abhaybhargav

Pwning Serverless Applications!

we45

Lab System

abhaybhargav

- Lab Instructions: `dc27.we45.training`
- Event Code: `dc27-sls-workshop-appsec-XtRHD`

we45

we45

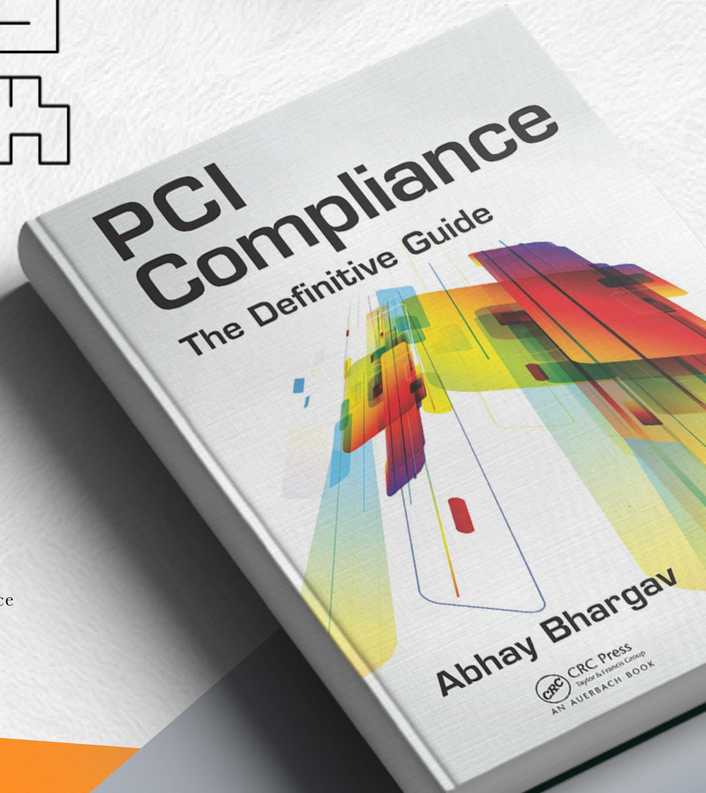
Yours Truly

- Founder @ we45
- Chief Architect - Orchestron
- Avid Pythonista and AppSec Automation Junkie
- Speaker at DEF CON, BlackHat, OWASP Events, etc world-wide
- Lead Trainer - we45 Training and Workshops
- Co-author of Secure Java For Web Application Development
- Author of PCI Compliance: A Definitive Guide

ORCHESTRON
by we45

THREAT PLAYPEN

DVFaaS
Damn Vulnerable Functions as a Service



we45

Follow/Connect

abhaybhargav

- Twitter: @abhaybhargav
- LinkedIn: linkedin.com/in/abhaybhargav
- Blog: we45.com
- Personal Blog: abhaybhargav.com
- Post regularly on blog and youtube

we45

we45

With me is…

abhaybhargav

- Tilak.T - Senior Solutions Engineer, we45
- Nithin Jois - Senior Solutions Engineer, we45
- Speakers and Trainers at Multiple International Events
- Accomplished Engineers with extensive DevSecOps Expertise

we45

we45

Learning Objectives

abhaybhargav

- An Intro to Serverless and Functions-as-a-Service Tech
- A View into popular deployment models and related technologies
- Serverless Security Threats and Common Vulnerabilities
- Attacking Serverless Deployments
- A View into defending serverless apps

