Security Automation & Orchestration (SAO)

Assessment, Audit, Certify and Accreditation workloads

Module – 4

How is auditing easier on AWS?

Auditors demand improvements:

- You're done with manual controls
- Sample testing is not representative
- AWS + Amazon Partner solutions = *Better Together* security capability
- AWS customer audits can be relied on

Goals – SAO Assessment, Audit, Certification

Through - ***Modernizing Technology Governance***:

*Adopting "**prevent**" controls, making "**detect**" controls more powerful and comprehensive*

AWS Shared Responsibility Model

Customers

Customer content

Platform, Applications, Identity & Access Management

Operating System, Network & Firewall Configuration

Client-side Data Encryption

Server-side Data Encryption

Network Traffic Protection

Customers are responsible for their security and compliance **IN** the Cloud

AWS Foundation Services

Compute

Storage

Database

Networking

AWS Global Infrastructure

Availability Zones

Regions

Edge Locations

AWS is responsible for the security **OF** the Cloud

AWS Shared Responsibility Model – Deep Dive

Will one model work for all services?

Infrastructure Services

Container Services

Abstract Services

aws

AWS Shared Responsibility Model:

for Infrastructure Services

Customer content

Platform & Applications Management

Operating System, Network & Firewall Configuration

Client-Side Data encryption & Data Integrity Authentication

Server-Side Encryption Fire System and/or Data

Network Traffic Protection Encryption / Integrity / Identity

Optional – Opaque data: 1's and 0's (in transit/at rest)

Customer IAM

Managed by

Customers

AWS Foundation Services

Compute | Storage | Database | Networking

AWS Global Infrastructure

Availability Zones

Regions

Edge Locations

AWS IAM

Managed by

amazon web services

aws

Infrastructure Service

Example – EC2

- Foundation Services — Networking, Compute, Storage
- AWS Global Infrastructure
- AWS IAM
- AWS API Endpoints

AWS

RESPONSIBILITIES

Customers

- Customer Data
- Customer Application
- Operating System
- Network & Firewall

- Customer IAM
- High Availability, Scaling
- Instance Management
- Data Protection (Transit, Rest, Backup)

AWS Shared Responsibility Model:

for Container Services

Customer content

Client-Side Data encryption
& Data Integrity Authentication

Network Traffic Protection
Encryption / Integrity / Identity

Optional – Opaque data: 1's and 0's (in transit/at rest)

Firewall Configuration

Customer IAM

Platform & Applications Management

Operating System, Network Configuration

AWS Foundation Services

- Compute
- Storage
- Database
- Networking

AWS Global Infrastructure

- Availability Zones
- Regions
- Edge Locations

AWS IAM

Managed by

Customers

Managed by

amazon web services

aws

Infrastructure Service

Example – RDS

AWS

- Foundational Services – Networking, Compute, Storage
- AWS Global Infrastructure
- AWS IAM
- AWS API Endpoints
- Operating System
- Platform / Application

RESPONSIBILITIES

Customers

- Customer Data
- Firewall (VPC)
- Customer IAM (DB Users, Table Permissions)
- High Availability
- Data Protection (Transit, Rest, Backup)
- Scaling

AWS Shared Responsibility Model:

for Abstract Services

Managed by

Customers

Customer content

(optional)

Opaque Data: 1's and 0's

(in flight / at rest)

Client-Side Data Encryption
& Data Integrity Authentication

Data Protection by the Platform
Protection of Data at Rest

Network Traffic Protection by the Platform
Protection of Data at in Transit

Platform & Applications Management

Operating System, Network & Firewall Configuration

AWS Foundation Services

- Compute
- Storage
- Database
- Networking

AWS Global Infrastructure

- Availability Zones
- Regions
- Edge Locations

AWS IAM

Managed by

amazon web services

aws

Infrastructure Service

Example – S3

AWS

- Foundational Services
- AWS Global Infrastructure
- AWS IAM
- AWS API Endpoints
- Operating System
- Platform / Application
- Data Protection (Rest - SSE, Transit)
- High Availability / Scaling

RESPONSIBILITIES

Customers

- Customer Data
- Data Protection (Rest – CSE)

Navigating Shared Responsibility

Achieving accreditation or certification on AWS is possible but how can we help?

