Session Goals

Considerations for Security on a Container Platform

Containers - A quick look

Container Security at Host Operating System

Container Security at Build time

Container Security at Runtime

Container Platform (K8S Cluster) - Additional Security Features

Application Security
Understanding Containers
Malicious Containers
Bad processes choking others
Processes stepping over each other
Container separation

- Containers all share the same kernel
- Containers security is about controlling the access to the kernel
- Limit the ability of the container to negatively impact
  - the infrastructure
  - other containers
Container Security
Importance of Host OS
Linux Containers Architecture
Isolation with Linux Namespaces

**PID namespace**
The first process is pid 1 in the container. Namespace destroyed as PID exits.
Cannot see/signal processes in parent pid namespace or other pid namespaces

**Network namespace**
Allows container to use separate virtual network stack, loopback device and process space.
veth pairs are created to add interfaces from initial network namespace to container network namespace

**Mount namespace**
Isolate the set of file system mount points. Mount sys directories as read only and nodev
Each process gets its own mount table i.e. view of the filesystem

**UTS namespace**
Isolate system identifiers – hostname and domain name

**IPC namespace**
Isolate certain interprocess communication (IPC) resources

**User namespace**
Allow you to specify a range of host UIDs dedicated to the container. A process can have full root privileges for operations inside the container mapped to non-root user on host.
Linux CGroups

Ensures that a single container cannot exhaust a large amount of system resources
CGroups allocate CPU time, system memory, network bandwidth, or combinations of these among user-defined groups of tasks

In a container:
# ls /sys/fs/cgroup/
blkio  cpu  cpuacct  cpuacct,cpu  cpuset  devices  freezer  hugetlb  memory  net_cls
  net_prio  net_prio,net_cls  perf_event  pids  systemd

# cat /sys/fs/cgroup/cpu/cpu.shares
2

# cat /sys/fs/cgroup/memory/memory.limit_in_bytes
9223372036854771712

Device CGroups
- specify which device nodes can be used within the container
- blocks the processes from creating and using device nodes that could be used to attack the host.
SELinux and Type Enforcement

- Implements Mandatory Access Control
- A LABELING system
- Every process has a LABEL. Every file, directory, and system object has a LABEL
- Policy rules control access between labeled processes and labeled objects
- The kernel enforces the rules.
- **Type Enforcement protects the host system from container processes**

All container processes run with type `svirt_lxc_net_t`
Content within the container is labeled with type `svirt_sandbox_file_t`

`svirt_lxc_net_t`
- allowed to manage any content labeled `svirt_sandbox_file_t`
- able to read/execute most labels under `/usr` on the host
- **not allowed** to open/write to any other labels on the system
- **not allowed** to read any default labels in `/var`, `/root`, `/home` etc
Multi Category Security (MCS) Enforcement

*MCS protects Containers from each other*

Based on Multi Level Security (MLS)

Docker Daemon picks out unique random MCS Label $s0:c1,c2$
Assigns MCS Label to all content
Launches the container processes with same label
Container Processes can only read/write their own files - Kernel enforces this
Linux Capabilities

Kernel 2.2 divides root privileges into 32 distinct capabilities

- CAP_SETPCAP: Modify process capabilities
- CAP_SYS_MODULE: Insert/Remove kernel modules
- CAP_SYS_RAWIO: Modify Kernel Memory
- CAP_SYS_PACCT: Configure process accounting
- CAP_SYS_NICE: Modify Priority of processes
- CAP_SYS_RESOURCE: Override Resource Limits
- CAP_SYS_TIME: Modify the system clock
- CAP_SYS_TTY_CONFIG: Configure tty devices
- CAP_AUDIT_WRITE: Write the audit log
- CAP_AUDIT_CONTROL: Configure Audit Subsystem
- CAP_MAC_OVERRIDE: Ignore Kernel MAC Policy
- CAP_MAC_ADMIN: Configure MAC Configuration
- CAP_SYSLOG: Modify Kernel printk behaviour
- CAP_NET_ADMIN: Configure the network:
  - Setting the hostname/domainname
  - mount(), unmount()
  - nfsservctl
  - ....

Root user with all capabilities is all powerful!!
Do really think containers need these default capabilities in production??

A good strategy is to drop all capabilities and just add the needed ones back. Example

```
# docker run -d --cap-add SYS_TIME ntpd
```
Secure Computing Mode - Seccomp

Filters system calls from a container to the kernel. Uses Berkeley Packet Filter (BPF) system.

Children to a container process will inherit the BPF filter.

Provides more fine-grained control than capabilities, giving an attacker a limited number of syscalls from the container.

