Amazon Security Automation & Scripting

Overview

Master AWS security automation at Amazon scale through boto3 scripting, multi-account security auditing, and business impact-driven automation strategies.

Amazon's Automation Interview Focus

Interview Structure (30 minutes of technical discussion)

- **Live coding challenge** - Write security automation scripts using boto3
- **Architecture design** - Design scalable security systems for 1000+ services
- **Business justification** - Calculate ROI and efficiency gains
- **AWS service integration** - Demonstrate knowledge of Security Hub, GuardDuty, Config
- **Scale considerations** - Solutions that work at Amazon's global scale

Common Interview Questions

- "How would you automate security scanning for 1000 microservices?"
- "Design a system to detect and respond to security incidents automatically"
- "Write a script to audit AWS security configurations across multiple accounts"
- "How would you calculate the business value of security automation?"
- "Design automated compliance monitoring for SOC2 and PCI DSS requirements"

Core Security Automation Examples

Enterprise-Scale AWS Security Auditing

Multi-Account Security Auditor Class

```
import boto3
import json
from concurrent.futures import ThreadPoolExecutor
from dataclasses import dataclass
from typing import List, Dict
import logging

@dataclass
class SecurityFinding:
    account_id: str
    service: str
    resource: str
    severity: str
    finding: str
    remediation: str
    business_impact: str
```

```python
class AmazonSecurityAuditor:
    """Enterprise-scale AWS security auditing for Amazon interview"""

    def __init__(self, assume_role_arn_template: str):
        self.role_template = assume_role_arn_template
        self.findings = []
        self.metrics = {
            'accounts_scanned': 0,
            'findings_by_severity': {'CRITICAL': 0, 'HIGH': 0, 'MEDIUM': 0, 'LOW':
0},
            'scan_duration': 0,
            'cost_avoidance': 0
        }

    def audit_all_accounts(self, account_ids: List[str]) -> List[SecurityFinding]:
        """Parallel security auditing across multiple AWS accounts"""

        with ThreadPoolExecutor(max_workers=10) as executor:
            futures = [
                executor.submit(self.audit_single_account, account_id)
                for account_id in account_ids
            ]

            for future in futures:
                account_findings = future.result()
                self.findings.extend(account_findings)
                self.metrics['accounts_scanned'] += 1

        return self.findings

    def audit_single_account(self, account_id: str) -> List[SecurityFinding]:
        """Comprehensive security audit for single AWS account"""

        # Assume role in target account
        session = self._assume_role(account_id)

        # Parallel service auditing
        audit_functions = [
            self._audit_iam_security,
            self._audit_s3_security,
            self._audit_ec2_security,
            self._audit_rds_security,
            self._audit_vpc_security,
            self._audit_cloudtrail_config,
            self._audit_guardduty_status
        ]

        account_findings = []
        with ThreadPoolExecutor(max_workers=5) as executor:
            futures = [executor.submit(func, session) for func in audit_functions]

            for future in futures:
                service_findings = future.result()
                account_findings.extend(service_findings)
```

```
        return account_findings

    def _audit_iam_security(self, session) -> List[SecurityFinding]:
        """IAM security assessment - Amazon's critical focus area"""

        iam = session.client('iam')
        findings = []

        # Check for root account MFA
        summary = iam.get_account_summary()
        if summary['SummaryMap'].get('AccountMFAEnabled', 0) == 0:
            findings.append(SecurityFinding(
                account_id=session.account_id,
                service='IAM',
                resource='Root Account',
                severity='CRITICAL',
                finding='Root account does not have MFA enabled',
                remediation='Enable MFA for root account immediately',
                business_impact='Complete account compromise possible - $50M+
potential impact'
            ))

        # Check for overprivileged users
        paginator = iam.get_paginator('list_users')
        for page in paginator.paginate():
            for user in page['Users']:
                user_policies =
iam.list_attached_user_policies(UserName=user['UserName'])

                # Check for AdministratorAccess
                for policy in user_policies['AttachedPolicies']:
                    if policy['PolicyName'] == 'AdministratorAccess':
                        findings.append(SecurityFinding(
                            account_id=session.account_id,
                            service='IAM',
                            resource=user['UserName'],
                            severity='HIGH',
                            finding=f"User {user['UserName']} has
AdministratorAccess",
                            remediation='Apply principle of least privilege',
                            business_impact='Excessive privileges increase breach
impact'
                        ))

        # Check for unused access keys
        for page in paginator.paginate():
            for user in page['Users']:
                access_keys = iam.list_access_keys(UserName=user['UserName'])

                for key in access_keys['AccessKeyMetadata']:
                    # Get last used information
                    last_used =
iam.get_access_key_last_used(AccessKeyId=key['AccessKeyId'])
```

```
                    if not last_used.get('AccessKeyLastUsed'):
                        findings.append(SecurityFinding(
                            account_id=session.account_id,
                            service='IAM',
                            resource=key['AccessKeyId'],
                            severity='MEDIUM',
                            finding='Unused access key detected',
                            remediation='Delete unused access keys',
                            business_impact='Reduces attack surface, improves
security posture'
                        ))

        return findings

    def _audit_s3_security(self, session) -> List[SecurityFinding]:
        """S3 security audit - Data protection focus"""

        s3 = session.client('s3')
        findings = []

        # List all buckets
        buckets = s3.list_buckets()

        for bucket in buckets['Buckets']:
            bucket_name = bucket['Name']

            try:
                # Check public access
                public_access = s3.get_public_access_block(Bucket=bucket_name)
                if not all([
                    public_access['PublicAccessBlockConfiguration']
['BlockPublicAcls'],
                    public_access['PublicAccessBlockConfiguration']
['IgnorePublicAcls'],
                    public_access['PublicAccessBlockConfiguration']
['BlockPublicPolicy'],
                    public_access['PublicAccessBlockConfiguration']
['RestrictPublicBuckets']
                ]):
                    findings.append(SecurityFinding(
                        account_id=session.account_id,
                        service='S3',
