International Journal of Web of Multidisciplinary Studies
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
website: *https://ijwos.com*
Volume 03, Issue 10,October 2026
OPEN ACCESS
E-ISSN : 3049-2424
DOI: https://doi.org/10.71366/ijwos

WEB-BASED RECONNAISSANCE AND VULNERABILITY SCANNER

Mrs. Tara V. K. (Guide)[1], Chetan B N[2], Akashraj V T[3], Evans J[4], Md. Faizan[5]
[1,2,3,4,5] *Department of Computer Science and Engineering, City Engineering College, Bengaluru, India*

Article Info

Article History:
Published: 27 Oct 2026

Publication Issue:
Volume 03, Issue 10
October 2026

Page Number:
430-437

Corresponding Author:
Chetan B N

Abstract:
Open ports, outdated services and unpatched software expose networked systems to attack, and checking for them by hand is slow and difficult to repeat across many hosts. Reconnaissance (finding live hosts, open ports and running services) and vulnerability assessment (comparing the discovered software with known weaknesses) are normally performed with separate command-line tools, so their results end up scattered and inconsistently presented. This paper describes a web-based scanner that combines target management, reconnaissance and vulnerability assessment in one browser interface. The system has three parts: a Django dashboard where users register targets and start scans; a controller that communicates with a lightweight agent over a TCP request–response protocol; and the agent itself, which performs port scanning, service and network discovery, and software vulnerability checks. Detected software is matched against vulnerability reference data, and each finding receives a severity rating and appears in a scan report. These matches are based on software versions, so they are candidate findings, not confirmed exploitable vulnerabilities. The paper presents the architecture, the scanning workflow and the outputs the dashboard produces. It does not benchmark the scanner against other tools or report detection accuracy.

Keywords: Web-based Security, Network Reconnaissance, Vulnerability Scanning, Common Vulnerabilities and Exposures (CVE), Port Scanning, Service Discovery, Risk and Severity Assessment, Security Dashboard

1. Introduction

Every host that accepts network connections is a possible entry point. Common weaknesses include open ports behind outdated services, misconfigured settings and software with publicly known flaws, and none of them goes away until someone finds it. Checking even a handful of machines by hand is tedious, and the approach does not scale.

Finding these weaknesses involves two activities. *Reconnaissance* collects information about a target: which hosts are reachable, which ports are open, and which services and software versions run behind them. *Vulnerability assessment* then compares that information with known vulnerability data, most often Common Vulnerabilities and Exposures (CVE) entries, to identify and rank weaknesses. Mature tools exist for both, but they are usually run separately from the command line, and someone has to copy, compare and interpret the output before it is useful to a security team.

For a small team or a student project, this separation causes two problems. The first is repeatability: without a common place to keep results, a scan cannot easily be compared with an earlier scan of the same host. The second is readability: raw scanner output is technical, and turning it into something a non-specialist can act on, such as a list of vulnerabilities ordered by severity, takes extra manual effort. Keeping targets, scan results and findings together behind one browser interface addresses both.

This project implements that idea. A user registers a target in the web dashboard, starts a reconnaissance and vulnerability scan, and views the open ports, detected services and identified vulnerabilities on the same pages. The work is divided among three components: the dashboard, a controller that communicates with target systems, and a lightweight agent that performs the scanning and returns the results. Each finding carries a severity rating so that the most serious issues can be handled first, and the platform is designed to generate a report after each scan.

Copyright © 2026 The Author(s) : This is an open access article under the CC BY license (http://creativecommons.org/licenses/by-nc/4.0/)

The project does not introduce a new scanning technique or vulnerability database. Its contribution is the combination of reconnaissance, vulnerability identification and reporting in one web-accessible tool built on a defined controller–agent protocol, implemented as a working final-year project. Section II reviews related work, Sections III and IV describe the methodology and architecture, Section V presents the system's outputs, and Section VI discusses limitations and future work.

