SESSION ID: DEV-F03

DEVOPS AND THE FUTURE OF ENTERPRISE SECURITY

Frank Kim
Founder
ThinkSec
@fykim
www.frankkim.net
“DevOps is just another excuse for developers to have root access in production.”
Walls of Confusion

I want change!

I want stability!

Development

Operations

Image concept: http://dev2ops.org/2010/02/what-is-devops
#1 Understand DevOps
“It’s not the strongest that survive or the most intelligent that survive. It’s the ones that are most adaptable to change.”

- Charles Darwin
Monolith Architecture Security Controls

- Common security controls are applied to each trust boundary in the monolith architecture:

1. Security Controls
   - Web Application Firewall
   - HTTPS, Rate Limiting

2. Security Controls
   - Authentication, Authorization
   - Access control, Validation

3. Security Controls
   - System Authentication, TLS
   - Encryption at rest
How does this change in a microservice architecture?
Consider the attack surface in a modern microservice architecture:
Microservice API Gateway Architecture

- Adding an API Gateway to provide perimeter security controls:

  - Single Page App
  - Mobile App
  - IoT Factory Device
  - API Gateway
  - Authorization Access Control
  - Public Subnet
    - Human Resources
    - Account Management
    - Employee
    - Customer Service
    - Discount Coupons
  - Private Subnet
    - MySQL Database Server
    - EBS Volume
    - Coupon Bucket

VPC
Serverless Computing

- Serverless refers to new, non-traditional architecture
  - Does not use dedicated containers
  - Event-triggered computing requests
  - Ephemeral environment
  - Servers fully managed by 3rd party (e.g. AWS)
  - Referred to as Functions as a service (FaaS)

- Example Technologies
  - AWS Lambda, MS Azure Functions, Google Cloud Functions
  - Amazon API Gateway
Serverless Security Benefits

How does serverless improve security?

• Attack surface is smaller
• No servers to patch
• No long running servers
  • That can be scanned or attacked
  • That can have malware installed on them
• Fewer compromised servers
  • If malware is installed the next request brings up new, clean “server”
Serverless Security Concerns

- How does serverless make security harder?
  - Attack surface is bigger (but different)
  - Authentication and access control
  - Compliance
Application security is even more important with serverless
- If attackers have less infrastructure to attack
- The focus naturally shifts to the application

Every function crosses a trust boundary
- Functions are designed to independent
- Therefore each function must be secured independently

Apply application security best practices
- Input validation / sanitization must be performed in each function
- Perform code review and automated scans
- Review dependencies and libraries used by functions
AWS WAF Security Automations

#2 Support DevOps
Can you identify the bug in this code snippet?

```java
<%  
String theme = request.getParameter("look");  
if (theme == null && session != null) {  
    theme = (String)session.getAttribute("look");  
}  
%

if (session !=null) session.setAttribute("look", theme);

<link rel="stylesheet" type="text/css" media="all" 
href="<%= request.getContextPath() %>/ui/theme/<%= theme %>/colors.css" />
```
Can you identify the bug in this code snippet?

```jsp
<% String theme = request.getParameter("look");
if (theme == null && session != null) {
    theme = (String)session.getAttribute("look");
}
%

<% if (session !=null) session.setAttribute("look", theme); %>

<link rel="stylesheet" type="text/css" media="all" href="<%= request.getContextPath() %>/ui/theme/<%= theme %>/colors.css" />
```
AngularJS Output Encoding

- **ngBind for HTML tags**

```html
<div ng-controller="ExampleController">
  <label>Enter name: <input type="text" ng-model="name"></label><br>
  Hello <span ng-bind="name"></span>!
</div>
```

- **Output from AngularJS expressions**

```html
<div ng-controller="ExampleController" class="expressions">
  Expression:<input type='text' ng-model="expr" size="80"/>
  <button ng-click="addExp(expr)">Evaluate</button>
  <ul>
    <li ng-repeat="expr in exprs track by $index">
      [ <a href="" ng-click="removeExp($index)">X</a> ]
      <code>{{expr}}</code> => <span ng-bind="$parent.$eval(expr)"></span>
    </li>
  </ul>
</div>
```
Static Analysis Tools

- **Free / open source:**
  - Find Security Bugs, Phan, Puma Scan, Brakeman, Bandit, Flawfinder, QARK

- **Commercial:**
  - HP Fortify, Checkmarx, Coverity, IBM AppScan Source, Klocwork, Veracode, Brakeman Pro
Secure Code Spell Checker

```csharp
31 }
32 }
33 
34 protected void lbEnter_Click(object sender, EventArgs e)
35 {
36     string name = Request["name"];
37     lblMessage.Text = "Thanks " + name + "! Thank you for your interest."
38 }
39 
40 protected void cboxAgree_CheckedChanged(object sender, EventArgs e)
41 {
42     SetupPage(cboxAgree.Checked);
43 }
```
Secure DevOps Toolchain

