an app or an OS
- From the user perspective, just like a VM
- Most containers can be run in a few milliseconds

Container popularity



COLLABORATIVE APPLICATION DEVELOPMENT PLATFORMS ARE CRITICAL FOR DEVELOPERS

318B Total Pulls on Docker Hub

30B Docker Hub Pulls in Q4

8.3M Repositories on Docker Hub

7.3M Docker Accounts

3.3M Docker Desktop Installations

docker

Figure 2: Container 2021 stats. References at the end

"To begin, there has now been a total of 318 billion all time pulls on Docker Hub, an increase of 145% year-over-year. That's right, the total number of pulls has increased by nearly 1.5x in the past year."

Actors using containers

- Swiss Post (e-Voting)
- CERN
- Microsoft
- Paypal
- Amazon

But why?

```
/usr/libexec/qemu-kvm -name vm-f16-buildmachine -S -M rhel6.4.0 -cpu
 Westmere -enable-kvm -m 2048 -smp 2,sockets=2,cores=1,threads=1 -uuid
 a8ccdb60-8a42-44f5-9669-d74c3b2eff43 -smbios type=1,manufacturer=Red Hat,
serial=30353036-3837-4247-3831-30394635324C_78:e7:d1:22:46:d8,
uuid=a8ccdb60-8a42-44f5-9669-d74c3b2eff43 -nodefconfig -nodefaults
-chardev socket,id=charmonitor,path=/var/lib/libvirt/qemu/
vm-f16-buildmachine.monitor,server,nowait -mon chardev=charmonitor
,id=monitor,mode=control -rtc base=2013-10-08T12:16:16,driftfix=slew
-no-shutdown -device piix3-usb-uhci,id=usb,bus=pci.0,addr=0x1.0x2
-device virtio-serial-pci,id=virtio-serial0,bus=pci.0,addr=0x4 -drive
if=none,media=cdrom,id=drive-ide0-1-0,readonly=on,format=raw,serial=
-device ide-drive,bus=ide.1,unit=0,drive=drive-ide0-1-0,id=ide0-1-0
-drive file=/rhev/data-center/f79b0b28-c82f-11e0-8739-78e7d1e48c4c/
```

But why?

```
docker run --cpus=2 --memory=2048M debian
```

But why?

- Extremely easy to deploy (i.e. Docker compose)
- **Millions** of images ready to use
- No provisioning, little setup, little maintenance: most containers last less than 10 minutes
- Very light ⇒ More can be hosted on the same hardware ⇒ **Less cost**
- "It just works": no hardware requirements, little skills, all OSes support containers

Tabula rasa

VMs replaced by containers in development environments as well as production environments

Why this work

Container use is on the rise. What about its security?

Container security

The hidden cost of containers

Containers are not as isolated as VMs:

- They share the same kernel
- Resources are not container allocated
    - GPU sharing
- *They only are a process in the host's OS*, which may run as `root`.

Security primitives

Containers are essentialy isolated by four security mechanisms.

- *Namespaces*, which allow the creation of separated environments for files (mounts), networking, users, and so on.
- CGroups, which restricts the amount of ressources a container has at its disposal
- Capabilities, which allow which privileges a process has
- SecComp, which restricts the system calls a process can call.

Breakout: an attacker has escaped the confines of the container and is onto the host

Breakouts

There is more than one way to kill a container:

- Container engine vulnerabilities
- *Container misconfigurations*
- Typosquatting
  - `ngixn` instead of `nginx`
- Host misconfiguration

The number of publications (and attacks) related to container security is on the rise, with the vast majority stating that containers cannot be considered as secure because of the kernel-sharing property.

We focus on the container misconfiguration part.

A sidenote on automated attacks today

- Shodan, Censys, botnets
- Hundreds of thousands of automated attacks
- Containers are no exceptions

Misconfigurations

- Each security primitive is responsible of *one specific task*
- Should one of these primitives fail for any reason, host is *immediately* exposed
- **Most errors are invisible**

Questions

We aim to answer the four following questions:

- Can containers be scanned from the inside to determine which attack is feasible?
- What are the risks with the current use of containers?
- What are the challenges in mitigating attacks with containers?
- What can be done to avoid breakouts on containers?

