AWS Inspector Dashboard

Contents

Executive Summary

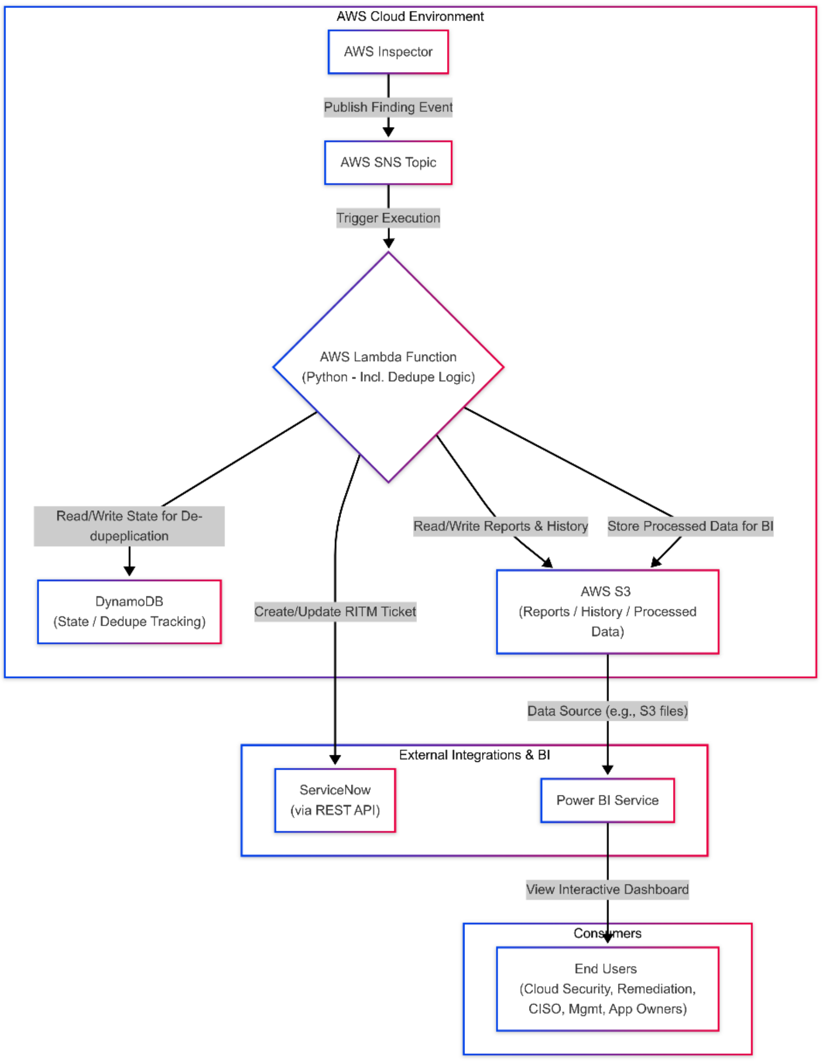
The AWS Inspector Dashboard provides an automated end-to-end solution for managing vulnerability findings from AWS Inspector at scale. Utilizing AWS Lambda triggered via SNS, the system processes weekly findings, intelligently deduplicates noise from superseded issues, and integrates directly with ServiceNow to assign remediation tasks. This automation eliminates the significant manual workload previously required for report compilation and stakeholder notification due to Inspector's volume and reporting constraints. A Power BI dashboard offers a centralized view for diverse stakeholders, including the Cloud Security team, Remediation teams, Application Owners, Team Managers, and the CISO, enabling effective tracking of the AWS security posture.

