

Enterprise Cloud Incident Response and Forensic Readiness on AWS

Author: Adedayo

Specialization: Cloud Security & Incident Response

Platform: Amazon Web Services (AWS)

## 1. Introduction

This document describes the design, implementation, and validation of an enterprise cloud incident response and forensic readiness framework in AWS.

I implemented this framework to ensure that security incidents involving IAM credentials, malicious API activity, data exposure, and infrastructure misconfigurations can be detected, investigated, contained, and remediated efficiently.

The framework integrates AWS-native security services, documented response playbooks, forensic preservation procedures, and automated remediation workflows.

## 2. Objectives

The primary objectives were to:

- Improve cloud incident detection and response capability
- Establish standardized response playbooks
- Preserve forensic evidence securely
- Reduce incident response time
- Enable automated containment
- Support audit and compliance requirements

## 3. Incident Response Evidence Repository

A dedicated S3 bucket was created to store forensic evidence.

The bucket was configured with:

- Encryption enabled
- Versioning enabled
- Restricted access policies
- Deletion protection

The screenshot shows the AWS CloudTrail Trails configuration page for the 'clouddefender-ir-trail'. The 'General details' section includes:

- Trail logging:** Logging is enabled.
- Trail name:** clouddefender-ir-trail.
- Multi-region trail:** Yes.
- Apply trail to my organization:** Enabled for all accounts.
- Trail log location:** clouddefender-ir-bucket/AWSLogs/o-pn0spcswbk/879381257906
- Last log file delivered:** -
- Log file SSE-KMS encryption:** Enabled.
- AWS KMS key:** arn:aws:kms:us-east-1:879381257906:key/31ad8ef2-e278-4bed-9a37-7df4dbd9a2d5
- AWS KMS key alias:** clouddefender-kmskey.
- Log file validation:** Enabled.
- Last file validation delivered:** -
- SNS notification delivery:** Disabled.
- Last SNS notification:** -

The 'CloudWatch Logs' section indicates 'No CloudWatch Logs log groups' and 'CloudWatch Logs is not configured for this trail'.

This repository stores CloudTrail logs, CloudWatch exports, snapshots, and investigation artifacts.

#### 4. Centralized Logging for Incident Response

AWS CloudTrail was configured for centralized API logging.

CloudTrail serves as the primary forensic data source and supports investigation of:

- Credential compromise
- Privilege escalation
- Resource tampering
- Data access

Logs are automatically delivered to the evidence repository.

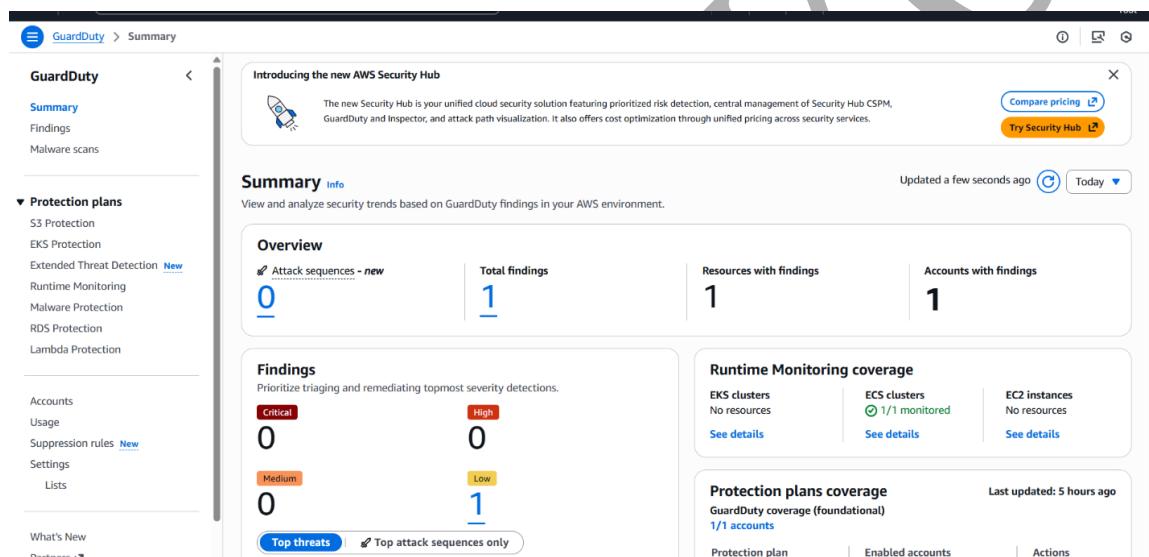
#### 5. Threat Detection Services

## 5.1 Amazon GuardDuty

GuardDuty was enabled for continuous threat detection.