Tesseract - our work

Automatic scanning

Containers are chatty, most data can be found from inside the container itself. Assuming the OS is Linux, most data can be obtained directly systematically.

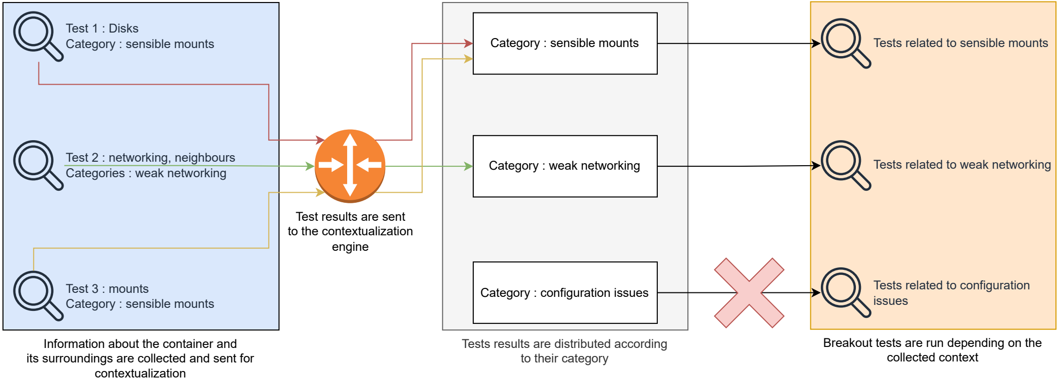
Collection process

We can collect data even with security primitives in place:

- Container tool used (Docker, Podman)
    - E.g `/.dockerenv` generally means Docker
- Hardware, possibly the type of actor running the container
    - E.g. ≥ 64Go of RAM?
- Neighbouring containers, MAC, IP, FQDN and their use
- Most importantly, the settings defining the container
    - `/proc` gives informations about SecComp, CGroups and so on

We can then store those data to search for weaknesses.

Contextualization



Test 1 : Disks
Category : sensible mounts

Test 2 : networking, neighbours
Categories : weak networking

Test 3 : mounts
Category : sensible mounts

Information about the container and its surroundings are collected and sent for contextualization

Test results are sent to the contextualization engine

Category : sensible mounts

Category : weak networking

Category : configuration issues

Tests results are distributed according to their category

Tests related to sensible mounts

Tests related to weak networking

Tests related to configuration issues

Breakout tests are run depending on the collected context

Figure 3: Contextualization process

Example of collected data

Demo

Attacks

- Excessive capabilities, including `CAP_DAC_OVERRIDE`, allowing automated breakouts or little additional requirements to breakouts
- Privileged container breakouts with `--privileged`, that allows an immediate breakout over the host as privileged containers are not containers
- Sensitive mounts, such as `/run/docker.sock`, that allow automatic complete control over the host

Infrastructures and results

Testing protocol:

- VM with 4 Cores 8 Go of RAM, 32 Go of storage
- Docker as well as Podman
- Infrastructures as close as community setups
- Tesseract is injected into each container and graded for each container tool
- Each infrastructure receives a grade, equal to the worst container score.
    - **One container to rule them all**

Grading

Based on the US grading system, A is an acceptable level of security and F representing a container that has been broken out of by our tool

Grade	Explanation
A	Optimal security
B	Sensitive information leakage
C	Additional dependencies required to breakout
D	Manual breakout is possible
F	Automated breakout has been achieved

Result - Simple website

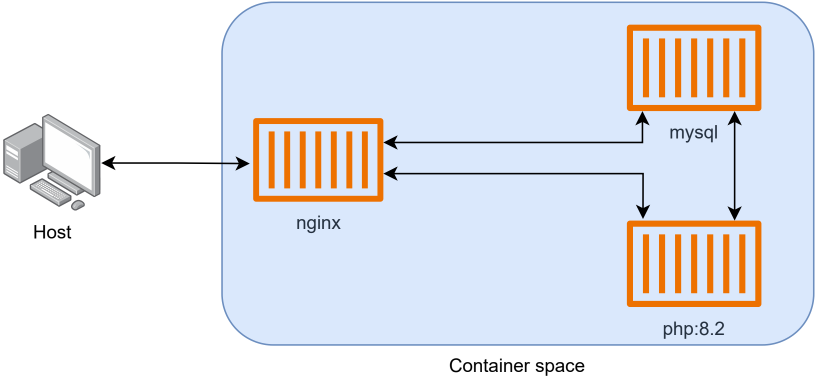


Figure 4: Simple website

Result - Simple website

Results are as follows.

