AWS S3 Bucket Self-Healing Implementation

Table of Contents

List of Figures

1. Architecture Overview

This implementation provides an automated self-healing mechanism for S3 buckets that detects and reverts unauthorized public access settings using AWS Lambda, EventBridge, and CloudWatch.

1. S3 bucket (secured initially)

2. Lambda function to detect and revert unauthorized public access

3. EventBridge rule to trigger the Lambda when S3 bucket policies change

4. CloudWatch metrics and alarms to monitor system effectiveness

5. Test scenarios to validate the solution

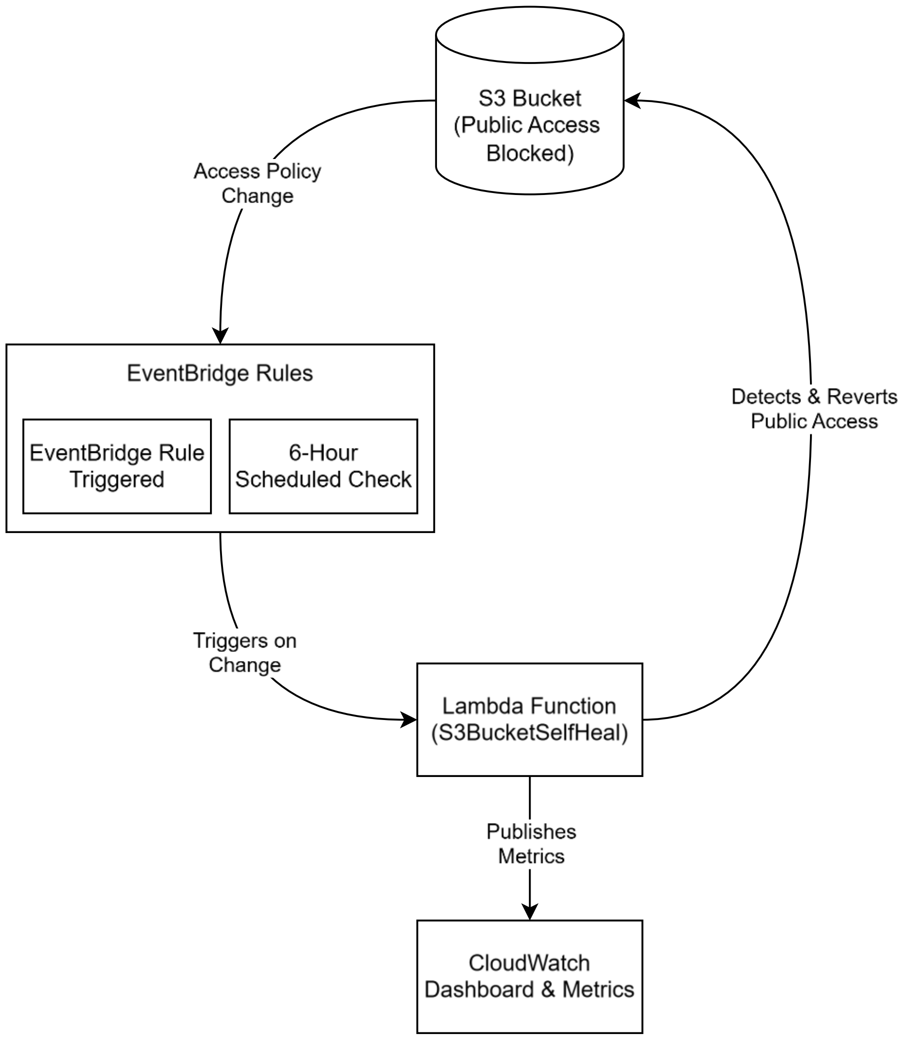
S3 Bucket (Public Access Blocked)
Access Policy Change
EventBridge Rules
EventBridge Rule Triggered
6-Hour Scheduled Check
Detects & Reverts Public Access
Triggers on Change
Lambda Function (S3BucketSelfHeal)
Publishes Metrics
CloudWatch Dashboard & Metrics

