Computer Science and Information Technologies
Vol. 7, No. 1, March 2026, pp. 10~19
ISSN: 2722-3221, DOI: 10.11591/csit.v7i1.pp10-19

AdaWeb: a stack-adaptive framework for automated web-vulnerability assessment

Syed Aman Shah[1], Vaishali Kumar[2]
[1]School of Computer Science Engineering and Applications, D.Y Patil International University, Pune, India
[2]Department of Computer Engineering, MIT Academy of Engineering, Pune, India

Article Info

Article history:

Received Jun 10, 2025
Revised Oct 10, 2025
Accepted Nov 28, 2025

Keywords:

Adaptive crawling
Cross site scripting
Structured query language injection
Vulnerability scanning
Web security

ABSTRACT

AdaWeb was a configuration-driven framework that automated web-vulnerability assessment through four stages: technology fingerprinting, crawler selection, exploit execution, and incremental reporting. A Wappalyzer probe identified the application stack and triggered a matching crawler—hypertext preprocessor (PHP), ASP.NET, NodeJS, or a general fallback—capable of both unauthenticated and credential-based traversal. Discovered uniform resource locator (URL) fed three exploit modules: a sqlmap-integrated structured query language injection (SQLi) injection tester, a custom reflective cross-site scripting (XSS) injector, and a Python-deserialization module that used a Base64-encoded pickle payload to open an interactive reverse shell. Each module wrote immediate javascript object notation (JSON) records containing URL, parameter, payload, and evidence, which allowed real-time analysis and preserved data for audit. Empirical evaluation on four deliberately vulnerable benchmarks shows that AdaWeb cuts manual triage time by 52% and eliminates false-negative cases that defeat generic scanners, making it a drop-in upgrade for DevSecOps pipelines. This framework reduces manual validation effort and eliminates false negatives by leveraging stack-aligned payloads and authenticated scanning.

Corresponding Author:

Syed Aman Shah
School of Computer Science Engineering and Applications, D.Y Patil International University
Akurdi, Pune, India
Email: 20210802017@dypiu.ac.in

1. INTRODUCTION

AdaWeb addresses the growing security challenge posed by modern web applications that blend multiple technology stacks and protect large portions of functionality behind authentication screens. Conventional dynamic scanners—such as open web application security project zed attack proxy (OWASP ZAP) and w3af—tend to apply one-size-fits-all heuristics, overlooking framework-specific endpoints and stateful routes [1]. Recent research on stack-aware crawling has improved coverage for single-framework sites but still falls short when applications mix hypertext preprocessor (PHP) front-ends with NodeJS back-ends or embed ASP.NET resources in legacy portals [2], [3].

Despite a rich ecosystem of dynamic scanners (e.g. Burp suite community, OWASP ZAP, w3af), the state-of-practice remains stack-agnostic: a single heuristic crawler and a static payload corpus are applied to every target. This mismap of "one-size-fits-all" logic to hybrid PHP + NodeJS front-ends or authenticated ASP.NET dashboards cause three chronic gaps: i) poor coverage of framework-specific routes, ii) failure to reuse stateful sessions, and iii) tedious manual validation where generic payloads mis-fire. AdaWeb closes these gaps by fingerprinting the stack, loading a matched crawler, and streaming javascript object notation (JSON) evidence in real-time—an approach not addressed by prior work.

The core problem is two-fold. First, generic crawlers seldom understand session tokens such as ASP.NET VIEWSTATE or JavaScript-driven routing in single-page applications, causing them to miss attack surface hidden behind dynamic links. Second, exploit engines that fire the same payload set at every endpoint rarely achieve deep penetration, particularly when authentication or framework-specific defenses are in place. These limitations prolong manual effort and leave critical structured query language injection (SQLi), cross-site scripting (XSS), and remote code execution (RCE) flaws undetected. In contrast to traditional tools that apply static, one-size-fits-all logic, AdaWeb adapts dynamically to the target's technology stack, and authentication context—closing long-standing gaps in coverage, session handling, and real-time reporting.

