EPIC and User Story Document

Project: Attack Surface Analyzer

Version: 1.0

Date: September 18, 2024

Author: Arvind Kumar Sharma

User Story

Title: Implement Enhanced User Account Management in Attack Surface Analyzer

As a system administrator,
I want to have enhanced user account management features in the Attack Surface Analyzer,
so that I can efficiently manage user accounts, including their roles, permissions, and activity logs.

Acceptance Criteria

1. **User Account Creation:**
    - **Given** a system administrator,
    - **When** they navigate to the user management section and click on "Create New User",
    - **Then** they should be able to input user details (name, UID, GID, home directory, shell, etc.) and save the new user.
2. **User Account Modification:**
    - **Given** a system administrator,
    - **When** they select an existing user and click on "Edit",
    - **Then** they should be able to modify user details and save the changes.
3. **User Account Deletion:**
    - **Given** a system administrator,
    - **When** they select an existing user and click on "Delete",
    - **Then** the user account should be removed from the system after confirmation.
4. **Role Assignment:**
    - **Given** a system administrator,
    - **When** they create or edit a user,

    - **Then** they should be able to assign roles (e.g., administrator, standard user) to the user.
5. **Permission Management:**
    - **Given** a system administrator,
    - **When** they assign roles to a user,
    - **Then** the system should automatically apply the corresponding permissions based on the role.
6. **Activity Logs:**
    - **Given** a system administrator,
    - **When** they view the user management section,
    - **Then** they should be able to see a log of user activities (e.g., login times, changes made).

Boundary Conditions

- **User Input Validation:**
    - Ensure that all user inputs (e.g., UID, GID, home directory) are validated for correct format and uniqueness.
- **Role Constraints:**
    - Ensure that only users with administrative privileges can create, edit, or delete user accounts.
- **Concurrency Handling:**
    - Ensure that concurrent modifications to user accounts are handled gracefully, with appropriate locking mechanisms.

Business Rules

1. **Role Definitions:**
    - Define roles such as "Administrator" and "Standard User" with specific permissions.
    - Administrators can manage user accounts, view logs, and perform system-wide changes.
    - Standard Users have limited access, primarily to their own account details and logs.
2. **Password Policies:**
    - Enforce strong password policies, including minimum length, complexity requirements, and expiration periods.
3. **Audit Trails:**

- Maintain audit trails for all user account activities, including creation, modification, and deletion.

Non-Functional Requirements (NFRs)

1. **Performance:**
    - The user management operations (create, edit, delete) should complete within 2 seconds under normal load conditions.
2. **Security:**
    - Ensure that all user data is encrypted in transit and at rest.
    - Implement multi-factor authentication (MFA) for administrative access.
3. **Usability:**
    - The user management interface should be intuitive and user-friendly, with clear instructions and feedback messages.
4. **Scalability:**
    - The system should support up to 10,000 user accounts without performance degradation.
5. **Reliability:**
    - Ensure high availability of the user management features, with a target uptime of 99.9%.
6. **Compliance:**
    - Ensure that the user management features comply with relevant data protection regulations (e.g., GDPR, CCPA).

User Story 1: Analyze System Changes

User Story: As a security analyst, I want to analyze system changes introduced by software installations or system misconfigurations so that I can identify potential security implications.

Acceptance Criteria:

1. The system should allow the user to select the type of analysis (e.g., software installation, system misconfiguration).
2. The system should scan the selected system and generate a report detailing the changes.
3. The report should categorize changes by type (e.g., file changes, registry changes, driver changes).
4. The report should highlight changes that have potential security implications.

Boundary Conditions:

- The system should support Windows, Linux, and MacOS platforms.

- The analysis should be limited to the scope defined by the user (e.g., specific directories or registry keys).

Business Rules:

- The system should use predefined rules to identify changes with potential security implications.
- The system should allow users to customize the rules for their specific needs.

Non-Functional Requirements (NFRs):

- The analysis should complete within a reasonable time frame (e.g., less than 30 minutes for a full system scan).
- The system should provide clear and concise reports that are easy to understand.
- The system should be secure and protect user data during the analysis process.

User Story 2: Customize Analysis Rules

User Story: As a security analyst, I want to customize the analysis rules so that I can tailor the analysis to my specific security requirements.

Acceptance Criteria:

1. The system should provide a user interface for managing analysis rules.
2. The user should be able to add, edit, and delete rules.
3. The user should be able to specify the conditions under which a rule is triggered (e.g., file creation, registry modification).
4. The user should be able to specify the actions to be taken when a rule is triggered (e.g., log the change, generate an alert).

Boundary Conditions:

- The system should validate the rules to ensure they are correctly formatted and do not conflict with existing rules.
- The system should provide default rules that can be used as a starting point.

Business Rules:

- The system should allow users to export and import rules for sharing and backup purposes.
- The system should log all changes to the rules for auditing purposes.

Non-Functional Requirements (NFRs):

- The rule management interface should be intuitive and easy to use.
- The system should provide real-time feedback on the impact of rule changes.

- The system should ensure that rule changes do not negatively impact the performance of the analysis.

User Story 3: Generate Detailed Reports

User Story: As a security analyst, I want to generate detailed reports of the analysis so that I can review and address potential security issues.

Acceptance Criteria:

1. The system should generate a report at the end of each analysis.
2. The report should include a summary of the analysis results.
3. The report should provide detailed information on each change detected, including the type of change, the affected system component, and the potential security implications.
4. The report should allow the user to filter and sort the results.

Boundary Conditions:

- The report should be available in multiple formats (e.g., PDF, HTML, CSV).
- The report should be accessible through the system's user interface and available for download.

Business Rules:

- The system should allow users to customize the report format and content.
- The system should retain historical reports for future reference and comparison.

Non-Functional Requirements (NFRs):

- The report generation process should be efficient and not significantly impact system performance.
- The reports should be clear, accurate, and easy to understand.
- The system should ensure the confidentiality and integrity of the report data.

User Story 4: Perform User Acceptance Testing (UAT)

User Story: As a project manager, I want to perform user acceptance testing (UAT) to ensure that the software meets the business requirements.

Acceptance Criteria:

1. The system should provide a test environment that mirrors the production environment.
2. The system should allow users to execute predefined test cases.
3. The system should log the results of each test case, including any issues encountered.
4. The system should provide a summary of the UAT results, highlighting any areas that require further attention.

Boundary Conditions:

- The test environment should be isolated from the production environment to prevent any impact on live systems.
- The test cases should cover all critical business requirements and scenarios.

Business Rules:

- The system should allow users to create and manage test cases.
- The system should provide tools for tracking and resolving issues identified during UAT.

Non-Functional Requirements (NFRs):

- The UAT process should be efficient and not significantly delay the project timeline.
- The system should provide clear and actionable feedback on the UAT results.
- The system should ensure the security and integrity of the test environment and data.

Epic: Attack Surface Analyzer CLI Application

Description

As a user of the Attack Surface Analyzer (ASA) CLI application, I want to be able to collect, monitor, compare, and export data related to the attack surface of my system, so that I can identify potential security vulnerabilities and changes over time.

User Stories

User Story 1: Setup Logging

As a user,
I want to configure logging settings,
so that I can control the verbosity and format of the logs generated by the application.

Acceptance Criteria:

- The application should accept logging settings via command-line options.
- The logging configuration should be applied at the start of the application.
- The application should support different log levels (e.g., DEBUG, INFO, WARN, ERROR).
- The application should support logging to a file and/or console.

Boundary Conditions:

- If no logging settings are provided, the application should use default settings.
- If invalid logging settings are provided, the application should display an error message and use default settings.

Business Rules:

- Logging settings should be configurable to meet organizational compliance requirements.

Non-Functional Requirements (NFRs):

- Logging should not significantly impact the performance of the application.
- Logs should be easily readable and searchable.

User Story 2: Setup Database

As a user,
I want to configure database settings and initialize the database connection,
so that the application can store and retrieve data efficiently.

Acceptance Criteria:

- The application should accept database settings via command-line options.
- The database connection should be initialized at the start of the application.
- The application should support different types of databases (e.g., SQLite, SQL Server).

Boundary Conditions:

- If no database settings are provided, the application should use default settings.
- If invalid database settings are provided, the application should display an error message and terminate.

Business Rules:

- Database settings should be configurable to meet organizational compliance requirements.

Non-Functional Requirements (NFRs):

- Database operations should be optimized for performance.
- The application should handle database connection failures gracefully.

User Story 3: Load Rules

As a user,
I want to load rules from a specified file or use embedded rules,
so that I can customize the analysis based on my requirements.

Acceptance Criteria:

- The application should load rules from a specified file if provided.
- If no file is provided, the application should use embedded rules.
- The application should validate the rules before using them.

Boundary Conditions:

- If the specified file does not exist or is invalid, the application should display an error message and use embedded rules.

Business Rules:

- Rules should be customizable to meet specific security requirements.

Non-Functional Requirements (NFRs):

- Rule loading should be efficient and not significantly impact the startup time of the application.

User Story 4: Guided Mode Command

As a user,
I want to execute a guided mode command,
so that I can collect baseline data, monitor changes, and compare results in a single workflow.

Acceptance Criteria:

- The application should collect baseline data.
- The application should monitor system changes.
- The application should compare the collected data and provide results.
- The application should provide progress updates during the guided mode operation.

Boundary Conditions:

- If any step in the guided mode fails, the application should display an error message and terminate the operation.

Business Rules:

- Guided mode should follow a predefined workflow to ensure consistency.

Non-Functional Requirements (NFRs):

- Guided mode operations should be optimized for performance.
- The application should handle large datasets efficiently.

User Story 5: Analyze Monitored Data

As a user,
I want to analyze monitored data,
so that I can understand the changes that occurred over time.

Acceptance Criteria:

- The application should analyze monitored data based on specified options.
- The application should return comparison results.
- The application should provide detailed information about the changes detected.

Boundary Conditions:

- If no monitored data is available, the application should display an error message.

Business Rules:

- Analysis should be accurate and reliable.

Non-Functional Requirements (NFRs):

- Analysis operations should be optimized for performance.
- The application should handle large datasets efficiently.

User Story 6: Verify Rules

As a user,
I want to verify the rules specified in the options,
so that I can ensure the rules are valid and applicable.

Acceptance Criteria:

- The application should verify the rules and provide feedback on their validity.
- The application should display detailed error messages for invalid rules.

Boundary Conditions:

- If no rules are specified, the application should use default rules.

Business Rules:

- Rules should be verified to ensure they meet security requirements.

Non-Functional Requirements (NFRs):

- Rule verification should be efficient and not significantly impact the startup time of the application.

User Story 7: Insert Compare Results

As a user,
I want to insert comparison results into the database,
so that I can store and retrieve the results for future reference.

Acceptance Criteria:

- The application should insert comparison results into the database with relevant metadata.
- The application should ensure data integrity during the insertion process.

Boundary Conditions:

- If the database connection fails, the application should display an error message and retry the operation.

Business Rules:

- Comparison results should be stored in a structured format for easy retrieval.

Non-Functional Requirements (NFRs):

- Database insertion operations should be optimized for performance.
- The application should handle large datasets efficiently.

User Story 8: Setup Database or Terminate

As a user,
I want the application to setup the database or terminate if setup fails,
so that I can ensure the application runs with a valid database connection.

Acceptance Criteria:

- The application should setup the database.
- If setup fails, the application should terminate with an appropriate error message.

Boundary Conditions:

- If the database file is missing or corrupted, the application should display an error message and terminate.

Business Rules:

- Database setup should be verified to ensure data integrity.

Non-Functional Requirements (NFRs):

- Database setup operations should be optimized for performance.
- The application should handle database connection failures gracefully.

User Story 9: GUI Command

As a user,
I want to execute a GUI command,
so that I can start a web server for the GUI.

