International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 06 | June - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Automated Security Testing Tool

L Mojesh

Objective: Develop a tool that automates security testing for web applications using Flask (for the tool's UI and control) and Selenium (for browser automation and interaction). The tool will navigate through a web application, perform security scans (leveraging external tools or libraries), and generate reports summarizing the findings.

Detailed Description:

This project aims to create a user-friendly web application that simplifies and automates common security testing tasks for web applications. It will provide a centralized interface to configure tests, launch scans, and view results. The core functionality will rely on Selenium to interact with the target web application and integrate with various security scanning tools or libraries to identify potential vulnerabilities.

Key Features:

1. **Web-Based User Interface (Flask):**

- **Target Configuration:** Allow users to input the URL of the web application to be tested.
- **Test Selection:** Provide options to select different types of security tests to perform (e.g., basic navigation checks, form submissions, specific vulnerability scans).
- **Authentication Management:** Enable users to configure authentication credentials (username/password, cookies, etc.) required to access protected parts of the application.
- **Scan Configuration:** Allow customization of scan parameters (e.g., scan depth, specific checks to enable/disable for integrated tools).
- **Real-time Status Monitoring:** Display the progress of the automated tests in real-time.
- **Report Generation:** Present the security scan results in a clear and organized format.
- **History and Management:** Allow users to view past scan results and manage saved configurations.

2. **Web Application Interaction (Selenium):**

- **Navigation:** Selenium will be used to programmatically navigate through the target web application's pages and links.
- **Form Handling:** Automate the submission of forms with various inputs (including potentially malicious payloads for testing).
- **State Management:** Maintain session and cookie information as it navigates through the application.
- **Dynamic Content Handling:** Interact with web pages that use JavaScript to dynamically load content.

3. **Security Scanning Integration:**

- **Passive Analysis:** While navigating with Selenium, the tool can perform passive analysis by:
  - Capturing HTTP requests and responses for manual review or later automated analysis.
  - Analyzing client-side code (JavaScript, HTML, CSS) for potential issues.
- **Integration with External Security Tools (Command-Line Interface):** The tool will be designed to integrate with existing command-line security scanners (e.g., OWASP ZAP, Nikto, Arachni) by:

- Dynamically generating configuration files or command-line arguments based on user input.
- Executing these tools against the target application (likely running them in the background).
- Parsing the output reports generated by these tools.

o **Integration with Python Security Libraries:** Explore the possibility of directly using Python security libraries for specific checks (e.g., for common web vulnerabilities).

4. **Reporting:**

o **Structured Output:** Present the findings in a well-organized manner, including:

- Vulnerability Description: A clear explanation of the identified security issue.
- Location/Affected URL: The specific part of the web application where the issue was found.
- Severity Level: Categorization of the risk (e.g., High, Medium, Low).
- Evidence: Details from the scan output or captured requests/responses that demonstrate the vulnerability.
- Recommendations: Basic guidance on how to remediate the identified issue (potentially linking to OWASP resources).

o **Multiple Formats:** Support generating reports in different formats (e.g., HTML, PDF, CSV).

o **Customization:** Allow some level of customization in the report generation process.

Working Mechanism:

1. **User Interaction (Flask UI):**

o The user opens the web interface of the automated security testing tool in their browser.

o They configure the target web application URL, authentication details (if required), and select the desired security tests or integrated tools.

o They might also be able to customize scan parameters specific to the chosen tests/tools.

o The user initiates the scan through the web interface.

2. **Test Execution (Flask Backend & Selenium):**

o The Flask backend receives the user's configuration.

o **Selenium Automation:** If basic navigation or interaction is required for passive analysis or to prepare the application state for external tools, the Flask backend will instruct the Selenium WebDriver to:

- Launch a browser instance (e.g., Chrome, Firefox).
- Navigate to the target URL.
- Handle authentication by filling forms or setting cookies.
- Explore different parts of the application by following links and submitting forms.
- Capture HTTP requests and responses during navigation.

o **External Tool Integration:**

- The Flask backend will generate the necessary command-line arguments or configuration files for the selected external security scanner (e.g., zap-cli, nikto.pl).

- It will then execute the external tool as a separate process, targeting the configured web application URL (and potentially providing authentication details).
- The Flask backend will monitor the progress of the external tool (if possible).

