GRADUATION PROJECT 2025-2026

WEB GUARD

A Comprehensive Web Application Vulnerability Scanner

King Abdullah II School of Information Technology | The University of Jordan

DONE BY:

1) Maria AlHajjaj (0218645)

2) Amer Farraj (2210098)

3) Abdallah AlZoubi (0218649)

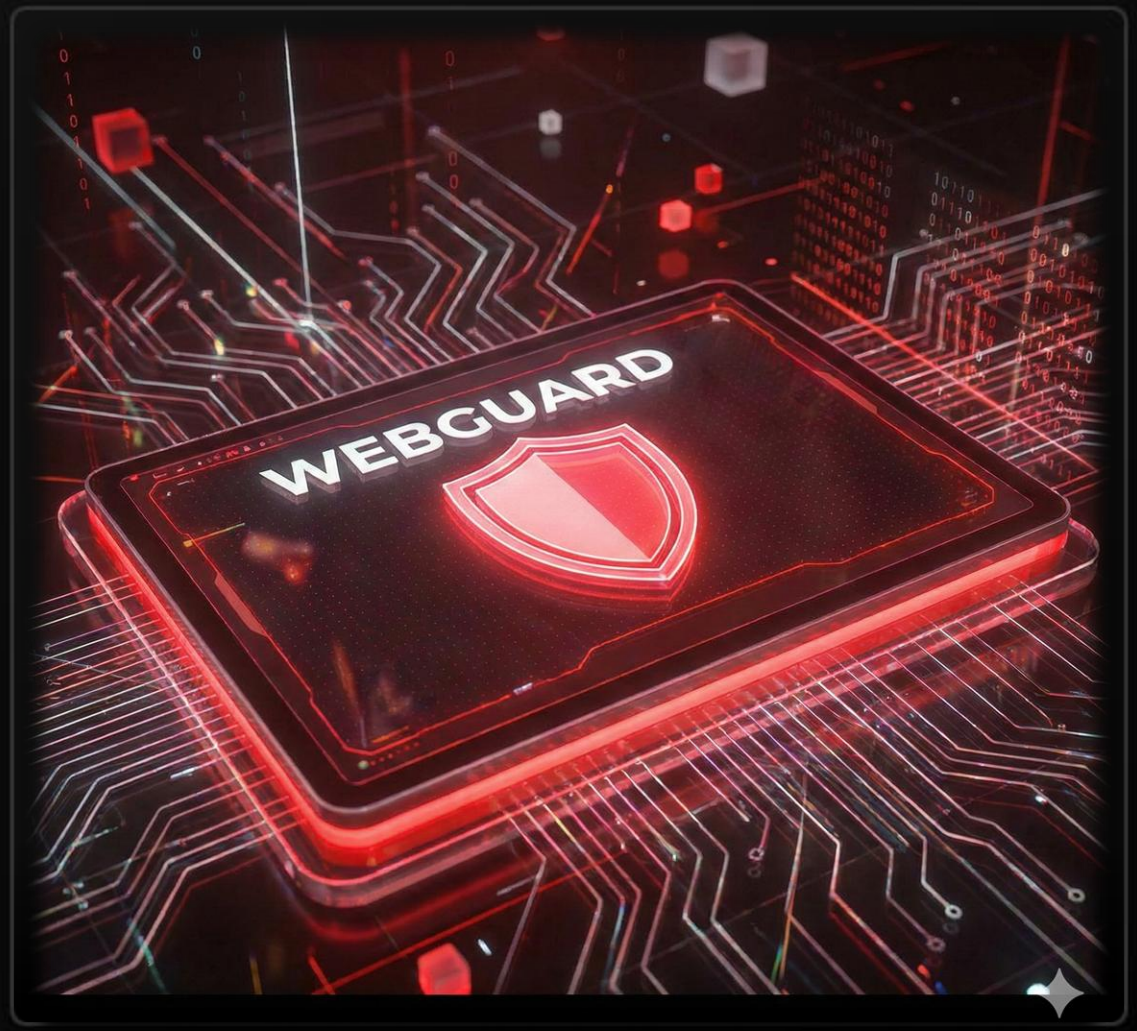
SUPERVISOR: Dr. Ahmad AlHuwaitat

1.1 INTRODUCTION

In today's digital landscape, web applications are essential for business, healthcare, and education. However, as complexity grows, so do vulnerabilities. **Cyberattacks can lead to data breaches, financial loss, and reputational damage. Recognizing this risk, Web Guard was developed.**

THE SOLUTION

A robust, intelligent security tool that proactively detects vulnerabilities before exploitation. It combines SAST, SCA, and IAST to identify critical issues like SQLi and XSS while maintaining performance.



