# AWS Security Audit Tool

## Project Report

## Group Members Roain Kumar 21k-3184 Bhavish kumar 21k-3450 Talha Shahid 21k-3355

## Abstract

This report details the development and implementation of an automated AWS Security Audit Tool. Leveraging AWS native services including Lambda, EventBridge, DynamoDB, S3, IAM, and Access Analyzer, the tool performs scheduled scans of an AWS environment to identify common security misconfigurations such as publicly exposed S3 buckets, overly permissive IAM policies, and misconfigured security groups on EC2 instances. Findings are stored in DynamoDB for structured analysis, detailed reports are saved to S3, and critical issues trigger real-time alerts via SNS. The project demonstrates a practical application of cloud security automation, enhancing the security posture of AWS infrastructure by providing continuous monitoring and actionable insights.

## 1. Introduction

### 1.1 Problem Statement

Cloud infrastructure, while offering flexibility and scalability, presents significant security challenges. Misconfigurations in services like S3, IAM, and EC2 security groups are common causes of data breaches and unauthorized access. Manually auditing these configurations across a dynamic environment is time-consuming and prone to error. There is a need for an automated, scalable, and cost-effective solution to continuously monitor AWS environments for security vulnerabilities and compliance deviations.

### 1.2 Project Objectives

The primary objectives of this project were to:

* Develop an automated tool to scan AWS services for security misconfigurations.
* Utilize AWS native services (Lambda, EventBridge, DynamoDB, S3, SNS, IAM, Access Analyzer) for the core functionality.
* Implement specific checks for public S3 buckets, overly permissive IAM policies, and misconfigured EC2 security groups.
* Store audit findings persistently in a structured database (DynamoDB).
* Generate human-readable reports of audit findings and store them securely (S3).
* Implement a real-time alerting mechanism for critical security findings (SNS).
* Automate the audit process using a scheduler (EventBridge).
* Create a simple web dashboard to view the latest findings.

### 1.3 Scope

The project focuses on auditing the following AWS services and misconfigurations within a single AWS account and specified region:

* **Amazon S3:** Detection of publicly accessible buckets via policies or ACLs.
* **AWS IAM:** Identification of overly permissive IAM policies attached to roles (specifically looking for Action: "\*" on Resource: "\*" or broad service access like iam:\*, sts:\*). Integration with IAM Access Analyzer findings for external access.
* **Amazon EC2 & VPC:** Identification of Security Groups allowing unrestricted inbound access (0.0.0.0/0 or ::/0) to sensitive ports (SSH, RDP, common database ports). Identification of EC2 instances with public IPs associated with these risky Security Groups.

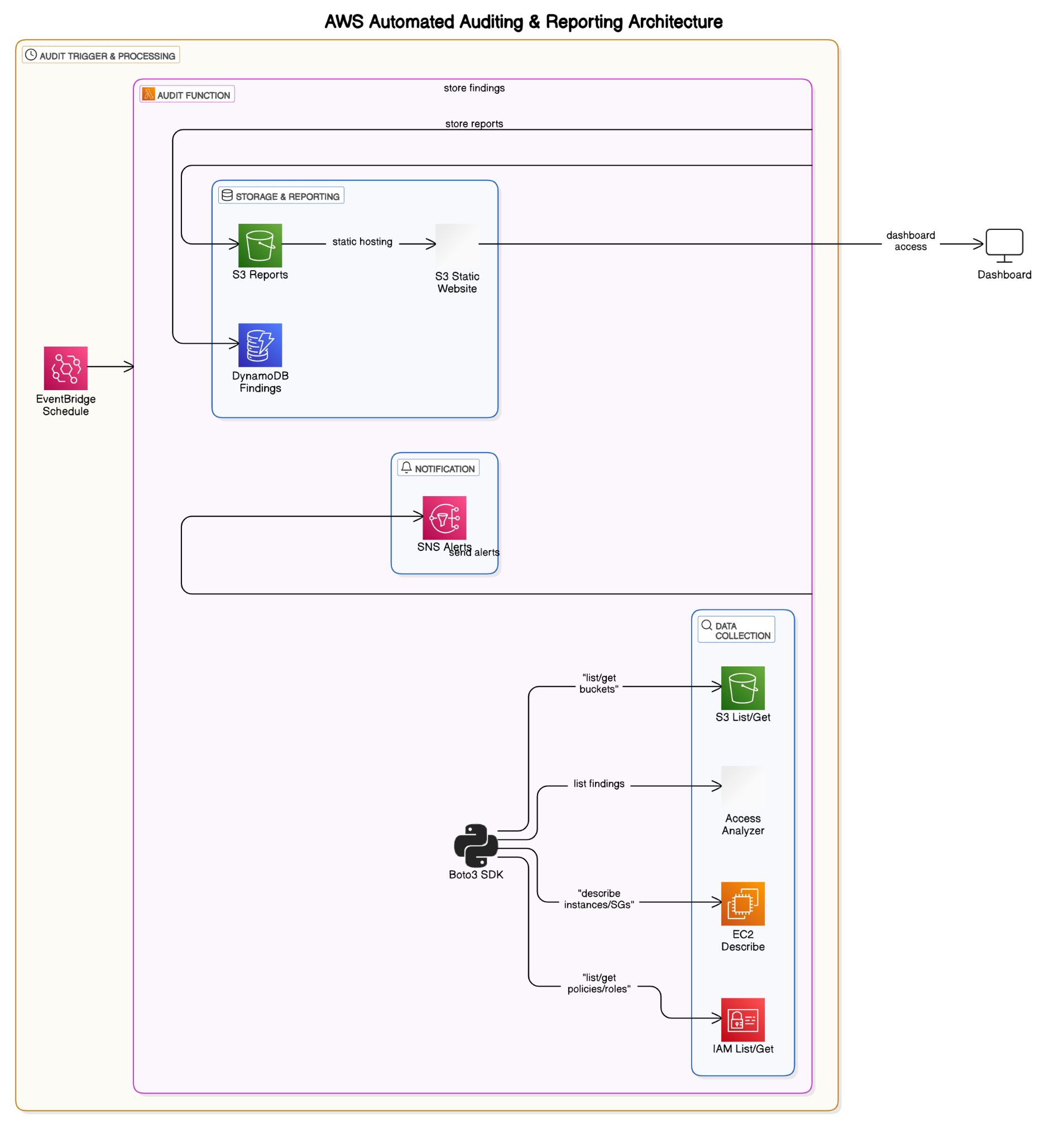
Out of scope for this project are:

* Automated remediation of findings.
* Scanning multiple AWS accounts (though the architecture is adaptable).
* Integrating with advanced vulnerability scanning services like AWS Inspector (beyond potentially checking SSM agent status).
* Complex compliance framework mapping.