Acceptance Criteria:

- The application should start a web server for the GUI based on specified options.
- The application should provide feedback on the status of the web server.

Boundary Conditions:

- If the web server fails to start, the application should display an error message.

Business Rules:

- The GUI should be accessible via a web browser.

Non-Functional Requirements (NFRs):

- The web server should be optimized for performance.
- The GUI should be responsive and user-friendly.

User Story 10: Sleep and Open Browser

As a user,
I want the application to sleep for a specified duration and then open the default web browser,
so that I can access the GUI after a delay.

Acceptance Criteria:

- The application should sleep for the specified duration.
- The application should open the default web browser.

Boundary Conditions:

- If the browser fails to open, the application should display an error message.

Business Rules:

- The delay should be configurable to meet user requirements.

Non-Functional Requirements (NFRs):

- The sleep operation should not significantly impact the performance of the application.

User Story 11: Configuration Command

As a user,
I want to execute a configuration command,
so that I can reset the database, list runs, delete runs, or trim the database.

Acceptance Criteria:

- The application should support resetting the database.
- The application should support listing runs.
- The application should support deleting runs.
- The application should support trimming the database.

Boundary Conditions:

- If any configuration operation fails, the application should display an error message.

Business Rules:

- Configuration operations should be verified to ensure data integrity.

Non-Functional Requirements (NFRs):

- Configuration operations should be optimized for performance.
- The application should handle large datasets efficiently.

User Story 12: Export Collect Command

As a user,
I want to export collected data to a specified format,
so that I can share or analyze the data externally.

Acceptance Criteria:

- The application should export collected data based on specified options.
- The application should support different export formats (e.g., JSON, CSV).

Boundary Conditions:

- If the export operation fails, the application should display an error message.

Business Rules:

- Exported data should be structured and formatted for easy analysis.

Non-Functional Requirements (NFRs):

- Export operations should be optimized for performance.
- The application should handle large datasets efficiently.

User Story 13: Export Compare Results

As a user,
I want to export comparison results to a specified format,
so that I can share or analyze the results externally.

Acceptance Criteria:

- The application should export comparison results based on specified options.
- The application should support different export formats (e.g., JSON, CSV).

Boundary Conditions:

- If the export operation fails, the application should display an error message.

Business Rules:

- Exported data should be structured and formatted for easy analysis.

Non-Functional Requirements (NFRs):

- Export operations should be optimized for performance.
- The application should handle large datasets efficiently.

User Story 14: Write SARIF Log

As a user,
I want to write a SARIF log to a specified file,
so that I can use the log for further analysis or reporting.

Acceptance Criteria:

- The application should write a SARIF log to the specified file.
- The SARIF log should be compliant with the SARIF standard.

Boundary Conditions:

- If the file path is invalid, the application should display an error message.

Business Rules:

- SARIF logs should be structured and formatted for easy analysis.

Non-Functional Requirements (NFRs):

- SARIF log writing operations should be optimized for performance.
- The application should handle large datasets efficiently.

User Story 15: Generate SARIF Log

As a user,
I want to generate a SARIF log from the specified output and rules,
so that I can create a standardized log for analysis.

Acceptance Criteria:

- The application should generate a SARIF log based on the specified output and rules.
- The SARIF log should be compliant with the SARIF standard.

Boundary Conditions:

- If the output or rules are invalid, the application should display an error message.

Business Rules:

- SARIF logs should be structured and formatted for easy analysis.

Non-Functional Requirements (NFRs):

- SARIF log generation operations should be optimized for performance.
- The application should handle large datasets efficiently.

User Story 16: Export Monitor Command

As a user,
I want to export monitored data to a specified format,
so that I can share or analyze the data externally.

Acceptance Criteria:

- The application should export monitored data based on specified options.
- The application should support different export formats (e.g., JSON, CSV).

Boundary Conditions:

- If the export operation fails, the application should display an error message.

Business Rules:

- Exported data should be structured and formatted for easy analysis.

Non-Functional Requirements (NFRs):

- Export operations should be optimized for performance.
- The application should handle large datasets efficiently.

User Story 17: Write Monitor JSON

As a user,
I want to write monitored data to a JSON file,
so that I can use the data for further analysis or reporting.

Acceptance Criteria:

- The application should write monitored data to a JSON file.
- The JSON file should be structured and formatted for easy analysis.

Boundary Conditions:

- If the file path is invalid, the application should display an error message.

Business Rules:

- JSON files should be structured and formatted for easy analysis.

Non-Functional Requirements (NFRs):

- JSON writing operations should be optimized for performance.
- The application should handle large datasets efficiently.

User Story 18: Monitor Command

As a user,
I want to execute a monitor command,
so that I can monitor system changes based on specified options.

Acceptance Criteria:

- The application should monitor system changes based on specified options.
- The application should provide real-time feedback on the monitoring status.

Boundary Conditions:

- If the monitoring operation fails, the application should display an error message.

Business Rules:

- Monitoring operations should be accurate and reliable.

Non-Functional Requirements (NFRs):

- Monitoring operations should be optimized for performance.
- The application should handle large datasets efficiently.

User Story 19: Retrieve Collectors

As a user,
I want to retrieve the list of collectors,
so that I can understand which collectors are available for data collection.

Acceptance Criteria:

- The application should return the list of collectors.
- The list should include detailed information about each collector.

Boundary Conditions:

- If no collectors are available, the application should display an appropriate message.

Business Rules:

- Collectors should be documented to ensure users understand their functionality.

Non-Functional Requirements (NFRs):

- Retrieving the list of collectors should be efficient and not significantly impact the performance of the application.

User Story 20: Retrieve Monitors

As a user,
I want to retrieve the list of monitors,
so that I can understand which monitors are available for monitoring system changes.

Acceptance Criteria:

- The application should return the list of monitors.
- The list should include detailed information about each monitor.

Boundary Conditions:

- If no monitors are available, the application should display an appropriate message.

Business Rules:

- Monitors should be documented to ensure users understand their functionality.

Non-Functional Requirements (NFRs):

- Retrieving the list of monitors should be efficient and not significantly impact the performance of the application.

User Story 21: Retrieve Comparators

As a user,
I want to retrieve the list of comparators,
so that I can understand which comparators are available for comparing data.

Acceptance Criteria:

- The application should return the list of comparators.
- The list should include detailed information about each comparator.

Boundary Conditions:

- If no comparators are available, the application should display an appropriate message.

Business Rules:

- Comparators should be documented to ensure users understand their functionality.

Non-Functional Requirements (NFRs):

- Retrieving the list of comparators should be efficient and not significantly impact the performance of the application.

User Story 22: Compare Runs

As a user,
I want to compare two runs based on specified options,
so that I can identify differences between the runs.

Acceptance Criteria:

- The application should compare two runs based on specified options.
- The application should return the comparison results.
- The comparison results should include detailed information about the differences detected.

Boundary Conditions:

- If no runs are available for comparison, the application should display an appropriate message.

Business Rules:

- Comparison operations should be accurate and reliable.

Non-Functional Requirements (NFRs):

- Comparison operations should be optimized for performance.
- The application should handle large datasets efficiently.

User Story 23: GUI Monitor Command

As a user,
I want to execute a GUI monitor command,
so that I can start monitoring in GUI mode.

Acceptance Criteria:

- The application should start monitoring in GUI mode based on specified options.
- The application should provide real-time feedback on the monitoring status.

Boundary Conditions:

- If the monitoring operation fails, the application should display an error message.

Business Rules:

- Monitoring operations should be accurate and reliable.

Non-Functional Requirements (NFRs):

- Monitoring operations should be optimized for performance.
- The application should handle large datasets efficiently.

User Story 24: Stop Monitors

As a user,
I want to stop all active monitors,
so that I can terminate monitoring activities.

Acceptance Criteria:

- The application should stop all active monitors.
- The application should provide feedback on the status of the monitors.

Boundary Conditions:

- If no monitors are active, the application should display an appropriate message.

Business Rules:

- Monitoring operations should be accurately tracked to ensure they can be stopped reliably.

Non-Functional Requirements (NFRs):

- Stopping monitors should be efficient and not significantly impact the performance of the application.

User Story 25: Admin Check

As a user,
I want the application to check if it is running with administrative privileges,
so that I can be warned if it is not.

Acceptance Criteria:

- The application should check for administrative privileges.
- The application should log a warning if it is not running with administrative privileges.

Boundary Conditions:

- If the application is running with administrative privileges, no warning should be displayed.

Business Rules:

- Administrative privileges should be verified to ensure the application can perform all required operations.

Non-Functional Requirements (NFRs):

- The admin check should be efficient and not significantly impact the startup time of the application.

User Story 26: Collect Command

As a user,
I want to execute a collect command,
so that I can collect data based on specified options.

Acceptance Criteria:

- The application should collect data based on specified options.
- The application should provide real-time feedback on the collection status.

Boundary Conditions:

- If the collection operation fails, the application should display an error message.

Business Rules:

- Data collection operations should be accurate and reliable.

Non-Functional Requirements (NFRs):

- Data collection operations should be optimized for performance.
- The application should handle large datasets efficiently.

User Story 27: Flush Results

As a user,
I want the application to flush results to the database,
so that I can ensure all collected data is stored.

Acceptance Criteria:

- The application should flush results to the database.
- The application should ensure data integrity during the flushing process.

Boundary Conditions:

- If the database connection fails, the application should display an error message and retry the operation.

Business Rules:

- Data flushing operations should be verified to ensure data integrity.

Non-Functional Requirements (NFRs):

- Data flushing operations should be optimized for performance.
- The application should handle large datasets efficiently.

User Story 28: Clear Collectors

As a user,
I want to clear the list of collectors,
so that I can reset the collectors for a new collection session.

Acceptance Criteria:

- The application should clear the list of collectors.
- The application should provide feedback on the status of the collectors.

Boundary Conditions:

- If no collectors are available, the application should display an appropriate message.

Business Rules:

- Collectors should be accurately tracked to ensure they can be cleared reliably.

Non-Functional Requirements (NFRs):

- Clearing collectors should be efficient and not significantly impact the

Sorry, the response hit the length limit. Please rephrase your prompt.

User Story 31: Display Help Information

As a user,
I want to display help information for the CLI commands,
so that I can understand how to use the application and its various options.

Acceptance Criteria:

- The application should provide a help command that displays usage information and available options for each command.

User Story 32: Validate Command-Line Arguments

As a user,
I want the application to validate command-line arguments,
so that I can be informed of any incorrect or missing arguments before execution.

Acceptance Criteria:

- The application should validate the provided command-line arguments.
- The application should display an error message if any arguments are incorrect or missing.

User Story 33: Provide Command Execution Feedback

As a user,
I want the application to provide feedback on the execution of commands,
so that I can understand the outcome of each command.

Acceptance Criteria:

- The application should display success or error messages after executing a command.
- The application should provide detailed error messages if a command fails.

User Story 34: Support Multiple Output Formats

As a user,
I want the application to support multiple output formats for exported data,
so that I can choose the format that best suits my needs.

Acceptance Criteria:

- The application should support exporting data in various formats such as JSON, CSV, and SARIF.
- The application should allow users to specify the desired output format via command-line options.

User Story 35: Schedule Data Collection

As a user,
I want to schedule data collection at specific intervals,
so that I can automate the process of collecting data over time.

Acceptance Criteria:

- The application should allow users to schedule data collection at specified intervals.
- The application should execute the data collection commands automatically based on the schedule.

User Story 36: Pause and Resume Monitoring

As a user,
I want to pause and resume monitoring activities,
so that I can temporarily stop monitoring without losing the current state.

Acceptance Criteria:

- The application should allow users to pause monitoring activities.

- The application should allow users to resume monitoring from the paused state.

User Story 37: Filter Collected Data

As a user,
I want to filter collected data based on specific criteria,
so that I can focus on the most relevant information.