2. RELATED WORK

Nmap is a widely used network reconnaissance tool. By sending crafted packets and analysing the replies, it identifies live hosts, open ports and running services, and often the operating system [1]. Most reconnaissance modules, including the port and service discovery module in this project, rely on the same basic idea: probe a port, observe how the target responds, and use banners or response patterns to infer the service and version behind it.

A known service and version can then be checked against published weaknesses. The CVE program run by the MITRE Corporation assigns identifiers and short descriptions to publicly disclosed vulnerabilities, so that different tools and organisations can refer to the same issue consistently [2]. The National Vulnerability Database (NVD), maintained by the U.S. National Institute of Standards and Technology, builds on the CVE list by adding structured technical detail, affected-product information and a severity score to each entry [3]. A scanner that reports more than open ports needs a reference of this kind to link a detected software version to a known weakness. Such a link shows only that the version is listed as affected; it does not show that the weakness can be exploited on the scanned host.

Severity is commonly expressed with the Common Vulnerability Scoring System (CVSS), an open framework originally developed under the U.S. National Infrastructure Advisory Council and described by Mell, Scarfone and Romanosky [4]. CVSS converts the technical characteristics of a vulnerability into a numeric score and a qualitative rating of Low, Medium, High or Critical. The framework has since been revised, and version 4.0 is maintained by the Forum of Incident Response and Security Teams (FIRST) [5]. Such ratings allow a scanner to present findings in priority order rather than as a flat list. The severity ratings in this project use the same four-level Low/Medium/High/Critical scale.

Several studies have examined how existing scanners perform. Amankwah et al. compared eight commercial and open-source web vulnerability scanners, including OWASP ZAP, Skipfish and Arachni. They found that the vulnerability classes each tool detects vary considerably, with some open-source tools competitive with commercial ones in certain classes [6]. A closer comparison of two open-source scanners reported that detecting SQL injection and cross-site scripting remained difficult [7]. The OWASP Web Security Testing Guide is commonly used as a checklist of the vulnerability categories that a web-facing assessment should cover [8].

These studies concentrate on how well scanning engines detect particular vulnerability classes. They do not address how scan results are stored, tracked across repeated scans or presented to users who are not security specialists, which is the area this project addresses. The project makes no attempt to outperform the tools above on any vulnerability class. Instead, it combines agent-based reconnaissance, a CVE-referenced vulnerability check and a web dashboard for target management and reporting. None of the sources cited here covers this combination, although the review is not exhaustive.

3. METHODOLOGY

The system has three parts: a web dashboard, a controller and an agent. The dashboard is a Django web application in which users register targets, start scans and view results; it also exposes a REST API for retrieving scan data programmatically. The controller communicates with the target systems and sends commands to the agent using a request–response protocol over TCP. The agent is a lightweight process that runs on or against the target, contains the reconnaissance and vulnerability-checking logic, and returns its results to the controller, which stores them through the dashboard's data model. The scanning workflow has four stages, described in Algorithms 1–4 and summarised in Fig. 1 at the end of this section.

A. Algorithm 1: Target Registration and Reconnaissance Initiation

Input: target address (IP address or hostname) and a scan profile.

Output: a scan job record and an established channel to the target agent.

1. The user adds a target in the dashboard, supplying its IP address or hostname (required) and an optional group or tag.
2. The user selects a scan profile (quick, standard, or deep/custom) and starts the scan.
3. The dashboard creates a scan job record for the target and forwards the request to the controller.
4. The controller connects to the agent listening on the target's designated port and sends a request containing a command code and a unique request identifier.
5. The agent accepts the connection and prepares to run the reconnaissance or assessment module selected by the command code.
6. If the agent cannot be contacted (for example, the target is offline or the port is closed), the scan job fails, and the reason is recorded in the job record.

B. Algorithm 2: Port and Service Discovery

Input: target address and a port range or list determined by the selected scan profile.

Output: a list of open ports with the service and, where available, the software banner found on each.