## 2. System Design

### 2.1 Architecture

The AWS Security Audit Tool follows a serverless architecture pattern. The key components and their interactions are illustrated below:



* **Amazon EventBridge:** Configured with a schedule (e.g., daily) to trigger the Lambda function.
* **AWS Lambda:** The core compute service. It executes the audit logic, interacts with AWS APIs, processes findings, and coordinates with other services.
* **AWS SDK (Boto3):** Used within the Lambda function to programmatically interact with other AWS services (S3, EC2, IAM, Access Analyzer, DynamoDB, SNS).
* **Amazon S3:** Used for storing:
  + Detailed audit reports (.txt, .csv).
  + A fixed-name latest.csv file for the web dashboard.
  + The static website content (index.html).
* **Amazon DynamoDB:** A NoSQL database used to store structured audit findings persistently. Each finding is stored as an item.
* **AWS IAM:** Provides the execution role for the Lambda function with least-privilege permissions to perform audit actions and interact with other services. Also, the service being audited for policy misconfigurations.
* **IAM Access Analyzer:** A service queried by the Lambda function to identify resources shared with external entities or other AWS accounts.
* **Amazon EC2 & VPC:** Services being audited (Security Groups, EC2 instances).
* **Amazon SNS:** Used as a notification service. The Lambda function publishes messages to an SNS topic for critical findings. Subscribers (e.g., email addresses) receive these alerts.
* **AWS CloudWatch Logs:** Automatically captures logs from the Lambda function, useful for monitoring execution and debugging errors.
* **S3 Static Website Hosting:** Configured on the S3 reports bucket to host the index.html file, providing a simple web interface to view the latest.csv report.

### 2.2 Data Model (DynamoDB)

The SecurityFindings DynamoDB table stores each identified security finding as an item.

* **Table Name:** SecurityFindings (Configured via Lambda Environment Variable)
* **Primary Key:**
  + **Partition Key:** ResourceARN (String) - ARN of the resource (e.g., S3 bucket ARN, IAM Role ARN, Security Group ARN, EC2 Instance ARN). This allows efficient querying of all findings related to a specific resource.
  + **Sort Key:** Timestamp (String - ISO 8601 format) - Allows findings for a resource to be ordered chronologically.
  + *(Alternative: A single FindingID (UUID) as the Partition Key could also be used for simplicity, but querying by resource is less direct.)*
* **Attributes:**
  + FindingID (String - UUID) - Unique identifier for each finding.
  + AccountID (String) - The AWS Account ID where the finding was detected.
  + Region (String) - The AWS region where the resource or finding originated (e.g., ap-south-1 for regional resources, global for IAM policies).
  + ResourceType (String) - Type of AWS resource (e.g., S3Bucket, IAMRole, SecurityGroup, EC2Instance, IAMAccessAnalyzerFinding).
  + ResourceID (String) - Identifier of the resource (e.g., Bucket Name, Role ARN, Security Group ID, Instance ID, Access Analyzer Finding ID).
  + CheckName (String) - The specific check that triggered the finding (e.g., S3PublicAccess, IAMRiskyManagedPolicyAttachment, SecurityGroupUnrestrictedAccess, EC2PublicIPWithOpenSG, IAMAccessAnalyzerFinding).
  + FindingDescription (String) - A detailed description of the misconfiguration or vulnerability.
  + Severity (String) - The severity level (e.g., Critical, High, Medium, Low, Info, ERROR).
  + Timestamp (String) - ISO 8601 formatted timestamp of when the finding was recorded.
  + Status (String - e.g., OPEN, ACKNOWLEDGED, RESOLVED) - Could be used for tracking remediation status (optional for initial scope).
  + AuditRunID (String - UUID) - Identifier for the specific audit run that generated the finding.
  + AssociatedResource (String - Optional) - ARN or ID of a related resource (e.g., the Security Group ID associated with a risky EC2 instance finding).
  + Reference (String - Optional) - Link or ID to an external finding (e.g., IAM Access Analyzer Finding ID).

### 2.3 Choice of AWS Services

* **AWS Lambda:** Chosen for its serverless nature, automatic scaling, and pay-per-execution pricing model, making it ideal for event-driven or scheduled tasks like security audits.
* **Amazon EventBridge:** Provides a reliable and easy-to-configure mechanism for scheduling the Lambda function at regular intervals without managing servers.
* **Amazon DynamoDB:** Selected as a highly available, scalable, and managed NoSQL database that is well-suited for storing semi-structured data like security findings with variable attributes. Its on-demand capacity mode simplifies management.
* **Amazon S3:** Used for cost-effective and highly durable storage of audit reports and static website content. Its static website hosting feature provides a simple way to host the dashboard.
* **AWS IAM:** Essential for defining fine-grained permissions for the Lambda function and is also a critical service being audited for policy misconfigurations.
* **IAM Access Analyzer:** A managed service that simplifies identifying resources shared with external entities, directly contributing to the IAM audit checks.
* **Amazon SNS:** Provides a simple, flexible, and scalable messaging service to send critical alerts via various protocols (email, SMS, etc.).
* **Boto3 (AWS SDK for Python):** The standard and comprehensive library for interacting with AWS services from Python, enabling the Lambda function to perform audit actions.

## 3. Implementation Details

### 3.1 Lambda Function Structure

The core logic resides in the Python Lambda function (lambda\_function.py).

* **Imports:** Necessary libraries (boto3, json, datetime, uuid, os, io, csv, time) are imported.
* **Initialization:** Boto3 clients for required services are initialized using boto3.Session() to ensure regional clients are correctly configured. Environment variables for configuration are retrieved.
* **lambda\_handler(event, context):** The main entry point. It generates a run ID and timestamp, calls individual check functions, collects all findings, stores them in DynamoDB, sends SNS alerts for critical findings, and uploads reports to S3. Includes basic error handling.
* **Check Functions (check\_s3\_public\_access, check\_iam\_permissions, check\_security\_groups, check\_ec2\_vulnerabilities):** Each function focuses on auditing a specific service or type of misconfiguration. They use Boto3 API calls to retrieve resource configurations and logic to identify findings based on defined rules. They return a list of finding dictionaries.
* **Helper Functions (store\_findings\_dynamodb, send\_sns\_alert, generate\_summary\_report, store\_reports\_to\_s3, check\_port\_range, create\_sg\_finding, check\_policy\_document\_for\_risks, parse\_policy\_statements):** Modular functions to handle specific tasks like storing data, sending notifications, formatting reports, and parsing policy documents or security group rules.

### 3.2 Audit Checks Implemented

* **S3 Public Access:** Iterates through all S3 buckets, checks IsPublic status from GetBucketPolicyStatus and GetBucketAcl.
* **IAM Permissions:**
  + Queries IAM Access Analyzer for active findings related to external access.
  + Iterates through IAM Roles, retrieves attached managed policies and inline policies. Parses policy documents to identify Allow statements with Action: "\*" on Resource: "\*" or risky service wildcards (iam:\*, sts:\*).
* **Security Groups:** Iterates through Security Groups, checks ingress (IpPermissions) rules for CidrIp: "0.0.0.0/0" or CidrIpv6: "::/0" combined with sensitive ports (22, 3389, 3306, etc.).
* **EC2 Vulnerabilities:** Iterates through running EC2 instances, identifies those with a PublicIpAddress, and checks if any of their associated Security Groups have unrestricted access to sensitive ports (reusing the SG check logic).

