AWS Security - Solution Architect MCQ Questions & Workshops

Table of Contents

1. MCQ Questions - IAM & Identity Management
2. MCQ Questions - AWS Organizations
3. MCQ Questions - KMS & Encryption
4. MCQ Questions - Security Services
5. Workshop 1: Multi-Account IAM Setup
6. Workshop 2: KMS Encryption Implementation
7. Workshop 3: Security Monitoring & Compliance
8. Workshop 4: Web Application Security

MCQ Questions - IAM & Identity Management {#mcq-iam}

Question 1: IAM Role for Microservices Architecture

Scenario: A healthcare company is migrating its patient management system to AWS. The application consists of multiple microservices running on ECS Fargate containers. Each microservice needs different AWS permissions:

- Patient Service: Read/Write to DynamoDB patient table
- Billing Service: Read from DynamoDB, Write to S3 billing bucket
- Analytics Service: Read-only access to all DynamoDB tables and S3 buckets

Question: What is the MOST secure and scalable way to implement this access pattern?

A) Create one IAM user with all permissions and share credentials across all containers
B) Create individual IAM users for each service and embed access keys in container environment variables
C) Create separate IAM roles with least-privilege policies for each service and assign them as ECS Task Roles
D) Create one IAM role with full DynamoDB and S3 access and share it across all services

Answer: C

Explanation: ECS Task Roles provide temporary credentials to containers without hardcoding access keys. Each service should have its own role with only the permissions it needs (principle of least privilege). This is more secure than sharing credentials and more manageable than embedding keys.

Question 2: Cross-Account Access Pattern

Scenario: Company X (Account A) needs to allow Company Y's data science team (Account B) to access specific S3 buckets containing shared datasets. The access should be auditable, temporary, and should not require sharing long-term credentials.

Question: Which solution provides the MOST secure cross-account access?

A) Create IAM users in Account A for each data scientist in Company Y
B) Use S3 bucket policies to allow public access with IP restrictions
C) Create an IAM role in Account A with S3 read permissions, establish a trust relationship with Account B, and allow Account B users to assume this role
D) Share the Account A root user credentials with Company Y's security team

Answer: C

Explanation: Cross-account IAM roles with trust policies are the recommended approach for secure cross-account access. Account B users can assume the role in Account A to get temporary credentials. This provides auditability through CloudTrail and doesn't require sharing long-term credentials.

Question 3: MFA Enforcement Strategy

Scenario: A financial services company wants to enforce MFA for all IAM users who access production resources. However, developers should be able to access development resources without MFA to speed up testing workflows.

Question: How can this requirement be implemented?

A) Enable MFA on the AWS account level, which automatically applies to all users
B) Create two IAM groups: one for production access with IAM policies that require MFA (using aws:MultiFactorAuthPresent condition), and one for dev access without MFA requirements
C) Use AWS Organizations SCPs to enforce MFA on all accounts
D) MFA cannot be selectively enforced; it must be all or nothing

Answer: B

Explanation: IAM policies can include conditions that check for MFA using `aws:MultiFactorAuthPresent`. By creating separate groups with different policies, you can enforce MFA only for production access while allowing developers to work freely in dev environments.

Question 4: IAM Permission Boundary Use Case

Scenario: A large enterprise wants to allow department managers to create IAM users and roles for their teams, but wants to ensure that managers cannot grant permissions beyond what they themselves have, preventing

privilege escalation.

Question: What is the BEST solution to implement this delegation with safety controls?

A) Give managers full IAM administrative permissions and trust them to follow policies
B) Create Service Control Policies (SCPs) that limit what managers can create
C) Use IAM Permission Boundaries to set maximum permissions that any user/role created by managers can have
D) Disable IAM user/role creation and require all access through SSO only

Answer: C

Explanation: Permission boundaries define the maximum permissions an entity can have, regardless of the policies attached to it. This allows safe delegation of IAM administration while preventing privilege escalation.

Question 5: IAM Identity Center (SSO) Implementation

Scenario: A global company with 5,000 employees uses Active Directory on-premises for authentication. They're expanding to AWS with 50 AWS accounts (dev, test, prod for different business units). Employees need single sign-on access to multiple AWS accounts based on their department and role.

Question: What is the MOST efficient solution?

A) Create IAM users in each AWS account for every employee
B) Implement AWS IAM Identity Center with Active Directory integration, define permission sets, and assign users to accounts based on groups
C) Use AWS Cognito User Pools for employee authentication
D) Set up VPN access and use on-premises credentials for AWS Console access

Answer: B

Explanation: IAM Identity Center (formerly AWS SSO) is designed exactly for this scenario. It integrates with existing Active Directory, provides SSO across multiple accounts, and uses permission sets for role-based access management at scale.

Question 6: IAM Policy Evaluation

Scenario: An IAM user is a member of a group that has an inline policy allowing `s3:PutObject` on bucket `project-data`. The user also has an attached policy denying `s3:*` on `project-data`. An SCP in AWS Organizations allows all S3 actions.

Question: Can the user upload objects to the `project-data` bucket?

A) Yes, because the group policy allows it
B) Yes, because the SCP allows it

C) No, because explicit deny always takes precedence
D) It depends on the order policies were attached

Answer: C

Explanation: In IAM policy evaluation, explicit denies always win. Even though the group policy and SCP allow the action, the explicit deny in the user's attached policy prevents the action.

MCQ Questions - AWS Organizations {#mcq-organizations}

Question 7: Service Control Policy Strategy

Scenario: A company has 30 AWS accounts organized into 3 OUs: Development, Staging, and Production. They want to ensure that:

- No account can disable CloudTrail
- Development can use only t2/t3 instance types
- Production cannot delete or modify S3 buckets

Question: How should these requirements be implemented?

A) Create IAM policies in each account
B) Create three different SCPs: one preventing CloudTrail disable (apply to root), one restricting instance types (apply to Dev OU), one protecting S3 (apply to Prod OU)
C) Use AWS Config rules to enforce these restrictions
D) SCPs cannot enforce these types of restrictions

Answer: B

Explanation: SCPs are the right tool for organization-wide guardrails. They act as permission boundaries for all accounts in an OU. Different SCPs can be applied to different OUs to enforce different controls.

Question 8: Multi-Account Strategy

Scenario: A startup is growing rapidly and needs to set up multiple AWS accounts for better isolation. They need:

- Consolidated billing
- Centralized security controls
- Ability to share some resources (like AMIs) across accounts
- Simplified account creation process

Question: Which combination of services addresses all requirements?

A) Just use one AWS account with IAM users
B) Use AWS Organizations for consolidated billing and account management, AWS Resource Access Manager (RAM) for resource sharing, and Control Tower for automated account setup
C) Create separate AWS accounts manually and manage them independently
D) Use AWS Organizations only without any additional services

Answer: B

Explanation: This requires multiple AWS services working together: Organizations for account management and billing, RAM for resource sharing, and Control Tower for automated, compliant account provisioning with guardrails.

Question 9: Tag Policy Enforcement

Scenario: A company wants to enforce cost allocation by requiring all resources to be tagged with `CostCenter`, `Environment`, and `Owner` tags. Resources without these tags should not be created.

Question: What is the proper implementation approach?

A) Use tag policies in AWS Organizations to enforce tagging at creation time
B) Use CloudWatch Events to trigger Lambda functions that add missing tags
C) Create IAM policies that deny resource creation without proper tags, and use tag policies for validation
D) Manually audit resources weekly and delete untagged ones

Answer: C

Explanation: While tag policies help validate and report on tags, they don't prevent resource creation. IAM policies with tag-based conditions (like `aws:RequestTag`) can enforce tags at creation time, combined with tag policies for organization-wide standards.

Question 10: Cross-Account Resource Sharing

Scenario: A central networking team manages a Transit Gateway that connects all VPCs across 20 AWS accounts. Individual application teams need to attach their VPCs to this Transit Gateway without having access to the networking account.

Question: What is the BEST solution?

A) Share the networking account credentials with all application teams
B) Use AWS Resource Access Manager (RAM) to share the Transit Gateway with other accounts
C) Create VPC peering connections between all accounts
D) Recreate the Transit Gateway in each account

Answer: B

Explanation: AWS RAM is specifically designed for sharing resources like Transit Gateways, subnets, and Route 53 Resolver rules across accounts within an organization, without sharing credentials or duplicating infrastructure.