1. The agent works through the ports in the scan profile: a common port set, or the full range for a deep scan.
2. The agent attempts to connect to each port and records whether it is open, closed or filtered.
3. For each open port, the agent retrieves a service banner or otherwise determines the protocol and, where available, the name and version of the software running behind it.
4. Active hosts and network interfaces are discovered through ARP-based network discovery, together with the devices and services found.
5. The collected port, service and device information is packaged in the agent–controller response format and returned to the controller.
6. The controller stores the data in the target and networkdevice records for use in the vulnerability assessment phase and by the dashboard.

C. Algorithm 3: Vulnerability Assessment

Input: the detected services, software and versions produced by Algorithm 2 for a target.

Output: a list of candidate vulnerabilities for the target, each matched to a known vulnerability reference.

1. The vulnerability scanning module compares the detected software products and versions with the system's vulnerability reference data.
2. If a detected version falls within a known vulnerable version range, a vulnerability record is created linking the detected version, the vulnerable range, the target and the CVE identifier (where one exists).
3. In addition to the service-based checks, the module reviews selected security-relevant configuration items exposed by the agent (for example, open high-risk ports and basic firewall or service exposure) for common misconfigurations.
4. Each finding is stored in the database with a description, the affected target, and the software or configuration item concerned, so that the same finding can be tracked across repeated scans.
5. New results are compared with those of the previous scan of the same target, so that each finding's status (first seen, still present or resolved) is maintained rather than only the latest result.

D. Algorithm 4: Severity Analysis and Reporting

Input: vulnerability results from Algorithm 3.

Output: severity-rated findings and a printable scan report.

1. Each finding receives a qualitative severity rating (Critical, High, Medium or Low) based on the underlying vulnerability or misconfiguration.

2. Findings for a target are aggregated into an overall picture of its exposure, for example the number of findings at each severity level.
3. The dashboard displays results grouped and sorted by severity, together with the reconnaissance data (open ports, services and devices) gathered for the same target.
4. The user can generate a report summarising the scan. The report is designed to be exportable (for example as a structured data file or a document) so that it can be shared with stakeholders.
5. Reports and findings are stored with the target record so that they can be compared with later scans of the same target.

E. Overall Workflow

From start to finish, a scan passes through the web dashboard, target registration, the controller, the agent (on or against the target), port and service discovery, network device discovery, vulnerability and CVE matching, severity classification, storage in the database, and finally dashboard visualisation and report generation. Fig. 1 groups these steps into three stages: reconnaissance, vulnerability scanning and report generation.

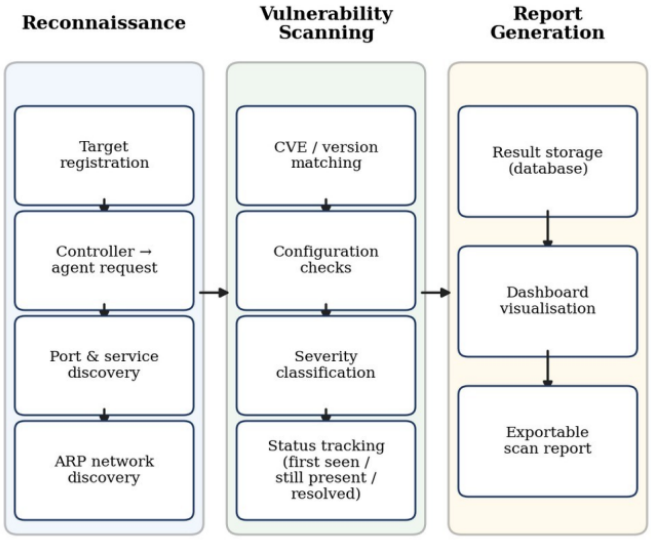

Fig. 1. Overall workflow, divided into reconnaissance, vulnerability scanning and report generation.

4. SYSTEM ARCHITECTURE

The implemented system has four components, shown in Fig. 2:

- **Web Dashboard:** a Django application for target management, scan initiation and viewing vulnerabilities, with a REST API for programmatic access to target, scan and vulnerability data.
- **Controller:** coordinates scanning. It maintains the target inventory, sends scan commands to agents, and collects and stores the results they return.
- **Agent:** a lightweight process that runs on or against the target and exposes a TCP listener. It implements the reconnaissance and vulnerability-checking modules and returns results to the controller in the format defined by the protocol.
- **Database:** a relational database holding target and target-group records, scan jobs and results, discovered network devices and services, and vulnerability findings, so that scan history can be stored and compared over time.

