

# Enterprise Continuous Monitoring and Threat Detection on AWS

Author: Adedayo

Specialization: Cloud Security & Threat Detection

Platform: Amazon Web Services (AWS)

## 1. Introduction

This document describes the design and implementation of a centralized monitoring and threat detection framework in an AWS environment.

I implemented this solution to improve visibility into API activity, network traffic, security misconfigurations, and threat indicators, while enabling rapid incident response.

## 2. Objectives

The primary objectives were to:

- Enable continuous security monitoring
- Detect suspicious IAM and API activity
- Identify network-based threats
- Centralize security findings
- Automate low-severity remediation
- Improve incident response readiness

## 3. Security Monitoring Architecture

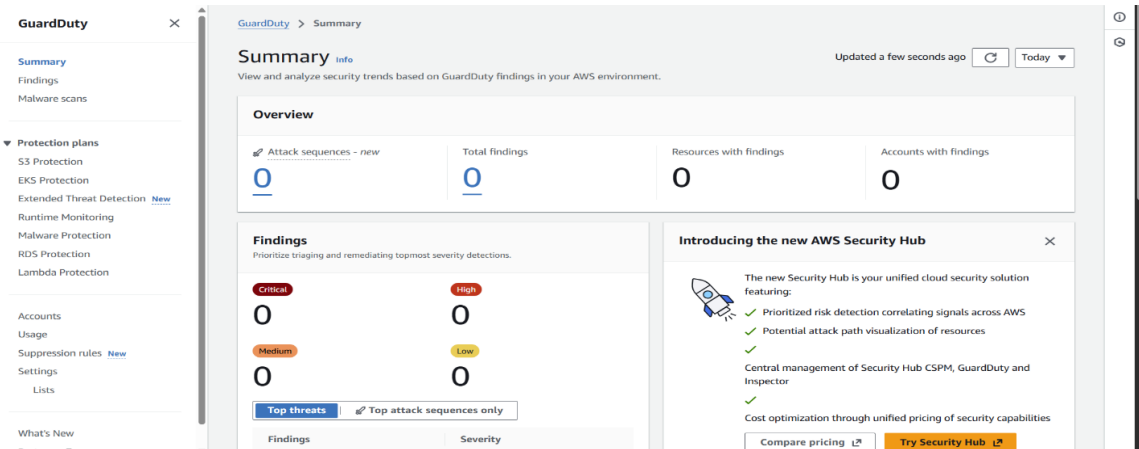
The monitoring framework was built using the following AWS services:

- Amazon GuardDuty
- AWS CloudTrail
- AWS Security Hub
- VPC Flow Logs
- Amazon CloudWatch
- Amazon SNS
- AWS Lambda

All security events and findings were centralized for analysis and response.

#### 4. GuardDuty Configuration

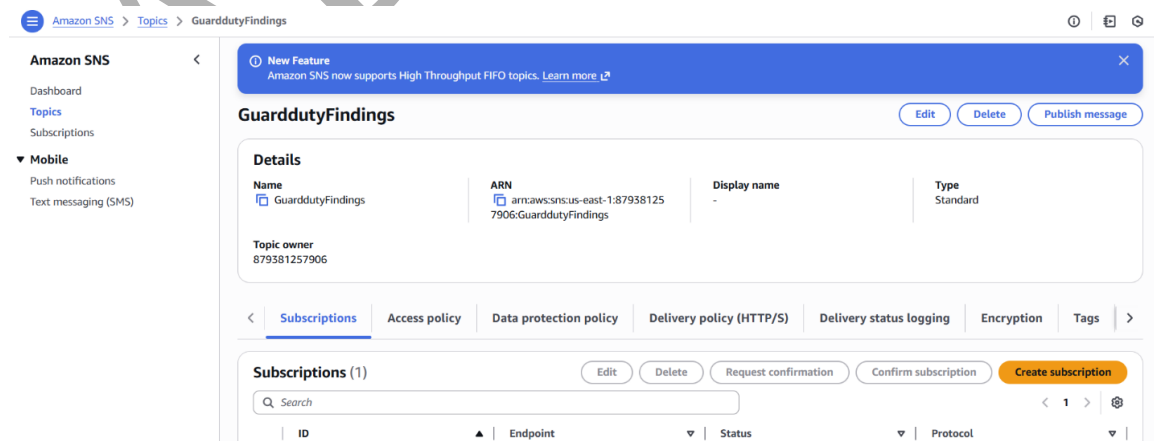
I enabled Amazon GuardDuty to monitor:



- CloudTrail logs
- VPC Flow Logs
- DNS activity

GuardDuty was configured to generate findings for suspicious behavior and compromised credentials.

