Why Security needs to be at the DevOps Table

Chris Perkins, Sr. Principal Security Technologist. Medtronic, PLC
Evolution of Compute and Development Methodologies

- Both processes evolved over time, with a similar pathway
- Technology advances helped drive speed in both arenas
- Pivotal concepts also helped “break norms”
The Evolution of Compute

- Physical
- Virtualization
- Cloud Compute
- Containers
- Serverless
Software Development Evolution

Waterfall

Agile

DevOps
Each application was “racked and stacked”

Each box housed a dedicated database, dedicated service
  - Web Services
  - Email
  - Authentication
Waterfall
• Introduced the Theory of Constraints (TOC) concept
• Process of Ongoing Improvement
• Critical Chain Project Management (CCPM)
• First published in 1984
Then came Virtualization

- The process of creating logical computing resources from available physical resources
- Layer of abstraction between workloads and the underlying physical hardware via Hypervisor
- Allowed for many to one, physical server usage optimized
Agile Development
Containers continued the concepts
Containers vs. VMs

Containers are isolated, but share OS and, where appropriate, bins/libraries

http://www.theplatform.net/2015/03/23/hpc-schedulers-snap-to-docker/
WORKED FINE IN DEV

OPS PROBLEM NOW
The Phoenix Project
3 Ways of DevOps
Strategies for Improving Operations
Container Threat Modeling

S.T.R.I.D.E.

- **Spoofing**: Attacker can prove that they are an authorized system of the user
- **Tampering**: Attacker can successfully add, modify and delete data
- **Repudiation**: Attacker can deny or make it impossible to prove their delinquency
- **Information Disclosure**: Attacker can gain access to Privileged information
- **Denial of Service**: Attacker can make the system unresponsive for legitimate use
- **Elevation of Privilege**: Attacker can elevate their privileges
Spoofing

- Authentication Attacks against Host
- Docker Daemon Direct Access
- Trojanized Docker Images
- Exposure of Private Docker Registry
- ARP Spoofing
- Docker Registry Certificate Spoofing
- Insecure Docker API configuration
Tampering

- Trojanized Docker Image
- Docker Daemon Direct Access
- Docker Daemon Configuration Attacks
- Docker Registry Certificate Spoofing
- Content Trust
- Host File System Integrity Breaches
- Docker Daemon Tampering Host Network Configurations
Repudiation

- No Audit/Delete Audit Docker Images
- Docker Daemon API Logs - Compromise
- Host File System Integrity Breaches
Information Disclosure

- Secrets being disclosed to outside entities
- Exposed Ports and Services
- Network Traffic Compromise
Denial of Service

- CPU/Memory Exhaust
- Network Exhaust
- HDD Exhaust
Elevation of Privileges

- Container Breakout
- Container Privileges
- Container Services - Compromise
DevOps
Containers managed individually..
Led to Orchestration
What is Container Orchestration?

- Deploy and Configure
- Fault Isolation & Healing
- Secure
- Upgrades
- Scaling Up and Down

Stateless Applications
- Nothing to disk
- Web front-end
- Can stop and start as many containers as you like
- Container is ephemeral
- No container instance-specific configuration

Stateful Applications
- Container-specific: Host names, IP addresses
- Big Data service configuration information
- Security secrets: passwords, KDC keys
The Kubernetes Threat Model

K8s Threat Model

User compromises the cluster
- Users can access Cluster/Controller without authentication
- Users can access the Cluster with stolen secrets/tokens to perform sensitive operations on Cluster
- User can tamper with user cert settings and gain access to Cluster as a genuine user
- User has unrestricted access across the Cluster
- User has highly privileged access across the Cluster

Malicious App (Container) Compromises the Cluster
- Attacker is able to RCE into a container and subsequently gain access to other services, pods etc on the Cluster to steal sensitive information
- Org runs backdoored/Compromised container in the Cluster, that is able to access other resources on/ across the cluster and steal sensitive information
- Attacker is able to perform CPU/mem intensive Ops across the cluster and bring it down
- Attacker gets a trojanized image to run on the cluster and compromise sensitive information
- Backdoored/compromised container accesses shared resources and runs ransomware/affects availability to shared resources

