Table of Contents

Summary

Add new topics

Instructions

Custom Github Apps

Start typing here...

AWS

Start typing here...

Tenable

Start typing here...

Code

Start typing here...

main.gs

This script, written in JavaScript for Google Apps Script, automates the process of fetching vulnerability data from AWS Inspector and Tenable APIs, updating Google Sheets, and managing a Plan of Action and Milestones (POA&M) workbook. It integrates multiple libraries and performs tasks such as conditional formatting, workbook updates, and error logging.

Structure and Organization

- **Main Function**: `fetchDataFromAPIs`
  - Orchestrates the script by calling other functions for data fetching, processing, and updating.
- **Sub-functions**:
  1. **Data Fetching**:
     - `fetchAWSInspectorData`: Fetches vulnerability data from AWS Inspector.
     - `fetchTenableData`: Fetches vulnerability data from Tenable.
  2. **Data Processing and Updates**:
     - `updateGoogleSheet`: Updates a Google Sheet with normalized data.
     - `applyConditionalFormatting`: Highlights critical vulnerabilities in Google Sheets.
     - `updatePOAMWorkbook`: Updates the POA&M workbook in a shared drive.
  3. **Error Handling and Logging**:
     - `logError`: Logs errors and sends email notifications.
  4. **Automation**:
     - `createTrigger`: Sets up a time-based trigger for automation.

Key Components

1. **API Integration**:
    - Connects to AWS Inspector and Tenable APIs to fetch vulnerability data.
    - Validates JSON responses to ensure data integrity.
2. **Data Normalization and Visualization**:
    - Normalizes fetched data using helper libraries.
    - Applies conditional formatting in Google Sheets to highlight critical vulnerabilities.
3. **POA&M Workbook Management**:
    - Ensures the existence of folders and files in a shared drive.
    - Creates new sheets for vulnerability data in the workbook.
4. **Error Handling**:
    - Logs errors and notifies users via email for better issue tracking.
5. **Automation**:
    - Schedules the script to run daily using time-driven triggers.

Dependencies and Imports

- **Libraries**:
    - BatchRequest, FilesApp, GoogleApiApp, MicrosoftDocsApp, RunAll, RangeListApp, DateFinder, DocsServiceApp, TriggerApp, TemplateApp, MoveFolder, cUseful.
- **Google Apps Script Services**:
    - `UrlFetchApp`: For API requests.
    - `SpreadsheetApp`: For Google Sheets operations.
    - `GmailApp`: For email notifications.
    - `Session`: For accessing user email.
    - `Logger`: For logging actions and errors.

Input/Output

- **Input**:
  - Data from AWS Inspector and Tenable APIs.
  - Vulnerability data stored in Google Sheets.
- **Output**:
  - Updated Google Sheets and POA&M workbook with formatted data.
  - Email notifications for errors.

Error Handling

- Validates API responses to ensure valid JSON.
- Logs and notifies users of errors, including stack traces.
- Ensures required folders and files exist before proceeding.

Performance Considerations

- **Strengths**:
  - Modular design for better maintainability.
  - Efficient data handling using libraries like `cUseful`.
- **Weaknesses**:
  - Assumes the presence of library dependencies, which may cause issues if missing.
  - Potential performance bottlenecks for large datasets or API rate limits.

Code Style and Readability

- Well-structured with clear separation of concerns.

- Descriptive function and variable names improve readability.
- Could benefit from additional comments for complex operations.

Potential Issues

1. **Hardcoded Values**:
   - URLs, sheet names, and folder/file names are hardcoded, reducing adaptability.
2. **Error Handling**:
   - Limited validation for input data and API responses.
3. **Scalability**:
   - May face challenges with large datasets or multiple concurrent API requests.

Security Considerations

- **Strengths**:
  - API keys are dynamically fetched for enhanced security.
  - Sensitive data is not hardcoded in the script.
- **Weaknesses**:
  - Assumes secure storage and access to library dependencies.
  - Email notifications may expose error details.

Extensibility and Reusability

- **Extensibility**:
  - Could be extended to include additional data sources or processing logic.
  - Parameterizing inputs (e.g., URLs, folder names) would enhance flexibility.
- **Reusability**:

- Functions like `logError` and `updateGoogleSheet` can be reused in similar projects.

