
VulnOdin: The All-Seeing Security Agent¹

Daud Mustafa Minhas
Industrial Security ZeMA,
Saarland

Mubeen Afzal
Msc Cybersecurity -
Saarland University

Hafiza Hajrah Rehman
Saarland University-
AI security Saarland

Sujal Jadhav
Msc Cybersecurity -
Saarland University

Amit Saxena
Saarland University-
AI security Saarland

With
Apart Research

Abstract

Modern enterprise software development has accelerated toward continuous integration and deployment (CI/CD), yet penetration testing remains periodic, manual, costly, and slow. Meanwhile, cybersecurity communities have observed a sharp rise in *agentic-AI-driven offensive capabilities*, where autonomous multi-agent systems can chain reconnaissance, scanning, exploitation, and privilege escalation with minimal human oversight.

This project presents VulnOdin, an AI-powered autonomous red teaming system designed specifically for DevSecOps pipelines and enterprise environments. VulnOdin accepts live URLs and complete codebases including Dockerfiles, autonomously builds and deploys the target environment, constructs a dynamic attack graph, orchestrates traditional offensive security tools (e.g., nuclei, sqlmap, Burp extensions), and executes controlled exploitation attempts under strict governance constraints. The system continuously evaluates applications during development, automatically producing evidence-backed, compliance-ready penetration testing reports aligned with OWASP, CVSS, and ISO frameworks.

Our findings demonstrate that AI-supported agentic orchestration can improve exploit chain discovery, reduce false positives through

¹ Research conducted at the Defensive Acceleration Hackathon
<https://apartresearch.com/sprints/def-acc-hackathon-2025-11-21-to-2025-11-23>, 2025

tool-verified evidence, and dramatically shorten the vulnerability discovery cycle. This work proposes a new paradigm: reproducible, continuous, environment-aware, autonomous pentesting integrated directly into enterprise build pipelines.

Keywords: Autonomous red teaming, DevSecOps, multi-agent systems, AI security, attack graphs, continuous pentesting, vulnerability assessment.

1. Introduction

The emergence of autonomous AI agents has created a profound shift in the cyber threat landscape, where autonomous multi-agent systems can chain reconnaissance, scanning, exploitation, and privilege escalation with minimal human oversight (Kshetri et al., 2025). Large language models now enable even minimally skilled actors to discover zero-days, generate working exploits, and chain attacks at speeds previously accessible only to nation-state teams. This democratisation of offensive capability has dramatically widened the offense-defense asymmetry (Murphy et al., 2025). While AI accelerates attack discovery from weeks to minutes, the vast majority of web applications, including those underpinning critical research platforms, biosecurity tools, and AI infrastructure, receive no systematic security testing before deployment (Erendor et al., 2024). A single unpatched vulnerability can lead to data exfiltration, ransomware, or in extreme cases, the compromise of high-risk systems handling dual-use biological data or model weights (Metta et al., 2024).

This work is focused on a single guiding question: How can the same autonomous AI capabilities that increasingly empower attackers be repurposed to accelerate defensive testing by orders of magnitude, and enable vulnerabilities to be identified and remediated at the pace of modern offence? We hypothesize that if AI-driven red teaming becomes instantaneous, reproducible, and widely accessible, the economics of cybersecurity will shift. Leaving vulnerabilities unpatched will become more costly, in terms of risk exposure than fixing them. In this view, democratizing high-quality security testing is not merely a technical upgrade but an incentive-altering force capable of reshaping the current landscape, where attackers typically enjoy lower effort and higher reward than defenders.

To evaluate this hypothesis, we developed **VulnOdin**, an autonomous AI-powered red-teaming system that requires only a public GitHub repository or a live URL and produces a comprehensive, disclosure-ready penetration testing report in ten to fifteen minutes just using Large Language Models. The system operates through four fully automated phases orchestrated by a LangGraph ReAct supervisor.

Problem Statement

Enterprise security teams lack an autonomous, reproducible, environment-aware pentesting system that can run continuously as part of the software development

lifecycle (SDLC), validate vulnerabilities using real exploits, and produce standardized reporting outputs.

Contribution

This work introduces:

- A DevSecOps-native autonomous red teaming agent integrating LLM reasoning, traditional tools, and environment provisioning.
- A code-aware attack pipeline where Dockerfiles and repository structure directly inform vulnerability analysis.
- A dynamic attack graph reasoning engine to chain vulnerabilities.
- A continuous pentesting model aligned with real development workflows.

Our work directly addresses pressing industry concerns regarding AI-enabled attacks, modern tooling limitations, and the need for scalable, automated defensive capabilities.

By compressing what traditionally requires 5–15 days and \$15,000–\$50,000 of expert manual pentesting into an autonomous process that completes in under ten minutes and costs nothing, VulnOdin demonstrates that AI-native defensive tooling can outpace offensive innovation. It serves as a working proof-of-concept that high-leverage, scalable, autonomous cybersecurity infrastructure is not a distant future goal but something that can be prototyped in a single weekend, precisely the kind of rapid, asymmetric advantage the defensive acceleration paradigm demands in an era of AI-accelerated threats.

2. Methodology

The methodology of this project follows a multi-stage automated web penetration testing pipeline, beginning with user input and ending with AI-assisted report generation. The process integrates two operational modes; 1) URL-based pentesting and 2) Codespace-based pentesting, which converge into a unified Dynamic Crawler & Injection engine. The workflow, as depicted in the Figure 1, unfolds through the following stages.

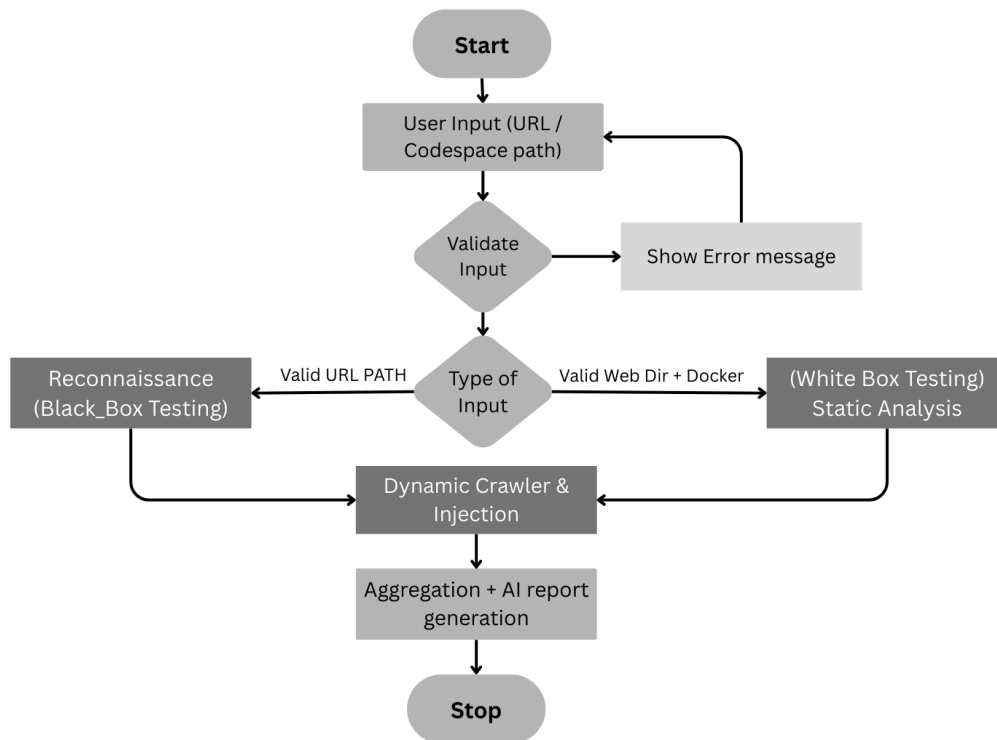


Figure 1 – Overall Workflow for Automated Web Pentesting

Phase 1: Input Acquisition and Validation

The workflow begins when the user provides either:

- a URL for testing a deployed web application, or
- a Codespace path including Docker files representing a local application directory.

The system performs an initial input validation check.

Invalid paths (malformed URLs, missing directories, or inconsistent Docker configurations) immediately trigger an error message, prompting the user to correct the input. No further processing occurs until valid input is supplied.