It supports detection of:

- IAM credential compromise
- Malicious API behavior
- Data exfiltration
- EC2 compromise
- Lambda abuse



GuardDuty findings trigger incident response workflows.

## 5.2 AWS Security Hub

Security Hub was enabled for centralized findings management.

Findings from GuardDuty and AWS security standards are aggregated and prioritized for response.

## 6. Incident Simulation and Testing

## 6.1 Incident 1: IAM Credential Misuse

Detection Source: CloudTrail

The screenshot shows the CloudTrail console with the 'Event history' tab selected. A search bar at the top right contains the query 'test-user'. Below it is a table with columns: Event name, Event time, User name, Event source, Resource type, and Resource name. The table lists several events from January 18, 2026, such as 'ListUsers', 'ListBuckets', 'DescribeInstances', 'ListRoles', 'GetCallerIdentity', and 'ListUsers' again, all performed by the user 'test-user' from the source 'iam.amazonaws.com'.

Unusual API enumeration activity was detected from an IAM user.

Triage Actions:

- Reviewed CloudTrail events
- Identified access key and source IP
- Confirmed suspicious behavior.

The screenshot shows the CloudTrail console with the 'Event history' tab selected and the specific event for 'ListUsers' highlighted. The 'JSON view' button is clicked, displaying the raw JSON event data. The JSON object includes fields like eventversion, userIdentity, principalId, arn, accountid, accesskeyid, username, eventtime, eventsource, eventname, awsregion, sourceipaddress, useragent, requestparameters, responseslements, requestid, eventid, readonly, eventtype, managementevent, recipientaccountid, eventcategory, and tlsdetails.

Severity: SEV 2 (High)

Forensic Preservation:

- CloudTrail logs archived
- IAM metadata documented

Containment:

- Access key deactivated

Outcome:

- No data loss
- Early containment

Lessons Learned:

- Manual log review remains critical
- Early isolation reduces impact

## 6.2 Incident 2: EC2 Public SSH Exposure

Detection Source: Security Hub

An EC2 instance was found with SSH open to 0.0.0.0/0.

Triage Actions:

- Reviewed security group rules
- Verified public IP exposure

Severity: SEV 2 (High)

## Forensic Preservation:

- EBS snapshot created
- Metadata recorded

## Containment:

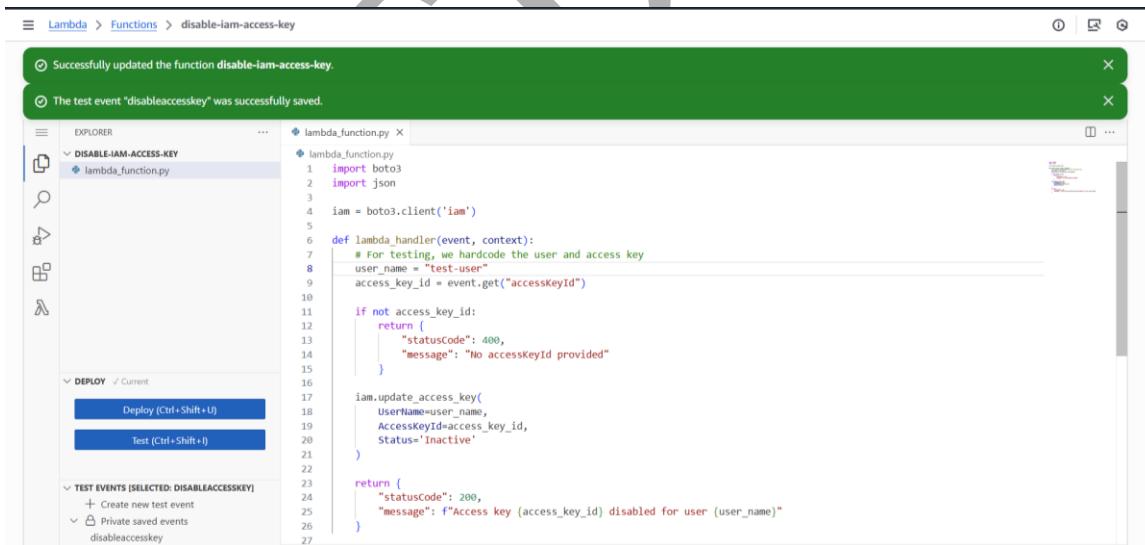
- SSH access restricted

## Outcome:

- No unauthorized access
- Risk eliminated

## 7. Automated Remediation Framework

An automated remediation workflow was implemented using AWS Lambda.