we45

we45

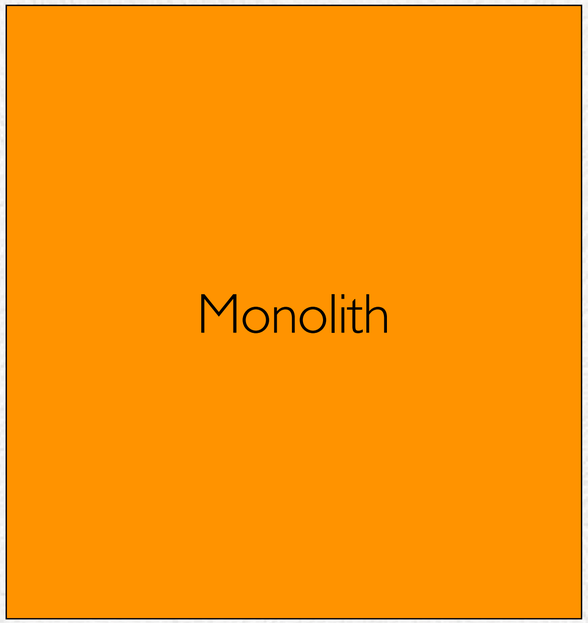
abhaybhargav

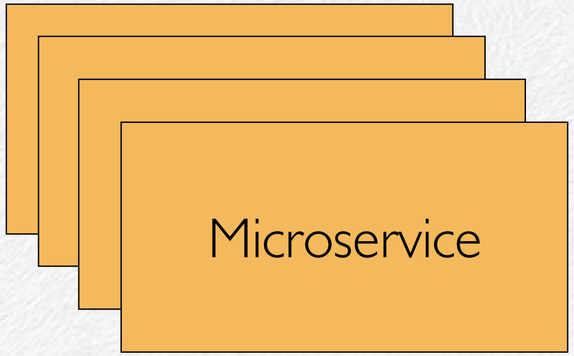
An Intro to Serverless Tech

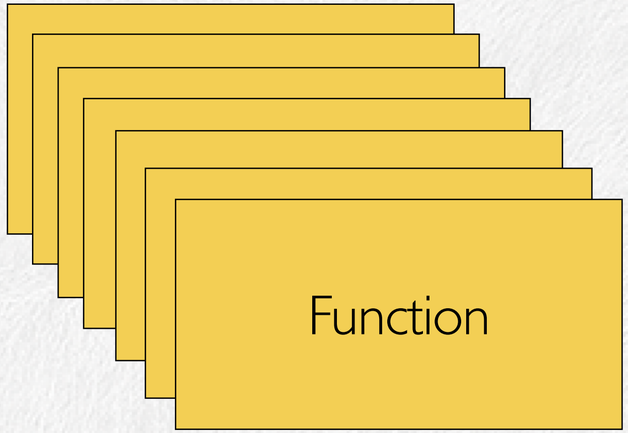
we45

Progression to Serverless

abhaybhargav





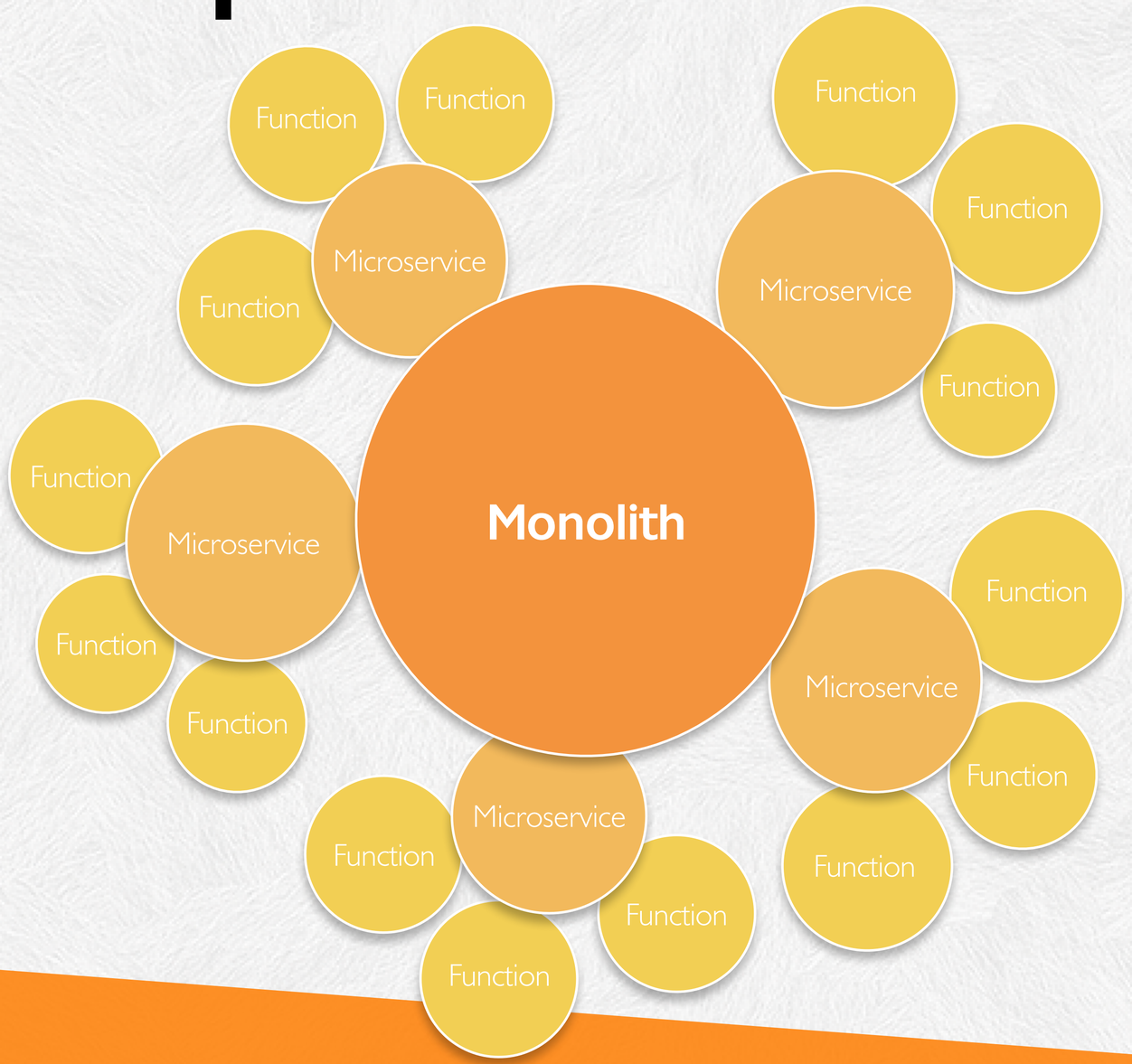


we45

we45

Another representation

abhaybhargav



Monolith

Serverless - Key Chars

abhaybhargav

- Two common variants “Backend-as-a-Service” (BaaS) or “Functions-as-a-service” (FaaS)
- More commonly used as FaaS - Implementation with Ephemeral Containers
- Extension to MicroServices Architectures
- Event-Driven, Horizontally Scalable, Framework Agnostic

Serverless - Key Chars (2)

abhaybhargav

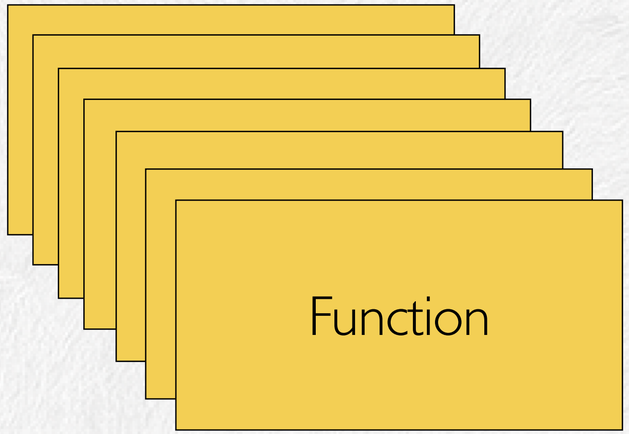
- FaaS functions are “first-class” and do not need a specific framework or library
- Architectural restrictions - state and execution duration
- Supports events from other applications (S3, SNS, SQS, etc) apart from user events
- Supports HTTP by default
- Behind an API/Event Gateway

we45

we45

Summary

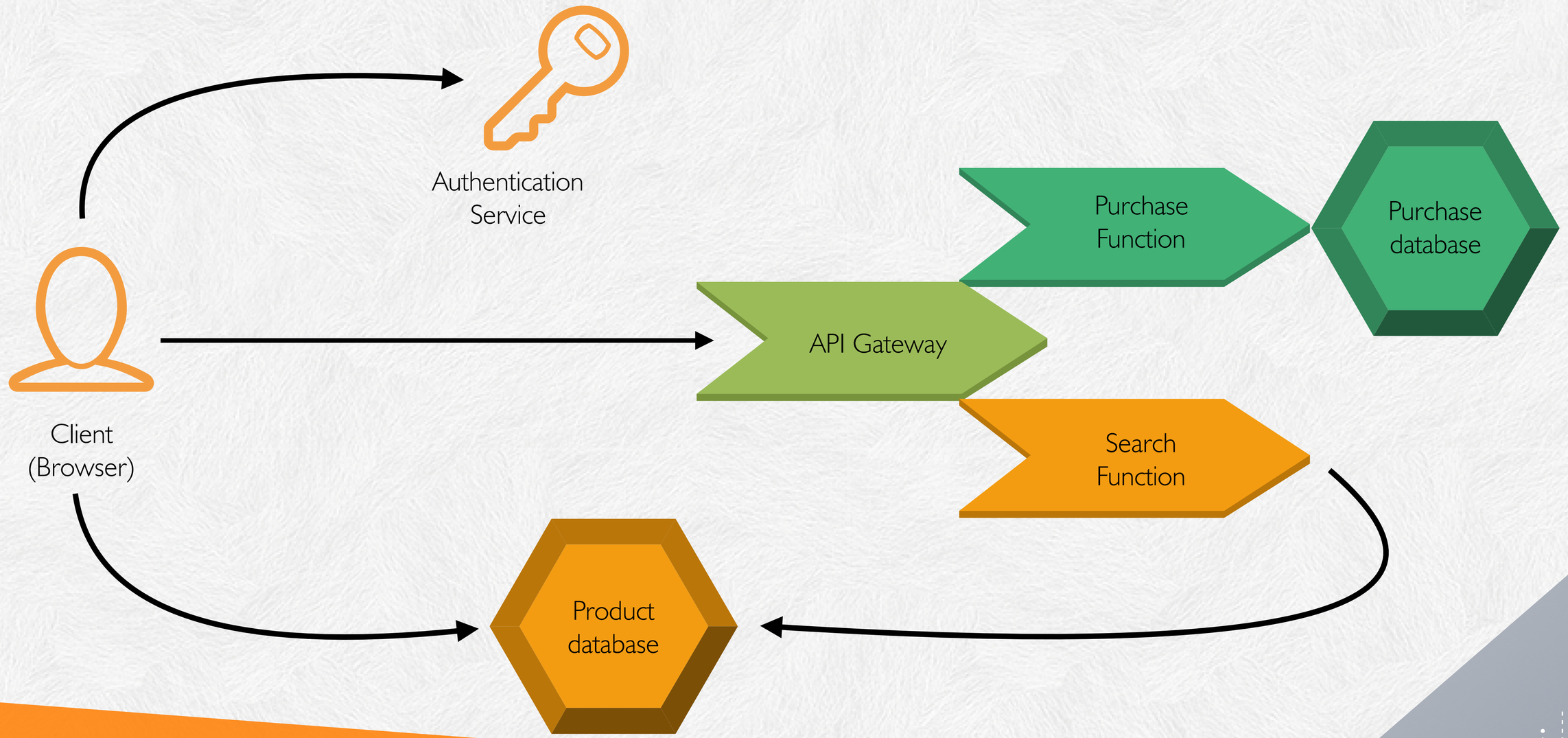
abhaybhargav



- Short lived
- No ports
- No state
- Single purpose

A View of Serverless Architecture

abhaybhargav



Benefits

abhaybhargav

- No Server management or Ops - FTW!!
- Pay for what you use!
- Horizontal Scalability - FTW
- No need to manage containers/orchestration, etc
- Cloud Providers - Extensive Tracing API and Monitoring tools (although not used that way)

