Linux Containers: Future or Fantasy?

Aaron Grattaflori
Technical Director
NCC Group (aka iSEC Partners/Matasano Security/Intrepidus Group)
Infosec, pentesting, Neg9/CTF

iSEC Partners for 5.5 years
  NCC Group for 0.1 years

Hacking Samsung Smart TVs @ BH USA 2013, Toorcon, etc
Macs in the age of the APT @ BH USA 2011, Source, etc
Disclaimer

These slides are not intended to be consumed without the corresponding presentation or whitepaper. The information contained within is designed for presenting and not 100% completeness with regards to risks, recommendations, findings, etc.
Story One: The Server
Once Upon a Time

Bob’s Ruby on Rails app gets popped or his SQL database server is compromised or his Wordpress plugin gives RCE or ....

He wants to add security... But how?
Chroot?

OLD

The tried and true still used today

Broken if you have root
mkdir("ncc");
chroot("ncc");
chdir("../.."); ← oh no...
chroot(".");
SELinux?

NSA made it

Complex type system for MLS systems

Good support on RHEL
Complexity

Linus Torvalds problem

The `setenforce 0` problem

Kernel enforces it: *Kernel gotta kernel*
OK, No MAC but grsecurity!

Well you’ve protected the kernel, apps and helped prevention memory corruption and hardened against other attacks but...
Full Virtual Machines?
Full Virtual Machines 😞

QEMU, KVM or ESX escapes

Recent Xen/QEMU updates anyone?

VM for single process?  
Nope.
Story Two: The Client
“Gulenn” talks to a potential source named “citizenfour”

He can’t use a Chromebook because he is is paranoid of Google
Hey, just use Linux!

“Malware is just for Windows”

“OSX sucks, it’s insecure”

Linox is like... super sakure right?
aaaaannndd broken...

He’s one webkit or gekco bug away from a TBB compromise. **What app sandboxes?**

Pidgin and libpurple don’t have a great track record

LiveCDs are stale code by definition
Story Three: The Embedded
Margaret is in charge of embedded security at D-LINK, Belkin, <insert IoT company>.

She wants to add isolation between the web app, wpa_supplicant and DLNA stack.

Tired of having **CSRF-able arbitrary code execution** via buggy input validation.
Margret isn’t alone!

Everything runs as root

No security is added (because $$$)

You can’t easily virtualize or segment ARM/MIPS within a router, but is there nothing we can do to improve IoT?
What do these stories have in common?
What do these stories have in common?

**Attack surface matters** *almost* more than anything else.

**Sandbox**es and containers at least let us pick our battles: they **should be the rule not the exception** (Props to Google Chrome Browser, Adobe Reader X, Apple Seatbelt, Google ChromeOS, etc).

How can we work to improve server, desktop and embedded security for Linux?
We have to try something new
Paul Smecker: They exited out the front door. They had no idea what they were in for. **Now they're staring at six men with guns drawn.** It was a fucking ambush.
Paul Smecker: This was a fucking bomb dropping on Beaver Cleaverville. For a few seconds, this place was Armageddon!
Officer Greenly: What if it was just one guy with six guns?
Paul Smecker: Why don't you let me do the thinking, huh, genius?
But Greenly was right... it was “il Duce”
What if it wasn’t **one cpu with multiple kernels**, but

**one kernel with multiple userlands?**
Linux Vservers
OpenBSD/NetBSD Sysjail
HP UX Containers

OpenVZ
FreeBSD Jails
Solaris Zones
AIX Workload Partitions
A little bit about OS Virtualization

Fundamentally less secure than hardware virtualization
Hardware virtualization creates software emulation for pretty much everything

Software or OS virtualization partitions a single kernel and attempts to restrict or control access to hardware
Hardware virtualization is even fundamentally less secure than physically different hardware…
(surrounded by guys with guns and fences)

But we don’t want to depend on a **single method** for security …
Namespaces
Namespaces

Plan9

Namespaces

Linux Kernel

- NET
- MOUNT
- USER
- UTS
- PID
It all starts with a **CLONE(2)**

```
clone(2)
```