name	grade
sw-nginx	A
sw-php	A
sw-mysql	A

Infrastructure grade: A

Result - Simple website with login

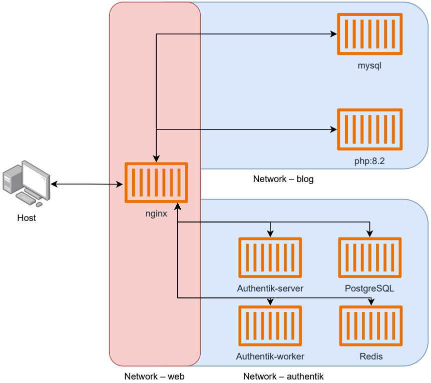


Figure 5: Simple website with login

Result - Simple website with login

name	grade
swl-nginx	A
swl-php	A
swl-mysql	A
postgresql	A
redis	A
authentik-server	A
authentik-worker	F, D

Infrastructure grade: F, D

Result - Storage monitoring

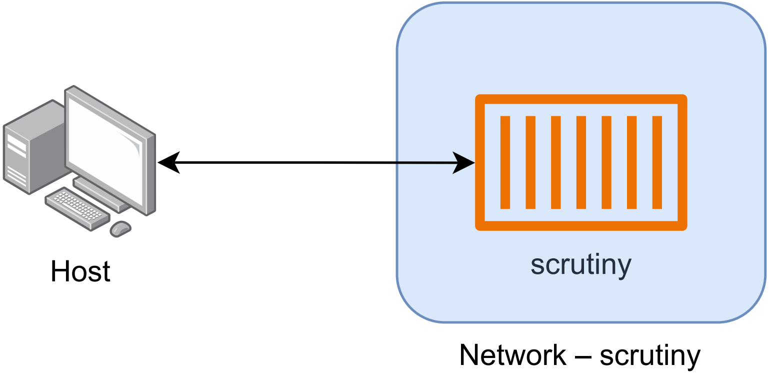


Figure 6: Storage monitoring

Result - Storage monitoring

name	grade
scrutiny	D

Infrastructure grade: D

Result - VPN

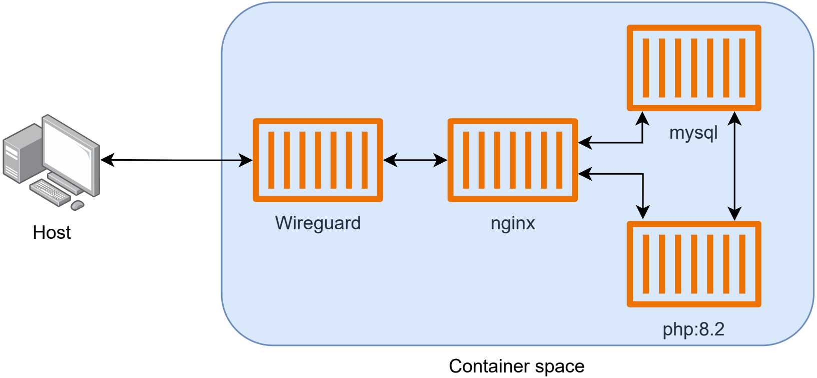


Figure 7: Simple website with VPN

Result - VPN

name	grade
vpn-nginx	A
vpn-php	A
vpn-mysql	A
vpn-wireguard	D

Infrastructure grade: D

Results - Overall

- Our tool shows a lack of isolation in 3 out of 4 tested infrastructures, either allowing automatic or manual breakouts
- All containers are from the most popular container Hub and have been downloaded more than 1 million times, for most being downloaded at least 10 and some at least more than 1 billion
- Problem is consistent with Docker and Podman, **with Docker being weaker in terms of security**
- **The provided configuration must not to be considered secure**

