VULNERABILITY EXPLOITATION IN DOCKER CONTAINER ENVIRONMENTS

ANTHONY BETTINI, FOUNDER & CEO, FLAWCHECK
ABETTINI@FLAWCHECK.COM

Presented at Black Hat Europe 2015

INTRODUCTION

Containers have been around for a long time. But only recently, have container-based virtualization solutions become commonplace within the enterprise. Docker in particular is everywhere. But why? And what does it mean for enterprise security? Is vulnerability exploitation of Docker containers any different from vulnerability exploitation of application vulnerabilities on virtual machines? Are the ways to secure them any different?

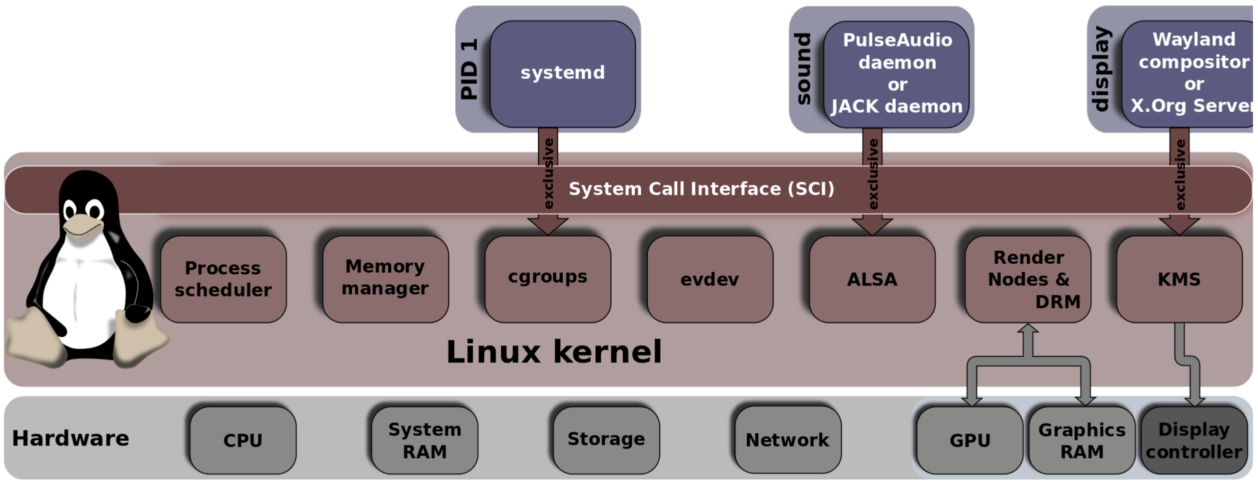
Before we jump into the vulnerability exploitation piece of the equation, it makes sense to review the security provided by the container solutions themselves – namely the workload isolation security. As with any new topic, it makes sense to start with a bit of history. How did we even get here?

MODERN HISTORY OF LINUX CONTAINERS

Today, Docker is the most widely used container-based virtualization technology. But Docker itself is an application (technically, a daemon), built on the container technology provided by the Linux kernel. The container technology provided by the Linux kernel isn't new though, it has been evolving over time, for a very long time.

Linux container technology is generally accepted to trace back to the days of chroot. Chroot was introduced way back in 1979 and started to address the isolation problem. Chroot, or "*ch*ange *root*" changes the view of the file system for the process and its children. This was particularly useful for applications such as ftpd, to restrict the view of the ftp client to subfolders of the chroot'd parent. But chroot itself wasn't built for security and can be easily subverted. For instance, there exists lots of sample code on the Internet for escaping from chroot jails[1].

After chroot, the next extremely relevant container precursor technology that was introduced to the Linux kernel was the introduction of cgroups (**c**ontrol **groups**). cgroups provide a way to partition sets of tasks in Linux.



In 2006, engineers, primarily from Google, started submitting the cgroups code changes so that Google could better isolate workloads. cgroups originally were deemed "process containers", but due to containers being an overloaded term, the team went with control groups or cgroups, as the name. An early, but still very relevant, paper on the cgroups implementation can be found at: [2]. cgroups reached mainstream audiences with the introduction of Linux kernel 2.6.24 in 2007. Today, cgroups serves as the isolation basis for at least systemd, CoreOS, Docker, lmctfy, and LXC.