“Kernel Execution Context”

```
set_ns(2)
unshare(2)
```
MOUNT Namespace

CLONE_NEWNS: Added in 2.4.19 kernel

Per user / via PAM

Per process view of files, disks, NFS
CLONE_NEWIPC: Added in 2.6.19

“System 4 IPC objects”
UTS Namespace

**CLONE_NEWUTS**: Added in 2.6.19

uname(2), setdomainname(2), sethostname(2)
PID Namespace

CLONE_NEWPID: Added in 2.6.24

Process IDs start at 1

Can be nested
PID NS example

$ lxc-create -t busybox -n foo ; lxc-start -n foo

$ lxc-attach -n foo -- ps
PID USER COMMAND
  1 root  init
  5 root /bin/sh
 10 root  ps
Unisys Stealth Solution Suite

You can’t hack what you can’t see. Changing the Security Paradigm.
NETWORK Namespace

CLONE_NEWNET: Added in 2.6.24

Separate network device, IP, MAC, routing table, firewall
USER Namespace

CLONE_NEWUSER: Added in 2.6.23 but finished 3.8

Important for actually securing containers
... also a high risk area of the kernel :/
USER NS example

$ lxc-attach -n foo -- sh

BusyBox v1.21.1 (Ubuntu 1:1.21.0-1ubuntu1) built-in shell (ash) ...
$ id
uid=0(root) gid=0(root)
$ sleep 1337

100000 17110 0.0 0.0 2184 260 pts/14 S+ 12:03 0:00 sleep 1337
Capabilities

TECHNOLOGY
The cause of, and solution to, all of life's problems.
Capabilities

Pros: Kernel devs adding them 😊

Cons: Busy (and lazy) kernel devs 😞

Result: Semi-working capabilities model!
Examples of Capabilities

CAP_NET_ADMIN
CAP_NET_RAW
CAP_NET_BIND_SERVICE
CAP_SYS_RESOURCE
CAP_SYS_PTRACE
CAP_SYS_RAWIO
CAP_KILL
Dropping Capabilities

What should be dropped?

Everything!

What if I leave just “CAP_FOO” enabled?

It depends...
Fixing ping

$ ls -l /bin/ping
-rwsr-xr-x 1 root root 44168 May 7 2014 /bin/ping

$ cp /bin/ping /tmp ; ls -l /tmp/ping
-rwxr-xr-x 1 root root 44168 Mar 18 11:02 /tmp/ping

$ /tmp/ping localhost
ping: icmp open socket: Operation not permitted
Fixing ping

$ sudo setcap cap_net_raw=p /tmp/ping

$ getcap /tmp/ping
/tmp/ping = cap_net_raw+p

$ /tmp/ping localhost
PING localhost (127.0.0.1) 56(84) bytes of data
64 bytes from localhost (127.0.0.1): icmp_seq ...
<table>
<thead>
<tr>
<th>Some Dangerous Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS_CHROOT</td>
</tr>
<tr>
<td>SYS_MODULE</td>
</tr>
<tr>
<td>SYS_RAWIO</td>
</tr>
<tr>
<td>SYS_PTRACE</td>
</tr>
<tr>
<td>MAC_ADMIN</td>
</tr>
<tr>
<td>MAC_OVERRIDE</td>
</tr>
<tr>
<td>DAC_READ_SEARCH</td>
</tr>
</tbody>
</table>
CAP_SYS_ADMIN == root

* Perform a range of system administration operations including: quotactl(2), mount(2), umount(2), swapon(2), swapoff(2), sethostname(2), and setdomainname(2);
* perform privileged syslog(2) operations (since Linux 2.6.37, CAP_SYSLOG should be used to permit such operations);
* perform VM86_REQUEST_IRQ vm86(2) command;
* perform IPC_SET and IPC_RMID operations on arbitrary System V IPC objects;
* perform operations on trusted and security Extended Attributes (see attr(5));
* use lookup_dcookie(2);
* use ioprio_set(2) to assign IOPRIO_CLASS_RT and (before Linux 2.6.25) IOPRIO_CLASS_IDLE I/O scheduling classes;
* forge UID when passing socket credentials;
* perform administrative operations on many device drivers.

* exceed /proc/sys/fs/file-max, the system-wide limit on the number of open files, in system calls that open files (e.g., accept(2), execve(2), open(2), pipe(2));
* employ CLONE_* flags that create new namespaces with clone(2) and unshare(2);
* call perf_event_open(2);
* access privileged perf event information;
* call setns(2);
* call fanotify_init(2);
* perform KEYCTL_CHOWN and KEYCTL_SETPERM keyctl(2) operations;
* perform madvise(2) MADV_HWPOISON operation;
* employ the TIOCSI ioctl(2) to insert characters into the input queue of a terminal other than the caller’s controlling terminal.
* employ the obsolete nfsservctl(2) system call;
* employ the obsolete bdflush(2) system call;
* perform various privileged block-device ioctl(2) operations;
* perform various privileged filesystem ioctl(2) operations;

See False Boundaries and Arbitrary Code Execution post by Spender
https://forums.grsecurity.net/viewtopic.php?f=7&t=2522
Control groups
Hierarchical and inheritable

Controls different subsystems
  (Dev, CPU, Mem, I/O, Network)

ulimit on steroids
Controlling access to resources based on subgroups: devices, CPU, I/O, Mem, ...