MCQ Questions - KMS & Encryption {#mcq-kms}

Question 11: Multi-Region Encryption Strategy

Scenario: A global e-commerce company stores customer data in DynamoDB Global Tables spanning `us-east-1`, `eu-west-1`, and `ap-southeast-1`. They need encryption with customer-managed keys and want to minimize decryption latency while maintaining the ability to independently revoke keys in each region if needed.

Question: What KMS key strategy should they use?

A) Create one CMK in us-east-1 and use it for all regions
B) Create separate CMKs in each region with region-specific key policies
C) Create a multi-region CMK with replicas in each region
D) Use AWS-managed keys (aws/dynamodb) as they automatically work across regions

Answer: B (or C depending on priority)

Explanation: Both B and C are valid, but for different priorities:

- **Option B** provides complete independence - keys can be revoked per region without affecting others, but requires managing separate keys
- **Option C** (multi-region keys) provides operational simplicity and is better if synchronized revocation is acceptable

For this scenario emphasizing "independently revoke keys," **B is technically better**, though C is AWS's newer recommended approach for most multi-region scenarios.

Question 12: S3 Encryption Compliance

Scenario: A healthcare company must ensure that all objects uploaded to their S3 bucket are encrypted using a specific CMK for HIPAA compliance. They want to prevent any unencrypted uploads and need detailed audit logs of all key usage.

Question: How should this be implemented?

A) Enable default bucket encryption with SSE-S3
B) Create a bucket policy that denies PutObject requests unless encrypted with SSE-KMS using the specific

CMK, enable CloudTrail for KMS events
C) Use SSE-C and distribute keys to all users
D) Enable S3 Object Lock to prevent unencrypted uploads

Answer: B

Explanation: To enforce a specific CMK, use a bucket policy with a condition that requires SSE-KMS with that key ARN. CloudTrail provides comprehensive audit logs of all KMS operations (encrypt, decrypt, key usage) needed for compliance.

Question 13: EBS Volume Encryption Migration

Scenario: A company has 100 unencrypted EBS volumes attached to production EC2 instances. They need to encrypt all volumes with minimal downtime while maintaining existing EBS volume IDs and no data loss.

Question: What is the correct approach?

A) Enable encryption on existing volumes in-place using AWS CLI modify-volume command
B) Create encrypted snapshots from unencrypted volumes, create new encrypted volumes from encrypted snapshots, then swap volumes with minimal downtime
C) EBS volumes cannot be encrypted after creation; recreate all EC2 instances
D) Use AWS Systems Manager to automatically encrypt volumes without snapshots

Answer: B

Explanation: EBS volumes cannot be encrypted in-place. The proper approach is: create snapshot → copy snapshot with encryption → create new encrypted volume from encrypted snapshot → detach old volume, attach new volume. This can be orchestrated with minimal downtime.

Question 14: KMS Key Policy Design

Scenario: A company has three teams: Developers, Operations, and Security. For a CMK protecting production database credentials:

- Developers need to encrypt new credentials
- Operations needs to decrypt credentials for deployments
- Security team needs full administrative control
- External auditors need to view key policies but not use the key

Question: How should the key policy be structured?

A) Grant all permissions to everyone and rely on IAM policies for restrictions
B) Create a key policy with different statement blocks: Security team as key administrators, Developers with

kms:Encrypt, Operations with kms:Decrypt, and Auditors with kms:DescribeKey and kms:GetKeyPolicy
C) Use only IAM policies without a key policy
D) Key policies cannot support this granular level of control

Answer: B

Explanation: KMS key policies support fine-grained permissions. Separate policy statements can grant different principals different levels of access (administrative, usage, view-only). This is more secure than relying solely on IAM policies.

Question 15: Automatic Key Rotation

Scenario: A financial services company uses customer-managed CMKs for encrypting sensitive financial records. Compliance requires encryption keys to be rotated annually, but existing encrypted data must remain decryptable without re-encryption.

Question: What is the correct implementation?

A) Enable automatic key rotation on the CMK; AWS handles everything automatically
B) Create a new CMK annually and re-encrypt all data
C) Automatic rotation is not possible with CMKs; only AWS-managed keys support it
D) Use AWS Certificate Manager for automatic rotation

Answer: A

Explanation: When you enable automatic key rotation for CMKs, AWS automatically rotates the key material annually while maintaining old versions for decrypting existing data. No re-encryption is needed, and the key ID remains the same.

Question 16: Cross-Account KMS Access

Scenario: Account A has an encrypted S3 bucket using a CMK. Account B needs to copy objects from this bucket to their own bucket and decrypt them.

Question: What must be configured for this to work?

A) Nothing; cross-account KMS access works by default
B) The CMK key policy in Account A must grant Account B permission to use the key (kms:Decrypt), and Account B must have IAM permissions to access S3 and KMS
C) Share the CMK private key with Account B
D) Copy the CMK to Account B

Answer: B

Explanation: Cross-account KMS access requires both resource-based permissions (key policy in Account A granting access to Account B) and identity-based permissions (IAM policy in Account B allowing users to call KMS operations). This is the dual authorization model.

MCQ Questions - Security Services {#mcq-security}

Question 17: GuardDuty Findings Response

Scenario: Amazon GuardDuty has detected unusual API calls from one of your IAM users' access keys originating from a suspicious IP address in a different country than where your company operates. The severity is marked as HIGH.

Question: What is the MOST appropriate immediate response sequence?

A) Ignore it; GuardDuty often has false positives
B) Immediately disable the IAM user's access keys, rotate credentials, review CloudTrail logs to identify unauthorized actions, investigate the scope of compromise, and implement additional security controls
C) Just change the user's password
D) Block the IP address at the WAF level

Answer: B

Explanation: High-severity GuardDuty findings require immediate action. The proper incident response includes: contain (disable keys), investigate (CloudTrail logs), assess damage, remediate, and prevent recurrence. Simply blocking IPs or changing passwords isn't sufficient.

Question 18: Inspector vs GuardDuty vs Macie

Scenario: A company needs to implement the following security requirements:

1. Detect if EC2 instances have vulnerabilities or missing patches
2. Identify unusual network traffic patterns and potential threats
3. Discover S3 buckets containing credit card numbers

Question: Which combination of services addresses all requirements?

A) GuardDuty for all three requirements
B) Amazon Inspector for vulnerabilities, GuardDuty for threat detection, Macie for sensitive data discovery
C) Amazon Macie for all three requirements
D) AWS Config for all three requirements

Answer: B

Explanation:

- **Inspector**: Vulnerability scanning (CVEs, missing patches, network accessibility)
- **GuardDuty**: Threat detection (unusual behavior, compromised instances, malicious IPs)
- **Macie**: Sensitive data discovery (PII, PHI, credit cards in S3)

Each service has a specific security focus.

Question 19: WAF Rule Implementation

Scenario: A web application behind CloudFront is experiencing:

- SQL injection attempts in URL query strings
- Bot traffic from a specific country accounting for 90% of requests
- Legitimate users globally, including from the bot-traffic country

Question: What WAF configuration best addresses this without blocking legitimate users?

A) Block all traffic from the bot-traffic country using geo-restriction
B) Implement AWS Managed Rules for SQL injection, create a rate-based rule to limit requests per IP (e.g., 2000 requests per 5 minutes), and use CAPTCHA for suspicious patterns rather than outright geo-blocking
C) Enable Shield Advanced which automatically handles all threats
D) Use IP reputation lists to block all known bad IPs

Answer: B

Explanation: Geo-blocking would affect legitimate users. The better approach combines: SQL injection protection (managed rules), rate limiting (stops bot floods but allows normal users), and CAPTCHA (challenges suspicious traffic without outright blocking). This provides defense in depth without false positives.

Question 20: Secrets Manager vs Parameter Store

Scenario: An application uses:

- Database passwords that must rotate automatically every 30 days
- API keys from third-party services (no rotation needed)
- Configuration values like environment names and feature flags
- SSL certificate private keys

Question: What is the MOST cost-effective and appropriate storage strategy?

A) Store everything in Secrets Manager
B) Store database passwords in Secrets Manager (for automatic rotation), store API keys and SSL certificates in Secrets Manager (for encryption at rest), store configuration values in Parameter Store (Standard tier, free)
C) Store everything in Parameter Store
D) Store secrets in S3 with encryption

Answer: B

Explanation: Secrets Manager is ideal for secrets requiring rotation (databases, some API keys) and high-value secrets (SSL certs). Parameter Store Standard tier is free and perfect for configuration values. Using both optimizes cost while maintaining appropriate security.