The Challenge: Managing AWS Inspector Findings Manually

Before the automated dashboard system, managing vulnerabilities identified by AWS Inspector across the organization's cloud environment involved a highly inefficient, labor-intensive, and largely unscalable manual process:

- **Cumbersome Data Extraction & Reporting Limits:** Findings were initially reviewed directly within the AWS Console. However, significant limitations in Inspector's native reporting capabilities (particularly when exporting data for more than 10 assets simultaneously) forced the Cloud Security team to perform **multiple, time-consuming manual data downloads** to extract the raw vulnerability information needed for reporting.
- **Laborious Report Compilation:** The downloaded, often fragmented, data required **extensive manual compilation and manipulation within Excel**. This involved using various formulas and significant effort simply to create a consolidated, usable report from the raw findings.
- **Manual ServiceNow Ticketing:** Once reports were compiled, actionable findings had to be **manually transferred into ServiceNow (SNOW) tickets**. This involved repetitive copy-pasting of details to assign tasks to the correct remediation teams or application owners, a process prone to errors and delays.
- **Overwhelming Volume & Tracking Impossibility:** The sheer scale, involving potentially "hundreds of thousands" of findings (compounded by persistent vulnerabilities from ongoing AWS migrations alongside new OS and package issues), made effective end-to-end tracking virtually impossible. As you noted, attempting to manage this **"ocean of vulnerabilities"** manually meant that comprehensive remediation status tracking was severely lacking or "next to impossible."
- **(Implied) Noise & Prioritization Difficulty:** Without automated deduplication of superseded findings (which the solution addresses), the manually compiled reports were inevitably cluttered, making it difficult for remediation teams to prioritize the most critical and currently relevant vulnerabilities effectively.
- **Unsustainable Workload & Growing Risk:** This entirely manual workflow resulted in a **constantly increasing backlog of reported vulnerabilities**. It placed an **enormous, unsustainable operational load** on both the Cloud Security team (responsible for data processing, reporting, and ticketing) and the various remediation teams tasked with fixation. This bottleneck hindered timely risk reduction for the organization.

High-Level Conceptual Architecture Diagram



AWS Cloud Environment
AWS Inspector
Publish Finding Event
AWS SNS Topic
Trigger Execution
AWS Lambda Function
(Python - Incl. Dedupe Logic)
Read/Write State for De-duplication
DynamoDB
(State / Dedupe Tracking)
Create/Update RITM Ticket
Read/Write Reports & History
Store Processed Data for BI
AWS S3
(Reports / History / Processed Data)
Data Source (e.g., S3 files)
External Integrations & BI
ServiceNow
(via REST API)
Power BI Service
View Interactive Dashboard
Consumers
End Users
(Cloud Security, Remediation, CISO, Mgmt, App Owners)

The Solution: Building AWS Inspector Dashboard

Conceptual Overview:

The core of the solution is a sophisticated, **automated workflow engine** (likely implemented as an AWS Lambda function triggered by security finding events via SNS, based on the project description). Its primary concept is to bridge AWS security findings (like Inspector, GuardDuty) with the organization's ServiceNow ticketing process efficiently and intelligently.

The system was designed to:

1. **Ingest findings** associated with various AWS accounts and internal services.
2. **Enrich findings** by dynamically querying ServiceNow to determine the correct IT support group responsible for the associated AWS account CI (Configuration Item).
3. **Automate ServiceNow Ticket (RITM) Creation**, assigning ownership correctly and attaching relevant evidence reports (CSVs from S3).
4. **Maintain state and prevent duplicates** using DynamoDB, ensuring findings aren't re-ticketed within a specific timeframe and managing potentially long-running, paginated processes across many accounts.
5. **Handle exceptions gracefully**, such as routing findings for accounts with missing ownership information to a default group via aggregate tickets.
6. Provide **specialized handling** for reporting overdue findings.

This serverless approach, interacting heavily with ServiceNow, DynamoDB, and S3 via modular helper functions, aimed to create a reliable, scalable, and intelligent automation layer between AWS security services and the enterprise ticketing system.

Key Features & Functionality:

The Python-based integration script provided several key capabilities :

- **Automated ServiceNow RITM Generation:** Automatically creates ServiceNow Service Catalog Requested Items (RITMs) for security findings grouped by AWS account.
- **Dynamic Support Group Assignment:** Intelligently assigns RITMs by querying ServiceNow in real-time (using the AWS Account's ServiceNow SysId) to find the correct support group owner.
- **Exception Handling for Missing Ownership:** If an AWS account lacks a defined support group in ServiceNow, findings are associated with an aggregate ticket assigned to a default group (e.g., Security Engineering) for investigation.