### 3.3 Data Storage (DynamoDB)

* Findings dictionaries returned by check functions are collected.
* The store\_findings\_dynamodb helper uses DynamoDB's batch\_writer to efficiently write multiple items to the SecurityFindings table.

### 3.4 Reporting (S3)

* The generate\_summary\_report helper creates a human-readable plain text summary of all findings from a single run.
* The store\_reports\_to\_s3 helper:
  + Generates a CSV report of all findings.
  + Uploads the plain text summary to s3://[your-bucket]/audit-reports/YYYY-MM-DD/report-uuid.txt.
  + Uploads the CSV report to s3://[your-bucket]/audit-reports/YYYY-MM-DD/report-uuid.csv.
  + Uploads a copy of the same CSV report to a fixed key s3://[your-bucket]/audit-reports/latest.csv for the dashboard.

### 3.5 Alerting (SNS)

* The lambda\_handler filters the collected findings to identify those with Severity: 'Critical'.
* If critical findings exist, the send\_sns\_alert helper is called to publish a summary message to the configured SNS topic.

### 3.6 Automation (EventBridge)

* An EventBridge schedule rule is configured to trigger the Lambda function periodically (e.g., daily).

### 3.7 Dashboard (S3 Static Website)

* A static HTML file (index.html) is uploaded to the root of the S3 reports bucket.
* S3 static website hosting is enabled on the bucket, pointing the index document to index.html.
* A bucket policy grants public s3:GetObject permissions to index.html and audit-reports/latest.csv.
* The JavaScript in index.html fetches the audit-reports/latest.csv file, parses the CSV data, and dynamically populates an HTML table to display the findings.

### 3.8 IAM Permissions

* A dedicated IAM Role (SecurityAuditLambdaRole) is created for the Lambda function.
* A custom IAM Policy is attached to this role, granting least-privilege permissions for:
  + Writing logs to CloudWatch Logs.
  + Reading necessary configurations from S3, EC2, IAM, and Access Analyzer.
  + Writing findings to the DynamoDB table.
  + Uploading reports to the S3 bucket.
  + Publishing messages to the SNS topic.
* The specific policy JSON with required actions and resource ARNs (updated with your account/region/resource names) is provided in the appendix.

## 4. Testing and Validation

### 4.1 Testing Strategy

A combination of manual triggering and creating intentional misconfigurations was used for testing.

### 4.2 Test Cases

* **Basic Execution:** Manually trigger the Lambda function with no expected findings to verify successful execution, logging, and report generation (showing "No findings detected").
* **S3 Public Bucket:** Create an S3 bucket with a public policy or ACL and verify a Critical finding is generated and stored in DynamoDB, included in reports, and triggers an SNS alert.
* **Overly Permissive IAM Policy:** Create a test IAM role with a policy allowing \*:\* or iam:\* on \* resources and verify a Critical or High finding is generated, stored, and reported.
* **IAM Access Analyzer Finding:** If Access Analyzer is enabled, ensure a finding (e.g., sharing a snapshot publicly) results in an IAMAccessAnalyzerFinding in the reports.
* **Risky Security Group:** Create a Security Group allowing 0.0.0.0/0 on port 22 or 3389 and verify a Critical finding is generated and stored.
* **Risky EC2 Instance:** Launch an EC2 instance with a public IP and associate it with the risky Security Group created above. Verify a Critical EC2PublicIPWithOpenSG finding is generated and stored.
* **SNS Alerting:** Confirm that critical findings successfully trigger an email notification via SNS.
* **Report Generation:** Verify that .txt and .csv reports are correctly generated with the expected content and uploaded to the S3 bucket with appropriate naming.
* **Dashboard:** Verify that the S3 static website endpoint loads the index.html page and correctly fetches and displays the data from latest.csv.
* **Scheduled Run:** Allow the EventBridge rule to trigger the Lambda function automatically and verify a successful run and updated reports/dashboard.

### 4.3 Results and Verification

* Screenshots were captured to demonstrate successful Lambda executions in CloudWatch Logs, entries in the DynamoDB table, the generated report files in the S3 bucket, the content of the SNS email alert, and the functional S3 hosted dashboard displaying findings.
* Verification confirmed that findings were accurately reported for intentionally created misconfigurations and that no findings were reported when the environment was clean.

## 5. Challenges and Learnings

* **IAM Permissions:** Determining the minimum necessary permissions for the Lambda role required careful analysis of Boto3 API calls and iterative refinement based on "Access Denied" errors during testing. The principle of least privilege is crucial for security tools themselves.
* **Boto3 API Details:** Understanding the exact structure of responses from various AWS APIs (e.g., Security Group rules, IAM policy documents, Access Analyzer findings) was necessary for accurate parsing and finding identification.
* **S3 Public Access Logic:** Combining checks for Bucket Policy Status, Bucket Policies, and ACLs was important for a comprehensive S3 public access audit.
* **Static Website Hosting & Permissions:** Configuring S3 for static website hosting and setting the correct bucket policy for public read access required specific steps and careful attention to detail.
* **CSV Parsing in JavaScript:** Implementing a simple client-side CSV parser was needed to display the data from the latest.csv file on the HTML dashboard.

## 6. Conclusion

The AWS Security Audit Tool successfully implements an automated, serverless solution for identifying key security misconfigurations in an AWS environment. By leveraging AWS native services, the tool provides continuous monitoring, centralized finding storage, detailed reporting, and real-time critical alerts. This project demonstrates the power of cloud automation in enhancing security posture and contributes to adopting DevSecOps practices. The implemented checks cover critical areas (S3, IAM, EC2 Security Groups) and the architecture is scalable and cost-effective.

## 7. Future Work

Potential enhancements include:

* Implementing more comprehensive security checks for additional AWS services (RDS, Lambda function policies, etc.).
* Adding checks for compliance with specific security benchmarks (e.g., CIS AWS Foundations Benchmark).
* Developing automated or manual remediation actions for identified findings.
* Integrating with AWS Security Hub or Amazon Detective for centralized security posture management and threat investigation.
* Adding support for scanning multiple AWS accounts via AWS Organizations.
* Enhancing the S3 dashboard with filtering, sorting, and visualization capabilities.
* Implementing a mechanism to track the status of findings over time (e.g., OPEN, ACKNOWLEDGED, RESOLVED).

