goto; conference

Click 'engage'
to rate sessions
and ask questions

Nils Magnus, LinuxTag Association

Docker Security:

Who can we trust, what should we verify?

LINUX TAG

where .com meets .org
Open Source events since 1996

Ubiquitous Containers:
Excitement vs. production use

where .com meets .org
Open Source events since 1996

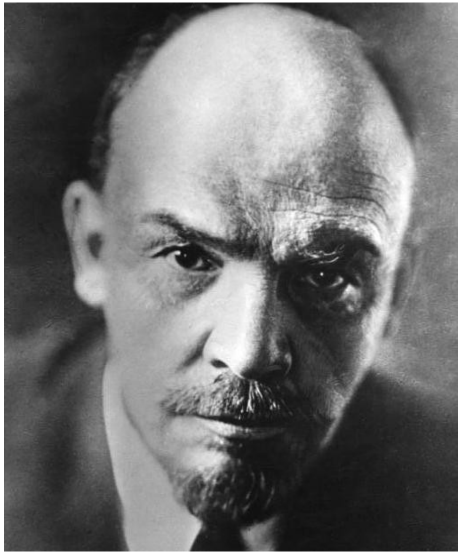
53% of decision-makers have concerns about security

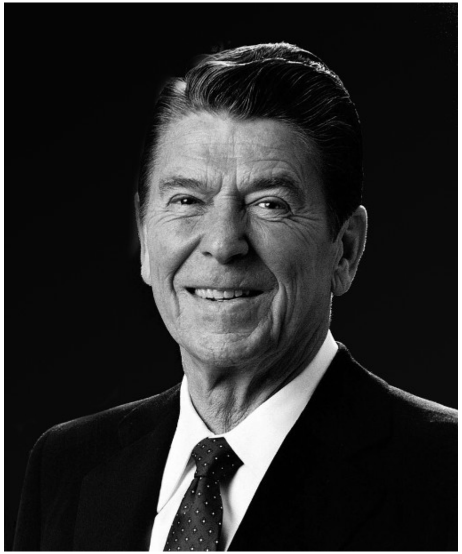
Source: Forrester/Red Hat, January 2015

Do container contain?

Доверяй, но проверяй

(„doveryai, no proveryai“)





„trust, but verify“

„Is Docker secure?“

- The term „secure“ depends on your security objectives
- To use Docker is most often more secure compared to not using Docker
- What do you plan to protect your software from?

→ Most of the time: **Isolation** (to the host and other containers)

„Protect against mistake, not abuse!“

– from the Docker, Inc. documentation

„Containers do not contain!“

– Dan Walsh, Red Hat

Isolation on Several Levels

	Physical Host	Virtual Maschine	Container
Shared Ressources	Share a network	Share host hardware	Share linux kernel
Attack scenario	Attack hosts and ports	Attack hypervisor	Attack kernel isolation via syscall (namespaces, cgroups, ...)
Security Controls	Portfilter, firewalls, segmention of networks	Good hypervisor	Inside containermanager, SE-Linux, capabilities
Complexity of Mitigation	Easy, best Practices	Complex, but possible to manage from a centralalized point	Tricky in detail due to large attack surface

Four Threat Types

- Tech Attacks
- Architecture
- Sources
- User

Threat 1: Complex attack surface

Capabilities

- Withdrawing of capabilitys prevents restoration of status quo ante (even with root permissions):

```
# getpcaps $$
Capabilities for `22424': =
cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_set
gid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service,cap_net_broadcast
,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sys_rawio,ca
p_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot,cap_sys_nice,cap
_sys_resource,cap_sys_time,cap_sys_tty_config,cap_mknod,cap_lease,cap_audit_write,cap
_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,cap_syslog,cap_wake_alarm,c
ap_block_suspend+ep
# docker run -it ubuntu /bin/bash
root@39ed301e0731:/# getpcaps $$
Capabilities for `1': =
cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_s
etpcap,cap_net_bind_service,cap_net_raw,cap_sys_chroot,
cap_mknod,cap_audit_write,cap_setfcap+eip
```

Issues with Capabilities

- Large number of usecases
- Only less than 40 capabilities are defined
- Semantics not very well defined
- Example: „CAP_SYS_PACCT“ has 30+ use cases, ranging from „random device management“ to „turning DMA on/off in xd driver“

Syscalls

- System call API to the kernel is rapidly growing:
- **Linux/arch/arm/include/asm/unistd.h:**

  22 #define __NR_syscalls (392)

- Each call is a potential attack vector into the kernel and thus to the host
- Within the kernel a single, tiny bug exploits the whole isolation

Issues with Syscalls

- Sebastian Kramer from the Openwall project released in June 2014 a proof-of-concept („Shocker“), enabling him to escape Docker 0.11 (pre-predecessor of version 1.0)
- Docker creates a new filesystem context and **bindmounts** new „/“.
- Container and host share within the kernel the **the same *struct fs*** in order to maintain bindmounts.
- Do you know **syscall *open_by_handle_at()***? To use it, you need **CAP_DAC_OVERRIDE**, which Docker had at that time.
- The resulting ressources allowed you to **traverse the inodes of the host.** That enabled you to read */etc/shadow,* for example.