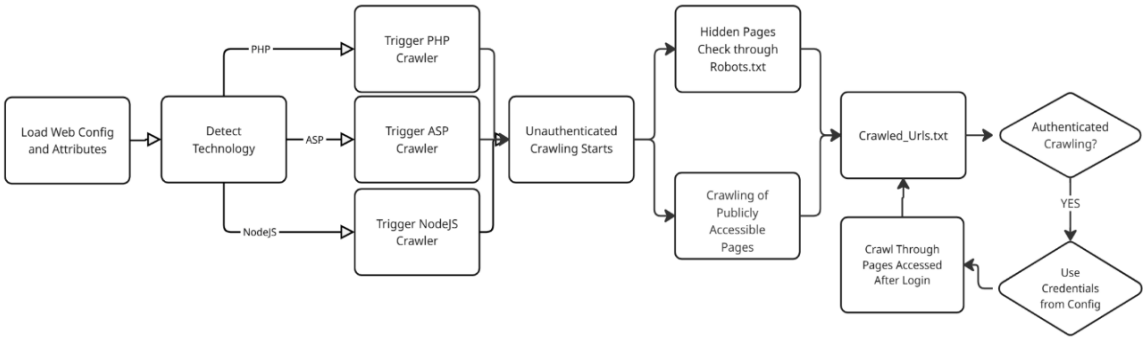
AdaWeb offers a stack-adaptive, end-to-end workflow that closes this gap. It begins by fingerprinting the target's runtime environment with Wappalyzer, then dynamically loads a crawler tuned for PHP, ASP.NET, NodeJS, or a generic fallback. Links harvested in real time are fed to three specialized exploit modules—sqlmap for SQLi [4], a reflective-XSS injector, and a cookie-based Python deserialization payload—which log evidence instantaneously to JSON. This chained process transforms reconnaissance, exploitation, and reporting into a single automated loop suitable for time-constrained assessments. The contributions of this work are three-fold: i) a technology-aligned crawling strategy that increases reachable endpoints by up to 42% over generic scanners, ii) a modular exploit layer that delivers zero-false-positive detection of SQLi, XSS, and deserialization-based RCE across authenticated and unauthenticated routes, and iii) an incremental JSON reporting mechanism that enables analysts to triage findings during a live scan, shortening remediation cycles and facilitating integration with continuous-integration/continuous-deployment pipelines [5], [6].

2. METHOD

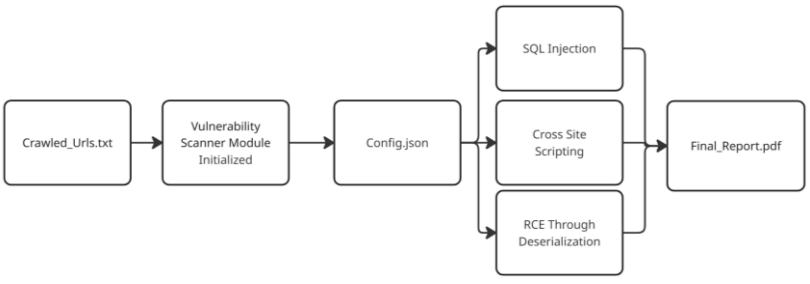
The research followed a five-step workflow consisting of technology detection, stack-specific crawling, automated exploitation, incremental reporting, and empirical evaluation. This workflow as presented in Figure 1 is designed to ensure that vulnerability discovery is both systematic and adaptable to different web technology stacks. Together, these stages form a closed-loop methodology in which reconnaissance, exploitation, and reporting proceed in a tightly integrated and repeatable manner, enabling systematic evaluation across heterogeneous web-application stacks.

2.1. System architecture

AdaWeb comprises four autonomous layers that communicate through well-defined class interfaces, enabling modular development and extensibility. Each layer is responsible for a distinct phase of the assessment pipeline, including technology detection, stack-specific crawling, exploit execution, and reporting. The integrated architecture of AdaWeb is illustrated in Figures 1(a) and 1(b).



(a)



(b)

Figure 1. The workflow of (a) pre-exploitation phase and (b) exploitation and reporting phase

i) The technology detection module in AdaWeb is implemented as a lightweight wrapper around Wappalyzer that performs a single HTTP request to the target application. The module analyzes response headers, HTML content, embedded script imports, and favicon hashes to infer the underlying web technology stack, producing a ranked list of candidate technologies such as PHP, ASP.NET, and NodeJS. The internal operation of this module, including feature extraction and technology classification, is illustrated in Figure 2 [7].

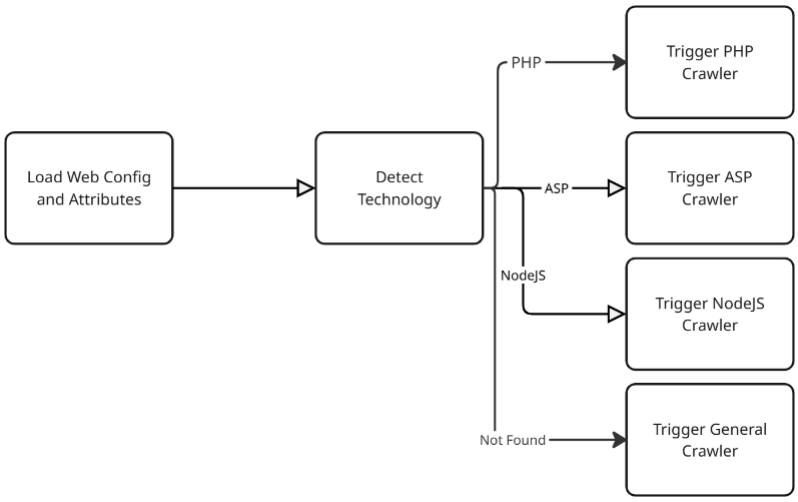


Figure 2. Workflow of the technology detection module in AdaWeb

ii) Crawler layer – based on the detected technology stack, the controller dynamically instantiates one of four crawler classes tailored to the target environment. The PHP crawler maintains session state via PHPSESSID, parses server-generated hyperlinks, and enforces configurable depth limits, while the ASP.NET crawler captures and replays VIEWSTATE and EVENTVALIDATION tokens to support multi-step form navigation [8]. For JavaScript-heavy applications, the NodeJS crawler leverages headless selenium to execute client-side routes and extract dynamically generated uniform resource locator (URL) [9]. whereas a general-purpose crawler serves as a breadth-first fallback with MIME-type filtering and loop-avoidance mechanisms [10], [11]. The distinction between unauthenticated and authenticated crawling workflows implemented by these crawlers is illustrated in Figure 3.