Acceptance Criteria:

- The application should allow users to specify filters for collected data.
- The application should apply the filters and display only the relevant data.

User Story 38: Generate Reports

As a user,
I want to generate detailed reports from the collected and compared data,
so that I can present the findings to stakeholders.

Acceptance Criteria:

- The application should generate detailed reports based on the collected and compared data.
- The reports should be available in various formats such as PDF and HTML.

User Story 39: Manage User Permissions

As an administrator,
I want to manage user permissions for accessing and executing commands,
so that I can control who can use the application and its features.

Acceptance Criteria:

- The application should support user authentication and authorization.
- The application should allow administrators to assign and manage permissions for different users.

User Story 40: Provide Real-Time Monitoring Dashboard

As a user,
I want to access a real-time monitoring dashboard,
so that I can visualize ongoing monitoring activities and system changes.

Acceptance Criteria:

- The application should provide a real-time monitoring dashboard accessible via a web interface.
- The dashboard should display ongoing monitoring activities and detected system changes in real-time.

User Story 41: Support Internationalization

As a user,
I want the application to support multiple languages,
so that I can use it in my preferred language.

Acceptance Criteria:

- The application should support internationalization and localization.
- The application should allow users to select their preferred language for the interface and messages.

User Story 42: Backup and Restore Database

As a user,
I want to backup and restore the database,
so that I can ensure data integrity and recover from potential data loss.

Acceptance Criteria:

- The application should provide commands to backup the database.
- The application should provide commands to restore the database from a backup.

User Story 43: Monitor Resource Usage

As a user,
I want the application to monitor its own resource usage,
so that I can ensure it is not consuming excessive system resources.

Acceptance Criteria:

- The application should monitor its CPU, memory, and disk usage.
- The application should provide feedback on its resource usage and alert users if it exceeds specified thresholds.

User Story 44: Provide Detailed Logs

As a user,
I want the application to provide detailed logs of its operations,
so that I can troubleshoot issues and understand the application's behavior.

Acceptance Criteria:

- The application should generate detailed logs for all operations.
- The logs should include timestamps, command details, and error messages.

User Story 45: Support Custom Plugins

As a developer,
I want to create and use custom plugins with the application,
so that I can extend its functionality to meet specific requirements.

Acceptance Criteria:

- The application should support a plugin architecture.
- Developers should be able to create and integrate custom plugins with the application.

User Story 46: Provide API for Integration

As a developer,
I want the application to provide an API,
so that I can integrate it with other systems and automate tasks.

Acceptance Criteria:

- The application should provide a RESTful API for integration.
- The API should support all major commands and operations of the application.

User Story 47: Handle Large Data Sets

As a user,
I want the application to handle large data sets efficiently,
so that I can analyze extensive data without performance issues.

Acceptance Criteria:

- The application should be optimized to handle large data sets.
- The application should provide feedback on the progress of operations involving large data sets.

User Story 48: Provide Contextual Help

As a user,
I want the application to provide contextual help for commands and options,
so that I can understand how to use specific features without referring to external documentation.

Acceptance Criteria:

- The application should provide contextual help for each command and option.
- The help information should be accessible via command-line options.

User Story 49: Support Multiple Platforms

As a user,
I want the application to support multiple platforms,
so that I can use it on different operating systems.

Acceptance Criteria:

- The application should be compatible with Windows, macOS, and Linux.
- The application should provide platform-specific installation and usage instructions.

User Story 50: Provide Version Information

As a user,
I want the application to provide version information,
so that I can ensure I am using the latest version.

Acceptance Criteria:

- The application should provide a command to display its version information.
- The version information should include the application version, build date, and any relevant metadata.

Epic : AsaAnalyzer class:

User Story 1: Initialize AsaAnalyzer with Custom Options

As a developer,
I want to initialize the AsaAnalyzer with custom options,
so that I can configure the analyzer according to my specific requirements.

Acceptance Criteria:

- The AsaAnalyzer should accept an optional AnalyzerOptions parameter during initialization.
- The custom property extraction and object-to-values delegates should be added during initialization.

User Story 2: Parse Custom ASA Object Values

As a developer,
I want to parse custom ASA object values,
so that I can extract key-value pairs from specific object types.

Acceptance Criteria:

- The ParseCustomAsaObjectValues method should handle objects of type Dictionary<(TpmAlgId, uint), byte[]>.
- The method should return a tuple indicating whether the object was processed and the extracted key-value pairs.

User Story 3: Parse Custom ASA Properties

As a developer,
I want to parse custom ASA properties based on an index,
so that I can retrieve specific properties from supported object types.

Acceptance Criteria:

- The ParseCustomAsaProperties method should handle objects of type Dictionary<(TpmAlgId, uint), byte[]>.

- The method should parse the index to retrieve the corresponding property value.
- The method should return a tuple indicating whether the property was found and the extracted value.

User Story 4: Analyze Compare Results

As a developer,
I want to analyze compare results using a set of rules,
so that I can determine which rules apply to the comparison.

Acceptance Criteria:

- The Analyze method should accept a collection of rules and a CompareResult object.
- The method should return the rules that apply to the comparison results.
- If the CompareResult is null, the method should return an empty collection of rules.

User Story 5: Handle Null Compare Results

As a developer,
I want the Analyze method to handle null CompareResult objects gracefully,
so that the application does not crash when a null comparison result is provided.

Acceptance Criteria:

- The Analyze method should check if the CompareResult is null.
- If the CompareResult is null, the method should return an empty collection of rules.

User Story 6: Extract Custom Properties and Values

As a developer,
I want the AsaAnalyzer to support custom property extraction and object-to-values conversion,
so that I can extend the analyzer's functionality for specific data types.

Acceptance Criteria:

- The AsaAnalyzer should include delegates for custom property extraction and object-to-values conversion.
- The custom delegates should be invoked during the analysis process to handle specific data types.

User Story 7: Convert Dictionary Values to Base64

As a developer,
I want the ParseCustomAsaObjectValues method to convert dictionary values to Base64 strings,
so that the extracted values are in a standardized format.

Acceptance Criteria:

- The ParseCustomAsaObjectValues method should convert the byte array values in the dictionary to Base64 strings.

- The method should return the key-value pairs with the keys as strings and the values as Base64 strings.

User Story 8: Parse Index for Custom Properties

As a developer,
I want the ParseCustomAsaProperties method to parse the index string correctly,
so that it can retrieve the corresponding property value from the dictionary.

Acceptance Criteria:

- The ParseCustomAsaProperties method should parse the index string to extract the algorithm ID and index.
- The method should retrieve the corresponding value from the dictionary based on the parsed index.

User Story 9: Support Multiple Data Types

As a developer,
I want the AsaAnalyzer to support multiple data types for custom property extraction and value conversion,
so that I can extend the analyzer to handle various types of data.

Acceptance Criteria:

- The AsaAnalyzer should be designed to support multiple data types for custom property extraction and value conversion.
- The custom delegates should be flexible enough to handle different types of data objects.

User Story 10: Provide Clear Error Messages

As a developer,
I want the AsaAnalyzer to provide clear error messages when parsing fails,
so that I can easily debug issues with custom property extraction and value conversion.

Acceptance Criteria:

- The AsaAnalyzer should provide clear error messages when parsing custom properties or values fails.
- The error messages should indicate the reason for the failure and the specific data that caused the issue.

Epic: NV Index Attribute Management

Description

As a developer working with the AsaNvIndex class, I want to manage and analyze NV index attributes, so that I can understand and utilize the properties and permissions associated with NV indices in my application.

User Stories

User Story 1: Retrieve NV Index Attributes

As a developer,
I want to retrieve the NV index attributes,
so that I can understand the properties and permissions associated with the NV index.

Acceptance Criteria:

- The AsaNvIndex class should provide a property to retrieve the NvAttr attributes.
- The attributes should be accessible via the Attributes property.

User Story 2: Check AuthRead Flag

As a developer,
I want to check if the AuthRead flag is set,
so that I can determine if authorization is required for reading the NV index.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property AuthRead.
- The AuthRead property should return true if the Authread flag is set in the NvAttr attributes.

User Story 3: Check AuthWrite Flag

As a developer,
I want to check if the AuthWrite flag is set,
so that I can determine if authorization is required for writing to the NV index.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property AuthWrite.
- The AuthWrite property should return true if the Authwrite flag is set in the NvAttr attributes.

User Story 4: Check Bits Flag

As a developer,
I want to check if the Bits flag is set,
so that I can determine if the NV index is configured as a bit field.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property Bits.
- The Bits property should return true if the Bits flag is set in the NvAttr attributes.

User Story 5: Check ClearStClear Flag

As a developer,
I want to check if the ClearStClear flag is set,
so that I can determine if the NV index is cleared by TPM2_Clear.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property ClearStClear.
- The ClearStClear property should return true if the ClearStclear flag is set in the NvAttr attributes.

User Story 6: Check Counter Flag

As a developer,
I want to check if the Counter flag is set,
so that I can determine if the NV index is configured as a counter.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property Counter.
- The Counter property should return true if the Counter flag is set in the NvAttr attributes.

User Story 7: Check Extend Flag

As a developer,
I want to check if the Extend flag is set,
so that I can determine if the NV index is configured for extend operations.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property Extend.
- The Extend property should return true if the Extend flag is set in the NvAttr attributes.

User Story 8: Check GlobalLock Flag

As a developer,
I want to check if the GlobalLock flag is set,
so that I can determine if the NV index is locked globally.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property GlobalLock.
- The GlobalLock property should return true if the Globallock flag is set in the NvAttr attributes.

User Story 9: Retrieve NV Index

As a developer,
I want to retrieve the NV index value,
so that I can identify the specific NV index being analyzed.

Acceptance Criteria:

- The AsaNvIndex class should provide a property to retrieve the NV index value.
- The NV index value should be accessible via the Index property.

User Story 10: Check NoDa Flag

As a developer,
I want to check if the NoDa flag is set,
so that I can determine if the NV index is not subject to dictionary attack protection.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property NoDa.
- The NoDa property should return true if the NoDa flag is set in the NvAttr attributes.

User Story 11: Check None Flag

As a developer,
I want to check if the None flag is set,
so that I can determine if no specific attributes are set for the NV index.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property None.
- The None property should return true if the None flag is set in the NvAttr attributes.

User Story 12: Check Orderly Flag

As a developer,
I want to check if the Orderly flag is set,
so that I can determine if the NV index is maintained in an orderly state.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property Orderly.
- The Orderly property should return true if the Orderly flag is set in the NvAttr attributes.

User Story 13: Check Ordinary Flag

As a developer,
I want to check if the Ordinary flag is set,
so that I can determine if the NV index is configured as an ordinary index.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property Ordinary.
- The Ordinary property should return true if the Ordinary flag is set in the NvAttr attributes.

User Story 14: Check OwnerRead Flag

As a developer,
I want to check if the OwnerRead flag is set,
so that I can determine if the owner authorization is required for reading the NV index.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property OwnerRead.

- The OwnerRead property should return true if the Ownerread flag is set in the NvAttr attributes.

User Story 15: Check OwnerWrite Flag

As a developer,
I want to check if the OwnerWrite flag is set,
so that I can determine if the owner authorization is required for writing to the NV index.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property OwnerWrite.
- The OwnerWrite property should return true if the Ownerwrite flag is set in the NvAttr attributes.

User Story 16: Check PinFail Flag

As a developer,
I want to check if the PinFail flag is set,
so that I can determine if the NV index is configured to track PIN failures.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property PinFail.
- The PinFail property should return true if the PinFail flag is set in the NvAttr attributes.

User Story 17: Check PinPass Flag

As a developer,
I want to check if the PinPass flag is set,
so that I can determine if the NV index is configured to track PIN successes.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property PinPass.
- The PinPass property should return true if the PinPass flag is set in the NvAttr attributes.