The dashboard passes scan requests to the controller, and the controller and agent exchange messages over TCP using a defined binary request–response protocol. A request contains a request identifier and a command code that selects the operation; the response carries the result, for example the ports found, the services found or vulnerability data.

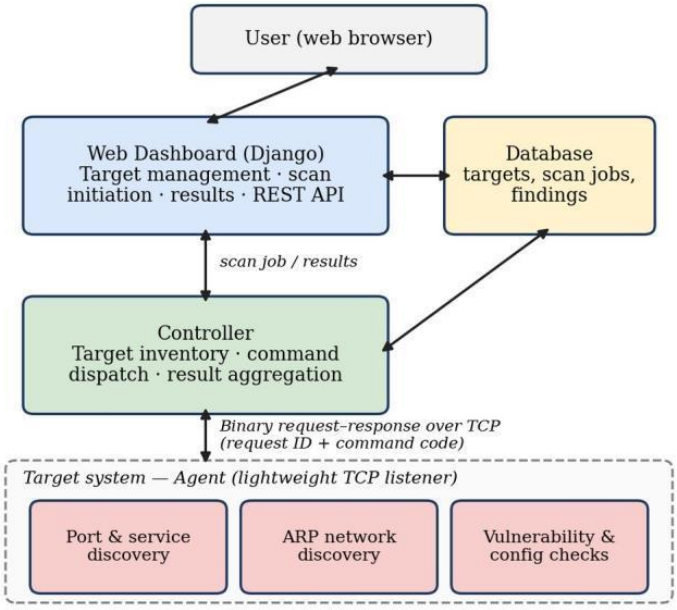

Fig. 2. System architecture of the web-based reconnaissance and vulnerability scanner.

5. RESULTS AND DISCUSSION

A. System Output

This section describes the outputs that the implemented system produces at each stage of a scan. Table I lists each analysis component and the output that the dashboard presents for a target.

TABLE I. SYSTEM OUTPUT BY ANALYSIS COMPONENT

Analysis component	**Output**
Target information	Registered IP address or hostname, target group and tags, and whether the agent is reachable.
Reconnaissance	Network devices discovered from the target using ARP-based discovery.
Port scanning	Open, closed and filtered ports, according to the selected scan profile.
Service detection	Service name and, where known, software banner or version for each open port.
Vulnerability assessment	Possible vulnerabilities identified for detected software versions and configuration checks.
CVE information	CVE identifier and description for each matched vulnerability, where the reference data contains a match.
Severity / risk analysis	Critical / High / Medium / Low rating for each finding, with per-target counts.
Final report	Exportable scan report with target information, reconnaissance data and prioritised findings.

B. Dashboard Screenshots

Figs. 3–5 are screenshots taken from the implemented dashboard. Fig. 3 shows the overview page, which displays counts of registered target systems, network devices, total scans, open ports, security alerts, vulnerabilities and system issues, with shortcuts to the matching scan functions. Fig. 4 shows the system information view, which lists operating-system details, boot time and patch status, along with categories of security-related information. Fig. 5 shows the network security alerts view, which can be filtered by severity, type and status; in this capture it lists possible port-scan alerts.