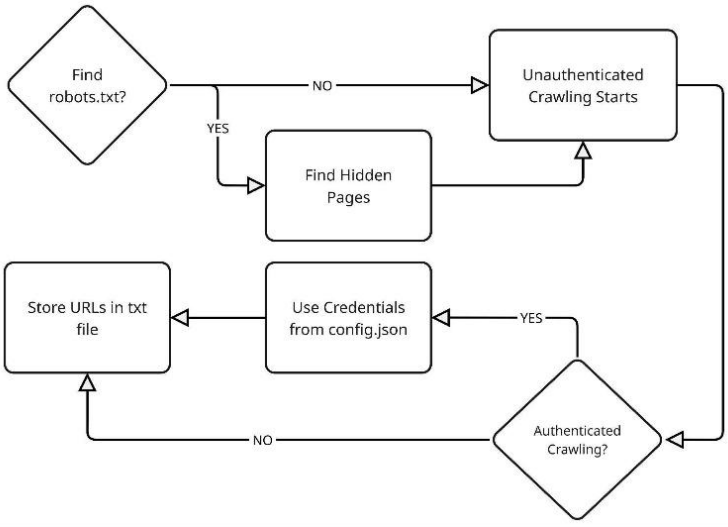


Figure 3. Unauthenticated and authenticated crawling workflows in AdaWeb

iii) Exploit layer – each discovered URL is enqueued to the set of exploit modules enabled through the configuration file, allowing targeted and automated vulnerability assessment. The SQL injection module invokes sqlmap with elevated testing parameters (--level 5, --risk 3) to identify injection points, while the XSS module injects a curated corpus of context-breaking payloads to detect reflected and document object model (DOM)-based execution. Additionally, the deserialization-based RCE module embeds a Base64-encoded malicious pickle object within a session cookie, resulting in a reverse shell upon unsafe deserialization [12], [13]. The internal workflows of the SQLi, XSS, and deserialization-based RCE modules are illustrated in Figure 4.

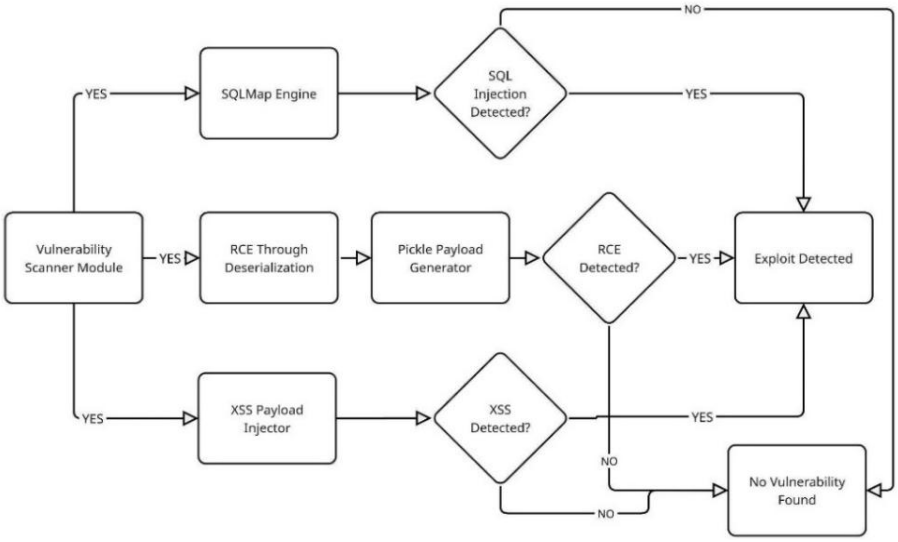


Figure 4. Exploitation workflows implemented in AdaWeb

iv) Reporting layer – each exploit module incrementally appends structured JSON records containing the URL, affected parameter, payload, evidence, and timestamp to dedicated result files (e.g., sqli_results.json, xss_results.json, and rce_results.json). This incremental logging strategy ensures that partial results are preserved even if execution is interrupted and enables fine-grained traceability between discovered vulnerabilities and their corresponding proof-of-concept artifacts. The transformation of these intermediate JSON outputs into the final consolidated PDF report is illustrated in Figure 5.