User Story 18: Check PlatformCreate Flag

As a developer,
I want to check if the PlatformCreate flag is set,
so that I can determine if the NV index was created by the platform.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property PlatformCreate.
- The PlatformCreate property should return true if the Platformcreate flag is set in the NvAttr attributes.

User Story 19: Check PolicyDelete Flag

As a developer,
I want to check if the PolicyDelete flag is set,
so that I can determine if the NV index can be deleted based on policy.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property PolicyDelete.
- The PolicyDelete property should return true if the PolicyDelete flag is set in the NvAttr attributes.

User Story 20: Check PolicyRead Flag

As a developer,
I want to check if the PolicyRead flag is set,
so that I can determine if the NV index can be read based on policy.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property PolicyRead.
- The PolicyRead property should return true if the Policyread flag is set in the NvAttr attributes.

User Story 21: Check PolicyWrite Flag

As a developer,
I want to check if the PolicyWrite flag is set,
so that I can determine if the NV index can be written based on policy.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property PolicyWrite.
- The PolicyWrite property should return true if the Policywrite flag is set in the NvAttr attributes.

User Story 22: Check Ppread Flag

As a developer,
I want to check if the Ppread flag is set,
so that I can determine if physical presence is required for reading the NV index.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property Ppread.
- The Ppread property should return true if the Ppread flag is set in the NvAttr attributes.

User Story 23: Check Ppwrite Flag

As a developer,
I want to check if the Ppwrite flag is set,
so that I can determine if physical presence is required for writing to the NV index.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property Ppwrite.
- The Ppwrite property should return true if the Ppwrite flag is set in the NvAttr attributes.

User Story 24: Check ReadLocked Flag

As a developer,
I want to check if the ReadLocked flag is set,
so that I can determine if the NV index is locked for reading.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property ReadLocked.
- The ReadLocked property should return true if the Readlocked flag is set in the NvAttr attributes.

User Story 25: Check ReadStClear Flag

As a developer,
I want to check if the ReadStClear flag is set,
so that I can determine if the NV index is cleared on TPM2_Clear.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property ReadStClear.
- The ReadStClear property should return true if the ReadStclear flag is set in the NvAttr attributes.

User Story 26: Check TpmNtBit0 Flag

As a developer,
I want to check if the TpmNtBit0 flag is set,
so that I can determine if the NV index is configured with the TPM NT bit 0 attribute.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property TpmNtBit0.
- The TpmNtBit0 property should return true if the TpmNtBit0 flag is set in the NvAttr attributes.

User Story 27: Check TpmNtBit1 Flag

As a developer,
I want to check if the TpmNtBit1 flag is set,
so that I can determine if the NV index is configured with the TPM NT bit 1 attribute.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property TpmNtBit1.
- The TpmNtBit1 property should return true if the TpmNtBit1 flag is set in the NvAttr attributes.

User Story 28: Check TpmNtBit2 Flag

As a developer,
I want to check if the TpmNtBit2 flag is set,
so that I can determine if the NV index is configured with the TPM NT bit 2 attribute.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property TpmNtBit2.
- The TpmNtBit2 property should return true if the TpmNtBit2 flag is set in the NvAttr attributes.

User Story 29: Check TpmNtBit3 Flag

As a developer,
I want to check if the TpmNtBit3 flag is set,
so that I can determine if the NV index is configured with the TPM NT bit 3 attribute.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property TpmNtBit3.
- The TpmNtBit3 property should return true if the TpmNtBit3 flag is set in the NvAttr attributes.

User Story 30: Check TpmNtBitLength Flag

As a developer,
I want to check if the TpmNtBitLength flag is set,
so that I can determine if the NV index is configured with the TPM NT bit length attribute.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property TpmNtBitLength.
- The TpmNtBitLength property should return true if the TpmNtBitLength flag is set in the NvAttr attributes.

User Story 31: Check TpmNtBitMask Flag

As a developer,
I want to check if the TpmNtBitMask flag is set,
so that I can determine if the NV index is configured with the TPM NT bit mask attribute.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property TpmNtBitMask.
- The TpmNtBitMask property should return true if the TpmNtBitMask flag is set in the NvAttr attributes.

User Story 32: Check TpmNtBitOffset Flag

As a developer,
I want to check if the TpmNtBitOffset flag is set,
so that I can determine if the NV index is configured with the TPM NT bit offset attribute.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property TpmNtBitOffset.
- The TpmNtBitOffset property should return true if the TpmNtBitOffset flag is set in the NvAttr attributes.

User Story 33: Retrieve NV Index Value

As a developer,
I want to retrieve the value stored in the NV index,
so that I can analyze or utilize the stored data.

Acceptance Criteria:

- The AsaNvIndex class should provide a property to retrieve the value stored in the NV index.
- The value should be accessible via the value property.

User Story 34: Check Writeall Flag

As a developer,
I want to check if the Writeall flag is set,
so that I can determine if the NV index allows writing to all locations.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property Writeall.
- The Writeall property should return true if the Writeall flag is set in the NvAttr attributes.

User Story 35: Check Writedefine Flag

As a developer,
I want to check if the Writedefine flag is set,
so that I can determine if the NV index allows writing to define the index.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property Writedefine.
- The Writedefine property should return true if the Writedefine flag is set in the NvAttr attributes.

User Story 36: Check Writelocked Flag

As a developer,
I want to check if the Writelocked flag is set,
so that I can determine if the NV index is locked for writing.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property Writelocked.
- The Writelocked property should return true if the Writelocked flag is set in the NvAttr attributes.

User Story 37: Check WriteStclear Flag

As a developer,
I want to check if the WriteStclear flag is set,
so that I can determine if the NV index is cleared on TPM2_Clear.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property WriteStclear.
- The WriteStclear property should return true if the WriteStclear flag is set in the NvAttr attributes.

User Story 38: Check Written Flag

As a developer,
I want to check if the Written flag is set,
so that I can determine if the NV index has been written to.

Acceptance Criteria:

- The AsaNvIndex class should provide a boolean property Written.
- The Written property should return true if the Written flag is set in the NvAttr attributes.

Epic: Utility Functions for Attack Surface Analyzer

Description

As a developer working on the Attack Surface Analyzer (ASA) project, I want a set of utility functions to handle common tasks such as generating metadata, retrieving OS information, converting data formats, and managing file operations, so that I can streamline the development process and ensure consistency across the application.

User Stories

User Story 1: Generate Metadata

As a developer,
I want to generate metadata about the current environment,
so that I can include this information in reports and logs.

Acceptance Criteria:

- The GenerateMetadata method should return a dictionary with keys compare-version, compare-os, and compare-osversion.
- The compare-version should be retrieved using the GetVersionString method.
- The compare-os should be retrieved using the GetOsName method.

- The compare-osversion should be retrieved using the GetOsVersion method.

Business Rules:

- Metadata should be accurate and reflect the current environment.

Validations:

- Ensure that the returned dictionary contains all required keys.
- Ensure that the values are non-empty strings.

Boundary Conditions:

- Handle cases where OS information cannot be retrieved.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 2: Retrieve OS Name

As a developer,
I want to retrieve the name of the operating system,
so that I can include this information in reports and logs.

Acceptance Criteria:

- The GetOsName method should return the name of the operating system.
- For Windows, it should use the GetPlatformString method.
- For Linux and macOS, it should use the uname -s command.

Business Rules:

- The OS name should be accurate and reflect the current environment.

Validations:

- Ensure that the returned string is non-empty.
- Ensure that the uname command executes successfully on Linux and macOS.

Boundary Conditions:

- Handle cases where the uname command fails or returns an empty string.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 3: Retrieve OS Version

As a developer,
I want to retrieve the version of the operating system,
so that I can include this information in reports and logs.

Acceptance Criteria:

- The GetOsVersion method should return the version of the operating system.
- For Windows, it should use Environment.OSVersion.VersionString.
- For Linux and macOS, it should use the uname -r command.

Business Rules:

- The OS version should be accurate and reflect the current environment.

Validations:

- Ensure that the returned string is non-empty.
- Ensure that the uname command executes successfully on Linux and macOS.

Boundary Conditions:

- Handle cases where the uname command fails or returns an empty string.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 4: Retrieve Platform Enum

As a developer,
I want to retrieve the platform as an enum value,
so that I can use it in conditional logic within the application.

Acceptance Criteria:

- The GetPlatform method should return a PLATFORM enum value.
- It should return PLATFORM.LINUX for Linux, PLATFORM.WINDOWS for Windows, and PLATFORM.MACOS for macOS.
- It should return PLATFORM.UNKNOWN for unsupported platforms.

Business Rules:

- The platform enum should accurately reflect the current environment.

Validations:

- Ensure that the returned enum value is one of the defined PLATFORM values.

Boundary Conditions:

- Handle cases where the platform cannot be determined.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 5: Retrieve Platform String

As a developer,
I want to retrieve the platform as a string,
so that I can include this information in reports and logs.

Acceptance Criteria:

- The GetPlatformString method should return the platform as a string.
- It should return "LINUX" for Linux, "WINDOWS" for Windows, and "MACOS" for macOS.
- It should return "UNKNOWN" for unsupported platforms.

Business Rules:

- The platform string should accurately reflect the current environment.

Validations:

- Ensure that the returned string is one of the defined platform strings.

Boundary Conditions:

- Handle cases where the platform cannot be determined.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 6: Generate Temporary Folder Path

As a developer,
I want to generate a unique temporary folder path,
so that I can use it for temporary file storage.

Acceptance Criteria:

- The GetTempFolder method should return a unique temporary folder path.
- The path should be generated using a random alphanumeric string.
- The method should ensure that the generated path does not already exist.

Business Rules:

- The temporary folder path should be unique and not conflict with existing paths.

Validations:

- Ensure that the returned path is unique and does not already exist.

Boundary Conditions:

- Handle cases where the temporary folder cannot be created.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 7: Retrieve Version String

As a developer,
I want to retrieve the version string of the application,
so that I can include this information in reports and logs.

Acceptance Criteria:

- The GetVersionString method should return the version string of the application.
- It should retrieve the version from the AssemblyInformationalVersionAttribute.

Business Rules:

- The version string should accurately reflect the current version of the application.

Validations:

- Ensure that the returned string is non-empty and correctly formatted.

Boundary Conditions:

- Handle cases where the version string cannot be retrieved.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 8: Convert Hex String to Bytes

As a developer,
I want to convert a hex string to a byte array,
so that I can process binary data represented as hex.

Acceptance Criteria:

- The HexStringToBytes method should convert a hex string to a byte array.
- It should handle cases where the hex string is null or invalid.

Business Rules:

- The conversion should be accurate and handle all valid hex strings.

Validations:

- Ensure that the returned byte array is correct for valid hex strings.
- Ensure that the method handles null or invalid hex strings gracefully.

Boundary Conditions:

- Handle cases where the hex string is null or contains invalid characters.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 9: Check if User is Admin

As a developer,
I want to check if the current user has administrative privileges,
so that I can perform actions that require elevated permissions.

Acceptance Criteria:

- The IsAdmin method should return true if the current user has administrative privileges.
- It should cache the result to avoid repeated checks.

Business Rules:

- The method should accurately determine the user's administrative status.

Validations:

- Ensure that the method returns the correct result based on the user's privileges.

Boundary Conditions:

- Handle cases where the user's administrative status cannot be determined.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 10: Check if Object is Dictionary

As a developer,
I want to check if an object is a dictionary,
so that I can handle it appropriately in my code.

Acceptance Criteria:

- The IsDictionary method should return true if the object is a dictionary.
- It should handle null objects gracefully.

Business Rules:

- The method should accurately determine if the object is a dictionary.

Validations:

- Ensure that the method returns the correct result for dictionary and non-dictionary objects.