1.2 PROJECT AIMS

Web Guard proactively identifies security vulnerabilities by crawling all accessible endpoints and input vectors.

1. **Automate the scanning process by simulating real-world attack patterns—submitting crafted requests and analyzing responses—to pinpoint exactly which page, parameter, or script is vulnerable within the target website.**

2- Prioritize findings through a risk-based scoring approach that accounts for CVSS metrics and contextual factors, highlighting critical vulnerabilities first to enable teams to address high-impact issues like missing patches and misconfigurations swiftly.

3- Support both ad hoc and scheduled scans, providing continuous monitoring of evolving codebases and newly deployed functionality to ensure no regressions slip through during rapid development cycles.

4- Generate clear, actionable reports—detailing the precise location of each vulnerability on the web page or within specific inputs, along with contextual remediation advice—to bridge the gap between detection and fix, foster a secure-by-design mindset, and significantly enhance overall application resilience

1.3 PROJECT OBJECTIVES

MAP & PROBE

Systematically crawl all reachable endpoints to unearth hidden threats like insecure deserialization and misconfigurations.

EMULATE ADVERSARIES

Drive automated, dynamic assaults using crafted requests and parsing server responses to identify specific vulnerable scripts.

SURFACE CRITICALITY

Apply context-aware scoring to guide teams to remediate high-impact risks swiftly.

SCAN ON YOUR TERMS

Offer on-demand and scheduled assessments for continuous coverage.

CLOSE THE LOOP WITH CLARITY

Produce developer-friendly reports with step-by-step remediation guidance, fostering a secure-by-design mindset.

1.4 PROJECT SCOPE: DETECTION METHODS

Web Guard leverages a multi-layered detection strategy combining three key methodologies:

SAST

Static Application Security Testing

Performs source code analysis to uncover vulnerabilities like hardcoded secrets, insecure APIs, and logic flaws.

SCA

Software Composition Analysis

Identifies vulnerabilities in open-source libraries and third-party packages (outdated/unmaintained components).

IAST

Interactive Application Security Testing

Analyzes runtime behavior and data flow during execution for context-aware vulnerability detection.

1.5 PROJECT OUTPUT FEATURES

- **User Interface:** Simple main screen for URL entry and scan type selection (SQLi, XSS, Weak Passwords).
- **Vulnerability Detection:** Automatic scanning finding SQLi, XSS, and weak passwords immediately.
- **Exploitation Simulation:** Optional exploit simulation to demonstrate real risk (e.g., unauthorized access).
- **Fix Recommendations:** Simple solutions provided (e.g., "Use safe queries" for SQLi).
- **Scan Results:** Clear reports (PDF/JSON) explaining issues, locations, and severity (High/Med/Low).
- **Performance:** Fast, accurate scanning with minimal false positives.

2.1 THE WEB GUARD WORKFLOW

Implemented in **Python**, the tool follows a clear 5-step sequence:

1

URL INPUT

User enters target URL.

2

DISCOVERY

Finds all input points (forms, parameters).

3

INJECTION

Injects payloads (SQL, JS snippets) to probe.

4

ANALYSIS

Checks responses for errors/unexpected behavior.

5

REPORTING

Lists vulnerabilities with exact locations.

2.2 & 2.3 TYPE AND COMPONENTS

2.2 WEB GUARD TYPE

- **DAST:** Tests in runtime, simulating real-world attacks.
- **Vulnerability Assessment:** Ranks issues by severity (CVSS).
- **Continuous Monitor:** Supports scheduled development cycles.
- **Reporting Tool:** Traceable reports for developers.

2.3 COMPONENTS

- **User Interface:** Simple design, one-button start, command line option for experts.
- **Detection Engine:** Analyzes code using rules to find SQLi, XSS, CSRF, SSRF. Fast and accurate.
- **Reports:** PDF/JSON outputs following OWASP/NIST rules for trusted results.

2.4 WEB GUARD & ETHICAL HACKING

Ethical hackers use Web Guard to simulate attacks in a controlled manner (White Hat Hacking).

THE PROCESS

1) **1 Authorized Testing:** Hired to find flaws.
1) **2 Analysis:** Monitors requests/responses to detect SQLi.
1) **3 Manual Check:** Crucial for privilege escalation or logic flaws.
1) **4 Report:** Guides developers on patching before malicious exploitation.