## 8. References

* AWS Boto3 Documentation: <https://boto3.amazonaws.com/v1/documentation/api/latest/index.html>
* AWS IAM Policy Documentation: <https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html>
* AWS S3 Static Website Hosting: <https://docs.aws.amazon.com/AmazonS3/latest/userguide/WebsiteHosting.html>
* AWS EventBridge Documentation: <https://docs.aws.amazon.com/eventbridge/latest/userguide/what-is-eventbridge.html>
* AWS Lambda Documentation: <https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>
* AWS DynamoDB Documentation: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html>
* AWS SNS Documentation: <https://docs.aws.amazon.com/sns/latest/dg/welcome.html>
* AWS IAM Access Analyzer Documentation: <https://docs.aws.amazon.com/IAM/latest/UserGuide/what-is-access-analyzer.html>
* AWS Security Best Practices (Referenced concepts): <https://aws.amazon.com/security/>

## 9. Appendix

### 9.1 Lambda Function Source Code

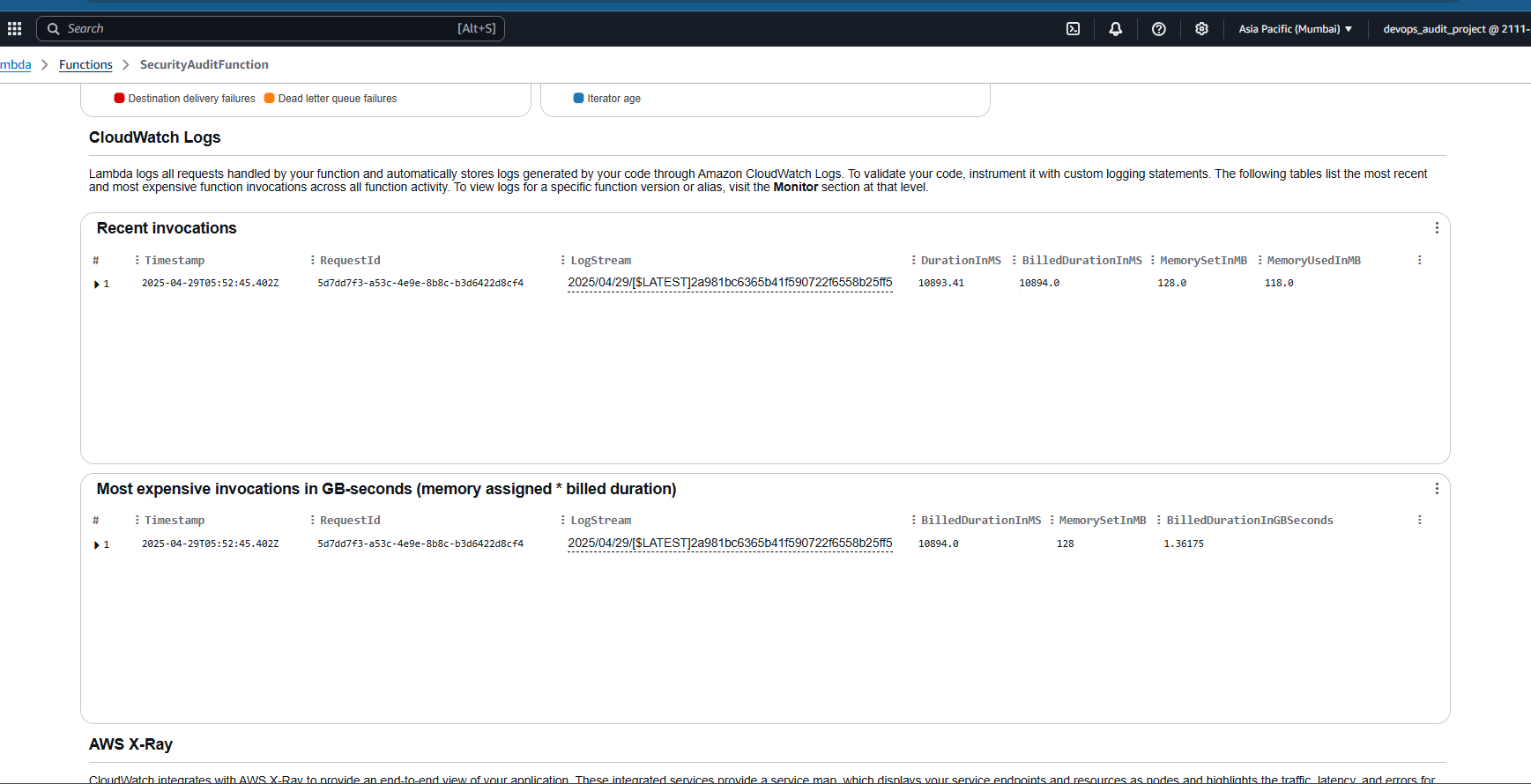
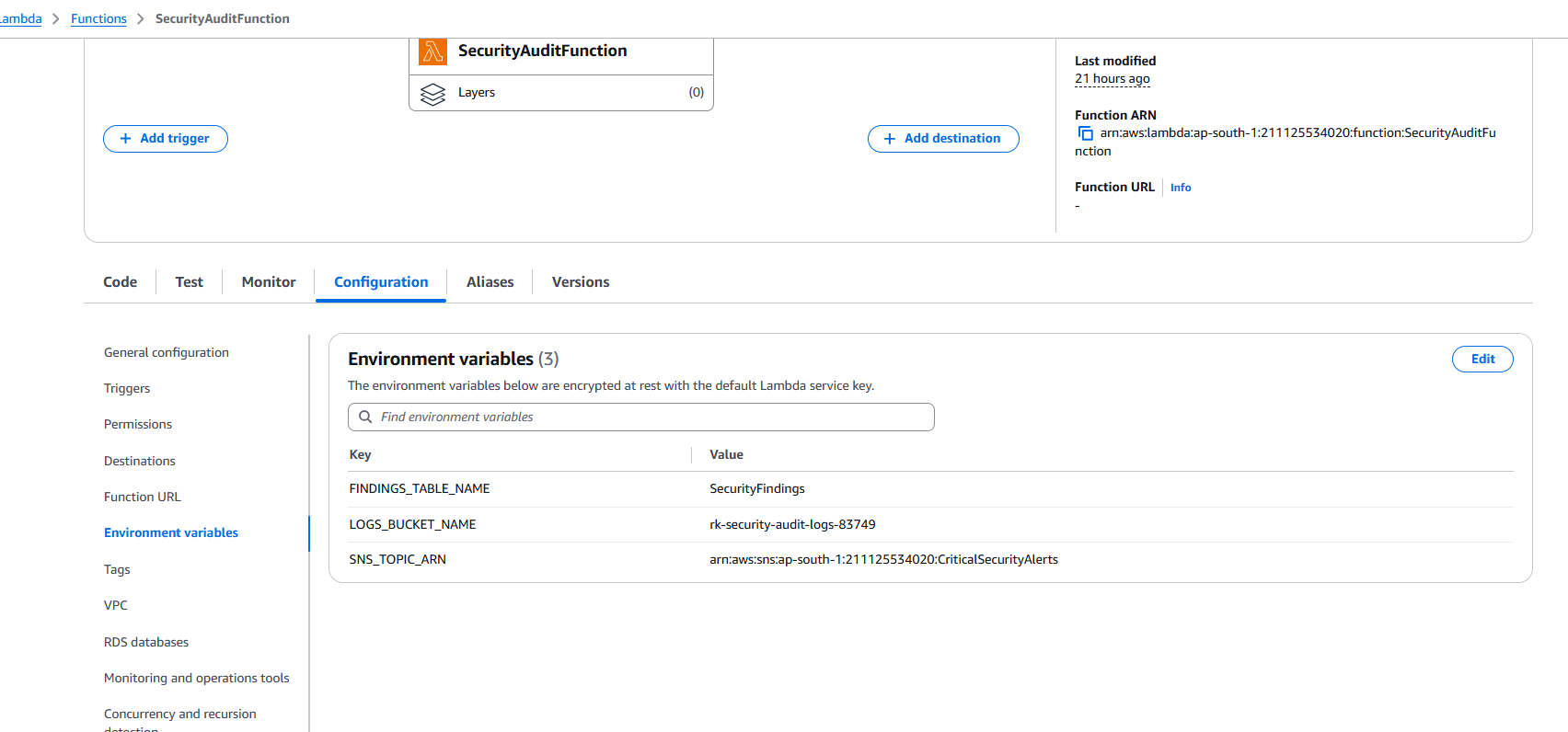
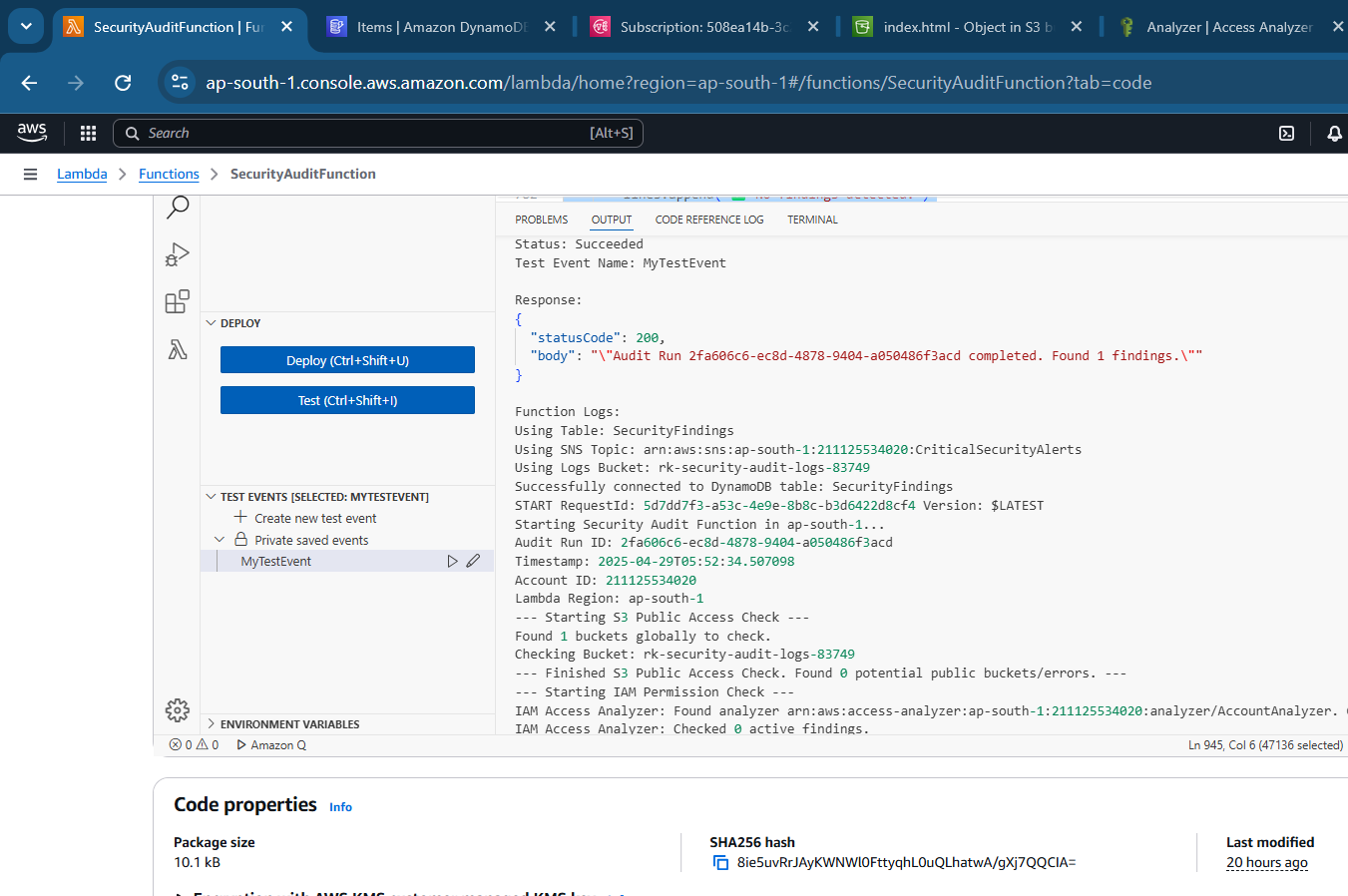
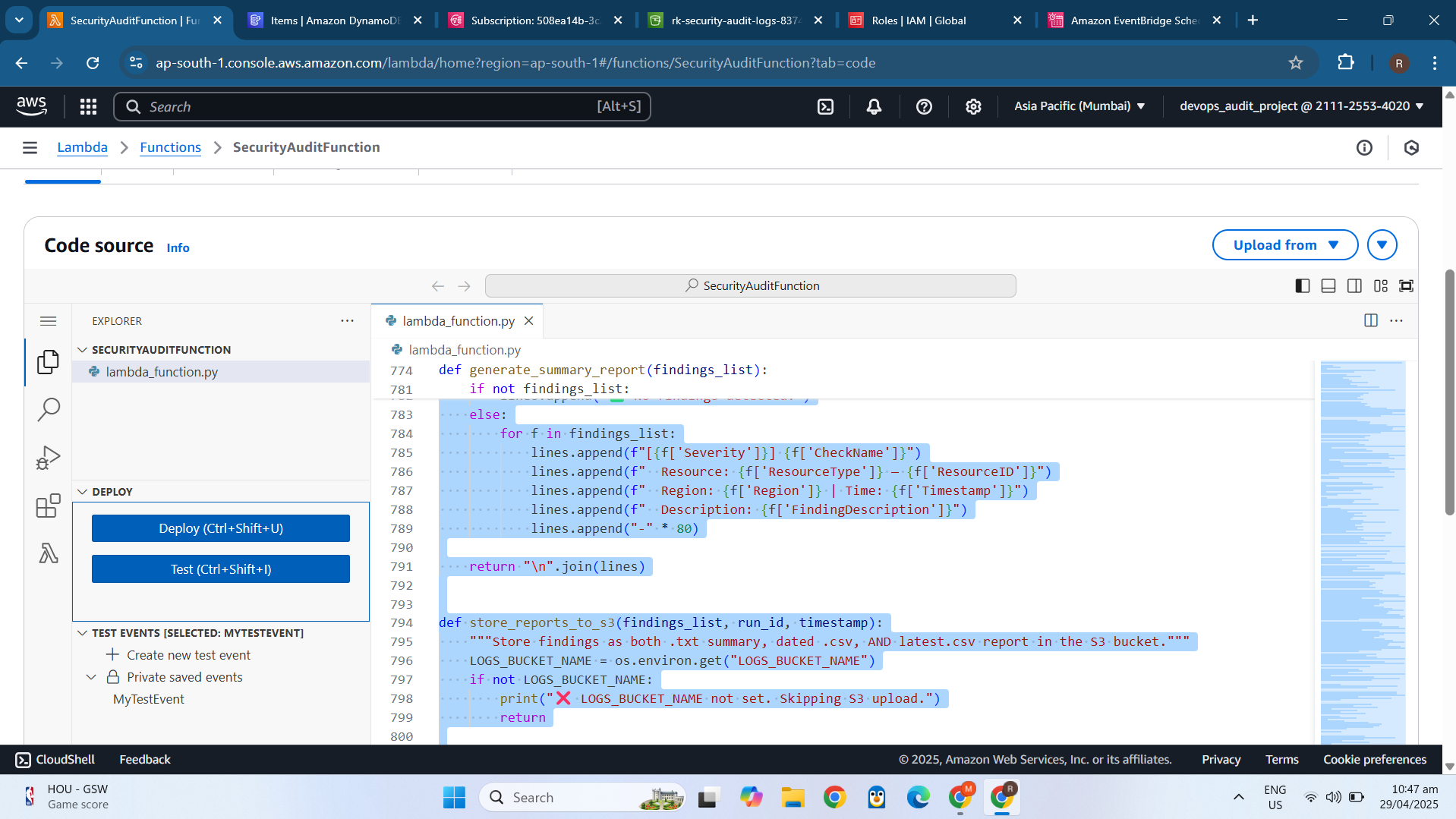
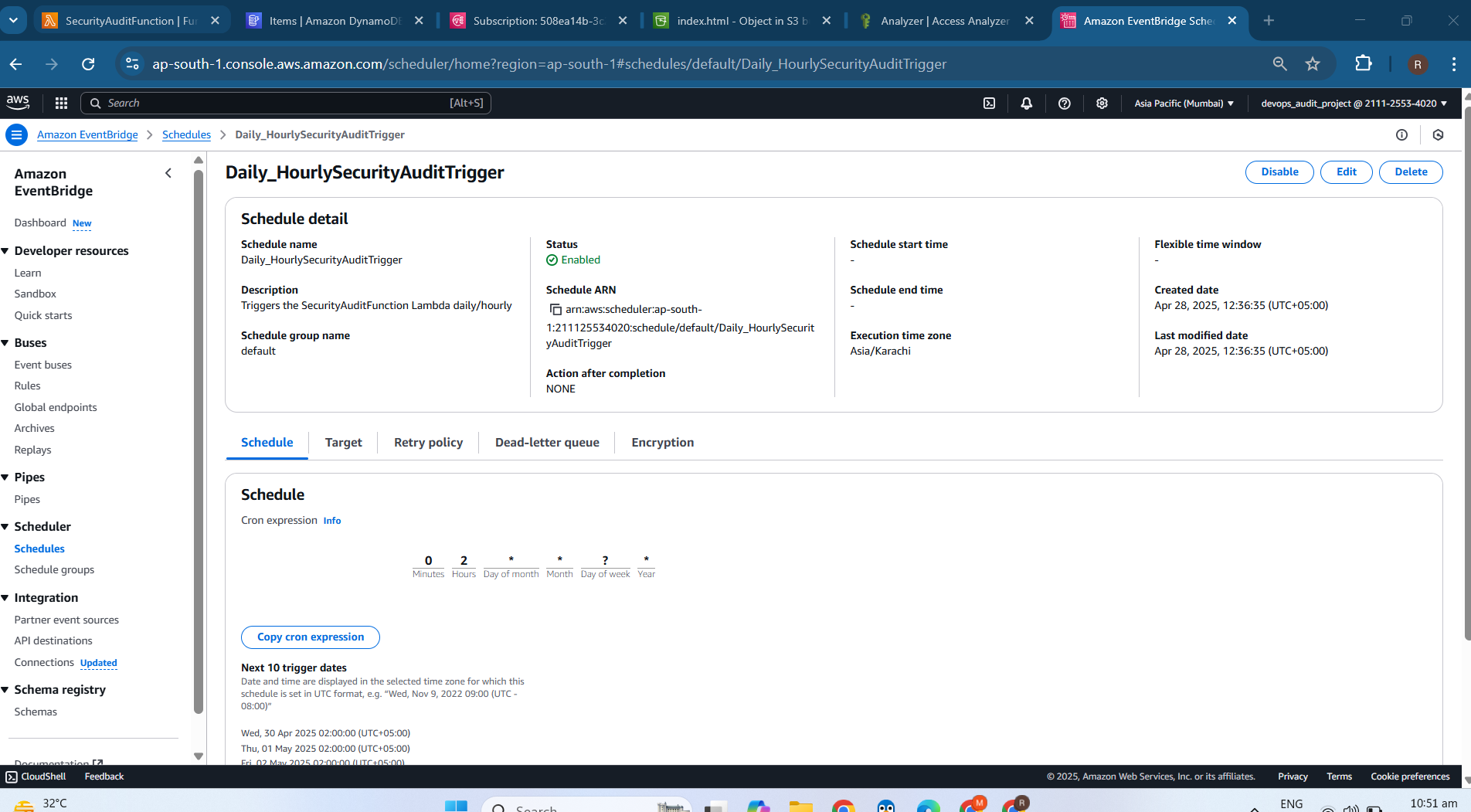
[**Placeholder for Lambda Code** - Include the complete Python code for your Lambda function here.]

