Docker & Security

Florian Barth, barth@stocard.de
Matthias Luft, mluft@ernw.de
#whoweare

- Founding members of the CTF team **squareroots**
- Long-time infosec/IT nerds
- Florian: CTO @stocard.de
- Matthias: CEO @ERNW Research
Agenda

- Basics & Tech Stack
- Security Aspects
- (Potential) Benefits?
- Architectural Implications
- Hands-on
- Challenges
- Dev/Deployment Lifecycle meets Security
What is Docker?

- Linux-based container solution
- Used to use LXC
- LXC = Legacy API for cgroups and namespaces
  - New development linuxcontainer
- cgroups: resource prioritization/limitation of
  - CPU
  - Memory
  - Network interfaces
- Namespaces: Isolation of system view as for
  - User IDs
  - Processes
  - Network interfaces
- Layered filesystem
- Shared Kernel!
Docker & Security?

Security Objectives:
- Isolation
- Governance
  - I.e. no abuse of Docker to subvert security mechanisms, such as
    - patch management
    - software from trusted sources
    - segregation of test/prod
## Virtualisierung im Vergleich

<table>
<thead>
<tr>
<th></th>
<th>Physikaler Host</th>
<th>Virtuelle Maschine</th>
<th>Container</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Gemeinsame Ressourcen</strong></td>
<td>Teilen sich das Netz</td>
<td>Teilen sich die Host-Hardware</td>
<td>Teilen sich den Kernel</td>
</tr>
<tr>
<td><strong>Angriffsszenario</strong></td>
<td>Angriff per Netz auf offene Ports etc.</td>
<td>Angriff auf Hypervisor</td>
<td>Angriff per Syscall auf Kernel-Isolation (Namespaces, Cgroups, ...)</td>
</tr>
<tr>
<td><strong>Schutzmaß-nahmen</strong></td>
<td>Portfilter, Firewalls, Segmentierung der Netze</td>
<td>Guter Hypervisor</td>
<td>Absicherung im Containermanager, SE-Linux, Capabilities</td>
</tr>
<tr>
<td><strong>Aufwand der Maßnahmen</strong></td>
<td>Einfach, Best Practices</td>
<td>Komplex, aber zentral zu managen</td>
<td>Vielschichtig durch relativ große Angriffsfläche</td>
</tr>
</tbody>
</table>

LXC

- Current version (Mar 2016): 1.10.0
- Before 1.0 (Feb 2014):
  - Containers could not be run as non-root users
  - Implicit possibility for container-root-user to break out of the container via sysfs
  - No user namespaces
User namespaces

- Before user namespaces:
  - UID 0 in container was uid 0 in host
  - UID 1000 in container was uid 1000 in host
  - => 1:1 mapping

- With user namespaces:
  - UIDs in container can be mapped to UID range on host
  - Root in container != root on host!
Breakout Vectors

Kernel Vulnerabilities

- Shared kernel between container and host => Kernel vulnerability violates isolation
- Attack surface: syscall interface
  - More than 600 syscalls...
Docker Attack Surface

- Governance: SW from trusted sources
  - Docker images from `docker pull`
- Docker as an vulnerable application:
  - Privilege escalation on the host
  - Remote code execution on the host
Known Attacks & Vulnerability History
Breakout

- Before 1.0: Breakout by design via sysfs
  - http://blog.bofh.it/debian/id_413
- “Containers do not contain”
  - http://opensource.com/business/14/7/docker-security-selinux
  - Devices are not namespaced:
    - /dev/mem
    - /dev/sd* file system devices
    - Kernel Modules
  - If you can communicate or attack one of these as a privileged process, you can own the system.
“Shocker”, Using capabilities:

- CAP_DAC_READ_SEARCH and CAP_DAC_OVERRIDE
- Allow to open files not only by pathname (which would be restricted to container layered filesystem), but also inodes
- Iterate through inode numbers (/ is starting at 2) to access any file on the host.
On the host

Privilege escalation:

- Docker socket was world-accessible (rw)
- Create root-container with host-fs mounted
On the host

- Remote code execution
  - Symlink attacks via downloaded docker bundles that are extracted
Counter Measures & Hardening
The Basics...

- Docker is "IT + X".
- Don’t ignore traditional controls such as
  - High patch level
  - Isolation of management interfaces
  - Least privilege
- The following slides only contain Docker specifics.
- CIS Benchmark most comprehensive source (see sources).
Linux Containers

- User namespaces (see above) and non-root containers
- `seccomp`: Restrict available syscalls
- `cap-drop`: Drop capabilities for the container (such as to access files based on inodes)
Docker improvements

- Read-only sysfs/procfs
- Command line options for cap-drop
Hardening Options

- Use SElinux enforcement (many distribution ship proper profiles)
- Use hardened host kernel (GRsec)
- Use non-privileged containers
- Use docker-bench-security to check for security best practices
  - https://github.com/docker/docker-bench-security
Benefits & Architecture
I DONT ALWAYS TEST MY CODE
BUT WHEN I DO, I DO IT IN PRODUCTION

WORKED FINE IN DEV
OPS PROBLEM NOW
What is DevOps?