3.1 - 3.4 HISTORY AND EVOLUTION

Our Web Guard (2025) is an entirely new Python development, redefining the name for active vulnerability scanning.

2001 (IBM)

Focused on Digital Rights Management (DRM) and content protection.

2024 (ACADEMIC)

Focused on in-app forensics to detect scanners and track malicious automation.

2025 (OUR PROJECT)

Python-based Active Scanner.
Designed for ethical hacking.
Crawls, Injects, Analyzes, and Reports.
Runs efficiently on Kali Linux.

Evolution (3.4): Web Guard has evolved from simple scanning to an advanced solution for corporate and government defense, reducing data leak risks.

3.3 MODERN WEB GUARD AND MALWARE

Web Guard identifies vulnerabilities before malware can exploit them. If a vulnerability goes unnoticed, malware can:

- Redirect users to malicious websites.
- Download harmful files onto devices.
- Gain remote access to the server.
- Steal sensitive data (passwords, personal info).

4.1 SECURITY GAPS AND LIMITATIONS

Honest assessment of areas for improvement based on real-world challenges:

THREAT DETECTION SCOPE

Status: Covers SQL, XSS, Auth.

Gap: **Limited testing of business logic vulnerabilities.**

ACCURACY

Status: Basic filtering.

Gap: Non-threatening code sometimes flagged (False Positives).

INTEGRATION

Status: Basic CI/CD support.

Gap: Limited API for external security platforms.

EXPLOIT TESTING

Status: Simulated executions.

Gap: No secure sandbox for real-world exploit validation.

4.2 ALIGNMENT WITH SECURITY STANDARDS

OWASP TOP 10

Coverage: SQL, XSS, Weak Auth.
Gap: SSRF, Insecure Deserialization.
Plan: Expand scanning rules.

ISO/IEC 27001

Coverage: Supports secure development.

Gap: Structured security auditing.

Plan: Integrate audit-based tracking.

NIST FRAMEWORK

Coverage: Automated scanning.

Gap: Insider threats.

Plan: Add user behavior analytics.