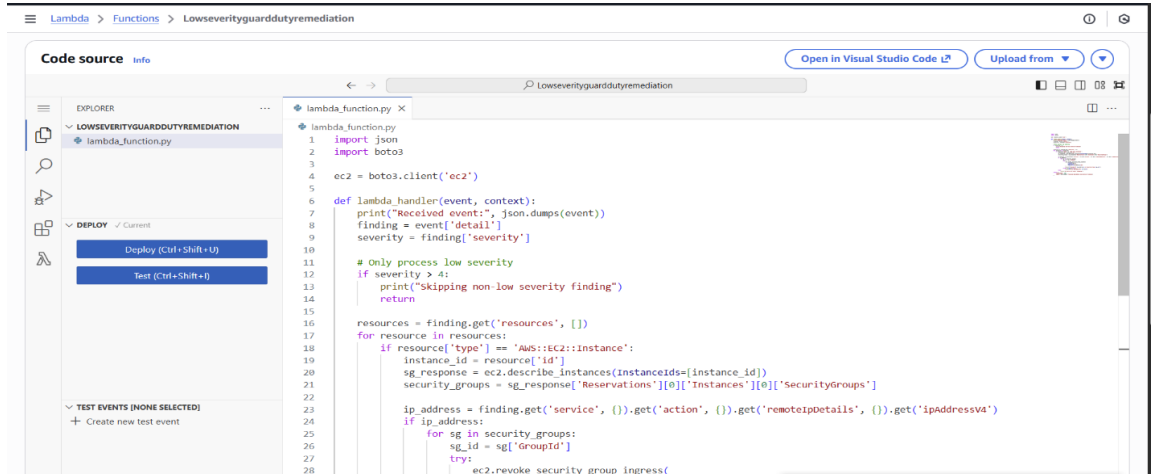
An SNS topic was created to distribute GuardDuty alerts to the security team.



Sample findings were generated to validate alert delivery.

## 5. Automated Remediation for Low-Severity Findings

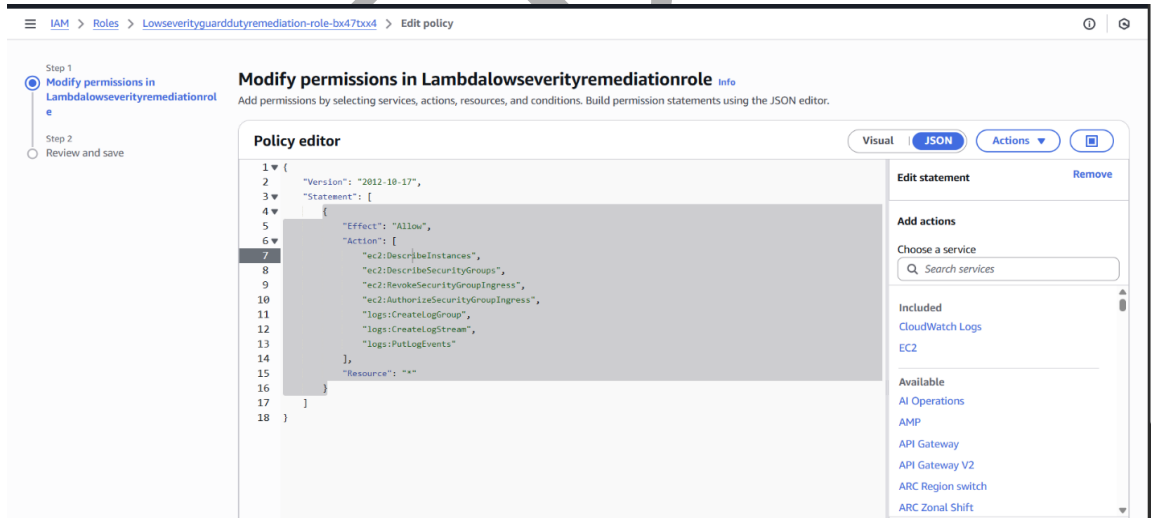
I implemented automated remediation for low-severity findings using AWS Lambda.