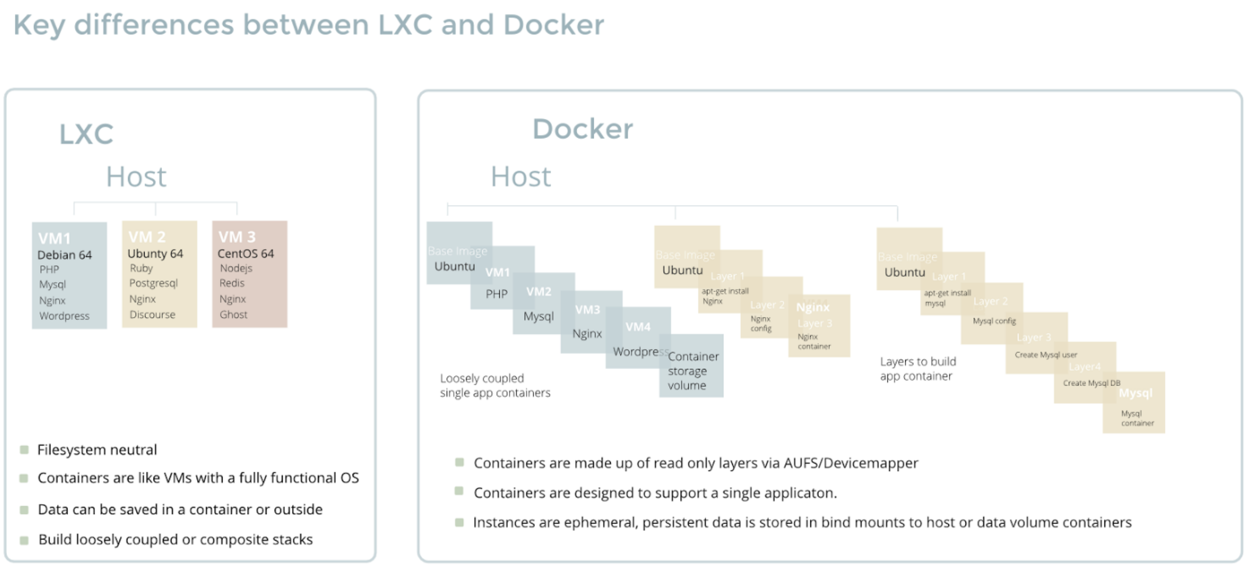
Just a year later, in 2008, LXC (Linux Containers) were introduced. LXC provides interfaces to all of the kernel containment features, such as:

- Kernel namespaces
- cgroups
- Apparmour & SELinux
- Policies

[1] https://gist.github.com/FiloSottile/6976188
[2] https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt

Docker was released years later, in 2013, and stemmed from dotCloud's attempt to isolate customer workloads from each other, using Linux container-based virtualization. Solomon Hykes, then at dotCloud and now the CTO of Docker, announced Docker via his lightning talk at PyCon 2013[3].

If LXC was released in 2008 and Docker in 2013, why did Docker receive so much (justified) hype? In the case of LXC, containers were running their own operating systems and were not much different than virtual machines.



- Filesystem neutral
- Containers are like VMs with a fully functional OS
- Data can be saved in a container or outside
- Build loosely coupled or composite stacks

- Containers are made up of read only layers via AUFS/Devicemapper
- Containers are designed to support a single applicaton.
- Instances are ephemeral, persistent data is stored in bind mounts to host or data volume containers

With the release of Docker, developers could create & run application containers very quickly. With the release of Docker Hub, developers could download & run application containers even quicker.

At this point in time, the Linux kernel provides 6 namespaces:

1. mnt (filesystems & mount points)
2. PID (processes)
3. net (network stack)
4. UTS (hostname)
5. IPC (Linux implementation of System V IPC)
6. user (user)

[3] https://www.youtube.com/watch?v=wW9CAH9nSLs

A Linux namespace abstracts a global resource and makes it appear to processes within the namespace that they have their own isolated instance of the global resource.

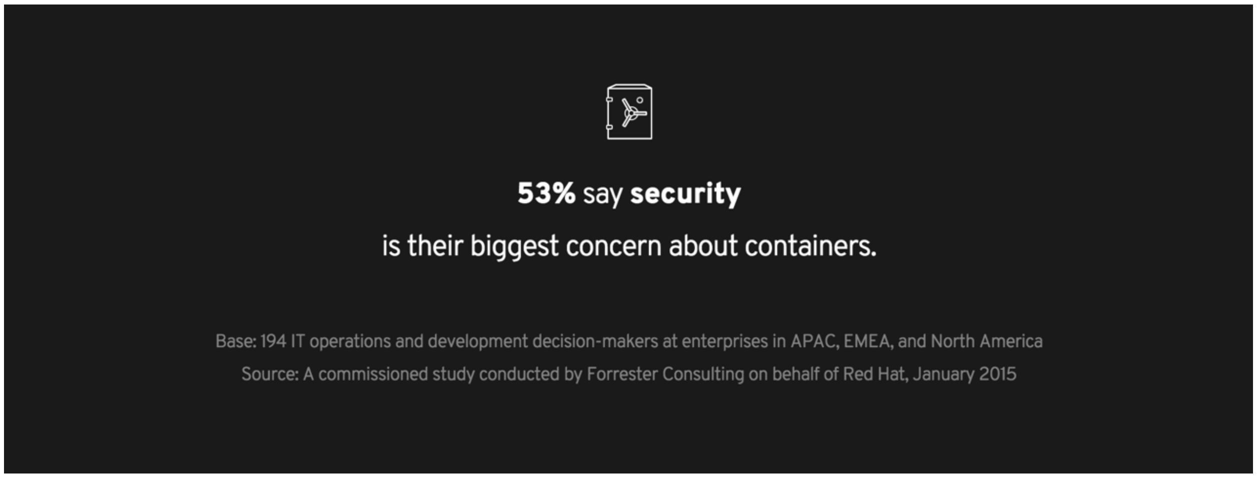
The newest of these namespace types is user namespaces. User namespaces were introduced in Linux kernel 3.8 in 2013. Consider[4]:

```
$ id -u           # Display effective user ID of shell process
1000
$ id -g           # Effective group ID of shell
1000
$ ./demo_userns
eUID = 65534;  eGID = 65534;  capabilities: =ep
```

Up until recently Docker did not fully implement user namespaces, which had security implications on both the application developers, the operations team, and the cloud service provider. As of late October 2015, Docker has now made user namespace support available.