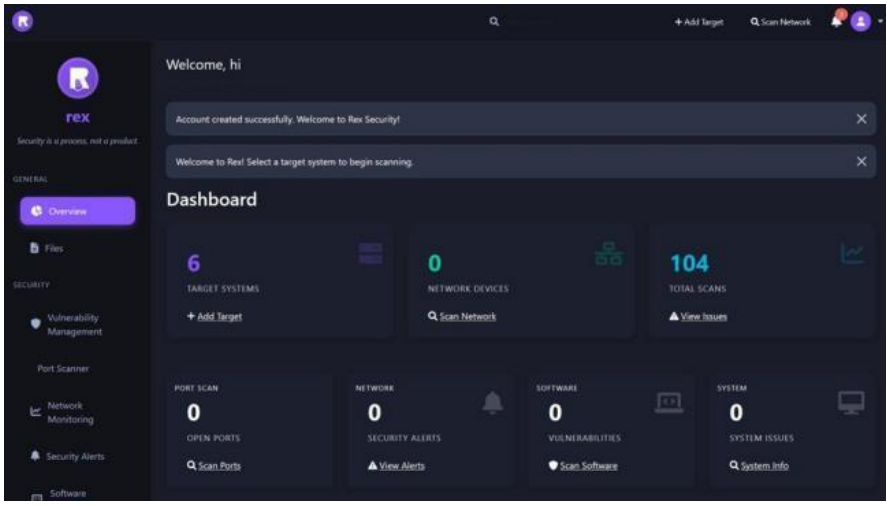

Fig. 3. Dashboard overview page.

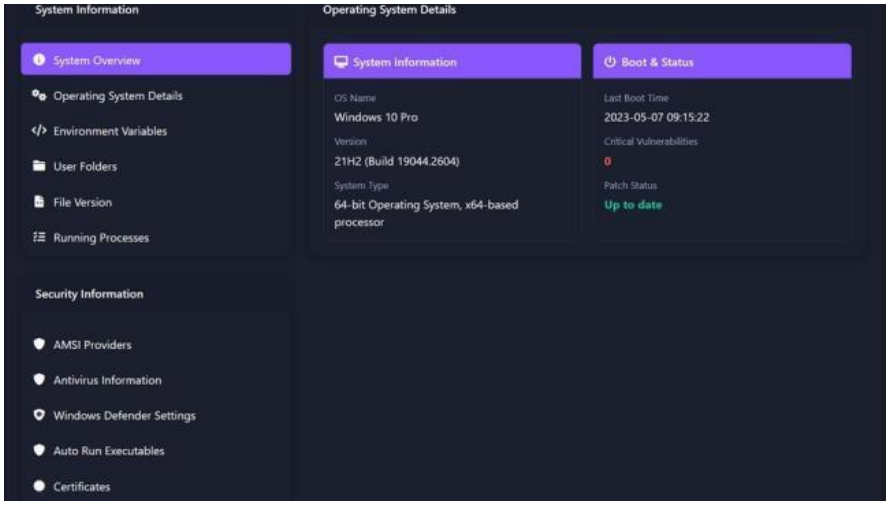

Fig. 4. System information view.

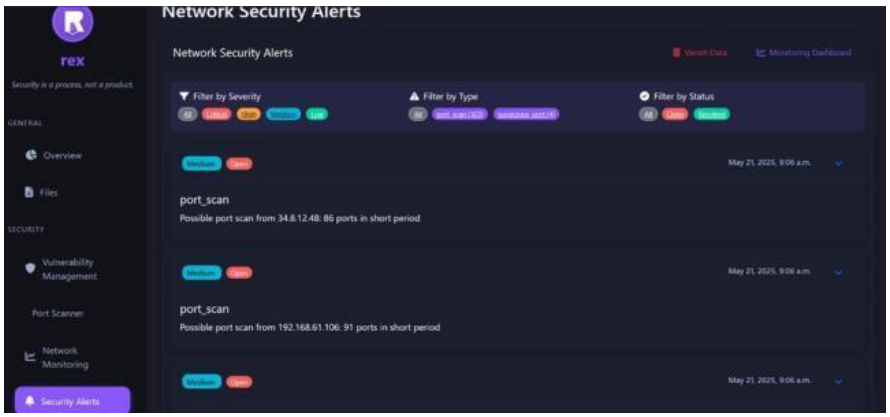

Fig. 5. Network security alerts view with severity, type and status filters.

C. Experimental Evaluation

Table II gives summary metrics for the test targets used to exercise the system. These are counts and an average scan time only; they are not a benchmark and are not compared with other scanners.