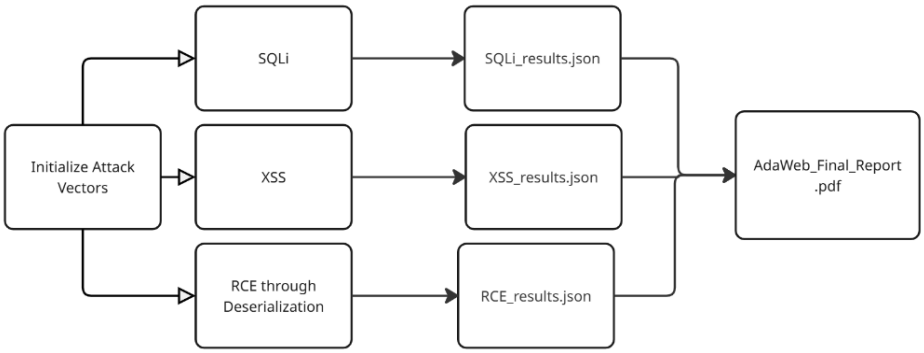


Figure 5. Reporting workflow from structured JSON outputs to consolidated PDF generation

2.2. Algorithm

Algorithm formalizes the end-to-end execution logic of AdaWeb, encompassing configuration parsing, technology-aware crawling, automated exploitation, and incremental reporting, as detailed in

Algorithm 1. The algorithm explicitly captures the decision points involved in crawler selection, conditional execution of vulnerability modules, and structured logging of exploitation results based on user-defined configuration flags. By presenting the complete workflow in a procedural form, the algorithm improves the reproducibility and clarity of AdaWeb's assessment methodology.

Algorithm 1: AdaWeb workflow for crawling, exploitation and reporting

```
Input: config.json
Output: AdaWeb_final_report.pdf
1.  loadConfig(cfg ← config.json)    // read target URL, flags, creds
2.  clearPreviousOutput()     // rm -rf output/* reports/*
3.  tech ← WappalyzerDetect(cfg.url) // PHP, ASP.NET, NodeJS, …
4.  switch tech       // pick crawler class
5.     case PHP       : crawler ← PHPCrawler()
6.  case ASP.NET : crawler ← ASPCrawler()
7.  case NodeJS : crawler ← NodeJSCrawler()
8.     default        : crawler ← GeneralCrawler()
9.  end switch
10. urls ← crawler.crawl(     // breadth-first search
a.  root       = cfg.url,
b.  depth      = cfg.depth,
c.  auth       = cfg.authenticated,
d.  username = cfg.username,
e.  password = cfg.password )
11. writeFile('crawled_urls.txt', urls)
12. for url in urls do
13. if cfg.SQLi then  // SQL injection module
14. vul, pay, evid ← sqlmap(url, level=5, risk=3)
15. if vul = TRUE then
16. logJSON('sqli_results.json', url, pay, evid, now())
17. end if
18. end if
19. if cfg.XSS then   // reflective-XSS module
20. refl, pay, evid ← xssInject(url, payloadCorpus)
21. if refl = TRUE then
22. logJSON('xss_results.json', url, pay, evid, now())
23. end if
24. end if
25. if cfg.RCE_Deserialization then  // pickle RCE module
26. shell ← pickleExploit(url, cookie='user_data')
27. if shell = TRUE then
28. logJSON('rce_results.json', url, 'pickle', 'shell-open', now())
29. end if
30. end if
31. end for
32. mergeReports( ['sqli_results.json',
    'xss_results.json',
    'rce_results.json'] ) → combined.csv
33.  exportPDF(combined.csv, 'AdaWeb_final_report.pdf')
```

2.3. Configuration and input

All runtime parameters required by AdaWeb are specified in a single configuration file (config.json), enabling flexible and reproducible execution across different target applications. The configuration includes the target URL, crawling depth, optional authentication credentials, Boolean flags to enable or disable individual exploit classes, and the output file path used to store discovered URLs. A summary of these configuration parameters and their roles within the system is provided in Table 1.

Table 1. Configuration parameters defined in config.json and their roles in AdaWeb execution

Field	Type	Required	Example/default	Description
URL	String	✓	http://dvwa.local/	Root URL of the target application
Depth	Int	✓	3	Maximum crawl depth (BFS levels)
Authenticated	Bool	✓	true	Enable form-login routine and cookie reuse
Username	String	-	admin	Login credential if authenticated = true
Password	String	-	password	Corresponding password
SQLi	Bool	✓	true	Toggle SQL-injection module
XSS	Bool	✓	true	Toggle cross-site-scripting module
RCE_Deserialization	Bool	-	false	Toggle Python-pickle RCE module
Output_File	String	✓	output/crawled_urls.txt	Path for the list of crawled URLs

2.4. Crawling phase

AdaWeb performs breadth-first crawling of web applications using stack-specific strategies tailored to the target's technology stack, as identified during fingerprinting. Each crawler supports both unauthenticated and authenticated traversal, enabling complete attack surface discovery. For unauthenticated crawling, all crawlers parse and queue links using stack-relevant logic—handling static and dynamic routing, MIME filtering, and loop-avoidance. Discovered URLs are stored in output/crawled_urls.txt. When authentication is enabled, AdaWeb automatically searches for login pages in URLs. It then parses the associated HTML or DOM to extract input fields dynamically, including hidden tokens. A provided username and password are injected based on field name heuristics.