Restricted and allowed calls are arranged in profiles, pass different profiles to different containers.

Specify your own policy.

```
# docker run --security-opt seccomp=/path/to/custom/profile.json <container>
```
Read Only Mounts

Linux kernel file systems have to be mounted in a container environment or processes would fail to run.

Fortunately, most of these file systems can be mounted as "read-only" on RHEL.
/sys
/proc/sys
/proc/sysrg-trigger
/proc/irq
/proc/bus

OpenShift also blocks the ability of the privileged container processes from re-mounting the file systems as read/write.
Container Build Time Security
Towering Container Content

Image governance and private registries

- What security meta-data is available for your images?
- Are the images in the registry updated regularly?
- Are there access controls on the registry? How strong are they?

- Red Hat Container Registry
- Policies to control who can deploy which containers
- Certification Catalog
- Trusted content with security updates
Trusted Container Content

Trusted containers from known sources that provide certified and supported images. Example Red Hat provides Trusted Container Images (registry.redhat.io) as part of OpenShift Service Catalog.

Publish fixes to the content in this registry and notify you.

Languages: PHP, Python, Ruby, Perl, Node.js, Java, .Net Core

Enterprise Grade JBoss Middleware: EWS, EAP, BPM, BRMS, RH SSO, Data Grid, DataVirt, 3Scale

Databases: MySQL, Mongo, PostgreSQL, Maria

CICD: Jenkins

Partners Images: Container Certification by Red Hat.
Container Health Index

The following grades and icons are used with a brief explanation of how they are calculated.

- **Grade A**: This image does not contain known unapplied errata that fix Critical or important flaws.
- **Grade B**: This image may be missing Critical or important security errata, but no missing Critical flaw is older than 7 days and no missing important flaw is older than 30 days.
- **Grade C**: This image may be missing Critical or important security errata, but no missing Critical flaw is older than 30 days and no missing important flaw is older than 90 days.
- **Grade D**: This image may be missing Critical or important security errata, but no missing Critical flaw is older than 90 days and no missing important flaw is older than 365 days.
- **Grade E**: This image may be missing Critical or important security errata, but no missing Critical or Important flaw is older than 365 days.
- **Grade F**: This image may be missing Critical or important security errata, and they are older than 365 days. Or the container is out of its lifecycle.
- **Grade Unknown**: This image cannot be scanned as it is missing metadata required to perform the Container Health Index calculation.
Container Image Provenance

- Did the image come from a trusted source?
- Are you verifying image signatures?
- Are you signing container images?
- Does your Container Platform allow you to sign images?

Example: Image signing on OpenShift Container Platform

```bash
$ oc adm policy add-cluster-role-to-user system:image-signer <user_name>
$ atomic push [--sign-by <gpg_key_id>] --type atomic <image>
```

Example: Image verification on OpenShift Container Platform

```bash
$ oc adm policy add-cluster-role-to-user system:image-auditor <user_name>
$ oc adm verify-image-signature sha256:2bba968aedb7dd2aafe5fa8c7453f5ac36a0b9639f1bf5b03f95de325238b288 --expected-identity 172.30.1.1:5000/openshift/nodejs:latest --public-key /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release --save
```
Private Registries

Enterprise security policies may not allow your Container Images to be pushed to a registry outside.

Or what if public registry (like DockerHub) is down?

Caching images in Private Registry

- Red Hat’s Quay
- JFrog Artifactory
- Docker Trusted Registry
Restrict Registry Sources

- Restrict pulling images from specific registries of your choice

Example: Restricting Container Registries in OpenShift

```yaml
- name: allow-images-from-internal-registry
  onResources:
    - resource: pods
    - resource: builds
      matchIntegratedRegistry: false

- name: allow-images-from-dockerhub
  onResources:
    - resource: pods
    - resource: builds
      matchRegistries:
        - docker.io
```
Container Image Management Responsibilities

Operations/ Infrastructure admin

- Maintain Trusted OS Base Images (RHEL, RHEL-Atomic)
- Ensure these are good with Linux Kernel

Middleware Engineers/Architects

- Maintain Middleware Images
- Control build process - S2I, CICD
- Reference Architectures with Trusted middleware

Development teams

- Write code that layers on approved images
- Ensure code is clean, open source software vulnerabilities are handled
Securing Pipelines and Runtime
Container Scanning

Frequency

Scan containers as soon as they are created

Scan containers that get into enterprise registry

Ongoing basis- Identify any new vulnerabilities
Scanning Containers at Creation

Run image scan at this point
Scanning Tools

Atomic Scan - configurable with different scanners
Claire Scanner
OpenScap Scanner
BlackDuck
JFrog XRay
Twistlock
Additional Container Platform Security Features
API Authentication

