
MUNICH
SIDES
SECURE CONTAINERS
Do component reduction strategies
fix your container security nightmares?
secure IO

PRESENTERS

Michael Wager

- Developer, Hacker, Consultant
- Topics: DevSecOps, Automated Security Assurance, Vulnerability Management
- Interested in music, traveling, cooking and all stuff cyber security related

Michael Helwig

- Strategic Consulting: Security Programs / SSDLC / DevSecOps
- Interested in all things security (Security Testing, Threat Modeling, Cloud, Reverse Engineering, …)
- Company founder

AGENDA

1. Intro: Container Security Challenge
2. Component reduction methods ("distroless" concept)
3. Demo (Node.js)
4. Research & Comparison
5. Conclusion

WHY ARE CONTAINERS A SECURITY CHALLENGE?

Lack of processes in early adoption

- Lack of transparency into vulnerabilites in early adoption phases (no container scanning, no awareness, no CI/CD integration)
- No trusted repositories / base image selection
- Containers are everywhere (Cloud Services, vendor delivieries, …)

Responsibility Shift (Shift-Left)

- Containers managed by dev teams; servers and OS traditionally managed by ops team.
- "It's not our code"

Complex attack surfaces

- Application
- OS layer / container images
- Configuration
- Network
- Hypervisor

Security degrades over time

- Security is not constant, new vulnerabilities and attack vectors appear. The more you have to maintain, the more effort you need.

WHY ARE CONTAINERS A SECURITY CHALLENGE?

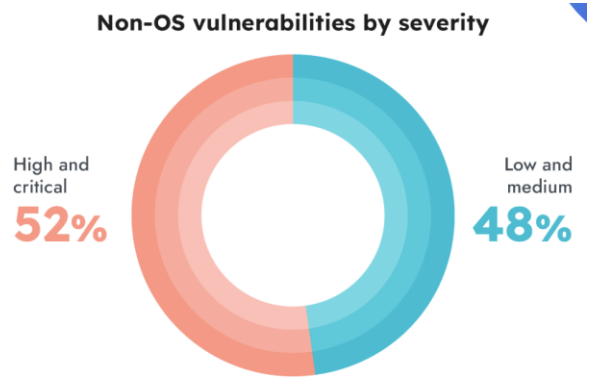
"the likelihood of a greater number of vulnerabilities increases with the complexity of the software architectural design and code."

Minimize your attack surface

https://github.com/OWASP/DevGuide/blob/master/02-Design/01-Principles%20of%20Security%20Engineering.md

CONTAINER SECURITY AND VULNERABILITY TRENDS

- High number of images with high or critical vulnerabilities
- Most of the vulnerable libraries are not actually used or needed by the application



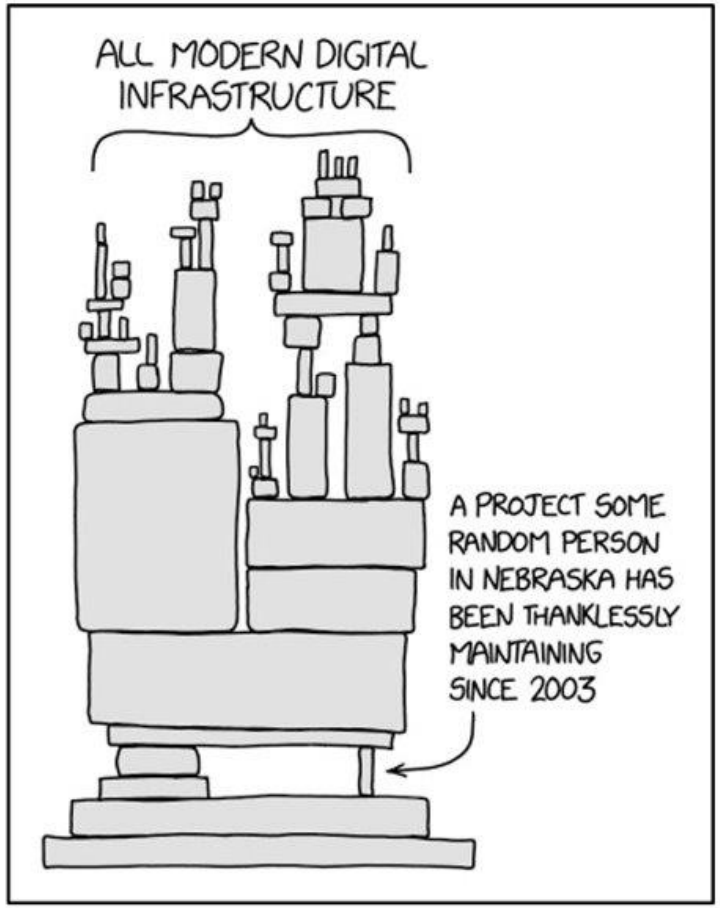
Source: https://sysdig.com/blog/2023-cloud-native-security-usage-report/