Namespaces

- Virtulize/isolate important system ressources like
  - PIDs,
  - network interfaces,
  - UIDs,
  - hostnames and more
- Old way: Access global variable within the kernel for a ressource
- New way (namespace enabled): Ask a nsproxy for the ressource as which is inside your current namespace
- All access paths inside the kernel need to be scrutinized

Namespace example

- „hostname“ command → syscall uname(2) → kernel space
  - **kernel/sys.c:**

    ```
    1141 SYSCALL_DEFINE1(newuname, struct new_utsname __user *, name)
    1146         if (copy_to_user(name, utsname(), sizeof *name))
    ```
  - **include/linux/utsname.h:**

    ```
    72 static inline struct new_utsname *utsname(void)
    74         return &current->nsproxy->uts_ns->name;
    ```
- „cat /proc/version“ command → open() → kernel space → procfs
  - **proc/version.c:**

    ```
     8 static int version_proc_show(struct seq_file *m, void *v)
    10         seq_printf(m, linux_proc_banner,
    11                 utsname()->sysname, [...]
    ```

Bind- and other mounts

```
root@5a5ec53ca213:/# mount
none on / type aufs (rw,relatime,si=39574450792819a9,dio,dirperm1)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev type tmpfs (rw,nosuid,mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=666)
sysfs on /sys type sysfs (ro,nosuid,nodev,noexec,relatime)
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,relatime,mode=755)
cgroup on /sys/fs/cgroup/cpuset type cgroup (ro,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/cpu type cgroup (ro,nosuid,nodev,noexec,relatime,cpu)
cgroup on /sys/fs/cgroup/cpuacct type cgroup (ro,nosuid,nodev,noexec,relatime,cpuacct)
cgroup on /sys/fs/cgroup/memory type cgroup (ro,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/devices type cgroup (ro,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/freezer type cgroup (ro,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/net_cls type cgroup (ro,nosuid,nodev,noexec,relatime,net_cls)
cgroup on /sys/fs/cgroup/blkio type cgroup (ro,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/perf_event type cgroup (ro,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/net_prio type cgroup (ro,nosuid,nodev,noexec,relatime,net_prio)
cgroup on /sys/fs/cgroup/hugetlb type cgroup (ro,nosuid,nodev,noexec,relatime,hugetlb)
systemd on /sys/fs/cgroup/systemd type cgroup (ro,nosuid,nodev,noexec,relatime,name=systemd)
/dev/disk/by-uuid/79bd203c-aea7-4564-b00d-7ac555e31168 on /etc/resolv.conf type ext4 (rw,relatime,errors=remount-ro,data=ordered)
/dev/disk/by-uuid/79bd203c-aea7-4564-b00d-7ac555e31168 on /etc/hostname type ext4 (rw,relatime,errors=remount-ro,data=ordered)
/dev/disk/by-uuid/79bd203c-aea7-4564-b00d-7ac555e31168 on /etc/hosts type ext4 (rw,relatime,errors=remount-ro,data=ordered)
shm on /dev/shm type tmpfs (rw,nosuid,nodev,noexec,relatime,size=65536k)
mqueue on /dev/mqueue type mqueue (rw,nosuid,nodev,noexec,relatime)
devpts on /dev/console type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
proc on /proc/asound type proc (ro,nosuid,nodev,noexec,relatime)
proc on /proc/bus type proc (ro,nosuid,nodev,noexec,relatime)
proc on /proc/fs type proc (ro,nosuid,nodev,noexec,relatime)
proc on /proc/irq type proc (ro,nosuid,nodev,noexec,relatime)
proc on /proc/sys type proc (ro,nosuid,nodev,noexec,relatime)
proc on /proc/sysrq-trigger type proc (ro,nosuid,nodev,noexec,relatime)
tmpfs on /proc/kcore type tmpfs (rw,nosuid,mode=755)
tmpfs on /proc/timer_stats type tmpfs (rw,nosuid,mode=755)
```

Cgroups

- Have only limited effect to isolation needs
- Restrict consumption of several ressources
  - RAM
  - CPU
  - I/O
  - Network bandwidth
- Can be useful if DoS scenarios are feasible
- Evaluate if ressource allocation interferes with overall system architecture

Threat 2: Insufficient Architecture

LINUX TAG

where .com meets .org
Open Source events since 1996

Daemon

- Docker manages all container operations by means of a permanant daemon with a REST-API.
- Uses Unix domain socket per default, runs as root user.
- Opton -H binds daemon to a TCP port. Necessary for orchestration. Access control is important (both authorization and authentication).
- If not protected by SSL/TLS, exposing this port is dangerous.
- New project Rocket (rkt) addresses these issues, but is still in an early stage work in progress.