3. **Result Processing:**

- **Selenium Data:** The Flask backend will analyze the HTTP requests/responses captured by Selenium for potential security headers, status codes, and other basic indicators.
- **External Tool Output Parsing:** Once the external security scanner completes, the Flask backend will:
  - Read the report file generated by the tool (e.g., XML, JSON, text).
  - Parse the report to extract relevant security findings (vulnerability name, description, URL, severity, evidence).
- **Python Library Analysis:** If Python security libraries are used directly, their output will be processed accordingly.

4. **Report Generation (Flask Backend & UI):**

- The Flask backend will aggregate all the security findings from the Selenium analysis and the parsed output of the external tools.
- It will structure this information into a user-friendly format.
- The web interface will display the generated report, allowing users to view the details of each identified vulnerability.
- Options to download the report in different formats might be provided.

Technology Stack:

- **Backend & UI:** Flask (Python web microframework)
- **Browser Automation:** Selenium (Python bindings for WebDriver)
- **Security Scanning Tools (Integration):** OWASP ZAP, Nikto, Arachni (or others as needed)
- **Python Security Libraries (Optional):** requests, beautifulsoup4 (for basic analysis), potentially others for specific vulnerability checks.
- **Reporting:** HTML, Jinja2 templating (for dynamic HTML reports), potentially libraries for PDF/CSV generation.

Development Steps (High-Level):

1. **Set up the Flask Application:** Create the basic web application structure with routes for configuration, scan initiation, and report display.
2. **Implement Selenium Integration:** Develop the functionality to navigate and interact with web applications using Selenium WebDriver.
3. **Integrate with External Security Tools:** Implement the logic to execute command- line security scanners and parse their output reports.
4. **Develop Basic Passive Analysis:** Implement checks on HTTP headers and client-side code using Selenium's capabilities.
5. **Design and Implement the Reporting Module:** Create the structure and templates for displaying and exporting security findings.
6. **Implement Authentication Management:** Allow users to configure and manage authentication for target

applications.

7. **Add Scan Configuration Options:** Provide users with granular control over the tests and tools being used.

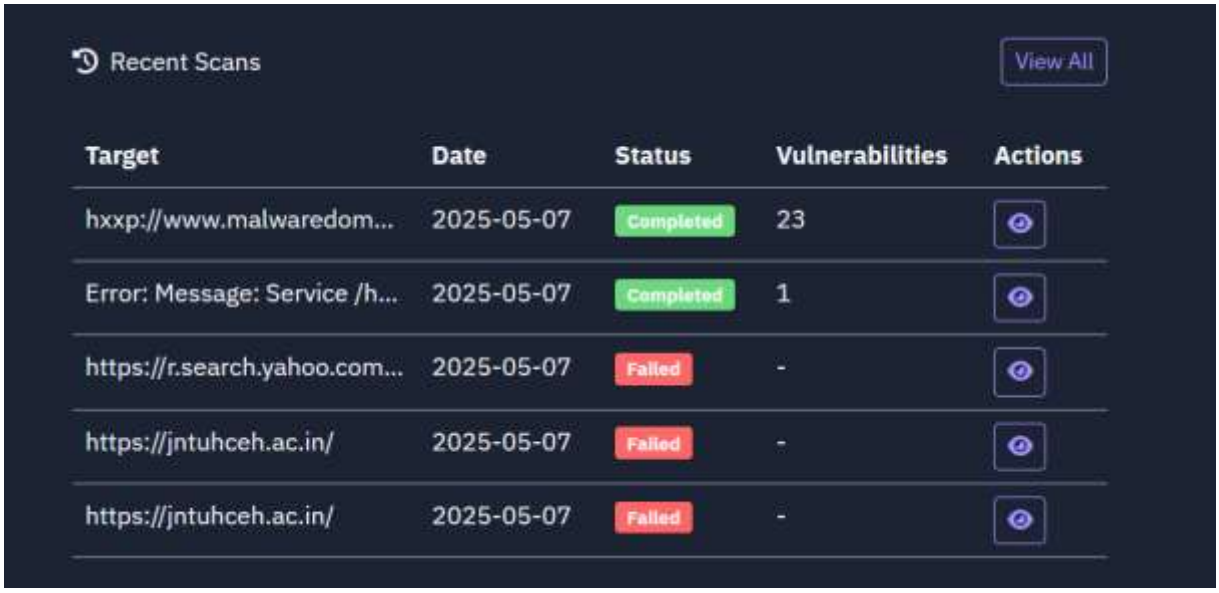
8. **Implement Real-time Status Monitoring:** Provide feedback to the user during the scan process.