Suggestions for Improvement

1. Parameterize URLs, folder/file names, and sheet names to improve adaptability.
2. Add detailed comments for critical operations to enhance maintainability.
3. Implement pagination handling for large datasets fetched from APIs.
4. Validate API responses and input data more rigorously.
5. Use environment variables or secure storage for sensitive data like API keys.

moveResolvedVulnerabilities.gs

This script, written in JavaScript for Google Apps Script, automates the management of resolved vulnerabilities in a spreadsheet. It compares current vulnerabilities fetched from AWS Inspector and Tenable with the entries in the "Open POA&M Items" sheet. Resolved vulnerabilities are moved to the "Closed POA&M Items" sheet.

Structure and Organization

- **Main Function**: `moveResolvedVulnerabilities`
  - Retrieves current vulnerabilities from AWS Inspector and Tenable data.
  - Identifies resolved vulnerabilities by comparing them with "Open POA&M Items".
  - Moves resolved vulnerabilities to the "Closed POA&M Items" sheet and deletes them from "Open POA&M Items".

Key Components

1. **Spreadsheet Interaction**:
   - Accesses "Open POA&M Items" and "Closed POA&M Items" sheets in the active spreadsheet.
   - Retrieves data from specific ranges based on headers and row indices.
2. **Vulnerability Comparison**:
   - Fetches current vulnerabilities from AWS Inspector and Tenable APIs.
   - Combines and compares vulnerabilities using `Weakness Name` and `Asset Identifier`.
3. **Data Movement**:
   - Identifies resolved vulnerabilities and appends them to the "Closed POA&M Items" sheet.
   - Deletes resolved entries from the "Open POA&M Items" sheet.

Logic and Workflow

1. **Data Retrieval**:
   - Fetches current vulnerabilities from `fetchAWSInspectorData` and `fetchTenableData`.
   - Combines results into a set for efficient comparison.
2. **Resolved Vulnerabilities Identification**:
   - Iterates over rows in the "Open POA&M Items" sheet.
   - Checks if vulnerabilities no longer exist in the current scan results.
3. **Data Movement**:
   - Appends resolved vulnerabilities to the "Closed POA&M Items" sheet.
   - Deletes resolved vulnerabilities from the "Open POA&M Items" sheet.

Dependencies and Imports

- **Google Apps Script Services**:
  - `SpreadsheetApp`: For accessing and manipulating the spreadsheet.
  - `Logger`: For logging actions and results.
- **External Functions**:
  - `fetchAWSInspectorData`: Fetches vulnerability data from AWS Inspector.
  - `fetchTenableData`: Fetches vulnerability data from Tenable.

Input/Output

- **Input**:
  - Vulnerability data from AWS Inspector and Tenable.
  - Data from the "Open POA&M Items" sheet.

- **Output**:
  - Resolved vulnerabilities moved to the "Closed POA&M Items" sheet.
  - Logs the number of resolved vulnerabilities.

Error Handling

- Throws an error if the required sheets ("Open POA&M Items" or "Closed POA&M Items") are missing.
- Assumes data from external functions (`fetchAWSInspectorData` and `fetchTenableData`) is correctly formatted.

Performance Considerations

- **Strengths**:
  - Efficiently identifies resolved vulnerabilities using a set for comparisons.
  - Handles data movement in bulk to minimize operations.
- **Weaknesses**:
  - Assumes the data in "Open POA&M Items" is consistently formatted.
  - Does not handle large datasets with pagination for API responses.

Code Style and Readability

- Code is modular and follows a clear logical structure.
- Descriptive variable names improve readability.
- Additional comments explaining key operations would enhance maintainability.

Potential Issues

1. **Hardcoded Column Indices**:

- Assumes specific columns for `Weakness Name` and `Asset Identifier`, which may lead to issues if the sheet structure changes.

2. **Error Handling**:
   - Limited error handling for data inconsistencies or unexpected API responses.
3. **Deletion Order**:
   - Deletes rows in reverse order, which may cause issues if other processes are dependent on row indices.

