National College of Ireland

A Comparative Analysis of OWASP ZAP, Arachni and Wapiti against the Reinforced WAVSEP Benchmark

MSc Research Project
MSc. In Cybersecurity

Chetanya Sharma
Student ID: x23422793

School of Computing
National College of Ireland

Supervisor: Dr. Arghir-Nicolae Moldovan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Chetanya Sharma ……
Student ID:	x23422793 ……
Programme:	MSc. In Cybersecurity …… **Year:** 2025 ……
Module:	MSc (Research) Practicum/Internship Part 2 ……
Supervisor:	Dr. Arghir-Nicolae Moldovan ……
Submission Due Date:	11th December,2025 ……
Project Title:	A Comparative Analysis of OWASP ZAP, Arachni and Wapiti Against the Reinforced WAVSEP Benchmark ……
Word Count:	7009 …… **Page Count** 20 ……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: 

……

Date: 9th December, 2025 ……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

A Comparative Analysis of OWASP ZAP, Arachni and Wapiti Against the Reinforced WAVSEP benchmark

Chetanya Sharma
x23422793

Abstract

The increasing complexity of the modern web applications has increased the need for an effective automated vulnerability assessment technique. The study evaluates the practical-effectiveness of three widely used Dynamic Application Security Testing (DAST) tools- OWASP ZAP, Arachni and Wapiti- using the Reinforced WAVSEP benchmark under realistic, full black-box conditions. The benchmark was deployed in a controlled virtualized environment, and every scanner was executed using a default-configurations to reflect common practitioner usage. A custom evaluation framework was developed to map scanner outputs to the ground truth cases and compute the detection accuracy.

The results highlighted substantial variations in tool performance. Arachni achieved highest overall detection capability (F1= 0.88), followed by the Wapiti (F1=0.68 while ZAP recorded the lowest performance of F1 =0.55, previously published Reinforced WAVSEP evaluations, all the tools demonstrated notably lower performance in this study also highlighting the gap between benchmark-optimised configurations and real-world scanning behaviour. These findings throw light on the need of improved scanner tuning, hybrid testing strategies and the enhanced benchmark realism in order to support the reliable we application security assessment.

1 INTRODUCTION

Web applications cover a significant portion of present world's digital infrastructure by delivering an essential-services across the sectors like finance, healthcare and the e-commerce. With the time, as they grow in complexity, their exposure to the security vulnerabilities also grows which makes them more vulnerable to the attackers. According to the recent studies, they highlight the long-standing weaknesses like SQL Injection (SQLi), Cross-Site Scripting (XSS), Path Traversal and Command Injection that continue to the most exploited categories of the web vulnerabilities in the real-world attacks**(Albahar et al., 2022) (Wicaksana & Fauzan, 2025)**. This ensures that these vulnerabilities are found early and accurately and is therefore an important for maintaining the security and reliability of modern web systems. Dynamic Application Security Testing (DAST) tools play a central and important role in finding these vulnerabilities by interacting with the running application from an external, attacker like perspective. Unlike the Static Application Security Testing (SAST), which needs the access to the source code, the DAST tools are completely black-box and also applicable to the third-party, legacy and the closed-source systems **(Koman, 2025)**. However, the previous work shows the considerable variation in the accuracy, false positive-rates and coverage of variety of different DAST tools, with the performance influenced by the design of payload, crawling, depth framework compatibility**(Albahar et al., 2022) (Chorell & Ekberg, 2024)**. This performance contradictions highlights the importance of independent, empirical benchmarking using the standardized test environments like the Reinforced WAVSEP.

- Research Problem and Motivation:

Though the DAST tools are majorly used in practice, there are some lacking like the updating, reproducibility and the independent comparative evaluations, specially like using the

modernized benchmarks like Reinforced WAVSEP. Researchers and the security engineers require the actual evidence in order to access how good these tools detect the real vulnerabilities and also to understand their limitations in a different vulnerability class. This vent then motivates the present study.

- **Research Questions:**

This project leads to answer the following questions:

1. How effectively do OWASP ZAP, Arachni and the Wapiti detect the vulnerabilities when evaluated against the Reinforced WAVSEP benchmark?
2. What are the detection patterns, strength and the limitations of each tool across diverse vulnerability categories?
3. Can an automated evaluation framework be developed to reliably benchmark DAST tools using standardized ground-truth data?