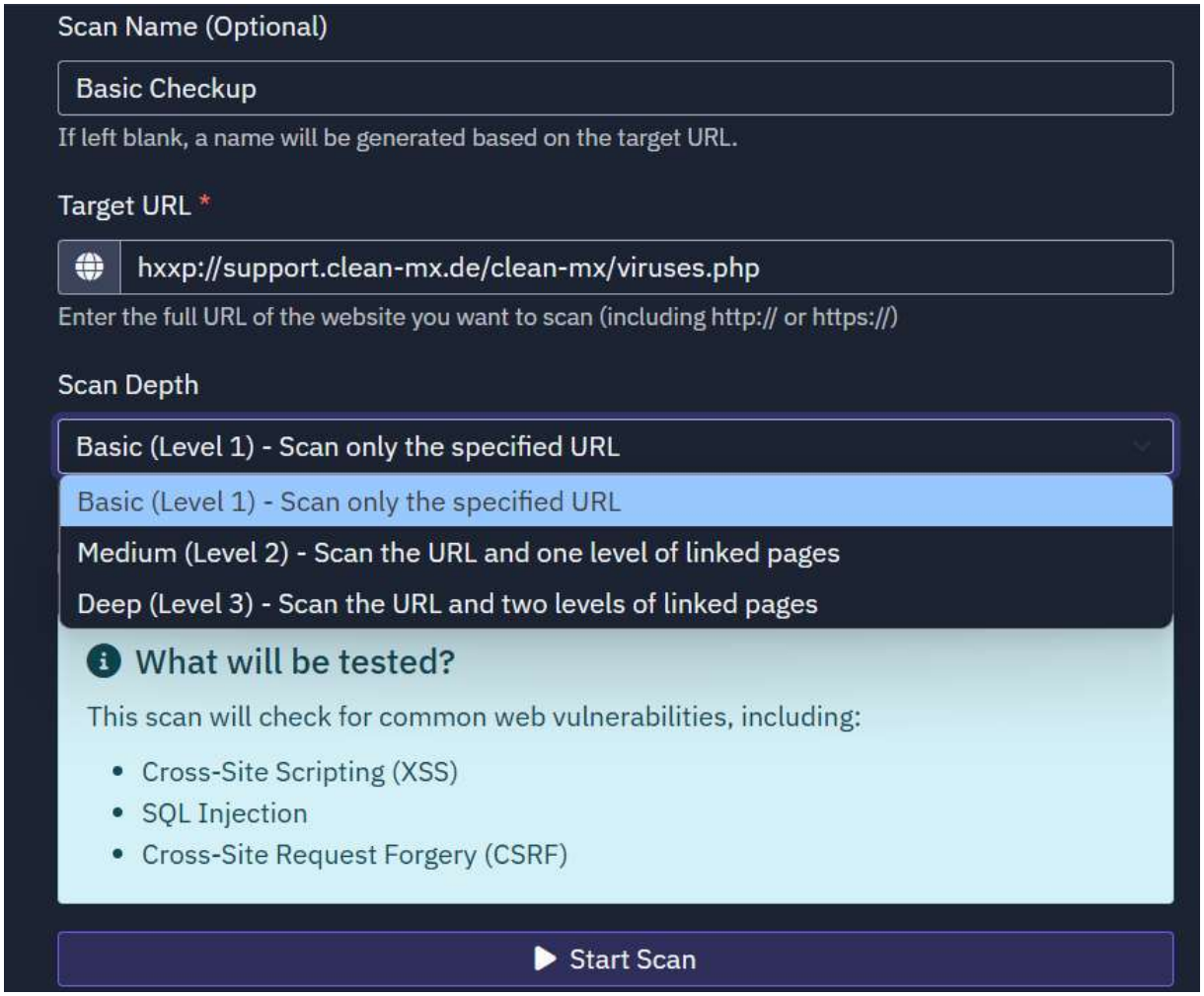
9. **Implement History and Management:** Allow users to view past scans and manage configurations.