„IT‘S SECURE BECAUSE IT‘S RUNNING IN A CONTAINER“

„IT‘S SECURE BECAUSE IT‘S RUNNING IN A CONTAINER“

COMPARISON WITH OPEN SOURCE COMPONENTS

- Container base images == OSS
- Teams are responsible for the functionality and security of OSS dependencies - so they are responsible for the security of the selected base images
- Container images have security vulnerabilities too



CONTAINER IMAGE SCANNERS

- Goal: identify known vulnerabilities (CVEs) in container images (Also: sensitive information and secrets like private keys or passwords inside the container)
- Some tools: trivy, Anchore grype, docker scout, twistcli
- Easy to integrate into CI/CD pipelines
- Limitation: Packages installed with official package managers (RUN apt install...) will be detected, manually installing stuff (e.g. "RUN pecl install smbclient-stable" or custom compiled code) NOT

COMPONENT REDUCTION TOOLS

Google "distroless"

- Open source project by google (since 2007)
- Provides prod ready images for several runtimes (java, node.js, go)
- Very small in size (e.g. static-debian11: ~2MB)

Ubuntu "chisel"

- Open source project by Canonical (since 2023)
- Provides some prod ready images, others need to be built yourself („chiseled“)
- Ubuntu long-term supported (LTS) releases (0 critical 0 high findings, 24h)

RedHat UBI "micro"

- Based on RedHat's "Universal base images"
- RedHat enterprise linux (RHEL) well maintained
- Same security response team, the same security hardening

- Minimal images containing only runtime environment and the application (no shells, no package managers, etc, removes entire classes of attack, fully disarming potential attackers)
- Therefore reduced attack surface (less findings of security scanners)
- Faster transfer times, less storage size, less costs