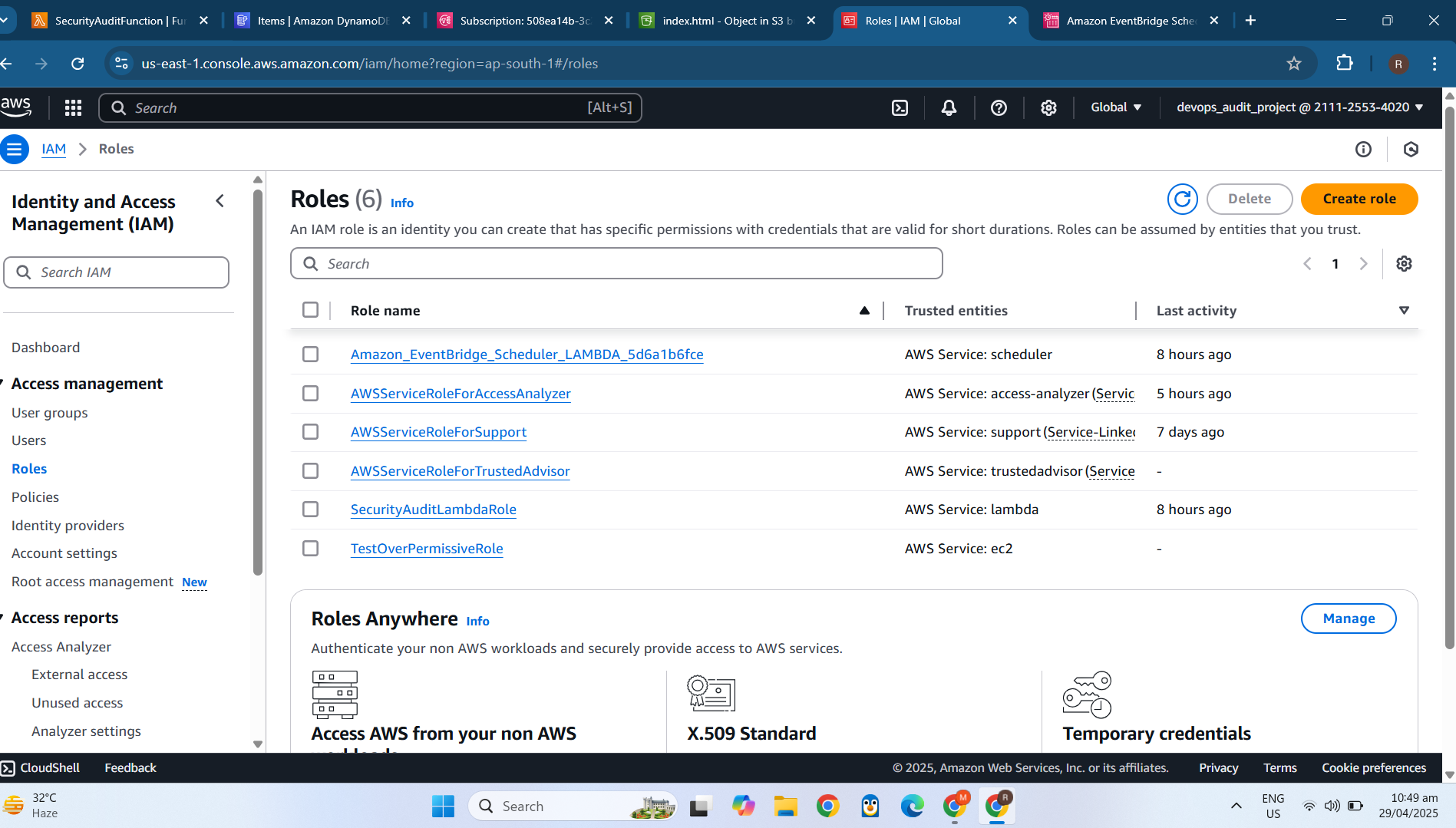
### 9.2 IAM Policy Document

**Placeholder for IAM Policy JSON** - Include the final JSON policy document for your Lambda execution role here, with your specific resource ARNs.]