                        resource=bucket_name,
                        severity='CRITICAL',
                        finding='S3 bucket allows public access',
                        remediation='Enable S3 Block Public Access',
                        business_impact='Data breach risk - $165 per exposed
record'
                    ))

                # Check encryption
                try:
                    encryption = s3.get_bucket_encryption(Bucket=bucket_name)
```

```
                except s3.exceptions.ClientError:
                    findings.append(SecurityFinding(
                        account_id=session.account_id,
                        service='S3',
                        resource=bucket_name,
                        severity='HIGH',
                        finding='S3 bucket not encrypted at rest',
                        remediation='Enable S3 default encryption with KMS',
                        business_impact='Compliance violation, data exposure risk'
                    ))

                # Check logging
                try:
                    logging_config = s3.get_bucket_logging(Bucket=bucket_name)
                    if 'LoggingEnabled' not in logging_config:
                        findings.append(SecurityFinding(
                            account_id=session.account_id,
                            service='S3',
                            resource=bucket_name,
                            severity='MEDIUM',
                            finding='S3 bucket logging not enabled',
                            remediation='Enable S3 access logging',
                            business_impact='Reduced incident response capability'
                        ))
                except:
                    pass

            except Exception as e:
                logging.error(f"Error auditing bucket {bucket_name}: {e}")

        return findings

    def _audit_ec2_security(self, session) -> List[SecurityFinding]:
        """EC2 security audit - Infrastructure protection"""

        ec2 = session.client('ec2')
        findings = []

        # Check security groups
        security_groups = ec2.describe_security_groups()

        for sg in security_groups['SecurityGroups']:
            for rule in sg['IpPermissions']:
                # Check for 0.0.0.0/0 access on sensitive ports
                for ip_range in rule.get('IpRanges', []):
                    if ip_range.get('CidrIp') == '0.0.0.0/0':
                        from_port = rule.get('FromPort', 0)

                        if from_port in [22, 3389, 1433, 3306, 5432]:  # SSH, RDP,
DB ports
                            findings.append(SecurityFinding(
                                account_id=session.account_id,
                                service='EC2',
                                resource=sg['GroupId'],
```

```
                                severity='CRITICAL',
                                finding=f'Security group allows 0.0.0.0/0 access
on port {from_port}',
                                remediation='Restrict access to specific IP
ranges',
                                business_impact='Direct server access for
attackers'
                            ))

        # Check for unencrypted EBS volumes
        volumes = ec2.describe_volumes()
        for volume in volumes['Volumes']:
            if not volume.get('Encrypted', False):
                findings.append(SecurityFinding(
                    account_id=session.account_id,
                    service='EC2',
                    resource=volume['VolumeId'],
                    severity='HIGH',
                    finding='EBS volume not encrypted',
                    remediation='Enable EBS encryption by default',
                    business_impact='Data at rest exposure risk'
                ))

        return findings

    def generate_executive_report(self) -> Dict:
        """Generate Amazon-style executive security report"""

        critical_findings = [f for f in self.findings if f.severity == 'CRITICAL']
        high_findings = [f for f in self.findings if f.severity == 'HIGH']

        # Calculate business impact
        total_risk_value = len(critical_findings) * 10000000  # $10M per critical
        total_risk_value += len(high_findings) * 1000000     # $1M per high

        report = {
            'executive_summary': {
                'total_accounts_audited': self.metrics['accounts_scanned'],
                'critical_findings': len(critical_findings),
                'high_findings': len(high_findings),
                'total_risk_exposure': total_risk_value,
                'immediate_action_required': len(critical_findings) > 0
            },
            'top_risks': [
                {
                    'finding': f.finding,
                    'business_impact': f.business_impact,
                    'accounts_affected': len([x for x in self.findings if
x.finding == f.finding])
                }
                for f in critical_findings[:5]  # Top 5 critical
            ],
            'remediation_priorities': self._generate_remediation_plan(),
            'compliance_status': self._assess_compliance(),
```

```
            'roi_analysis': self._calculate_security_roi()
        }

        return report

    def _generate_remediation_plan(self) -> List[Dict]:
        """Amazon-style remediation prioritization"""

        # Group findings by remediation action
        remediation_groups = {}
        for finding in self.findings:
            action = finding.remediation
            if action not in remediation_groups:
                remediation_groups[action] = []
            remediation_groups[action].append(finding)

        # Prioritize by impact
        prioritized = []
        for action, findings in remediation_groups.items():
            critical_count = len([f for f in findings if f.severity ==
'CRITICAL'])
            high_count = len([f for f in findings if f.severity == 'HIGH'])

            prioritized.append({
                'action': action,
                'affected_resources': len(findings),
                'critical_impact': critical_count,
                'high_impact': high_count,
                'estimated_effort': self._estimate_remediation_effort(action),
                'business_value': critical_count * 10000000 + high_count * 1000000
            })

        return sorted(prioritized, key=lambda x: x['business_value'],
reverse=True)

# Usage example for Amazon interview
def demonstrate_amazon_security_automation():
    """Show Amazon-scale security automation"""

    # Initialize auditor for multi-account organization
    auditor = AmazonSecurityAuditor(
        assume_role_arn_template="arn:aws:iam::
{account_id}:role/SecurityAuditRole"
    )

    # Audit across 100+ accounts (Amazon scale)
    account_ids = [f"12345678901{i:1d}" for i in range(10)]  # Simplified for demo

    print("Starting Amazon-scale security audit...")
    findings = auditor.audit_all_accounts(account_ids)

    print(f"Audit complete: {len(findings)} findings across {len(account_ids)}
accounts")
```

```
    # Generate executive report
    report = auditor.generate_executive_report()

    print(f"Critical findings: {report['executive_summary']
['critical_findings']}")
    print(f"Total risk exposure: ${report['executive_summary']
['total_risk_exposure']:,}")

    return report
```