Boundary Conditions:

- Handle cases where the object is null.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 11: Check if Object is List

As a developer,
I want to check if an object is a list,
so that I can handle it appropriately in my code.

Acceptance Criteria:

- The IsList method should return true if the object is a list.
- It should handle null objects gracefully.

Business Rules:

- The method should accurately determine if the object is a list.

Validations:

- Ensure that the method returns the correct result for list and non-list objects.

Boundary Conditions:

- Handle cases where the object is null.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 12: Make Valid File Name

As a developer,
I want to convert a string to a valid file name,
so that I can use it for file operations without encountering errors.

Acceptance Criteria:

- The MakeValidFileName method should replace invalid characters in the string with underscores.
- It should handle null or empty strings gracefully.

Business Rules:

- The resulting file name should be valid and not contain any invalid characters.

Validations:

- Ensure that the method correctly replaces all invalid characters.
- Ensure that the method handles null or empty strings gracefully.

Boundary Conditions:

- Handle cases where the input string is null or empty.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 13: Open URL in Browser

As a developer,
I want to open a URL in the default web browser,
so that I can direct users to web resources.

Acceptance Criteria:

- The OpenBrowser method should open the specified URL in the default web browser.
- It should handle null URLs gracefully.

Business Rules:

- The method should accurately open the URL in the default browser for the current platform.

Validations:

- Ensure that the method opens the correct URL.
- Ensure that the method handles null URLs gracefully.

Boundary Conditions:

- Handle cases where the URL is null or invalid.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 14: Generate Compare ID from Run IDs

As a developer,
I want to generate a compare ID from two run IDs,
so that I can uniquely identify a comparison between two runs.

Acceptance Criteria:

- The RunIdsToCompareId method should concatenate the two run IDs with an ampersand.
- It should handle null or empty run IDs gracefully.

Business Rules:

- The compare ID should be unique and accurately represent the two run IDs.

Validations:

- Ensure that the method correctly concatenates the two run IDs.
- Ensure that the method handles null or empty run IDs gracefully.

Boundary Conditions:

- Handle cases where one or both run IDs are null or empty.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 15: Convert SID to Name

As a developer,
I want to convert a security identifier (SID) to a user-friendly name,
so that I can display meaningful information about users.

Acceptance Criteria:

- The SidToName method should convert a SID to a user-friendly name.
- It should cache the results to avoid repeated lookups.
- It should handle null SIDs gracefully.

Business Rules:

- The method should accurately convert SIDs to user-friendly names.

Validations:

- Ensure that the method returns the correct name for valid SIDs.
- Ensure that the method handles null or invalid SIDs gracefully.

Boundary Conditions:

- Handle cases where the SID cannot be converted to a name.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.
- The method should cache results to improve performance.

User Story 16: Handle Random Number Generation

As a developer,
I want to generate random numbers for various purposes,
so that I can use them in scenarios like generating unique identifiers.

Acceptance Criteria:

- The AsaHelpers class should provide a static Random instance for generating random numbers.
- The random number generation should be thread-safe.

Business Rules:

- The random numbers should be uniformly distributed and unpredictable.

Validations:

- Ensure that the random numbers are uniformly distributed.
- Ensure that the random number generation is thread-safe.

Boundary Conditions:

- Handle cases where the random number generator fails.

Non-Functional Requirements (NFRs):

- The random number generation should be efficient and not block the main thread.

User Story 17: Manage SID Cache

As a developer,
I want to manage a cache of SIDs and their corresponding names,
so that I can improve the performance of SID-to-name conversions.

Acceptance Criteria:

- The AsaHelpers class should provide a static ConcurrentDictionary for caching SIDs and their corresponding names.
- The cache should be thread-safe and support concurrent access.

Business Rules:

- The cache should improve the performance of SID-to-name conversions by avoiding repeated lookups.

Validations:

- Ensure that the cache is thread-safe and supports concurrent access.
- Ensure that the cache correctly stores and retrieves SIDs and their corresponding names.

Boundary Conditions:

- Handle cases where the cache is empty or does not contain the requested SID.

Non-Functional Requirements (NFRs):

- The cache operations should be efficient and not block the main thread.

User Story 18: Handle Elevated Privileges

As a developer,
I want to check if the application is running with elevated privileges,
so that I can perform actions that require administrative permissions.

Acceptance Criteria:

- The AsaHelpers class should provide a static boolean field _elevated to indicate if the application is running with elevated privileges.
- The _elevated field should be initialized once and cached for subsequent checks.

Business Rules:

- The elevated privileges check should be accurate and reflect the current user's permissions.

Validations:

- Ensure that the _elevated field is correctly initialized based on the user's permissions.
- Ensure that the field is cached and not re-evaluated on subsequent checks.

Boundary Conditions:

- Handle cases where the elevated privileges cannot be determined.

Non-Functional Requirements (NFRs):

- The elevated privileges check should be efficient and not block the main thread

Epic: Database Management for Attack Surface Analyzer

Description

As a developer working on the Attack Surface Analyzer (ASA) project, I want a robust and flexible DatabaseManager class to handle various database operations, so that I can efficiently manage data collection, comparison, and analysis.

User Stories

User Story 1: Destroy Database Files

As a developer,
I want to destroy database files,
so that I can clean up old or unnecessary database files.

Acceptance Criteria:

- The Destroy method should delete all files matching the provided SQLite filename.
- If the directory is not specified, it should default to the current directory.
- The method should handle cases where the directory or files do not exist gracefully.

Business Rules:

- The method should ensure that all matching files are deleted.
- The method should not throw exceptions if the directory or files do not exist.

Validations:

- Ensure that the specified files are deleted.
- Ensure that no exceptions are thrown if the directory or files do not exist.

Boundary Conditions:

- Handle cases where the directory is null or empty.
- Handle cases where no files match the specified filename.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 2: Calculate Modulo of String

As a developer,
I want to calculate the modulo of a string,
so that I can use it for sharding or partitioning data.

Acceptance Criteria:

- The ModuloString method should calculate the sum of the ASCII values of the characters in the string.
- The method should return the result of the sum modulo the specified sharding factor.

Business Rules:

- The method should accurately calculate the sum of the ASCII values.
- The method should handle empty strings gracefully.

Validations:

- Ensure that the method returns the correct modulo value.
- Ensure that the method handles empty strings gracefully.

Boundary Conditions:

- Handle cases where the string is empty.
- Handle cases where the sharding factor is zero.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 3: Begin Database Transaction

As a developer,
I want to begin a database transaction,
so that I can perform multiple operations atomically.

Acceptance Criteria:

- The BeginTransaction method should start a new database transaction.
- The method should ensure that subsequent operations are part of the transaction.

Business Rules:

- The method should ensure that the transaction is started successfully.
- The method should handle cases where a transaction is already in progress.

Validations:

- Ensure that the transaction is started successfully.
- Ensure that subsequent operations are part of the transaction.

Boundary Conditions:

- Handle cases where a transaction is already in progress.
- Handle cases where the database connection is not available.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 4: Close Database Connection

As a developer,
I want to close the database connection,
so that I can release resources and ensure data integrity.

Acceptance Criteria:

- The CloseDatabase method should close the database connection.
- The method should ensure that all pending transactions are committed or rolled back.

Business Rules:

- The method should ensure that the database connection is closed successfully.
- The method should handle cases where the connection is already closed.

Validations:

- Ensure that the database connection is closed successfully.
- Ensure that all pending transactions are committed or rolled back.

Boundary Conditions:

- Handle cases where the connection is already closed.
- Handle cases where there are pending transactions.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 5: Commit Database Transaction

As a developer,
I want to commit a database transaction,
so that I can ensure that all operations within the transaction are saved.

Acceptance Criteria:

- The Commit method should commit the current database transaction.

- The method should ensure that all operations within the transaction are saved.

Business Rules:

- The method should ensure that the transaction is committed successfully.
- The method should handle cases where there is no active transaction.

Validations:

- Ensure that the transaction is committed successfully.
- Ensure that all operations within the transaction are saved.

Boundary Conditions:

- Handle cases where there is no active transaction.
- Handle cases where the commit operation fails.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 6: Delete Run Data

As a developer,
I want to delete data associated with a specific run,
so that I can remove outdated or unnecessary data.

Acceptance Criteria:

- The DeleteRun method should delete all data associated with the specified run ID.
- The method should ensure that the data is removed from all relevant tables.

Business Rules:

- The method should ensure that all data associated with the run ID is deleted.
- The method should handle cases where the run ID does not exist.

Validations:

- Ensure that all data associated with the run ID is deleted.
- Ensure that no exceptions are thrown if the run ID does not exist.

Boundary Conditions:

- Handle cases where the run ID does not exist.
- Handle cases where the database connection is not available.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 7: Delete Compare Run Data

As a developer,
I want to delete data associated with a specific compare run,
so that I can remove outdated or unnecessary comparison data.

Acceptance Criteria:

- The DeleteCompareRun method should delete all data associated with the specified compare run IDs and analysis hash.
- The method should ensure that the data is removed from all relevant tables.

Business Rules:

- The method should ensure that all data associated with the compare run IDs and analysis hash is deleted.
- The method should handle cases where the compare run IDs or analysis hash do not exist.

Validations:

- Ensure that all data associated with the compare run IDs and analysis hash is deleted.
- Ensure that no exceptions are thrown if the compare run IDs or analysis hash do not exist.

Boundary Conditions:

- Handle cases where the compare run IDs or analysis hash do not exist.
- Handle cases where the database connection is not available.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 8: Destroy Database

As a developer,
I want to destroy the database,
so that I can reset the database to a clean state.

Acceptance Criteria:

- The Destroy method should delete all data and reset the database to a clean state.
- The method should ensure that all relevant tables are cleared.

Business Rules:

- The method should ensure that all data is deleted and the database is reset.
- The method should handle cases where the database connection is not available.

Validations:

- Ensure that all data is deleted and the database is reset.

- Ensure that no exceptions are thrown if the database connection is not available.

Boundary Conditions:

- Handle cases where the database connection is not available.
- Handle cases where the database is already in a clean state.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 9: Retrieve All Missing Data

As a developer,
I want to retrieve all missing data between two runs,
so that I can analyze the differences.

Acceptance Criteria:

- The GetAllMissing method should return all data that is present in one run but missing in the other.
- The method should accept two run IDs as parameters.

Business Rules:

- The method should accurately retrieve all missing data between the two runs.
- The method should handle cases where one or both run IDs do not exist.

Validations:

- Ensure that the method returns the correct missing data.
- Ensure that no exceptions are thrown if one or both run IDs do not exist.

Boundary Conditions:

- Handle cases where one or both run IDs do not exist.
- Handle cases where there is no missing data.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 10: Retrieve Common Result Types

As a developer,
I want to retrieve the common result types between two runs,
so that I can understand the similarities in the results.

Acceptance Criteria:

- The GetCommonResultTypes method should return a list of common result types between two runs.

- The method should accept two run IDs as parameters.

Business Rules:

- The method should accurately retrieve the common result types between the two runs.
- The method should handle cases where one or both run IDs do not exist.

Validations:

- Ensure that the method returns the correct common result types.
- Ensure that no exceptions are thrown if one or both run IDs do not exist.

Boundary Conditions:

- Handle cases where one or both run IDs do not exist.
- Handle cases where there are no common result types.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 11: Check if Comparison is Completed

As a developer,
I want to check if a comparison between two runs is completed,
so that I can determine if the comparison results are available.

Acceptance Criteria:

- The GetComparisonCompleted method should return a boolean indicating if the comparison is completed.
- The method should accept two run IDs and an analysis hash as parameters.

Business Rules:

- The method should accurately determine if the comparison is completed.
- The method should handle cases where one or both run IDs or the analysis hash do not exist.

Validations:

- Ensure that the method returns the correct boolean value.
- Ensure that no exceptions are thrown if one or both run IDs or the analysis hash do not exist.