Security Considerations

- Relies on external functions (`fetchAWSInspectorData` and `fetchTenableData`) for data integrity.
- Assumes that API responses and spreadsheet data are free of malicious inputs.

Extensibility and Reusability

- **Extensibility**:
  - Could be extended to handle additional data sources or new vulnerability attributes.
  - Parameterizing column indices would improve adaptability.
- **Reusability**:
  - Functions for data comparison and movement can be reused in other spreadsheet management tasks.

Suggestions for Improvement

1. Parameterize column indices to avoid hardcoding and improve adaptability to sheet structure changes.
2. Add detailed comments for key operations to improve code clarity.
3. Enhance error handling for API responses and data inconsistencies.

4. Implement pagination handling for large datasets fetched from APIs.
5. Use soft deletion (e.g., marking resolved items) instead of direct deletion for better traceability.

fetchTenableData.gs

⚠ This script, written in JavaScript for Google Apps Script, integrates with Tenable's API to fetch and process data related to assets and vulnerabilities. It supports dynamic API key assignment via an HTTP `POST` request or GitHub Secrets retrieval. The processed data is structured to associate vulnerabilities with their corresponding assets.

Structure and Organization

- **Main Functions**:
  1. **`doPost(e)`**:
     - Handles incoming HTTP `POST` requests to set the Tenable API key dynamically.
  2. **`fetchTenableData`**:
     - Retrieves assets and vulnerabilities from Tenable's API, processes the data, and maps vulnerabilities to assets.
- **Helper Functions**:
  1. **`fetchTenableApiKeyFromGitHub`**:
     - Assumes a function that retrieves the API key securely from GitHub Secrets.

Key Components

1. **Tenable API Integration**:
   - Fetches data from the `/assets` and `/vulnerabilities/export` endpoints to retrieve asset and vulnerability information.
2. **Dynamic API Key Handling**:
   - API key is set either via an HTTP `POST` request or securely fetched from GitHub

Secrets.

3. **Data Processing**:
   - Processes vulnerabilities to associate them with corresponding assets using attributes like hostnames or IP addresses.

Logic and Workflow

1. **API Key Setup**:
   - API key is dynamically set via `doPost` or retrieved from GitHub Secrets.
   - Ensures the API key is available before proceeding with Tenable API requests.
2. **Data Retrieval and Processing**:
   - Fetches assets and vulnerabilities from Tenable's API.
   - Maps vulnerabilities to their associated assets for better contextual insights.
3. **Error Handling**:
   - Logs errors for API key retrieval and data fetching operations.
   - Throws descriptive errors for missing API keys or failed API requests.

Dependencies and Imports

- **Google Apps Script Services**:
  - `UrlFetchApp`: For making HTTP requests to Tenable's API.
  - `Utilities`: Assumed for cryptographic operations if used elsewhere.
  - `Logger`: For debugging and logging.
  - `ContentService`: For HTTP responses.

Input/Output

- **Input**:
  - API key (`TENABLE_API_KEY`) via HTTP `POST` or GitHub Secrets.
  - Tenable API endpoints for assets and vulnerabilities.
- **Output**:
  - Processed data associating vulnerabilities with their corresponding assets.
  - Key attributes include weakness name, CVE, asset identifiers, severity, description, discovery date, and source.

Error Handling

- Validates the presence of an API key before making API requests.
- Handles and logs errors during API requests and data processing.
- Provides descriptive error messages for troubleshooting.

Performance Considerations

- **Strengths**:
  - Efficiently processes data using mapping and filtering operations.
  - Supports dynamic API key management to enhance security.
- **Weaknesses**:
  - Assumes that asset and vulnerability data can be fetched without pagination handling.

Code Style and Readability

- Code is modular with clear separation of responsibilities.
- Descriptive variable names improve readability.
- Additional comments explaining key operations would enhance maintainability.

Potential Issues

1. **Pagination**:
   - The script does not handle paginated responses, which may lead to incomplete data retrieval for large datasets.
2. **API Key Management**:
   - Dynamic key assignment via HTTP `POST` lacks validation for the request source, which may pose security risks.
3. **Error Handling**:
   - Errors in data processing (e.g., missing attributes) might not be adequately logged or handled.