Filling some gaps of namespaces
Controlling cgroups is typically performed via a virtual filesystem: /sys/fs/cgroup

Main configuration (besides container configs):
/etc/cgrules.conf, /etc/cgconfig.conf
Container platforms make it easy
Putting that all together...
Putting it all together...

**Namespaces** logically isolate kernel elements.

**Capabilities** help enforce namespaces and reduce undesired privileges.

**Cgroups** limit hardware resources.
Enter: Containers (LXC, Docker, CoreOS rkt, Heroku, Flockport, Kubernetes, Joyant, etc)

Linux Containers

Better than chroot!

Still not virtualization...
Mount options

Beyond ro, nodev, noexec, nosuid

Bind, Overlay, Union, CoW, Versioning, even sshfs
Namespaces, Capabilities and Cgroups: where are they now on Linux servers?

Self-hosted PaaS systems
Amazon EC2
Google App Engine
Rackspace, Heroku
Namespaces, Capabilities and Cgroups: where are they now on Linux clients?

ChromeOS and the Chrome browser
Limited use in Android
Some Linux distros
Sandboxing tools: minijail, mbox
LinuX Containers: **LXC**
lxc.rootfs = /var/lib/lxc/defcon-ctf/rootfs
lxc.utsname = isec
lxc.start.auto = 1
lxc.mount.entry = /lib lib none ro,bind,nodev 0 0
lxc.mount.entry = /lib64 lib64 none ro,bind,noexec 0 0
LXC: Template: Cgroups

lxc.cgroup.tasks.limit = 256
lxc.cgroup.devices.deny = a
lxc.cgroup.devices.allow = b 9:0 r
lxc.cgroup.memory.limit_in_bytes = 4000000
lxc.cap.keep = sys_time sys_nice
lxc.aa_profile = lxc-container-default
lxc.seccomp = /path/to/seccomp.rules
Recent Advancements
Unprivileged Containers

Non-root users can now create/start containers and be “root” inside the container

Weird things can obviously happen

More work and auditing to be done
What about that kernel attack surface?

There are **190 syscalls** in Linux **2.2**
There are **337 syscalls** in Linux **2.6**
There are **340 syscalls** in Linux **4.1**

How many does your app *really* need?
Secomp-bpf

SECure COMPUTing

Filtering the kernel (yet again)

“System call filtering isn't a sandbox. It provides a clearly defined mechanism for minimizing the exposed kernel surface.” – Will @redpig Drewry, Google
Syscall arguments can also be filtered (mostly)

Large number of filters = performance hit

Only really supports x86 and x86_64 (for now)

You’ll need LXC, Minijail or Mbox
(Docker /contrib now, release branch soon (1.8?))
prctl(2) – operations on a process

PR_SET_SECCOMP:

SECCOMP_MODE_STRING (old)
SECCOMP_MODE_FILTER (new hotness)
Seccomp-bpf

```c
struct sock_filter filter[] = {
    BPF_STMT(BPF_LD+BPF_W+BPF_ABS, syscall_nr),
    BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, __NR_ptrace, 1, 0),
    BPF_STMT(BPF_RET+BPF_K, SECCOMP_RET_ALLOW),
    BPF_STMT(BPF_RET+BPF_K, SECCOMP_RET_KILL)
};

struct sock_fprog prog = {((unsigned short) (sizeof(filter) / sizeof(filter[0])), filter );

prct1(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0);
prct1(PR_SET_SECCOMP, SECCOMP_MODE_FILTER, &prog);
```
Berkeley Packet Filter

```
# tcpdump -p -nqi wlan0 -d 'tcp and port 80'

(000) ldh [12]
(001) jeq #0x86dd jt 2 jf 8
(002) ldb [20]
(003) jeq #0x6 jt 4 jf 19
(004) ldh [54]
(005) jeq #0x50 jt 18 jf 6
(006) ldh [56]
(007) jeq #0x50 jt 18 jf 19
(008) jeq #0x800 jt 9 jf 19
(009) ldb [23]

.......
```
Seccomp-bpf: where

ChromeOS / Google Chrome
Firejail     OpenSSH
Capsicum     Tor
Mbox         vsftpd    BIND
LXC          QEMU      Opera Browser
Docker (/contrib)
So who is implementing and supporting containers?