Figure 1: Architecture Diagram

2. Flow Diagram

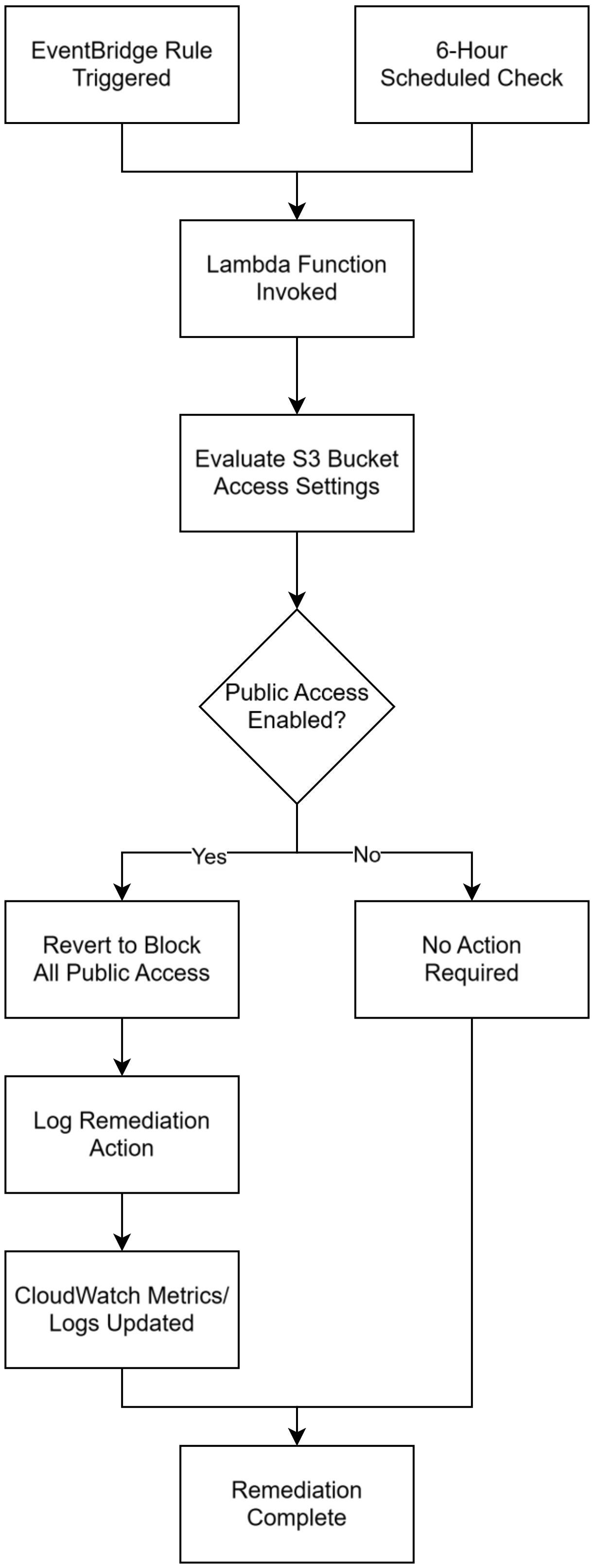

Figure 2: Flow Diagram for Self-Heal System

3. Implementation Steps

3.1 Create the S3 Bucket

1. Sign in to the AWS Management Console and navigate to S3
2. Click "Create bucket", enter a unique name, select region
3. Enable "Block all public access" settings and create the bucket
4. Verify under Permissions that Block all public access is ON

3.2 Create the IAM Role for Lambda

1. In IAM, create a new role with trusted entity "Lambda"
2. Attach "AmazonS3FullAccess" and "AWSLambdaBasicExecutionRole" policies
3. Name the role "S3BucketSelfHealRole" and create it

3.3 Create the Lambda Function

1. In Lambda, create function "Author from scratch", Python 3.9 runtime
2. Assign the existing "S3BucketSelfHealRole" role
3. Add the provided Python code to detect and revert public access settings
4. Set Timeout to 30 seconds and deploy the function

3.4 Create the EventBridge Rule

1. In EventBridge, create rule "S3BucketPublicAccessDetection" with custom event pattern for S3 policy changes
2. Set target to the "S3BucketSelfHeal" Lambda function
3. Review and create the rule

3.5 Set up Additional Scheduled Check

1. Create EventBridge rule "S3BucketPeriodicCheck" scheduled every 6 hours (cron 0 */6 * * ? *)
2. Configure same Lambda as target with constant JSON input

3.6 Set up CloudWatch Alarms and Dashboard

1. In CloudWatch, create dashboard "S3SecuritySelfHealDashboard" with widgets for RemediationActions, EvaluationsPerformed, RemediationResponseTime
2. Create alarm "S3BucketRemediationActionAlarm" on RemediationActions >= 1 (optional)

3.7 Create CloudTrail Trail

1. In CloudTrail, create a new trail named "SelfHealTrail"
2. Configure to log S3 Data Events and management events
3. Enable multi-region operation and specify a dedicated S3 bucket for logs

3.8 Monitoring and Metrics

Given the controlled, manual simulation environment, the most meaningful metric to track is the remediation response time. Other metrics such as invocation count are less relevant without a production workload.

- Primary Metric:
  Remediation Response Time: Time between detection and fix (ms)

4. Testing

1. Use the Python script `s3_remediation_test.py` to simulate public access toggles.
2. Launch the test: `python s3_remediation_test.py --bucket secure-self-healing-bucket --function S3BucketSelfHeal --iterations 50`.
3. Review the generated `s3_remediation_test.log` for detailed latency and response data.