Question 21: Shield Advanced vs Standard

Scenario: An online gaming platform experiences frequent DDoS attacks during peak gaming hours. Attacks range from 50 Gbps to 200 Gbps and cause application unavailability. The company has lost revenue due to downtime.

Question: Should they upgrade from Shield Standard to Shield Advanced?

A) No, Shield Standard handles all DDoS attacks automatically
B) Yes, Shield Advanced provides DDoS Response Team (DRT) support, advanced attack analytics, cost protection from scaling, and detection of application-layer attacks - critical for a business losing revenue from attacks
C) No, just use WAF rate-based rules
D) Yes, but only because Shield Standard doesn't provide any DDoS protection

Answer: B

Explanation: Shield Standard provides automatic infrastructure (Layer 3/4) DDoS protection, but Shield Advanced adds: 24/7 DDoS Response Team, advanced attack visibility, cost protection (credits for scaling during attacks), and application layer (Layer 7) protection. For revenue-impacting attacks, Shield Advanced is justified.

Question 22: API Gateway Security Implementation

Scenario: A fintech company is building a RESTful API using API Gateway for a mobile banking app. Requirements:

- Only authenticated users can access APIs
- Rate limiting per user (100 requests per minute)
- Protection against SQL injection in API parameters
- Compliance logging of all API access

Question: What combination of security controls should be implemented?

A) Use Amazon Cognito User Pools for authentication, API Gateway usage plans for rate limiting, WAF with SQL injection rules attached to API Gateway, and enable CloudWatch Logs and CloudTrail
B) Use IAM authentication only
C) Rely on application-level security only
D) Use Lambda authorizers for everything including rate limiting

Answer: A

Explanation: This requires multiple layers:

- **Cognito User Pools**: OAuth2/OIDC authentication for mobile users
- **Usage Plans**: Built-in API Gateway rate limiting per API key
- **WAF**: Protection against injection attacks
- **CloudWatch/CloudTrail**: Comprehensive logging for compliance

---

Question 23: Certificate Manager Use Case

Scenario: A company hosts a web application on EC2 instances behind an Application Load Balancer. They need SSL/TLS certificates for `app.example.com` and `*.example.com`. Certificates must renew automatically, and the company wants to minimize operational overhead.

Question: What is the BEST solution?

A) Purchase certificates from a third-party CA and manually upload/renew them
B) Use AWS Certificate Manager to provision free public certificates for ALB, enable automatic renewal, and use Amazon Route 53 for DNS validation
C) Generate self-signed certificates
D) Use AWS KMS to create certificates

Answer: B

Explanation: ACM provides free public SSL/TLS certificates, automatic renewal (every 13 months), and seamless integration with ALB, CloudFront, and API Gateway. DNS validation with Route 53 makes issuance quick and automated. This is zero operational overhead.

Question 24: GuardDuty + Security Hub Integration

Scenario: A company has 50 AWS accounts. They want centralized security monitoring, automated remediation for some findings, and compliance reports across all accounts.

Question: What architecture should they implement?

A) Enable GuardDuty in each account independently
B) Enable GuardDuty in all accounts and aggregate findings to a central administrator account, enable Security Hub to collect findings from GuardDuty, Inspector, and Macie, create EventBridge rules to trigger Lambda for automated remediation, and generate compliance reports from Security Hub
C) Use only CloudWatch Logs for centralized logging
D) GuardDuty doesn't support multi-account monitoring

Answer: B

Explanation: This is the AWS security monitoring best practice architecture:

- **GuardDuty**: Multi-account deployment with aggregation
- **Security Hub**: Central finding aggregation from multiple services
- **EventBridge + Lambda**: Event-driven automated remediation
- **Security Hub compliance reports**: CIS, PCI-DSS, etc.

Question 25: Firewall Manager Use Case

Scenario: A large enterprise has 100 AWS accounts across multiple OUs. They need to:

- Apply the same WAF rules to all CloudFront distributions
- Ensure all VPCs have consistent Security Group rules
- Enforce that all accounts use Shield Advanced
- Audit compliance continuously

Question: What is the MOST efficient solution?

A) Manually configure each account
B) Use AWS Firewall Manager to create organization-wide security policies for WAF, Security Groups, and Shield, with automatic application to new accounts and resources
C) Use CloudFormation StackSets
D) Create Lambda functions to enforce policies

Answer: B

Explanation: Firewall Manager is specifically designed for centralized security policy management across AWS Organizations. It automatically applies and audits WAF rules, Security Groups, Shield, and Network Firewall policies across all accounts and resources.

Workshop 1: Multi-Account IAM Setup {#workshop-1}

Objective

Set up a multi-account AWS environment with proper IAM controls, cross-account access, and MFA enforcement.

Prerequisites

- 3 AWS accounts (Management, Production, Development)
- AWS CLI configured
- Administrative access to all accounts

Duration: 2 hours

Architecture Overview

```
Management Account (Organizations)
├── Production Account (OU: Production)
│   ├── IAM Roles (Admin, ReadOnly, DataScientist)
│   └── S3 Buckets (prod-data)
└── Development Account (OU: Development)
    ├── IAM Roles (Developer, Tester)
    └── EC2 Instances (dev environment)
```

Tasks

Task 1: Set Up AWS Organizations (30 minutes)

1. **In Management Account:**

```
bash
```

```
# Create organization
aws organizations create-organization --feature-set ALL

# Create Production OU
aws organizations create-organizational-unit \
  --parent-id r-xxxx \
  --name Production

# Create Development OU
aws organizations create-organizational-unit \
  --parent-id r-xxxx \
  --name Development
```

2. **Create SCPs:**
    - Create SCP to prevent CloudTrail deletion
    - Create SCP to restrict Development to specific regions
    - Attach SCPs to appropriate OUs

   Example SCP (Prevent CloudTrail Deletion):

```
json

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "cloudtrail:StopLogging",
        "cloudtrail:DeleteTrail"
      ],
      "Resource": "*"
    }
  ]
}
```

Task 2: Configure IAM Identity Center (45 minutes)

1. **Enable IAM Identity Center** in Management Account
2. **Configure Active Directory Integration** (or use IAM Identity Center directory)
3. **Create Permission Sets:**
    - Administrator Access

- Read-Only Access
- Developer Access (EC2, Lambda, S3)
- Data Analyst Access (S3, Athena, Glue)

4. **Create Groups and Assign Access:**

```
Group: ProductionAdmins
├── Permission Set: Administrator Access
└── AWS Account: Production Account

Group: Developers
├── Permission Set: Developer Access
└── AWS Account: Development Account

Group: DataScientists
├── Permission Set: Data Analyst Access
└── AWS Accounts: Production, Development
```

Task 3: Implement Cross-Account S3 Access (30 minutes)

1. **In Production Account:**
   - Create S3 bucket `prod-data-shared`
   - Create IAM role `CrossAccountDataAccess`

   Trust Policy:

```
json
```

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Principal": {
    "AWS": "arn:aws:iam::DEVELOPMENT-ACCOUNT-ID:root"
   },
   "Action": "sts:AssumeRole",
   "Condition": {
    "StringEquals": {
     "sts:ExternalId": "unique-external-id-12345"
    }
   }
  }
 ]
}
```

2. **Configure S3 Bucket Policy:**

json

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Principal": {
    "AWS": "arn:aws:iam::PRODUCTION-ACCOUNT-ID:role/CrossAccountDataAccess"
   },
   "Action": [
    "s3:GetObject",
    "s3:ListBucket"
   ],
   "Resource": [
    "arn:aws:s3:::prod-data-shared",
    "arn:aws:s3:::prod-data-shared/*"
   ]
  }
 ]
}
```

3. **In Development Account:**

   - Create IAM role for developers to assume the cross-account role

- Test access using AWS CLI

Task 4: Enforce MFA (15 minutes)

1. **Create IAM Policy requiring MFA:**

```json
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "AllowAllActionsWithMFA",
   "Effect": "Allow",
   "Action": "*",
   "Resource": "*",
   "Condition": {
    "BoolIfExists": {
     "aws:MultiFactorAuthPresent": "true"
    }
   }
  },
  {
   "Sid": "DenyAllExceptMFAManagementWithoutMFA",
   "Effect": "Deny",
   "NotAction": [
    "iam:CreateVirtualMFADevice",
    "iam:EnableMFADevice",
    "iam:GetUser",
    "iam:ListMFADevices",
    "iam:ListVirtualMFADevices",
    "iam:ResyncMFADevice",
    "sts:GetSessionToken"
   ],
   "Resource": "*",
   "Condition": {
    "BoolIfExists": {
     "aws:MultiFactorAuthPresent": "false"
    }
   }
  }
 ]
}
```

2. **Apply policy to production access groups**

Validation

- [ ] Users can access assigned accounts via SSO portal
- [ ] Cross-account S3 access works from Development to Production
- [ ] CloudTrail cannot be deleted in any account
- [ ] MFA is enforced for production access
- [ ] Development account can only create resources in allowed regions

Discussion Points

1. Why use IAM Identity Center instead of IAM users?
2. What is the principle of least privilege and how did we apply it?
3. How do SCPs differ from IAM policies?
4. What are the security benefits of using external IDs in trust policies?