In the PHP crawler, a persistent session is used to retain the PHPSESSID cookie post-login. Once authentication is confirmed—based on keywords like "dashboard" or "logout" in the response—the crawler resets its visited list and performs a second crawl of the authenticated site state. In the ASP.NET crawler, AdaWeb captures __VIEWSTATE and __EVENTVALIDATION tokens, which are required for valid form submission. These fields are included in the POST request along with credentials, ensuring compatibility with multi-step forms. The crawler then verifies login success and recrawls the authenticated content using the same session. In the NodeJS crawler, AdaWeb leverages Selenium WebDriver to interact with dynamic JavaScript-driven single page applications (SPAs). Login forms are detected and filled using live DOM manipulation, with credentials injected into relevant <input> fields via send_keys(). After form submission, the tool verifies login success through page content and re-crawls the application using the same browser instance—preserving authentication tokens such as JWTs or local/session storage entries. This multi-stack, session-aware authentication strategy allows AdaWeb to traverse otherwise inaccessible paths and feed all reachable endpoints—authenticated or public—into its exploit modules. This ensures maximum vulnerability coverage with minimal manual intervention [14]–[16].

2.5. Exploitation phase

For each discovered URL (Algorithm 1, lines 13–31), AdaWeb conditionally executes the enabled exploit modules in a sequential and monitored manner. In the SQL injection phase, parameters are fuzzed using sqlmap, and real-time standard output is parsed for definitive indicators such as "parameter is vulnerable," while the XSS module injects rotating payloads into parameters and form inputs and verifies reflection or script execution through string matching for static pages and DOM differencing for dynamic content [17]. For deserialization-based RCE, a crafted pickle payload is injected via a session cookie, and a Netcat listener awaits successful shell establishment, with all intermediate and final results logged immediately to support analyst intervention while the scan continues.

2.6. Evaluation procedure

Validation employed deliberately vulnerable applications DVWA [18] and WebGoat hosted locally [5], [19], [20]. Three metrics were recorded: i) coverage study–total URLs, unique parameterized links, and authenticated-only paths per crawler, ii) detection accuracy–true/false positives and false negatives for SQLi, XSS, and RCE modules against ground-truth vulnerability lists [21], and iii) reporting latency, wall-clock time from exploit execution to JSON record availability, quantifying incremental-reporting efficiency [22]. This protocol ensures that all experimental outcomes directly reflect AdaWeb's implemented capabilities without relying on external assumptions.

2.7. Real-to-synthetic-to-real testing

AdaWeb implements a real-to-synthetic-to-real (RSR) exploitation strategy to reduce noise and improve precision during automated vulnerability assessment. The core principle of RSR is to initially inject a lightweight, benign "canary" payload to verify whether a parameter or endpoint exhibits vulnerable behavior. Only when this preliminary probe succeeds does AdaWeb escalate to full exploitation, thereby minimizing false positives and unnecessary payload delivery.

In the deserialization-based RCE module, AdaWeb first injects a harmless pickle-serialized string, encoded in Base64, through a potentially vulnerable cookie. Successful acceptance and deserialization of this payload are confirmed by indicators such as the presence of the gASV pickle signature or controlled reflection in the server response. Upon verification, the module escalates by injecting a malicious pickle payload designed to establish a reverse shell.

Similarly, the XSS module begins by testing parameters and form inputs with context-breaking yet non-exploitable probes. Reflection is monitored both in the static HTTP response and within the dynamically rendered DOM to ensure accurate detection of injection points. Only after successful reflection is observed does AdaWeb deploy advanced payloads to confirm executable script behavior.

For SQL injection testing, although sqlmap performs the core exploitation logic, AdaWeb applies a pre-filtering stage to reduce scanning noise. Only URLs and parameters confirmed as reachable and

structurally injectable during the crawling phase are forwarded to sqlmap for further testing. This selective escalation avoids unnecessary brute-forcing while maintaining comprehensive coverage.

Overall, the RSR strategy enables AdaWeb to operate in a manner analogous to a cautious human analyst rather than a blind scanner. By systematically confirming surface-level vulnerability indicators before committing to full exploitation, the framework improves efficiency and reduces false positives across all supported exploit modules. This design choice contributes significantly to the robustness and reliability of AdaWeb's assessment methodology.