Docker
CoreOS
Flockport
Sandstorm.io
RancherOS

Heroku (ish)
Joyent
Amazon
VMware
Google/Kubernetes

... and many more
Lets talk about the big two
What is the “big deal”

Packaging and deployment focused – one app per container

Devs and Ops, DevOps, DevCyberOps, DevSecOps, BlackOps, etc

Developing PaaS

Makes it easy
So Docker is just LXC? Nope.

libcontainer, libchan, libswarm, etc

Written in go

REST API

Running docker daemon (as root)
Docker Ecosystem

Docker **images**:

```
$ docker run --name mynginx -v \
   /opt/content:/usr/share/nginx/html:ro -d nginx
```

Docker **Hub**:

```
$ sudo docker run ubuntu:14.04 /bin/echo 'Hello world'
Hello world
```

Orchestration, Communication, Management
CoreOS

Minimal OS for hosting containers

Launching the rkt and app container spec

App container spec picked up by VMware Photon

Separation from Docker and LXC
Why Docker, Rocket, etc?

Takes some of the configuration away

FreeBSD::OSX → LXC::Docker

Additional packaged tools | features
Why Docker, Rocket, etc?

**LXC:** You want to run a containerized OS or single app. **Hard mode with the most flexibility.**

**Docker:** You want to run a single app per container. **Easy mode with some costs.**

**CoreOS:** You want to host Docker containers or try and use rkt. **So much bleeding it’s rated R.**
Going on the attack
Lets think about this....

Container to other container
Container to itself
Container to host
Container to support infrastructure
Container to local network
Container to ...
Starting at the top
Starting at the top

I want a cookie.
Give me a cookie,
NOW!
Kernel who?

Lots of drivers, old code, weird file systems, old syscalls, platform specific problems strange or unused network protocols
Not... Dropping caps

If you don’t drop the right ones: **game over**

Not dropping caps also allows kernel code exec... **CAP_NET_ADMIN** (CVE-2013-4588, CVE-2011-2517, CVE-2011-1019, ...)

Not…. Dropping caps

Speaking of dropping capabilities, a Docker shocker: `CAP_DAC_READ_SEARCH`

"Invoke `open_by_handle_at(2)`"

Brute force the inode of `/etc/shadow`

Props to Stealth aka Sebastian Kramer
Without a MAC system, capability dropping and the user namespace are your only line of defense.
Not.... Limiting access

Procfs: /proc/kcore,
        /proc/sys/modprobe,
        /proc/sys/kernel/sysrq

Sysfs: /sys

Cgroups does not limit: mknod

Kernel ring buffer: dmesg

Network access: br0

Unintended devfs: /dev, /dev/shm
Not... Limiting resources

Forkbomb! :(){ :|:& };:

Memory, disk, entropy...
When was the **last time you updated** OpenSSL **in** your Docker container?

How do you deal with *updates in place* if apt-get upgrade is a “no-no”?
“The flawed assumption of modern computing environments”

Eggs in one (kernel) basket

AppArmor does a decent job
LXC Weaknesses

**Bad defaults:** Capability dropping, networking,

Unprivileged containers finished-ish

A few security fixes have lagged :/
Docker Weaknesses

Capability dropping: a shocker

Root daemon plus root to use it

Weak REST API authentication defaults

Docker “github all the way down”
Docker Weaknesses

Does not drop all capabilities by default, drops all except “those needed” (still includes some dangerous capabilities CAP_NET_RAW, CAP_FOWNER, CAP_MKNOD, …)

Docker binds container port maps to all interfaces by default

Base images are huge… apt-get is hungry

Docker networking defaults allow cross-container networking and access to Docker host
Docker Weaknesses

Giving low rights users access to Docker means giving them root on the Docker host

Currently missing support for key security features: seccomp-bpf and the User Namespace