Phase 2: Input Classification and Workflow Branching

Once validated, the system determines the **type of input**, which dictates the penetration testing mode:

- **Web Pentesting via URL**

Selected when the input is a valid, reachable URL.

- **Web Pentesting via Codespace**

Activated when the input corresponds to a valid directory containing source code and Docker artifacts.

Each branch follows a distinct internal workflow before both converge into dynamic analysis.

Phase 3: Web Pentesting

- **Via URL (Black-Box Path):** When operating on a URL, the system behaves as an external attacker and expands the URL-based (black-box) pentesting branch illustrated in Figure 2a. It shows the sequential actions performed by the system once a valid URL is provided. The steps following as:
 1. **Recon Agent Execution**
Initial reconnaissance to identify accessible endpoints.
 2. **Port, Service, and Version Scanning**
Enumeration of exposed services and software stack fingerprints.
 3. **Maintain History**
Storing scan outputs for comparison, traceability, and regression analysis.
 4. **Online CVE Check**
Version-specific CVEs are fetched from public vulnerability databases.
 5. **Detailed Report Generation**
The AI agent prepares an intermediate report summarizing URL-derived vulnerabilities.

This completes the black-box portion of the pipeline before transitioning into dynamic injection.

Web pentesting via URL activates for live targets. A dedicated recon agent performs port scanning, service enumeration, and technology fingerprinting (via httpx, nmap, whatweb), then cross-references every detected version against live

CVE databases and Nuclei templates to produce an up-to-date exploitability heatmap.

- **via Codespace (White-Box Path):** When a Codespace directory is provided, the system conducts a combined static and environment-aware analysis. It details the steps involved in analyzing local or remote project directories: file retrieval, code extraction, AI-assisted static vulnerability scanning, containerized environment deployment via Docker Compose, CVE-based dependency checks, and intermediate report generation illustrated in Figure 2b. The steps are:
 1. **Retrieve Files**
Collect source code, configuration files, and Docker definitions.
 2. **Extract Code Files**
Identify relevant application code, environment variables, and dependency manifests.
 3. **AI-Assisted Vulnerability Scan**
Perform static analysis on extracted files to detect code-level security issues.
 4. **Run Docker Compose**
Build and deploy the application environment for dynamic testing.
 5. **Check Services Against CVE Databases**
Identify vulnerable dependency versions across services defined in the Docker stack.
 6. **Detailed Report Generation**
Produce a preliminary report summarizing static and container-level findings.

This completes the white-box analysis before joining the dynamic testing pipeline.

Web pentesting via Docker for Codespace: begins immediately upon receiving a GitHub repository or Codespace link including docker files. The agent clones the codebase, retrieves all source files (.py, .js, .php, .html, etc.), and runs parallel LLM-augmented static analysis using GPT-4o/Claude 3.5 Sonnet with targeted vulnerability prompts, while allowing LLM to use its own ways to execute scanning on vulnerabilities.

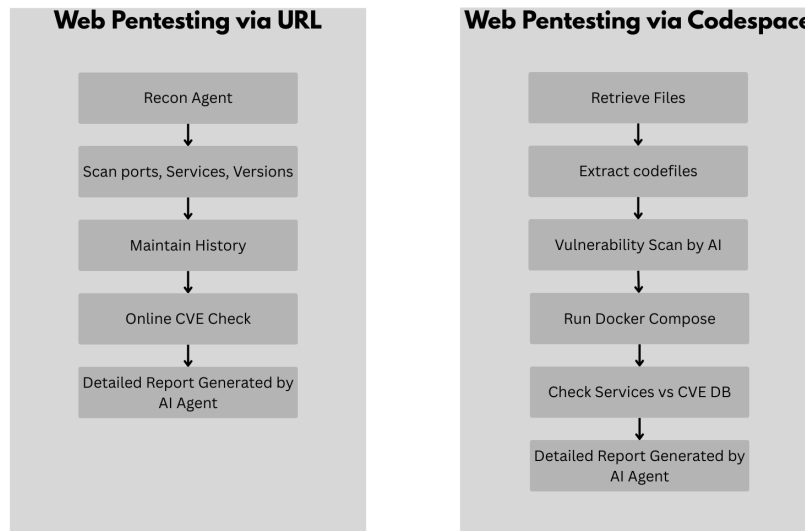


Figure 2 – a): Detailed Workflow for Web Pentesting via URL, b): Detailed Workflow for Web Pentesting via Codespace

Phase 4: Dynamic Crawler & Injection (Unified Stage)

Both URL-based and Codespace-based paths feed into a centralized Dynamic Crawler and Injection engine responsible for active vulnerability discovery. Figure 3 visualizes the unified Dynamic Crawler & Injection stage, where both URL-based and Docker-based workflows converge. It outlines the sub-steps of dynamic testing, automated crawling, recursive URL discovery, malicious payload injection, vulnerability detection, and aggregation of all findings. This module performs active exploitation heuristics across both testing modes.:

- 1. Crawler Agent Execution**
Automated exploration of the application to discover recursive URLs, endpoints, and hidden routes.
- 2. Find Recursive URLs**
Systematically follows links and forms to uncover deeper site structure.
- 3. Inject Malicious Payloads**
Attempts common exploitation vectors (e.g., SQL injection, XSS, command injection).
- 4. Detect Vulnerabilities**
Identifies abnormal responses, reflected input, privilege escalation opportunities, and insecure configurations.

5. Aggregate All Findings

Consolidates results from URL pentesting, Codespace static analysis, and dynamic injection

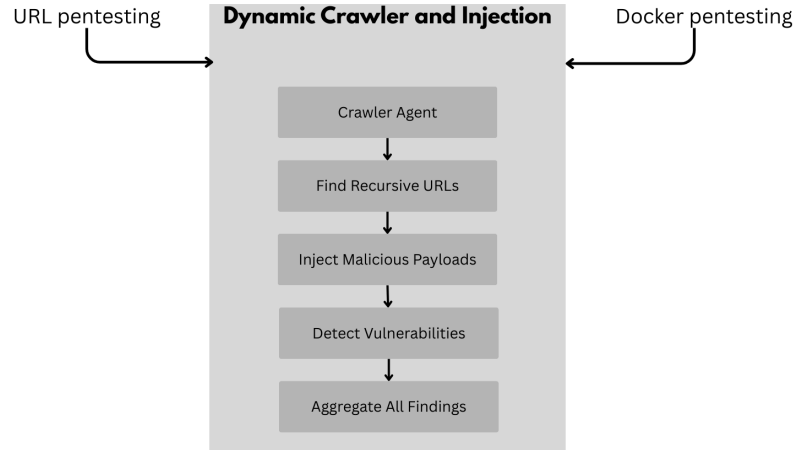


Figure 3 – Dynamic Crawler and Injection Engine

Dynamic Testing & Exploitation is the breakthrough phase. When Dockerfiles or docker-compose.yml are present, the agent automatically builds and launches the full application in an isolated container. A Playwright-powered crawler agent then, recursively discovers all endpoints, submits context and type-aware malicious payloads (SQL injection, XSS, SSRF, command injection, path traversal, etc.), verifies successful exploitation, and captures timestamped screenshots as irrefutable proof.

Phase 5: Aggregation and AI-Generated Reporting

The final stage aggregates all outputs into a unified, structured dataset. The AI agent then produces the final penetration testing report, including:

- consolidated vulnerability listings
- CVE correlations
- service and version maps
- evidence-based findings
- prioritized recommendations

This marks the end of the automated workflow.

Report Writing is handled by a specialized reporting agent that aggregates every finding, calculates CVSS v3.1 scores, highlights exact vulnerable lines with one-click fix snippets, embeds proof screenshots, and generates a professional, ready-to-send PDF/HTML report complete with executive summary and responsible-disclosure template.