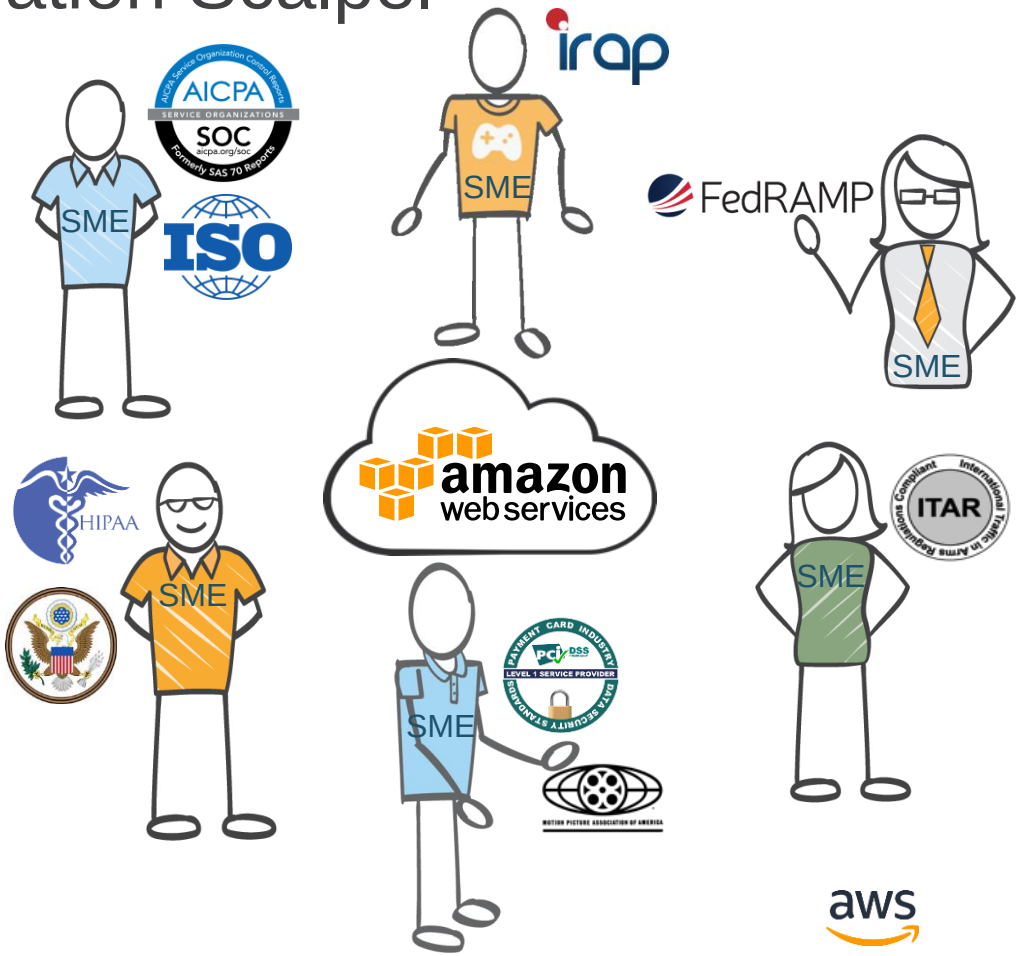
AWS Assurance Programs

Expert Audits: The Validation Scalpel

Experts auditors give a 360° view of the cloud.

Constantly engaged: the overall process never stops.

