Docker Foundation

An in depth introduction to Docker and Containers

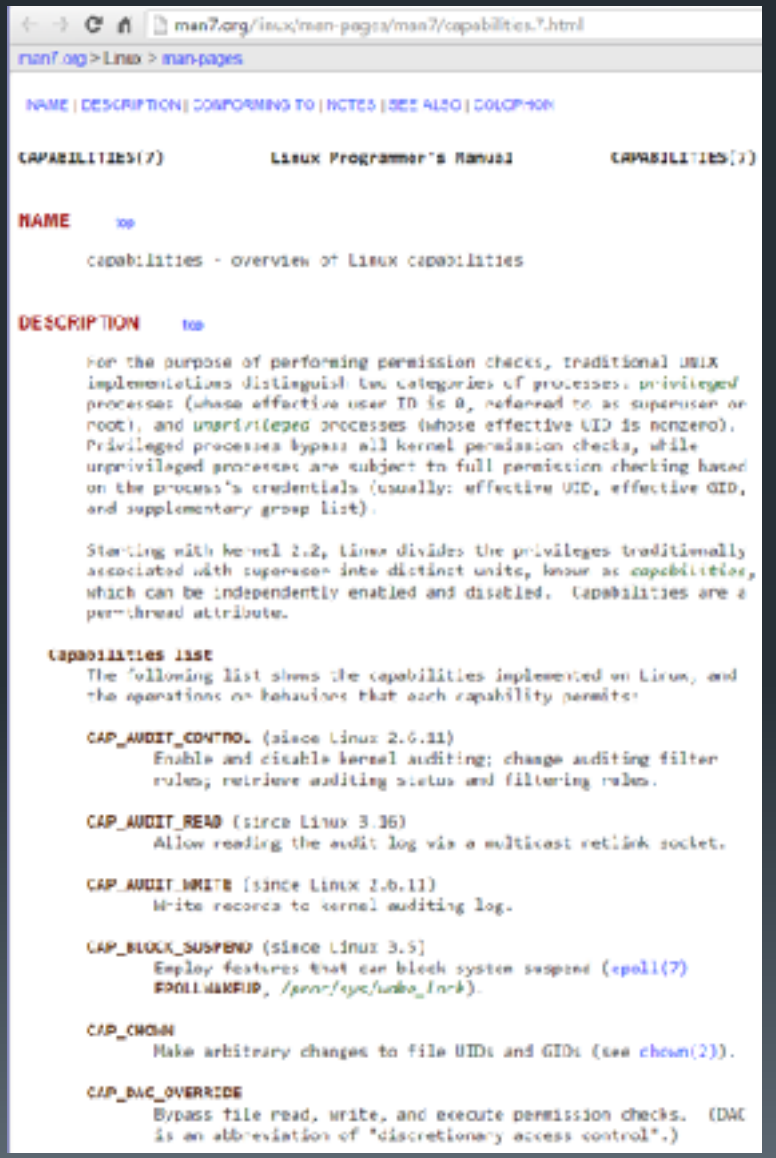
2: Security

Objectives

- Explain the nature and use of Linux Capabilities with Docker
- Describe the benefits and dangers of `--privileged`
- List the container device options
- Explore the Docker `--security-opt` systems supported
- Understand user namespaces
- Discuss security best practices

Capabilities

- The docker container run command offers several security related switches
  - `--privileged`
    - Grants all capabilities inside a container (no whitelist)
    - Not recommended for production (container processes run as if directly on the host)
  - `--cap-add` & `--cap-drop` (v1.2)
    - Linux Capabilities turn the binary "root/non-root" dichotomy into a fine-grained access control system
    - Containers can be given complete capabilities or they can follow a whitelist of allowed capabilities
    - `--cap-add/drop` give you fine grain control over Linux Capabilities granted to a particular container
    - Examples:
      - `docker container run --cap-add=NET_ADMIN \ ubuntu sh -c "ip link eth0 down"`
      - `docker container run --cap-drop=CHOWN ...`
      - `docker container run --cap-add=ALL \ --cap-drop=MKNOD ...`
    - Linux Capabilities documentation: http://man7.org/linux/man-pages/man7/capabilities.7.html
      - There are currently 38 separate capabilities



Default Capabilities

- Default Capabilities
  - https://github.com/moby/moby/blob/master/oci/defaults.go