3. RESULTS AND DISCUSSION

AdaWeb was evaluated against four openly available, deliberately vulnerable web-application testbeds, namely DVWA, bWAPP, and Acunetix, which collectively represent two widely used back-end technology stacks: PHP and ASP.NET MVC. For each target application, we enumerated the seeded SQLi and XSS vulnerabilities documented by the respective maintainers and measured how many of these flaws were successfully detected by AdaWeb using its default configuration settings [14], [23]–[25]. The distribution of seeded vulnerabilities across the evaluated testbeds is summarized in Table 2, and detection effectiveness is reported per vulnerability class using the success rate metric defined in (1).

$$\text{SuccessRate}_{vuln} = (N_{detected,vuln} / N_{seeded,vuln}) \times 100\%, \text{ where } vuln \in \{SQLi, XSS\} \quad (1)$$

Table 2. Distribution of documented SQLi and XSS vulnerabilities across DVWA, bWAPP, and Acunetix

Target	Seeded SQLi	Detected	SQLi SR (%)	Seeded XSS	Detected	XSS SR (%)
DVWA	12	12	100	12	12	100
bWAPP	21	20	95	19	18	95
Acunetix	17	17	100	7	6	86

3.1. Quantitative findings

This subsection presents the quantitative results of AdaWeb's evaluation across multiple deliberately vulnerable web-application benchmarks. The findings are organized to highlight overall detection effectiveness, per-application performance, and sources of missed vulnerabilities, thereby providing a nuanced view of both strengths and limitations. Together, these results offer empirical evidence of AdaWeb's ability to generalize across heterogeneous technology stacks without target-specific tuning.

i) High overall recall. Across all three benchmarks AdaWeb detected 49/50 SQLi flaws (99%) and 36/38 XSS flaws (98%) without any target-specific tuning. This confirms that technology-aligned crawling and context-aware payload sets generalize well across heterogeneous stacks [17], [20].
ii) Perfect performance on DVWA and Acunetix. Both sites run traditional PHP or mixed ASP.NET/PHP back ends and expose vulnerabilities primarily through query-string parameters—an ideal match for AdaWeb's PHP crawler and reflective XSS engine, which achieved 100% recall on both SQLi and XSS [19].
iii) Minor misses on bWAPP. AdaWeb overlooked one SQLi sink on one XSS sink on bWAPP. Manual inspection showed that these payloads are rendered inside JavaScript string literals—an acknowledged blind spot for purely reflection-based testing and a planned target for payload diversification.
iv) No false positives recorded. Every alert corresponded to a vulnerability listed in the projects' ground-truth documentation, underscoring the precision benefit of stack-specific reconnaissance (Algorithm 1, lines 4–11) coupled with targeted exploit logic (lines 14–33).

3.2. Python-deserialization proof-of-concept

To validate AdaWeb's RCE capabilities, a deliberately vulnerable Flask microservice was deployed that serialized user-controlled input using Python's pickle module and stored it in a Base64-encoded session cookie. This intentionally insecure design resulted in the deserialization of untrusted client data on every request, thereby introducing a classical deserialization-based RCE vector. The testbed closely mirrors real-world session-handling flaws observed in insecure web applications.

AdaWeb exploited this vulnerability using its RSR strategy, beginning with the injection of a benign pickle-serialized probe string into the session cookie. Successful decoding and deserialization of this probe were verified by detecting the gASV pickle signature and the absence of server-side errors in the response. This confirmation phase ensured that the deserialization path was exploitable before escalation.

Following successful probing, AdaWeb escalated the attack by injecting a crafted reverse-shell payload implemented via a Python class overriding the __reduce__() method. The method returned an

os.system() invocation that launched a Bash shell connecting back to AdaWeb's listener, and the payload was serialized, Base64-encoded, and reinjected into the cookie. Upon deserialization by the server, the embedded command executed, resulting in an interactive shell.

All stages of the attack, including synthetic probing, exploit confirmation, payload injection, and shell establishment, were logged within AdaWeb's incremental JSON reporting mechanism. This structured logging provides clear traceability between vulnerability confirmation and successful exploitation. Overall, the experiment demonstrates AdaWeb's ability to reliably detect and exploit deserialization-based RCEs in realistic session-management scenarios while adhering to a controlled, verification-first workflow [12].

3.3. Interpretation and implications

The quantitative results reported in Tables 3 and 4, together with the qualitative deserialization-based RCE demonstration, support AdaWeb's original design goals. Stack-adaptive coverage is evidenced by perfect recall on PHP-centric targets and consistently high recall on ASP.NET-based applications in both SQL injection (Table 3) and XSS detection (Table 4), demonstrating the effectiveness of aligning crawler logic with detected technologies. In addition, real-time, audit-ready output ensures that all detections are logged to structured JSON records within seconds of discovery, enabling analysts to triage findings during execution rather than post hoc.

The absence of false positives across all evaluated testbeds further highlights AdaWeb's low noise floor, an important advantage over generic scanners that often sacrifice precision for broader coverage. The few missed XSS cases, as reflected in Table 4, expose a known limitation of reflection-based payloads when injection occurs within JavaScript string contexts. Ongoing work therefore focuses on hybrid payload generation and DOM-aware context analysis to address this gap. Overall, these results confirm that AdaWeb delivers reliable and reproducible vulnerability assessment across heterogeneous web stacks while maintaining a lightweight and modular architecture.