- **Culture / Mindset**
- **Goal:** improve quality and speed at which innovation is delivered
- **Embrace**
  - Communication
  - Collaboration
  - Integration of Development & Operations
Micro Services

- Architectural Pattern
- “Do One Thing and Do it Well!”
- Break apart monolithic apps into micro service clusters/clouds

- e.g. Amazon, Netflix, SoundCloud
Monolith Rant

**Obstacles to frequent deploys**
- Need to redeploy *everything*
- Long running jobs (?)
- Increased risk of failure

**Effects**
- Infrequent updates, long QA cycles
- Slow iterations, inhibiting experimentation
- Slows down development
- Communication overhead
- Locked into tech stack
Micro Service Tribute

Benefits:
- Smaller, more understandable apps
- No dependency hell
- Reduced startup times
- Smaller & faster deploys
- Fine-grained scaling
- Fast & Reproducible tests
- No tech lock-in
- Fault isolation
... but beware

- Complexity cost
  - Deployment
  - Overhead
  - Monitoring
  - Implicit interfaces
  - Service discovery / routing
  - Shared state

- Need to build devops XP & skills
12 Factor App

- Codebase in VCS
- Dependencies explicit and isolated
- Configuration vs Code
- Backing Services
- Build, release, run
- Stateless, isolated processes
- Disposability
- Dev/Prod/* Parity
- Logs as event streams
Docker?

- Content agnostic
- Hardware agnostic
- Content isolation & interaction
- Automation
- Highly efficient
- Separation of duties
Docker Engine

- Core of docker platform
- Offers baseline services to create and operate container
- Plugin-friendly
  - Networking
  - Logging
  - Volumes
  - Event Stream
Anatomy of a dockerized app

- Dockerfile – describes one service
- Images – runtime environments
- Containers – instance of app
- Volumes – non-ephemeral data
- Networks – communication
Docker Registry

- Distribution of docker images
- CI / CD stores images in registry
- Docker Engines pull images and run them
Docker Compose

- Definition and running of multi-container applications
- yaml-based definition of your application, including:
  - Images, services, network, volumes, ...
- docker-compose up - app running!
Docker Machine

- Automatic provisioning
  - Provision host
  - Install docker engine
  - Setup secure communication

- Provisioning Drivers
  - virtualbox, bare metal, AWS, GCE, Digital Ocean, Azure, and many more
Docker Swarm

- Built-in clustering tool for docker
- Combines a pool of hosts into one virtual docker host
- Discovery Services (consul, etcd, ZooKeeper)
- Basic filtering and scheduling
Challenges

- CI/CD
- Configuration/Orchestration Management
- Service Routing
- Log Management & Monitoring
CI / CD – Concept

[Diagram showing the CI/CD pipeline with stages: Version Control → Build → Unit Test → Deploy → Auto Test → Deploy to Production → Measure & Validate → Production Feedback]
CI / CD

Instantiation

Docker Hub

GitHub

sources with dockerfile

build image with sources

CircleCI

run tests on image

run image

AWS
SecDevOps?

NoSQL Borat
@NoSQLBorat

To make mistake is human. To automatically deploy mistake to all of servers is DevOps.
What is SecDevOps?

- Movement to make security work “DevOps” as well.
  - Haven’t seen a good implementation yet.
- More interesting question:
  - How can we integrate “security” into the described CI/CD/DevOps approach?
Objectives

- Security of the deployed application
- Security of the docker host OS + container OS
Traditional Approach
If it only would be...

- Security concept at beginning of the project
- Security assessment before releases
- Final approval before go-live
Opportunities

- There is no such thing as “out-of-band-\-patch”.
  - Also not on the OS level!
- Integrate automatic assessment tools into the deployment process
  - Nothing new though
- As ITSec: Enable yourself to have a faster dialogue with the developers
  - Establish tools (e.g. issue tracker)
  - Vuln/risk rating metric – the simpler the better!
  - Business-reasonable risk recommendations
Conclusions
I have no idea what you're talking about...

...so here's a bunny with a pancake on its head.
There's never enough time...

THANK YOU...

...for yours!

@der_cthulhu
@uchi_mata

barth@stocard.de
mluft@ernw.de
Sources

- Docker Bench: Checking for best practices
  - https://github.com/docker/docker-bench-security
- Jérôme Petazzoni on Docker Security
  - E.g.: Containers, Docker, and Security: State of the Union
- http://opensource.com/business/14/9/security-for-docker
- CIS Hardening Guide
  - https://benchmarks.cisecurity.org/tools2/docker/CIS_Docker_1.6_Benchmark_v1.0.0.pdf
Sources

- Spender, High Impact Capabilities
  - https://forums.grsecurity.net/viewtopic.php?f=7&t=2522
- http://www.schibsted.pl/2015/06/how-we-used-docker-to-deploy-schibsted-pl/