DevOps and Container Security

Mike Bartock  
IT Specialist  
NIST

Paul Cichonski  
Cloud Architect  
Lancop

John Morello  
Chief Technology Officer  
Twistlock

Raghu Yeluri  
Principal Engineer  
Intel
Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.
Agenda

- Introductions
- Purpose of the Panel
- Panelist Container Work
  - Intel
  - Twistlock
  - Lancope
- Discussion on container security and applications
- Questions from the Audience
Purpose

• Introduction to containers and their uses
• Different methods of security for containers
• Discussion of what industry is doing
Trusted Containers

Raghu Yeluri
Principal Engineer, Lead Cloud Security Architect
Datacenter Group, Intel Corporation
Legal Information

Intel technologies, features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel, the Intel logo, Intel vPro, Look Inside., the Look Inside. logo, Intel Xeon Phi, and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.
Containers

Lightweight, fast, disposable virtual environments

- App portability, maintenance and deployment.

Technically:

- Set of processes running atop shared kernel
- Isolated from rest of the system (limitations)

From a distance… looks like a VM (SSH, root access, eth0, mount file systems)

Have been around for 10+ years (Solaris* containers, Linux* Containers..)

Efficient way to build, ship, run, deliver apps
What is Docker*?

Lightweight, open source engine for creating, deploying containers
- Provides work flow for running, building and containerizing apps.
- Separates apps from where they run; enables Micro-services; scale by composition
- Underlying building blocks: Linux* kernel's namespaces (isolation) + cgroups (resource control) + ..

Components of Docker*
- Docker Engine: Runtime for running, building Docker containers
- Docker Repositories(Hub): SaaS for sharing/managing images
- Docker Images (layers)
  - Images hold Apps. Shareable snapshot of software. Container is a running instance of image.

Orchestration: OpenStack*, Docker Swarm, Kubernetes*, Mesos*, CoreOs Tectonic, Fleet
Container Security – Key Customer Asks

1. Docker® Host Integrity
   - Do you trust the Docker daemon?
   - Do you trust the Docker host has booted with Integrity?

2. Docker Container Integrity verification
   - Who wrote the container image? Do you trust the image? Did the right Image get launched?

3. Runtime Protection of containers & Enhanced Isolation
   - How can Intel help with runtime Integrity, Isolation?

4. Intelligent orchestration – OpenStack as singular control plane for Trusted VMs and Containers

Intel’s Focus: Hardware-based Integrity Assurance for Containers – Trusted Docker Containers
Trusted Docker* Containers – 3 Focus Areas

Launch Integrity of Docker* Host & Docker Engine

Integrity of Docker Images & Containers

Looking ahead…Runtime Integrity of Docker Host, H/w-based enhanced Isolation
Trusted VMs - Summary

Launch VMs on Servers with demonstrated Boot Integrity – **Trusted Boot**

Chain of Trust to VMs – **Trusted VMs**

Control where Trusted VMs are launching and migrating: **Boundary Control of VMs**

Measurements done at the time of boot (Server boot and VM Launch)

- Host OS/Hypervisor
- Kernel, Initrd++
- HW w/ Intel TXT/TPM

Measurements match!

**Trusted Platform Module (TPM)**
**Intel® Trusted Execution Technology (Intel® TXT)**

Enable same model and use-cases for Trusted Containers
Trusted Docker* Containers - 1

Ensure Docker* Containers are launched on **Trusted Docker Hosts**

Boot-time integrity of the Docker Host
- **Measured** Launch of Boot Process and components with Intel® Trusted Execution Technology (Intel® TXT)

Docker daemon and associated components added to TCB and Measured

Chain of Trust: **H/W → FW → BIOS → OS → Docker Engine**

Remote attestation using Intel Cloud Integrity Technology (Attestation Authority)

Assure and attest the Integrity of Docker host/platform
Trusted Docker* Containers - 2

Ensure that Docker* Images not tampered prior to Launch

Two Models:

1. Measure and verify Docker images, Chain of Trust: H/W → FW → BIOS → OS → Docker Engine → Docker image layers

2. Sign images in Docker Hub. Verify images signature prior to launch with root Cert signature that is ‘Sealed’ to Intel® Trusted Execution Technology (Intel® TXT) measurements in the Trusted Platform Module (TPM). – Can work with Notary* - Docker Content Trust Model.

Boundary Control/Geo-Tagging applies equally to Docker Containers as well for compliance needs

- Orchestrator determines location/boundary at launch time

Assure and attest the Integrity of Docker images/containers
How about Docker* Containers in VMs?

Leverage Trusted VMs for asserting trust of the host, and the VMs.

Include Docker* Daemon as part of VMs TCB – measure and verify Docker Daemon as part of VM launch attestation.

Boot-time integrity of Host + VMM

Integrity assurance of VM and Docker Daemon

Chain of Trust: H/W → FW → BIOS-OS/VMM-VM → Docker Engine

Assure and attest the Integrity of Host and the VM w/ Docker Engine
What is Measured for Trusted Containers

- **Bootloader**, **Tboot and OS Kernel**
- **Initrd++ (includes tboot-xm)**
- **Docker® Daemon**
  - Container management engine (e.g., Docker engine)
  - Measurement Agents
- **Intel® Trusted Execution Technology (Intel® TXT)**
- **ACM signed by manufacturer**
- **Intel TXT + TPM**

**Chain of Trust extended to application launch**

**Containerized application layers (e.g., Docker image layers)**

**Load-time creation of a component’s Identity (i.e., Hash of component)**

**Apache® Patch v2**
- Apache Patch v1
- Apache
- Ubuntu
- Ubuntu 14.04

**Intel® Trusted Execution Technology (Intel® TXT) chain of trust extended up the stack**
Looking Ahead: Hardware-based Runtime Integrity

Intel® Kernel Guard Technology (Intel® KGT)
- Policy specification and enforcement framework
- Ensuring runtime integrity of kernel and platform assets

Extends launch-time integrity to run-time integrity

Based on a thin Intel VT-x (VMX-root) layer software component called xmon
- De-privileges OS
- Monitors/controls access to critical assets (CRs, MSRs, Memory Pages..)