Exposing the socket/REST API inside a container for introspection <- don’t do that
Hi all, I'm a maintainer of Docker. As others already indicated this doesn't work on 1.0. But it could have. Please remember that at this time, we don't claim Docker out-of-the-box is suitable for containing untrusted programs with root privileges. So if you're thinking "pfew, good thing we upgraded to 1.0 or we were toast", you need to change your underlying configuration now. Add apparmor or selinux containment, map trust groups to separate machines, or ideally don't grant root access to the application. Docker will soon support user namespaces, which is a great additional security layer but also not a silver bullet! When we feel comfortable saying that Docker out-of-the-box can safely contain untrusted uid0 programs, we will say so clearly.
Posted one year ago :/
CoreOS “rkt” Weaknesses

Rocket (rkt) is extremely new

No root daemon but rkt still requires root…
CoreOS “rkt” Weaknesses

Rocket does **not drop many dangerous Capabilities or support the User namespace**
CoreOS “rkt” Weaknesses

Seccomp ? **Nope.**

Apparmor ? **Nope.**

SELinux? **Kinda.**

Root inside container? **Yep.**

/proc, /proc/sys limits? **Nope.**
The Dream
The Implementation
Open Container Project (OCP)

Robert 'Bob' Morton: At Security Concepts, we're projecting the end of crime in Old Detroit within forty days.

There's a new guy in town. His name is RoboCop.
Open Container Initiative (OCI?)

Working on a joint specification (OCF) for containers

Launched **runc**. An OCF implementation using libcontainer from Docker.

Unfortunately still not working on **RoboCop**.
That all sounds bad/easy to mess up

... and how to make it better
Recommendations
Kernel Hardening

Grsecurity/PaX is the only serious kernel hardening patchset. Just do it

Typical sysctl hardening

Minimal kernel modules
Dropping all the Capabilities

Gotta drop them all!

Design for the smallest set

Assume the worst
Adding a MAC Layer

AppArmor
Grsecurity RBAC

SMACK
SELinux
Defaults to enabled for LXC and Docker!

Can be nested!

Path based, but hey it works
Docker Specific Hardening

Don’t allow access to docker user or group

Don’t run privileged or root containers

Drop additional capabilities

Upgrade to 1.8 when released (or use /contrib now) which has seccomp-bpf and User namespace support, w00t!

Checkout docker-bench-security and other solid work by Docker Security team

Use small base images
Use a whitelist if you can but a blacklist will do OK

Docker is exploring a “high”, “med”, “low” default for 1.8+ but what is really needed is profiles for each Containerized app.
Mount security, Extended filesystem attributes, Access controls, Permissions, Logging, Firewalls, Auditing, Hardened Toolchain, Safe languages, Attack surface reduction, Least privileges, Least Access, Resource Limits, 2FA, Reduced Complexity, Pentesting
Listening on “all interfaces” (includes docker0/lxcbr0)

Containers are great for network auditing/traceflow!
**Trust A**

- Hardened application
- User namespace w/o caps
- Mount protections
- Minimal container distro
- Syscall Filtering w/ seccomp-bpf
- Linux kernel with grsecurity+pax
- HYPERVISOR/HARDWARE

**Trust B**

**Trust C**

**Trust D**
Where do we go from here?
Where do we go from here?

More namespaces (proc, dev)

Minimal hypervisors (ClearContainers)

Minimal container distros

Android or other non-x86 that needs app/system segmentation/sandboxing
Where do we go from here?

“Desktop” applications in containers

Improved seccomp-bpf argument filtering

Hopefully more granular capabilities

….. more vulnerabilities too! :/
Where do we go from here?

Microservices
Where do we go from here?

POST /user

```json
{
    emailAddress: "foo@bar.com",
    password: "xyz"
}
```

User Registration Service

MongoDB

NewUserNotification

RabbitMQ

Exchange

Queue

Queue

User Management Service

Email Service
Where do we go from here?
Conclusion
In closing

It’s not about perfect security but improving the current state and making attackers work harder.

The technologies to support containers can be used to help secure existing non-container Linux systems.

Microservices architecture fits a least-privilege and least-access container/security model.

Physically separate critical security barriers and isolate by trust.
Coming soon!

My whitepaper: "Understanding and Hardening Linux Containers"…

Covers everything here in muuuch more depth!

(background, namespaces, all the capabilities, cgroups, explores MAC, seccomp-bpf, past container attacks, overall and specific weaknesses, security recommendations for LXC, Docker, rkt deployments)
Coming soon!

**When** will the whitepaper be released?

Hopefully in the **next few weeks**!

**How** can I make sure I get it?

Email me! or follow me on Twitter! @dyn___ (totally not a ploy for more followers)
Thanks!

Any Questions/Comments?

Aaron.Grattafiori@nccgroup.trust
https://twitter.com/@dyn___