STATE OF THE UNION: CONTAINERS IN THE ENTERPRISE

In January 2015, Red Hat commissioned Forrester to survey enterprises about why they weren't running containers in production. 53% of enterprises reported their bigger concern about containers was security.



53% say **security**
is their biggest concern about containers.

Base: 194 IT operations and development decision-makers at enterprises in APAC, EMEA, and North America
Source: A commissioned study conducted by Forrester Consulting on behalf of Red Hat, January 2015

[4] https://lwn.net/Articles/532593/

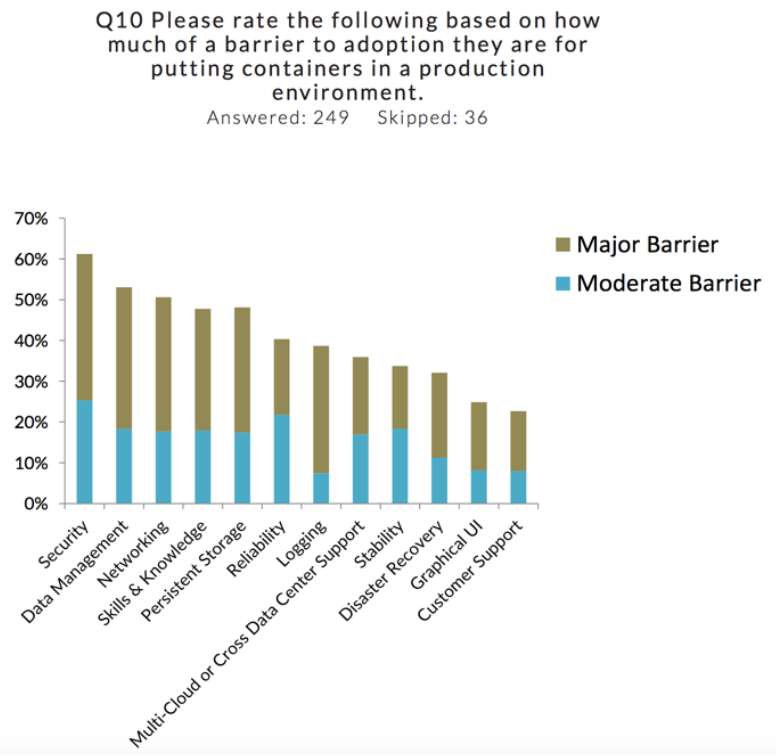
In June 2015, ClusterHQ asked enterprises “What are the biggest barriers to putting containers in a production environment?” This time, an even higher percentage of enterprises (>60%) said that security was the #1 barrier to putting containers in a production environment.

WHAT ARE THE BIGGEST BARRIERS TO PUTTING CONTAINERS IN A PRODUCTION ENVIRONMENT?

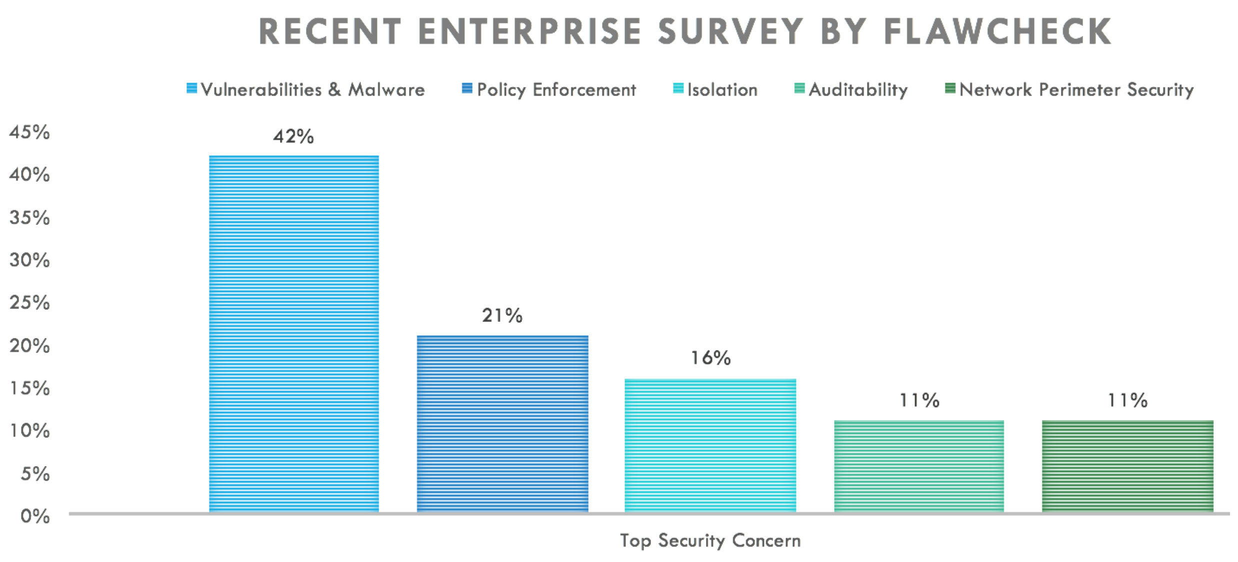
In this question respondents had the option of rating certain categories as a major barrier, moderate barrier, minor barrier or no barrier at all.