Security Considerations

- **Strengths**:
  - Supports secure API key retrieval from GitHub Secrets.
  - Dynamic key management avoids hardcoding sensitive information.
- **Weaknesses**:
  - HTTP `POST` method for key assignment requires secure transmission (e.g., HTTPS).
  - Assumes API keys retrieved from GitHub are valid and correctly configured.

Extensibility and Reusability

- **Extensibility**:
  - Can be extended to include additional Tenable API endpoints or data processing logic.
  - Parameterizing API endpoints and query parameters would improve adaptability.

- **Reusability**:
  - Functions like `fetchTenableApiKeyFromGitHub` and data processing logic can be reused in other Tenable integrations.

Suggestions for Improvement

1. Implement pagination handling for Tenable API responses to ensure complete data retrieval.
2. Validate incoming `POST` requests to ensure API key integrity and prevent misuse.
3. Add detailed comments to explain critical operations like data processing and API interactions.
4. Log specific errors during data processing to enhance troubleshooting capabilities.
5. Parameterize API endpoints and attributes for improved flexibility and reuse.

fetchTenableApiKeyFromGitHub.gs

This script is written in JavaScript for Google Apps Script. It fetches the `TENABLE_API_KEY` secret from a GitHub repository by using a GitHub App's JWT authentication. The script retrieves and uses a private key stored in the Script Properties for generating the JWT and acquiring an installation access token.

Structure and Organization

- **Main Function**: `fetchTenableApiKeyFromGitHub`
  - Responsible for orchestrating the process of fetching the `TENABLE_API_KEY` from a GitHub repository.
  - It uses helper functions to generate a JWT and fetch the installation access token before retrieving the secret.
- **Helper Function 1**: `generateGitHubJwt`
  - Generates a JWT for GitHub App authentication using the provided `appId` and `privateKey`.
  - Uses `RS256` algorithm for signing.
- **Helper Function 2**: `getInstallationAccessToken`
  - Exchanges the JWT for an installation access token using GitHub's REST API.

Key Components

1. **Script Properties**:
   - Retrieves the private key from the Google Apps Script Properties (`GITHUB_APP_TENABLE`).
   - Ensures the private key exists before proceeding.
2. **JWT Creation**:

- Includes `iat` (issued at) and `exp` (expiration) claims, ensuring the token is valid for 10 minutes.

3. **Access Token Retrieval**:
   - Makes a `POST` request to GitHub's `/access_tokens` endpoint to get the installation access token.
4. **Secret Retrieval**:
   - Uses a `GET` request to fetch the `TENABLE_API_KEY` from the repository's secrets endpoint.
5. **Error Handling**:
   - Validates the presence of the private key, secret, and handles API response errors.

Logic and Workflow

1. Fetches the private key from the Script Properties.
2. Generates a JWT using the private key and app ID.
3. Obtains an installation access token using the JWT and installation ID.
4. Fetches the `TENABLE_API_KEY` secret from the GitHub repository.
5. Validates the presence of the secret and logs success or error messages.

Dependencies and Imports

- `PropertiesService`: To retrieve script properties (e.g., the private key).
- `UrlFetchApp`: For making HTTP requests to GitHub's API.
- `Utilities`: For cryptographic and encoding utilities (e.g., RSA-SHA256 signature, base64 encoding).

Input/Output

- **Input**:
  - Private key: Retrieved from the script properties (`GITHUB_APP_TENABLE`).
  - GitHub API interaction details: App ID, installation ID, repository owner, and repository name.
- **Output**:
  - Returns the `TENABLE_API_KEY` if successfully fetched.
  - Logs success or error messages using `Logger`.

Error Handling

- Validates the presence of the private key in script properties.
- Handles missing or improperly configured secrets by throwing specific errors.
- Catches and logs errors during API calls.

Performance Considerations

- The use of JWTs with short expiration times (10 minutes) enhances security.
- Efficient use of HTTP requests to interact with GitHub API endpoints.

Code Style and Readability

- Code is well-organized into distinct functions, improving readability and maintainability.
- Error messages are clear and provide actionable feedback.
- Could benefit from additional comments explaining specific logic.

Potential Issues

- If the private key in `Script Properties` is misconfigured, the script will fail.

