SANS

Continuous Security and DevOps:
Three Keys for Modern Security Success

Introduction

Frank Kim

- ThinkSec
  - Founder
- SANS Institute
  - Former CISO
  - Curriculum Lead
    - Management and Application Security
  - Author & Instructor
    - MGT514, DEV540, DEV541
- Shout out to Eric Johnson
  - Summit Co-Chair
  - Author of many of these slides

Security Perceptions

"DevOps is just another excuse
for developers to have
root access in production."

- Traditional security expert

Walls of Confusion

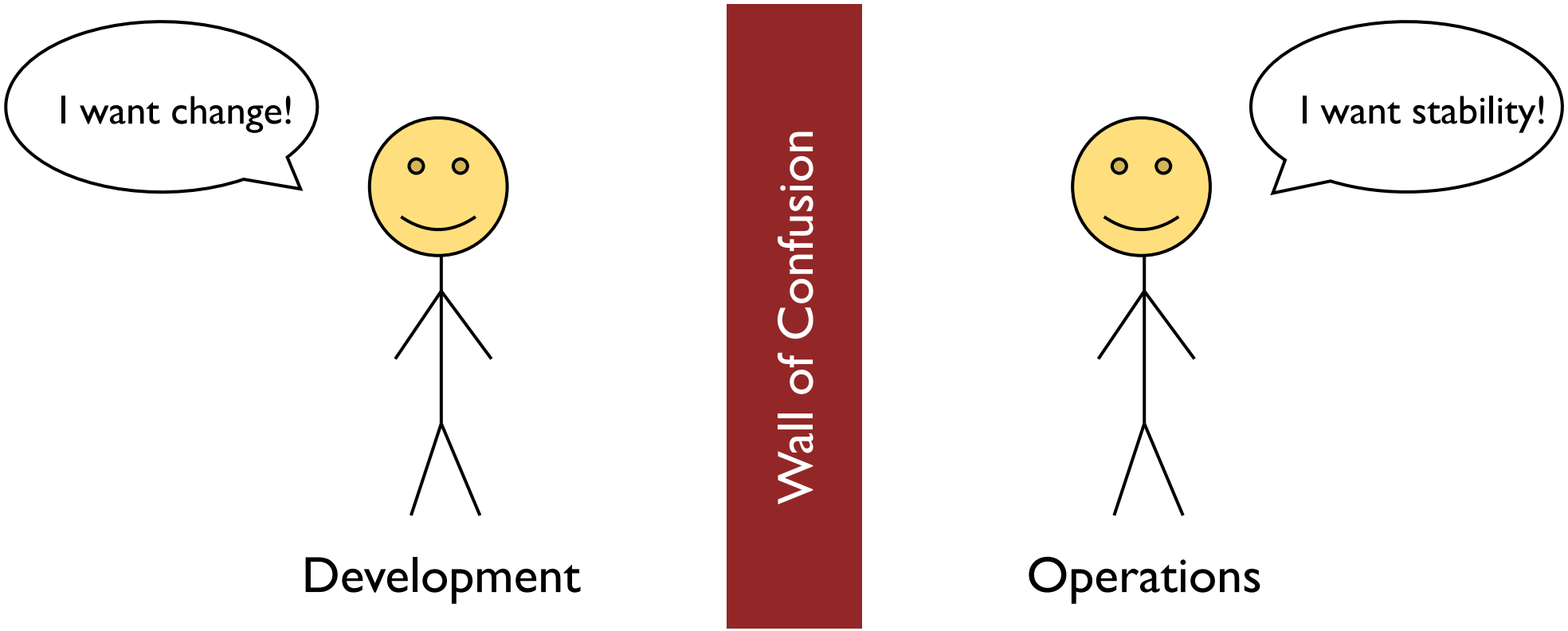


Image concept: http://dev2ops.org/2010/02/what-is-devops

#1 | Champion Change

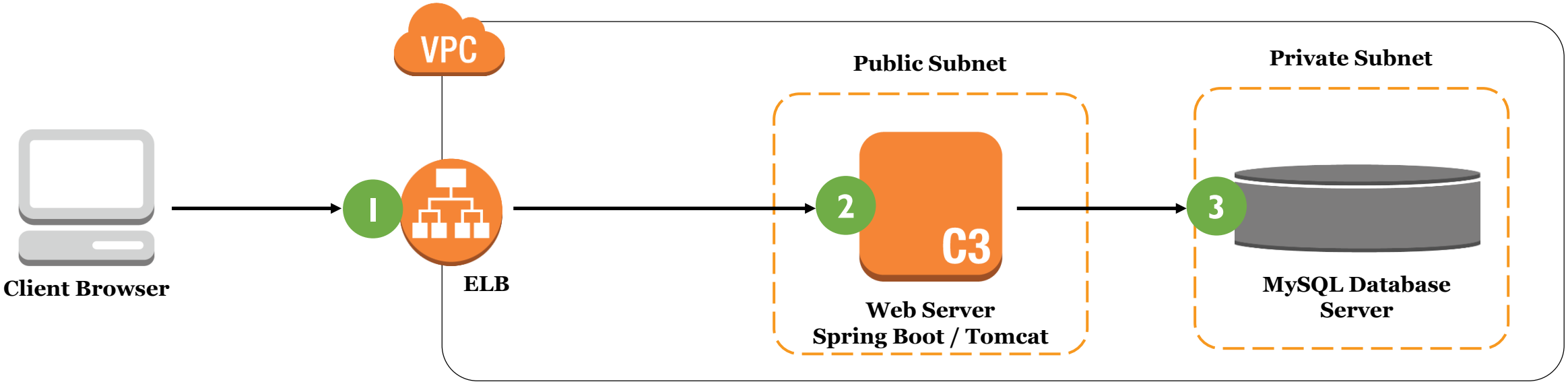
Embracing Change

"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:



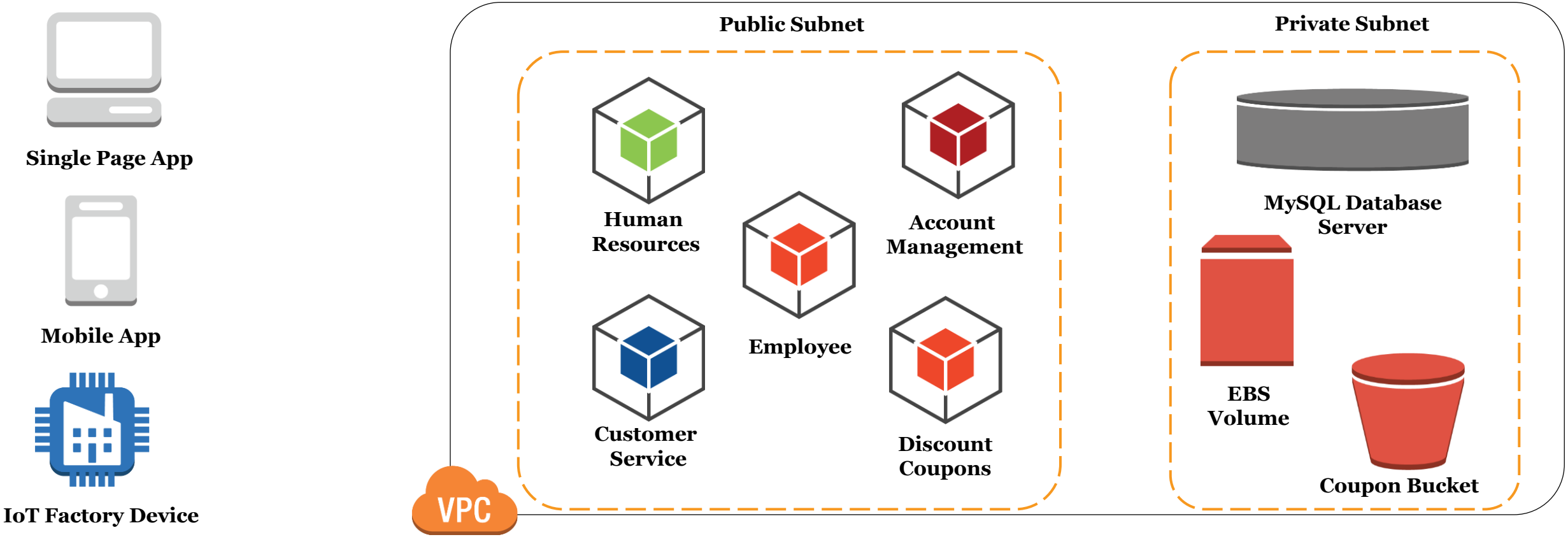
1. Security Controls
Web Application Firewall
HTTPS, Rate Limiting