• OAuth Access Token
  – Obtained from OAuth server using endpoints
    <master>/oauth/authorize          <master>/oauth/token
  – Sent as
    Authorization: Bearer access_token=…

• X.509 Client Certs
  – HTTPS connection
  – API Server verifies against a trusted cert authority bundle
  – API Server creates and distributes certs to Controllers to authenticate themselves

• Request with invalid token or cert gets a 401
• No cert or token gets system:anonymous user and system: unauthenticated virtual group
Authentication

Client Types:
- openshift-web-console
- openshift-browser-client
- openshift-challenging-client

Authentication Requests:
- <master>/oauth/authorize
- WWW-authenticate
- <master>/oauth/token/request

Identity Provider

Configured Identity Providers:
- LDAP
- GitHub
- GitLab
- Google
- OpenID Connect
- HTPasswd
- Keystone
Authorization

RoleBinding associates Users/Groups with Roles
- Cluster RoleBindings
- Project RoleBindings

In Kubernetes, containers in pods run with the role assigned to a special user called Service Account
Container Deployment Permissions with Pod Security Policies (PSP) or with SCC

```bash
[root@semester -]# oc get scc
NAME            PRIV      CAPS      SELinux                  RUNASUSER   FSGROUP   SUPGROUP   PRIORITY   READONLYROOTFS   VOLUMES
anyuid          false      []        MustRunAs                RunAsAny    RunAsAny  RunAsAny   10         false           [configMap downwardAPI emptyDir persistentVolumeClaim secret]
hostaccess      false      []        MustRunAs                MustRunAsRange RunAsAny  RunAsAny   <none>     false           [configMap downwardAPI emptyDir hostPath persistentVolumeClaim]
hostnetwork     false      []        MustRunAs                MustRunAsRange MustRunAs  MustRunAs   <none>     false           [configMap downwardAPI emptyDir hostPath nfs persistentVolumeClaim]
nonroot         false      []        MustRunAs                MustRunAsNonRoot RunAsAny  RunAsAny   <none>     false           [configMap downwardAPI emptyDir persistentVolumeClaim]
privileged      true       []        RunAsAny                 RunAsAny    RunAsAny  RunAsAny   <none>     false           [configMap downwardAPI emptyDir persistentVolumeClaim]
restricted      false      []        MustRunAs                MustRunAsRange MustRunAs  RunAsAny    <none>     false           [configMap downwardAPI emptyDir persistentVolumeClaim]
```

```bash
[root@semester -]# oc describe scc restricted
Name:          restricted
Priority:      <none>
Access:        Users:  <none>
                      Groups: system:authenticated
Settings:
  Allow Privileged: false
  Default Add Capabilities: <none>
  Required Drop Capabilities: KILL, MKNOD, SYS CHROOT, SETUID, SETGID
  Allowed Capabilities:      <none>
  Allowed Volume Types:      configMap, downwardAPI, emptyDir, persistentVolumeClaim, secret
  Allowed Host Network:      false
  Allowed Host Ports:        false
  Allowed Host PID:          false
  Allowed Host IPC:          false
  Read Only Root Filesystem: false
  Run As User Strategy:      MustRunAsRange
    UID:                      <none>
    UID Range Min:           <none>
    UID Range Max:           <none>
  SELinux Context Strategy: MustRunAs
    User:                    <none>
    Role:                    <none>
    Type:                    <none>
    Level:                   <none>
  FSGroup Strategy:         MustRunAs
    Ranges:                  <none>
  Supplemental Groups Strategy: RunAsAny
    Ranges:                  <none>
```
Secured Communications between Hosts

Secures cluster communications with IPsec

- Encryption between all Master and Node hosts (L3)
- Uses OpenShift CA and existing certificates
- Simple setup via policy defn
  - Groups (e.g. subnets)
  - Individual hosts
Microsegmentation with Network Policy Objects

- Lock everything by default
- Add Network Policies to allow specific ingress traffic
Securing Storage attached to Containers

Secure storage by using

- SELinux access controls
- Secure mounts
- Supplemental group IDs for shared storage
Isolate Workloads by labeling Nodes

```bash
$ oadm new-project myproject \
   --node-selector='type=user-node,region=east'
```

Diagram:
- Master / Scheduler
- Node 1
  - east
  - pod
- Node 2
  - east
  - pod
- Node 1
  - west
- Node 2
  - west
Securing Applications (running as containers)
API Management

Container platform & application APIs

- Authentication and authorization
- LDAP integration
- End-point access controls
- Rate limiting
SSL at Ingress