- API rate limits from GitHub might impact functionality if the script is invoked frequently.

Security Considerations

- **Strengths**:
  - Uses short-lived JWTs for authentication.
  - Access token is scoped to a specific installation and repository.
- **Weaknesses**:
  - Private key handling relies on script properties, which must be securely configured.

Extensibility and Reusability

- **Extensibility**:
  - Could be extended to fetch multiple secrets by parameterizing the secret name.
- **Reusability**:
  - Functions like `generateGitHubJwt` and `getInstallationAccessToken` can be reused in other scripts interacting with GitHub Apps.

Suggestions for Improvement

1. Add retry logic for handling transient API failures.
2. Include detailed comments for complex sections like JWT generation.
3. Use environment variables for better security practices if supported.

fetchAwsCredentialsFromGitHub.gs

This script, written in JavaScript for Google Apps Script, integrates with AWS and GitHub to retrieve AWS credentials stored as GitHub Secrets. It uses these credentials to make authenticated requests to AWS services. The script ensures secure integration by leveraging a GitHub App with JWT authentication.

Structure and Organization

- **Function**: `fetchAwsCredentialsFromGitHub`
  - Main function to retrieve AWS credentials (`Access Key`, `Secret Key`) from GitHub Secrets.
  - Sets the `AWS_REGION` explicitly to `us-east-1`.
- **Helper Functions**:
  1. **`fetchGitHubSecret`**:
     - Fetches specific secrets from a GitHub repository using a provided access token.
  2. **`generateGitHubJwt`**:
     - Generates a JWT for authenticating the GitHub App using RS256 signing.
  3. **`getInstallationAccessToken`**:
     - Retrieves an installation access token for the GitHub App.
  4. **`fetchAWSData`**:
     - Uses the retrieved AWS credentials to send an authenticated request to AWS.
  5. **`getCurrentDate`**:
     - Generates the current date in AWS-compatible format for signing requests.

Key Components

1. **GitHub Secrets Retrieval**:
    - Connects to GitHub's REST API to fetch AWS credentials stored as repository secrets.
2. **JWT Authentication**:
    - Uses a private key to sign and generate a short-lived JWT for secure GitHub App authentication.
3. **AWS Integration**:
    - Constructs AWS-compatible signed headers and sends an authenticated request to AWS services.

Logic and Workflow

1. Retrieves a private key from Google Apps Script Properties.
2. Generates a JWT using the private key and app ID.
3. Acquires an installation access token via GitHub's API.
4. Fetches AWS credentials (`Access Key`, `Secret Key`) stored as GitHub Secrets.
5. Uses the retrieved credentials to make a signed request to an AWS endpoint.

Dependencies and Imports

- **Google Apps Script Services**:
    - `PropertiesService`: To store and retrieve sensitive keys (e.g., private key).
    - `UrlFetchApp`: For making HTTP requests to GitHub and AWS APIs.
    - `Logger`: To log actions and responses for debugging.
    - `Utilities`: To handle cryptographic and encoding utilities.

Input/Output

- **Input**:
  - GitHub Secrets (`DEVOPS_DOOP_AUTOMATION_AWS_ACCESS_KEY_ID` and `DEVOPS_DOOP_AUTOMATION_AWS_SECRET_ACCESS_KEY`).
  - AWS Region: Hardcoded to `us-east-1`.
- **Output**:
  - Authenticated request sent to AWS.
  - Logs response data from AWS.

Error Handling

- Throws errors if the private key is not found in Script Properties.
- Uses default values (`N/A`) if secrets are unavailable or improperly configured.
- Relies on GitHub and AWS API responses for error detection.

Performance Considerations

- **Strengths**:
  - Short-lived JWTs improve security and reduce exposure risk.
  - Efficiently retrieves secrets and credentials in a streamlined process.
- **Weaknesses**:
  - Hardcoded region limits flexibility for multi-region setups.

Code Style and Readability

- Code is modular, with functions clearly separated by responsibility.
- Descriptive variable names improve readability.
- Could benefit from additional comments explaining key steps.