Workshop 2: KMS Encryption Implementation {#workshop-2}

Objective

Implement comprehensive encryption for data at rest and in transit using AWS KMS across S3, EBS, and RDS.

Prerequisites

- AWS account with admin access
- AWS CLI and SDKs configured
- Basic understanding of encryption concepts

Duration: 2.5 hours

Architecture Overview

```
Application Stack (Encrypted)
├── S3 Bucket (SSE-KMS with CMK)
├── RDS MySQL (Encrypted with CMK)
├── EBS Volumes (Encrypted with CMK)
└── Lambda Functions (Environment variables encrypted)
```

Tasks

Task 1: Create and Configure Customer Managed Keys (30 minutes)

1. **Create CMKs for different purposes:**

```bash
# Create CMK for S3
aws kms create-key \
  --description "CMK for S3 bucket encryption" \
  --key-policy file://s3-key-policy.json

# Create CMK for RDS
aws kms create-key \
  --description "CMK for RDS encryption" \
  --key-policy file://rds-key-policy.json

# Create alias
aws kms create-alias \
  --alias-name alias/s3-encryption \
  --target-key-id <key-id>
```

2. **Key Policy Example (s3-key-policy.json):**

```json
```

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "Enable IAM User Permissions",
   "Effect": "Allow",
   "Principal": {
    "AWS": "arn:aws:iam::ACCOUNT-ID:root"
   },
   "Action": "kms:*",
   "Resource": "*"
  },
  {
   "Sid": "Allow application role to encrypt/decrypt",
   "Effect": "Allow",
   "Principal": {
    "AWS": "arn:aws:iam::ACCOUNT-ID:role/ApplicationRole"
   },
   "Action": [
    "kms:Decrypt",
    "kms:Encrypt",
    "kms:GenerateDataKey"
   ],
   "Resource": "*"
  },
  {
   "Sid": "Allow S3 to use the key",
   "Effect": "Allow",
   "Principal": {
    "Service": "s3.amazonaws.com"
   },
   "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
   ],
   "Resource": "*"
  }
 ]
}
```

3. **Enable Automatic Key Rotation:**

```
bash

  aws kms enable-key-rotation --key-id <key-id>
```

Task 2: Implement S3 Encryption (30 minutes)

1. **Create S3 Bucket with Default Encryption:**

bash

```
# Create bucket
aws s3api create-bucket \
  --bucket encrypted-data-bucket-$(date +%s) \
  --region us-east-1

# Enable default encryption
aws s3api put-bucket-encryption \
  --bucket encrypted-data-bucket \
  --server-side-encryption-configuration '{
    "Rules": [{
      "ApplyServerSideEncryptionByDefault": {
        "SSEAlgorithm": "aws:kms",
        "KMSMasterKeyID": "alias/s3-encryption"
      },
      "BucketKeyEnabled": true
    }]
  }'
```

2. **Create Bucket Policy to Enforce Encryption:**

json

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "DenyUnencryptedObjectUploads",
   "Effect": "Deny",
   "Principal": "*",
   "Action": "s3:PutObject",
   "Resource": "arn:aws:s3:::encrypted-data-bucket/*",
   "Condition": {
    "StringNotEquals": {
     "s3:x-amz-server-side-encryption": "aws:kms"
    }
   }
  },
  {
   "Sid": "DenyWrongKMSKey",
   "Effect": "Deny",
   "Principal": "*",
   "Action": "s3:PutObject",
   "Resource": "arn:aws:s3:::encrypted-data-bucket/*",
   "Condition": {
    "StringNotEquals": {
     "s3:x-amz-server-side-encryption-aws-kms-key-id": "arn:aws:kms:us-east-1:ACCOUNT-ID:key/KEY-ID"
    }
   }
  }
 ]
}
```

3. **Test Upload with Python SDK:**

```
python
```

```python
import boto3

s3 = boto3.client('s3')
kms = boto3.client('kms')

# Upload with SSE-KMS
s3.put_object(
    Bucket='encrypted-data-bucket',
    Key='test-file.txt',
    Body=b'Sensitive data',
    ServerSideEncryption='aws:kms',
    SSEKMSKeyId='alias/s3-encryption'
)

# Verify encryption
response = s3.head_object(
    Bucket='encrypted-data-bucket',
    Key='test-file.txt'
)
print(f"Encryption: {response['ServerSideEncryption']}")
print(f"KMS Key: {response.get('SSEKMSKeyId', 'N/A')}")
```

Task 3: Encrypt RDS Database (40 minutes)

1. **Create Encrypted RDS Instance:**

```bash
aws rds create-db-instance \
  --db-instance-identifier encrypted-mysql-db \
  --db-instance-class db.t3.micro \
  --engine mysql \
  --master-username admin \
  --master-user-password MySecurePass123! \
  --allocated-storage 20 \
  --storage-encrypted \
  --kms-key-id alias/rds-encryption \
  --backup-retention-period 7
```

2. **Migrate Unencrypted to Encrypted (if existing):**

```bash
```

```
# Step 1: Create snapshot
aws rds create-db-snapshot \
  --db-instance-identifier unencrypted-db \
  --db-snapshot-identifier unencrypted-snapshot

# Step 2: Copy snapshot with encryption
aws rds copy-db-snapshot \
  --source-db-snapshot-identifier unencrypted-snapshot \
  --target-db-snapshot-identifier encrypted-snapshot \
  --kms-key-id alias/rds-encryption \
  --copy-tags

# Step 3: Restore from encrypted snapshot
aws rds restore-db-instance-from-db-snapshot \
  --db-instance-identifier encrypted-mysql-db \
  --db-snapshot-identifier encrypted-snapshot
```

3. **Enable IAM Database Authentication:**

bash

```
aws rds modify-db-instance \
  --db-instance-identifier encrypted-mysql-db \
  --enable-iam-database-authentication \
  --apply-immediately
```

4. **Connect Using IAM Authentication (Python):**

python

```
import boto3
import pymysql

rds = boto3.client('rds')

# Generate authentication token
token = rds.generate_db_auth_token(
    DBHostname='encrypted-mysql-db.xxxxx.us-east-1.rds.amazonaws.com',
    Port=3306,
    DBUsername='iam_user',
    Region='us-east-1'
)

# Connect to database
connection = pymysql.connect(
    host='encrypted-mysql-db.xxxxx.us-east-1.rds.amazonaws.com',
    user='iam_user',
    password=token,
    database='mydb',
    ssl_ca='/path/to/rds-ca-cert.pem'
)
```

Task 4: Encrypt EBS Volumes (30 minutes)

1. **Create Encrypted EBS Volume:**

```
bash

aws ec2 create-volume \
  --size 10 \
  --availability-zone us-east-1a \
  --volume-type gp3 \
  --encrypted \
  --kms-key-id alias/ebs-encryption
```

2. **Enable EBS Encryption by Default:**

```
bash

aws ec2 enable-ebs-encryption-by-default --region us-east-1

aws ec2 modify-ebs-default-kms-key-id \
  --kms-key-id alias/ebs-encryption \
  --region us-east-1
```

3. **Encrypt Existing Unencrypted Volume:**

```bash
# Step 1: Create snapshot
aws ec2 create-snapshot \
  --volume-id vol-xxxxxxxxx \
  --description "Snapshot for encryption"

# Step 2: Copy snapshot with encryption
aws ec2 copy-snapshot \
  --source-region us-east-1 \
  --source-snapshot-id snap-xxxxxxxxx \
  --destination-region us-east-1 \
  --encrypted \
  --kms-key-id alias/ebs-encryption

# Step 3: Create volume from encrypted snapshot
aws ec2 create-volume \
  --snapshot-id snap-yyyyyyyyy \
  --availability-zone us-east-1a
```

Task 5: Monitor and Audit KMS Usage (20 minutes)