Security was the highest rated barrier to increased adoption. The second biggest barrier was data management.

Note: we combined the major and moderate barrier responses and grouped them to weigh biggest barriers.



In August 2015, FlawCheck and one of our partners, surveyed enterprises asking which piece of the security equation was their top concern about running containers in production environments.

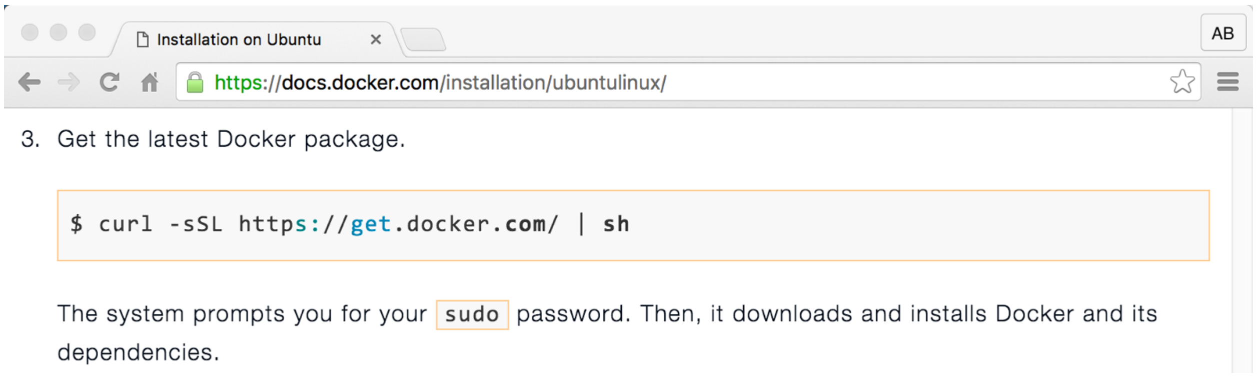


At 42%, Vulnerabilities & Malware in container workloads was the top container security concern among those surveyed.

VULNERABILITIES

Vulnerability exploitation isn't always about buffer overflows & ROP chains. As we've seen with cryptography[5], implementation weaknesses (ultimately, design imperfections) often negatively impact the security of the larger system.

To begin looking at vulnerabilities in the Docker ecosystem, let's start by installing Docker:



3. Get the latest Docker package.

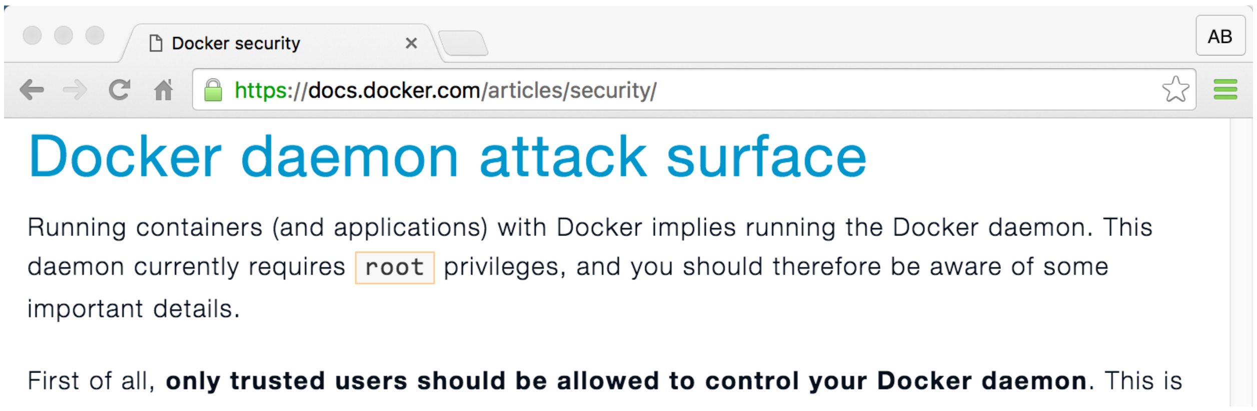
```
$ curl -sSL https://get.docker.com/ | sh
```

The system prompts you for your `sudo` password. Then, it downloads and installs Docker and its dependencies.

5 https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf

Downloading code without looking at it, that changes often and lacks any integrity check, then piping it to an interpreter and executing it as root (via putting your password into sudo, hopefully), is a fail. At least curl validates the certificate and get.docker.com doesn't support DHE or export ciphers, but still it's an awful workflow from a security perspective.