2. Security Controls
Authentication, Authorization
Access control, Validation

3. Security Controls
System Authentication, TLS
Encryption at rest

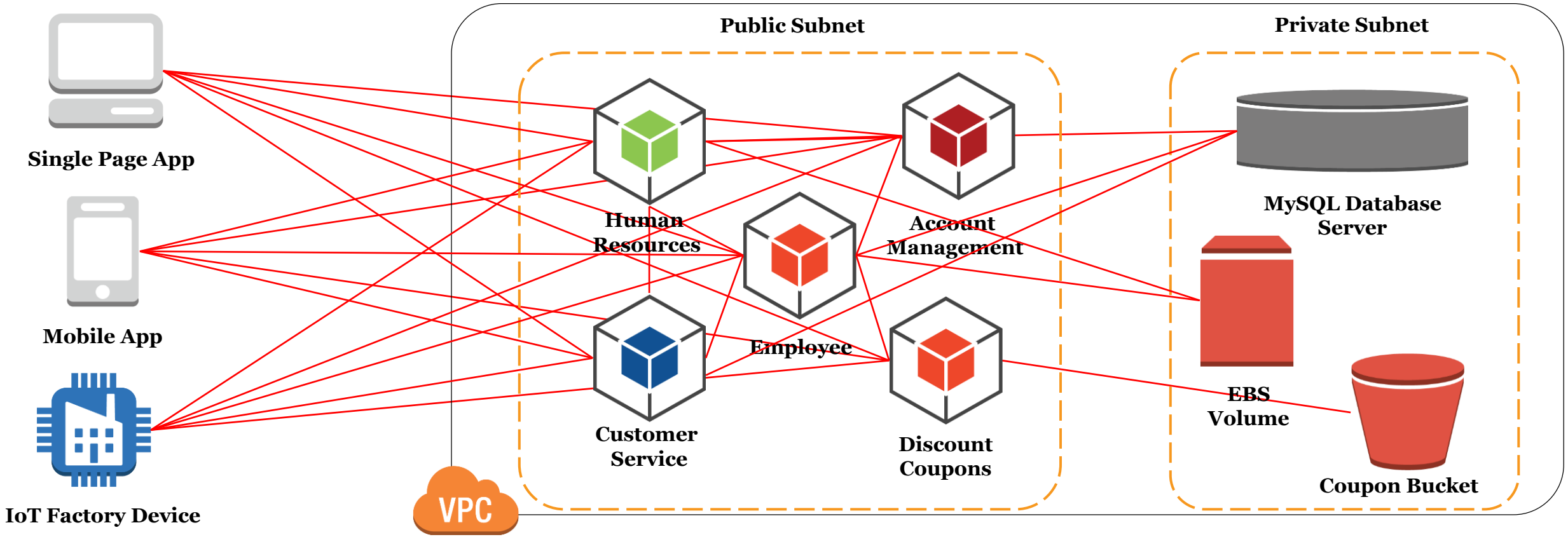
Microservice Architecture

- How does this change in a microservice architecture?