Continuous Risk Treatment



Meet your own security objectives

Customers

Your own Certification

Your own Accreditation

Your own external audits

Customer scope and effort is **reduced**

Better results through **focused efforts**

AWS Foundation Services

Compute | Storage | Database | Networking

AWS Global Infrastructure

Availability Zones

Regions

Edge Locations

Built on AWS **consistent** baseline controls

Governance as Code – review…

Is the process of managing and provisioning machine-readable definition files, templates, scripts and recipes for regulatory workload configurations:

- **Declarative (functional**) – Aspirational (e.g. **desired state**) target configuration against regulatory requirements.
- **Imperative (procedural)** – Defines code management (**desired conclusion**) and assertion to the regularity adherence.
- **Intelligent (environment aware)** – Specifies the configuration (**correct desired state**) based on relationships, dependencies and interaction in a regulated production environment.

Type Accreditation (Concepts & Practices)

A form of accreditation which is used to authorize multiple architectures for an application deployment or General Support System (GSS) for operation within an approved deployment recommendation with the same type of computing environment. (e.g. AWS region)

A type accreditation can satisfy certification (***Pre-Audit***) requirements only if the application or system consists of a common set of tested and approved environments, software, and underlying infrastructure.

Focus areas for Type Accreditations

- **Configuration Management of Operational Controls** - Common controls implementation, (e.g. Provisioning, Managing, Controlling, and Documenting) installations and changes for each deployment is critical to success.

    - It is especially important that configuration of the common security control tracking (e.g. Dashboard) which can control, constrain and guard-rail this process.
    - Deviations should be treat (real-time) based on risks associated with the modifications prior to authorization.

- **Communications** - Effective controls status ***MUST*** be maintained and security control responsibilities tracked in support of the master assessment plan and accreditation.

- **Environmenta**l -Specific Security Impact Analysis based "***As-Built"*** *and* correct desired state
    - **Specific deviations** - an impact analysis must be conducted to determine if any additional risk has been introduced to the overall system due to site

-

Assessment/Audit workpapers “AS-Built”

- Code files become Assessment workpapers and related materials, represent the evidence generated and/or gathered assessor and/or audit team

Filter events

Time (UTC +00:00)	Message
2016-11-15	


```
2016-11-15T23:43:15.458Z 45b929db-ab8d-11e6-ad33-678ba8b527ac Received S3 API Event:
{
    "version": "0",
    "id": "c08e1742-322a-4952-b216-0832d2a99ddc",
    "detail-type": "AWS API Call via CloudTrail",
    "source": "aws.s3",
    "account": "[redacted]",
    "time": "2016-11-15T23:43:11Z",
    "region": "us-east-1",
    "resources": [],
    "detail": {
        "eventVersion": "1.05",
        "userIdentity": {
            "type": "Root",
            "principalId": "[redacted]",
            "arn": "arn:aws:iam::[redacted]:root",
            "accountId": "[redacted]",
            "sessionContext": {
                "attributes": {
                    "creationDate": "2016-11-15T19:22:22Z",
                    "mfaAuthenticated": "false"
                }
            }
        },
        "eventTime": "2016-11-15T23:43:11Z",
```


powered by ace


```
1  {
2    "AWSTemplateFormatVersion" : "2010-09-09",
3    "Description" : "AWS CloudTrail API Activity Alarm Template for CloudWatch Logs",
4    "Parameters" : {
5        "LogGroupName" : {
6            "Type" : "String",
7            "Default" : "CloudTrail/DefaultLogGroup",
8            "Description" : "Enter CloudWatch Logs log group name. Default is CloudTrail/DefaultLogGroup"
9        },
10       "Email" : {
11           "Type" : "String",
12           "Description" : "Email address to notify when an API activity has triggered an alarm"
13       }
14   },
15   "Resources" : {
16       "SecurityGroupChangesMetricFilter": {
17           "Type": "AWS::Logs::MetricFilter",
18           "Properties": {
```