The screenshot shows the AWS Lambda console interface for the function 'Lowseverityguarddutyremediation'. The 'Code source' tab is active, displaying the Python code for the lambda\_handler function. The code imports 'json' and 'boto3', creates an EC2 client, and defines a handler that processes low-severity findings by describing instances and their associated security groups, then revoking the security group ingress.

```
1 import json
2 import boto3
3
4 ec2 = boto3.client('ec2')
5
6 def lambda_handler(event, context):
7     print("Received event:", json.dumps(event))
8     finding = event['detail']
9     severity = finding['severity']
10
11     # Only process low severity
12     if severity > 4:
13         print("Skipping non-low severity finding")
14         return
15
16     resources = finding.get('resources', [])
17     for resource in resources:
18         if resource['type'] == 'AWS::EC2::Instance':
19             instance_id = resource['id']
20             sg_response = ec2.describe_instances(instanceids=[instance_id])
21             security_groups = sg_response['Reservations'][0]['Instances'][0]['SecurityGroups']
22
23             ip_address = finding.get('service', {}).get('action', {}).get('remoteipdetails', {}).get('ipaddress')
24             if ip_address:
25                 for sg in security_groups:
26                     sg_id = sg['GroupId']
27                     try:
28                         ec2.revoke_security_group_ingress(
```

A dedicated IAM role was created with permissions to modify tagged security groups.



The screenshot shows the AWS IAM console interface for the role 'Lowseverityguarddutyremediationrole'. The 'Policy editor' tab is active, displaying the JSON policy document. The policy grants 'Allow' permissions for actions such as 'ec2:DescribeInstances', 'ec2:DescribeSecurityGroups', 'ec2:RevokeSecurityGroupIngress', 'ec2:AuthorizeSecurityGroupIngress', 'logs:CreateLogGroup', 'logs:CreateLogStream', and 'logs:PutLogEvents' on all resources.

```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": [
7         "ec2:DescribeInstances",
8         "ec2:DescribeSecurityGroups",
9         "ec2:RevokeSecurityGroupIngress",
10        "ec2:AuthorizeSecurityGroupIngress",
11        "logs:CreateLogGroup",
12        "logs:CreateLogStream",
13        "logs:PutLogEvents"
14      ],
15      "Resource": "*"
16    }
17  ]
18 }
```

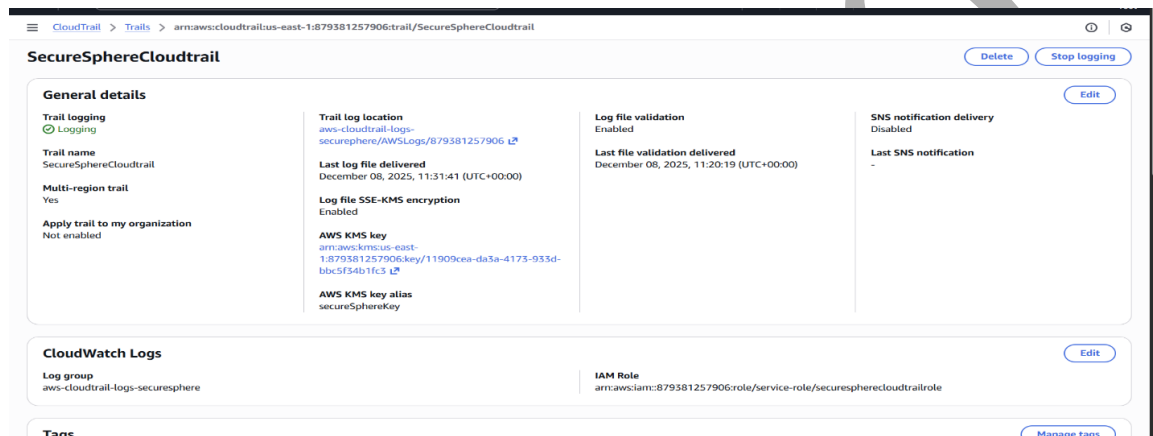
In production environments, permissions would be restricted to specific ARNs and tagged resources to enforce least privilege.

A Lambda function was deployed and triggered using EventBridge when low-severity findings occurred.

## 6. CloudTrail and API Monitoring

I configured a centralized CloudTrail trail to capture all AWS API activity.