Potential Issues

1. Hardcoded AWS Region (`us-east-1`) limits adaptability for other regions.
2. Missing validation for GitHub Secrets content may cause runtime errors.
3. Relies on a specific structure for Script Properties and GitHub Secrets.

Security Considerations

- **Strengths**:
  - Uses secure JWT-based authentication for GitHub.
  - AWS credentials are retrieved dynamically and not hardcoded.
- **Weaknesses**:
  - If the private key is compromised, the entire workflow could be at risk.
  - Secrets are assumed to be correctly configured in GitHub.

Extensibility and Reusability

- **Extensibility**:
  - Could be extended to handle additional AWS operations by reusing the retrieved credentials.
  - Parameterizing the AWS region improves adaptability.
- **Reusability**:
  - Functions like `generateGitHubJwt` and `fetchGitHubSecret` can be reused for other GitHub App integrations.

Suggestions for Improvement

1. Parameterize the AWS region to enhance flexibility for multi-region setups.

2. Add validation to ensure that fetched secrets are in the correct format.

3. Implement logging for potential GitHub and AWS API errors.

4. Securely store and handle sensitive data to prevent exposure risks.

closingOpenPOAMItems.gs

⚠ This script, written in JavaScript for Google Apps Script, automates the management of POA&M (Plan of Action and Milestones) items in a spreadsheet. Specifically, it moves rows marked as "False Positive" (`Yes`) from a sheet named "Open POA&M Items" to another sheet called "Closed POA&M Items" and deletes the original row from the source sheet.

Structure and Organization

- **Function**: `openPOAMItems(e)`
  - Triggered by an edit event in the spreadsheet.
  - Handles row transfer between two sheets based on the value in the "False Positive" column.
- **Event Object**: `e`
  - Captures the edit event details, such as the edited range and sheet.

Key Components

1. **Spreadsheet Interaction**:
   - Accesses the active spreadsheet and retrieves references to the "Open POA&M Items" and "Closed POA&M Items" sheets.
   - Dynamically identifies the "False Positive" column by scanning the header row.
2. **Event Handling**:
   - Detects if the edit occurred in the "False Positive" column of the "Open POA&M Items" sheet.
   - Moves the row to the "Closed POA&M Items" sheet if the edited value is `Yes`.

3. **Row Manipulation**:
    - Appends the row to the target sheet (`Closed POA&M Items`).
    - Deletes the corresponding row from the source sheet (`Open POA&M Items`).

Logic and Workflow

1. Captures the edit event and identifies the sheet and cell edited.
2. Checks if the edit is in the "False Positive" column of the "Open POA&M Items" sheet.
3. If the value is `Yes`, retrieves the row data, appends it to the "Closed POA&M Items" sheet, and deletes the original row.

Dependencies and Imports

- **Google Apps Script Services**:
    - `SpreadsheetApp`: To access and manipulate the active spreadsheet.
    - `Logger`: To log actions for debugging.

Input/Output

- **Input**:
    - User edits in the "Open POA&M Items" sheet, specifically in the "False Positive" column.
- **Output**:
    - Moves the edited row to the "Closed POA&M Items" sheet if marked `Yes`.
    - Logs actions such as row transfer.

Error Handling

- The script assumes the presence of the "False Positive" column and corresponding sheets.
- It does not handle scenarios where these elements are missing, which may lead to runtime errors.

Performance Considerations

- **Strengths**:
  - Dynamically locates the "False Positive" column, making it resilient to column reordering.
  - Efficiently processes the row transfer in real time.
- **Weaknesses**:
  - Deletes rows directly, which might disrupt references or formulas in the sheet.

Code Style and Readability

- Code is well-structured and uses descriptive variable names.
- Additional comments explaining the workflow would improve maintainability.

Potential Issues

1. Assumes the presence of "Open POA&M Items" and "Closed POA&M Items" sheets.
2. Direct row deletion can lead to unintended consequences if other processes depend on row indices.

Security Considerations

- Relies on edit events, which might be exploited if malicious or erroneous edits are made.
- No validation is performed on the data before appending to the target sheet.

Extensibility and Reusability

- **Extensibility**:
  - Could be extended to handle other columns or automate further actions based on different criteria.
- **Reusability**:
  - Limited to spreadsheets with a specific structure. Generalizing the column and sheet names would improve reusability.