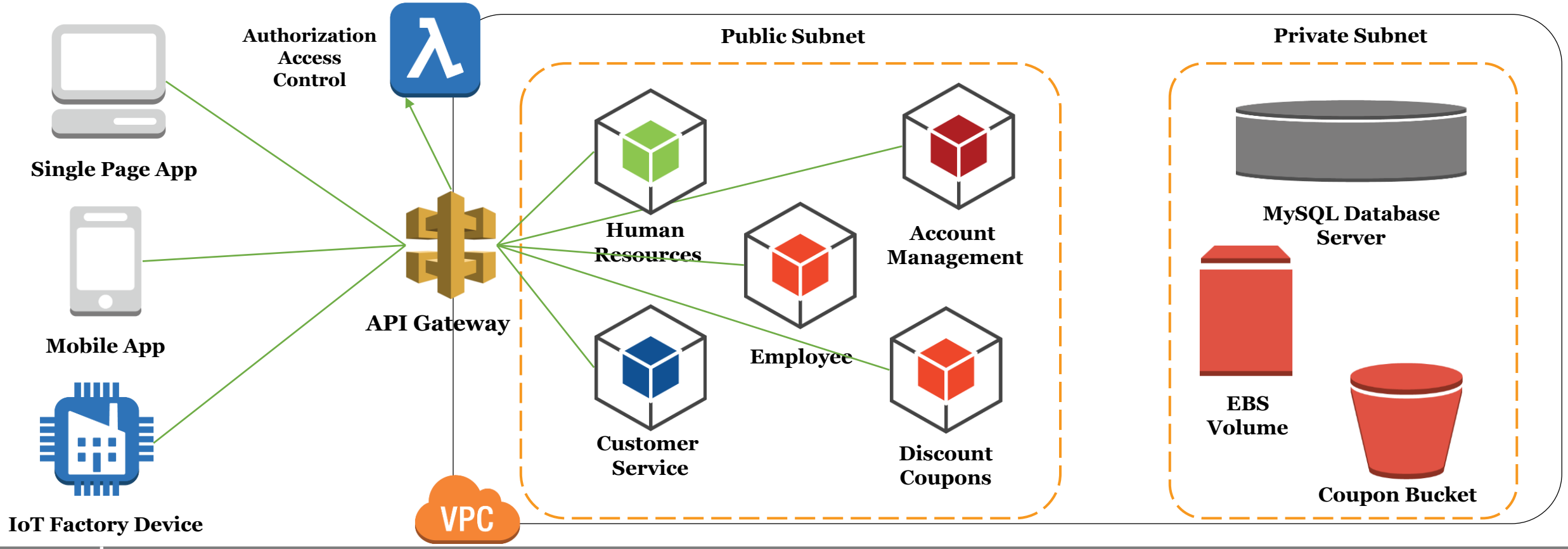
Microservice Architecture Attack Surface

- Consider the attack surface in a modern microservice architecture:



Microservice API Gateway Architecture

- Adding an API Gateway to provide perimeter security controls:



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)
    - Anyone can deploy a function
    - Cost to deploy a function is effectively zero (not charged for idle time)
    - Difficult to track and delete functions
  - Authentication and access control
    - Who has permission to deploy functions?
    - How do you lock down what each function can do?
  - Compliance
    - Which functions will handle sensitive data and where is it stored?
    - Are there multitenancy risks?
    - Is your implementation itself compliance?