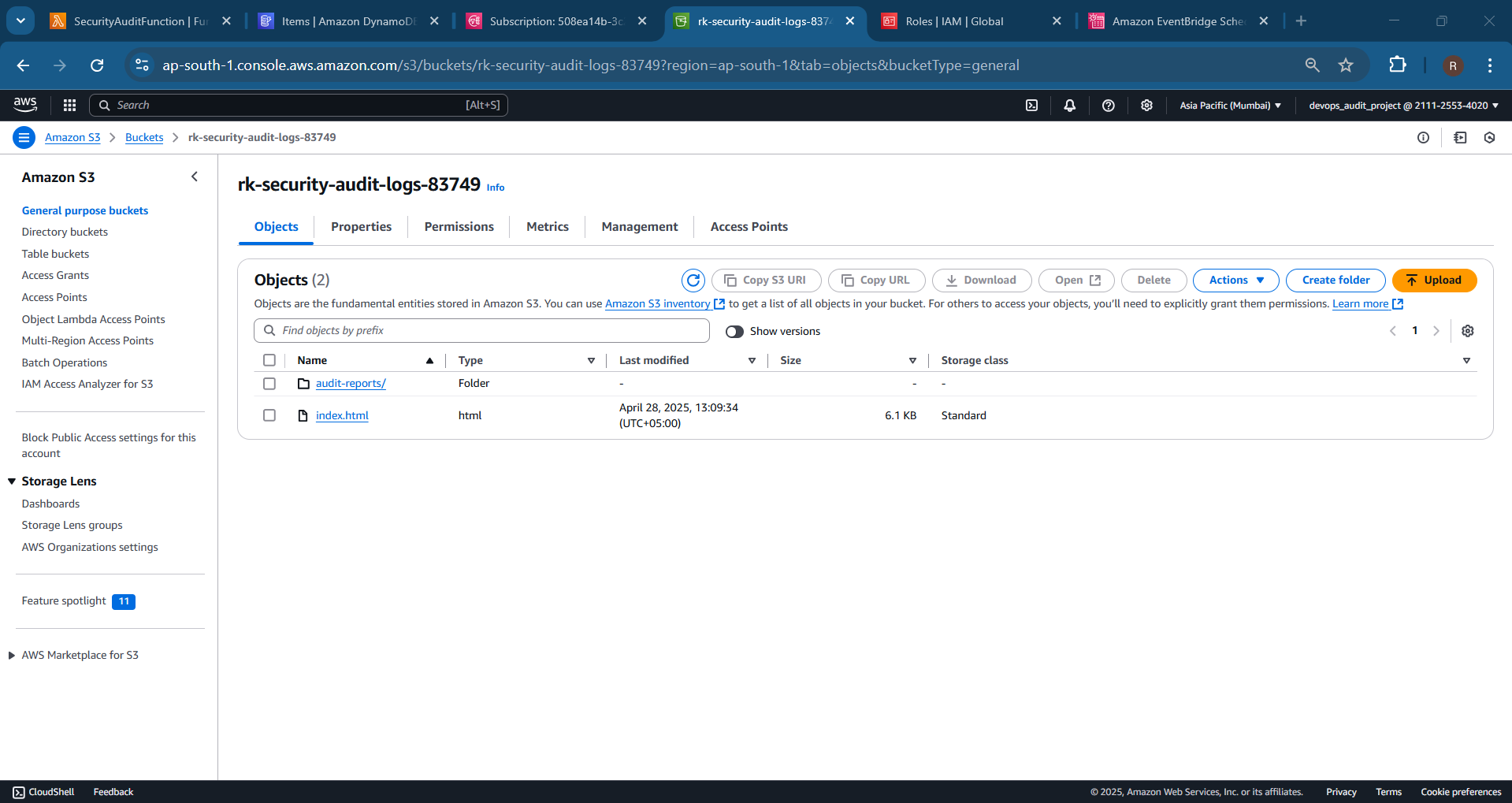
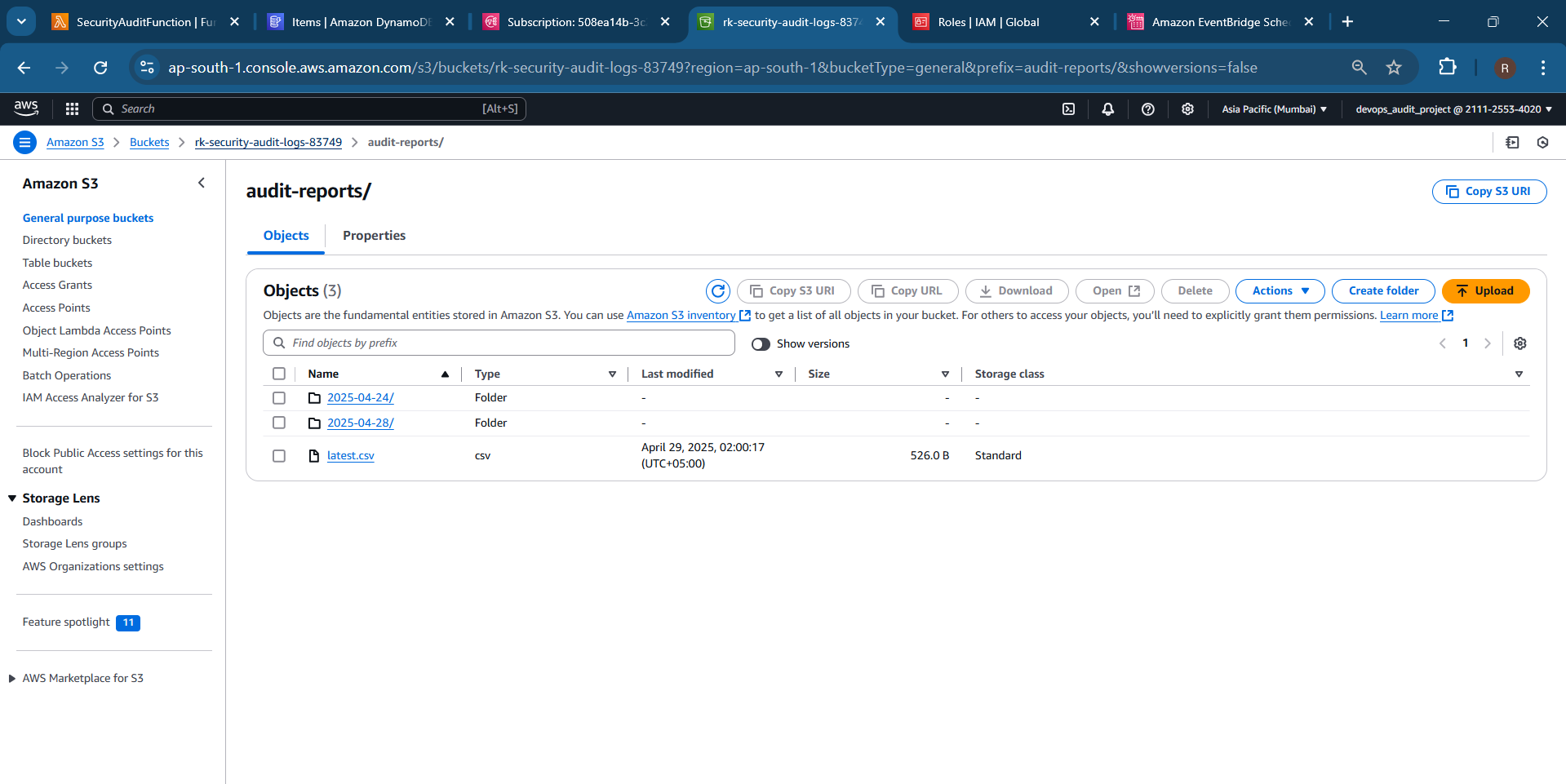
{  
 "Version": "2012-10-17",  
 "Statement": [  
 {  
 "Effect": "Allow",  
 "Action": [  
 "logs:CreateLogGroup",  
 "logs:CreateLogStream",  
 "logs:PutLogEvents"  
 ],  
 "Resource": "arn:aws:logs:\*:\*:\*"  
 },  
 {  
 "Effect": "Allow",  
 "Action": [  
 "s3:ListAllMyBuckets",  
 "s3:GetBucketLocation",  
 "s3:GetBucketPolicyStatus",  
 "s3:GetBucketAcl"  
 ],  
 "Resource": "\*"  
 },  
 {  
 "Effect": "Allow",  
 "Action": [  
 "s3:PutObject"  
 ],  
 "Resource": "arn:aws:s3:::rk-security-audit-logs-83749/audit-reports/\*"  
 },  
 {  
 "Effect": "Allow",  
 "Action": [  
 "ec2:DescribeSecurityGroups",  
 "ec2:DescribeInstances"  
 ],  
 "Resource": "\*"  
 },  
 {  
 "Effect": "Allow",  
 "Action": [  
 "iam:ListRoles",  
 "iam:ListAttachedRolePolicies",  
 "iam:ListRolePolicies",  
 "iam:GetPolicy",  
 "iam:GetPolicyVersion",  
 "iam:GetRolePolicy"  
 ],  
 "Resource": "\*"  
 },  
 {  
 "Effect": "Allow",  
 "Action": [  
 "access-analyzer:ListAnalyzers",  
 "access-analyzer:ListFindings"  
 ],  
 "Resource": "\*"  
 },  
 {  
 "Effect": "Allow",  
 "Action": [  
 "dynamodb:BatchWriteItem"  
 ],  
 "Resource": "arn:aws:dynamodb:ap-south-1:211125534020:table/SecurityFindings"  
 },  
 {  
 "Effect": "Allow",  
 "Action": [  
 "sns:Publish"  
 ],  
 "Resource": "arn:aws:sns:ap-south-1:211125534020:CriticalSecurityAlerts"  
 }  
 ]  
}

### 9.4 Screenshots

* Lambda function configuration.
* 
* EventBridge rule.
* DynamoDB table with findings.
* IAM Role Policy of AWS SECURITY LAMBDA



* S3 bucket with reports (.txt, .csv, latest.csv)/S3 hosted dashboard displaying findings.].

* SNS email alert received.