After Docker is installed, you'll realize that it's actually a daemon that runs as root:

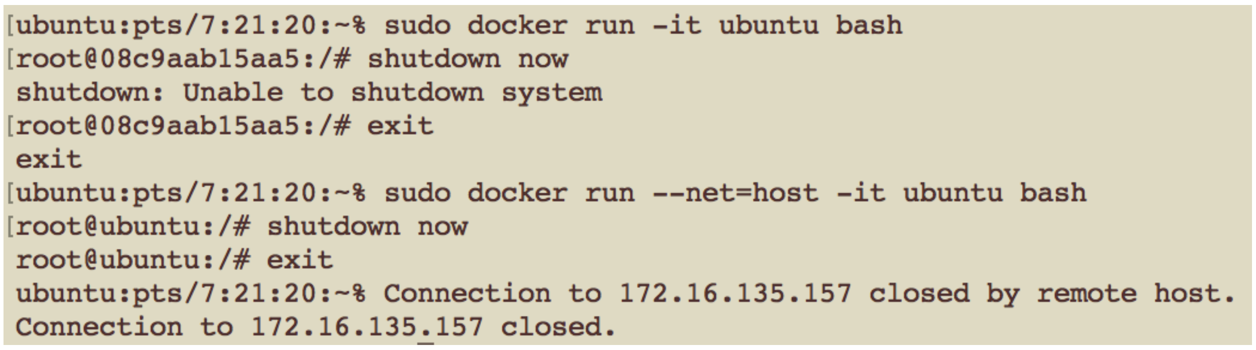


Docker daemon attack surface

Running containers (and applications) with Docker implies running the Docker daemon. This daemon currently requires `root` privileges, and you should therefore be aware of some important details.

First of all, **only trusted users should be allowed to control your Docker daemon**. This is

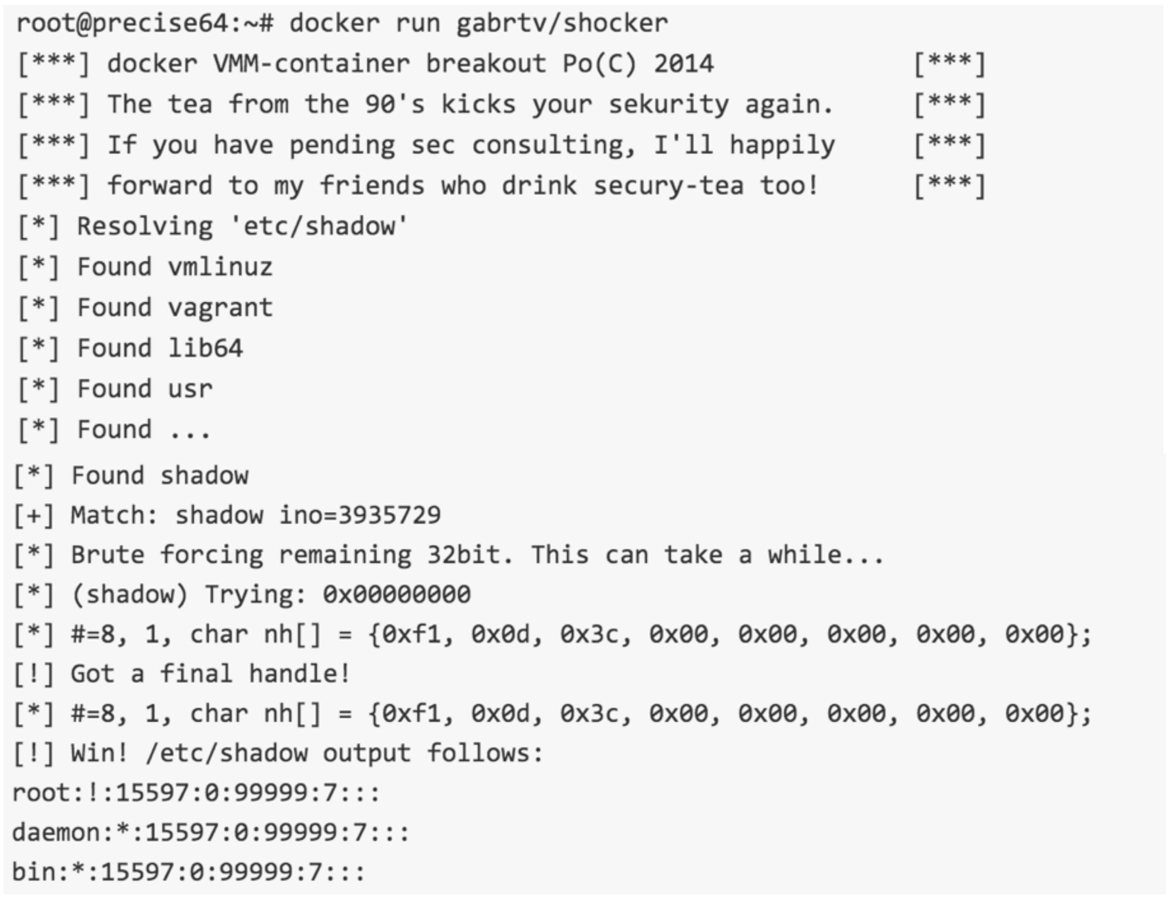
In the event your PaaS is starting Docker with the incorrect parameters, such as host networking, users can actually shutdown the container host!



```
[ubuntu:pts/7:21:20:~% sudo docker run -it ubuntu bash
[root@08c9aab15aa5:/# shutdown now
 shutdown: Unable to shutdown system
[root@08c9aab15aa5:/# exit
 exit
[ubuntu:pts/7:21:20:~% sudo docker run --net=host -it ubuntu bash
[root@ubuntu:/# shutdown now
 root@ubuntu:/# exit
 ubuntu:pts/7:21:20:~% Connection to 172.16.135.157 closed by remote host.
 Connection to 172.16.135.157 closed.
```