Thoughts

Our test set is small: 20 containers at best. However:

- All those containers are popular and configuration was taken verbatim from recommended configuration files
- Most containers are based on others: all children containers can therefore be impacted as well
- Problem in container, not image

Thoughts

Our environment is not as close as the industry as it should: no orchestrator, no multiple hosts. However:

- Orchestrators expose additional services, spreading the attack surface
- Some orchestrators provide security contexts, that are often left off
  - If it works standalone, it should work in a cluster
  - Reminder: one misconfiguration is enough

Mitigation

What is mitigation

- Fix container security issues
- Usually with an availability impact
- In our case, not an automatic process

Existing policies

- NIST
- United Kingdom Government

Contents of existing policies

- Complete isolation in between the apps and/or sensitivity levels
- Exclusive use of signed images
- Strict set of allowed settings
- Full use of security primitives

Nowadays

- 2020: Shringarputale et al.: *Co-residency Attacks on Containers are Real*
- Applied on *mature* organizations: Microsoft, Amazon, and so on
  - Extensive use of technologies designed to reduce the attack surface, such as IBM Nabla, Amazon Firecracker, Kata

What about “smaller” organizations?

In case of hack, break glass and pull cables

- Full inventory of containers, list gaps between recommended and running configuration
- Kill containers with gap
- Rework the container, perhaps the image, then perhaps the software running in it as well
    - Repeat for each container
- Be fired because you cost too much

Let's try that again, but this time good

- Full inventory of containers, list gaps between recommended and running configuration
- Start alternative container configuration with reduced permissions (capabilities, namespaces, hardware, and so on)
  - Privileged: very few containers need it
  - Hardware: ?
  - Capabilities: tedious but can be done manually
- Long and complex process

Can we do better?

- Full inventory of containers, list gaps between recommended and running configuration
- Use tools, for instance to map system calls that are required
  - SecComp compatible
  - Allows to deduce capabilities
- **Implement strict policies at the first occurence of container use**

Does it fix all container issues?

No.

Container issues that remain

- Container engine
  - Leaky Vessels CVE-2024-21626 just last week
- Host vulnerabilities
  - Outdated OS, outdated exposed software
- Hardware dependencies
  - GPU acceleration
- Network requirements
  - MTU, VLANs

So what do we do?

- Enforce default configuration as much as possible
- In cases where this is not possible, ensure other mechanisms can compensate (SecComp, and so on)
- In cases where this is still not possible, consider VMs
- **Implement strict policies at the first occurence of container use**

Conclusion

Answers

Can containers be scanned from the inside to determine which attack is feasible?

Yes, with less information than the host, but with sufficient informations to know where to look

Answers

What are the risks with the current use of containers?

Breakouts due to misconfiguration, vulnerabilities, **overabstraction**

Answers

What are the challenges in mitigating attacks with containers?

Time, complexity, roadblocks: **The cost**

Answers

What can be done to avoid breakouts on containers?

Strict pipelines in between the container maintainers and the infrastructure

Future works

- Very few attacks have been implemented
  - Instead of implementing our own attacks, numerous exploits have been published in exploit-db
  - Requires the development of our contextualization engine
- Testing our tool in hardened environments

Thank you

References

References

- 5: Jones, Doug. « Containers vs. Virtual Machines (VMs): What's the Difference? | NetApp Blog », 16 mars 2018. https://www.netapp.com/blog/containers-vs-vms/.
- 7: Donnie Berkholz. « Docker Index Shows Continued Massive Developer Adoption and Activity to Build and Share Apps with Docker | Docker », 10 febr. 2021. https://www.docker.com/blog/docker-index-shows-continued-massive-developer-adoption-and-activity-to-build-and-share-apps-with-docker/.

References

- 8: « Customers | Docker », 17 décembre 2021. https://www.docker.com/customers/.
- 8: « cern's Profile | Docker Hub ». https://hub.docker.com/u/cern.
- 48: Snyk. « Leaky Vessels: Docker and runc Container Breakout Vulnerabilities - January 2024 », 31 janvier 2024. https://snyk.io/blog/leaky-vessels-docker-runc-container-breakout-vulnerabilities/.