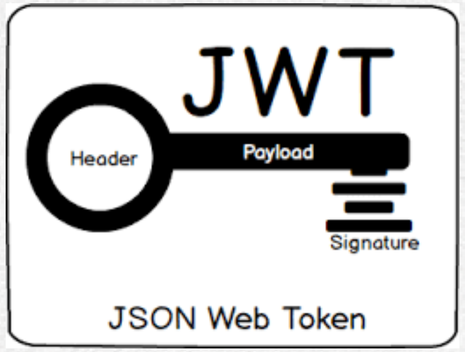
Demerits

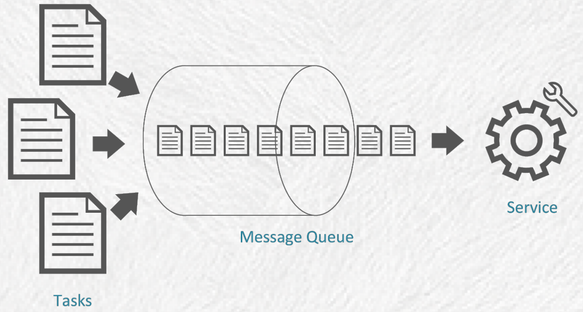
abhaybhargav

- Vendor Lock-in
- Visibility/Debugging (a Distributed Computing/MicroServices Problem)
- Security Concerns - Where's there's Visibility Problems, there are security problems
- Testing
- Function Execution Duration - Limitations in long-term functions
- Architectural Changes
- Limited Framework Support - Major "Batteries included" frameworks don't support Serverle
  FaaS type implementations

Common Players - Serverless

abhaybhargav

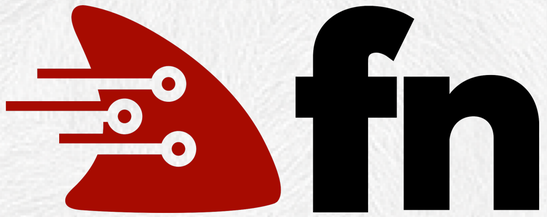




we45

we45

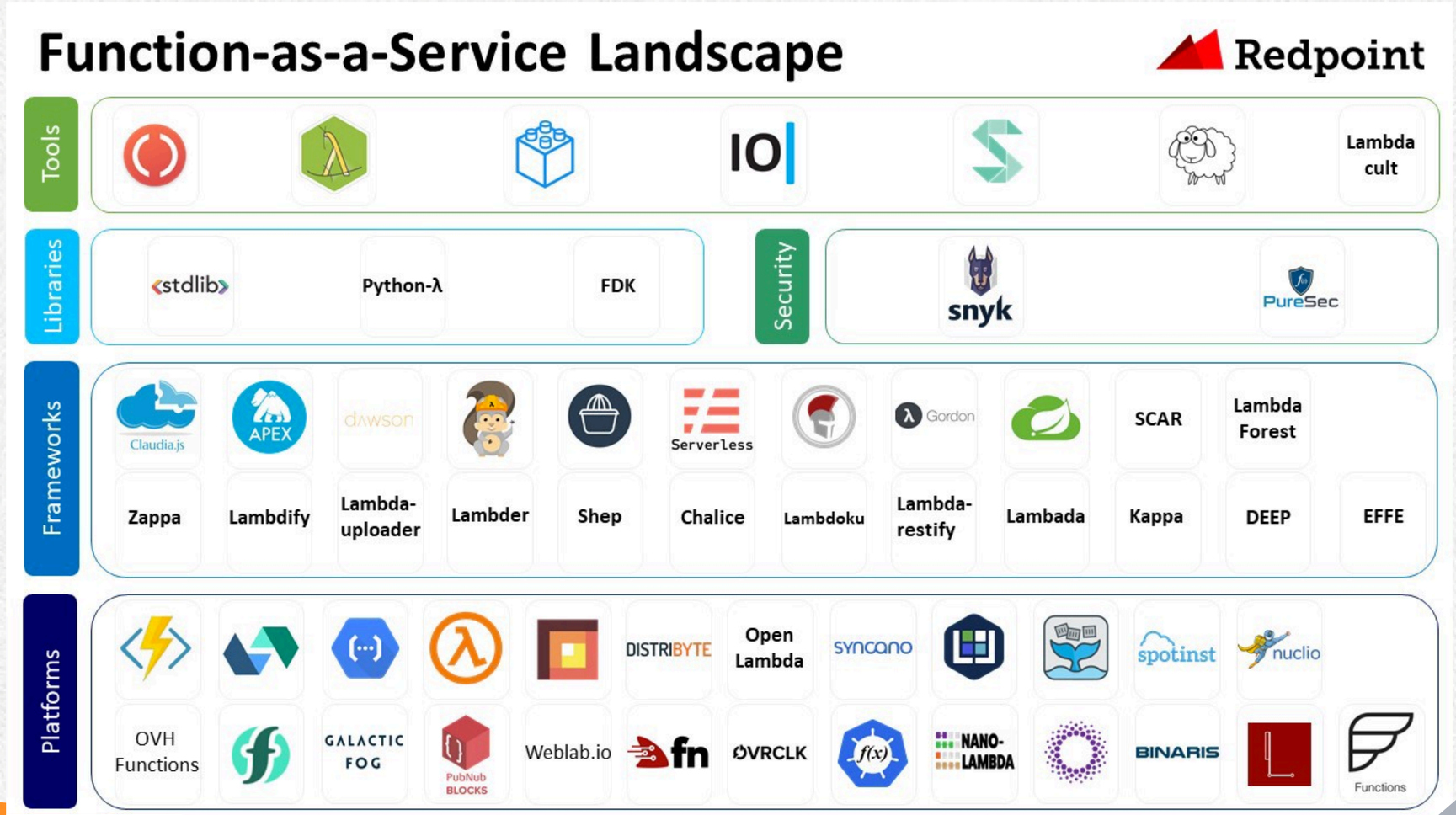
Serverless Solutions



OPENFAAS

The Serverless Landscape

abhaybhargav



Serverless Function Lifecycle

abhaybhargav

Look for a “warm” container

A warm container is a container that previously serviced a request to the same function and has been kept loaded to handle another request

An event trigger occurs

The FaaS infrastructure determines which function should be a trigger and validates the security requirements before invoking an instance of the function.

Otherwise, start a new container

If there are no “warm” containers or all are in the process of handling a different request, the platform will “cold start” a new one, assigning an idle container, loading, decompressing, and initializing your code. This takes much longer so it’s in everyone’s interest to avoid cold starts.

The container hangs around for a while

Based on complex and ever-changing decision-making logic, the platform may choose to immediately recycle the container or keep it around for hours or even days.