Boundary Conditions:

- Handle cases where one or both run IDs or the analysis hash do not exist.
- Handle cases where the comparison is not completed.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 12: Retrieve Comparison Results

As a developer,
I want to retrieve the comparison results between two runs,
so that I can analyze the differences.

Acceptance Criteria:

- The GetComparisonResults method should return a list of comparison results between two runs.
- The method should accept two run IDs, an analysis hash, and a result type as parameters.
- The method should support pagination with offset and number of results parameters.

Business Rules:

- The method should accurately retrieve the comparison results between the two runs.
- The method should handle cases where one or both run IDs or the analysis hash do not exist.

Validations:

- Ensure that the method returns the correct comparison results.
- Ensure that no exceptions are thrown if one or both run IDs or the analysis hash do not exist.

Boundary Conditions:

- Handle cases where one or both run IDs or the analysis hash do not exist.
- Handle cases where there are no comparison results.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 13: Retrieve Comparison Results Count

As a developer,
I want to retrieve the count of comparison results between two runs,
so that I can understand the volume of differences.

Acceptance Criteria:

- The GetComparisonResultsCount method should return the count of comparison results between two runs.
- The method should accept two run IDs, an analysis hash, and a result type as parameters.

Business Rules:

- The method should accurately retrieve the count of comparison results between the two runs.
- The method should handle cases where one or both run IDs or the analysis hash do not exist.

Validations:

- Ensure that the method returns the correct count of comparison results.
- Ensure that no exceptions are thrown if one or both run IDs or the analysis hash do not exist.

Boundary Conditions:

- Handle cases where one or both run IDs or the analysis hash do not exist.
- Handle cases where there are no comparison results.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 14: Retrieve Current Database Settings

As a developer,
I want to retrieve the current database settings,
so that I can understand the configuration of the database.

Acceptance Criteria:

- The GetCurrentSettings method should return the current database settings.
- The method should ensure that the settings are accurate and up-to-date.

Business Rules:

- The method should accurately retrieve the current database settings.

Validations:

- Ensure that the method returns the correct database settings.
- Ensure that the settings are accurate and up-to-date.

Boundary Conditions:

- Handle cases where the database settings cannot be retrieved.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 15: Retrieve Latest Run IDs

As a developer,
I want to retrieve the latest run IDs,
so that I can access the most recent data.

Acceptance Criteria:

- The GetLatestRunIds method should return a list of the latest run IDs.
- The method should accept the number of IDs to retrieve and the run type as parameters.

Business Rules:

- The method should accurately retrieve the latest run IDs.

Validations:

- Ensure that the method returns the correct latest run IDs.
- Ensure that the IDs are accurate and up-to-date.

Boundary Conditions:

- Handle cases where there are no run IDs available.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 16: Retrieve Missing Data from First Run

As a developer,
I want to retrieve data that is missing from the first run but present in the second run,
so that I can analyze the differences.

Acceptance Criteria:

- The GetMissingFromFirst method should return all data that is present in the second run but missing in the first run.
- The method should accept two run IDs as parameters.

Business Rules:

- The method should accurately retrieve the missing data from the first run.

Validations:

- Ensure that the method returns the correct missing data.
- Ensure that no exceptions are thrown if one or both run IDs do not exist.

Boundary Conditions:

- Handle cases where one or both run IDs do not exist.
- Handle cases where there is no missing data.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 17: Retrieve Modified Data

As a developer,
I want to retrieve data that has been modified between two runs,
so that I can analyze the changes.

Acceptance Criteria:

- The GetModified method should return all data that has been modified between the two runs.
- The method should accept two run IDs as parameters.

Business Rules:

- The method should accurately retrieve the modified data between the two runs.

Validations:

- Ensure that the method returns the correct modified data.
- Ensure that no exceptions are thrown if one or both run IDs do not exist.

Boundary Conditions:

- Handle cases where one or both run IDs do not exist.
- Handle cases where there is no modified data.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 18: Retrieve Monitor Results

As a developer,
I want to retrieve the monitor results for a specific run,
so that I can analyze the monitored data.

Acceptance Criteria:

- The GetMonitorResults method should return the monitor results for the specified run ID.
- The method should support pagination with offset and number of results parameters.

Business Rules:

- The method should accurately retrieve the monitor results for the specified run ID.

Validations:

- Ensure that the method returns the correct monitor results.
- Ensure that no exceptions are thrown if the run ID does not exist.

Boundary Conditions:

- Handle cases where the run ID does not exist.
- Handle cases where there are no monitor results.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 19: Retrieve Monitor Run IDs

As a developer,
I want to retrieve the run IDs for monitor runs,
so that I can access the monitored data.

Acceptance Criteria:

- The GetMonitorRuns method should return a list of run IDs for monitor runs.

Business Rules:

- The method should accurately retrieve the run IDs for monitor runs.

Validations:

- Ensure that the method returns the correct run IDs.
- Ensure that the IDs are accurate and up-to-date.

Boundary Conditions:

- Handle cases where there are no monitor run IDs available.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 20: Retrieve Number of Monitor Results

As a developer,
I want to retrieve the number of monitor results for a specific run,
so that I can understand the volume of monitored data.

Acceptance Criteria:

- The GetNumMonitorResults method should return the number of monitor results for the specified run ID.

Business Rules:

- The method should accurately retrieve the number of monitor results for the specified run ID.

Validations:

- Ensure that the method returns the correct number of monitor results.
- Ensure that no exceptions are thrown if the run ID does not exist.

Boundary Conditions:

- Handle cases where the run ID does not exist.
- Handle cases where there are no monitor results.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 21: Retrieve Number of Results by Type

As a developer,
I want to retrieve the number of results for a specific result type and run,
so that I can understand the volume of data for that type.

Acceptance Criteria:

- The GetNumResults method should return the number of results for the specified result type and run ID.

Business Rules:

- The method should accurately retrieve the number of results for the specified result type and run ID.

Validations:

- Ensure that the method returns the correct number of results.
- Ensure that no exceptions are thrown if the run ID does not exist.

Boundary Conditions:

- Handle cases where the run ID does not exist.
- Handle cases where there are no results for the specified type.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 22: Retrieve Result Models by Status

As a developer,
I want to retrieve the result models for a specific run status,
so that I can analyze the data based on the run status.

Acceptance Criteria:

- The GetResultModels method should return a list of result

Epic: Elevation Utility Functions for Attack Surface Analyzer

Description

As a developer working on the Attack Surface Analyzer (ASA) project, I want a set of utility functions to handle elevation checks and retrieve process integrity levels, so that I can ensure the application runs with the necessary permissions and can provide accurate security assessments.

User Stories

User Story 1: Check if User is in Admin Group

As a developer,
I want to check if the primary access token of the process belongs to a user account that is a member of the local Administrators group,
so that I can determine if the user has administrative privileges even if the process is not elevated.

Acceptance Criteria:

- The IsUserInAdminGroup method should return true if the user is in the local Administrators group.
- The method should handle both elevated and non-elevated tokens.
- The method should throw a Win32Exception if any native Windows API call fails.

Business Rules:

- The method should accurately determine the user's membership in the Administrators group.

Validations:

- Ensure that the method returns true for users in the Administrators group.
- Ensure that the method returns false for users not in the Administrators group.
- Ensure that the method throws a Win32Exception with the correct error code if a native API call fails.

Boundary Conditions:

- Handle cases where the process token cannot be opened.
- Handle cases where the system is running on an OS version prior to Windows Vista.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.
- The method should handle resource cleanup efficiently.

User Story 2: Check if Process is Run as Admin

As a developer,
I want to check if the current process is run as an administrator,
so that I can determine if the process has elevated privileges.

Acceptance Criteria:

- The IsRunAsAdmin method should return true if the process is run as an administrator.
- The method should use the WindowsPrincipal class to check the role.

Business Rules:

- The method should accurately determine if the process is run as an administrator.

Validations:

- Ensure that the method returns true for processes run as an administrator.
- Ensure that the method returns false for processes not run as an administrator.

Boundary Conditions:

- Handle cases where the WindowsIdentity cannot be retrieved.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 3: Check if Process is Elevated

As a developer,
I want to check if the current process is elevated,
so that I can determine if the process has elevated privileges on Windows Vista and newer operating systems.

Acceptance Criteria:

- The IsProcessElevated method should return true if the process is elevated.
- The method should throw a Win32Exception if any native Windows API call fails.
- The method should handle systems running on OS versions prior to Windows Vista.

Business Rules:

- The method should accurately determine if the process is elevated.

Validations:

- Ensure that the method returns true for elevated processes.
- Ensure that the method returns false for non-elevated processes.
- Ensure that the method throws a Win32Exception with the correct error code if a native API call fails.

Boundary Conditions:

- Handle cases where the process token cannot be opened.
- Handle cases where the system is running on an OS version prior to Windows Vista.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.
- The method should handle resource cleanup efficiently.

User Story 4: Get Process Integrity Level

As a developer,
I want to retrieve the integrity level of the current process,
so that I can determine the security level of the process on Windows Vista and newer operating systems.

Acceptance Criteria:

- The GetProcessIntegrityLevel method should return the integrity level of the process.
- The method should throw a Win32Exception if any native Windows API call fails.
- The method should handle systems running on OS versions prior to Windows Vista.

Business Rules:

- The method should accurately determine the process integrity level.

Validations:

- Ensure that the method returns the correct integrity level for the process.
- Ensure that the method throws a Win32Exception with the correct error code if a native API call fails.

Boundary Conditions:

- Handle cases where the process token cannot be opened.
- Handle cases where the system is running on an OS version prior to Windows Vista.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.
- The method should handle resource cleanup efficiently.

User Story 5: Query Elevation Status

As a developer,
I want to query if the current process is run as an administrator,
so that I can determine the elevation status of the process.

Acceptance Criteria:

- The QueryElevation method should return true if the process is run as an administrator.
- The method should use the IsRunAsAdmin method internally.

Business Rules:

- The method should accurately determine the elevation status of the process.

Validations:

- Ensure that the method returns true for processes run as an administrator.
- Ensure that the method returns false for processes not run as an administrator.

Boundary Conditions:

- Handle cases where the IsRunAsAdmin method fails.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 6: Check if Running as Root

As a developer,
I want to check if the current process is running as the root user on Unix-like systems,
so that I can determine if the process has elevated privileges.

Acceptance Criteria:

- The IsRunningAsRoot method should return true if the process is running as the root user.
- The method should use the whoami command to determine the current user.

Business Rules:

- The method should accurately determine if the process is running as the root user.

Validations:

- Ensure that the method returns true for processes running as the root user.
- Ensure that the method returns false for processes not running as the root user.

Boundary Conditions:

- Handle cases where the whoami command fails or returns an unexpected result.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

User Story 7: Check if User is Administrator

As a developer,
I want to check if the current user is an administrator,
so that I can determine if the user has administrative privileges.

Acceptance Criteria:

- The IsAdministrator method should return true if the user is an administrator.
- The method should use the WindowsPrincipal class to check the role.

- The method should handle PlatformNotSupportedException gracefully.

Business Rules:

- The method should accurately determine if the user is an administrator.

Validations:

- Ensure that the method returns true for users who are administrators.
- Ensure that the method returns false for users who are not administrators.
- Ensure that the method handles PlatformNotSupportedException gracefully.

Boundary Conditions:

- Handle cases where the WindowsIdentity cannot be retrieved.

Non-Functional Requirements (NFRs):

- The method should execute quickly and not block the main thread.

Non-Functional Requirements (NFRs) for All User Stories

- **Performance**: All methods should execute quickly and not block the main thread.
- **Resource Management**: All methods should handle resource cleanup efficiently.
- **Error Handling**: All methods should handle exceptions gracefully and provide meaningful error messages.
- **Platform Compatibility**: Methods should handle platform-specific differences and provide fallbacks where necessary.
- **Security**: Methods should not expose sensitive information and should follow best practices for security.