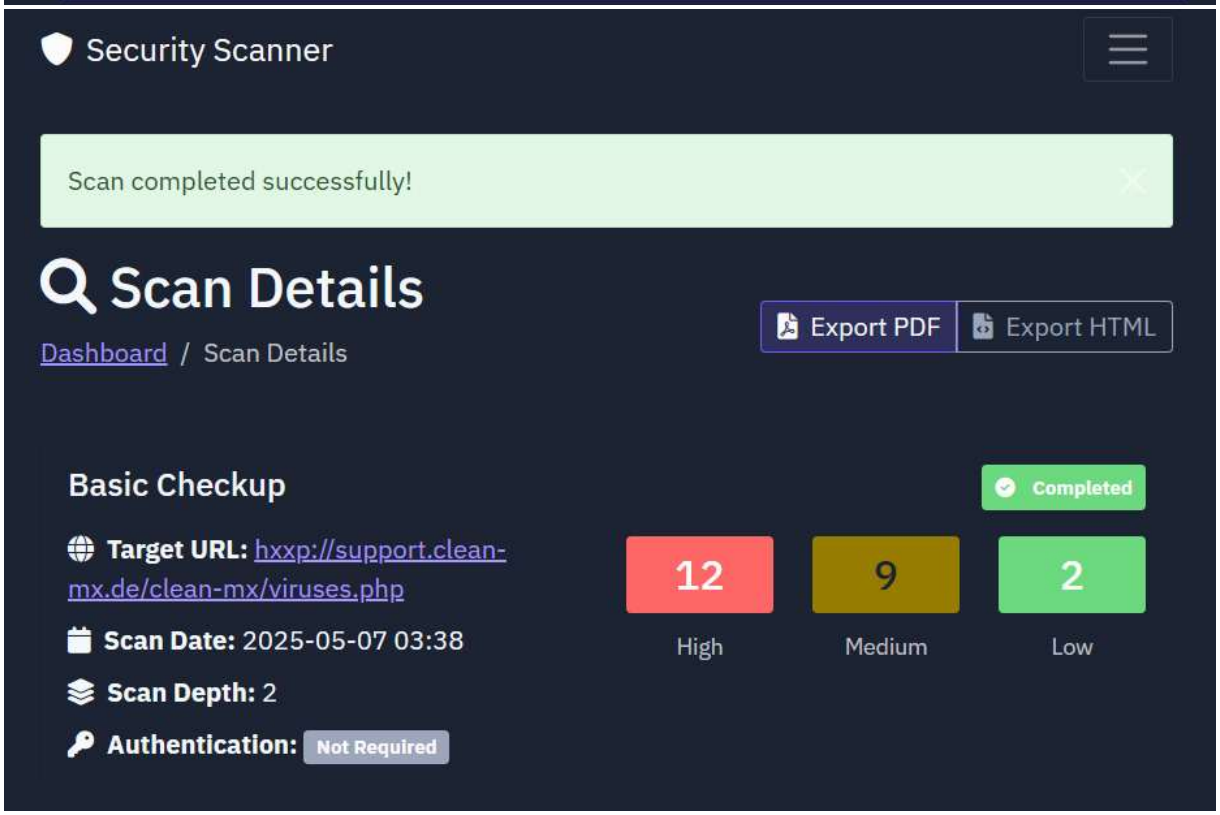
10. **Testing and Refinement:** Thoroughly test the tool with various web applications and security scenarios, and refine its functionality and user interface.