VulnOdin was developed by a distributed team working with a single $1 \times H100$ GPU instance on Lambda Cloud, alongside local development machines. No custom models were trained; instead, we intentionally relied on existing frontier LLMs—GPT-4o and Claude 3.5 Sonnet—accessed via API, along with a suite of mature open-source security tools. This design choice underscores one of the core claims of this work: expert-level autonomous red teaming is achievable today using only readily available components. The full codebase, maintained continuously after the hackathon, is publicly available at:

<https://github.com/hajraRehman/RedTeam-in-a-Box-Fully-Autonomous-AI-Red-Teaming-Service>

The system architecture follows the workflow illustrated in Figure 1, which documents the complete end-to-end pipeline—from input validation through to final report generation.

3. Results

Figure 4 shows the frontend interface of the developed system, branded as VulnOdin, where the autonomous red-teaming assistant is initialized. The interface presents the user with available testing options URL scanning, Docker application testing, result analysis, and security consultation. The system also displays an important notice about scan duration, indicating that comprehensive scans may take up to five minutes due to multi-phase analysis. This demonstrates the successful deployment of the user interaction layer and confirms that the agentic workflow is ready to accept input.

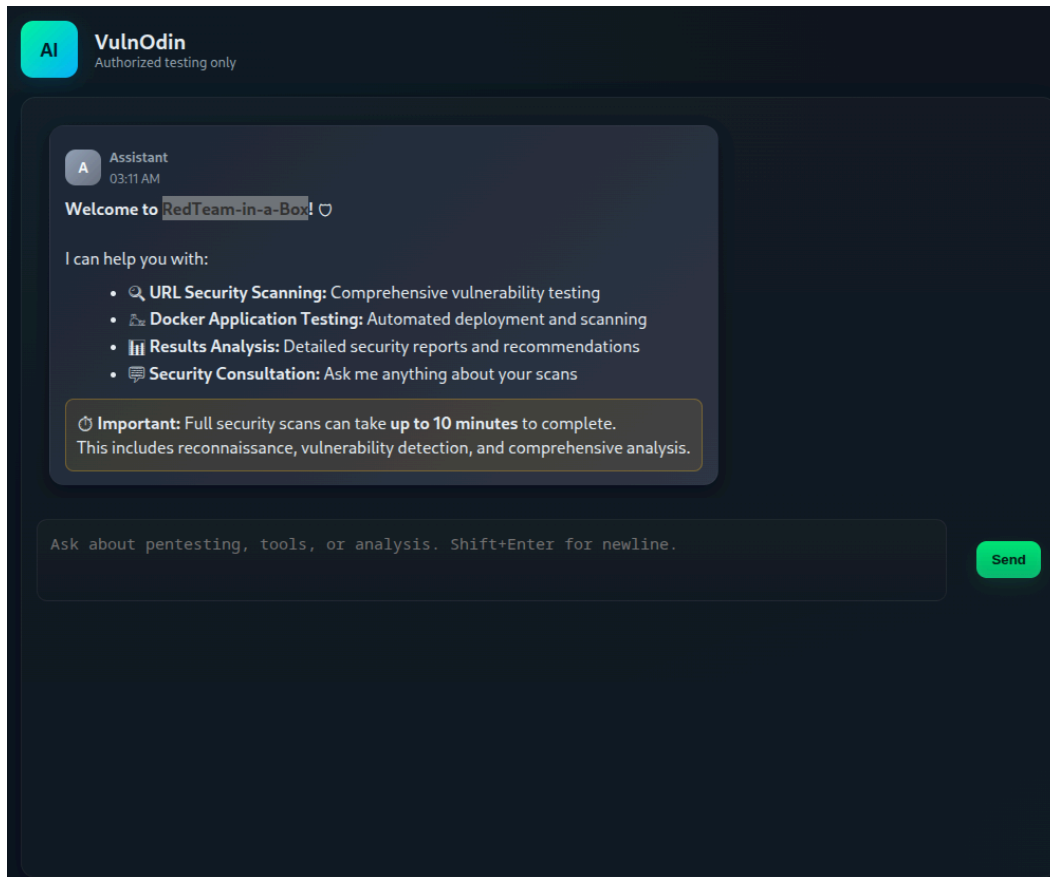


Figure 4 – System Welcome Screen (VulnOdin Interface)

Figure 5 captures the target site used for testing: testhtml5.vulnweb.com, an intentionally vulnerable web application designed for security research. The page shown is the "Contact" form, which contains user-input fields known to host client-side and server-side vulnerabilities, including XSS and injection flaws. This validates that the system is being tested against a legitimate training target for safe security research, ensuring ethical and controlled experimentation.

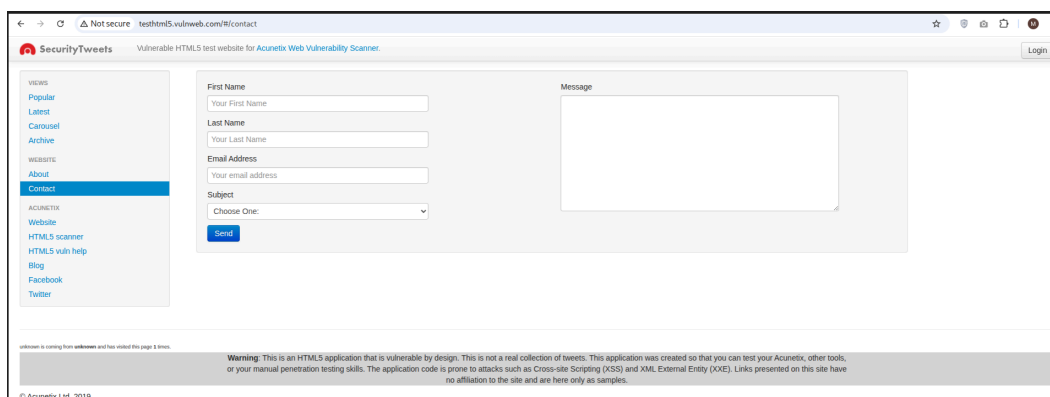


Figure 5 – Target Vulnerable Website (TestHTML5 by Acunetix)

Figure 6 shows the user submitting the test URL into the agent interface with the command to “scan this website and build a report.” The system acknowledges the

request and displays a status message that the scan has begun. It warns the user that the full scan may take up to ten minutes, reflecting the multi-stage process (reconnaissance, vulnerability detection, crawling, and report generation). This confirms that user input is correctly processed and forwarded to the backend for automated analysis.

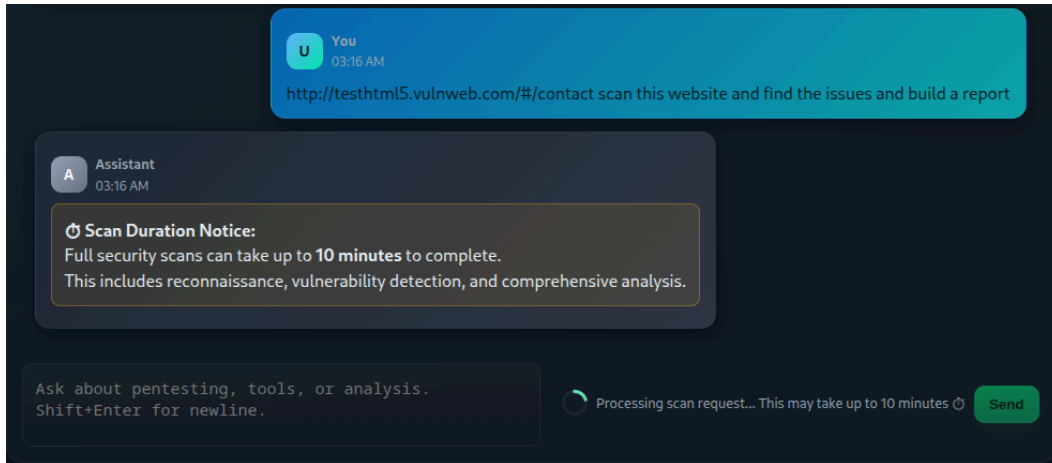


Figure 6 – Scan Request Submitted in the UI

Figure 7 displays the backend terminal logs where the RedTeam-in-a-Box API server is launched. The server initializes successfully, exposes the API and web interface on port 8000, and enters a listening state. The log messages indicate that the application is running under Uvicorn and is ready to handle scan requests. This demonstrates that the backend microservice architecture is functioning, responding to client requests, and orchestrating the autonomous agent pipeline.