Business Rules for All User Stories

- **Accuracy**: All methods should provide accurate and reliable results based on the current environment and user permissions.
- **Consistency**: Methods should follow a consistent approach to handle elevation checks and process integrity levels.
- **Compliance**: Methods should comply with relevant security and coding standards.

Validations for All User Stories

- **Input Validation**: Ensure that all inputs are valid and handle invalid inputs gracefully.
- **Output Validation**: Ensure that all outputs are correct and meet the expected criteria.
- **Exception Handling**: Ensure that all exceptions are handled gracefully and provide meaningful error messages.

Boundary Conditions for All User Stories

- **Edge Cases**: Handle edge cases such as null inputs, unsupported platforms, and unexpected results from native API calls.
- **Resource Availability**: Handle cases where required resources (e.g., process tokens, memory) are not available.
- **Platform Differences**: Handle platform-specific differences and provide fallbacks where necessary.

Epic: Enhance SqliteDatabaseManager Functionality

Description

As a developer working on the Attack Surface Analyzer (ASA) project, I want to enhance the SqliteDatabaseManager class to improve database management, data retrieval, and performance, so that I can ensure efficient and reliable operations within the application.

User Stories

User Story 1: Initialize SqliteDatabaseManager with Custom Settings

As a developer,
I want to initialize the SqliteDatabaseManager with custom database settings,
so that I can configure the database manager according to specific requirements.

Acceptance Criteria:

- The SqliteDatabaseManager constructor should accept an optional DBSettings parameter.
- If no DBSettings parameter is provided, default settings should be used.
- The database file location should be set based on the provided filename or default to asa.sqlite in the current directory.

Business Rules:

- The database settings should be configurable and allow for customization.

Validations:

- Ensure that the DBSettings parameter is correctly applied.
- Ensure that the database file location is valid and accessible.

Boundary Conditions:

- Handle cases where the filename is null or empty.
- Handle cases where the provided DBSettings parameter is null.

Non-Functional Requirements (NFRs):

- The initialization should be efficient and not block the main thread.

User Story 2: Manage Database Connections

As a developer,
I want to manage multiple database connections,
so that I can handle concurrent database operations efficiently.

Acceptance Criteria:

- The Connections property should maintain a list of SqlConnectionHolder objects.
- The EstablishMainConnection method should establish the main database connection if none exists.
- The PopulateConnections method should create additional connections based on the sharding factor.

Business Rules:

- The database connections should be managed efficiently to support concurrent operations.

Validations:

- Ensure that the main connection is established correctly.
- Ensure that additional connections are created based on the sharding factor.

Boundary Conditions:

- Handle cases where the main connection cannot be established.
- Handle cases where the sharding factor is set to a low or high value.

Non-Functional Requirements (NFRs):

- The connection management should be efficient and not block the main thread.

User Story 3: Perform Database Transactions

As a developer,
I want to perform database transactions,
so that I can ensure data integrity during complex operations.

Acceptance Criteria:

- The BeginTransaction, Commit, and RollBack methods should manage transactions across all connections.
- The BeginTransaction method should start a transaction on all connections.
- The Commit method should commit the transaction on all connections.
- The RollBack method should roll back the transaction on all connections.

Business Rules:

- Transactions should ensure data integrity and consistency.

Validations:

- Ensure that transactions are started, committed, and rolled back correctly.
- Ensure that exceptions during transactions are handled gracefully.

Boundary Conditions:

- Handle cases where a transaction cannot be started, committed, or rolled back.

Non-Functional Requirements (NFRs):

- The transaction management should be efficient and not block the main thread.

User Story 4: Retrieve and Delete Run Data

As a developer,
I want to retrieve and delete run data from the database,
so that I can manage the data associated with specific runs.

Acceptance Criteria:

- The GetRuns, GetRuns(RUN_TYPE type), and DeleteRun methods should retrieve and delete run data.
- The GetRuns method should return a list of all run IDs.
- The GetRuns(RUN_TYPE type) method should return a list of run IDs filtered by type.
- The DeleteRun method should delete all data associated with a specific run ID.

Business Rules:

- Run data should be managed efficiently to support data retrieval and deletion.

Validations:

- Ensure that run data is retrieved and deleted correctly.
- Ensure that exceptions during data retrieval and deletion are handled gracefully.

Boundary Conditions:

- Handle cases where the run ID does not exist.
- Handle cases where the run type is invalid.

Non-Functional Requirements (NFRs):

- The data retrieval and deletion should be efficient and not block the main thread.

User Story 5: Retrieve Comparison Results

As a developer,
I want to retrieve comparison results between runs,
so that I can analyze the differences between different runs.

Acceptance Criteria:

- The GetComparisonResults and GetComparisonResultsCount methods should retrieve comparison results.
- The GetComparisonResults method should return a list of CompareResult objects based on the provided parameters.
- The GetComparisonResultsCount method should return the count of comparison results based on the provided parameters.

Business Rules:

- Comparison results should be retrieved efficiently to support data analysis.

Validations:

- Ensure that comparison results are retrieved correctly.
- Ensure that exceptions during data retrieval are handled gracefully.

Boundary Conditions:

- Handle cases where the comparison results do not exist.
- Handle cases where the provided parameters are invalid.

Non-Functional Requirements (NFRs):

- The data retrieval should be efficient and not block the main thread.

User Story 6: Insert and Update Run Data

As a developer,
I want to insert and update run data in the database,
so that I can manage the data associated with specific runs.

Acceptance Criteria:

- The InsertRun, InsertCompareRun, and UpdateCompareRun methods should insert and update run data.
- The InsertRun method should insert a new run into the database.
- The InsertCompareRun method should insert a new comparison run into the database.
- The UpdateCompareRun method should update the status of an existing comparison run.

Business Rules:

- Run data should be managed efficiently to support data insertion and updates.

Validations:

- Ensure that run data is inserted and updated correctly.
- Ensure that exceptions during data insertion and updates are handled gracefully.

Boundary Conditions:

- Handle cases where the run data already exists.
- Handle cases where the provided parameters are invalid.

Non-Functional Requirements (NFRs):

- The data insertion and updates should be efficient and not block the main thread.

User Story 7: Manage Database Settings

As a developer,
I want to manage the database settings,
so that I can configure the database manager according to specific requirements.

Acceptance Criteria:

- The GetCurrentSettings and SetSettings methods should manage the database settings.
- The GetCurrentSettings method should return the current database settings.
- The SetSettings method should update the database settings.

Business Rules:

- The database settings should be configurable and allow for customization.

Validations:

- Ensure that the database settings are retrieved and updated correctly.
- Ensure that exceptions during settings management are handled gracefully.

Boundary Conditions:

- Handle cases where the settings cannot be retrieved or updated.

Non-Functional Requirements (NFRs):

- The settings management should be efficient and not block the main thread.

User Story 8: Perform Database Maintenance

As a developer,
I want to perform database maintenance tasks,
so that I can ensure the database operates efficiently.

Acceptance Criteria:

- The Vacuum and TrimToLatest methods should perform database maintenance tasks.
- The Vacuum method should optimize the database by reclaiming unused space.
- The TrimToLatest method should delete old run data, keeping only the latest run.

Business Rules:

- Database maintenance tasks should be performed efficiently to support optimal database operations.

Validations:

- Ensure that database maintenance tasks are performed correctly.
- Ensure that exceptions during maintenance tasks are handled gracefully.

Boundary Conditions:

- Handle cases where the database maintenance tasks cannot be performed.

Non-Functional Requirements (NFRs):

- The maintenance tasks should be efficient and not block the main thread.

User Story 9: Handle Low Memory Usage Mode

As a developer,
I want to handle low memory usage mode,
so that I can ensure the application operates efficiently under low memory conditions.

Acceptance Criteria:

- The StallIfHighMemoryUsageAndLowMemoryModeEnabled method should handle low memory usage mode.
- The method should stall the collector if the queue size exceeds the low memory cutoff.

Business Rules:

- The low memory usage mode should be handled efficiently to support optimal application performance.

Validations:

- Ensure that the method stalls the collector correctly under low memory conditions.
- Ensure that exceptions during low memory handling are handled gracefully.

Boundary Conditions:

- Handle cases where the low memory usage mode is not enabled.

Non-Functional Requirements (NFRs):

- The low memory handling should be efficient and not block the main thread.

User Story 10: Generate SQL Commands

As a developer,
I want to generate SQL commands for various database operations,
so that I can perform database operations efficiently.

Acceptance Criteria:

- The class should define constants for various SQL commands used in database operations.
- The SQL commands should be used in the appropriate methods for database operations.

Business Rules:

- The SQL commands should be defined and used efficiently to support database operations.

Validations:

- Ensure that the SQL commands are defined correctly.
- Ensure that the SQL commands are used correctly in the appropriate methods.

Boundary Conditions:

- Handle cases where the SQL commands cannot be executed.

Non-Functional Requirements (NFRs):

- The SQL command generation should be efficient and not block the main thread.

Non-Functional Requirements (NFRs) for All User Stories

- **Performance**: All methods should execute quickly and not block the main thread.
- **Resource Management**: All methods should handle resource cleanup efficiently.
- **Error Handling**: All methods should handle exceptions gracefully and provide meaningful error messages.
- **Platform Compatibility**: Methods should handle platform-specific differences and provide fallbacks where necessary.
- **Security**: Methods should not expose sensitive information and should follow best practices for security.

Business Rules for All User Stories

- **Accuracy**: All methods should provide accurate and reliable results based on the current environment and user permissions.
- **Consistency**: Methods should follow a consistent approach to handle database operations and settings.
- **Compliance**: Methods should comply with relevant security and coding standards.

Validations for All User Stories

- **Input Validation**: Ensure that all inputs are valid and handle invalid inputs gracefully.
- **Output Validation**: Ensure that all outputs are correct and meet the expected criteria.

- **Exception Handling**: Ensure that all exceptions are handled gracefully and provide meaningful error messages.

Boundary Conditions for All User Stories

- **Edge Cases**: Handle edge cases such as null inputs, unsupported platforms, and unexpected results from database operations.
- **Resource Availability**: Handle cases where required resources (e.g., database connections, memory) are not available.
- **Platform Differences**: Handle platform-specific differences and provide fallbacks where necessary. **Epic: Enhance Analyze Page Functionality**
- **Description**
- As a user of the Attack Surface Analyzer (ASA) web application, I want to enhance the functionality of the Analyze page to improve the user experience, ensure accurate data analysis, and handle various states effectively, so that I can perform and review analysis scans efficiently.
- **User Stories**
- **User Story 1: Display Analysis Options**
- **As a** user,
  I want to see analysis options when I navigate to the Analyze page,
  so that I can select the runs I want to analyze.
- **Acceptance Criteria:**
- The page should display a form with dropdowns for selecting Run ID and Second Run ID.
- The dropdowns should be populated with the list of available runs.
- The form should include a button to start the analysis scan.
- **Business Rules:**
- The Run ID and Second Run ID dropdowns should be populated with valid run IDs from the database.
- The Second Run ID is optional and can be left unselected.
- **Validations:**
- Ensure that the dropdowns are populated with valid run IDs.
- Ensure that the Run ID and Second Run ID are correctly bound to the respective properties.
- **Boundary Conditions:**
- Handle cases where there are no runs available in the database.

