Click ‘engage’ to rate sessions and ask questions.
Nils Magnus, LinuxTag Association

Docker Security:

Who can we trust, what should we verify?
Ubiquitous Containers: Excitement vs. production use
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

<table>
<thead>
<tr>
<th></th>
<th>Physical Host</th>
<th>Virtual Machine</th>
<th>Container</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared Resources</td>
<td>Share a network</td>
<td>Share host hardware</td>
<td>Share Linux kernel</td>
</tr>
<tr>
<td>Attack Scenario</td>
<td>Attack hosts and ports</td>
<td>Attack hypervisor</td>
<td>Attack kernel isolation via syscall (namespaces, cgroups, ...)</td>
</tr>
<tr>
<td>Security Controls</td>
<td>Portfilter, firewalls, segmentation of networks</td>
<td>Good hypervisor</td>
<td>Inside containermanager, SE-Linux, capabilities</td>
</tr>
<tr>
<td>Complexity of Mitigation</td>
<td>Easy, best Practices</td>
<td>Complex, but possible to manage from a centralized point</td>
<td>Tricky in detail due to large attack surface</td>
</tr>
</tbody>
</table>
Four Threat Types

- Tech Attacks
- Architecture
- Sources
- User
Threat 1:
Complex attack surface
Capabilities

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

```
# getpcaps $$
Capabilities for `22424': =
cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service,cap_net_broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sys_rawio,cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot,cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_config,cap_mknod,cap_lease,cap_audit_write,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,cap_mac_resource,cap_mac_time,cap_mac_capset,cap_mac_veth,cap_mac Bridge,cap_macvlan,cap_macvlan_filter,cap_macvlan_gre,cap_macvlan_gpt,cap_macvlan_incast,cap_macvlan_ipinip,cap_macvlan_label,cap_macvlan_llabel,cap_macvlan_l2cap,cap_macvlan_nexthdr,cap_macvlan_norm, cap_macvlan_payment,cap_macvlan_l2t,cap_macvlan_c都想,cap_macvlan_t盖,cap_macvlan_erp,cap_macvlan_dot1q,cap_macvlan_hip,cap_macvlan_l2tp,cap_macvlan_udt,cap_macvlan_vrrp,cap_macvlan_vxlan,cap_macvlan_xnet,cap_netppp,cap_netfilter, cap_duid, cap_net_clsact
```
```bash
# docker run -it ubuntu /bin/bash
root@39ed301e0731:/# getpcaps $$
Capabilities for `1': =
```
Issues with Capabilities

- Large number of use cases
- 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 (predecessor of version 1.0).
- Docker creates a new filesystem context and bindmounts new „/“.
- Container and host share within the kernel 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 resources allowed you to traverse the inodes of the host. That enabled you to read /etc/shadow, for example.
Namespaces

- Virtulize/isolate important system resources like
  - PIDs,
  - network interfaces,
  - UIDs,
  - hostnames and more

- Old way: Access global variable within the kernel for a resource

- New way (namespace enabled): Ask a nsproxy for the resource 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, [...]

where .com meets .org
Open Source events since 1996
### Bind- and other mounts

```
root@5a5ec53ca213:/# mount
none on / type aufs (rw,relatime,si=3957445079281929d0,dir,dirperm1)
proc on /proc type proc (rw,nosuid,nodev,nodev,relatime)
tmpfs on /dev type tmpfs (rw,nosuid,nodev,relatime)
devpts on /dev/pts type devpts (rw,nosuid,nodev,nodev,relatime,gid=5,mode=620,ptmxmode=666)
sysfs on /sys type sysfs (ro,nosuid,nodev,nodev,relatime)
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,nodev,relatime,mode=755)
cgroup on /sys/fs/cgroup/cpu type cgroup (ro,nosuid,nodev,nodev,relatime,cpu)
cgroup on /sys/fs/cgroup/cpuset type cgroup (ro,nosuid,nodev,nodev,relatime,cpuset)
cgroup on /sys/fs/cgroup/memory type cgroup (ro,nosuid,nodev,nodev,relatime,mem)
cgroup on /sys/fs/cgroup/devices type cgroup (ro,nosuid,nodev,nodev,relatime,devices)
cgroup on /sys/fs/cgroup/freezer type cgroup (ro,nosuid,nodev,nodev,relatime,freezer)
cgroup on /sys/fs/cgroup/net_cls type cgroup (ro,nosuid,nodev,nodev,relatime,net_cls)
cgroup on /sys/fs/cgroup/hugetlb type cgroup (ro,nosuid,nodev,nodev,relatime,hugetlb)
/systemd on /sys/fs/cgroup/systemd type cgroup (ro,nosuid,nodev,nodev,relatime,name=systemd)
/dev/disk/by-uuid/79bd293c-aea7-456a-b0ed-7ac555e31168 on /etc/resolv.conf type ext4 (rw,relatime,errors=remount-ro,metadata=ordered)
/dev/disk/by-uuid/79bd293c-aea7-456a-b0ed-7ac555e31168 on /etc/hostname type ext4 (rw,relatime,errors=remount-ro,metadata=ordered)
/dev/disk/by-uuid/79bd293c-aea7-456a-b0ed-7ac555e31168 on /etc/hosts type ext4 (rw,relatime,errors=remount-ro,metadata=ordered)
shm on /dev/shm type tmpfs (rw,nosuid,nodev,nodev,relatime,size=65536k)
proc on /proc/consol type proc (ro,nosuid,nodev,nodev,relatime)
proc on /proc/bus type proc (ro,nosuid,nodev,nodev,relatime)
proc on /proc/fs type proc (ro,nosuid,nodev,nodev,relatime)
proc on /proc/irq type proc (ro,nosuid,nodev,nodev,relatime)
proc on /proc/sys type proc (ro,nosuid,nodev,nodev,relatime)
proc on /proc/sysrq-trigger type proc (ro,nosuid,nodev,nodev,relatime)
tmpfs on /proc/kcore type tmpfs (rw,nosuid,nodev,mode=755)
tmpfs on /proc/timer_stats type tmpfs (rw,nosuid,nodev,mode=755)
```
Cgroups

- Have only limited effect to isolation needs
- Restrict consumption of several resources
  - RAM
  - CPU
  - I/O
  - Network bandwidth
- Can be useful if DoS scenarios are feasible
- Evaluate if resource allocation interferes with overall system architecture
Threat 2: Insufficient Architecture
Daemon

- Docker manages all container operations by means of a permanent daemon with a REST-API.
- Uses Unix domain socket per default, runs as root user.
- Option -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 availability (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 containers 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 containers 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.
Please remember to rate session.

Thank you!

Follow us on Twitter @GOTOber