Suggestions for Improvement

1. Add validation to ensure that required sheets and columns are present before execution.
2. Implement logging for errors or missing prerequisites to improve debugging.
3. Use soft deletion (e.g., marking rows as inactive) instead of direct deletion for better traceability.
4. Provide configuration options for sheet names and column headers to enhance adaptability.

fetchAWSInspectorData.gs

⚠ This script, written in JavaScript for Google Apps Script, integrates with AWS Inspector to fetch findings data. It uses AWS credentials that can either be passed via an HTTP `POST` request or fetched from GitHub Secrets. The script generates AWS signature headers for authentication and retrieves vulnerability findings from AWS Inspector.

Structure and Organization

- **Main Functions**:
  1. **`doPost(e)`**:
     - Handles incoming HTTP `POST` requests to set AWS credentials and region dynamically.
  2. **`fetchAWSInspectorData`**:
     - Fetches findings from AWS Inspector using the configured AWS credentials.
  3. **`generateAwsHeaders`**:
     - Generates AWS signature headers for making authenticated API requests.
  4. **`getSignatureKey`**:
     - Computes the signature key for AWS signature generation.

Key Components

1. **AWS Credentials Handling**:
   - Credentials (`AWS_ACCESS_KEY`, `AWS_SECRET_KEY`, `AWS_REGION`) can be dynamically set via HTTP `POST` or fetched from GitHub Secrets.
2. **AWS Inspector Integration**:

- Fetches findings using the Inspector API's `/findings/list` endpoint.
- Processes and returns a structured summary of findings.

3. **AWS Signature Version 4**:
   - Implements AWS SigV4 signing for secure API authentication.

Logic and Workflow

1. **AWS Credentials Setup**:
   - Credentials can be provided via HTTP `POST` (handled by `doPost`) or fetched using `fetchAwsCredentialsFromGitHub`.
2. **AWS Inspector Data Retrieval**:
   - Constructs a payload and sends an authenticated `POST` request to AWS Inspector's API endpoint.
   - Processes the findings and maps them into a structured output format.
3. **AWS SigV4 Signing**:
   - Uses the `generateAwsHeaders` and `getSignatureKey` functions to create the necessary headers for authenticated requests.

Dependencies and Imports

- **Google Apps Script Services**:
  - `UrlFetchApp`: For making HTTP requests to AWS APIs.
  - `Utilities`: For cryptographic operations (e.g., HMAC signatures, SHA-256 hashing).
  - `ContentService`: For HTTP responses.
  - `Logger`: For debugging and logging.

Input/Output

- **Input**:
  - AWS credentials (`AWS_ACCESS_KEY`, `AWS_SECRET_KEY`, `AWS_REGION`) via HTTP `POST` or GitHub Secrets.
  - Vulnerability data request payload.
- **Output**:
  - Processed findings from AWS Inspector, including key details like weakness name, CVE, asset identifiers, severity, description, and discovery date.

Error Handling

- Validates the presence of AWS credentials before making API requests.
- Handles and logs errors during API requests to AWS Inspector.
- Throws detailed error messages for missing credentials or network issues.

Performance Considerations

- **Strengths**:
  - Uses efficient AWS SigV4 signing for secure authentication.
  - Processes up to 100 findings per request (configurable).
- **Weaknesses**:
  - Hardcoded API limits may need adjustments for larger datasets.
  - Potential latency due to real-time signing and API calls.

Code Style and Readability

- Code is modular and follows a clear structure with descriptive function and variable names.

- Comments explaining complex operations (e.g., SigV4 signing) would improve readability.

Potential Issues

1. **Credential Management**:
   - Relies on dynamic credential handling via HTTP `POST` or GitHub Secrets, which might be prone to misconfiguration.
2. **Error Propagation**:
   - Errors in AWS signature generation or API responses might not provide enough context for troubleshooting.

Security Considerations

- **Strengths**:
  - Implements AWS SigV4 signing for secure API authentication.
  - Credentials are dynamically set and not hardcoded in the script.
- **Weaknesses**:
  - Sensitive credentials passed via HTTP `POST` need secure transmission (e.g., HTTPS).
  - Does not validate the source of incoming `POST` requests.