- **Research Objectives:**

In order to address these queries, the project highlights the following objectives:

1. Deployment of Reinforced-WAVSEP benchmark in a reproducible Docker-based environment.
2. Execution of OWASP-ZAP, Arachni and Wapiti (DAST) tools in a black-box mode across the full benchmark test suite.
3. Develop the Python script to parse the XML/JSON outputs, then further map identified vulnerabilities to WAVSEP case identifiers and compute performance metrics (TP, FP, Precision, Recall, F1-score).
4. Comparison of detection capabilities across the tools and vulnerability categories.
5. Identify the practical strengths, weaknesses and the implications for real world use.

- **Hypotheses**:

Analysing from the existing literature, this study highlights the following hypotheses:

1. The tools will be demonstrating the higher accuracy on the injection-based vulnerabilities like XSS, SQLi and CMD whereas on traversal based like Path traversal and file inclusion (Albahar et al., 2022) (Koman, 2025).
2. The OWASP ZAP will show high precision and lower recall which is consistent with the previous evaluations showing the conservative attack payload strategy (Albahar et al., 2022).
3. In the categories which require deeper payload testing, Arachni will outperform Wapiti because of its modular and extensible scanning framework (Chorell & Ekberg, 2024).

2 RELATED WORK

Evaluating the effectiveness of the Dynamic Application Security Testing (DAST) tools needs an understanding of the both methodologies used in prior comparative studies and the broader view of landscape of the web application security testing. This section reviews the existing literature across the two main domains: (1) the security testing techniques with focus on DAST, and (2) the benchmark driven evaluations of the vulnerability scanners that includes WAVSEP and its successors. The review combines the quantitative findings from the recent studies which contextualise the strengths and the limitations of the current approaches.

2.1 Web Application Security Testing and DAST

The web application security testing has been categorized in to four different techniques namely, Static Application Security Testing (SAST), Dynamic Application Security (DAST), Interactive Application Security Testing (IAST) and hybrid approaches that combines the multiple methods for evaluation. The systematic reviews focus that while the SAST identifies the code-level defects, DAST is important for discovering the runtime vulnerabilities like SQL

Injection, XSS, Path Traversal, the session management flaws and authentication bypasses. These issues majorly cannot be detected purely through the code analyses(Wicaksana & Fauzan, 2025) (Mnasri et al., 2023)

DAST tools functions in a black-box manner, which means they interact with the live application (in runtime) by crawling the reachable endpoints and injecting the designed payloads. There are many studies that highlights the dependency of DAST effectiveness on the crawling depth, design of payload and the request orchestration. Wicaksana and Fauzan's systematic review discovered that across 67 actual studies, DAST tools achieved for injection vulnerabilities than for the logic or the traversal vulnerabilities (Mnasri et al., 2023). In the same way, Mnasri et al. points that DAST tools commonly shows the false positive rates between 8% to 35% which is influenced by the server response variations and the insufficient response parsing (Hanssen, 2023)

The Comparative studies between the SAST and the DAST supports this pattern. Dencheva's evaluation that used the OWASP benchmark exhibits that while the SAST tools were able to achieve higher recall for the code-level vulnerabilities with an average of 74%, DAST outperformed SAST on the runtime vulnerabilities where the dynamic payload execution is essential by achieving 80-92% for the SQLi and the XSS across multiple scanners (Dencheva, 2023). Qadir et al. assures this complementary nature that DAST excelled in detecting the SQLi and XSS while the SAST is better in detecting the input validation and the business logic issues (Qadir, 2023)

But again, the limitation still exists. Chorell and Ekberg were able to found that ZAP, which is one of the mostly used DAST tools were able to reach high precision which was greater than 90% but also faced difficulty in traversal vulnerabilities by missing up to 65%-70% of path traversal cases on WAVSEP (Chorell & Ekberg, 2024). The study also observed that Arachni showed 88-94% recall on the SQLi and XSS controlled experiments, but had a longer scan time. The recent studies with WAPITI similarly predict moderate recall which is 50-60% and a tendency to miss the complex multi stage payloads (Rahman, 2024).

Across the literature, key limitations cited includes:

- Limited evasion capabilities against the modern WAFs.
- Difficulty in handling JavaScript- heavy applications.
- Shallow crawling in deep, stateful workflows
- Variability in scan depth across the tools.

The above-mentioned flaws highlight the ongoing need for the actual benchmarking using the controlled, labelled datasets which provide actionable performance insights beyond simple vulnerability counts.