The function handles the request

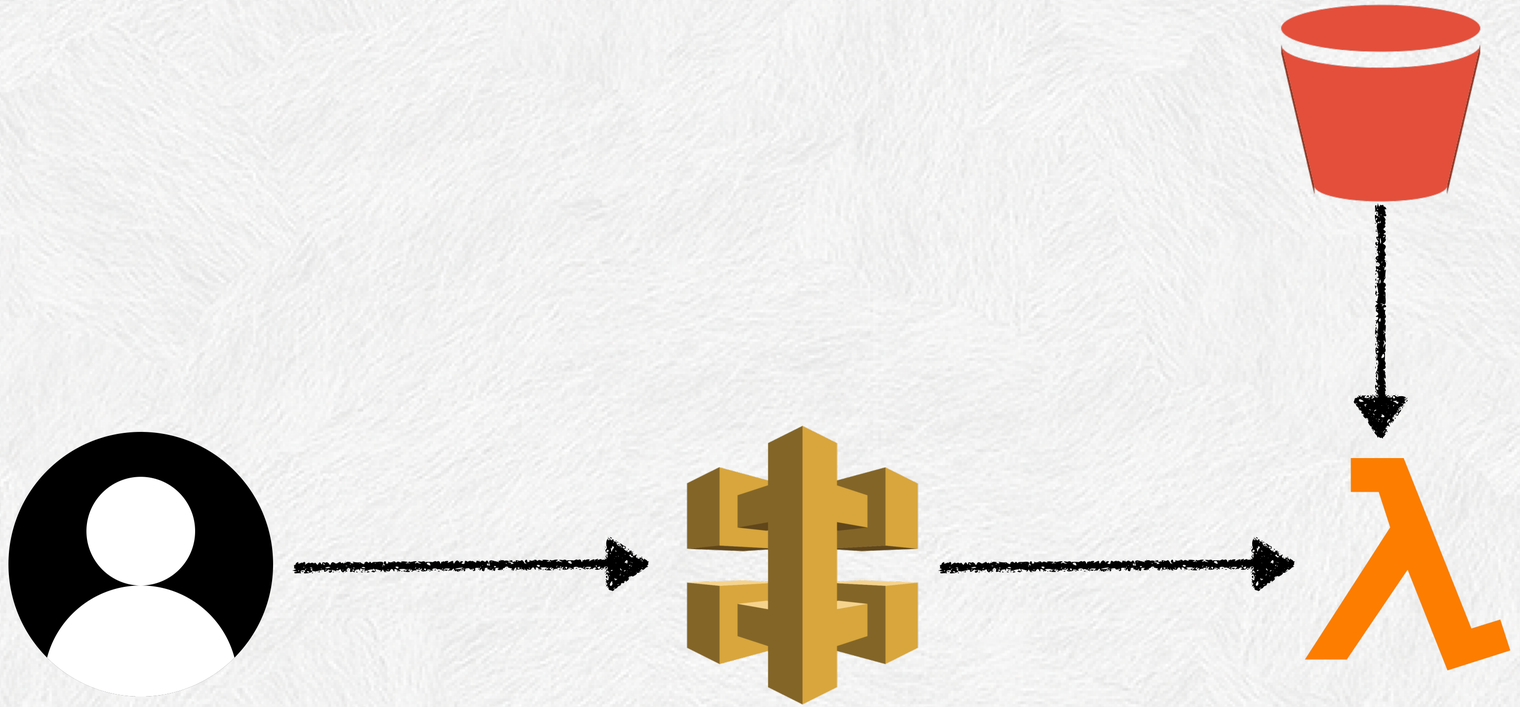
The function is then called to handle the request. Once it’s done, any response is returned the caller, and the container is “frozen” but “kept warm” (i.e. still resident in memory on some machine)

we45

we45

Lifecycle

abhaybhargav



Deploy into Lambda with zip file

Containers are “thawed” when they are invoked again

Additional Containers are spawned based on concurrent invocations

Function is invoked launching a container to run. Destroyed after.

we45

we45

Lab: Basic API

abhaybhargav

Lab: Basic Events

abhaybhargav

Security Considerations - FaaS

abhaybhargav

- No* Frameworks => Back to Plain ol' platform code
- No Network Attack Surface
- Observability/Debugging is a challenge
- Events from Multiple Sources
- Highly disciplined approach to Architecture

we45

we45

abhaybhargav

reading between the lines….

Considerations - FaaS

abhaybhargav

- No* Frameworks => Back to Plain ol' platform code
- No Network Attack Surface
- Observability/Debugging is a challenge
- Highly disciplined approach to Architecture

- **No Batteries included Security Features (Frameworks)**
- **DIY Validation**
- **Access Control per Function**
- **Logging Per Function**
- **and other things we don't too too well.....**

Considerations - FaaS

abhaybhargav

- No* Frameworks => Back to Plain ol' platform code
- No Network Attack Surface
- Observability/Debugging is a challenge
- Highly disciplined approach to Architecture

- **Monitoring Attacks is a challenge unless you architect for it**
- **Security Logging => FUHGEDDABOUDIT!**

Considerations - FaaS

abhaybhargav

- No* Frameworks => Back to Plain ol' platform code
- No Network Attack Surface
- Observability/Debugging is a challenge
- Events from Multiple Sources

- **Functions triggered from events like S3, SNS, SQS,etc**
- **Larger Attack Surface**
- **Traditional Security Controls - WAFs, etc may be ineffective**
- **DAST/Testing is hard to exec**

we45

we45

abhaybhargav

Serverless - Security

we45

Serverless - Security Overview

abhaybhargav

- Functions as REST API - Similar to other Web Services Security Issues
- Events add to the Attack Surface
- Functions are *usually* stateless
- Not *only* HTTP
- “Testability” of a Serverless deployment is limited

we45

we45

High-Level Security Challenges - Serverless

abhaybhargav

- Authentication and Access Control
  - Difficult to Centralize - Unless you use BaaS Auth Providers
  - Poor implementation of Stateless Authentication/Authorization (JWT)
  - IAM Privileges and Permissions are often the cause of great insecurity
- Validation
  - Few libs for input validation (no framework support)
  - Validation code usually needs to go into every function you write

High-Level Security Challenges - 2

abhaybhargav

- Interfaces:
  - Serverless Apps can behave as API or work as event-driven services
  - Traditional Security (WAFs/IPS) cant protect against several events*
- Vulnerability Assessment
  - Testing for Security vulns (especially automated is hard)
  - SAST and SCA must be relied on extensively for identifying security issues

Serverless - Security Responsibility Model

abhaybhargav

Application Owner

Responsible for security "in" the cloud

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

FaaS Provider

Responsible for security "of" the cloud

- Operating systems + Virtual machines + computers
- Compute
- Storage
- Database
- Networking
- Regions
- Availability zones
- Edge locations

we45

we45

Serverless - Top 10

abhaybhargav

- New Project from OWASP, Protego and PureSec
- Interesting Take on highlighting specific Serverless Security Issues with the OWASP Top 10 style Framework
- Captures a good subsection of the serverless risk landscape

we45

we45

Serverless Architectures Security - Top 10

abhaybhargav

- A1: Injection
- A2: Broken Authentication
- A3: Sensitive Data Exposure
- A4: XML External Entities
- A5: Broken Access Control
- A6: Security Misconfiguration
- A7: Cross-Site Scripting (XSS)
- A8: Insecure Deserialization

we45

we45

abhaybhargav

Function Data Event Injection and other Remote Code Execution

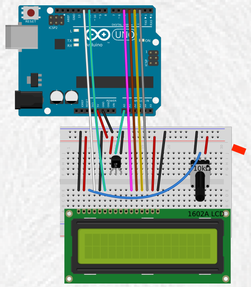
Function Data Event Injection

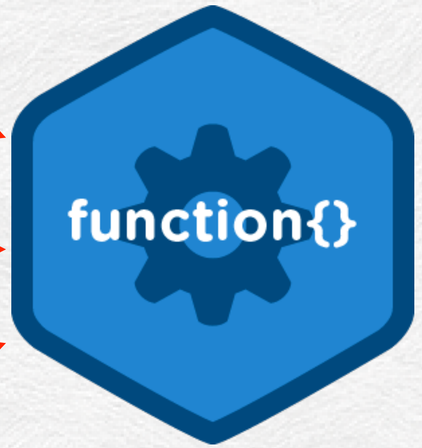
abhaybhargav

- Injection is back!!
- Multiple Possibilities with Functions:
  - Insecure Deserialization
  - XXE
  - SQL Injection
  - NoSQL Injection
  - Server-Side Request Forgery
  - Template Injection