Analyze and Document – Output

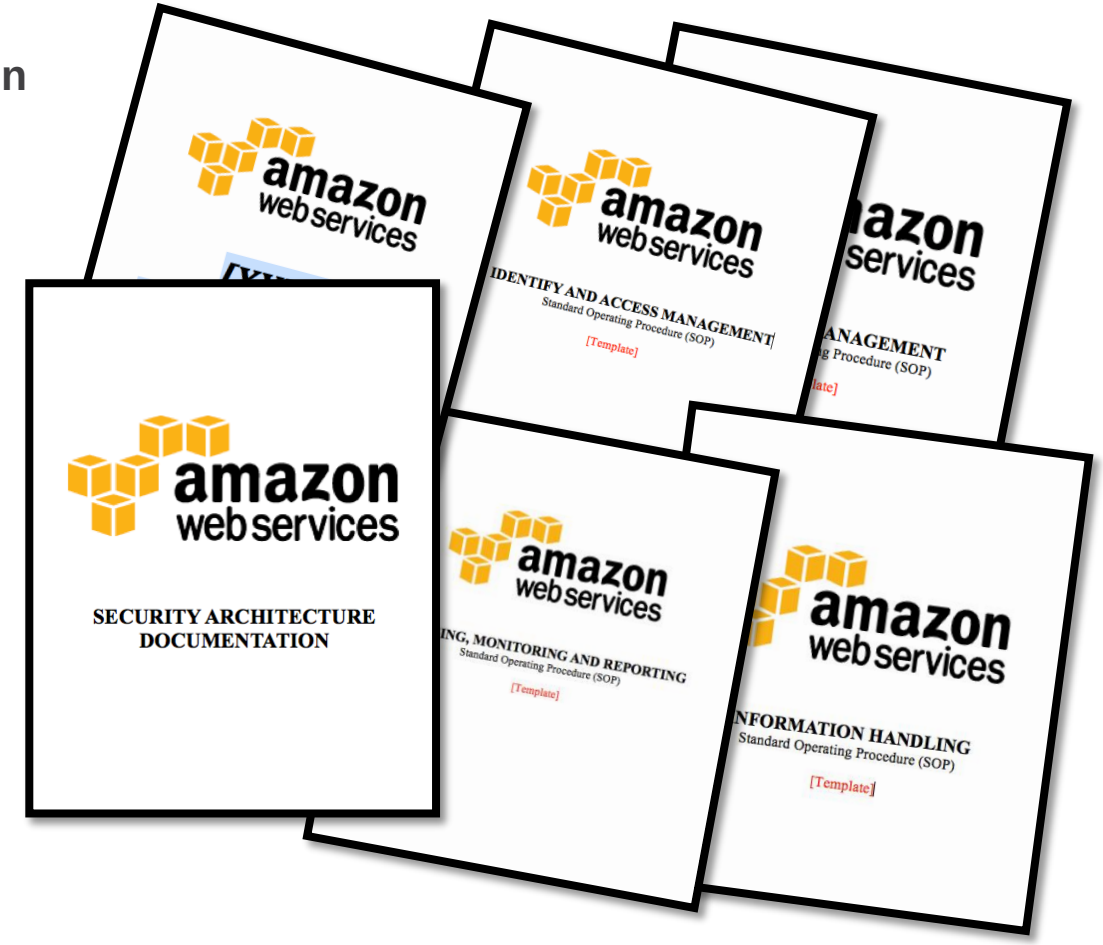
Security Architecture Documentation

Governance Policy

1. Security Policy Template

Standard Operating Procedures

1. Identify and Access Management
2. Information Handling
3. Logging, Monitoring and Reporting
4. Security Awareness & Training
5. Incident Management
6. Encryption
7. Asset & Information/Data Classification
8. Network Security
9. Fault Tolerance & Backup
10. Operational Security



CloudFormation – SecOps Use Case

Infrastructure **is** code -

- We already know how to manage “code” lifecycle
- Let Security as code perform state changes for governance

Separation of Duties - templates

- Merge code from SecOps and DevOps
- Baseline and build/state management

Provides inventory and documentation

- Resources are the same, just scaled

Integrate with constrained repository – Configuration Management

- Use constraints (example: tags)

Traditional Structured Deployment

```
{
  "Description" : "A short description th
  "Parameters": {
    // Variables that can be passed into
  },
  "Resources" : {
    // Where you declare which AWS re
  },
  "Outputs" : {
    // Values that can be returned after
  },
  "AWSTemplateFormatVersion" : "201
}
```