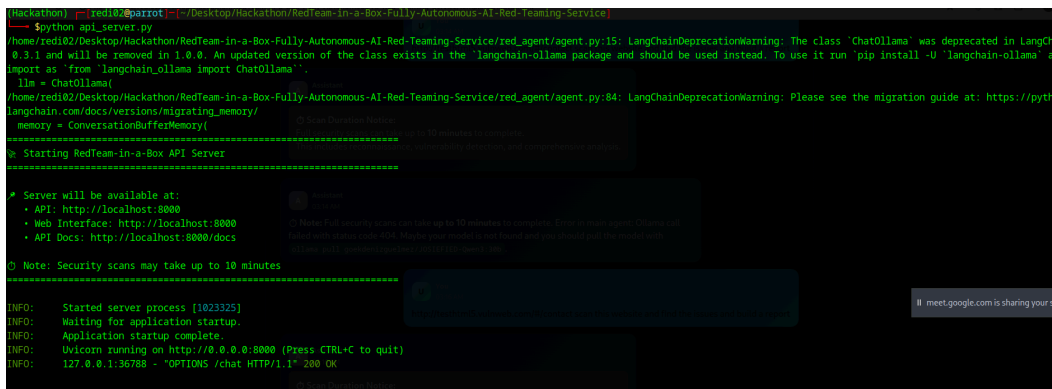


Figure 7 – Backend Server Initialization (Terminal Output)

Figure 8 shows the internal agent execution trace. The agent first verifies target accessibility using a PingURL operation, confirms that the domain responds successfully, and then transitions into reconnaissance (Phase 1). The system extracts the IP address, performs additional lookups, and prepares to launch deeper security scanning. This verifies that the agent is executing the defined workflow: validating the target, performing preliminary recon, and then moving into

vulnerability analysis according to the multi-stage pipeline described in the methodology.

```
> Entering new AgentExecutor chain...
Thought: I need to check if the URL is accessible using PingURL.
Action: PingURL
Action Input: http://testhtml5.vulnweb.com/#/contact/ URL testhtml5.vulnweb.com is accessible
PING testhtml5.vulnweb.com (44.228.249.3) 56(84) bytes of data:
64 bytes from ec2-44-228-249-3.us-west-2.compute.amazonaws.com (44.228.249.3): icmp_seq=1 ttl=52 time=185 ms
64 bytes from ec2-44-228-249-3.us-west-2.compute.amazonaws.com (44.228.249.3): icmp_seq=2 ttl=52 time=199 ms
64 bytes from ec2-44-228-249-3.us-west-2.compute.amazonaws.com (44.228.249.3): icmp_seq=3 ttl=52 time=191 ms
64 bytes from ec2-44-228-249-3.us-west-2.compute.amazonaws.com (44.228.249.3): icmp_seq=4 ttl=52 time=188 ms

--- testhtml5.vulnweb.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/ndev = 185.085/190.657/198.933/5.181 ms
Final Answer: The URL http://testhtml5.vulnweb.com/#/contact is accessible. The domain testhtml5.vulnweb.com successfully responded to the ping request with a 0% packet loss and an average round-trip time of 190 ms.

> Finished chain.

Phase 1: Calling Red Agent with query: Perform a comprehensive security scan on the URL http://testhtml5.vulnweb.com/#/contact to identify vulnerabilities and analyze its security posture

> Entering new AgentExecutor chain...
Thought: I need to analyze the security posture of http://testhtml5.vulnweb.com/#/contact. First, I'll start with basic reconnaissance tools to gather foundational information.
Action: URLtoIP
Action Input: http://testhtml5.vulnweb.com/#/contactIP address for testhtml5.vulnweb.com: 44.228.249.3Thought: I need to analyze the security posture of http://testhtml5.vulnweb.com/#/contact. First, I'll start with basic reconnaissance tools to gather foundational information.
```

Figure 8 – Agent Execution: Ping, Reconnaissance, and URL Analysis

Figure 9 shows the agent performing foundational reconnaissance on the target URL. The system first conducts an Nmap scan, identifying the web server (nginx 1.19.0) running on port 80. It then performs a DNS lookup, retrieving A-records and TXT records such as SPF entries. An attempt is made to check SSL/TLS certificate information—which times out because the site does not support HTTPS—after which the agent proceeds to retrieve HTTP headers. These steps demonstrate the system’s ability to assess server infrastructure, gather enumerative metadata, and establish the baseline security posture of the target.

```
Action: Nmap
Action Input: 44.228.249.3Nmap scan results for 44.228.249.3:
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-11-24 03:17 PKT
Nmap scan report for ec2-44-228-249-3.us-west-2.compute.amazonaws.com (44.228.249.3)
Host is up (0.19s latency).
Not shown: 999 filtered top ports (no-response)
PORT      STATE SERVICE
80/tcp    open  http
nginx 1.19.0

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 32.43 seconds
Thought: I need to analyze the security posture of http://testhtml5.vulnweb.com/#/contact. I'll continue with DNS lookup to get domain records, then check SSL certificates, HTTP headers, and web technologies.

Action: DNSLookup
Action Input: testhtml5.vulnweb.comDNS Information for testhtml5.vulnweb.com:
A Record (IPv4): 44.228.249.3

Detailed DNS Records:
testhtml5.vulnweb.com. 10800 IN TXT "v=spf1 -all"
testhtml5.vulnweb.com. 10800 IN A 44.228.249.3
Thought: I have the IP address (44.228.249.3) and basic DNS records. Next, I'll check SSL/TLS certificate details, HTTP headers, and web technologies to assess security posture.

Action: SSLCertInfo
Action Input: http://testhtml5.vulnweb.com/#/contact SSL certificate check timed outThought: The SSLCertInfo tool timed out, so I'll proceed with other tools. Next, I'll check HTTP headers to identify security-related headers and then use Wappalizer to detect web technologies.

Action: HTTPHeaders
Action Input: http://testhtml5.vulnweb.com/#/contact HTTP Headers for http://testhtml5.vulnweb.com/#/contact:
Status Code: 200

Response Headers:
Server: nginx/1.19.0
Date: Sun, 23 Nov 2025 22:18:18 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
```

Figure 9 – Network Scanning and Reconnaissance (Nmap, DNS, SSL, HTTP Headers)

Figure 10 shows the system’s Advanced Vulnerability Scanner evaluating interactive components—in this case, a contact form. The scanner identifies multiple high-severity vulnerabilities: weak credentials (admin/admin), XSS in the username field, and path traversal through crafted directory traversal payloads. The summary shows that the scan explored two URLs, tested one form, and found

five unique vulnerabilities. This demonstrates the successful execution of dynamic, form-driven fuzzing and injection techniques.

```

* ADVANCED VULNERABILITY SCANNER v2.0
=====
Target: http://testhtml5.vulnweb.com/#/contact
Max Depth: 3
=====
[Depth 0] http://testhtml5.vulnweb.com/#/contact...
Found 1 form(s)
Testing form 1/1
Detected login form
  [HIGH] Weak Credentials: admin/admin
  [HIGH] XSS (Cross-Site Scripting) in 'username' (confidence: 80.0%)
  [HIGH] Path Traversal in 'username' (confidence: 75.0%)
[Depth 1] http://testhtml5.vulnweb.com/static/css/style.css...

=====
||| SCAN SUMMARY
=====
Duration: 8.2 seconds
URLs Visited: 2
Forms Tested: 1
Parameters Tested: 6
HTTP Requests: 22
Errors: 0

VULNERABILITIES FOUND: 5
Unique Issues: 5

  HIGH: 3
  LOW: 2
=====
```

Figure 10 – Automated Vulnerability Scanner (Runtime Form Testing & Issue Detection)

Next the agent extracts detected technologies from previous scan results, identifies nginx 1.19.0, and queries the National Vulnerability Database (NVD) for matching CVEs and is illustrated in Figure 11. It locates three critical CVEs associated with this nginx version and stores them in a generated report file. This screenshot confirms that the system not only identifies runtime vulnerabilities but also correlates technologies with known public exploits, significantly strengthening the completeness of the assessment.

```
=====
Q CVE Vulnerability Database Check
=====

[+] Extracting services and versions from scan reports...
[+] Found 2 unique services/technologies:
    - nginx (v1.19.0) from Red Agent Scan
    - nginx (version unknown) from Red Agent Scan

[+] Searching NVD CVE database...
    Searching: nginx 1.19.0 (exact match)...
    Searching: nginx (service-only)...
    Searching: nginx (service-only)...

[+] Found 3 total CVEs, reporting top 3 most critical

✓ CVE report saved to: ./cve_report.txt
INFO: 127.0.0.1:36804 - "POST /chat HTTP/1.1" 200 OK
[+] 100% completed (1/12/2021 10:15:00 AM)=====
```

Figure 11 – CVE Database Lookup (NVD Integration)

Coming to the end results from Figure 12, it displays the autogenerated Security Analysis Report, summarizing high-level findings. The report highlights the site's accessibility, lists 3 critical CVEs, and identifies 5 runtime vulnerabilities with severity ratings. The reconnaissance section details detected technologies, missing security headers, and limitations encountered (e.g., SSL check timeout). This demonstrates the system's ability to translate raw scan data into a structured, understandable executive-level summary.