Function Data Event Injection - Sources

abhaybhargav



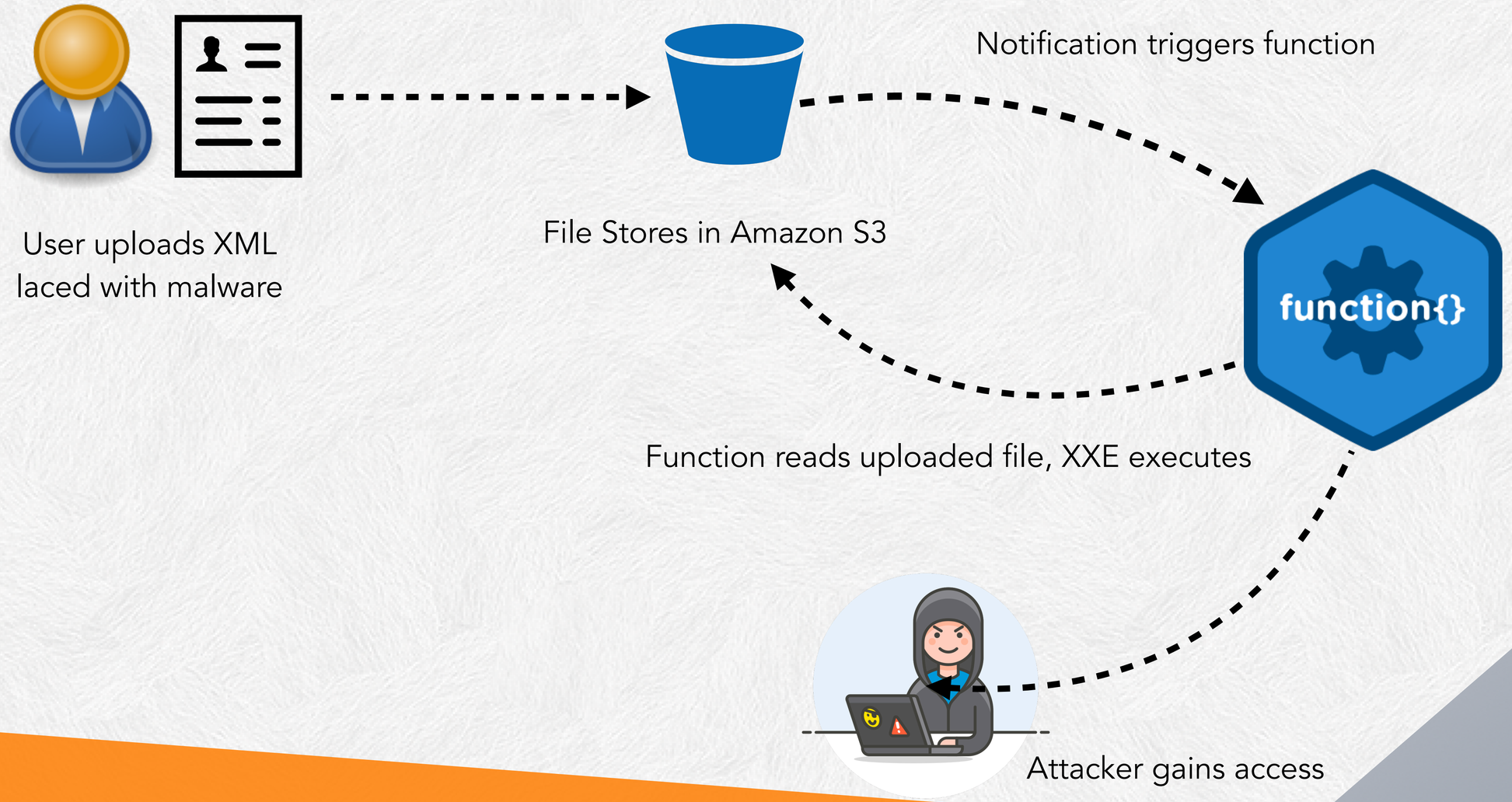


- Command Injection
- SQL/NoSQL Injection
- Insecure Deserialization
- XXE

we45

Case Study

abhaybhargav



Lab: XXE Event Injection

abhaybhargav

Challenges - Function Data Event Injection

abhaybhargav

- Hard to test for => Execution is largely Out-of-Band
- Hard to Protect with WAFs (other Network Security) => Several non-HTTP Protocols can be used to trigger this
- Wide variety of execution scenarios

Protections

abhaybhargav

- Consider Event Data Injection into your Threat Model
- Input Validation
- Static Application Security Testing
- Source Composition Scanning
- Other Protections:
  - WAF for HTTP(s) calls to your FaaS implementation

abhaybhargav

Broken Access Control

Focus Points

abhaybhargav

- Broken Authentication
- Attacks and Defenses against Stateless Authentication - JSON Web Tokens
- Bad/Good Practices for Identity and Access Management

we45

we45

Broken Authentication

abhaybhargav

- Authentication for Functions is hard => You need to implement Authentication in every function you write
- Authentication should ideally be centralized => Okta, Amazon Cognito, etc to make things easier
- FaaS implementations are especially complex as they receive events from multiple sources => All need authentication
- Functions might require different authentication schemes for different cloud services
- Lack of Tooling for testing Authentication

Access Control

abhaybhargav

- JSON Web Tokens => Attack and Defense
- IAM Configuration Practices and Missteps for FaaS implementations

abhaybhargav

Let’s talk about JWTs

we45

Got Stateless?

abhaybhargav

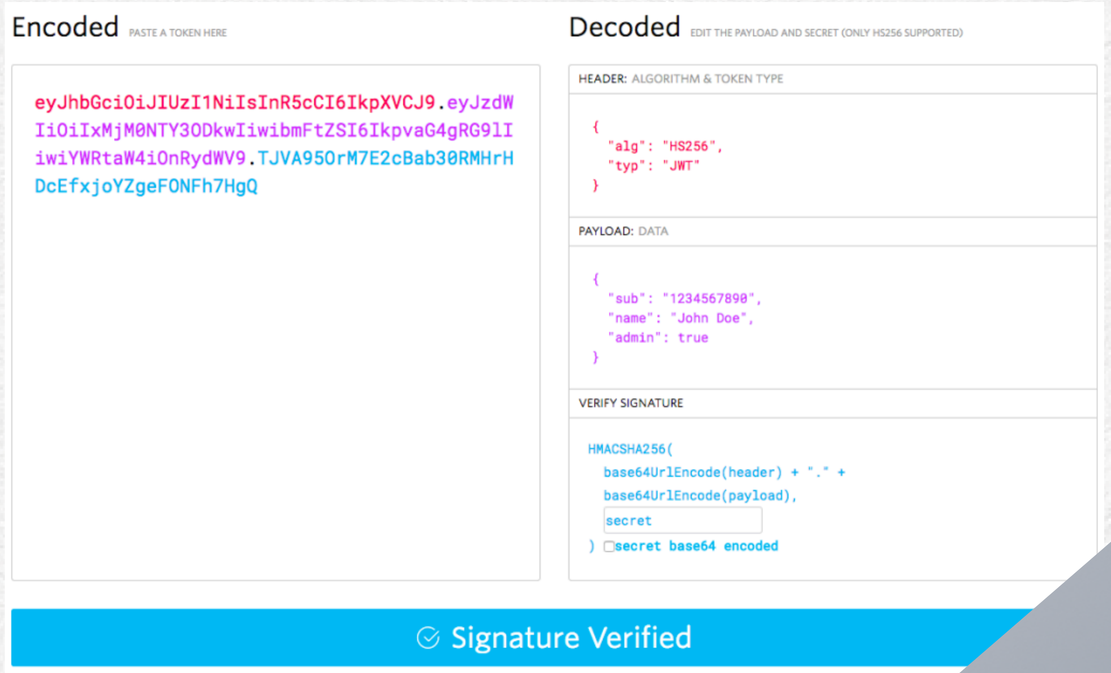
- JSON Web Tokens have emerged as a very popular (nearly de-facto) authorization practice for stateless authorization.