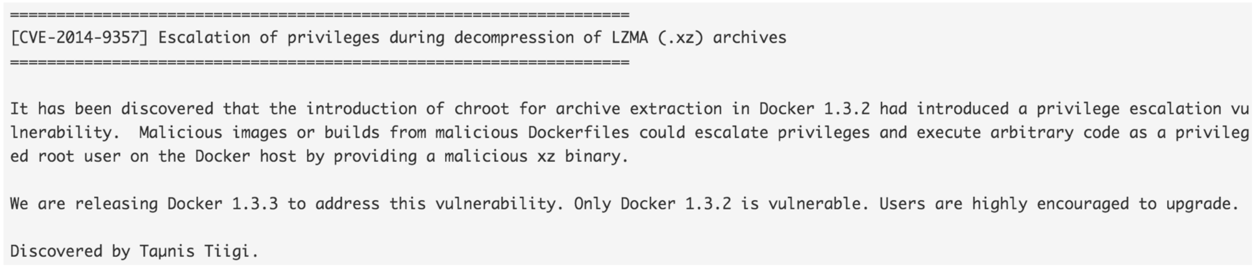
Docker does actually provide a warning message against this and in practice, it's easy to avoid, but enabling host networking has surprising consequences.

In an older release of Docker, Docker actually blacklisted kernel calls (remember Docker is basically acting as a man-in-the-middle between the container and the kernel). By blacklisting kernel calls, Docker missed an important one, CAP_DAC_READ_SEARCH, allowing users to break out of the isolation of Docker containers and be on the container host itself. The Shocker code was released to demonstrate this exploit:



```
root@precise64:~# docker run gabrtv/shocker
[***] docker VMM-container breakout Po(C) 2014          [***]
[***] The tea from the 90's kicks your sekurity again.   [***]
[***] If you have pending sec consulting, I'll happily   [***]
[***] forward to my friends who drink secury-tea too!    [***]
[*] Resolving 'etc/shadow'
[*] Found vmlinuz
[*] Found vagrant
[*] Found lib64
[*] Found usr
[*] Found ...
[*] Found shadow
[+] Match: shadow ino=3935729
[*] Brute forcing remaining 32bit. This can take a while...
[*] (shadow) Trying: 0x00000000
[*] #=8, 1, char nh[] = {0xf1, 0x0d, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x00};
[!] Got a final handle!
[*] #=8, 1, char nh[] = {0xf1, 0x0d, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x00};
[!] Win! /etc/shadow output follows:
root:!:15597:0:99999:7:::
daemon:*:15597:0:99999:7:::
bin:*:15597:0:99999:7:::
```

Numerous CVEs have been reported against Docker as well. The most interesting of which is likely CVE-2014-9357, which was a privilege escalation via exploiting a bug in the parsing of XZ files. Docker supports a variety of decompression algorithms (such as XZ, GZ, and TAR). This is likely the highest ROI attack vector for privilege escalation against Docker (at least for now).



```
=================================================================
[CVE-2014-9357] Escalation of privileges during decompression of LZMA (.xz) archives
=================================================================

It has been discovered that the introduction of chroot for archive extraction in Docker 1.3.2 had introduced a privilege escalation vu
lnerability.  Malicious images or builds from malicious Dockerfiles could escalate privileges and execute arbitrary code as a privileg
ed root user on the Docker host by providing a malicious xz binary.

We are releasing Docker 1.3.3 to address this vulnerability. Only Docker 1.3.2 is vulnerable. Users are highly encouraged to upgrade.

Discovered by Taµnis Tiigi.
```

Shellshock is also particularly relevant for Docker containers as most Docker containers contain bash. So there exists a chance the container is exploitable if the running process in the container leverages it. (And old copies of bash are particularly common).