Edge termination

Router

https://myapp.mydomain.com

Passthrough termination

Router

https://myapp.mydomain.com

Reencrypt

Router

https://myapp.mydomain.com
Secrets

Sensitive Info: Passwords, Client Config files, dockercfg etc used by application containers provided as secrets

Never come to rest on Nodes

Stored in etcd and encrypted

```
kubernetesMasterConfig:
apiServerArguments:
experimental-encryption-provider-config:
- /path/to/encryption.config
```
Calling External Services using Egress Router

The OpenShift egress router runs a service that redirects egress pod traffic to one or more specified remote servers, using a pre-defined source IP address that can be whitelisted on the remote server.

```
... - name: EGRESS_DESTINATION
  value: |
    80 tcp 1.2.3.4
    8080 tcp 5.6.7.8 80
    8443 tcp 9.10.11.12 443
    13.14.15.16
...
```
Securing Microservices on Service Mesh
Istio Concepts - Sidecar Proxy

**SideCar Proxy**
- Intercepts all network communication between microservices
- Encapsulates Service Infrastructure code
- Application code (business logic) unaware of Sidecar proxy
- Examples - Linkerd, Envoy
Service Mesh is a dedicated infrastructure layer to handle service-service communications. Typically implemented as an array of lightweight network proxies deployed alongside application code. Interconnected Proxies form a mesh network.
Istio Service Mesh on OpenShift

Connect, Manage, and Secure Microservices, transparently

- Intelligent Routing
- Load Balancing
- Service Resilience
- Telemetry and Reporting
- Policy Enforcement
- Content based Filtering (Layer 7)
- mTLS between services
- East-West traffic control
Application Traffic Encryption with Istio Auth

Uses Service Account as Identity. SPIFFE Id format

spiffe://<domain>/<ns>/<namespace>/sa/<serviceaccount>

Mutual TLS between sidecars

Istio CA

- Generate cert pair and SPIFFE key for each SA
- Distribute key and cert pairs
- Rotate keys and certs periodically
- Revoke key and cert when need
Questions
Thank you!!
<table>
<thead>
<tr>
<th>PROCESS TYPES</th>
<th>CAT</th>
<th>DOG</th>
</tr>
</thead>
</table>

TYPE ENFORCEMENT
TYPE ENFORCEMENT

ALLOW + CAT + CAT_CHOW:FOOD + EAT
ALLOW + DOG + DOG_CHOW:FOOD + EAT
TYPE ENFORCEMENT

DELICIOUS INTERACTION!

I ♥ INTERACTING WITH FOOD!
TYPE ENFORCEMENT

CAT_CHOW, PLEASE?

SURE!
TYPE ENFORCEMENT

DOG

KERNEL

CAT CHOW
SELINUX - MAC - MCS - Process

system_u:system_r:container_runtime_t:s0

[root@osemaster ~]# ps -efZ | grep docker-containerd-shim-current
system_u:system_r:container_runtime_t:s0 root 3035 1479 0 Feb15 ? 00:00:01
/usr/bin/docker-containerd-shim-current
4d254785cbc6ee7aae8facc48555251e2385f65d89553b319b6324b1501e4b16
/var/run/docker/libcontainerd/4d254785cbc6ee7aae8facc48555251e2385f65d89553b319b6324b1501e4b16
/usr/libexec/docker/docker-runc-current

The OOTB SElinux policy container.te defines what you can execute and access with the label container_runtime_t

SElinux Policy module for the container
SELINUX - MAC - MCS - Files

container_var_lib_t / svirt_sandbox_file_t

root@osemaster ~# ls -lZ
/var/lib/docker/containers/97de4217a04b6532e312cfe3e4638529aeb7dfa281a2cc067e092fcee82e6737 /
-rw-r----- root root system_u:object_r:container_var_lib_t:s0
  97de4217a04b6532e312cfe3e4638529aeb7dfa281a2cc067e092fcee82e6737-json.log
-rw-rw-rw-. root root system_u:object_r:container_var_lib_t:s0 config.v2.json
-rw-rw-rw-. root root system_u:object_r:container_var_lib_t:s0 hostconfig.json
-rw-r--- root root system_u:object_r:svirt_sandbox_file_t:s0 hostname
-rw-r--- root root system_u:object_r:svirt_sandbox_file_t:s0:c0,c1 hosts
-rw-r--- root root system_u:object_r:svirt_sandbox_file_t:s0 resolv.conf
-rw-r--- root root system_u:object_r:container_var_lib_t:s0 resolv.conf.hash
drwxr-xr-x root root system_u:object_r:svirt_sandbox_file_t:s0:c0,c1 secrets
drax------ root root system_u:object_r:container_var_lib_t:s0 shm