Allows specification of policy from user-mode via configfs
- Policy describes assets to be monitored and actions to be taken when monitoring events occur

Policy can be locked down until next reboot

Intel KGT: Flexible, low overhead integrity framework; open source
“We’re going to build a software layer to make the internet programmable”

- Docker, DockerCon 2015
Docker is the best known example…

… but these trends are being driven by the growth of software across all industries and the need to rapidly build, iterate, and improve it

… all major IT players are investing
What are the challenges?

Containers don’t keep themselves up to date

Many more containers but fewer tools for protecting them

Many more, and more diverse, places where your containers run

‘All or nothing’ administrative model
Why not existing solutions?

Containers are portable and minimal

Deployment is frictionless

Cramming containers full of agents and tools is antithetical to the model
What is Twistlock?

The first security solution built for containerized computing

… that secures the entire lifecycle of containerized apps…

… across all the environments they run in

A company that contributes back to the open source community
Defend your containers

Vulnerability management, with an intelligence stream of the latest CVEs and proactive defense

Advanced authorization capabilities, including Kerberos support and role based access control

Runtime defense, monitoring container memory space, storage, and networking to detect and block anomalous behaviors

© 2015 Twistlock
Purpose built

Agentless

Runs anywhere your containers run

API driven for continuous integration
Container Security Console

Configure

Monitor

Visualize risk

© 2015 Twistlock
Vulnerability management “demo”

- Block deployment of vulnerable images
- Tag resources and apply granular policies
Security hardening “demo”

Ensure regulatory compliance

Prevent configuration drift
RBAC “demo”
Evolution of roles with containers

‘Traditional’

With Twistlock

© 2015 Twistlock
The containers are coming

… if they’re not already on your network

Balance security and capability with tools purpose built for the new model

Twistlock
Container Security

Paul Cichonski
Cloud Architect

@paulcichonski
Why Containers? Two Areas of Focus

1. Software Delivery:
   - Build pipelines now produce consistent, immutable artifacts
   - Immutable artifacts offer many benefits for security

2. Software Deployment:
   - Software deployment mechanism is common across all technologies (e.g., python, JVM, c, perl)
   - If it can go into a container, you can deploy it
   - Incredible for devs, but creates many challenges for security
# Evolution of Software Delivery

<table>
<thead>
<tr>
<th>Era*</th>
<th>Characteristics</th>
</tr>
</thead>
</table>
| Custom Bash Scripts (1990s – late 2000s) | • Mutable infrastructure (e.g., send EAR to server)  
  • Servers are pets, we even gave them names  
  • Many differences between environments  
  • Deployed a few times per month (if lucky) |
| Configuration Management (Late 2000s – current) | • Immutable infrastructure  
  • Servers start becoming cattle  
  • Still many differences between environments  
  • Deployments happen more, but still slow |
| Containers (now) | • Immutable infrastructure  
  • Servers become more like cattle, OS provides bare minimum to run container  
  • Software systems now fully reproducible in all environments  
  • Build/Deployment pipeline is the center of the universe  
  • Deploy as frequently as your build pipeline can produce new image |

*Time periods are rough estimates, they change depending on who you ask.*
Deployment Pipelines with Containers

Steps taken inside CI (fail fast between steps):

1. Standup dependent services (using containers) for testing
2. Run unit and integration tests on code
3. Create final Docker Image of tested code
4. Start container using newly created image
5. Run black box functional tests on container
6. **Run security scans on container**
   1. Examples: SCAP scan, GAUNTLET tests, CIS Docker Benchmark
7. Push validated image to registry IFF all previous steps pass
Why this is good for security?

• A successful run of deployment pipeline gives us an immutable image for deploying to production
  – Never run anything not validated by pipeline
• Before ever getting to prod, we have already instantiated the container and run:
  – Blackbox functional tests
  – Full security scans (both blackbox and whitebox)

This means we can catch security issues before ever releasing software into the wild

Side bonus: devs can run all this from their laptop
Manifest encodes:
- Docker image to launch on cluster
- Number of instances to deploy (e.g., run 3 instances of nginx container)
- Resource requirements (e.g., each container needs 2 cores and 8gb memory)
- Custom rules (e.g., don’t run container X and container Y on same host)
Deploying Containers (high level)

- Each server (or resource) is only there to run containers
- Stripped down kernel (lower attack surface)
- Orchestration tooling required to help schedule containers across a cluster
Benefits for Dev/Ops

Containers provide a common point of abstraction for deploying any arbitrary software stack

Great for microservices and polyglot infrastructures

We can start thinking about creating infrastructure-level patterns and sharing them via GitHub (think: package manager for your datacenter)
What about security? Here be dragons

Orchestration layer adds new set of distributed communication protocols that must be secured

Host-level isolation for different workloads still required until container isolation is on par with OS isolation

Storing secrets becomes more complex in a dynamic world

Image validation tooling required to forbid untrusted images (it is not enabled by default)

Burden of patching software shifts to devs
Questions?