Automated Incident Response System

```
class AmazonIncidentResponse:
    """Automated incident response for Amazon-scale environments"""

    def __init__(self):
        self.sns_client = boto3.client('sns')
        self.lambda_client = boto3.client('lambda')
        self.ec2_client = boto3.client('ec2')

    def handle_security_incident(self, event):
        """Automated incident response workflow"""

        incident_type = event.get('incident_type')
        severity = event.get('severity')
        affected_resources = event.get('affected_resources', [])

        # Immediate containment for critical incidents
        if severity == 'CRITICAL':
            self._immediate_containment(affected_resources)

        # Automated evidence collection
        evidence = self._collect_evidence(affected_resources)

        # Notify security team
        self._notify_security_team(incident_type, severity, evidence)

        # Customer communication (if needed)
        if self._affects_customers(incident_type):
            self._prepare_customer_communication(incident_type, severity)

        return {
            'incident_id': self._generate_incident_id(),
            'containment_actions': self._get_containment_actions(),
            'evidence_collected': len(evidence),
            'notifications_sent': True
        }

    def _immediate_containment(self, resources):
        """Amazon-scale automated containment"""
```

```
        for resource in resources:
            resource_type = resource.get('type')
            resource_id = resource.get('id')

            if resource_type == 'ec2_instance':
                # Isolate compromised instance
                self._isolate_instance(resource_id)
            elif resource_type == 's3_bucket':
                # Block public access immediately
                self._block_s3_public_access(resource_id)
            elif resource_type == 'iam_user':
                # Disable compromised user
                self._disable_iam_user(resource_id)

    def _isolate_instance(self, instance_id):
        """Quarantine EC2 instance while preserving evidence"""

        # Create forensic security group
        forensic_sg = self.ec2_client.create_security_group(
            GroupName=f'forensic-{instance_id}',
            Description='Forensic isolation security group'
        )

        # Remove all existing security groups, apply forensic SG
        self.ec2_client.modify_instance_attribute(
            InstanceId=instance_id,
            Groups=[forensic_sg['GroupId']]
        )

        # Create EBS snapshot for forensics
        instance = self.ec2_client.describe_instances(InstanceIds=[instance_id])
        for volume in instance['Reservations'][0]['Instances'][0]
['BlockDeviceMappings']:
            volume_id = volume['Ebs']['VolumeId']
            self.ec2_client.create_snapshot(
                VolumeId=volume_id,
                Description=f'Forensic snapshot for incident {instance_id}'
            )
```