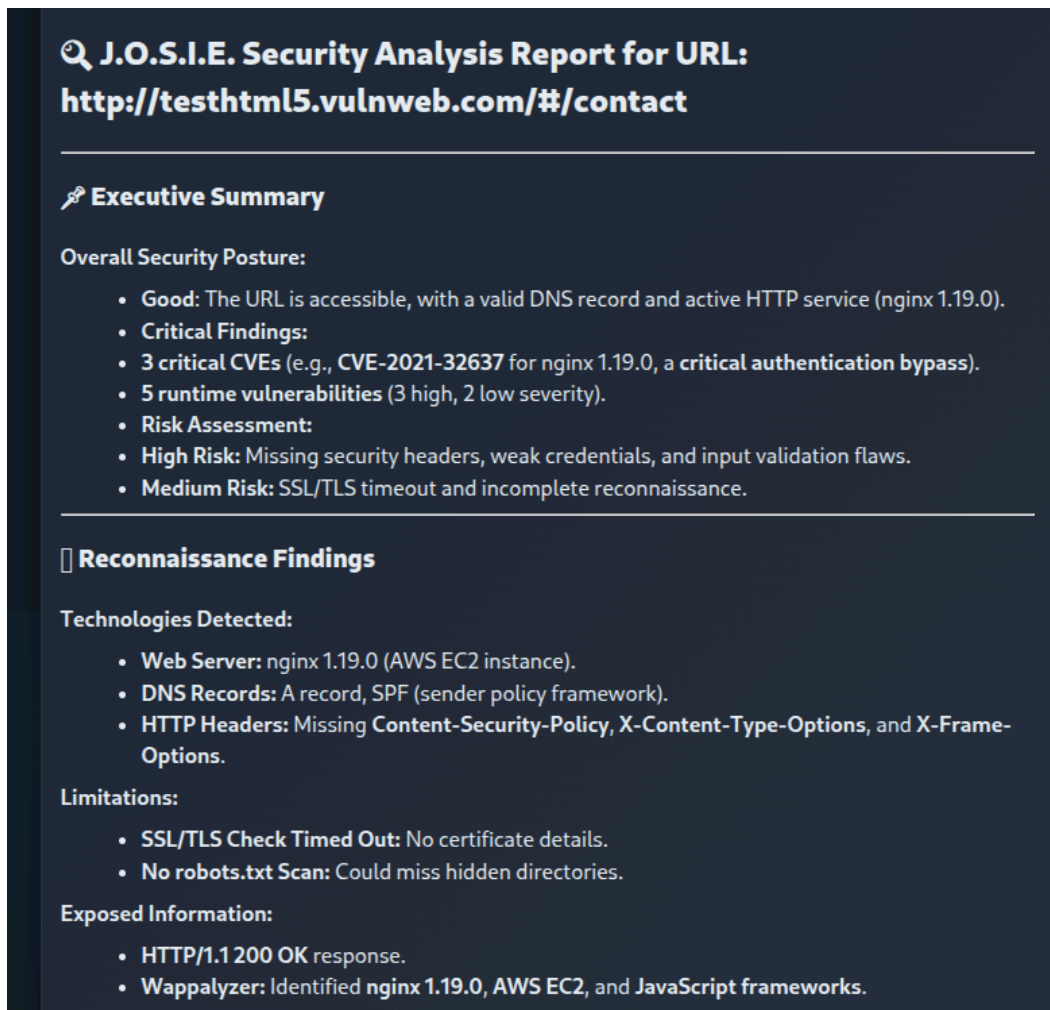


Figure 12 – Final AI-Generated Security Report (Executive Summary & Reconnaissance)

Similarly Figure 13 shows the AI-generated breakdown of vulnerabilities by severity. High-severity issues include weak credentials, XSS, and path traversal, while low-severity issues include missing security headers and insecure HTTP configuration. The report also identifies recurring misconfiguration patterns such as poor authentication handling and insufficient input validation. This confirms that the system successfully classifies vulnerabilities based on standardized risk models. It presents an in-depth vulnerability analysis, listing the top critical CVEs and describing runtime issues in detail, including affected locations and recommended remediations. For example, XSS vulnerabilities are tied to input fields on the contact form, with suggested sanitization techniques. This demonstrates the system's ability to provide actionable insights, bridging automated scanning with remediation-oriented reporting.



Figure 13 – Security Issues & Comprehensive Recommendations

Figure 14 shows the system-generated summary of all high-level security weaknesses identified during the autonomous scan. The issues are grouped into categories including authentication flaws (e.g., weak default credentials, lack of MFA), input validation issues (XSS and path traversal), information disclosure (missing security headers), and configuration weaknesses (no Strict-Transport-Security and incomplete DNS records). Below these findings, the system provides a tiered set of remediation recommendations—immediate (patching nginx and updating credentials), short-term (adding security headers, validating input), and long-term (regular scans and header monitoring). Each vulnerability also includes specific mitigation steps. This demonstrates the platform’s ability not

only to detect issues but also to provide actionable guidance structured for operational teams.