- Started with a standard called JOSE (JavaScript Object Signing and Encryption):
  - JSON Web Token (JWT)
  - JSON Web Signature (JWS)
  - JSON Web Key (JWK)
  - JSON Web Encryption (JWE)

JSON Web Token

abhaybhargav

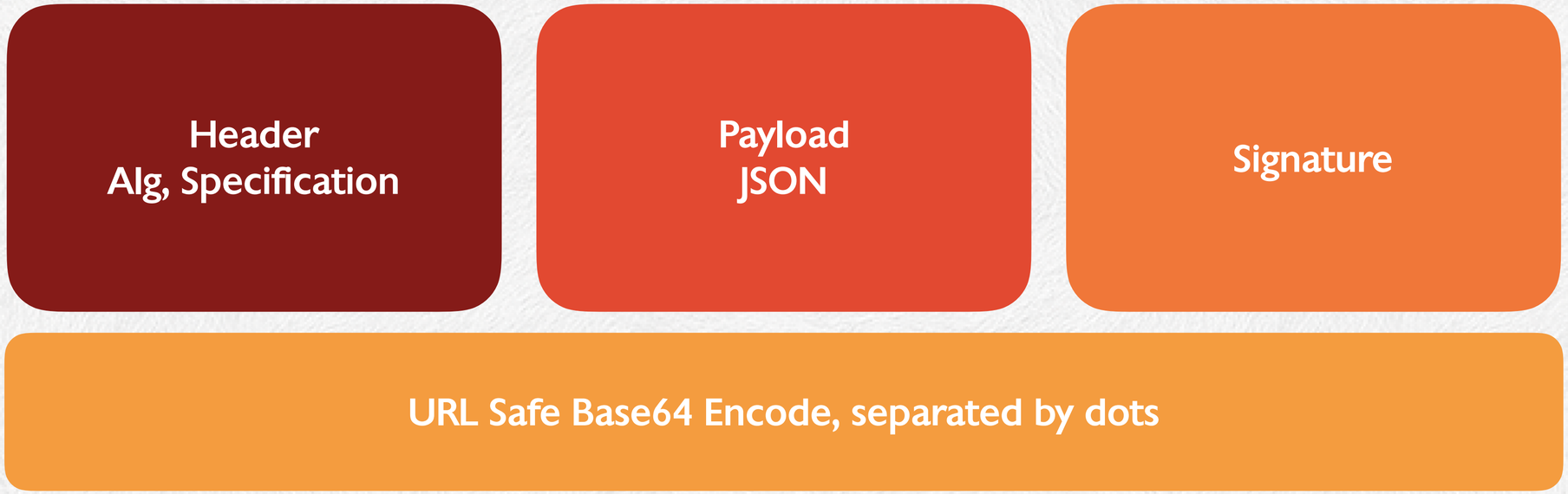
- JSON Web Tokens are the key piece of most stateless authorization systems
- The process is relatively simple (typically):
  - Once a user authenticates, the server generates some JSON payload (with some info) and signs the JSON payload with a key
  - This can be a HMAC Based Key (HS256) or a Asymmetric System (RS256)
  - The token is sent by the client (like a session cookie)
  - The server attempts to verify the token based on the signature and allows/disallows the user to perform actions



we45

Anatomy of a JWT

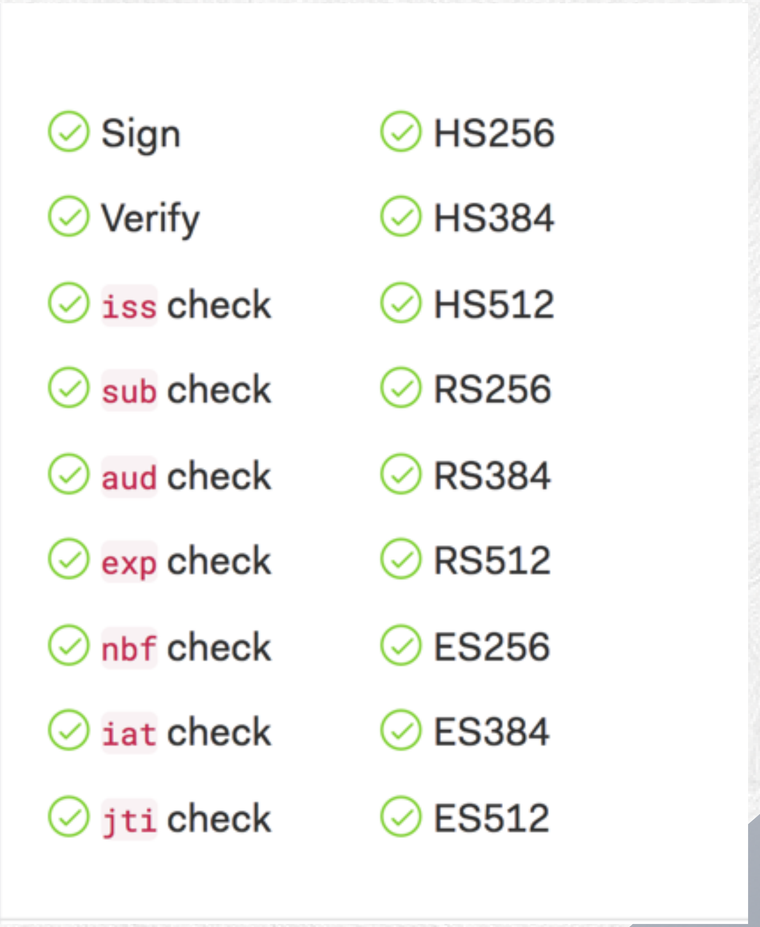
abhaybhargav



Who uses it? and How?

abhaybhargav

- Very very popular for REST API
- OAuth
- Popular systems like K8s use JWTs for Token based authorization
- Distributed Computing Systems/Microservices ❤️ JWTs



Lots of ways to get JWT wrong

abhaybhargav

- JWT allows for a “none” signature for a token. Yes. really :(
- When secrets aren’t really secrets
- Algo Confusion Attacks:
  - CVE-2017-11424
  - CVE-2015-9235
- JWT verification on non-unique private claims