Compliance Monitoring Automation

```
class AmazonComplianceAutomation:
    """Automated compliance monitoring and reporting"""

    def __init__(self):
        self.config_client = boto3.client('config')
        self.securityhub_client = boto3.client('securityhub')

    def assess_compliance_posture(self):
        """Real-time compliance assessment across frameworks"""
```

```
        frameworks = ['PCI_DSS', 'SOC2', 'ISO27001', 'GDPR']
        compliance_status = {}

        for framework in frameworks:
            rules = self._get_framework_rules(framework)
            compliance_status[framework] = self._evaluate_compliance(rules)

        return {
            'overall_score': self._calculate_overall_score(compliance_status),
            'framework_scores': compliance_status,
            'remediation_required':
self._get_remediation_requirements(compliance_status),
            'business_impact':
self._assess_compliance_business_impact(compliance_status)
        }

    def _evaluate_compliance(self, rules):
        """Evaluate compliance against specific rules"""

        compliant_rules = 0
        total_rules = len(rules)

        for rule in rules:
            evaluation = self.config_client.get_compliance_details_by_config_rule(
                ConfigRuleName=rule['name']
            )

            if evaluation['EvaluationResults'][0]['ComplianceType'] ==
'COMPLIANT':
                compliant_rules += 1

        return {
            'score': (compliant_rules / total_rules) * 100,
            'compliant_rules': compliant_rules,
            'total_rules': total_rules,
            'non_compliant_rules': total_rules - compliant_rules
        }
```

Amazon Interview Response Framework

Sample Question: "Automate security scanning for 1000 microservices"

Winning Response Structure (5 minutes):

Scale Consideration (30 seconds):

> "For 1000 microservices at Amazon scale, we need distributed, parallel processing that doesn't impact service performance. I'd design an event-driven architecture using SQS, Lambda, and Step Functions to handle this volume."

Technical Architecture (2 minutes):

> "I'd implement a three-tier scanning system:
>
> 1. **Real-time scanning** - API Gateway webhooks trigger Lambda functions on code commits
> 2. **Scheduled deep scans** - EventBridge triggers comprehensive security analysis using Step Functions
> 3. **Continuous monitoring** - Security Hub aggregates findings with automated prioritization
>
> Each service scans in isolated containers to prevent production impact, with results stored in DynamoDB for fast retrieval and trend analysis."

Business Impact (90 seconds):

> "This automation delivers measurable business value:
>
> - Reduces manual security review time from 2000 to 50 hours monthly - $150K monthly savings
> - Increases deployment coverage from 15% to 100% - 80% reduction in security incidents
> - Enables 24/7 security monitoring without additional headcount
> - Provides executive dashboards for compliance reporting and risk visibility"

Scalability Validation (60 seconds):

> "The serverless architecture auto-scales for Amazon's deployment volumes. Using parallel Lambda executions, we scan 1000 services simultaneously, completing full scans in under 10 minutes versus 40+ hours manually. Cost scales linearly with usage - no idle infrastructure costs."

Amazon Automation Success Principles

1. **Customer Obsession**: Every automation improves customer security or experience
2. **Scale Thinking**: Solutions must work for Amazon's global infrastructure
3. **Operational Excellence**: Reliable, maintainable, observable automation
4. **Frugality**: Clear ROI and resource efficiency
5. **Ownership**: End-to-end responsibility for automated systems
6. **Bias for Action**: Automated responses to security events

Business Impact Quantification

ROI Calculation Framework

```
def calculate_security_automation_roi(manual_hours_saved, hourly_cost,
automation_cost):
    """Calculate ROI for security automation initiatives"""
    annual_savings = manual_hours_saved * 12 * hourly_cost
    roi_percentage = ((annual_savings - automation_cost) / automation_cost) * 100

    return {
        'annual_savings': annual_savings,
        'automation_cost': automation_cost,
        'net_savings': annual_savings - automation_cost,
        'roi_percentage': roi_percentage,
        'payback_months': automation_cost / (annual_savings / 12)
    }
```

```
# Example: Multi-account security auditing automation
roi = calculate_security_automation_roi(
    manual_hours_saved=500,  # 500 hours/month saved
    hourly_cost=75,          # $75/hour security engineer cost
    automation_cost=100000  # $100K automation development
)

print(f"ROI: {roi['roi_percentage']:.0f}%")
print(f"Payback period: {roi['payback_months']:.1f} months")
```

This comprehensive automation approach demonstrates the scale thinking, AWS expertise, and business impact focus that Amazon seeks in security engineers.