4.3 RECOMMENDATIONS FOR IMPROVEMENT

1 Expand Coverage: Include business logic errors, SSRF, and insecure deserialization.

2 Enhance Automation: Improve REST API and native integrations with Jenkins/GitLab CI.

3 Secure Sandbox: Create a safe environment for exploit simulation without affecting live systems.

4 Better UI/Reports: Visual dashboards for trends and remediation progress.

5 Performance: Multi-threaded scanning for large apps.

6 Compliance: Add audit logs and user analytics (NIST/ISO alignment).

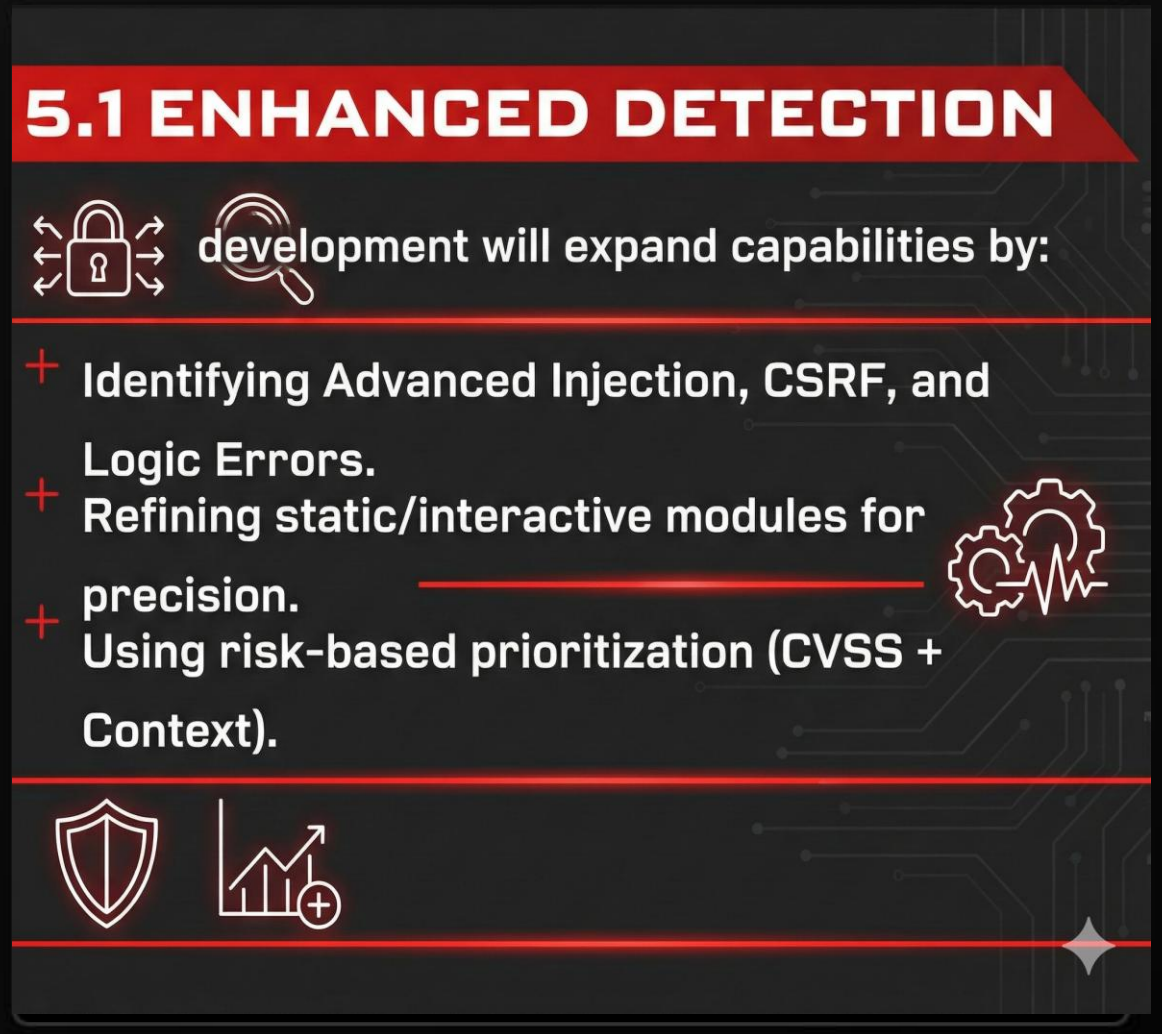
"Closing these gaps is essential for aligning with frameworks like OWASP and NIST, moving from a functional scanner to a comprehensive platform." (4.4)

5.1 & 5.2 FUTURE: DETECTION & AUTOMATION

5.1 ENHANCED DETECTION

Future development will expand capabilities by:

+ Identifying Advanced Injection, CSRF, and Logic Errors.
+ Refining static/interactive modules for precision.
+ Using risk-based prioritization (CVSS + Context).



5.3 & 5.4 REPORTING & SCALABILITY

5.3 REPORTING ENHANCEMENTS

Making reports more useful:

- **Precise Mapping:** Exact page/input field location.
- **Step-by-Step Guidance:** Safe query practices, input sanitization advice.
- **Formats:** Export to PDF, HTML, JSON for audit workflows.

5.4 ENTERPRISE SCALABILITY

Supporting complex apps:

- **Multi-threading**: Parallel processing for speed on large sites.
- **Simplification:** Easier manual scan runs and tool integration.
- **Dashboards**: Visualizing security trends and recurring issues.

5.5 IMPLEMENTATION TIMELINE

2025

Extend Detection Coverage. Refine Scanning Methodology.

2026

Enhanced Automation. Scheduled Scans. Improved Prioritization.

2027

Enterprise Scalability. Parallel Processing. Big Data Analytics.

BEYOND

Continuous Integration. Long-term Security Updates.

5.1 & 5.2: FUTURE DET

5.1 ENHANCED DETECTION

EXPANDED CAPABILITIES:

- Adv. Injection, CSRF, Logic Errors.
- Refined Modules (Static/Interactive).
- Risk-Based Prioritization (CVSS+).

6.1 SUMMARY OF ACHIEVEMENTS

Web Guard connects academic learning with real-world application.

COMPREHENSIVE FRAMEWORK

Built in Python. Integrates SAST, SCA, and IAST methods.

REALISTIC SIMULATION

Safely imitates SQLi and XSS to help developers understand risk.

DETAILED REPORTING

CVSS Scores, Location Mapping, Multi-format export.

AUTOMATION

Webhooks, API, and CI/CD integration (GitHub/GitLab).

6.2 & 6.3 REAL-WORLD IMPACT

Tangible Difference: Efficiently detects vulnerabilities before they become threats.

Cost Reduction: Accurately lowers costs associated with breaches.

Integration: Seamlessly fits into modern dev environments.

Continuous Mandate: A one-time test is not enough. Future versions will use background tasks to secure apps before production.