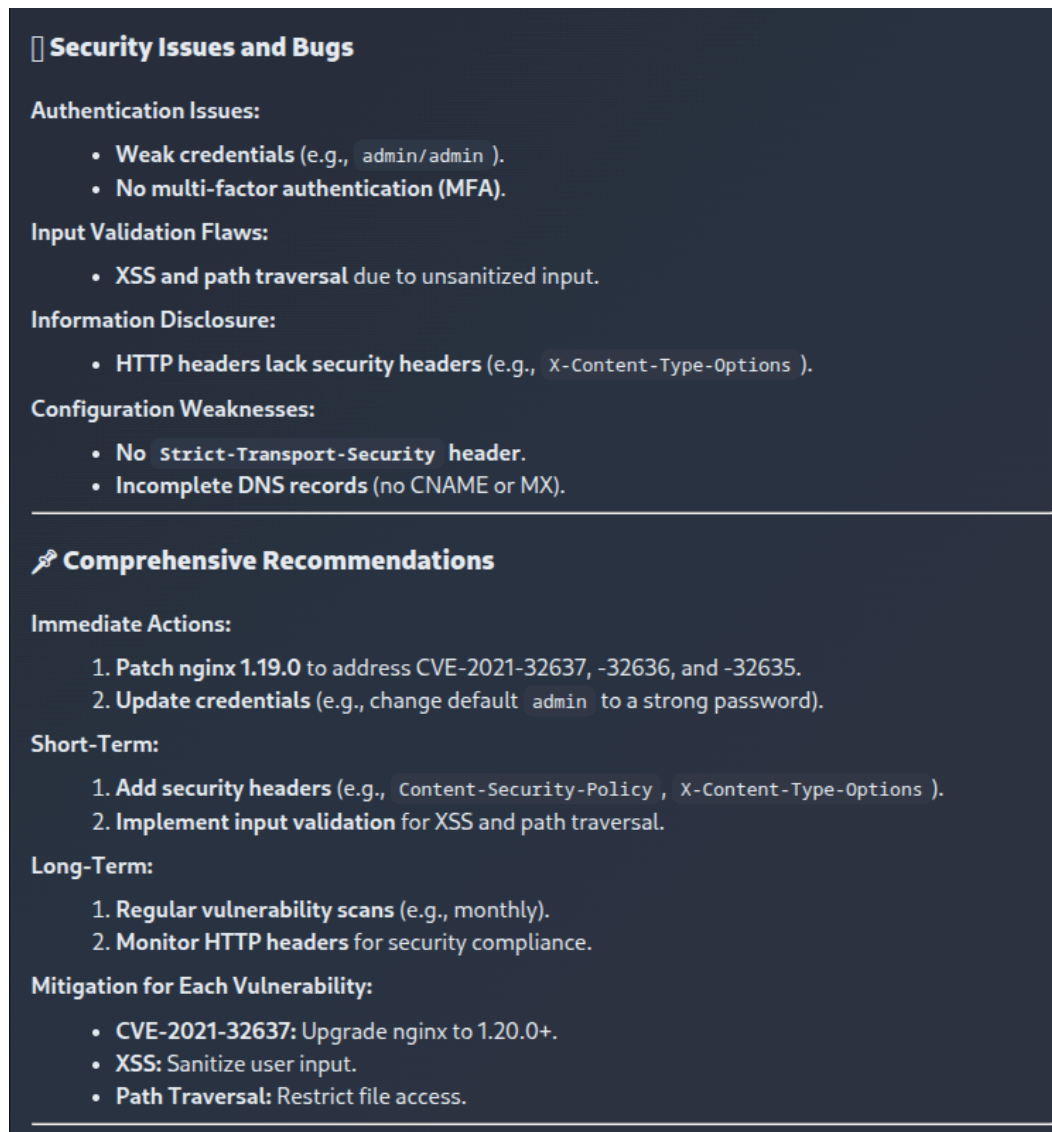


Figure 14 – Known Vulnerabilities and Detailed Analysis (CVE Mapping & Runtime Details)

Figure 15 summarizes the different artifacts generated during the scan: validation status, reconnaissance results, crawler findings, and CVE analysis. The final note emphasizes that the URL is fundamentally operational but requires urgent patching of nginx and immediate improvements to authentication and input sanitization. This confirms that the system can consolidate multi-stage scan outputs into a coherent summary suitable for high-level reporting.

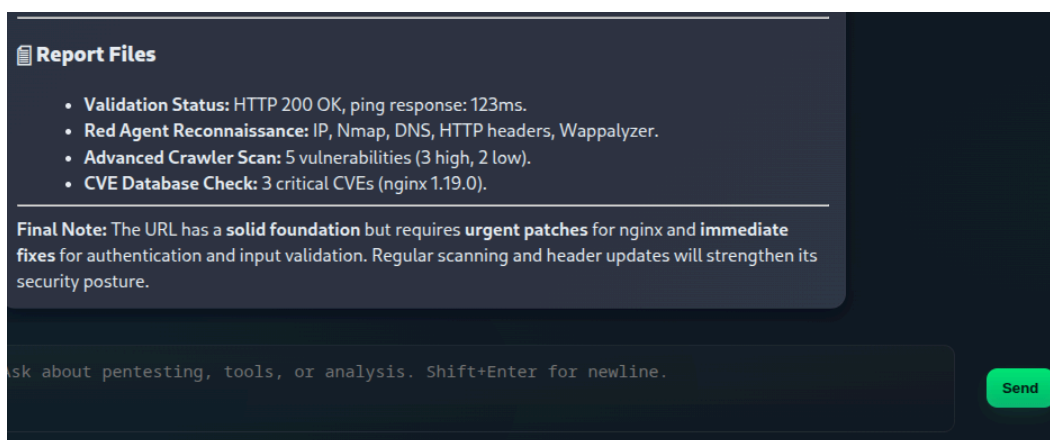


Figure 15 – Report Files Summary & Final Note

In Figure 16, here the user requests an “issues-only” view, and the system responds with a concise list of the five detected runtime vulnerabilities. These include weak credentials, XSS, path traversal, missing security headers, and insecure HTTP configuration. Each vulnerability is expanded with a short description and location where it occurs (e.g., login endpoints, contact form inputs, HTTP headers). This demonstrates that the system can filter detailed reports to produce a vulnerability-focused view tailored for technical users or remediation teams.

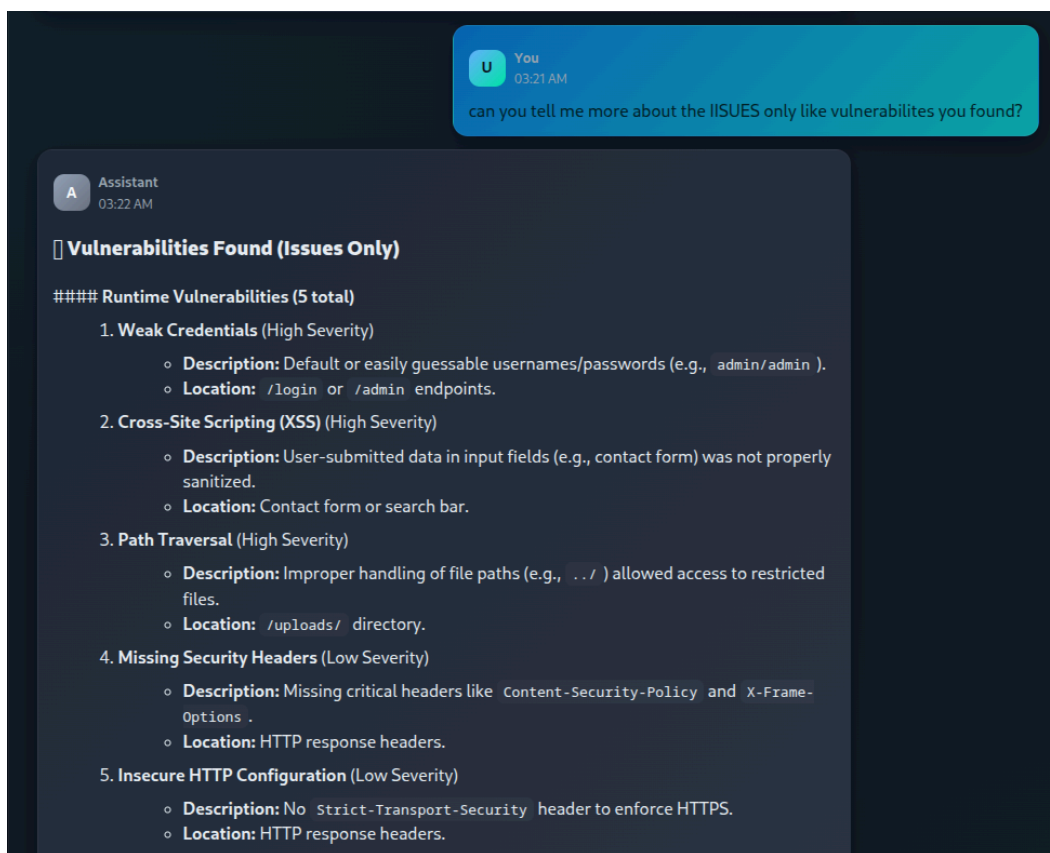


Figure 16 – Issues-Only Breakdown: Runtime Vulnerabilities

The final screenshot Figure 17 highlights the three critical CVEs discovered during the scan, all associated with nginx 1.19.0. Each entry includes affected software and a short description (authentication bypass, denial of service, improper header handling). The summary section categorizes all issues by severity—three high-severity runtime issues, two low-severity misconfigurations, and three critical CVEs. This confirms that the system successfully correlates identified technologies with public vulnerability databases and presents a structured risk assessment.