Serverless and Application Security

- 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 Architecture

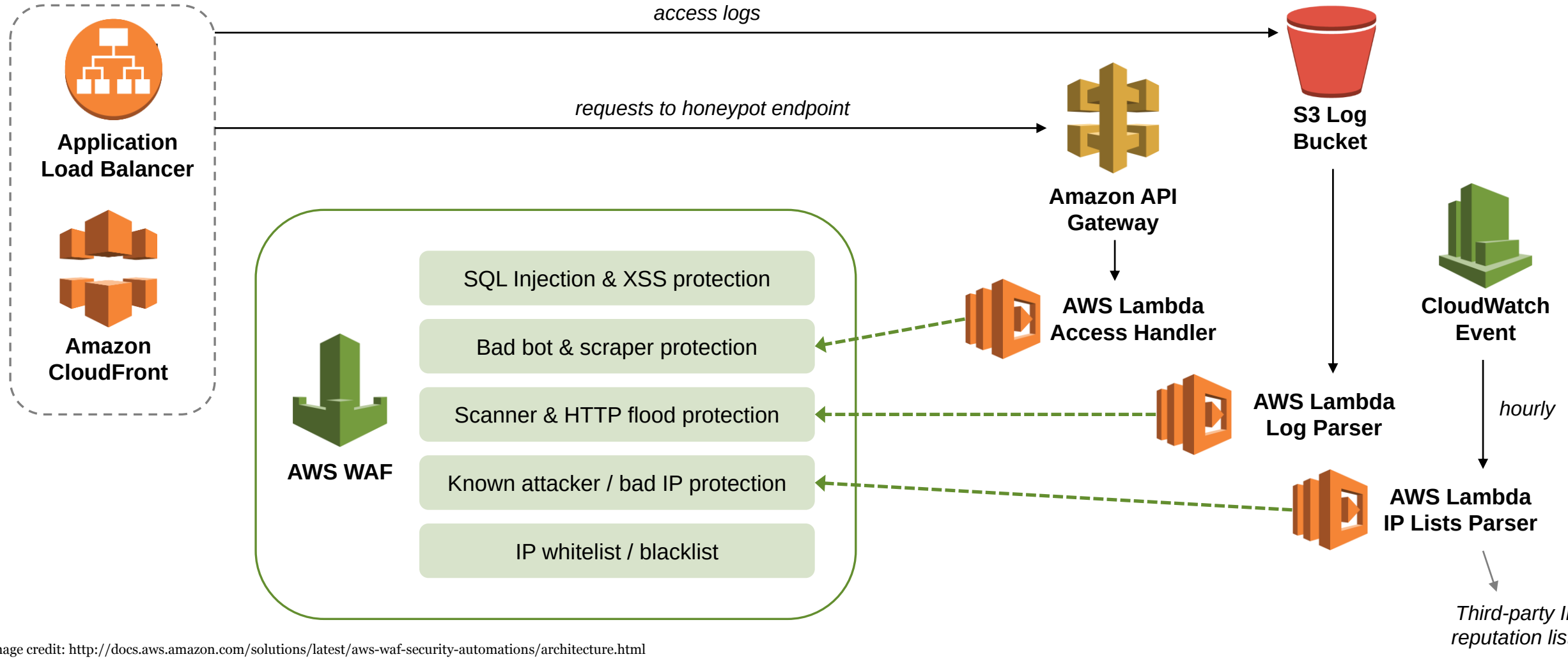


Image credit: http://docs.aws.amazon.com/solutions/latest/aws-waf-security-automations/architecture.html

#2 | Seize the Simple

Keep it Simple

"Simplicity is the ultimate form of sophistication."

– Leonardo Da Vinci

Hunt the Bug

Can you identify the bug in this code snippet?

```
1  <%
2  String theme = request.getParameter("look");
3  if (theme == null && session != null) {
4      theme = (String)session.getAttribute("look");
5  }
6
7  if (session !=null) session.setAttribute("look", theme);
8  %>
9
10 <link rel="stylesheet" type="text/css" media="all"
11     href="<%= request.getContextPath() %>
12          /ui/theme/<%= theme %>/colors.css" />
```

Hunt the Bug

Can you identify the bug in this code snippet?

```
1  <%
2  String theme = request.getParameter("look");
3  if (theme == null && session != null) {
4      theme = (String)session.getAttribute("look");
5  }
6
7  if (session !=null) session.setAttribute("look", theme);
8  %>
9
10 <link rel="stylesheet" type="text/css" media="all"
11     href="<%= request.getContextPath() %>
12         /ui/theme/<%= theme %>/colors.css" />
```

AngularJS ngSanitize

- Output encoding automatically applied for:
  - HTML tags using ngBind directive (e.g. **ng-bind**)
  - Output from Angular expressions (e.g. **{{var}}**)
- When certain HTML tags should be allowed, use the ngBindHtml directive (e.g. **ng-bind-html**)
  - Output is tokenized
  - Tokens are passed through a whitelist
  - Non-whitelist tokens are encoded

ngSanitize Examples

- ngBind

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

- Angular expression

```
<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>
```

Content Security Policy (CSP)

- AngularJS uses features that violate standard CSP rules
- eval() and Function(string) generated functions are used to improve performance when evaluating expressions
- Inline styles are used to inject custom styling (e.g. ngCloak and ngHide)

  Linking to angular-csp.css will allow the directives to work when CSP is blocking inline styles

ngCsp

- The ngCsp (e.g. **ng-csp**) directive will instruct AngularJS to not use CSP-breaking features

```
<!doctype html>
<html ng-app ng-csp>
 ...
</html>
```

Static Analysis Tools

Security tools for static analysis:

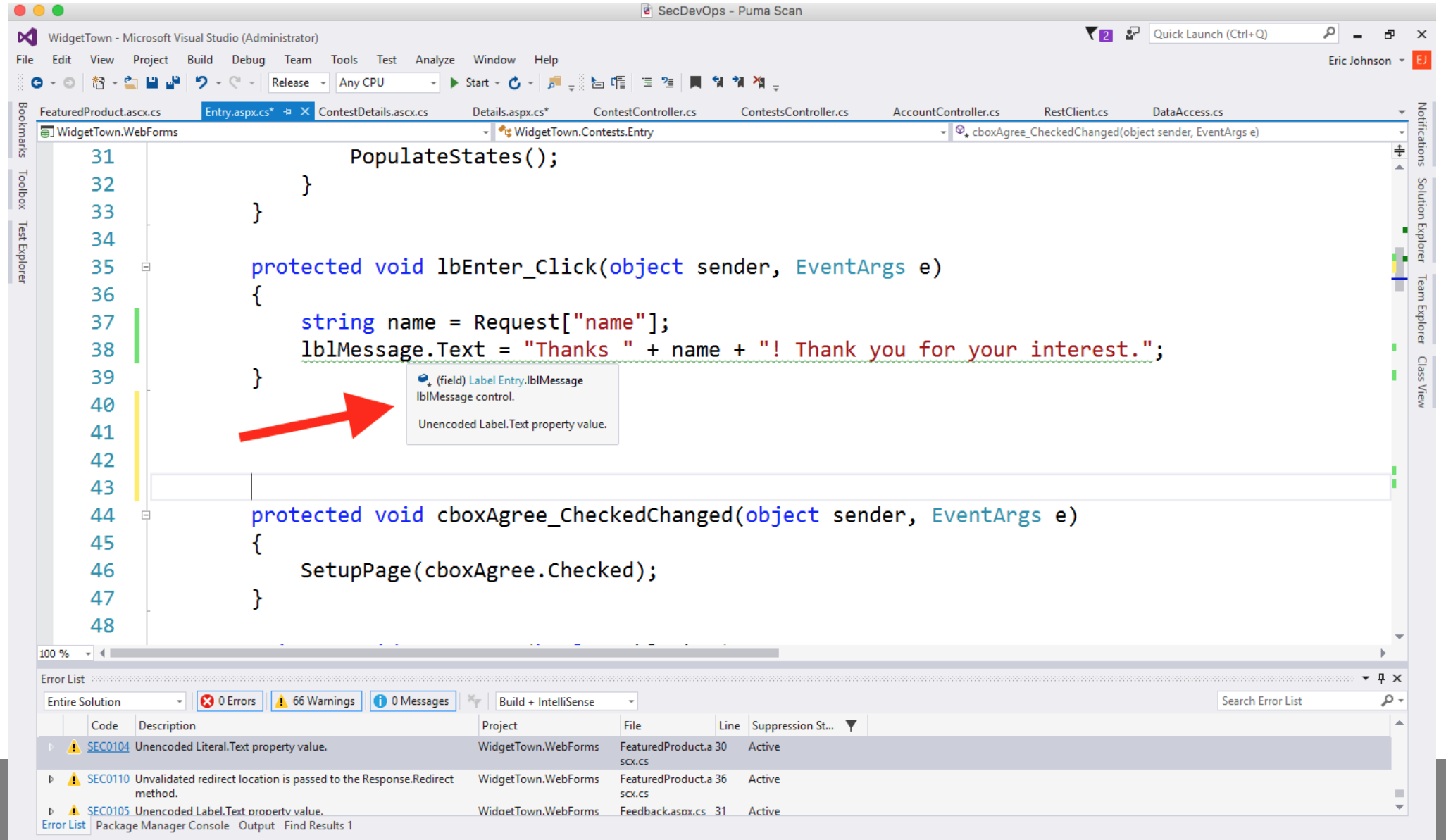
- Free / open source:

  Find Security Bugs, Phan, Puma Scan, Brakeman, Bandit, Flawfinder, QARK

- Commercial:

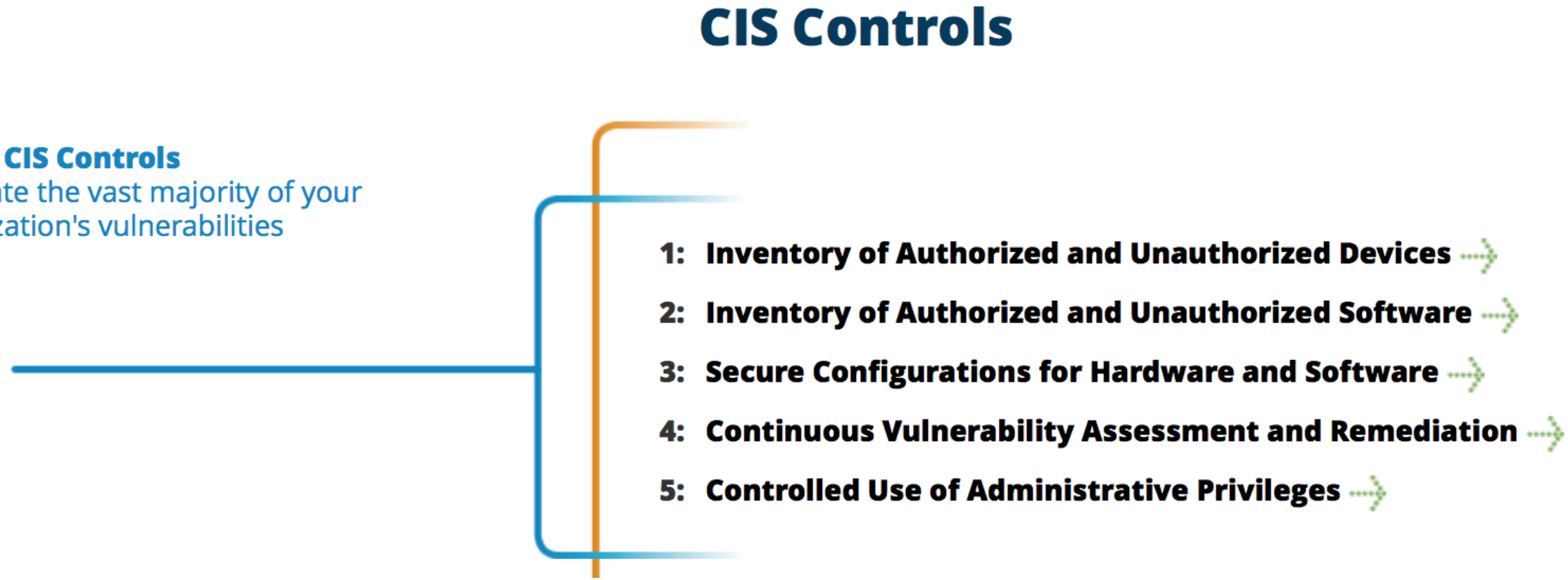
  HP Fortify, Checkmarx, Coverity, IBM AppScan Source, Klocwork, Veracode, Brakeman Pro

Secure Code Spell Checker



#3 | Automate Everything

Critical Security Controls (CSC)



Infrastructure as Code

- Different approaches to set up and manage systems
  1) Traditional: manual checklists and scripts, ad hoc changes/fixes made by system administrators at runtime
  2) Modern: treating Infrastructure as Code and configuration management as system engineering
- Configuration management with scripts is not scalable or traceable and is error prone
  - Leads to configuration drift over time
- Configuration management tools
  - Chef, Puppet, Ansible, Salt/Saltstack, CFEngine

AWS CloudFormation Example

Creating an EC2 instance

```
1  InstancePublic:
2    Type: AWS::EC2::Instance
3    Properties:
4    IamInstanceProfile: !Ref InstanceProfilePhotoReadOnly
5    ImageId: !FindInMap [Images, !Ref "AWS::Region", ecs]
6    InstanceType: "t2.micro"
7    KeyName: "secretKey"
8    SecurityGroupIds:
9      - !Ref SecurityGroupPublic
10   SubnetId: !Ref SubnetPublic
11   UserData:
12     Fn::Base64:
13       !Sub |
14       #!/bin/bash -xe
15       yum update -y
```

Retrieving All EC2 Instances

Command line call to retrieve all AWS EC2 instances:

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

Output:

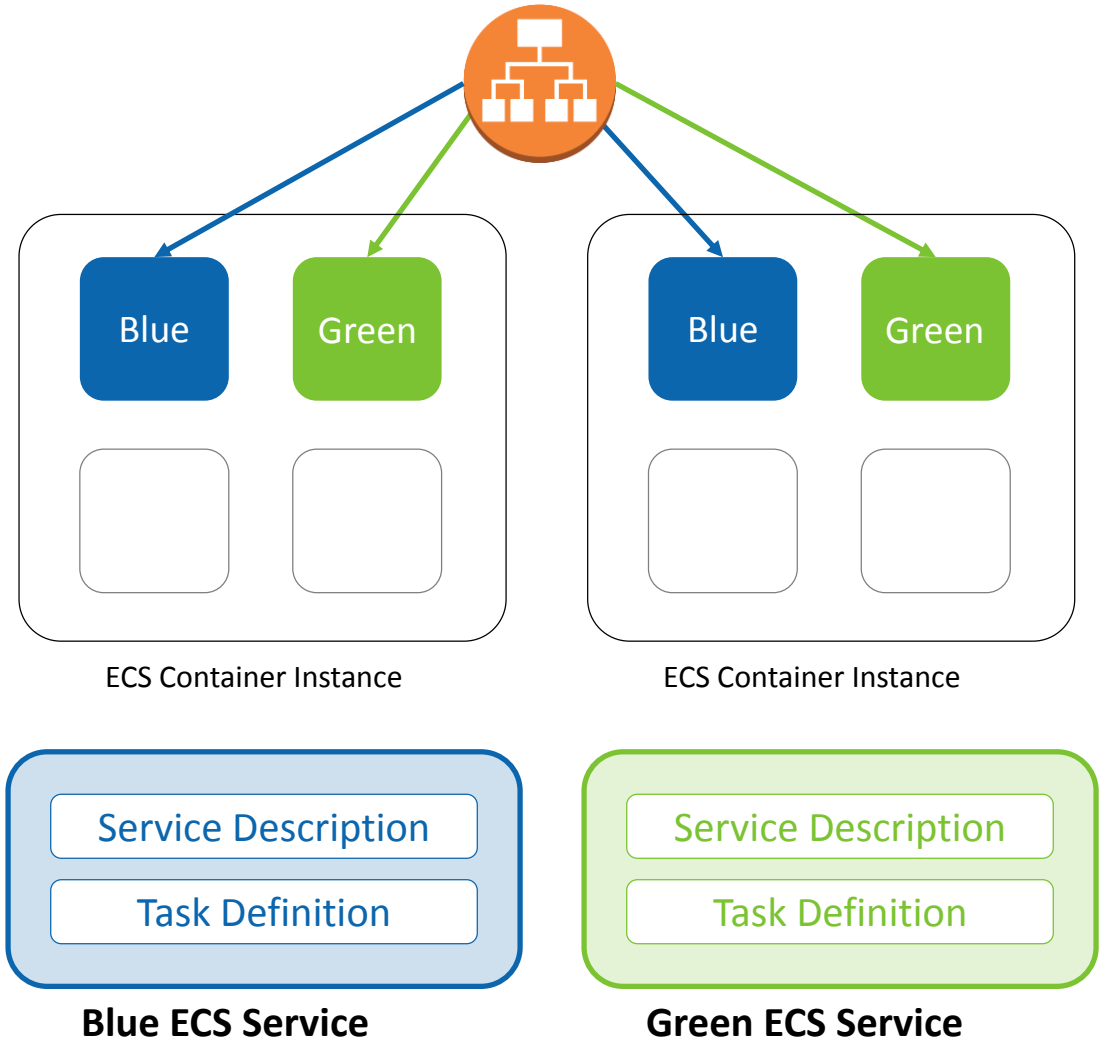
```
[ "2017-01-08T18:51:46.000Z", "t2.micro", "i-0500510e3f808d2ee", "sg-7caf4600"
  , "prod-springline-aws-web", "Springboot MVC target application"
  , "SANS\\app.user"
]
[
  "2017-01-08T18:55:02.000Z", "t2.micro", "i-0e74e490c2ebc5d37", "sg-79af4605"
  ,"qa-springline-aws-web", "QA Springboot MVC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

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

AWS EC2 Container Service (ECS) – Example

- Deployment process
  - Use original blue service and task def
  - Create new green service and task def
  - Map new green service to ELB
  - Scale up green service by increasing number of tasks
  - Remove blue service, setting tasks to 0



Swapping the AWS ECS Service

- Create a new “green” ECS Service

```
aws cloudformation deploy --template-file green-web-ecs-service.yaml --stack-name green-web-ecs-service
```

- Increase the desired count for the “green” service

```
aws ecs update-service --cluster DM-ecs --service $GreenService --desired-count 1
```

- Turn off the “blue” service when ready

```
aws ecs update-service --cluster DM-ecs --service $BlueService --desired-count 0
```

Getting to Yes

Three Keys for Security Success

Champion Change

Seize the Simple

Automate Everything

Questions?

SANS

Frank Kim
frank.kim@thinksec.com
@fykim