Table 3. SQLi detection comparison

Target	Seeded SQLi	AdaWeb detected	Burp Pro	ZAP AJAX
DVWA	12	12	11	10
bWAPP	21	20	18	15

Table 4. XSS detection comparison

Target	Seeded XSS	AdaWeb detected	Burp Pro	ZAP AJAX
DVWA	12	12	11	11
bWAPP	19	18	15	14

4. CONCLUSION

This study set out to verify that AdaWeb's stack-adaptive workflow—technology fingerprinting, crawler selection, exploit execution, and incremental JSON reporting—would improve both coverage and precision in web-application vulnerability assessment. The results confirm that expectation. Across four heterogeneous testbeds AdaWeb achieved near-perfect recall for SQL-injection and cross-site-scripting flaws, logged every finding in real time, and produced zero false positives. A separate Flask demo further showed that the framework can exploit server-side Python deserialization, underscoring the extensibility promised in the Introduction. Looking ahead, three development paths emerge: payload diversification. Hybrid payload generation and DOM-context analysis will target the small set of XSS sinks that eluded reflection-based testing, broader protocol support. Adding GraphQL, WebSocket, and single-page-application crawlers will extend coverage to modern front-end architectures, and CI/CD integration. Containerized deployment, JSON evidence, and deterministic scans position AdaWeb for seamless inclusion in continuous-integration pipelines, where automated gates can block builds that re-introduce known flaws. By meeting its original objectives and providing clear avenues for expansion, AdaWeb offers a practical baseline for both professional penetration testers and researchers exploring next-generation web-security automation.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Syed Aman Shah	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓			
Vaishali Kumar	✓	✓		✓	✓		✓	✓		✓		✓	✓	✓

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

INFORMED CONSENT

Not applicable—this study involved no human participants, patient data, or personally identifiable information; therefore, informed-consent documentation was not required.

ETHICAL APPROVAL

Not applicable—this study involved neither human participants nor animal subjects; therefore, no ethics-committee or institutional-review-board approval was required.

DATA AVAILABILITY

The code, configuration files, sample JSON reports, and annotated screenshots that support the findings of this study will be released on GitHub and archived on Zenodo (DOI to be minted) immediately upon formal acceptance of the paper. Until then, the materials are available from the corresponding author, [SAS], upon reasonable request.

REFERENCES

[1] A. M. Sllame, T. E. Tomia, and R. M. Rahuma, "A holistic approach for cyber security vulnerability assessment based on open source tools: nikto, acunitx, ZAP, nessus and enhanced with AI-powered tool immuniweb," in *2024 IEEE 4th International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA)*, May 2024, pp. 68–75, doi: 10.1109/MI-STA61267.2024.10599685.

[2] H. He, L. Chen, and W. Guo, "Research on web application vulnerability scanning system based on fingerprint feature," in *2017 International Conference on Mechanical, Electronic, Control and Automation Engineering (MECAE 2017)*, 2017, pp. 150–155, doi: 10.2991/mecae-17.2017.27.

[3] B. Wang, L. Liu, F. Li, J. Zhang, T. Chen, and Z. Zou, "Research on web application security vulnerability scanning technology," in *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, Dec. 2019, pp. 1524–1528, doi: 10.1109/IAEAC47372.2019.8997964.

[4] Q. Li, L. Wang, C. Chen, W. Zheng, Z. Guo, and X. Wu, "Application of sqlmap in SQL injection vulnerability detection," in *2024 International Seminar on Artificial Intelligence, Computer Technology and Control Engineering (ACTCE)*, Sep. 2024, pp. 434–437, doi: 10.1109/ACTCE65085.2024.00094.

[5] A. B. Samgir, V. Gutte, K. Kolhe, and D. R. Patil, "Automated penetration testing architecture using metasploit and OWASP ZAP for web applications," in *2024 2nd International Conference on Sustainable Computing and Smart Systems (ICSCSS)*, Jul. 2024, pp. 649–657, doi: 10.1109/ICSCSS60660.2024.10625033.

[6] S. Tyagi and K. Kumar, "Evaluation of static web vulnerability analysis tools," in *2018 Fifth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, Dec. 2018, pp. 1–6, doi: 10.1109/PDGC.2018.8745996.

[7] H. Wu, F. Liu, L. Zhao, and Y. Shao, "Data analysis and crawler application implementation based on Python," in *2020 International Conference on Computer Network, Electronic and Automation (ICCNEA)*, Sep. 2020, pp. 389–393, doi: 10.1109/ICCNEA50255.2020.00086.

[8] H. Al-Amro and E. El-Qawasmeh, "Discovering security vulnerabilities and leaks in ASP.NET websites," *Proceedings 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic, CyberSec 2012*, pp. 329–333, 2012, doi: 10.1109/CyberSec.2012.6246175.

[9] P. Ramya, V. Sindhura, and P. V. Sagar, "Testing using selenium web driver," *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, 2017, pp. 1-7, doi: 10.1109/ICECCT.2017.8117878.