Application Architecture

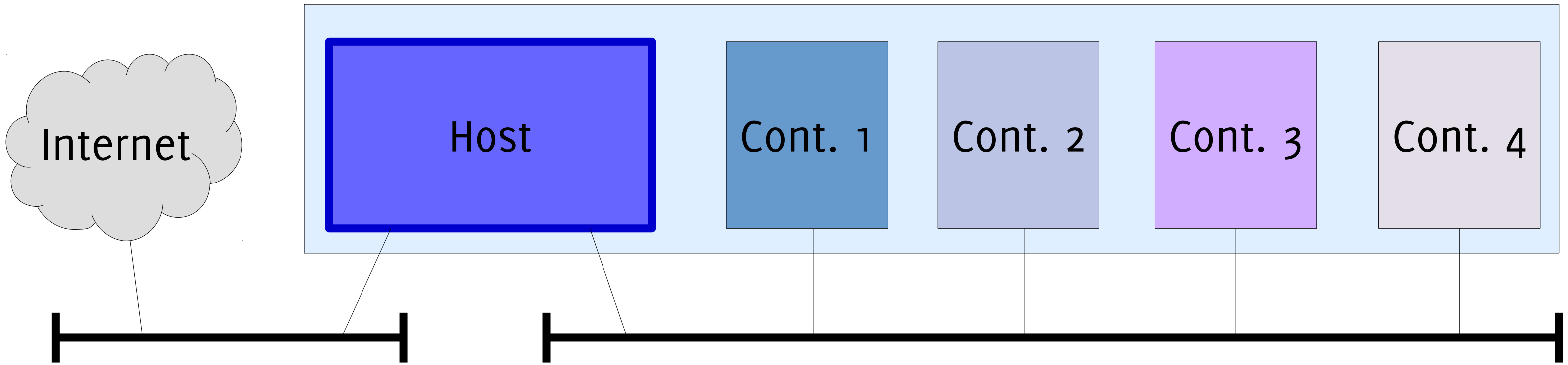
- Single containers are fun, but effectively neat toys
- Mature applications have resilience objectives:
  - Scale out (being able to deal with any number of containers)
  - High availibility (failover if single containers die)
  - Load balancing (distribute workloads to idle containers)
  - Statelessness (so you can replace and upgrade any part of the application)
  - Separation of duties
- Application design, orchestration, and integration into a CI/CD pipeline are serious tasks by themselves

Security Architecture

- Defense in Depth:
  Never rely on a single measure to protect your application and data
  - Container implement isolation
  - Firewalls provide additional network access
  - Encryptions protects data
- Single applications in a single container
- No need to SSH into containers in production (pet vs. cattle)

Network Security

- Per default, all containers share a common bridge.
- All container are thus part of the same segment/subnet: there is no special separation on the network layer between single containers.
- A host firewall is not sufficient to prevent from neighbor attacks:



- Communication of cluster management (etcd, k8s etc.) needs to be authentic and confidential (no default)

Threat 3: Unauthentic sources

Images

- Images are a convenient feature of Docker
- **Malware shipped in containers**
  - Necessity to check content
  - Experimental feature: signed images (since 1.3). Rkt has this built in
- **Malware shipped in packages**
  - Patchmanagement remains important
  - Update path for active containers (pet vs. Cattle)
- Run your own Repository? Don't store credentials!

CI pipeline

- **Build your own images (really easy with Dockerfiles)**
  - Version control for Dockerfiles
  - Integrate build into CI pipeline
  - Run your own image registry
  - Never have credentials inside your containers

- Validate sources for conatiners and packages

Threat 4: Turning off security features

What could possibly go wrong?

```
# sudo docker run --privileged=true -it ubuntu
```

- Containers are not „small desktops“
- Don't try to enable every feature that exists in a legacy distro
- Bind mounts can be nasty → use a data storage or object store
- There's no need to access raw hardware features inside containers

Four essential facts and tipps about container security:

(1) A lot of security measures for isolation are built-in, more are to come.

(2) Containers protect against mistake, not abuse!

(3) Don't put all your eggs in one basket!

(4) Never rely on sources you don't trust!

Questions about container security?

– Send me an email: magnus@linuxtag.org

Credits:
Thanks for great footage under free and open licenses from Wikipedia, MorgueFile, Federal archive of Germany, Executive Office of the President of the United States, and Toronto Public Library.

goto; conference

Please

Remember to rate session

Thank you!