Malicious node compromises the cluster
- Malicious node registers itself as a genuine node to the cluster and compromises the node therefrom
- Exploit against Node escalates privileges to Kube deployment and compromises cluster

Compromise secrets
- Gain access to DB/FS/Sensitive Information
- Gain access to other namespaces for Cross-Cluster compromise
- Access Kube API and Controller to access K8s Management Sensitive Info

Compromise secrets
- Gain access to DB/FS/Sensitive Information
- Gain access to other namespaces for Cross-Cluster compromise
- Access Kube API and Controller to access K8s Management Sensitive Info
And Finally, Serverless!
# Advantages of Serverless Computing

<table>
<thead>
<tr>
<th></th>
<th>Bare Metal</th>
<th>VM</th>
<th>Container</th>
<th>Serverless</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Boot Time</strong></td>
<td>~20 mins</td>
<td>~2 mins</td>
<td>2 secs</td>
<td>~0.0003 secs</td>
</tr>
<tr>
<td><strong>App deployment lifecycle</strong></td>
<td>Deploy in Weeks, Live for years</td>
<td>Deploy in minutes, Live for weeks</td>
<td>Deploy in Seconds, Live for minutes/hours</td>
<td>Deploy in milliseconds, Live for seconds</td>
</tr>
<tr>
<td><strong>Investment</strong></td>
<td>Buy/rent dedicated server</td>
<td>Rent a dedicated VM, on a shared server</td>
<td>Rent Containers, pay for the actual runtime</td>
<td>Pay for compute resources used during runtime</td>
</tr>
<tr>
<td><strong>Scaling</strong></td>
<td>Takes months, Should be approved by a panel of experts</td>
<td>Takes hours, Should be approved by administrators</td>
<td>Takes seconds, Policy driven scaling</td>
<td>Takes milliseconds, Scaling is event driven</td>
</tr>
</tbody>
</table>
Serverless - Security Responsibility Model

- Client side
- Data in cloud
- Data in transit
- Applications (functions)
- Identity and access management
- Cloud services configuration

Operating systems + Virtual machines + computers
- Compute
- Storage
- Database
- Networking
- Regions
- Availability zones
- Edge locations
Serverless Architectures Security - Top 10

- SAS 1: Function Event Data Injection
- SAS 2: Broken Authentication
- SAS 3: Insecure Serverless Deployment Configuration
- SAS 4: Over-Privileged Function Permissions and Roles
- SAS 5: Inadequate Function Monitoring and Logging
- SAS 6: Insecure 3rd Party Dependencies
- SAS 7: Insecure Application Secrets Storage
- SAS 8: Denial of Service and Financial Resource Exhaustion
- SAS 9: Serverless Function Flow Manipulation
- SAS 10: Improper Exception Handling and Verbose Errors
DevSecOps
Continuous Build, Integration and Delivery—Foundational and Automated (CI/CD)
Example CI/CD pipeline with AppSec addition

DevOps Pipeline

AppSec Pipeline
Great place to start!

DevSecOps Studio is one of its kind, self contained DevSecOps environment/distribution to help individuals in learning DevSecOps concepts. It takes lots of efforts to setup the environment for training/demos and more often, its error prone when done manually. DevSecOps Studio is easy to get started and is mostly automatic.

DevSecOps Studio project aims to reduce the time to bootstrap the environment and help you in concentrating on learning/teaching DevSecOps practices.

Features:
- Easy to setup environment with just one command “vagrant up”
- Teaches Security as Code, Compliance as Code, Infrastructure as Code
- With built-in support for CI/CD pipeline
- OS hardening using ansible
- Compliance as code using Inspec
- QA security using ZAP, BDD-Security and Gauntlt
- Static tools like bandit, brakeman, windbags, gitrob, gitsecrets
- Security Monitoring using ELK stack.

https://www.owasp.org/index.php/OWASP_DevSecOps_Studio_Project