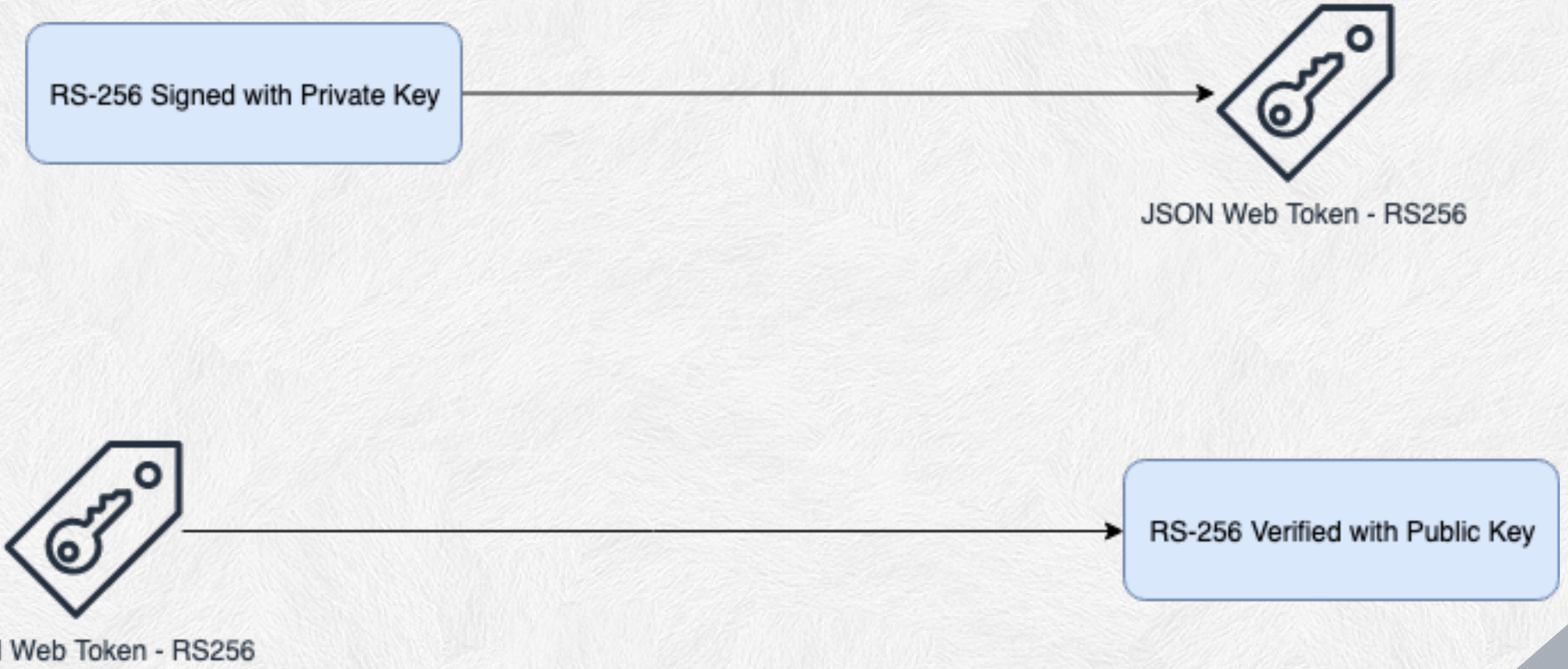
Installation Instruction

abhaybhargav

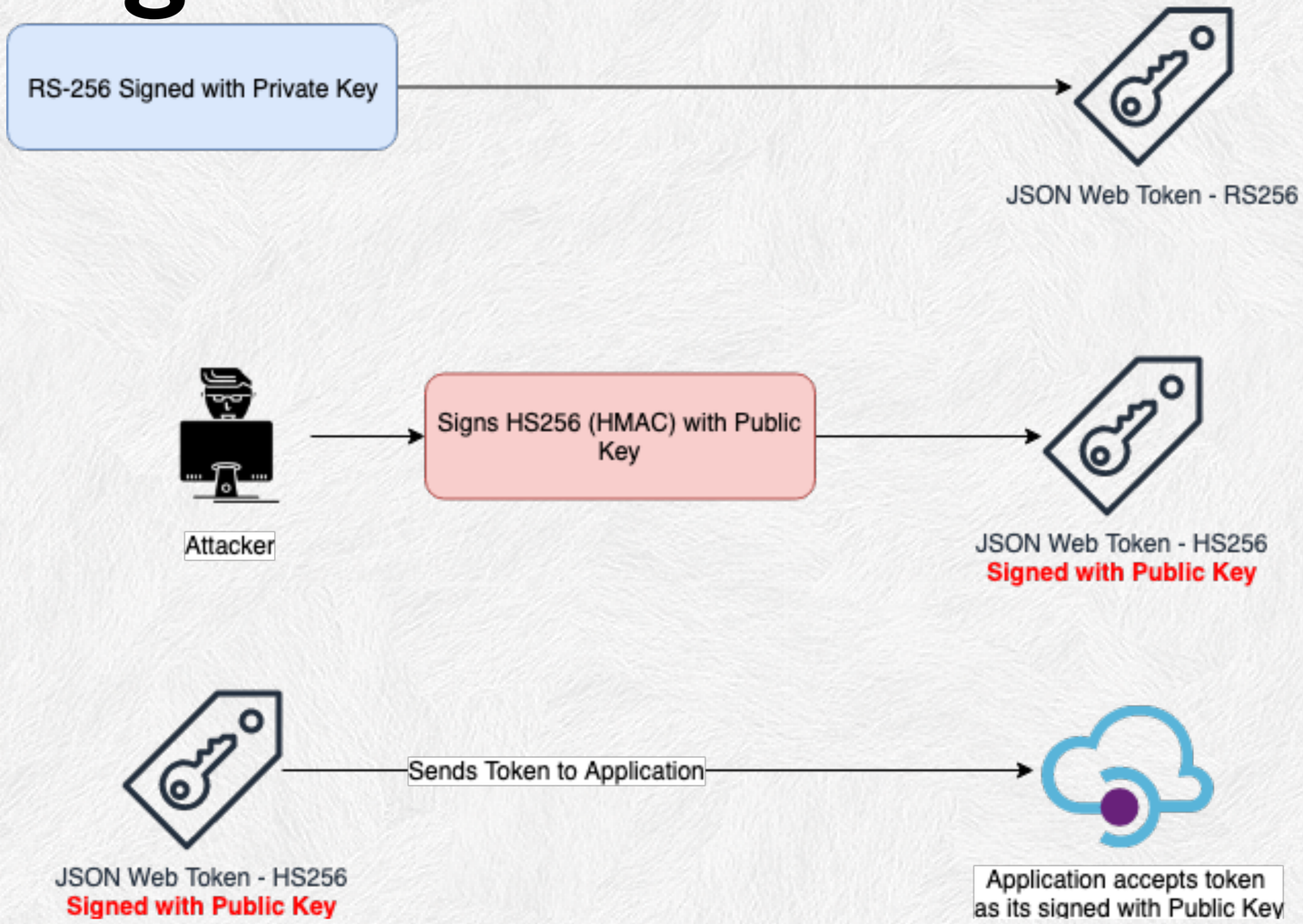
- `cd /root/labs/container_training/Serverless/Algorithm-Confusion`
- Install a couple of packages
  - `npm install jsonwebtoken@4.2.0`
  - `npm install colors`

JSON Web Token - Typical Flow

abhaybhargav

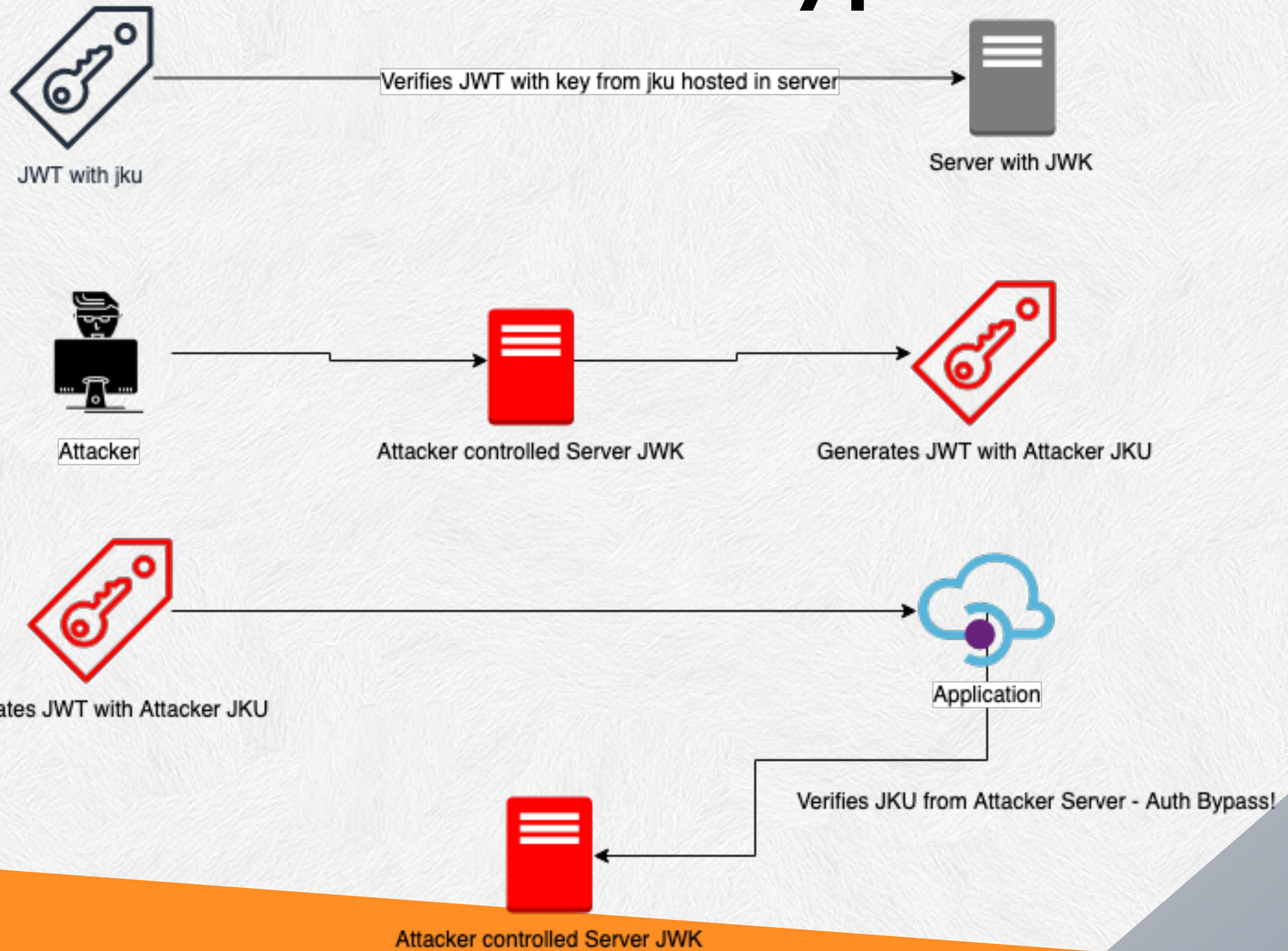


Lab: Algorithm Confusion



Lab: JWK Authentication Bypass

abhaybhargav



CORS and FaaS

abhaybhargav

- Cross-Origin Resource Sharing (CORS) is critical feature => Responses to Cross-Domain Requests
- Deployment Frameworks and Platforms usually provide it => **AND set it to * by default**
- CORS Functionality and specifics can be defined at the app or at the API Gateway level