"NOTHING BEFORE THEY REACH PRODUCTION."

SYSTEM LIMITATIONS: ACCESS BARRIERS

-> Why Some URLs Block Scanning Operations:

- **Web Application Firewalls (WAF):** Servers detect and drop packets from automated scanning signatures.
- **Rate Limiting (Anti-DoS):** Webservers block IP addresses that exceed request thresholds to protect stability.
- **User-Agent Filtering:** Security mechanisms deny access to non-browser identities used by scripts.
- **Robots.txt Restrictions:** Target paths may be explicitly flagged to prevent external crawling.
- **Permission Gating (HTTP 403):** Access is restricted due to required authentication or private file settings.

Implementation Status (Code vs Slide Claims)

This slide aligns the delivered code with the original project vision.

Implemented in Current Code

- DAST (safe passive checks): headers, cookies, CSRF heuristic, fingerprinting
- Optional ACTIVE tests (LOCAL/PRIVATE only): SQLi & reflected XSS on GET params
- SAST: secrets, dangerous functions, weak crypto, debug exposure
- SCA: dependency hygiene for requirements.txt (unpinned + URL/VCS sources)
- Reporting: JSON + professional PDF (Top 5 prioritized + details/recommendations)

Planned / Future Work (Not Implemented Yet)

- Real IAST (runtime instrumentation + data-flow / taint tracking)
- Context-aware risk scoring (CVSS + exposure + confidence + asset criticality)
- CI/CD + Webhooks + REST API integrations (GitHub/GitLab/Jenkins)
- Expanded coverage: SSRF, insecure deserialization, business logic testing

Known External Barriers (Real-World)

- WAF and bot detection can block automated scanners
- Rate limiting (anti-DoS) may throttle high request volumes
- User-Agent filtering and 403 gating restrict access to endpoints
- robots.txt may disallow crawling certain paths

How We Will Address Them

- Adaptive throttling + retry/backoff + request pacing profiles
- WAF detection heuristics and clearer access diagnostics in reports
- Auth/session support for permitted targets (cookie jar + login flows)
- Safer sandboxed PoC validation in a controlled lab environment

Future Work (Gap-Driven Roadmap)

Next iterations to reach enterprise-grade coverage, automation, and usability.

Detection & Coverage

- Add SSRF, insecure deserialization, and deeper auth/logic testing
- Improve active testing safely (non-destructive, evidence-only probes)
- Reduce false positives with confidence scoring + verification steps
- Optional authenticated scanning (only with explicit permission)

Automation & Integration

- Scheduled scans (cron/systemd/Task Scheduler) + diff between scans
- REST API + webhooks for notifications and pipeline hooks
- CI/CD integration: fail builds on High/Critical findings
- Multi-thread tuning + rate-limit aware concurrency

Reporting & UX

- Add HTML report export (filters, search, severity chips)
- Dashboards for trends + remediation progress tracking
- Better evidence: request/response snippets (sanitized) + timestamps

Safety & Validation

- Docker-based sandbox lab for PoC validation (no impact on live systems)
- Explicit safety gates for active tests (scope allowlist + private targets)
- Stronger compliance support: audit logs and structured findings history

6.4 FINAL THOUGHTS

Web Guard shows how security can be included at every step of making software. It proves the utility of accessible security tools.

While gaps exist (logic issues, UI guidance), Web Guard meets important standards like the OWASP Top 10.

"A hands-on learning project leading to meaningful impact."

REFERENCES

[1] OWASP Foundation, OWASP Top Ten Web Application Security Risks, 2021. [Online]. Available: https://owasp.org/www-project-top-ten/

[2] ISO/IEC, ISO/IEC 27001: Information Technology – Security Techniques – Information Security Management Systems – Requirements, International Organization for Standardization, 2013.

[3] National Institute of Standards and Technology (NIST), Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1, 2018. [Online]. Available: https://www.nist.gov/cyberframework

[4] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, “Static analysis of security vulnerabilities in open-source web applications,” in Proc. IEEE Int. Conf. on Software Testing, Verification and Validation, 2013, pp. 63–72.

[5] OWASP Foundation, Static Application Security Testing (SAST). [Online]. Available: https://owasp.org/www-community/Static_Application_Security_Testing

[6] OWASP Foundation, Software Composition Analysis (SCA). [Online]. Available: https://owasp.org/www-community/Component_Analysis

[7] OWASP Foundation, Interactive Application Security Testing (IAST). [Online]. Available: https://owasp.org/www-community/IAST

THANK YOU