DEMO

Sourcecode available: github.com/mwager/nodejs_exploit

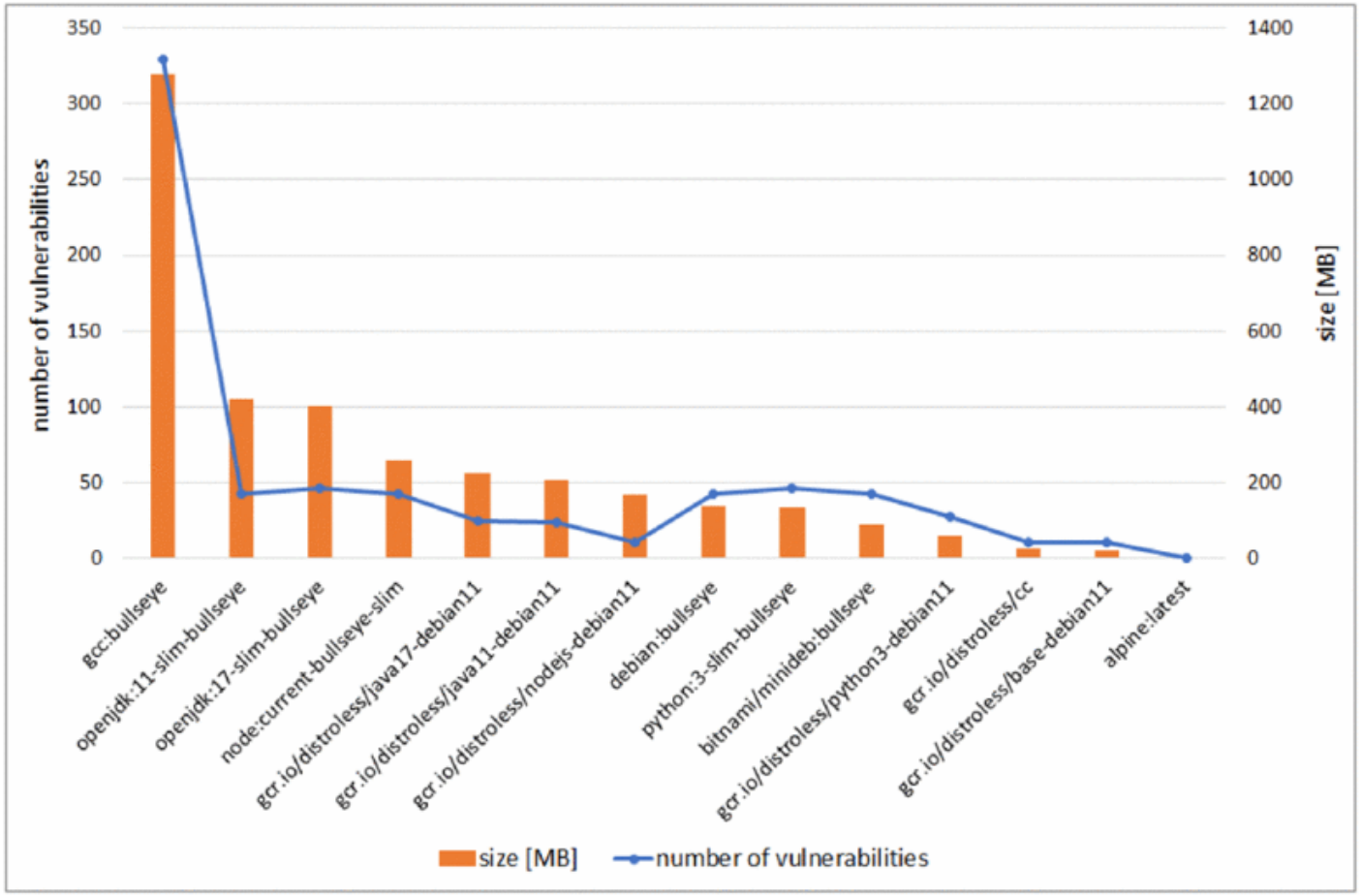
RESEARCH

- Research in collaboration with University of Applied Sciences Augsburg
- 3 Research Questions:
  - RQ1: Does the reduction of components significantly reduce the amount of vulnerabilities within a container image?
  - RQ2: Are typical vulnerabilities found through container security scanners actually exploitable and therefore a risk to the application?
  - RQ3: What are implications on development, deployment and maintenance when introducing component reduction methods?

Hochschule Augsburg University of Applied Sciences

RESEARCH: RELATED PAPER #1

- Publish date: September 2022
- Arkadiusz Maruszczak Et al. are discussing security of base systems, focusing on distroless
- Comparison of well known application based base images with google distroless images
- Conclusion:
  - *Component reduction in images doesn't always positively affect number of vulnerabilities (e.g. OpenJDK images)*
  - *Concerning Python, Node.js and GCC images, positive correlation between size and vulnerabilities is observed*



RESEARCH: RELATED PAPER #2 AND #3 (PUBLISHED AUGUST 2014 & MARCH 2022)

From paper #2

Paper compares "Vulnerability Severity and Exploits Using Case-Control Studies":

Our analysis reveals that (a) fixing a vulnerability just because it was assigned a high CVSS score is equivalent to randomly picking vulnerabilities to fix; (b) the existence of proof-of-concept exploits is a significantly better risk factor; (c) fixing in response to exploit presence in black markets yields the largest risk reduction

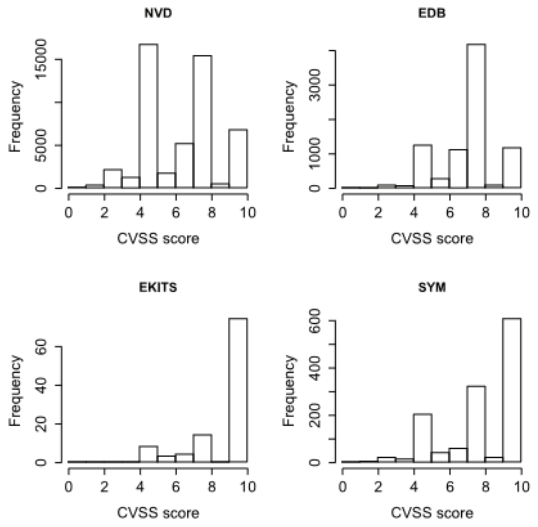


Fig. 1. Distribution of CVSS scores per dataset.

From paper #3

Focus on impact and exploitability of found vulnerabilities in base images

Finding 4: HE vulnerabilities in large OS base-images are showing an increasing trend

Finding 6: Exploitation of bash vulnerabilities can result in complete unavailability of the impacted container

Finding 7: HI vulnerabilities are observed more in large OS base-images

Finding 10: Nearly half of DH official base-images contain at least a vulnerability with PoC exploit

ADVANTAGES

- Minimal images containing only runtime environment and the application

  (no shells, no package managers, etc)
- Reduced attack surface
- Less findings of security scanners
- Removes entire classes of attacks
- Faster transfer times, less storage size, resource efficiency => less costs
- Faster build times

DISADVANTAGES & CHALLENGES

- **Complexity**
  Requires deep understanding of all underlying systems, from user i/o to kernel namespace, docker internals etc

- **Compatibility Issues**
  Some applications may rely on specific features or libraries that are missing in distroless containers

- **Debugging / No shell access**
  If your application needs to execute system commands, Distroless won't work . If you really need a shell in production, add it manually inside your Dockerfile

- **No support for certain languages**
  Google Distroless does not support PHP out of the box, but there are solutions available like this fork. You need to build it yourself.

EXAMPLE OF POTENTIAL ISSUE: MISSING BINARIES

- Example: node.js app depending on NPM package "node-rdkafka"
- Wrapper for Kafka C/C++ library *librdkafka*
- *librdkafka* depends on *zlib1g (*native shared library for compression support)
- Led to runtime error (on startup)
- Using ldd -> libz.so.1
- Solution: manual installation in stage 1, copy over in stage 2

Conclusion here: Good understanding of linux and underlying OS functionality required (always a good idea to understand the technology you are using 😉)

CONCLUSION

- Teams are responsible for the selection and security assurance of their base images (same as with their source code and open source dependencies)
- Distroless methods make your apps more secure
- Depends on your application architecture

- **Recommendations**
  - Scan your images (fail your build!)
  - Do not build your images as root!
  - Create awareness / establish community
  - Use Cloud Workload Protection or Kubernetes security features

Contact

Twitter:
@michael_wager
@c0dmtr1x

www.secure-io.de