we45

we45

abhaybhargav

IdentityAccessManagement - Function Privilege Management

we45

IAM & Other Misconfigurations

abhavbhargav

- Permissions are often the greatest bugbear in a FaaS implementation
- Devs tend to provide overly permissive capabilities for resources that interact with FaaS implementations
- Permissions are usually set in cloud IAM environments with Policies, Roles, etc
- This includes misconfigurations like Public S3 buckets and access to all DynamoDB tables, etc

Examples of IAM

abhaybhargav

```
- Effect: Allow
  Action:
    - 'dynamodb:*'
  Resource:
    - 'arn:aws:dynamodb:us-east-1:****************:table/TABLE_NAME'
```

Allows ALL actions on a DynamoDB Table

```
- Effect: Allow
  Action:
    - dynamodb:PutItem
  Resource: 'arn:aws:dynamodb:us-east-1:****************:table/TABLE_NAME'
```

Only PUT allowed on Table

Security Measures

abhaybhargav

- Validate your Functions for “Least Privilege” Requirements before provisioning resources and permissions
- Function Deployment Managers often create permissions (Roles and Policies). Audit them
- Regularly validate IAM privileges through manual/ automated means

A Note on IAM and Serverless Deployment Tools

abhaybhargav

- Serverless Deployment Apps are very popular
  - GoServerless
  - ClaudiaJS
  - Zappa
  - Apex
  - Architect
  - Sparta

They have a big impact on IAM!

alhashbhargav

- They NEED to run with High Privileges
- The Deployment Manager - auto-provisions roles and policies for that role
- They allow granular policy creation for your cloud environment

Lab: Deploy Architect Function in your AWS Lambda

abhaybhargav

```
service: serverless-rest-api-with-dynamodb
frameworkVersion: "2.0"
provider:
  name: aws
  runtime: nodejs6.10
  environment:
    DYNAMODB_TABLE: test-table
  iamRoleStatements:
    - Effect: Allow
      Action:
        - dynamodb:Query
        - dynamodb:Scan
        - dynamodb:GetItem
        - dynamodb:PutItem
        - dynamodb:UpdateItem
        - dynamodb:DeleteItem
      Resource: "*"
```

Example of a poorly configured IAM Policy

DynamoDB Injection

abhaybhargav

- We identified and reported instances where DynamoDB Injection could occur
- This attack is made much worse with privileged access to DynamoDB tables
- Its predicated on abusing the scan function of DynamoDB
- Original Article and Blogpost: https://medium.com/appsecengineer/dynamodb-injection-1db99c2454ac

we45

DynamoDB Injection

abhaybhargav

```
client.scan(TableName = 'dynamo-user', Select = 'ALL_ATTRIBUTES',
ScanFilter = {
'first_name':
  {"AttributeValueList": [{"S": "Joe"}],
  "ComparisonOperator": "EQ"}
})
```

EQ|NE|IN|LE|LT|GE|GT|BETWEEN|
NOT_NULL|NULL|CONTAINS|
NOT_CONTAINS|BEGINS_WITH

Standard “scan” with DynamoDB

```
client.scan(TableName = 'dynamo-user', Select = 'ALL_ATTRIBUTES',
ScanFilter = {'first_name': {"AttributeValueList": [{"S": "*"}],
"ComparisonOperator": "GT"}})
```

Equivalent of ‘OR 1=1, Retrieves all values from the Table

we45

abhaybhargav

Lab: DynamoDB Injection

we45

Protections

abhaybhargav

- Audit Deployment Scripts (yaml/Terraform, etc) for IAM privs provided to functions
- Audit Roles and Policies for IAM extensively before deploying the policy to production
- Consider using IAM audit tools and provisioning checks like `serverless-puresec-cli` to derive the appropriate IAM policy

we45

we45

abhaybhargav

Serverless Vulnerability Assessment

we45

LambdaGuard

abhaybhargav

- Python based Lambda focused security scanning tool from SkyScanner Engineering
- Looks for several common misconfiguration and security errors in Lambda deployments and related components like SQS, SNS and so forth
- Delivers a useful Dashboard-like HTML report with details of the findings

we45

we45

Lab: LambdaGuard

abhaybhargav

abhaybhargav

Denial of Service and Financial Resource Exhaustion

we45

Availability Challenges - Serverless

abhaybhargav

- Distributed Denial of Service Attacks are on the up since 2016
- While Serverless Platforms provide Automated Scalability and Availability:
  - you do have to keep specific things in mind
- Serverless Resource Exhaustion can have impact on:
  - Financial Limits
  - Availability => VPC IP Addresses

we45

we45

Serverless Resource Exhaustion

abhaybhargav

- Memory Allocation for Function
- Execution Duration of Function
- Payload Size
- Concurrency Limit
  - Per Account
  - Per Function

Example: Resource Exhaustion

abhaybhargav

- Re-DOS vulnerability in AWS Multipart Parser (identified by PureSec in March 2018)
- Payload can cause 100% CPU Utilization
  - Concurrency + Memory limits would be reached under attack scenario

```
module.exports.parse = (event, spotText) => {
    const boundary = getValueIgnoringKeyCase(event.headers, 'Content-
Type').split('=')[1];
    const body = (event.isBase64Encoded ? Buffer.from(event.body,
'base64').toString('binary') : event.body)
        .split(new RegExp(boundary))
        .filter(item => item.match(/Content-Disposition/))
```

ReDOS on the rise

abhaybhargav

- Lots of application eval <things> based on regex patterns:
  - URL params
  - Input Validation

we45

we45

Vulnerable Regex

abhaybhargav

```
echo '^([a-zA-Z0-9])(([\-.]|[_]+)?([a-zA-Z0-9]+))*(@){1}[a-z0-9]+[.]{1}(([a-z]{2,3})|([a-z]{2,3}[.]{1}
[a-z]{2,3}))$' > input.txt && ./scan.bin -i input.txt

= [1] =
INPUT: ^([a-zA-Z0-9])(([\-.]|[_]+)?([a-zA-Z0-9]+))*(@){1}[a-z0-9]+[.]{1}(([a-z]{2,3})|([a-z]{2,3}[.]{1}
[a-z]{2,3}))$
PARSE: OK
SIZE: 53
PUMPABLE: YES
VULNERABLE: YES {}
KLEENE: (([\-.]|[_]+)?([a-zA-Z0-9]+))*
PREFIX: 0
PUMPABLE: a0
SUFFIX:
TIME: 0.000359 (s)
>> TOTAL: 1
>> PARSED: 1
>> MAX NFA SIZE: 53
>> AVG NFA SIZE: 53
>> PUMPABLE: 1
>> VULNERABLE: 1
>> INTERRUPTED: 0
>> PRUNED: 0
>> TIME TOTAL: 0.000000 (s)
>> TIME MAX: 0.000000 (s)
```

we45

we45

Demo: Resource Exhaustion with Serverless + GraphQL

abhaybhargav

we45

we45

abhaybhargav

Thank You

we45