Logs were forwarded to CloudWatch Logs for near real-time analysis.



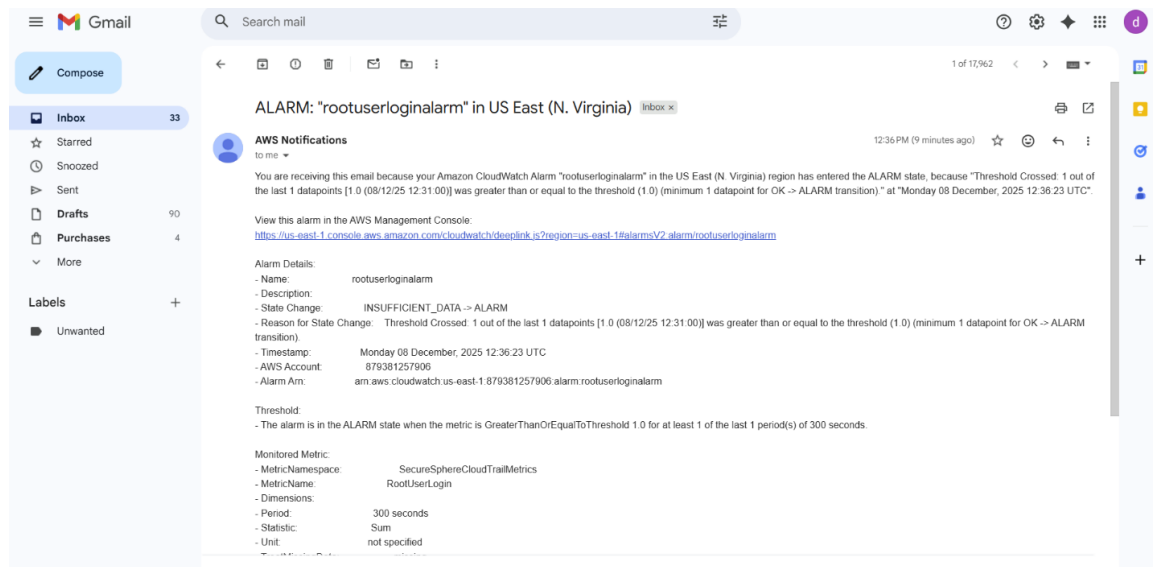
SNS notifications were configured to alert the security team when suspicious events were detected.

## 7. CloudWatch Metric Filters and Alarms

Metric filters were created for critical security events, including:

- Unauthorized API calls
- Console login without MFA
- Root account usage
- IAM policy changes
- CloudTrail configuration changes
- AWS Config changes
- S3 bucket policy changes

Each metric was linked to a CloudWatch alarm and SNS notification.



Manual test events were generated to validate metric activation and alert delivery.

## 8. Security Hub Integration

AWS Security Hub was enabled to aggregate findings from multiple services, including:

- AWS Config
- GuardDuty
- Inspector
- Macie

Security standards were reviewed and findings were prioritized based on risk level.

## 9. S3 Security Monitoring and Remediation

A test S3 bucket was created with public access enabled to validate detection capabilities.

GuardDuty and Security Hub detected the misconfiguration.

The screenshot shows the AWS Security Hub console with the 'All findings' view selected. A filter is applied: Status = New, Status = In Progress, Status = Other, Status = Unknown, Activity name != Close, and Severity = High. The finding is categorized as 'High' severity. The description states: 'All bucket-level block public access settings were disabled for the S3 bucket. Access to the bucket is controlled by ...'. The finding is associated with the account 879381257906 in the us-east-1 region, created 3 hours ago on December 08, 2025, at 16:43 UTC+00:00.