TABLE II. EXPERIMENTAL EVALUATION METRICS

Metric	**Result**
Number of targets tested	6
Open ports detected	10
Services identified	10
Vulnerabilities detected	9
Critical vulnerabilities	1
High vulnerabilities	2
Medium vulnerabilities	0
Low vulnerabilities	3
Average scan time	35 minutes

D. Performance and Complexity

Port scanning dominates the cost of reconnaissance. Scanning p ports on one host requires a separate connection attempt for each port, so scan time grows roughly in proportion to p. Three factors determine the actual duration: the number of ports in the scan profile (a quick scan of a short list of common ports is faster than a full-range scan); the per-port timeout used to decide that a port is filtered rather than open, since an unanswered port still costs a connection attempt or a timeout; and the network latency to the target, which the agent–controller design does not remove. This discussion describes the expected behaviour of the scan procedure.

Once reconnaissance data is available, vulnerability matching is comparatively cheap. It works on a small number of software–version pairs per target, so its cost follows the number of detected services rather than the size of the vulnerability reference data. Report generation and dashboard rendering are expected to grow with the number of findings stored for a target, but they should not limit performance compared with the scanning stages. Targets are scanned sequentially over a single agent connection, so a job covering many targets will take longer than one covering a single target; this is one of the limitations discussed in Section VI.

6. CONCLUSION

This paper presented a web-based scanner that lets users register targets, run network reconnaissance, discover ports and services, and carry out CVE-referenced vulnerability assessment from a single dashboard, using a controller–agent architecture and a defined protocol. Reconnaissance and vulnerability identification are usually done with separate command-line tools. The implementation shows that both can be driven from one interface and that the results can be stored, so a target's security posture can be followed across repeated scans rather than judged from a single one.

The scanner has clear limits. Users still need the skill to interpret its results, it cannot guarantee that every vulnerability class is detected, and it does not by itself make a network secure. Its findings come from matching detected versions and configuration items against reference data, so they are candidate vulnerabilities rather than confirmed, exploitable ones. Like any automated scanner, it can produce false positives and false negatives, and its coverage is only as good as the reference data it uses. In practice, an agent process must be reachable on the target, large sets of targets are

scanned one after another, and coverage depends on keeping the reference data current. The tool is meant for authorised use against systems that the user owns or has explicit permission to test

Future work, proposed but not yet implemented, includes widening vulnerability coverage by regularly updating the reference data from public CVE/NVD feeds, scanning many targets concurrently, adding stronger authentication between the dashboard, controller and agent, supporting richer report formats, and improving the visualisation of trends across repeated scans of a target.

References

[1] G. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Sunnyvale, CA, USA: Nmap Project, 2009.

[2] The MITRE Corporation, "Common Vulnerabilities and Exposures (CVE)." [Online]. Available: https://cve.mitre.org/

[3] National Institute of Standards and Technology (NIST), U.S. Department of Commerce, "National Vulnerability Database (NVD)." [Online]. Available: https://nvd.nist.gov/

[4] P. Mell, K. Scarfone, and S. Romanosky, "The Common Vulnerability Scoring System (CVSS) and its applicability to federal agency systems," NIST Interagency Report 7435, National Institute of Standards and Technology, Gaithersburg, MD, USA, 2007.

[5] FIRST.Org, Inc., *Common Vulnerability Scoring System Version 4.0 Specification Document*. Forum of Incident Response and Security Teams, 2023.

[6] R. Amankwah, J. Chen, P. K. Kudjo, and D. Towey, "An empirical comparison of commercial and open-source web vulnerability scanners," *Software: Practice and Experience*, vol. 50, no. 9, pp. 1842–1857, 2020.

[7] B. Zukran and M. M. Siraj, "Performance comparison on SQL injection and XSS detection using open source vulnerability scanners," in *Proc. 2021 Int. Conf. Data Science and Its Applications (ICoDSA)*, Bandung, Indonesia, 2021, pp. 61–65.

[8] OWASP Foundation, *OWASP Web Security Testing Guide*. [Online]. Available: https://owasp.org/www-project-websecurity-testing-guide/