1. **Enable CloudTrail for KMS Events:**

```bash
aws cloudtrail put-event-selectors \
  --trail-name my-trail \
  --event-selectors '[{
    "ReadWriteType": "All",
    "IncludeManagementEvents": true,
    "DataResources": [{
      "Type": "AWS::KMS::Key",
      "Values": ["arn:aws:kms:us-east-1:ACCOUNT-ID:key/*"]
    }]
  }]'
```

2. **Create CloudWatch Alarm for Key Usage:**

```bash
```

```
aws cloudwatch put-metric-alarm \
  --alarm-name kms-decrypt-errors \
  --alarm-description "Alert on KMS decrypt errors" \
  --metric-name UserErrorCount \
  --namespace AWS/KMS \
  --statistic Sum \
  --period 300 \
  --threshold 10 \
  --comparison-operator GreaterThanThreshold \
  --evaluation-periods 1
```

3. **Query KMS Usage with CloudWatch Insights:**

```
fields @timestamp, userIdentity.principalId, requestParameters.keyId, eventName
| filter eventSource = "kms.amazonaws.com"
| filter eventName in ["Decrypt", "Encrypt", "GenerateDataKey"]
| stats count() by eventName, requestParameters.keyId
```

Validation Checklist

- [ ] All CMKs created with proper key policies
- [ ] Automatic key rotation enabled
- [ ] S3 bucket enforces KMS encryption
- [ ] RDS instance encrypted with CMK
- [ ] EBS encryption by default enabled
- [ ] CloudTrail logging KMS events
- [ ] Test decryption with proper IAM permissions
- [ ] Test denial of access without proper permissions

Discussion Points

1. What's the difference between AWS-managed keys and Customer-managed keys?
2. When should you use S3 bucket keys for cost optimization?
3. How does envelope encryption work with data keys?
4. What happens to old data when you rotate a CMK?
5. How do you share encrypted snapshots across accounts?

Workshop 3: Security Monitoring & Compliance {#workshop-3}

Objective

Implement comprehensive security monitoring using GuardDuty, Inspector, Macie, and Security Hub with automated remediation.

Prerequisites

- AWS account with admin access
- Running EC2 instances and S3 buckets
- CloudFormation or Terraform knowledge (optional)

Duration: 3 hours

Architecture Overview

```
Security Monitoring Stack
├── GuardDuty (Threat Detection)
├── Amazon Inspector (Vulnerability Scanning)
├── Amazon Macie (Data Discovery)
├── AWS Security Hub (Centralized Findings)
├── EventBridge (Event Routing)
└── Lambda (Automated Remediation)
```

Tasks

Task 1: Enable and Configure GuardDuty (30 minutes)

1. **Enable GuardDuty:**

```
bash

  aws guardduty create-detector \
    --enable \
    --finding-publishing-frequency FIFTEEN_MINUTES
```

2. **Configure Trusted IP Lists:**

```
bash
```

```
# Create trusted IP list (company IPs)
cat > trusted-ips.txt << EOF
203.0.113.0/24
198.51.100.0/24
EOF

# Upload to S3
aws s3 cp trusted-ips.txt s3://security-lists-bucket/

# Add to GuardDuty
aws guardduty create-ip-set \
  --detector-id <detector-id> \
  --name CompanyTrustedIPs \
  --format TXT \
  --location s3://security-lists-bucket/trusted-ips.txt \
  --activate
```

3. **Create Suppression Rules:**

bash

```
# Suppress findings for known internal security scanners
aws guardduty create-filter \
  --detector-id <detector-id> \
  --name SuppressInternalScans \
  --finding-criteria '{
    "Criterion": {
      "service.action.networkConnectionAction.remoteIpDetails.ipAddressV4": {
        "Eq": ["10.0.1.100"]
      },
      "type": {
        "Eq": ["Recon:EC2/PortProbeUnprotectedPort"]
      }
    }
  }' \
  --action ARCHIVE
```

4. **Generate Sample Findings for Testing:**

bash

```
aws guardduty create-sample-findings \
  --detector-id <detector-id> \
  --finding-types \
    "Backdoor:EC2/C&CActivity.B!DNS" \
    "CryptoCurrency:EC2/BitcoinTool.B!DNS" \
    "UnauthorizedAccess:IAMUser/MaliciousIPCaller.Custom"
```

Task 2: Configure Amazon Inspector (40 minutes)

1. **Enable Inspector:**

```
bash

aws inspector2 enable \
  --resource-types EC2 ECR LAMBDA
```

2. **Tag EC2 Instances for Scanning:**

```
bash

aws ec2 create-tags \
  --resources i-xxxxxxxxx \
  --tags Key=InspectorScan,Value=true \
         Key=Environment,Value=production
```

3. **Create Custom Assessment Target (Legacy Inspector):**

```
bash

# If using Inspector Classic
aws inspector create-assessment-target \
  --assessment-target-name ProductionServers \
  --resource-group-arn arn:aws:inspector:us-east-1:ACCOUNT-ID:resourcegroup/xxxxx

# Create assessment template
aws inspector create-assessment-template \
  --assessment-target-arn <target-arn> \
  --assessment-template-name WeeklySecurityScan \
  --duration-in-seconds 3600 \
  --rules-package-arns \
    arn:aws:inspector:us-east-1:316112463485:rulespackage/0-R01qwB5Q \
    arn:aws:inspector:us-east-1:316112463485:rulespackage/0-gEjTy7T7
```

4. **Review Inspector Findings:**

```python
import boto3

inspector = boto3.client('inspector2')

# List findings
response = inspector.list_findings(
    filterCriteria={
        'severity': [{'comparison': 'EQUALS', 'value': 'HIGH'}],
        'resourceType': [{'comparison': 'EQUALS', 'value': 'AWS_EC2_INSTANCE'}]
    },
    maxResults=50
)

for finding in response['findings']:
    print(f"Finding: {finding['title']}")
    print(f"Severity: {finding['severity']}")
    print(f"Resource: {finding['resources'][0]['id']}")
    print(f"Remediation: {finding.get('remediation', {}).get('recommendation', {}).get('text', 'N/A')}")
    print("---")
```

Task 3: Enable and Configure Macie (45 minutes)

1. **Enable Macie:**

```bash
aws macie2 enable-macie
```

2. **Create Data Discovery Job:**

```bash
```

```
# Create classification job for S3 buckets
aws macie2 create-classification-job \
  --name "Sensitive-Data-Discovery" \
  --job-type ONE_TIME \
  --s3-job-definition '{
    "bucketDefinitions": [{
      "accountId": "ACCOUNT-ID",
      "buckets": ["customer-data-bucket", "financial-records-bucket"]
    }],
    "scoping": {
      "includes": {
        "and": [{
          "simpleScopeTerm": {
            "comparator": "EQ",
            "key": "OBJECT_EXTENSION",
            "values": ["csv", "json", "txt", "pdf"]
          }
        }]
      }
    }
  }' \
  --managed-data-identifier-ids \
    $(aws macie2 list-managed-data-identifiers --query 'items[?name==`CREDIT_CARD_NUMBER`].id' --output text) \
    $(aws macie2 list-managed-data-identifiers --query 'items[?name==`USA_SOCIAL_SECURITY_NUMBER`].id' --outpu
```

3. **Create Custom Data Identifier:**

```
bash

# For company-specific sensitive data patterns
aws macie2 create-custom-data-identifier \
  --name "Internal-Employee-ID" \
  --description "Company employee ID format" \
  --regex "EMP-[0-9]{6}" \
  --maximum-match-distance 50 \
  --keywords "employee" "staff" "worker"
```

4. **Review Macie Findings:**

```
python
```

```
import boto3

macie = boto3.client('macie2')

# Get findings
response = macie.list_findings(
    findingCriteria={
        'criterion': {
            'severity.description': {'eq': ['High']},
            'category': {'eq': ['CLASSIFICATION']}
        }
    },
    maxResults=50
)

if response['findingIds']:
    findings = macie.get_findings(findingIds=response['findingIds'])

    for finding in findings['findings']:
        print(f"Bucket: {finding['resourcesAffected']['s3Bucket']['name']}")
        print(f"Object: {finding['resourcesAffected']['s3Object']['key']}")
        print(f"Sensitive Data: {finding['classificationDetails']['result']['sensitiveData']}")
        print("---")
```

Task 4: Configure Security Hub (40 minutes)

1. **Enable Security Hub:**

```
bash

aws securityhub enable-security-hub \
  --enable-default-standards
```