Docker Linux Capability defaults (White List)

```
s.Process.Capabilities = []string{
        "CAP_CHOWN",
        "CAP_DAC_OVERRIDE",
        "CAP_FSETID",
        "CAP_FOWNER",
        "CAP_MKNOD",
        "CAP_NET_RAW",
        "CAP_SETGID",
        "CAP_SETUID",
        "CAP_SETFCAP",
        "CAP_SETPCAP",
        "CAP_NET_BIND_SERVICE",
        "CAP_SYS_CHROOT",
        "CAP_KILL",
        "CAP_AUDIT_WRITE",
}

s.Linux = specs.Linux{
        MaskedPaths: []string{
                "/proc/kcore",
                "/proc/latency_stats",
                "/proc/timer_list",
                "/proc/timer_stats",
                "/proc/sched_debug",
        },
        ReadonlyPaths: []string{
                "/proc/asound",
                "/proc/bus",
                "/proc/fs",
                "/proc/irq",
                "/proc/sys",
                "/proc/sysrq-trigger",
        },
        Namespaces: []specs.Namespace{
                {Type: "mount"},
                {Type: "network"},
                {Type: "uts"},
                {Type: "pid"},
                {Type: "ipc"},
        },
```

Displaying Capabilities

- *capsh* can be used to display and change capabilities

```
user@ubuntu:~$ capsh --print
Current: =
…

root@ubuntu:/# capsh --print
Current: = cap_chown, cap_dac_override, cap_fowner, cap_fsetid, cap_kill,
cap_setgid, cap_setuid, cap_setpcap, cap_net_bind_service, cap_net_admin,
cap_net_raw, cap_sys_chroot, cap_mknod, cap_audit_write, cap_setfcap+eip
…


user@ubuntu:~$ sudo capsh --print
[sudo] password for user:
Current: = 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_syslog,
cap_wake_alarm, cap_block_suspend, 37+ep
```

Devices

- Containers have a minimal set of devices by default
- Other needed devices from the host can be mapped into a container
  - You can use devices in privileged containers by bind mounting them (with `-v`)
  - A better option may be `--device` (v1.2)
    - The `--device` flag lets you use a device without `--privileged`
    - Examples:
      - `docker container run --device=/dev/snd:/dev/snd ...`

```
user@ubuntu:~$ docker run -it busybox
/ # ls -l /dev
total 0
crw-------    1 root     root      136,  16 Aug 23 20:47 console
lrwxrwxrwx    1 root     root            11 Aug 23 20:47 core -> /proc/kcore
lrwxrwxrwx    1 root     root            13 Aug 23 20:47 fd -> /proc/self/fd
crw-rw-rw-    1 root     root        1,   7 Aug 23 20:47 full
crw-rw-rw-    1 root     root       10, 229 Aug 23 20:47 fuse
drwxrwxrwt    2 root     root            40 Aug 23 20:47 mqueue
crw-rw-rw-    1 root     root        1,   3 Aug 23 20:47 null
lrwxrwxrwx    1 root     root             8 Aug 23 20:47 ptmx -> pts/ptmx
drwxr-xr-x    2 root     root             0 Aug 23 20:47 pts
crw-rw-rw-    1 root     root        1,   8 Aug 23 20:47 random
drwxrwxrwt    2 root     root            40 Aug 23 20:47 shm
lrwxrwxrwx    1 root     root            15 Aug 23 20:47 stderr -> /proc/self/fd/2
lrwxrwxrwx    1 root     root            15 Aug 23 20:47 stdin -> /proc/self/fd/0
lrwxrwxrwx    1 root     root            15 Aug 23 20:47 stdout -> /proc/self/fd/1
crw-rw-rw-    1 root     root        5,   0 Aug 23 20:47 tty
crw-rw-rw-    1 root     root        1,   9 Aug 23 20:47 urandom
crw-rw-rw-    1 root     root        1,   5 Aug 23 20:47 zero
/ #
```