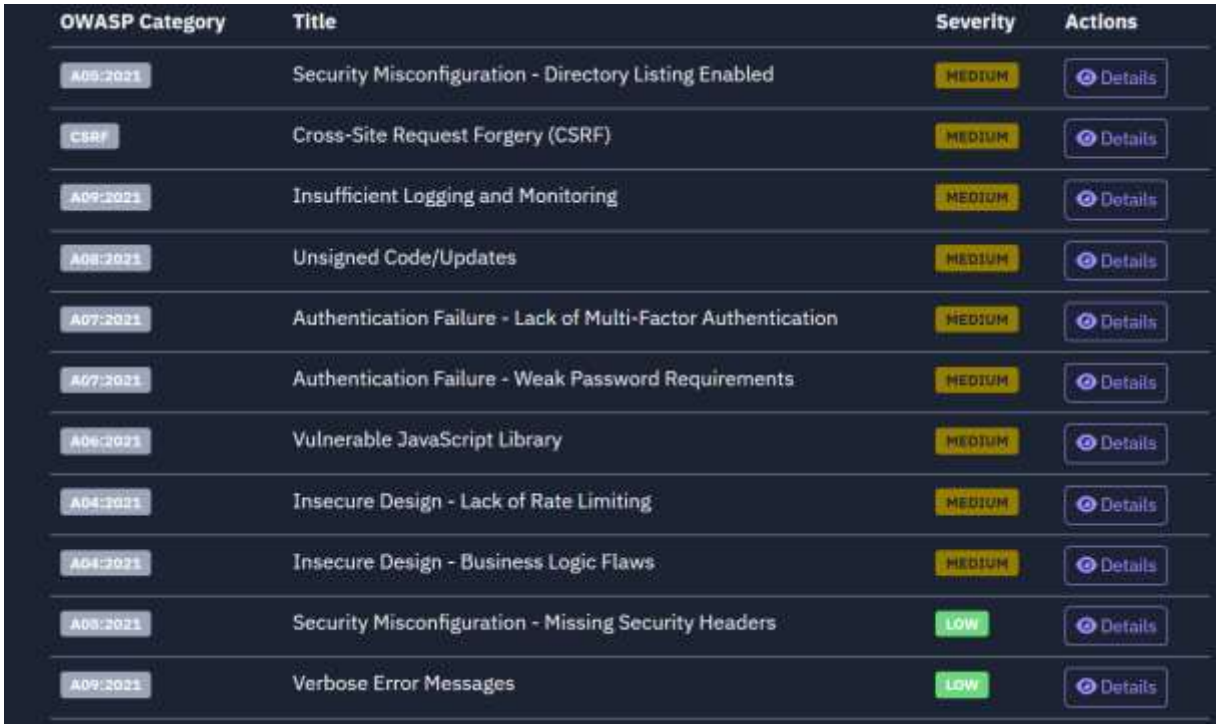
Benefits of this Tool:

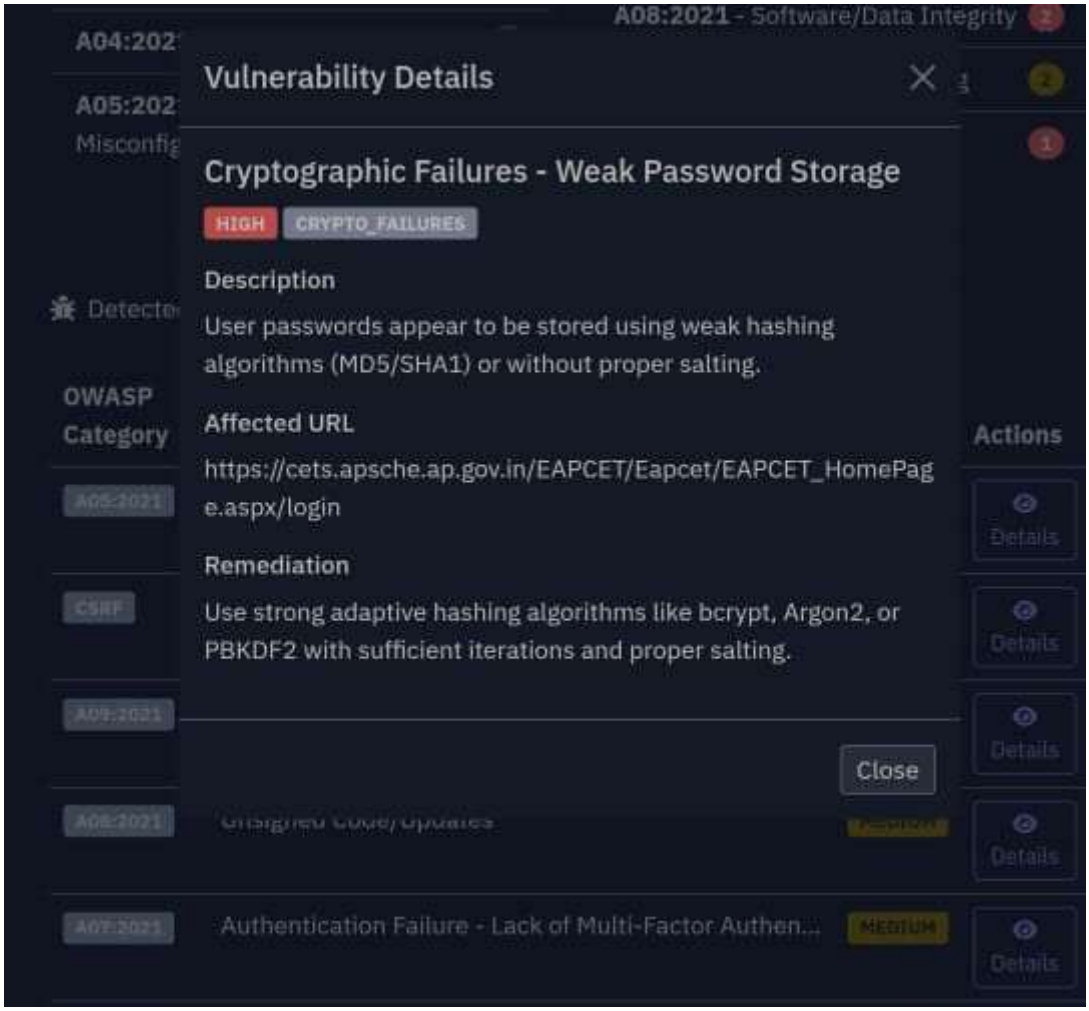
- **Automation:** Reduces the manual effort involved in performing repetitive security testing tasks.
- **Efficiency:** Speeds up the security testing process.
- **Consistency:** Ensures that tests are performed in a consistent manner.
- **Centralized Management:** Provides a single interface for configuring, running, and viewing security scans.
- **Integration:** Leverages the power of existing, well-established security scanning tools.
- **User-Friendly Interface:** Makes security testing more accessible to developers and testers.