- Handle cases where the Run ID or Second Run ID is not selected.
- **Non-Functional Requirements (NFRs):**
- The page should load quickly and not block the main thread.
- The form should be responsive and accessible.
- **User Story 2: Start Analysis Scan**
- **As a** user,
  I want to start an analysis scan by clicking a button,
  so that I can initiate the comparison of selected runs.
- **Acceptance Criteria:**
- The Start Analysis Scan button should trigger the BeginAnalyze method.
- The BeginAnalyze method should update the page state to Analyzing.
- The analysis scan should be performed asynchronously.
- **Business Rules:**
- The analysis scan should compare the selected runs and store the results in the database.
- **Validations:**
- Ensure that the BeginAnalyze method is triggered when the button is clicked.
- Ensure that the page state is updated to Analyzing.
- Ensure that the analysis scan is performed asynchronously.
- **Boundary Conditions:**
- Handle cases where the analysis scan fails or encounters an error.
- **Non-Functional Requirements (NFRs):**
- The analysis scan should be performed efficiently and not block the main thread.
- The user should receive feedback on the progress of the analysis scan.
- **User Story 3: Display Analyzing State**
- **As a** user,
  I want to see a visual indication that the analysis scan is in progress,
  so that I know the application is working on my request.
- **Acceptance Criteria:**
- The page should display an Analyzing component when the page state is Analyzing.
- The Analyzing component should provide a visual indication of progress.
- **Business Rules:**

- The Analyzing component should be displayed only when the page state is Analyzing.
- **Validations:**
- Ensure that the Analyzing component is displayed when the page state is Analyzing.
- Ensure that the Analyzing component provides a clear visual indication of progress.
- **Boundary Conditions:**
- Handle cases where the analysis scan takes longer than expected.
- **Non-Functional Requirements (NFRs):**
- The Analyzing component should be responsive and not block the main thread.
- The user should receive real-time feedback on the progress of the analysis scan.
- **User Story 4: Display Analysis Results**
- **As a** user,
  I want to see the results of the analysis scan when it is finished,
  so that I can review the comparison of the selected runs.
- **Acceptance Criteria:**
- The page should display a button to analyze again when the page state is Finished.
- The analysis results should be stored in the database and accessible for review.
- **Business Rules:**
- The analysis results should be accurate and reflect the comparison of the selected runs.
- **Validations:**
- Ensure that the button to analyze again is displayed when the page state is Finished.
- Ensure that the analysis results are stored correctly in the database.
- **Boundary Conditions:**
- Handle cases where the analysis results cannot be stored in the database.
- **Non-Functional Requirements (NFRs):**
- The results display should be responsive and not block the main thread.
- The user should be able to review the results efficiently.
- **User Story 5: Handle Analysis Errors**
- **As a** user,
  I want to see an error message if the analysis scan fails,
  so that I know there was an issue with my request.
- **Acceptance Criteria:**

- The page should display an error message when the page state is Error.
- The error message should provide information about the issue.
- **Business Rules:**
- The error message should be displayed only when the page state is Error.
- **Validations:**
- Ensure that the error message is displayed when the page state is Error.
- Ensure that the error message provides clear information about the issue.
- **Boundary Conditions:**
- Handle cases where the error message cannot be displayed.
- **Non-Functional Requirements (NFRs):**
- The error message display should be responsive and not block the main thread.
- The user should receive clear and actionable information about the issue.
- **User Story 6: Reset to Options State**
- **As a** user,
  I want to reset the page to the options state after an analysis scan,
  so that I can perform another analysis if needed.
- **Acceptance Criteria:**
- The Analyze Again button should trigger the GoToOptions method.
- The GoToOptions method should update the page state to Options.
- **Business Rules:**
- The page should be reset to the options state to allow for a new analysis scan.
- **Validations:**
- Ensure that the GoToOptions method is triggered when the button is clicked.
- Ensure that the page state is updated to Options.
- **Boundary Conditions:**
- Handle cases where the page state cannot be updated.
- **Non-Functional Requirements (NFRs):**
- The page reset should be efficient and not block the main thread.
- The user should be able to start a new analysis scan quickly.
- **Non-Functional Requirements (NFRs) for All User Stories**
- **Performance**: All methods should execute quickly and not block the main thread.

- **Resource Management**: All methods should handle resource cleanup efficiently.
- **Error Handling**: All methods should handle exceptions gracefully and provide meaningful error messages.
- **User Experience**: The user interface should be responsive and provide clear feedback to the user.
- **Accessibility**: The user interface should be accessible to users with disabilities.
- **Security**: Methods should not expose sensitive information and should follow best practices for security.
- **Business Rules for All User Stories**
- **Accuracy**: All methods should provide accurate and reliable results based on the current environment and user inputs.
- **Consistency**: Methods should follow a consistent approach to handle page states and user interactions.
- **Compliance**: Methods should comply with relevant security and coding standards.
- **Validations for All User Stories**
- **Input Validation**: Ensure that all inputs are valid and handle invalid inputs gracefully.
- **Output Validation**: Ensure that all outputs are correct and meet the expected criteria.
- **Exception Handling**: Ensure that all exceptions are handled gracefully and provide meaningful error messages.
- **Boundary Conditions for All User Stories**
- **Edge Cases**: Handle edge cases such as null inputs, unsupported platforms, and unexpected results from database operations.
- **Resource Availability**: Handle cases where required resources (e.g., database connections, memory) are not available.
- **Platform Differences**: Handle platform-specific differences and provide fallbacks where necessary.
- **Epic: Enhance Sandbox Page Functionality**
- **Description**
- As a user of the Attack Surface Analyzer (ASA) web application, I want to enhance the functionality of the Sandbox page to improve the user experience, ensure accurate data handling, and provide robust error management, so that I can create, view, and manage sandbox objects efficiently.
- **User Stories**
- **User Story 1: Display Sandbox Tabs**

- **As a** user,
  I want to see tabs for creating, viewing, and managing sandbox objects,
  so that I can easily navigate between different functionalities.
- **Acceptance Criteria:**
- The page should display three tabs: Create, View, and Manage.
- Each tab should be associated with its respective content section.
- The Create tab should be active by default.
- **Business Rules:**
- The tabs should be clearly labeled and accessible.
- **Validations:**
- Ensure that the tabs are displayed correctly.
- Ensure that the correct content section is displayed when a tab is selected.
- **Boundary Conditions:**
- Handle cases where there are no sandbox objects to display.
- **Non-Functional Requirements (NFRs):**
- The tabs should be responsive and accessible.
- The page should load quickly and not block the main thread.
- **User Story 2: Create Sandbox Objects**
- **As a** user,
  I want to create sandbox objects by selecting a type and clicking a button,
  so that I can add new objects to the sandbox.
- **Acceptance Criteria:**
- The Create tab should display a dropdown for selecting the type of object to create.
- The dropdown should be populated with available types.
- The tab should include a button to add the selected object.
- The tab should include a button to remove the last object, which should be disabled if there are no objects.
- **Business Rules:**
- The dropdown should be populated with valid types from the Types dictionary.
- The Add button should create an object of the selected type and add it to AppState.TestObjects.
- **Validations:**

- Ensure that the dropdown is populated with valid types.
- Ensure that the Add button creates and adds the object correctly.
- Ensure that the Remove button removes the last object correctly.
- **Boundary Conditions:**
- Handle cases where the selected type is invalid or not found.
- Handle cases where there are no objects to remove.
- **Non-Functional Requirements (NFRs):**
- The object creation should be efficient and not block the main thread.
- The user interface should be responsive and accessible.
- **User Story 3: View Sandbox Objects**
- **As a** user,
  I want to view and analyze sandbox objects,
  so that I can see the details and analysis results of each object.
- **Acceptance Criteria:**
- The View tab should display a list of sandbox objects.
- Each object should be displayed with its details and analysis results.
- The tab should include a button to remove the last object, which should be disabled if there are no objects.
- **Business Rules:**
- The analysis results should be accurate and reflect the current state of the objects.
- **Validations:**
- Ensure that the objects and their details are displayed correctly.
- Ensure that the analysis results are accurate.
- Ensure that the Remove button removes the last object correctly.
- **Boundary Conditions:**
- Handle cases where there are no objects to display.
- Handle cases where the analysis results are not available.
- **Non-Functional Requirements (NFRs):**
- The object display and analysis should be efficient and not block the main thread.
- The user interface should be responsive and accessible.
- **User Story 4: Manage Sandbox State**

- **As a** user,
  I want to load and save the sandbox state,
  so that I can persist and restore the state of the sandbox objects.
- **Acceptance Criteria:**
- The Manage tab should include a file upload component to load the sandbox state from a JSON file.
- The tab should include a button to save the current sandbox state to a JSON file.
- The tab should include a button to clear the sandbox state.
- **Business Rules:**
- The sandbox state should be loaded and saved accurately.
- The clear button should remove all objects and errors from the sandbox state.
- **Validations:**
- Ensure that the sandbox state is loaded correctly from the JSON file.
- Ensure that the sandbox state is saved correctly to the JSON file.
- Ensure that the clear button removes all objects and errors correctly.
- **Boundary Conditions:**
- Handle cases where the JSON file is invalid or cannot be read.
- Handle cases where there are no objects to clear.
- **Non-Functional Requirements (NFRs):**
- The state management should be efficient and not block the main thread.
- The user interface should be responsive and accessible.
- **User Story 5: Handle Errors Gracefully**
- **As a** user,
  I want to see error messages when something goes wrong,
  so that I can understand and address issues with the sandbox operations.
- **Acceptance Criteria:**
- The page should display error messages in an alert component when errors occur.
- The error messages should be clear and provide information about the issue.
- **Business Rules:**
- Error messages should be displayed only when errors occur.
- **Validations:**
- Ensure that error messages are displayed correctly when errors occur.

- Ensure that error messages provide clear and actionable information.
- **Boundary Conditions:**
- Handle cases where multiple errors occur simultaneously.
- **Non-Functional Requirements (NFRs):**
- The error handling should be efficient and not block the main thread.
- The user interface should be responsive and accessible.
- **User Story 6: Refresh Page State**
- **As a** user,
  I want the page state to refresh automatically when changes occur,
  so that I can see the latest state of the sandbox objects.
- **Acceptance Criteria:**
- The page should refresh automatically when objects are added, removed, or modified.
- The StateHasChanged method should be called to refresh the page state.
- **Business Rules:**
- The page state should be kept up-to-date with the latest changes.
- **Validations:**
- Ensure that the page state is refreshed correctly when changes occur.
- Ensure that the StateHasChanged method is called appropriately.
- **Boundary Conditions:**
- Handle cases where multiple changes occur simultaneously.
- **Non-Functional Requirements (NFRs):**
- The page refresh should be efficient and not block the main thread.
- The user interface should be responsive and accessible.
- **Non-Functional Requirements (NFRs) for All User Stories**
- **Performance**: All methods should execute quickly and not block the main thread.
- **Resource Management**: All methods should handle resource cleanup efficiently.
- **Error Handling**: All methods should handle exceptions gracefully and provide meaningful error messages.
- **User Experience**: The user interface should be responsive and provide clear feedback to the user.
- **Accessibility**: The user interface should be accessible to users with disabilities.

- **Security**: Methods should not expose sensitive information and should follow best practices for security.
- **Business Rules for All User Stories**
- **Accuracy**: All methods should provide accurate and reliable results based on the current environment and user inputs.
- **Consistency**: Methods should follow a consistent approach to handle page states and user interactions.
- **Compliance**: Methods should comply with relevant security and coding standards.
- **Validations for All User Stories**
- **Input Validation**: Ensure that all inputs are valid and handle invalid inputs gracefully.
- **Output Validation**: Ensure that all outputs are correct and meet the expected criteria.
- **Exception Handling**: Ensure that all exceptions are handled gracefully and provide meaningful error messages.
- **Boundary Conditions for All User Stories**
- **Edge Cases**: Handle edge cases such as null inputs, unsupported platforms, and unexpected results from database operations.
- **Resource Availability**: Handle cases where required resources (e.g., database connections, memory) are not available.
- **Platform Differences**: Handle platform-specific differences and provide fallbacks where necessary.
-