2. **Enable Security Standards:**

```
bash
```

```
# Enable AWS Foundational Security Best Practices
aws securityhub batch-enable-standards \
  --standards-subscription-requests '[{
    "StandardsArn": "arn:aws:securityhub:us-east-1::standards/aws-foundational-security-best-practices/v/1.0.0"
  },
  {
    "StandardsArn": "arn:aws:securityhub:us-east-1::standards/cis-aws-foundations-benchmark/v/1.4.0"
  }]'
```

3. **Configure Product Integrations:**

bash

```
# Enable GuardDuty integration
aws securityhub enable-import-findings-for-product \
  --product-arn arn:aws:securityhub:us-east-1::product/aws/guardduty

# Enable Inspector integration
aws securityhub enable-import-findings-for-product \
  --product-arn arn:aws:securityhub:us-east-1::product/aws/inspector

# Enable Macie integration
aws securityhub enable-import-findings-for-product \
  --product-arn arn:aws:securityhub:us-east-1::product/aws/macie
```

4. **Create Custom Insights:**

bash

```
aws securityhub create-insight \
  --name "High-Severity-Production-Findings" \
  --filters '{
    "SeverityLabel": [{"Value": "HIGH", "Comparison": "EQUALS"}],
    "WorkflowStatus": [{"Value": "NEW", "Comparison": "EQUALS"}],
    "ResourceTags": [{
      "Key": "Environment",
      "Value": "production",
      "Comparison": "EQUALS"
    }]
  }' \
  --group-by-attribute "ResourceType"
```

Task 5: Automated Remediation with EventBridge & Lambda (45 minutes)

1. **Create Lambda Remediation Functions:**

Function 1: Disable Exposed Access Key

python

```python
# lambda_disable_exposed_key.py
import boto3
import json

iam = boto3.client('iam')

def lambda_handler(event, context):
    # Parse GuardDuty finding
    finding = event['detail']['findings'][0]
    finding_type = finding['type']

    if 'UnauthorizedAccess:IAMUser' in finding_type:
        # Extract access key ID from finding
        access_key_id = finding['resource']['accessKeyDetails']['accessKeyId']
        username = finding['resource']['accessKeyDetails']['userName']

        try:
            # Disable the compromised access key
            iam.update_access_key(
                UserName=username,
                AccessKeyId=access_key_id,
                Status='Inactive'
            )

            print(f"Disabled access key {access_key_id} for user {username}")

            # Optionally send SNS notification
            sns = boto3.client('sns')
            sns.publish(
                TopicArn='arn:aws:sns:us-east-1:ACCOUNT-ID:SecurityAlerts',
                Subject='ALERT: Compromised IAM Access Key Disabled',
                Message=f"Access key {access_key_id} for user {username} has been automatically disabled due to suspicious act
            )

            return {
                'statusCode': 200,
                'body': json.dumps('Remediation successful')
            }
        except Exception as e:
            print(f"Error: {str(e)}")
            return {
                'statusCode': 500,
                'body': json.dumps(f'Remediation failed: {str(e)}')
            }

    return {
```

```
    'statusCode': 200,
    'body': json.dumps('No action required')
}
```

Function 2: Isolate Compromised Instance

python

```python
# lambda_isolate_instance.py
import boto3

ec2 = boto3.client('ec2')

def lambda_handler(event, context):
    finding = event['detail']['findings'][0]
    instance_id = finding['resource']['instanceDetails']['instanceId']

    try:
        # Get current security groups
        response = ec2.describe_instances(InstanceIds=[instance_id])
        vpc_id = response['Reservations'][0]['Instances'][0]['VpcId']

        # Create isolation security group if it doesn't exist
        try:
            sg_response = ec2.create_security_group(
                GroupName='IsolationSecurityGroup',
                Description='Isolates compromised instances',
                VpcId=vpc_id
            )
            isolation_sg_id = sg_response['GroupId']
        except ec2.exceptions.ClientError as e:
            if 'InvalidGroup.Duplicate' in str(e):
                # Security group exists, get its ID
                sgs = ec2.describe_security_groups(
                    Filters=[{'Name': 'group-name', 'Values': ['IsolationSecurityGroup']}]
                )
                isolation_sg_id = sgs['SecurityGroups'][0]['GroupId']
            else:
                raise

        # Apply isolation security group (allows only SSH from specific IP for forensics)
        ec2.modify_instance_attribute(
            InstanceId=instance_id,
            Groups=[isolation_sg_id]
        )

        # Add tag
        ec2.create_tags(
            Resources=[instance_id],
            Tags=[
                {'Key': 'SecurityStatus', 'Value': 'Isolated'},
                {'Key': 'IsolationReason', 'Value': finding['type']}
            ]
        )
```

```
        print(f"Isolated instance {instance_id}")
        return {'statusCode': 200, 'body': 'Instance isolated successfully'}

    except Exception as e:
        print(f"Error: {str(e)}")
        return {'statusCode': 500, 'body': f'Error: {str(e)}'}
```

2. **Deploy Lambda Functions:**

```
bash

# Package and deploy
zip lambda_disable_key.zip lambda_disable_exposed_key.py
zip lambda_isolate.zip lambda_isolate_instance.py

aws lambda create-function \
  --function-name DisableExposedAccessKey \
  --runtime python3.11 \
  --role arn:aws:iam::ACCOUNT-ID:role/LambdaRemediationRole \
  --handler lambda_disable_exposed_key.lambda_handler \
  --zip-file fileb://lambda_disable_key.zip \
  --timeout 60

aws lambda create-function \
  --function-name IsolateCompromisedInstance \
  --runtime python3.11 \
  --role arn:aws:iam::ACCOUNT-ID:role/LambdaRemediationRole \
  --handler lambda_isolate_instance.lambda_handler \
  --zip-file fileb://lambda_isolate.zip \
  --timeout 60
```

3. **Create EventBridge Rules:**

```
bash
```

```
# Rule for exposed IAM credentials
aws events put-rule \
  --name GuardDutyExposedCredentials \
  --event-pattern '{
    "source": ["aws.guardduty"],
    "detail-type": ["GuardDuty Finding"],
    "detail": {
      "type": ["UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration.OutsideAWS"]
    }
  }'

aws events put-targets \
  --rule GuardDutyExposedCredentials \
  --targets "Id"="1","Arn"="arn:aws:lambda:us-east-1:ACCOUNT-ID:function:DisableExposedAccessKey"

# Rule for compromised EC2 instance
aws events put-rule \
  --name GuardDutyCompromisedInstance \
  --event-pattern '{
    "source": ["aws.guardduty"],
    "detail-type": ["GuardDuty Finding"],
    "detail": {
      "type": ["Backdoor:EC2/C&CActivity.B!DNS", "CryptoCurrency:EC2/BitcoinTool.B!DNS"]
    }
  }'

aws events put-targets \
  --rule GuardDutyCompromisedInstance \
  --targets "Id"="1","Arn"="arn:aws:lambda:us-east-1:ACCOUNT-ID:function:IsolateCompromisedInstance"
```

4. **Create Security Alert SNS Topic:**

```
bash

aws sns create-topic --name SecurityAlerts

aws sns subscribe \
  --topic-arn arn:aws:sns:us-east-1:ACCOUNT-ID:SecurityAlerts \
  --protocol email \
  --notification-endpoint security-team@company.com
```

Validation Checklist

- [ ] GuardDuty enabled and generating findings
- [ ] Inspector scanning EC2 instances for vulnerabilities

- [ ] Macie discovering sensitive data in S3
- [ ] Security Hub aggregating findings from all services
- [ ] EventBridge rules triggering Lambda remediation
- [ ] SNS notifications being sent for critical findings
- [ ] Test remediation with sample GuardDuty findings

Discussion Points

1. What's the difference between GuardDuty, Inspector, and Macie?
2. When should you use automated remediation vs manual review?
3. How do you prevent false positives in security monitoring?
4. What are the compliance benefits of Security Hub?
5. How do you implement this in a multi-account environment?

Workshop 4: Web Application Security {#workshop-4}

Objective

Secure a web application using WAF, Shield, CloudFront, ACM, and API Gateway with comprehensive protection layers.