Scan Name (Optional)
Basic Checkup
If left blank, a name will be generated based on the target URL.
Target URL *
hxxp://support.clean-mx.de/clean-mx/viruses.php
Enter the full URL of the website you want to scan (including http:// or https://)
Scan Depth
Basic (Level 1) - Scan only the specified URL
Basic (Level 1) - Scan only the specified URL
Medium (Level 2) - Scan the URL and one level of linked pages
Deep (Level 3) - Scan the URL and two levels of linked pages
What will be tested?
This scan will check for common web vulnerabilities, including:
Cross-Site Scripting (XSS)
SQL Injection
Cross-Site Request Forgery (CSRF)
Start Scan


Security Scanner
Scan completed successfully!
Scan Details
Dashboard / Scan Details
Export PDF
Export HTML
Basic Checkup
Completed
Target URL: hxxp://support.clean-mx.de/clean-mx/viruses.php
Scan Date: 2025-05-07 03:38
Scan Depth: 2
Authentication: Not Required
12
High
9
Medium
2
Low


OWASP Category
Title
Severity
Actions
Security Misconfiguration - Directory Listing Enabled
MEDIUM
Details
Cross-Site Request Forgery (CSRF)
MEDIUM
Details
Insufficient Logging and Monitoring
MEDIUM
Details
Unsigned Code/Updates
MEDIUM
Details
Authentication Failure - Lack of Multi-Factor Authentication
MEDIUM
Details
Authentication Failure - Weak Password Requirements
MEDIUM
Details
Vulnerable JavaScript Library
MEDIUM
Details
Insecure Design - Lack of Rate Limiting
MEDIUM
Details
Insecure Design - Business Logic Flaws
MEDIUM
Details
Security Misconfiguration - Missing Security Headers
LOW
Details
Verbose Error Messages
LOW
Details


A08:2021 - Software/Data Integrity
Vulnerability Details
Cryptographic Failures - Weak Password Storage
HIGH
CRYPTO_FAILURES
Description
User passwords appear to be stored using weak hashing algorithms (MD5/SHA1) or without proper salting.
Affected URL
https://cets.apsche.ap.gov.in/EAPCET/Eapcet/EAPCET_HomePage.aspx/login
Remediation
Use strong adaptive hashing algorithms like bcrypt, Argon2, or PBKDF2 with sufficient iterations and proper salting.
Close
OWASP Category
Actions
Authentication Failure - Lack of Multi-Factor Authen...
MEDIUM

Conclusion

The development of this Automated Security Testing tool successfully addresses the critical need for streamlined and consistent web application security assessment in modern development environments. By creating a comprehensive solution that integrates web-based interface management with automated browser interaction and established security scanning methodologies, the project delivers a framework that significantly enhances the efficiency and effectiveness of security testing processes.

The tool's centralized approach to security testing management represents a paradigm shift from fragmented, manual testing methodologies to an integrated, automated framework. Through its web-based interface, security professionals and developers can now configure complex security scans, manage authentication credentials, and monitor real-time testing progress from a single platform. This consolidation not only reduces the technical barrier to entry for security testing but also ensures consistency across different testing scenarios and team members.