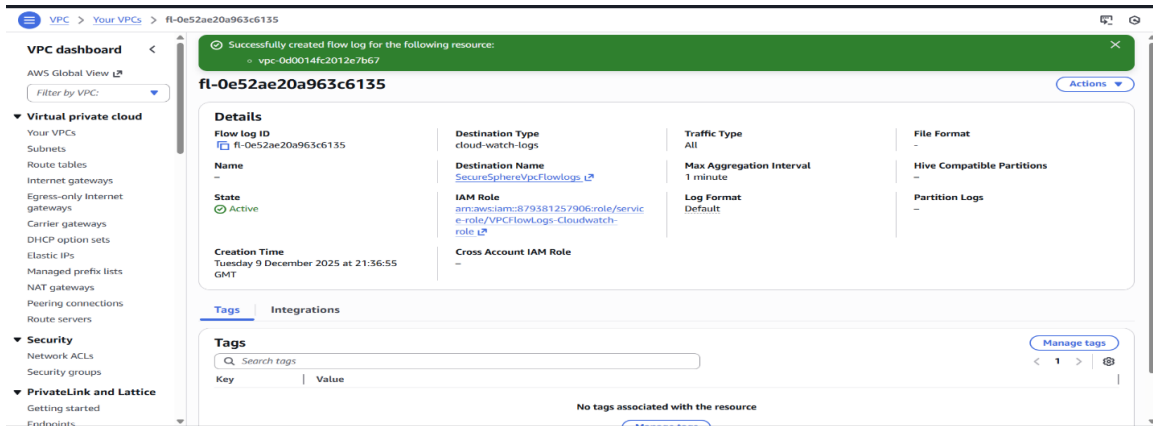
Public access was removed to remediate the high severity finding.

The screenshot shows the 'Overview' and 'Compliance' sections for the finding. The 'Overview' section includes a 'Finding count' graph showing a peak in findings around October 27. The 'Compliance' section shows the finding is in 'Pass' status, with a control score of 53.8. The config rule is 'securityhub-s3-bucket-level-public-access-prohibited-f5c4e845'. The standards are 'standards/aws-foundational-security-best-practices/v1.0.0' and 'standards/cis-aws-foundations-benchmark/v/5.0.0'. The requirements are 'CIS AWS Foundations Benchmark v5.0.0/2.1.4.2'.

Server access logging was enabled to improve audit visibility.

## 10. Network Security and VPC Flow Logs

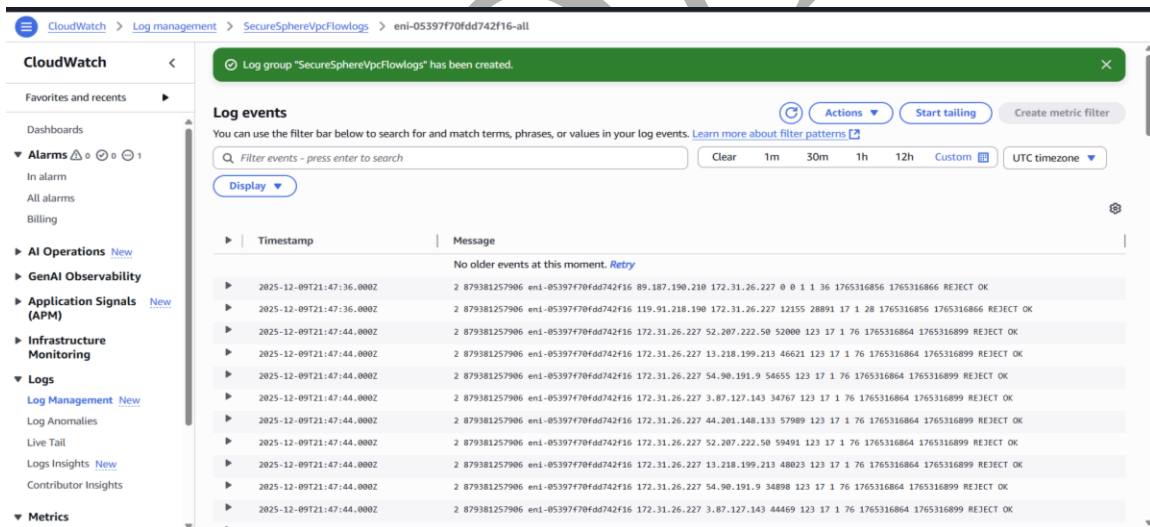
VPC Flow Logs were enabled for all production VPCs.



Logs were sent to CloudWatch Log Groups for monitoring and analysis.

An EC2 instance was used to generate network traffic for validation.