The console device is created by the run `-t` switch

Security Options

- `--security-opt` (v1.3)
  - Sets custom SELinux labels, AppArmor profiles and SecComp files
  - All confine programs (not users) to a limited set of resources
- SELinux
  - Docker offers two forms of SELinux protection:
    - Type enforcement
    - Multi-category security (MCS) separation
  - A policy ("svirt_apache") allowing a container process to listen only on Apache ports can be applied as follows:
    - `docker container run \ --security-opt label:type:svirt_apache \ -it centos bash`
- AppArmor
  - The Docker binary installs a docker-default AppArmor profile generated from the template at: https://github.com/moby/moby/blob/master/profiles/apparmor/template.go
    - Versions 1.13 and later, generated in tmpfs and loaded into the kernel
    - Versions earlier than 1.13, profile is generated in: /etc/apparmor.d/docker
  - Moderately protective while providing wide application compatibility
  - When you run a container, it uses the docker-default policy unless you override it with the security-opt option
    - Save a custom profile to disk in /etc/apparmor.d/containers/ (as my_profile for example)
    - `$ docker container run \ --security-opt apparmor=my_profile \ hello-world`
  - A profile for the Docker Engine daemon exists but it is not currently installed with the deb packages
    - located in contrib/apparmor in the Docker Engine source

```
// +build linux

package apparmor

// baseTemplate defines the default apparmor profile for containers.

const baseTemplate = `
{{range $value := .Imports}}
{{$value}}
{{end}}
profile {{.Name}} flags=(attach_disconnected,mediate_deleted) {
{{range $value := .InnerImports}}
  {{$value}}
{{end}}

  network,
  capability,
  file,
  umount,

  deny @{PROC}/* w,   # deny write for all files directly in /proc (not in a subdir)
  # deny write to files not in /proc/<number>/** or /proc/sys/**
  deny @{PROC}/{[^1-9],[^1-9][^0-9],[^1-9s][^0-9y][^0-9s],[^1-9][^0-9][^0-9][^0-9]*}/** w,
  deny @{PROC}/sys/[^k]** w,  # deny /proc/sys except /proc/sys/k* (effectively /proc/sys/
kernel)
  deny @{PROC}/sys/kernel/{?,??,[^s][^h][^m]**} w,  # deny everything except shm* in /proc/sys/
kernel/
  deny @{PROC}/sysrq-trigger rwklx,
  deny @{PROC}/mem rwklx,
  deny @{PROC}/kmem rwklx,
  deny @{PROC}/kcore rwklx,

  deny mount,

  deny /sys/[^f]*/** wklx,
  deny /sys/f[^s]*/** wklx,
  deny /sys/fs/[^c]*/** wklx,
  deny /sys/fs/c[^g]*/** wklx,
  deny /sys/fs/cg[^r]*/** wklx,
  deny /sys/firmware/** rwklx,
  deny /sys/kernel/security/** rwklx,

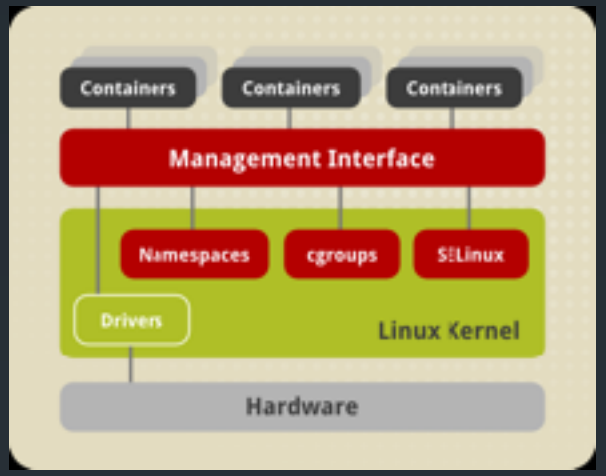
{{if ge .Version 208095}}
  # suppress ptrace denials when using 'docker ps' or using 'ps' inside a container
  ptrace (trace,read) peer={{.Name}},
{{end}}
}
`
```

Docker v1.10 Security Improvements

- Many security improvements were added in Docker v1.10
- Seccomp
  - Secure computing mode (Seccomp) is a Linux kernel feature
    - Requires a kernel configured with `CONFIG_SECCOMP`
  - Seccomp was developed by Google to remove system calls from a process (used with Chrome plugins)
    - There are 600 syscalls and a bug in any one could enable privilege escalation
  - libseccomp was created by Red Hat's Paul Moore to simplify the management of the syscall tree, now used in tools like qemu, system, lxc tools and Docker
  - Docker v1.10 `--security-opt` now allows admins to configure seccomp profiles per container
    - ```
      docker container run --rm -it --security-opt \
            seccomp:/path/to/seccomp/profile.json hello-world
      ```
- Seccomp profiles require seccomp 2.2.1
  - Only available with
    - Debian 9 "Stretch" +
    - Ubuntu 15.10 "Wily" +
    - Fedora 22 +
    - RHEL 7 & CentOS 7 +
    - Oracle Linux 7 +
  - Seccomp backports
    - Ubuntu 14.04, Debian Wheezy, or Debian Jessie can download a static Docker Linux binary (not available for other distributions)
      ```