- **Automated Report Attachment:** Identifies and attaches relevant detailed finding reports (CSVs likely sourced from S3) directly to the corresponding RITM in ServiceNow.
- **Stateful Processing & Deduplication:** Leverages DynamoDB tables to track the processing state of findings for each account/service across multiple stages. This prevents the creation of duplicate ServiceNow tickets for the same findings within a defined time window (e.g., 8 hours).
- **Pagination Control / Skip Logic:** Includes logic to check the DynamoDB state and skip processing for accounts/services that have already been successfully processed recently, optimizing performance and preventing redundant actions.
- **Dedicated Overdue Finding Ticketing:** Contains specific logic to generate separate ServiceNow RITMs summarizing findings that have passed their remediation deadlines, attaching pre-generated overdue reports.
- **Configuration Driven:** Utilizes environment variables and AWS Secrets Manager for managing operational settings like DynamoDB table names, ServiceNow credentials, S3 bucket names, and default assignment groups.
- **Modular & Instrumented Design:** Built using distinct Python helper modules for interacting with ServiceNow, DynamoDB, S3, managing configuration, handling pagination logic, and structured logging, promoting maintainability and observability.

My Contribution & Development Process:

This was a large-scale project developed collaboratively by a team. My specific contributions focused on two critical areas: **implementing the vulnerability deduplication logic** within the Python backend and **developing the Power BI reports and dashboards** for visualization.

My key activities included:

- **Deduplication Logic Development (Python Backend):**
    - While the overall end-to-end architecture was designed by an architect, I was responsible for designing and writing the specific Python code module that handled the crucial task of identifying and filtering out superseded AWS Inspector findings.
    - This involved working within the AWS Lambda execution environment and likely utilizing libraries such as **Pandas** for data comparison and **Boto3** to interact with AWS services like **DynamoDB** or **S3** where state information or findings data might have been stored to perform the deduplication accurately.
- **ServiceNow Integration Understanding:** To ensure the deduplicated data integrated correctly downstream, I researched and familiarized myself with the relevant **ServiceNow REST API** documentation, specifically regarding authentication methods and the processes used by the team for creating RITM tickets.
- **Power BI Dashboard Development:** I was actively involved in the frontend aspect, specifically **creating the reports and interactive dashboards in Power BI**. This involved connecting to the processed data source (prepared by the backend system), designing visualizations, and implementing filtering logic to present the vulnerability data effectively to the various stakeholders (Cloud Security, Remediation teams, CISO, etc.).

Impact & Organizational Benefits

This was a large-scale project developed collaboratively by a team. My specific contributions focused on two critical areas: **implementing the vulnerability deduplication logic** within the Python backend and **developing the Power BI reports and dashboards** for visualization.

My key activities included:

- **Deduplication Logic Development (Python Backend):**
    - While the overall end-to-end architecture was designed by an architect, I was responsible for designing and writing the specific Python code module that handled the crucial task of identifying and filtering out superseded AWS Inspector findings.
    - This involved working within the AWS Lambda execution environment and likely utilizing libraries such as **Pandas** for data comparison and **Boto3** to interact with AWS services like **DynamoDB** or **S3** where state information or findings data might have been stored to perform the deduplication accurately.
- **ServiceNow Integration Understanding:** To ensure the ticket creation using APIs, I researched and familiarized myself with the relevant **ServiceNow REST API** documentation, specifically regarding authentication methods and the processes used by the team for creating RITM tickets.
- **Power BI Dashboard Development:** I was actively involved in the frontend aspect, specifically **creating the reports and interactive dashboards in Power BI**. This involved connecting to the processed data source (prepared by the backend system), designing visualizations, and implementing filtering logic to present the vulnerability data effectively to the various stakeholders (Cloud Security, Remediation teams, CISO, etc.).

Conclusion

In summary, this script is a sophisticated automation engine for integrating AWS security findings into a ServiceNow ticketing workflow. It intelligently groups findings by account, attempts to assign tickets to correct support groups via ServiceNow lookups, handles exceptions gracefully (missing support groups), attaches evidence, prevents duplicate ticketing using DynamoDB state tracking, and has provisions for handling overdue items. Its heavy reliance on helper modules and configuration makes it flexible but also complex, requiring all dependencies (helper code, AWS resources, ServiceNow config) to be correctly set up.