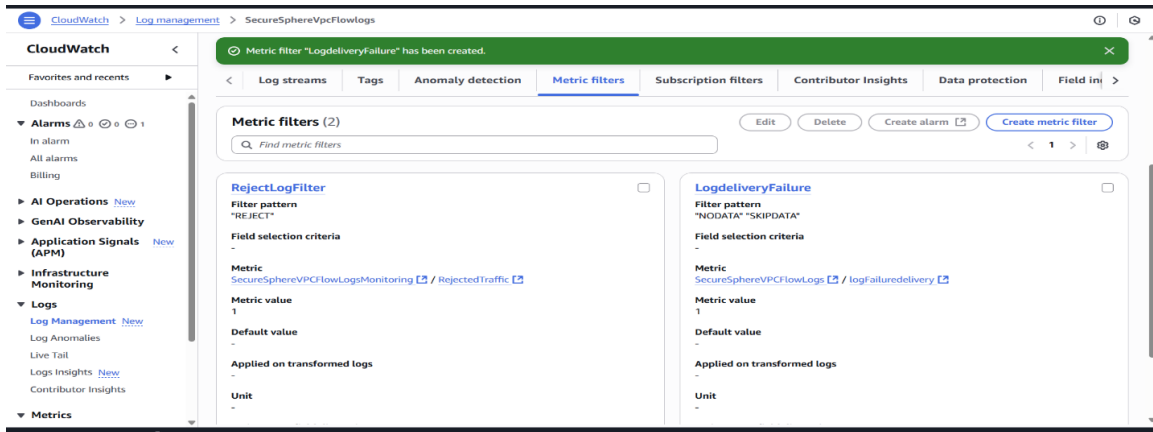
Both ACCEPT and REJECT traffic was verified in log streams.



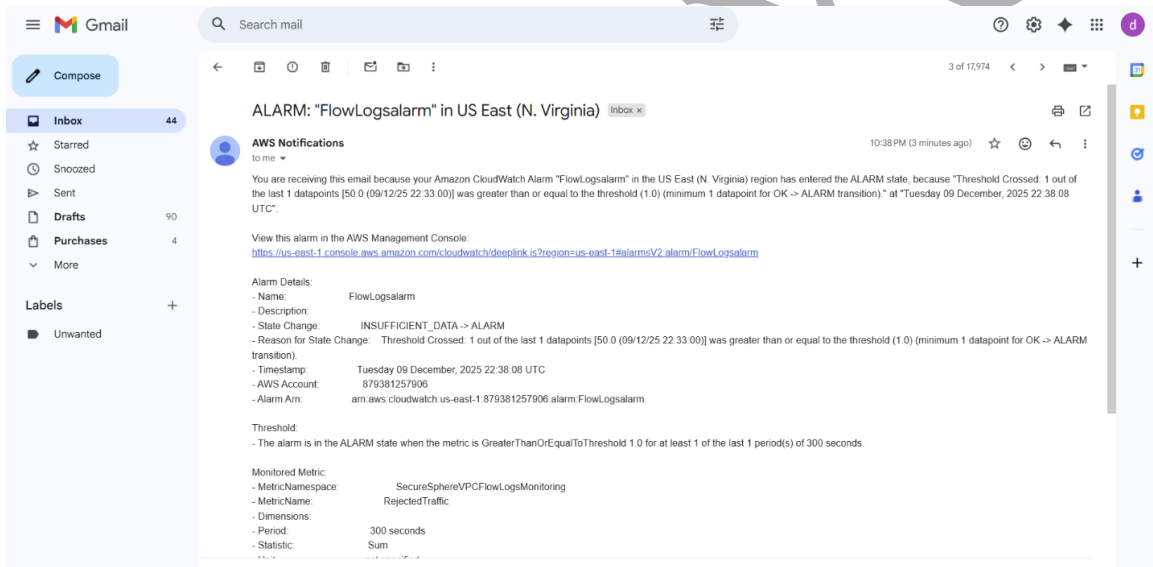
## 11. Network Monitoring Alerts

Metric filters were created for:

- Rejected traffic events
- Log delivery failures



CloudWatch alarms were configured to notify the security team of network anomalies.



## 12. Documentation and Governance

All configurations, alerting workflows, and remediation procedures were documented.

This documentation supported audits, incident response, and knowledge transfer.

## 13. Outcomes and Impact

This implementation delivered the following improvements:



- Centralized threat detection
- Faster incident response
- Improved network visibility
- Automated remediation
- Reduced security misconfigurations
- Enhanced compliance monitoring

#### 14. Conclusion

This monitoring and threat detection framework provides comprehensive visibility into cloud security risks.

Through centralized logging, automated response, and continuous analysis, the environment supports secure and resilient operations.