Extensibility and Reusability

- **Extensibility**:
  - Can be extended to handle other AWS services using similar signature headers.
  - Parameterizing payloads would allow more flexible data retrieval.
- **Reusability**:

- Functions like `generateAwsHeaders` and `getSignatureKey` can be reused for other AWS integrations.

Suggestions for Improvement

1. Add validation for incoming HTTP `POST` requests to ensure only authorized sources can set credentials.
2. Enhance error logging for better debugging and troubleshooting.
3. Include comments to explain key operations, especially AWS SigV4 signing.
4. Support pagination for fetching large datasets from AWS Inspector.
5. Securely store and retrieve AWS credentials to minimize exposure risks.

closingConfigurationItems.gs

> ⚠ This script, written in JavaScript for Google Apps Script, automates the management of findings in a spreadsheet. Specifically, it moves rows marked as "False Positive" (`Yes`) from a sheet named "Configuration Findings" to another sheet called "Closed POA&M Items" and deletes the original row from the source sheet.

Structure and Organization

- **Function**: `configurationFindings(e)`
  - Triggered by an edit event in the spreadsheet.
  - Handles row transfer between two sheets based on the value in the "False Positive" column.
- **Event Object**: `e`
  - Captures the edit event details, such as the edited range and sheet.

Key Components

1. **Spreadsheet Interaction**:
   - Accesses the active spreadsheet and retrieves references to the "Configuration Findings" and "Closed POA&M Items" sheets.
   - Dynamically identifies the "False Positive" column by scanning the header row.
2. **Event Handling**:
   - Detects if the edit occurred in the "False Positive" column of the "Configuration Findings" sheet.
   - Moves the row to the "Closed POA&M Items" sheet if the edited value is `Yes`.

3. **Row Manipulation**:
    - Appends the row to the target sheet (`Closed POA&M Items`).
    - Deletes the corresponding row from the source sheet (`Configuration Findings`).

Logic and Workflow

1. Captures the edit event and identifies the sheet and cell edited.
2. Checks if the edit is in the "False Positive" column of the "Configuration Findings" sheet.
3. If the value is `Yes`, retrieves the row data, appends it to the "Closed POA&M Items" sheet, and deletes the original row.

Dependencies and Imports

- **Google Apps Script Services**:
    - `SpreadsheetApp`: To access and manipulate the active spreadsheet.
    - `Logger`: To log actions for debugging.

Input/Output

- **Input**:
    - User edits in the "Configuration Findings" sheet, specifically in the "False Positive" column.
- **Output**:
    - Moves the edited row to the "Closed POA&M Items" sheet if marked `Yes`.
    - Logs actions such as row transfer.

Error Handling

- The script assumes the presence of the "False Positive" column and corresponding sheets.
- It does not handle scenarios where these elements are missing, which may lead to runtime errors.

Performance Considerations

- **Strengths**:
  - Dynamically locates the "False Positive" column, making it resilient to column reordering.
  - Efficiently processes the row transfer in real time.
- **Weaknesses**:
  - Deletes rows directly, which might disrupt references or formulas in the sheet.

Code Style and Readability

- Code is well-structured and uses descriptive variable names.
- Additional comments explaining the workflow would improve maintainability.

Potential Issues

1. Assumes the presence of "Configuration Findings" and "Closed POA&M Items" sheets.
2. Direct row deletion can lead to unintended consequences if other processes depend on row indices.

Security Considerations

- Relies on edit events, which might be exploited if malicious or erroneous edits are made.
- No validation is performed on the data before appending to the target sheet.

Extensibility and Reusability

- **Extensibility**:
  - Could be extended to handle other columns or automate further actions based on different criteria.
- **Reusability**:
  - Limited to spreadsheets with a specific structure. Generalizing the column and sheet names would improve reusability.

Suggestions for Improvement

1. Add validation to ensure that required sheets and columns are present before execution.
2. Implement logging for errors or missing prerequisites to improve debugging.
3. Use soft deletion (e.g., marking rows as inactive) instead of direct deletion for better traceability.
4. Provide configuration options for sheet names and column headers to enhance adaptability.