Create Skeleton

```
{
  "Description" : "Create an E
  "Parameters" : {
    "KeyPair" : {
      "Description" : "The EC2
      "Type" : "String"
    }
  },
  "Resources" : {
    "Ec2Instance" : {
      "Type" : "AWS::EC2::Instance",
      "Properties" : {
        "KeyName" : { "Ref" :
        "ImageId" : "ami-3b35
      }
    }
  },
  "Outputs" : {
    "InstanceId" : {
      "Descript
      "Valu
    }
  },
  "AWST
  2010-09-09"
}
```

Define Resources

```
"install_chef" : {
  "sources" : {
    "/var/chef/chef-repo" : "http://github.com/opscode/
                                                  ll/master"
  },
  "files" : {
    "/tmp/install.sh" : {
      "source" : "https://www.opscode.com/chef/install.sh",
      "mode"   : "000400",
      "owner" : "root",
      "group" : "root"
    },
```

Execute

```
  "commands" : {
    "01_make_chef_readable" : {
      "command" : "chmod +rx /var/chef"
    },
    "02_install_chef" : {
      "command" : "bash /tmp/install.sh",
      "cwd"   : "/var/chef"
    },
    "03_create_node_list" : {
      "command" : "chef-client -z -c /var/chef/chef-repo/.chef/client.rb",
      "cwd" : "/var/chef/chef-repo",
      "env" : { "HOME" : "/var/chef" }
    }
```

Split Ownership Configurations

Who knows your solution best?

- DevOps, Infrastructure, Security…?
- Delegate ownership and constrain compliance treatments

Split file into chunks or functions – modular

- Separate file sources with access control – Use IAM/VPC-E/etc.
- Push files -> **Validate** -> Merge files -> **Validate** -> Deploy -> **Validate**

GitHub, Puppet and Chef for deployment

- Promotion flows
  - Move from manual to Automation based on validation quality
- Excellent for merging jobs of split configurations

aws

Merging

From single file or multiple files

- Maintain access control using Governance policies
- Use different source stores if necessary…

Based on function/state

Reusable patterns

Maintain order, especially of validation

- Security validation last to execute
- Security should always win

```yaml
baseline: &baseline
  AWSTemplateFormatVersion: "2010-09-09"
  Description: "Service Catalog dependencies."
  Parameters: {}
  Mappings: {}
  Resources: &Resources
    ManagementSecurityGroup: &ManagementSecurityGroup
      Type: "AWS::EC2::SecurityGroup"
      Properties:
        GroupDescription: "Security group for management traffic"
        VpcId: "<VPC-ID>"
  Outputs: {}

SecDevOps-Production:
  <<: *baseline
  Resources:
    <<: *Resources
    ManagementGroupIngressRuleA:
      Type: "AWS::EC2::SecurityGroupIngress"
      Properties:
        GroupId:
          Ref: "ManagementSecurityGroup"
        IpProtocol: "tcp"
        FromPort: 22
        ToPort: 22
        CidrIp: "10.10.10.0/24"
    ManagementGroupIngressRuleB:
      Type: "AWS::EC2::SecurityGroupIngress"
      Properties:
        GroupId:
          Ref: "ManagementSecurityGroup"
        IpProtocol: "tcp"
        FromPort: 22
        ToPort: 22
        CidrIp: "10.10.20.0/24"

SecDevOps-Test:
  <<: *baseline
  Resources:
    <<: *Resources
    ManagementGroupIngressRule:
      Type: "AWS::EC2::SecurityGroupIngress"
      Properties:
        GroupId:
          Ref: "ManagementSecurityGroup"
        IpProtocol: "tcp"
        FromPort: 22
```