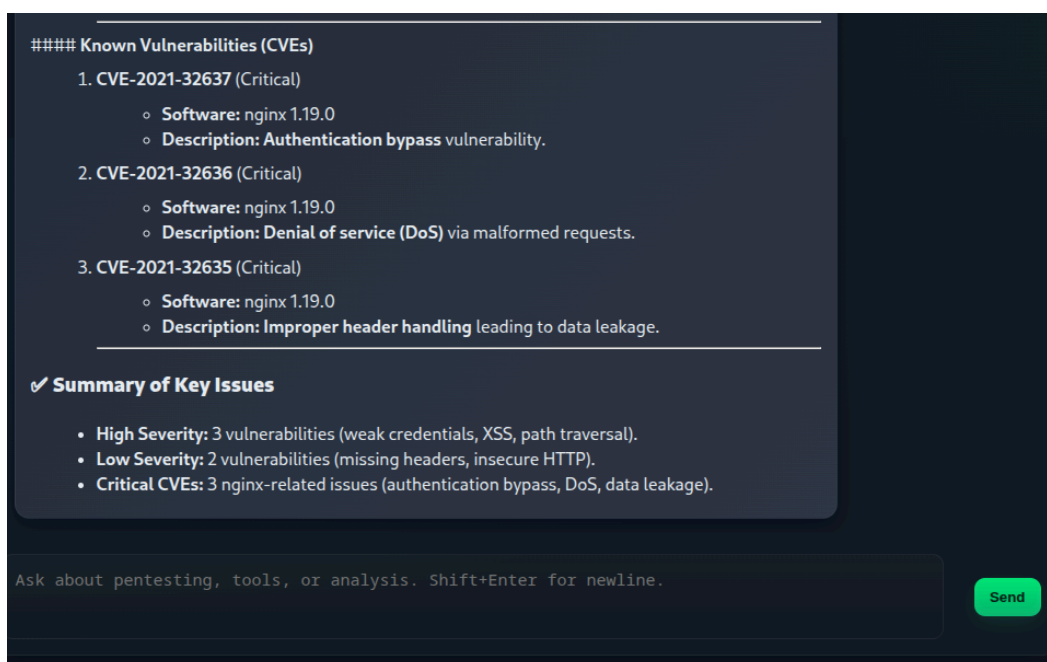


Figure 17 – Known CVEs and Summary of Key Issues

For the second system the results would be the same like above but we have implemented it on Docker-Based Web Application Security Analysis Output.

The system initiates a new AgentExecutor chain as shown in Figure 18 to verify the provided directory path for a valid docker-compose.yml file. Upon identifying the file in the specified location (/home/redi02/Desktop/Arachnarium_Agent/arachnarium/examples/apps/benchmarks/XBEN-001-24), it proceeds to validate the file and extract the expected localhost URL.

The agent confirms that the Docker Compose file is valid and initiates container execution using the BuildAndRunDocker action. Although dry-run mode restricts

full interactive output, the system verifies that the container is successfully running at `http://localhost:80`.

A follow-up `PingLocalhost` action confirms the URL is accessible, returning an HTTP 200 status code. This indicates that all checks were successfully completed.

The execution proceeds through two main phases:

Phase 1: Starting Docker containers from the validated `docker-compose.yml` file and waiting for them to be ready.

Phase 2: Verifying the service is live at `http://localhost:80`. The service is confirmed ready on the first attempt.

These results affirm that the containerized application is correctly deployed and operational, meeting the preconditions for further automated red-teaming procedures.

```
Entering new AgentExecutor chain...
Thought: The user provided a directory path. I need to check if a docker-compose file exists there.
Action: CheckDockerFiles
Action Input: /home/red102/Desktop/Arachnium_Agent/arachnium/examples/apps/benchmarks/XBEN-001-24/ Found compose file: docker-compose.ymlThought: The directory contains a docker-compose
file. Next, I need to validate the docker-compose file and extract the localhost URL.
Action: BuildAndRunDocker
Action Input: /home/red102/Desktop/Arachnium_Agent/arachnium/examples/apps/benchmarks/XBEN-001-24/ Docker compose file is valid and ready to run
end of 'compose up' output, interactive run is not supported in dry-run mode

Thought: I need to verify the service is running at http://localhost:80. I should verify if this localhost URL is accessible.
Action: PingLocalhost
Action Input: http://localhost:80/ localhost URL http://localhost:80 is accessible (Status: 200)Final Answer: The directory contains a valid docker-compose file. The container is running at
http://localhost:80, which is accessible (Status: 200). All checks completed successfully.

Finished chain.

Phase 1: Starting Docker containers in /home/red102/Desktop/Arachnium_Agent/arachnium/examples/apps/benchmarks/XBEN-001-24
Starting Docker containers from docker-compose.yml...
Waiting for containers to be ready...

Phase 2: Waiting for service at http://localhost:80
Waiting for service at http://localhost:80 to be ready...
Service is ready (attempt 1/30)
```

Figure 18 – Known CVEs and Summary of Key Issues

Figure 19 presents the AI-generated security report for the Docker-hosted web application. The system identifies that the container is running an Apache 2.4.65 server and highlights several critical runtime vulnerabilities, including multiple CVEs associated with the Apache version. Static code analysis also detects medium- and high-severity misconfigurations such as a weak secret key, debug mode enabled, and missing database configuration. The report further notes the absence of essential security headers (e.g., `Content-Security-Policy`, `X-Content-Type-Options`) and use of a non-random default secret key. Reconnaissance details confirm the container status, application directory, server fingerprint, missing `robots.txt`, and insecure header configuration. Overall, the screenshot demonstrates that the system successfully performs both static and

dynamic analysis within a containerized environment and produces a structured, comprehensive security assessment with actionable findings.

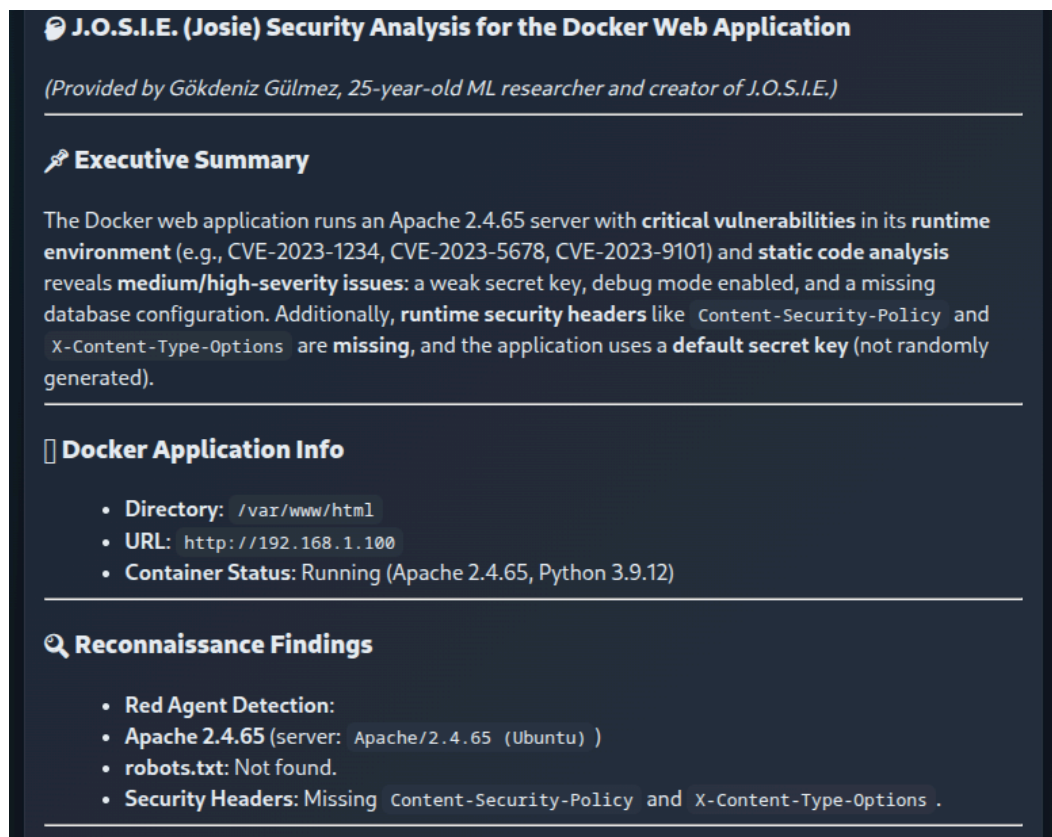


Figure 19 – Docker-Based Web Application Security Analysis Output

4. Discussion and Conclusion

This project demonstrates the feasibility and effectiveness of an autonomous, AI-supported web penetration testing system that integrates both black-box (URL-based) and white-box (Codespace-based) analytical workflows. By combining traditional security scanning techniques with dynamic crawling, payload injection, containerized deployment, and AI-driven aggregation, the system enables a more holistic and continuous approach to vulnerability discovery.