Prerequisites

- Deployed web application (or sample app)
- Domain name configured in Route 53
- AWS CLI and Terraform/CloudFormation (optional)

Duration: 2.5 hours

Architecture Overview

```
Secure Web Application Stack
│
├── Route 53 (DNS)
│   └── CloudFront Distribution
│       ├── AWS WAF (Protection Rules)
│       ├── Shield Standard (DDoS Protection)
│       └── ACM Certificate (SSL/TLS)
│           └── Application Load Balancer
│               ├── API Gateway (REST APIs)
```

```
|           |  ├── Cognito User Pool (Authentication)
|           |  └── Lambda Authorizers
|           └── EC2 / ECS (Application)
```

Tasks

Task 1: Provision SSL/TLS Certificate with ACM (20 minutes)

1. **Request Public Certificate:**

```
bash

aws acm request-certificate \
  --domain-name myapp.example.com \
  --subject-alternative-names "*.myapp.example.com" \
  --validation-method DNS \
  --region us-east-1
```

2. **Validate Domain Ownership via Route 53:**

```
bash

# Get validation record
cert_arn=$(aws acm list-certificates --query 'CertificateSummaryList[0].CertificateArn' --output text)

aws acm describe-certificate --certificate-arn $cert_arn

# Add CNAME record to Route 53 (automated)
aws acm wait certificate-validated --certificate-arn $cert_arn
```

3. **Verify Certificate Status:**

```
bash

aws acm describe-certificate --certificate-arn $cert_arn --query 'Certificate.Status'
```

Task 2: Configure CloudFront with WAF (60 minutes)

1. **Create WAF Web ACL:**

```
bash
```

```
aws wafv2 create-web-acl \
  --name MyAppWAF \
  --scope CLOUDFRONT \
  --default-action Allow={} \
  --region us-east-1 \
  --rules file://waf-rules.json
```

waf-rules.json:

```
json
```

```
[
  {
    "Name": "AWSManagedRulesCommonRuleSet",
    "Priority": 1,
    "Statement": {
      "ManagedRuleGroupStatement": {
        "VendorName": "AWS",
        "Name": "AWSManagedRulesCommonRuleSet"
      }
    },
    "OverrideAction": {
      "None": {}
    },
    "VisibilityConfig": {
      "SampledRequestsEnabled": true,
      "CloudWatchMetricsEnabled": true,
      "MetricName": "CommonRuleSetMetric"
    }
  },
  {
    "Name": "AWSManagedRulesSQLiRuleSet",
    "Priority": 2,
    "Statement": {
      "ManagedRuleGroupStatement": {
        "VendorName": "AWS",
        "Name": "AWSManagedRulesSQLiRuleSet"
      }
    },
    "OverrideAction": {
      "None": {}
    },
    "VisibilityConfig": {
      "SampledRequestsEnabled": true,
      "CloudWatchMetricsEnabled": true,
      "MetricName": "SQLiRuleSetMetric"
    }
  },
  {
    "Name": "RateLimitRule",
    "Priority": 3,
    "Statement": {
      "RateBasedStatement": {
        "Limit": 2000,
        "AggregateKeyType": "IP"
      }
    },
```

```
    "Action": {
      "Block": {}
    },
    "VisibilityConfig": {
      "SampledRequestsEnabled": true,
      "CloudWatchMetricsEnabled": true,
      "MetricName": "RateLimitMetric"
    }
  },
  {
    "Name": "GeoBlockingRule",
    "Priority": 4,
    "Statement": {
      "GeoMatchStatement": {
        "CountryCodes": ["KP", "IR", "SY"]
      }
    },
    "Action": {
      "Block": {}
    },
    "VisibilityConfig": {
      "SampledRequestsEnabled": true,
      "CloudWatchMetricsEnabled": true,
      "MetricName": "GeoBlockMetric"
    }
  }
]
```

2. **Create CloudFront Distribution:**

```
bash

aws cloudfront create-distribution --distribution-config file://cloudfront-config.json
```

cloudfront-config.json (simplified):

```
json
```

```
{
 "CallerReference": "my-app-cf-2024",
 "Aliases": {
  "Quantity": 1,
  "Items": ["myapp.example.com"]
 },
 "DefaultRootObject": "index.html",
 "Origins": {
  "Quantity": 1,
  "Items": [{
   "Id": "ALB-Origin",
   "DomainName": "myapp-alb-123456.us-east-1.elb.amazonaws.com",
   "CustomOriginConfig": {
    "HTTPPort": 80,
    "HTTPSPort": 443,
    "OriginProtocolPolicy": "https-only",
    "OriginSslProtocols": {
     "Quantity": 1,
     "Items": ["TLSv1.2"]
    }
   }
  }]
 },
 "DefaultCacheBehavior": {
  "TargetOriginId": "ALB-Origin",
  "ViewerProtocolPolicy": "redirect-to-https",
  "AllowedMethods": {
   "Quantity": 7,
   "Items": ["GET", "HEAD", "OPTIONS", "PUT", "POST", "PATCH", "DELETE"],
   "CachedMethods": {
    "Quantity": 2,
    "Items": ["GET", "HEAD"]
   }
  },
  "ForwardedValues": {
   "QueryString": true,
   "Cookies": {"Forward": "all"},
   "Headers": {
    "Quantity": 1,
    "Items": ["Host"]
   }
  },
  "MinTTL": 0,
  "DefaultTTL": 86400,
  "MaxTTL": 31536000,
  "Compress": true
```

```
  },
  "ViewerCertificate": {
    "ACMCertificateArn": "arn:aws:acm:us-east-1:ACCOUNT-ID:certificate/xxxxx",
    "SSLSupportMethod": "sni-only",
    "MinimumProtocolVersion": "TLSv1.2_2021"
  },
  "WebACLId": "arn:aws:wafv2:us-east-1:ACCOUNT-ID:global/webacl/MyAppWAF/xxxxx",
  "Enabled": true
}
```

3. **Configure Custom WAF Rules:**

python

```python
import boto3

wafv2 = boto3.client('wafv2', region_name='us-east-1')

# Create IP set for whitelist
ip_set_response = wafv2.create_ip_set(
    Name='TrustedIPs',
    Scope='CLOUDFRONT',
    IPAddressVersion='IPV4',
    Addresses=[
        '203.0.113.0/24',  # Office IP range
        '198.51.100.0/24'   # Partner IP range
    ]
)

# Create custom rule using IP set
rule = {
    'Name': 'AllowTrustedIPs',
    'Priority': 0,
    'Statement': {
        'IPSetReferenceStatement': {
            'ARN': ip_set_response['Summary']['ARN']
        }
    },
    'Action': {'Allow': {}},
    'VisibilityConfig': {
        'SampledRequestsEnabled': True,
        'CloudWatchMetricsEnabled': True,
        'MetricName': 'TrustedIPsMetric'
    }
}
```

Task 3: Configure API Gateway with Security (50 minutes)

1. **Create Cognito User Pool:**

```
bash

aws cognito-idp create-user-pool \
  --pool-name MyAppUserPool \
  --policies '{
    "PasswordPolicy": {
      "MinimumLength": 12,
      "RequireUppercase": true,
      "RequireLowercase": true,
      "RequireNumbers": true,
      "RequireSymbols": true
    }
  }' \
  --auto-verified-attributes email \
  --mfa-configuration OPTIONAL

# Create app client
aws cognito-idp create-user-pool-client \
  --user-pool-id us-east-1_xxxxxxxx \
  --client-name MyAppClient \
  --explicit-auth-flows ALLOW_USER_PASSWORD_AUTH ALLOW_REFRESH_TOKEN_AUTH \
  --generate-secret
```

2. **Create REST API with Cognito Authorizer:**

```
bash
```

```
# Create API
api_id=$(aws apigateway create-rest-api \
  --name SecureAPI \
  --endpoint-configuration types=REGIONAL \
  --query 'id' --output text)

# Get root resource
root_id=$(aws apigateway get-resources \
  --rest-api-id $api_id \
  --query 'items[0].id' --output text)

# Create Cognito authorizer
auth_id=$(aws apigateway create-authorizer \
  --rest-api-id $api_id \
  --name CognitoAuthorizer \
  --type COGNITO_USER_POOLS \
  --provider-arns arn:aws:cognito-idp:us-east-1:ACCOUNT-ID:userpool/us-east-1_xxxxxxxx \
  --identity-source method.request.header.Authorization \
  --query 'id' --output text)
```

3. **Create Protected API Resource:**

bash

```
# Create /users resource
resource_id=$(aws apigateway create-resource \
  --rest-api-id $api_id \
  --parent-id $root_id \
  --path-part users \
  --query 'id' --output text)

# Create GET method with authorization
aws apigateway put-method \
  --rest-api-id $api_id \
  --resource-id $resource_id \
  --http-method GET \
  --authorization-type COGNITO_USER_POOLS \
  --authorizer-id $auth_id \
  --request-parameters method.request.header.Authorization=true

# Create usage plan
plan_id=$(aws apigateway create-usage-plan \
  --name BasicPlan \
  --throttle rateLimit=100,burstLimit=200 \
  --quota limit=10000,period=DAY \
  --query 'id' --output text)
```