Validation

Keep track of what section you are validating

- Stage vs Prod
- Merged vs separated

Validate often and log/alert

- Validate part and end result
- Run-time validation

Use external agents

- AWS Simple Work Flow
- AWS Lambda
- Etc.

```
f = File.open("template.json", "rb")
g = f.read
g.gsub(",",",\n").each_line {|line|
    if (line.downcase.include? "cidrip") && !line.include?("0.0.0.0/0")
        cfnCorpCIDRArray.push line[/(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|2
    elsif (line.downcase.include? "cidrip") && line.include?("0.0.0.0/0")
        cfnCIDRArray.push line[/(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-
        p errorCIDRSource
        fail = true
    elsif (line.downcase.include?("fromport") && line.include?("0"))
        cfnPortArray.push line[/(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-
        p errorPortSource
        fail = true
    end
    }

# Check Corp CIDR blocks
strFail = ""
cfnCorpCIDRArray.each {|y|
    found = false
    corpCIDRArray.each {|x|
        cidr = NetAddr::CIDR.create(x)
        if ((cidr.contains?(y)) || (y == x))
            found = true
        end
        }
    if !found
        p errorCorpCIDRSource + " (#{y})"
        fail = true
    end
}
```

Structured deployment using Split ownershin

```
{
  "Description" : "A short description th
  "Parameters": {
    // Variables that can be passed into
  },
  "Resources" : {
    // Where you dedare which AWS re
  },
  "Outputs" : {
    // Values that can be returned after
  },
  "AWSTemplateFormatVersion" : "201
}
```

Create Skeleton - Infra team

```
{
  "Description" : "Create an E
  "Parameters" : {
    "KeyPair" : {
      "Description" : "The EC2
      "Type" : "String"
    }
  },
  "Resources" : {
    "Ec2Instance" : {
      "Type" : "AWS::EC2::Instance",
      "Properties" : {
        "KeyName" : { "Ref" :
        "ImageId" : "ami-3b35
      }
    }
  },
  "Outputs" : {
    "InstanceId" : {
      "Descript
      "Valu
    }
  },
  "AWST
}
```

2010-09-09"

Define Resources - DevOps

```
"install_chef" : {
  "sources" : {
    "/var/chef/chef-repo" : "http://github.com/opscode/
                                                  ll/master"
  },
  "files" : {
    "/tmp/install.sh" : {
      "source" : "https://www.opscode.com/chef/install.sh",
      "mode"   : "000400",
      "owner" : "root",
      "group" : "root"
    },
```

Execute - Security

```
"commands" : {
  "01_make_chef_readable" : {
    "command" : "chmod +rx /var/chef"
  },
  "02_install_chef" : {
    "command" : "bash /tmp/install.sh",
    "cwd"   : "/var/chef"
  },
  "03_create_node_list" : {
    "command" : "chef-client -z -c /var/chef/chef-repo/.chef/client.rb",
    "cwd" : "/var/chef/chef-repo",
    "env" : { "HOME" : "/var/chef" }
  }
```

aws

Config-Rule Example

approved_amis_by_tag

Description	Checks whether running instances are using specified AMIs. Specify the tags that identify the AMIs. Running instances with AMIs that don't have at least one of the specified tags are noncompliant.
Trigger type	Configuration changes
Scope of changes	Resources
Resource types	EC2 Instance
Config rule ARN	arn:aws:config:us-east-1:194518515516:config-rule/config-rule-oxa8ly
Parameters	amisByTagKeyAndValue: sec_approved:yes
Overall rule status	Last successful invocation on September 25, 2016 at 6:01:08 PM Last successful evaluation on September 25, 2016 at 6:01:08 PM

Resources evaluated

Click on the icon to view configuration details for the resource when it was last evaluated with this rule.

Resource type	Resource identifier	Compliance	Last successful invocation	Last successful evaluation	Config timeline
EC2 Instance	i-019b0e790a67dd7f1	Noncompliant	September 23, 2016 3:58:10 PM	September 23, 2016 3:58:11 PM	
EC2 Instance	i-0caf40fcb4f48517c	Compliant	September 25, 2016 5:48:26 PM	September 25, 2016 5:48:28 PM	

Questions?