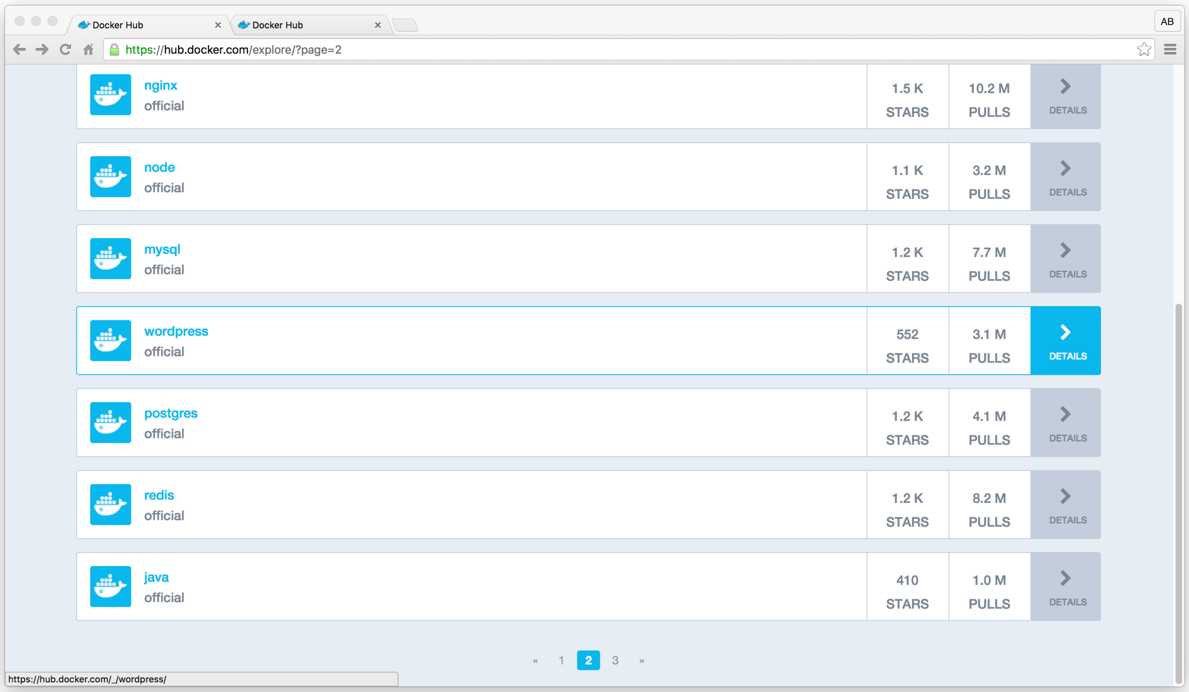
Recently there was also the first reported case[6] of a Docker container breached in the wild using CVE-2014-3120 (an Elasticsearch vulnerability). There is a MetaSploit plugin[7] for this available as well.

TEARING APART CONTAINERS

So what about containers in the real world? On Docker Hub, there are over 15,000+ pre-built containers ready to be downloaded (and potentially deployed). Further, Docker Hub boasts over 500 million downloads of those containers. But are the containers Docker Hub hosts, actually safe for enterprises to deploy in production? A previous study did a random sampling of the containers on Docker Hub and found over 30% contained vulnerabilities. But Docker Hub actually has two different types of containers. First, there is the general population of containers. Second, there are the "Docker Official Images", shown below:

[6] https://www.youtube.com/watch?v=EJ9ey6-DbGM

[7] https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/multi/elasticsearch/script_mvel_rce.rb