4. **Create Lambda Authorizer (Custom):**

```
python
```

```
# lambda_authorizer.py
import json
import time

def lambda_handler(event, context):
    token = event['authorizationToken']

    # Validate token (simplified example)
    if token == 'allow':
        effect = 'Allow'
    else:
        effect = 'Deny'

    policy = {
        'principalId': 'user123',
        'policyDocument': {
            'Version': '2012-10-17',
            'Statement': [{
                'Action': 'execute-api:Invoke',
                'Effect': effect,
                'Resource': event['methodArn']
            }]
        },
        'context': {
            'userId': 'user123',
            'role': 'admin'
        }
    }

    return policy
```

Task 4: Implement Shield Advanced & Configure Alarms (30 minutes)

1. **Subscribe to Shield Advanced (Optional - costs $3000/month):**

```
bash

# Note: This incurs significant cost
aws shield subscribe-to-shield-advanced
```

2. **Create CloudWatch Alarms for WAF:**

```
bash
```

```
# Alarm for blocked requests
aws cloudwatch put-metric-alarm \
  --alarm-name WAF-High-Block-Rate \
  --alarm-description "Alert when WAF blocks spike" \
  --metric-name BlockedRequests \
  --namespace AWS/WAFV2 \
  --statistic Sum \
  --period 300 \
  --threshold 1000 \
  --comparison-operator GreaterThanThreshold \
  --evaluation-periods 2 \
  --dimensions Name=Rule,Value=ALL Name=WebACL,Value=MyAppWAF

# Alarm for rate limit hits
aws cloudwatch put-metric-alarm \
  --alarm-name API-Gateway-Throttling \
  --metric-name Count \
  --namespace AWS/ApiGateway \
  --statistic SampleCount \
  --period 60 \
  --threshold 5000 \
  --comparison-operator GreaterThanThreshold \
  --dimensions Name=ApiName,Value=SecureAPI
```

3. **Create Dashboard:**

```
python
```

```
import boto3
import json

cloudwatch = boto3.client('cloudwatch')

dashboard_body = {
    "widgets": [
        {
            "type": "metric",
            "properties": {
                "metrics": [
                    ["AWS/WAFV2", "BlockedRequests", {"stat": "Sum"}],
                    [".", "AllowedRequests", {"stat": "Sum"}]
                ],
                "period": 300,
                "stat": "Sum",
                "region": "us-east-1",
                "title": "WAF Request Status"
            }
        },
        {
            "type": "metric",
            "properties": {
                "metrics": [
                    ["AWS/ApiGateway", "4XXError", {"stat": "Sum"}],
                    [".", "5XXError", {"stat": "Sum"}],
                    [".", "Count", {"stat": "SampleCount"}]
                ],
                "period": 300,
                "stat": "Sum",
                "region": "us-east-1",
                "title": "API Gateway Metrics"
            }
        }
    ]
}

cloudwatch.put_dashboard(
    DashboardName='WebAppSecurity',
    DashboardBody=json.dumps(dashboard_body)
)
```

Task 5: Testing Security Controls (30 minutes)

1. **Test WAF SQL Injection Protection:**

```bash
# Should be blocked
curl -X GET "https://myapp.example.com/api/users?id=1' OR '1'='1"

# Check WAF logs
aws wafv2 get-sampled-requests \
  --web-acl-arn arn:aws:wafv2:us-east-1:ACCOUNT-ID:global/webacl/MyAppWAF/xxxxx \
  --rule-metric-name SQLiRuleSetMetric \
  --scope CLOUDFRONT \
  --time-window StartTime=$(date -u -d '10 minutes ago' +%s),EndTime=$(date -u +%s) \
  --max-items 10
```

2. **Test Rate Limiting:**

```bash
# Simulate high request rate
for i in {1..3000}; do
  curl -s https://myapp.example.com/ > /dev/null &
done
wait

# Check if requests are being throttled
```

3. **Test API Authentication:**

```bash
# Without token (should fail)
curl -X GET https://api.myapp.example.com/users

# With valid Cognito token (should succeed)
# First, get token
token=$(aws cognito-idp initiate-auth \
  --client-id <client-id> \
  --auth-flow USER_PASSWORD_AUTH \
  --auth-parameters USERNAME=testuser,PASSWORD=TestPass123! \
  --query 'AuthenticationResult.IdToken' --output text)

# Use token
curl -X GET https://api.myapp.example.com/users \
  -H "Authorization: Bearer $token"
```

4. **Verify HTTPS Enforcement:**

```bash
# HTTP request should redirect to HTTPS
curl -I http://myapp.example.com/

# Check certificate
echo | openssl s_client -servername myapp.example.com -connect myapp.example.com:443 2>/dev/null | openssl x509 -noou
```

Validation Checklist

- [ ] ACM certificate issued and validated
- [ ] CloudFront distribution created with WAF attached
- [ ] WAF rules blocking SQL injection and XSS attempts
- [ ] Rate limiting preventing request floods
- [ ] Cognito authentication working for API Gateway
- [ ] CloudWatch alarms configured for security events
- [ ] All traffic enforced to HTTPS
- [ ] Geo-blocking working for specified countries

Discussion Points

1. What's the difference between CloudFront and ALB for web applications?
2. When should you use Cognito vs Lambda authorizers?
3. How does WAF protect against Layer 7 attacks while Shield protects Layer 3/4?
4. What are the cost implications of Shield Advanced vs Standard?
5. How do you balance security and performance with WAF rules?

Additional Comprehensive MCQ Questions

Question 26: Secrets Manager Auto-Rotation

Scenario: A company uses RDS MySQL databases across development, staging, and production environments. Database credentials are stored in Secrets Manager. The security policy requires production credentials to rotate every 30 days, but development credentials can remain static.

Question: How should this be implemented?

A) Enable auto-rotation for all secrets with 30-day interval
B) Enable auto-rotation only for production secrets with 30-day interval, keep dev secrets without rotation
C) Manually rotate all credentials monthly
D) Store credentials in Parameter Store instead

Answer: B

Explanation: Secrets Manager supports per-secret rotation configuration. Enable automatic rotation with a Lambda function for production secrets while leaving development secrets without rotation to reduce operational complexity where security requirements are lower.

Question 27: Control Tower Guardrails

Scenario: A financial institution needs to enforce that:

- No S3 buckets can be publicly accessible
- MFA delete must be enabled on all S3 buckets
- CloudTrail must be enabled in all accounts
- All EBS volumes must be encrypted

Question: What's the best implementation approach using Control Tower?

A) Use only preventive guardrails to block non-compliant actions
B) Implement both preventive (SCPs) and detective (AWS Config rules) guardrails for comprehensive coverage
C) Use only detective guardrails to monitor compliance
D) Control Tower cannot enforce these requirements

Answer: B

Explanation: Control Tower provides both preventive guardrails (using SCPs to prevent actions) and detective guardrails (using AWS Config to detect non-compliance). Using both provides defense in depth: prevention stops issues before they occur, detection catches anything that slips through.

Question 28: VPC Flow Logs Analysis

Scenario: A security team needs to analyze network traffic patterns to detect potential data exfiltration attempts and identify the top talkers in their AWS environment.

Question: What's the most effective solution?

A) Enable VPC Flow Logs, send to S3, analyze with Athena and create QuickSight dashboards
B) Use only CloudTrail for network monitoring

C) Enable GuardDuty only
D) Install third-party agents on all EC2 instances

Answer: A

Explanation: VPC Flow Logs capture IP traffic information. Storing in S3 allows analysis with Athena (SQL queries), and QuickSight can visualize patterns. While GuardDuty uses Flow Logs for threat detection, direct analysis provides custom insights for data exfiltration patterns and traffic analysis.

Conclusion

These workshops and questions cover comprehensive AWS security topics including:

- IAM identity management and access control
- Multi-account strategies with Organizations
- Encryption using KMS
- Security monitoring with GuardDuty, Inspector, Macie, and Security Hub
- Web application security with WAF, Shield, CloudFront, and API Gateway
- Automated remediation and compliance

Each workshop is designed to be hands-on and can be completed in 2-3 hours, providing practical experience with AWS security services.