Docker v1.10 Security Improvements

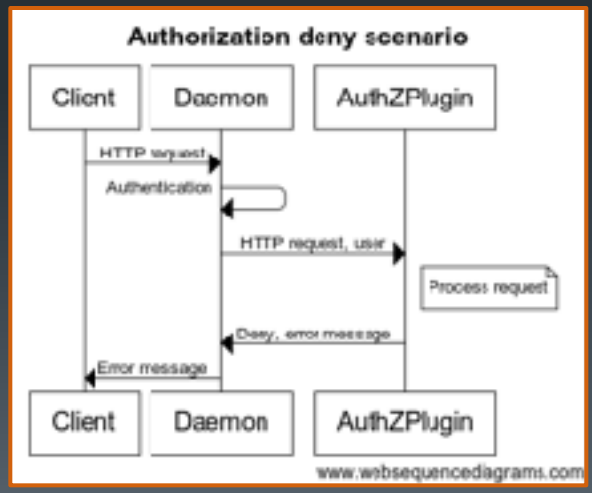
- User namespaces
  - A kernel namespace allowing separation between UIDs on the host and the container
  - Added to Linux kernel 2.6 in 2008, enabled in most distros after 2014
  - A range of container UID/GIDs (e.g. 0-1,000) map into the host user namespace as 70,000-71,000
    - e.g. the kernel treats UID 0 (root) inside the container as UID 70,000 outside the container (nobody)
  - Any UID on a file or a process that is not in the mapped range would be treated as UID=-1 and not be accessible in the container
    - Many kernel subsystems are not namespaced and are accessible to containers making this a key security improvement (e.g. SELinux, Kernel Modules and some paths for /sys, /proc, and /dev)
  - Now the best user to use inside a container is ROOT!
- Options for `--userns-remap`
  - default – maps container users (including root) to a range of high number host users
  - uid – maps root to the specified user
  - uid:gid – maps root to the specified user and group
  - username – maps root to the specified user
  - username:groupname – maps root to the specified user and group




```
$ dockerd --userns-remap=default
```

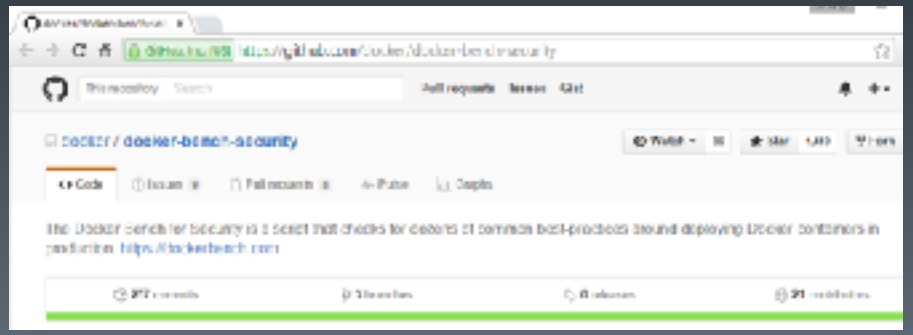
Docker v1.10 Security Improvements

- Auth
  - Prior to Docker v1.10, if you can talk to Docker's socket you can do anything
  - The new Docker daemon `--authorization-plugin` flag allows admins to add RBAC (role based access control) authorization features
  - Auth plugins act as interceptors that can allow/deny any docker API request based on rules created by admins
    - Plugins receive the current authentication context and the command context
  - You can install multiple plugins and chain them together in a defined order
    - All plugins must GRANT access for access to succeed
  - Shortfall
    - Docker does not currently supply any authorization plugins (3rd parties?)
    - Docker only offers certificate-based authentication presently