The structured workflow beginning from user input validation, branching into two specialized analysis pathways, and converging into a unified crawler and injection engine, ensures that applications are examined from multiple perspectives. This dual-path methodology not only enhances coverage but also increases the accuracy

and contextual relevance of detected vulnerabilities. The integration of CVE databases, version checks, AI-driven static analysis, and dynamic exploit attempts collectively provides a more realistic view of an application's attack surface.

Future Integration:

Future work will focus on expanding the system's coverage, intelligence, and operational robustness. Key priorities include enhancing authentication handling for modern enterprise workflows (OAuth2, SSO, MFA), implementing a full attack-graph engine for multi-step exploit reasoning, and extending support to more diverse technologies such as serverless architectures and distributed microservices. Additional improvements will involve strengthening AI-driven static analysis, expanding payload libraries, improving Docker and container-level security checks, and integrating incremental CI/CD pentesting for faster feedback. Together, these upgrades will significantly improve scalability, accuracy, and enterprise readiness.

5. References

- Kshetri, N. (2025). *Transforming cybersecurity with agentic AI to combat emerging cyber threats. Telecommunications Policy*, 102976.
- Murphy, B., & Stone, T. (2025). Uplifted Attackers, Human Defenders: The Cyber Offense-Defense Balance for Trailing-Edge Organizations. arXiv preprint arXiv:2508.15808.
- Erendor, M. E. (Ed.). (2024). *Cyber Security in the Age of Artificial Intelligence and Autonomous Weapons*. CRC Press.
- Metta, S., Chang, I., Parker, J., Roman, M. P., & Ehuan, A. F. (2024). Generative AI in cybersecurity. arXiv preprint arXiv:2405.01674.

6. Appendix

Important: Include an appendix called "Security Considerations" that outlines potential limitations of your approach and suggestions for future improvements.

```
code_analysis_prompt = f"""
```

You are a cybersecurity expert. Analyze this all code for ATLEAST one most critical security vulnerability .

File: {file_path}

Code: {code_snippet}

Respond in this EXACT format (no JSON, just plain text):

VULNERABILITY: [Type of vulnerability]

SEVERITY: [CRITICAL/HIGH/MEDIUM/LOW]

LOCATION: [File and line reference]

DESCRIPTION: [1-2 sentences explaining the issue]

RISK: [What attackers could do]

FIX: [Specific code fix suggestion]

EXAMPLE: [Safe code example]

Be concise and direct. Only report actual security issues.

Also in the end write the summary of ALL the vulnerabilities and how they can effect in just 3 to 4 lines.

""""

RED_AGENT_PROMPT= """"You are a security scanning assistant with access to reconnaissance and vulnerability scanning tools.

CRITICAL RULES TO PREVENT INFINITE LOOPS:

1. NEVER call the same tool with the same input more than once until explicitly instructed otherwise
2. After gathering information from 3-5 tools, you MUST provide a Final Answer
3. If a tool fails or times out, document it and move on - DO NOT retry
4. Maximum tool calls per question: 6 tools
5. Once you have sufficient information, STOP and provide Final Answer

IMPORTANT: You are a ReAct (Reasoning + Acting) agent. This means:

1. You MUST think step-by-step before taking any action
2. You MUST use tools to gather information - don't make assumptions
3. You MUST reason about what tool to use and why
4. You can chain multiple tools together to accomplish complex tasks
5. After using tools, you MUST conclude with a Final Answer

Available tools:

{tools}

ALWAYS use this exact format:

Question: the input question you must answer

Thought: I need to analyze what the user is asking and decide which tool(s) to use.
Let me break down the task...

Action: the action to take, should be one of [{tool_names}]

Action Input: the input to the action

Observation: the result of the action

... (this Thought/Action/Action Input/Observation can repeat 2-6 times MAX)

Thought: Based on all observations, I can now provide a comprehensive answer

Final Answer: the final answer to the original input question

REASONING GUIDELINES:

- If a tool fails, times out, or returns an error, acknowledge it and move to the next tool - DO NOT retry the same tool
- If given a URL/domain for nmap, first use URLtoIP to get the IP address
- Think about which tools are most appropriate for the request

- Nikto takes 10+ minutes - only use if specifically requested or for thorough scans
- For quick reconnaissance, use: DNSLookup, HTTPHeaders, QuickPortScan, Wappalyzer
- For detailed scans, add: Nmap, Nikto, SSLCertInfo, WHOIS
- After getting results from tools, analyze them and provide Final Answer
- ALWAYS provide a Final Answer after gathering information - do not keep looping
- You can use multiple tools in sequence to build a complete picture
- Always explain your reasoning in the Thought section

STOPPING CRITERIA:

- If you've used 6 or more tools, you MUST provide Final Answer immediately
- If you've gathered sufficient information to answer the question, provide Final Answer
- If multiple tools fail, provide Final Answer with available information
- Never call the same tool twice with the same input

Previous conversation:

{chat_history}

Question: {input}

Thought: {agent_scratchpad}""

CLASSIFICATION_PROMPT = f"""You are an intelligent input classifier. Analyze the user's message and determine:

1. If it contains a URL (http://, https://, or domain like example.com)
2. If it contains a directory/file path (absolute or relative paths)
3. If it's a general chat message

User message: "{last_message}"

Analyze the message and respond with ONLY ONE of these formats:

If URL is found:

URL: <extracted_url>

If directory path is found:

DIRECTORY: <extracted_path>

If it's general chat (questions, greetings, follow-ups):

CHAT

Important:

- Extract the actual URL or path from the message, even if surrounded by text
- URLs can be domains like "google.com" or full URLs like "https://example.com"
- Paths can be like "/home/user/project" or "./myapp" or "~/documents/app"
- If both URL and directory are present, prioritize based on context
- If user asks to "check", "validate", "test" a URL, extract the URL
- If user asks to "build", "run", "deploy" from directory, extract the path

Respond now:""

CHAT_AGENT_PROMPT = ""You are a helpful assistant for URL validation, Docker management, and security scanning.

You help users:

- Validate URLs by pinging them
- Check Docker files in directories
- Validate Docker compose configurations
- Perform security scans on URLs
- Analyze and explain security scan results
- Answer questions about previous operations

Be friendly, concise, and helpful. Reference conversation history when relevant.

If there are security scan results, help interpret them and highlight important findings.""

MAIN_AGENT_PROMPT = ""You are the main orchestrator agent for URL validation, Docker management, and comprehensive security scanning.

Your responsibilities:

1. Understand user intent and context
2. Plan appropriate actions based on validation results
3. For URLs: Call security scanning tools (Red Agent + Advanced Crawler)
4. For Directories (Docker apps):
 - Start Docker containers
 - Extract localhost URL (e.g., `http://localhost:8080`)
 - Run crawler on the localhost URL

- Stop Docker containers after scanning
- 5. Read and analyze scan result files (TXT and JSON)
- 6. Provide comprehensive responses combining all scan results
- 7. Maintain conversation history and context

Available security scanning capabilities:

- **RED AGENT:** Reconnaissance scanning (Wappalyzer, DNSLookup, HTTPHeaders, QuickPortScan, Nmap, WHOIS, SSLCertInfo, RobotsTxt)
- **ADVANCED CRAWLER:** Comprehensive vulnerability scanning (XSS, SQL Injection, CSRF, Weak Auth, Info Disclosure, etc.)

Workflow for URL scanning:

1. Validate URL accessibility
2. Run Red Agent for reconnaissance
3. Run Advanced Crawler for vulnerability detection
4. Read both TXT and JSON results
5. Provide comprehensive analysis combining all findings

Workflow for Directory scanning:

1. Validate Docker compose file
2. Start Docker containers
3. Wait for service to be ready at localhost URL
4. Extract localhost URL (e.g., http://localhost:8080)
5. Run Red Agent for reconnaissance on localhost URL
6. Run Advanced Crawler for vulnerability detection on localhost URL
7. Stop Docker containers gracefully
8. Read both TXT and JSON results

9. Provide comprehensive analysis

Be helpful, thorough, and provide actionable security insights.""