5. Results and Discussion

Based on the simulation runs in `s3_remediation_test.log`, the following averages were observed:

- Client-side round-trip time (RTT) over 50 runs: 597.88 ms
- Lambda-reported response time over 50 runs: 426.26 ms

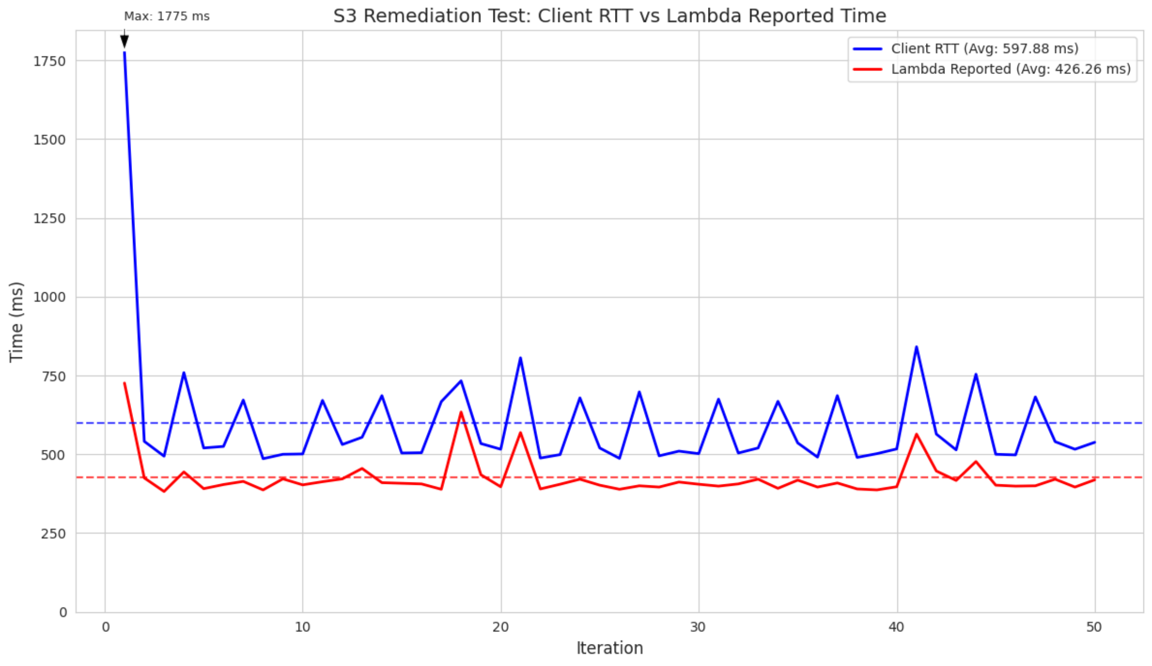

Figure 3: S3 Remediation Test: Client RTT vs Lambda Reported Time

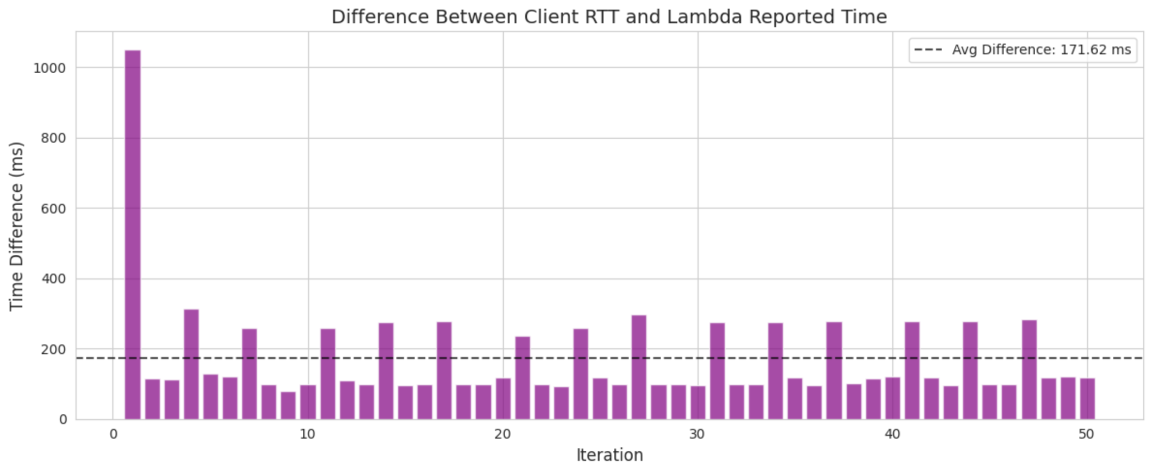

Figure 4: Difference Between Client RTT and Lambda Reported Time

These results demonstrate consistent remediation performance, with the client-side overhead averaging under 600 ms and the Lambda function processing taking approximately 426 ms on average.