Building a DevSecOps Program (CALMS)

Culture
Break down barriers between Development, Security, and Operations through education and outreach

Automation
Embed self-service automated security scanning and testing in continuous delivery

Leverage value stream analysis on security and compliance processes to optimize flow

Measurement
Metrics to drive design and drive decisions

Sharing
Share threats, risk, and vulnerabilities by adding them to engineering tooling

First Steps in Automation
- Start Your DevOps Metrics Program
  - Build a security-aware test lab (e.g., JEP Integration Test)
  - Build and deployment pipeline
  - Scanning and testing in a GitOps workflow
  - Attaching your system to a test lab (e.g., Jira)
  - Add hardened steps into configuration pipelines (e.g., devsecops)
  - Number of attacks (true and false) impacting your application

Start a DevOps Maturity Model

Learn to build, deliver, and deploy modern applications using secure DevOps and cloud principles, practices, and tools.

bit.ly/secdevops-toolchain
#3: Adopt DevOps
Critical Security Controls (CSC)

CIS Controls

First 5 CIS Controls
Eliminate the vast majority of your organization’s vulnerabilities

1: Inventory of Authorized and Unauthorized Devices
2: Inventory of Authorized and Unauthorized Software
3: Secure Configurations for Hardware and Software
4: Continuous Vulnerability Assessment and Remediation
5: Controlled Use of Administrative Privileges

THINKSEC
Different approaches to set up and manage systems

- Traditional: manual checklists and scripts, ad hoc changes/fixes made by system administrators at runtime
- Modern: treating Infrastructure as Code and configuration management as system engineering

Configuration management with scripts is not scalable

- Error prone and leads to configuration drift over time

Configuration management tools

- Chef, Puppet, Ansible, Salt/Saltstack, CFEngine
**AWS CloudFormation to create EC2 instance**

```yaml
InstancePublic:
  Type: AWS::EC2::Instance
  Properties:
    IamInstanceProfile: !Ref InstanceProfilePhotoReadOnly
    ImageId: !FindInMap [Images, !Ref "AWS::Region", ecs]
    InstanceType: "t2.micro"
    KeyName: "secretKey"
    SecurityGroupIds:
      - !Ref SecurityGroupPublic
    SubnetId: !Ref SubnetPublic
  UserData:
    Fn::Base64:
      !Sub |
      #!/bin/bash -xe
      yum update -y
```
Conduct Asset Inventory

- **Command line call to retrieve all EC2 instances**

```
aws ec2 describe-instances --output json | jq '.Reservations[].Instances[] | [.LaunchTime, .InstanceType, .InstanceId, .SecurityGroups[].GroupId, .Tags[].Value]'
```

- **Output**

```
[ "2017-01-08T18:51:46.000Z", "t2.micro", "i-0500510e3f808d2ee", "sg-7caf4600", "prod-springline-aws-web", "Springboot MVC target application", "SANS\app.user"
]
[
 "2017-01-08T18:55:02.000Z", "t2.micro", "i-0e74e490c2ebc5d37", "sg-79af4605", "qa-springline-aws-web", "QA Springboot MVC target application", "SANS\app.user"
]
```
Continuous Vulnerability Remediation

- **Blue/Green Deployment**
  - Divert traffic from one environment to another
  - Each running a different version of the application

- **Benefits of blue/green deployments**
  - Reduced downtime
  - Improved ability to rollback
  - Faster deployment of features and bug fixes

- **Use blue/green deploys when you have:**
  - Immutable infrastructure
  - Well defined environment boundary
  - Ability to automate changes
- **Deployment process**
  - Use original blue service and task def
  - Create new green service and task def
  - Map new green service to the Application Load Balancer (ALB)
  - Scale up green service by increasing number of tasks
  - Remove blue service, setting tasks to 0
Deploying Application Updates

- Create a new “green” ECS Service
  
  ```bash
  aws cloudformation deploy --template-file green-web-ecs-service.yaml --stack-name green-web-ecs-service
  ```

- Increase the desired count for the “green” service
  
  ```bash
  aws ecs update-service --cluster DM-ecs --service $GreenService --desired-count 1
  ```

- Turn off the “blue” service when ready
  
  ```bash
  aws ecs update-service --cluster DM-ecs --service $BlueService --desired-count 0
  ```
Key Takeaways

- **Understand DevOps**
  - Next week: Begin to understand the DevOps CI/CD pipeline and modern architectures used in your organization

- **Support DevOps**
  - In three months: Inject security into the CI/CD pipeline in an easy to use way

- **Adopt DevOps**
  - In six months: Leverage DevOps principles and practices to improve your security program
Frank Kim
@fykim
www.frankkim.net

Material based on SANS DEV540
Secure DevOps and Cloud Application Security