The screenshot shows the AWS Lambda function editor interface. The top navigation bar shows 'Lambda > Functions > disable-iam-access-key'. A green status bar at the top indicates 'Successfully updated the function disable-iam-access-key.' and 'The test event "disableaccesskey" was successfully saved.' The left sidebar has sections for 'EXPLORER', 'TEST EVENTS [SELECTED: DISABLEACCESSKEY]', and 'DEPLOY'. Under 'TEST EVENTS', there are buttons for 'Deploy (Ctrl+Shift+U)' and 'Test (Ctrl+Shift+J)'. The main area displays the code for 'lambda\_function.py':

```
lambda_function.py
1 import boto3
2 import json
3
4 iam = boto3.client('iam')
5
6 def lambda_handler(event, context):
7     # For testing, we hardcode the user and access key
8     user_name = "test-user"
9     access_key_id = event.get("accessKeyId")
10
11    if not access_key_id:
12        return {
13            "statusCode": 400,
14            "message": "No accessKeyId provided"
15        }
16
17    iam.update_access_key(
18        UserName=user_name,
19        AccessKeyId=access_key_id,
20        Status='Inactive'
21    )
22
23    return {
24        "statusCode": 200,
25        "message": f"Access key {access_key_id} disabled for user {user_name}"
26    }
```

## The workflow:

- Receives compromised access key IDs
- Disables keys

- Logs actions to CloudWatch

Keys are deactivated, not deleted, to preserve evidence.

Testing confirmed successful automated containment.

The screenshot displays two screenshots from the AWS console. The top screenshot shows the Lambda function logs for 'disable-iam-access-key'. It includes a green success message and a log entry indicating the function succeeded with a status code of 200 and a message about disabling an access key for user 'test-user'. The bottom screenshot shows the IAM user 'test-user' configuration. It lists the ARN as arn:aws:iam:879381257906:user/test-user, with console access disabled. It shows two access keys: one inactive and used yesterday, and another active and created yesterday. The 'Permissions' tab is selected, showing a single policy named 'ReadOnlyAccess' attached directly to the user. Other tabs include 'Groups', 'Tags', 'Security credentials', and 'Last Accessed'.

CloudWatch > Log management > /aws/lambda/disable-lam-access-key > 2026/01/19/[LATEST]0b641ecca99540629fc7d6631fcfb

**Log events**

You can use the filter bar below to search for and match terms, phrases, or values in your log events. [Learn more about filter patterns](#)

Filter events - press enter to search

Actions ▾ Start tailing Create metric filter

Clear 1m 30m 1h 12h Custom UTC timezone

Timestamp	Message
2026-01-19T22:52:43.211Z	INIT_START Runtime Version: python:3.14.v32 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:1ee4e6d61a50fb29d03b87572cc627d0a...
2026-01-19T22:52:43.716Z	START RequestId: 8fe9c674-9e5a-4a9a-bbad-6b79ec7c193b Version: \$LATEST
2026-01-19T22:52:44.012Z	END RequestId: 8fe9c674-9e5a-4a9a-bbad-6b79ec7c193b
2026-01-19T22:52:44.012Z	REPORT RequestId: 8fe9c674-9e5a-4a9a-bbad-6b79ec7c193b Duration: 295.03 ms Billed Duration: 797 ms Memory Size: 128 MB Max Memory Use...

No newer events at this moment. [Auto retry paused.](#) [Resume](#)

## 8. Alerting for Logging Tampering

CloudWatch alarms were created to detect:

- StopLogging
- DeleteTrail

These alerts provide early warning of audit log tampering.

## 9. Incident Response Playbooks

Cloud-specific response playbooks were developed for:

- Credential compromise
- API abuse
- Data exfiltration
- Resource misconfiguration

Each playbook defines detection, triage, preservation, containment, and escalation steps.

## 10. Forensic Preservation Procedures

Tested procedures include:

- CloudTrail archival
- CloudWatch export to S3
- EBS snapshots
- IAM key deactivation

All evidence is encrypted and access-controlled.

#### 11. Governance and Documentation

All response workflows, evidence handling procedures, and automation processes were documented.

Documentation supports audits, investigations, and team readiness.

#### 12. Outcomes and Impact

This implementation delivered:

- Faster incident response
- Strong forensic readiness
- Automated containment
- Improved detection coverage
- Reduced operational risk
- Audit-ready processes

#### 13. Conclusion

I designed and implemented a comprehensive cloud incident response and forensic readiness framework on AWS.

Through integrated detection, evidence preservation, automation, and governance, this solution enables effective and resilient security operations.