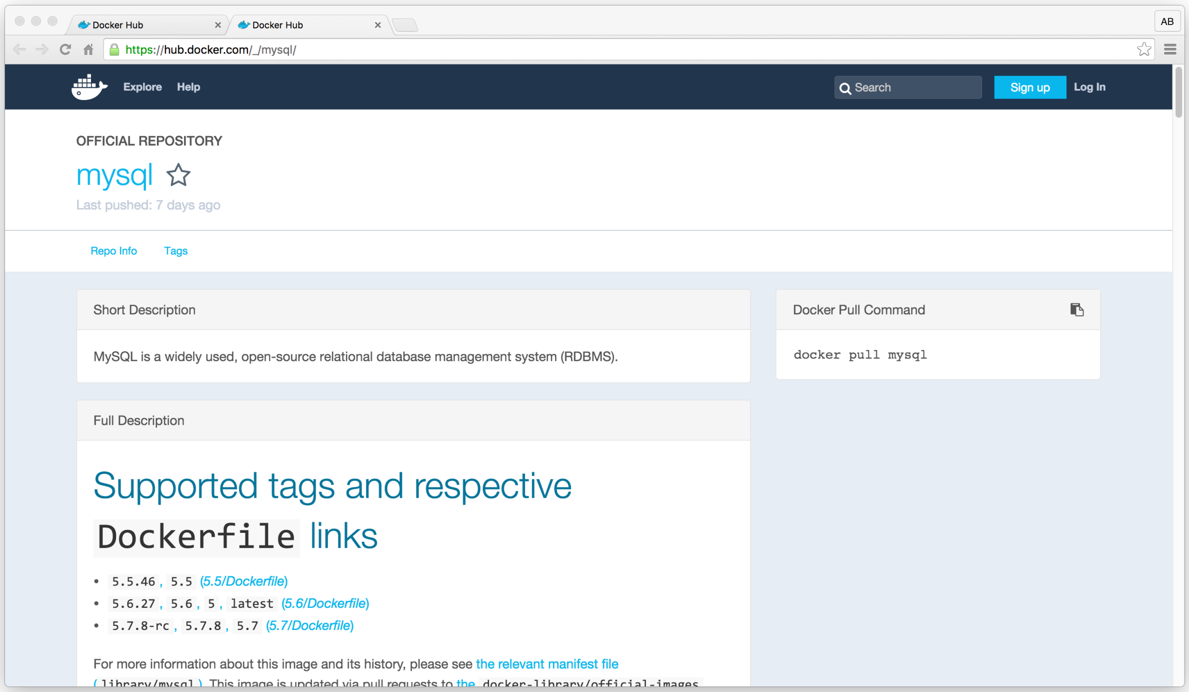
https://hub.docker.com/explore/?page=2

nginx	official	1.5 K STARS	10.2 M PULLS	DETAILS
node	official	1.1 K STARS	3.2 M PULLS	DETAILS
mysql	official	1.2 K STARS	7.7 M PULLS	DETAILS
wordpress	official	552 STARS	3.1 M PULLS	DETAILS
postgres	official	1.2 K STARS	4.1 M PULLS	DETAILS
redis	official	1.2 K STARS	8.2 M PULLS	DETAILS
java	official	410 STARS	1.0 M PULLS	DETAILS

« 1 2 3 »

https://hub.docker.com/_/wordpress/

An example Docker Official Repository is “mysql”, which hosts (at time of writing) ~9 pre-built containers, shown below:



OFFICIAL REPOSITORY

mysql

Last pushed: 7 days ago

Short Description

MySQL is a widely used, open-source relational database management system (RDBMS).

Docker Pull Command

```
docker pull mysql
```

Full Description

Supported tags and respective `Dockerfile` links

- `5.5.46`, `5.5` (5.5/Dockerfile)
- `5.6.27`, `5.6`, `5`, `latest` (5.6/Dockerfile)
- `5.7.8-rc`, `5.7.8`, `5.7` (5.7/Dockerfile)

For more information about this image and its history, please see the relevant manifest file (`library/mysql`). This image is updated via pull requests to the `docker-library/official-images`

We used FlawCheck to inspect the *latest* available container images for all of the Docker Official Repositories (slightly under 100 repositories at time of writing). FlawCheck found >90% of container images contained vulnerabilities. For instance, the mysql container image above contained glibc and bash vulnerabilities, among others.

Certainly, as container images host a lot of code and may only have one running process, it does not mean that >90% of container images contain ***exploitable*** vulnerabilities. But as we came to see the affect of bash vulnerabilities affecting network processes, it certainly makes sense to both measure the risk & impact of vulnerabilities present in containers, and eliminate (by rebuilding containers or taking other mitigating actions) network facing vulnerabilities from being exploited. Further, non-network facing vulnerabilities should also be remediated to reduce the attack surface.

CONCLUSION

DevOps is more than a job description change. When enterprises begin embracing DevOps personnel, roles, and teams, they (typically, unconsciously) put at least some level of operational control in the developer's hands. This is particularly true of container-based virtualization environments, because instead of a developer passing an application to an operations engineer to deploy to a host or VM provisioned by the Ops team, a developer is self-provisioning the application in the container. This is a fundamental change that requires process, workflow, and security changes to the enterprise. From a technical perspective, containers contain applications, which are subject to normal application security controls (or should be) and if containers are coming from a third-party resource, such as Docker Hub, and are pre-built, have you checked what they may contain?

Docker Hub is similar to the early days of the Android Market. Fast forward a few years and we see Google Play plagued with malware and Google performing automated analysis to try to weed it out before millions are infected – and a plethora of cybersecurity companies helping protect enterprises and consumers from the risks of apps. With Docker Hub and containers, will we see history repeat itself?

ABOUT ANTHONY BETTINI

Anthony Bettini is the Founder & CEO of FlawCheck, the leader in industrial-grade container security. Prior to FlawCheck, Anthony was the founding CEO of Appthority, the leader in mobile app security, SINET 16 award winner, and winner of the "Most Innovative Company of the Year" award at RSA Conference 2012. His security experience comes from leadership roles in companies such as Appthority, Intel, McAfee, Foundstone, Guardent, Bindview, and Netect. He specializes in growing and advising early-stage enterprise security companies and innovative security research. Anthony's presentations have been delivered at conferences such as Black Hat, RSA, AVTokyo, FIRST, SINET, SyScan, the OpenStack Summit, the CARO Workshops, and many others. In addition to contributions to Cybersecurity books, Anthony was the technical editor for Hacking Exposed, the best-selling Cybersecurity book of all time, which has been used in courseware at universities such as Harvard and MIT. Recently, Anthony reported on the greatest risks in mobile apps in In-Q-Tel's quarterly publication.

ABOUT FLAWCHECK

FlawCheck's container cybersecurity protection platform helps organizations understand the risks present in Linux Containers and take action to protect them. By detecting and taking action against the top cybersecurity threats affecting container environments, FlawCheck is able to help organizations adopt containers in production environments securely. FlawCheck is headquartered in San Francisco. For more information, visit https://www.flawcheck.com/ or follow on Twitter: @FlawCheck