Security Best Practices

- Run Docker Engine with AppArmor, SELinux, or Seccomp to provide better process containment (`--security-opt`)
  - Capabilities is the default security system used by Docker engine
  - AppArmor profile generator for docker containers: https://github.com/jessfraz/bane
- Map groups of mutually-trusted containers to separate machines
- Do not run untrusted applications with root privileges (`--userns-remap`)
- Follow the “Principle of least privilege”
  - In a particular abstraction layer of a computing environment, every module (such as a process, a user or a container) must be able to access only the information and resources that are necessary for its legitimate purpose (`--cap-add/drop`, `seccomp`, or `--device`; never `--privileged`)
- Resources
  - Center for Internet Security Benchmark for Docker 1.1 (7-6-2017):
    - https://www.cisecurity.org/benchmark/docker/
  - Docker Inc’s Introduction to Container Security:
    - https://www.docker.com/sites/default/files/WP_IntrotoContainerSecurity_08.19.2016.pdf
  - Docker Security Center
    - http://www.docker.com/docker-security
  - Docker Bench for Security
    - https://github.com/docker/docker-bench-security



Container Value Add

Docker security non-events

Estimated reading time: 3 minutes

This page lists security vulnerabilities which Docker mitigated, such that processes run in Docker containers were never vulnerable to the bug—even before it was fixed. This assumes containers are run without adding extra capabilities or not run as `--privileged`.

The list below is not even remotely complete. Rather, it is a sample of the few bugs we've actually noticed to have attracted security review and publicly disclosed vulnerabilities. In all likelihood, the bugs that haven't been reported far outnumber those that have. Luckily, since Docker's approach to secure by default through apparmor, seccomp, and dropping capabilities, it likely mitigates unknown bugs just as well as it does known ones.

Bugs mitigated:

- CVE-2013-1956, 1957, 1958, 1959, 1979, CVE-2014-4014, 5206, 5207, 7970, 7975, CVE-2015-2925, 8543, CVE-2016-3134, 3135, etc.: The introduction of unprivileged user namespaces lead to a huge increase in the attack surface available to unprivileged users by giving such users legitimate access to previously root-only system calls like `mount()`. All of these CVEs are examples of security vulnerabilities due to introduction of user namespaces. Docker can use user namespaces to set up containers, but then disallows the process inside the container from creating its own nested namespaces through the default seccomp profile, rendering these vulnerabilities unexploitable.
- CVE-2014-0181, CVE-2015-3339: These are bugs that require the presence of a setuid binary. Docker disables setuid binaries inside containers via the `NO_NEW_PRIVS` process flag and other mechanisms.
- CVE-2014-4699: A bug in `ptrace()` could allow privilege escalation. Docker disables `ptrace()` inside the container using apparmor, seccomp and by dropping `CAP_PTRACE`. Three times the layers of protection there!
- CVE-2014-9529: A series of crafted `keyctl()` calls could cause kernel DoS / memory corruption. Docker disables `keyctl()` inside containers using seccomp.
- CVE-2015-3214, 4036: These are bugs in common virtualization drivers which could allow a guest OS user to execute code on the host OS. Exploiting them requires access to virtualization devices in the guest. Docker hides direct access to these devices when run without `--privileged`. Interestingly, these seem to be cases where containers are "more secure" than a VM, going against common wisdom that VMs are "more secure" than containers.
- CVE-2016-0728: Use-after-free caused by crafted `keyctl()` calls could lead to privilege escalation. Docker disables `keyctl()` inside containers using the default seccomp profile.
- CVE-2016-2383: A bug in eBPF – the special in-kernel DSL used to express things like seccomp filters – allowed arbitrary reads of kernel memory. The `bpf()` system call is blocked inside Docker containers using (ironically) seccomp.
- CVE-2016-3134, 4997, 4998: A bug in setsockopt with `IPT_SO_SET_REPLACE`, `ARPT_SO_SET_REPLACE`, and `ARPT_SO_SET_REPLACE` causing memory corruption / local privilege escalation. These arguments are blocked by `CAP_NET_ADMIN`, which Docker does not allow by default.

Summary

- The Linux Kernel API is organized into capabilities which can be added or dropped per container
- The `--privileged` switch extends all capabilities, making it particularly troublesome in security conscious settings
- Devices can be mapped to containers as needed
- Docker `--security-opts` allow container processes to be constrained by AppArmor, SELinux and/or SecComp
  - Default is Capabilities
- User namespaces allow container users to be mapped to unprivileged users on the host

Lab 2

- Docker Security