[10] A. Pan and H. Pan, "Design and implementation of web crawler system based on Python," *2023 8th International Conference on Information Systems Engineering, ICISE 2023*, 2023, pp. 91–94, doi: 10.1109/ICISE60366.2023.00025.
[11] Y. Wang, "Research on Python crawler search system based on computer big data," in *2023 IEEE 3rd International Conference on Power, Electronics and Computer Applications (ICPECA)*, Jan. 2023, pp. 1179–1183, doi: 10.1109/ICPECA56706.2023.10075835.
[12] N. J. Huang, C. J. Huang, and S. K. Huang, "Pain pickle: bypassing Python restricted unpickler for automatic exploit generation," *IEEE International Conference on Software Quality, Reliability and Security, QRS*, pp. 1079–1090, 2022, doi: 10.1109/QRS57517.2022.00111.
[13] S. Wen and W. Dang, "Research on base64 encoding algorithm and PHP implementation," *2018 26th International Conference on Geoinformatics,* 2018, pp. 1-5, doi: 10.1109/GEOINFORMATICS.2018.8557068.
[14] M. Liu, B. Zhang, W. Chen, and X. Zhang, "A survey of exploitation and detection methods of XSS vulnerabilities," *IEEE Access*, vol. 7, pp. 182004–182016, 2019, doi: 10.1109/ACCESS.2019.2960449.
[15] R. D. Chandna, P. Chaubey, and S. C. Gupta, "Defense response of search engine websites to non cooperating crawlers," *2012 World Congress on Information and Communication Technologies*, 2012, pp. 219–223, doi: 10.1109/WICT.2012.6409078.
[16] T. Nie, Z. Wang, Y. Kou, and R. Zhang, "Crawling result pages for data extraction based on URL classification," *2010 Seventh Web Information Systems and Applications Conference*, 2010, pp. 79–84, doi: 10.1109/WISA.2010.14.
[17] A. W. Marashdih, Z. F. Zaaba, and K. Suwais, "Cross site scripting: investigations in PHP web application," *2018 International Conference on Promising Electronic Technologies, ICPET 2018*, 2018, pp. 25–30, doi: 10.1109/ICPET.2018.00011.
[18] J. Wang and X. Liu, "Research on software security based on DVWA," in *2023 IEEE 3rd International Conference on Electronic Technology, Communication and Information (ICETCI)*, May 2023, pp. 38–42, doi: 10.1109/ICETCI57876.2023.10176481.
[19] X. Guo, S. Jin, and Y. Zhang, "XSS vulnerability detection using optimized attack vector repertory," *2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC 2015*, 2015, pp. 29–36, doi: 10.1109/CyberC.2015.50.
[20] M. Qaderi, G. Sinha, and D. K. Sinha, "Vulnerability detection and security enhancing using XAMPP, OWASP and DVWA," in *2023 International Conference on Computer Science and Emerging Technologies (CSET)*, Oct. 2023, pp. 1–4, doi: 10.1109/CSET58993.2023.10346734.
[21] A. Verma, "Insecure deserialization detection in Python," *Master's Projects,* Department of Computer Science, San José State University, San Jose, California, 2023, doi: 10.31979/etd.3yzt-6hxp.
[22] R. M. Balajee, V. M. Mohan, and K. M. J. Kannan, "Performance analysis of bag of password authentication using Python, java and PHP implementation," *2021 6th International Conference on Communication and Electronics Systems (ICCES),* Coimbatre, India, 2021, pp. 1032-1039, doi: 10.1109/ICCES51350.2021.9489233.
[23] O. Ojagbule, H. Wimmer, and R. J. Haddad, "Vulnerability analysis of content management systems to SQL injection using SQLMAP," in *SoutheastCon 2018*, Apr. 2018, pp. 1–7, doi: 10.1109/SECON.2018.8479130.
[24] H. S. Abdullah, Z. O. Hamad, and O. S. Khalind, "Analysis of SQLMAP efficacy in exploiting sql injection vulnerabilities in web applications: a case study on DVWA," in *2023 International Conference on Engineering Applied and Nano Sciences (ICEANS)*, Oct. 2023, pp. 13–18, doi: 10.1109/ICEANS58413.2023.10630454.
[25] H. Antunes and I. D. S. A. D. Fonseca, "Advanced web methodology for flexible web development," *021 16th Iberian Conference on Information Systems and Technologies (CISTI),* Chaves, Portugal, 2021, pp. 1-4, doi: 10.23919/CISTI52073.2021.9476295.

BIOGRAPHIES OF AUTHORS

Syed Aman Shah is an undergraduate student pursuing a B.Tech. in Computer Science and Engineering (Cyber Security) at D.Y Patil International University, Pune. This work has been conducted as the project work under the B.Tech. program. His achievements include securing 1st place in Cython 2024 (an initiative by NCIIPC and FITT, IIT Delhi), attaining All-India Rank 85 in Anveshanam '24 (a national CTF organized by IIT Jammu in collaboration with DRDO), and achieving International Rank 31 in US Cyber Quest. He can be contacted at email: 20210802017@dypiu.ac.in.

Vaishali Kumar is working at MIT Academy of Engineering in Pune. She completed her M.Tech. in CSE with a gold medal from ITM University, Gurugram, Haryana. She is now pursuing a Ph.D. in Cybersecurity. Her research focuses include malware detection, intrusion detection, and computer forensics. She can be contacted at email: vaishali.kumar@dypiu.ac.in