2.2 Benchmarking and Comparative Evaluations of DAST Tools

The benchmarking frameworks like OWASP benchmark and WAVSEP and other custom vulnerable applications are central to evaluating the vulnerability scanners. The main WAVSEP benchmark provides the test cases for the SQLi, XSS, Path traversal, RFI/LFI and other categories which enables the systematic measurement of the tool accuracy. Urbano et al. introduced the Reinforced WAVSEP, that extended the original dataset significantly with the harder testcases that improved the categorization and a formalized ground truth results file. The experiments done by them highlighted that several commercial and the open-sourced

scanners showed decreased recall when tested against the Reinforced WAVSEP, with some of the categories that dropped by 2-40 percentage points as compared to the results reported in the older versions of WAVSEP (Urbano, 2023)

Many studies have used the WAVSEP in order to compare the tools. Albahar et al. evaluated different scanners and found that ZAP achieved a recall of 62% on XSS and 58% on SQLi, while the commercial scanners reached 78%-84% recall on the same cases (Albahar, Alansari and Jurcut, 2022). A different study evaluates NodeGoat and DVNA applications found ZAP and burp-suite performing good on simple injection vulnerabilities but detecting less than 40% of path traversal issues (Azwar, 2025). Wapiti, which is lightweight, was able to detect only 56% of SQLi and 49% of XSS cases in same study (Azwar, 2025).

SCAnME, which is a 2025 benchmarking methodology and is proposed by Koman et al., introduced a unified comparison framework for the scanner evaluation, measuring the accuracy, coverage, and performance across multiple applications (Koman, 2025). The findings indicate that Arachni achieved superior overall recall around 78% as compared to several competitors but also required significantly more processing time. But, their evaluations did not include Reinforced-WAVSEP and focused more on the scanner architecture and metrics designs.

Other research includes mixed-target evaluations using vulnerable applications such as DVWA, WebGoat OWASP Juice Shop and the custom-built Node.jd applications. Chahal et al. compared several open-source scanners and observed that the ZAP was able to detect 72% of XSS cases, Arachni detected 81%, and Wapiti detected 55%, though the study lacked a fully labelled ground truth and relied on manual verification (Chahal, 2022). Dibbets worked on improving the Burp Suite's scanning by using the BChecks and was able to show that commercial tools can achieve up to 90% recall on injection vulnerabilities when they are enhanced with the custom payloads which suggests that the diversity of payload is important factor in analysing the scanner performance (Dibbets, 2023)

Even after the extensive prior work, the limitations remain across existing studies:

- There was use of partial subsets of WAVSEP and many of them used outdated benchmarks.
- Most of them did not evaluate all the three widely used DAST open-source tools like ZAP, Arachni and wapiti together.
- Few of them were able to provide fully automated evaluation pipeline that mapped raw scanned results to per-test-case ground test.
- Lastly, most of them lacked transparency in the methodology and did not release the code for the reproducibility.

STUDY	TOOLS EVALUATED	BENCHMARK/TARGETS	KEY FINDINGS	LIMITATIONS
Urbano et al. (2022)	Multiple OSS and Commercial	Reinforced WAVSEP	Detection recall dropped by 20-40% vs old WAVSEP and many tools weak in traversal	Pipeline not reusable and Wapiti not included
Albahar et al. (2022)	ZAP, Arachni, commercial tools	WAVSEP	ZAP: 62% XSS, 58% SQLi; Arachni: ~90% SQLi, 89% XSS	No Dockerisation/ automation and partial dataset was used

Azwar et al. (2023/25)	ZAP, Wapiti, Arachni, Burp	DVNA, NodeGoat	ZAP/Burp strong in SQLi; Path Traversal < **40%**; Wapiti ~ **50–60%** overall	It is not benchmark-based; also lacks full ground truth
Chorell and Ekberg (2024)	ZAP, Arachni and others	WAVSEP, WebGoat	ZAP precision > **90%**; Arachni recall **88–94%**	There are mixed targets and no unified scoring
Koman et al. (2025)	Arachni and others	Curated testbeds	Overall Arachni recall ~78%	Did not use Reinforced WAVSEP
Chahal et al. (2022)	ZAP, Arachni, Wapiti	Custom Apps	ZAP 72% XSS, Arachni 81%, Wapiti 55%	No Benchmark and Manual Verification

Table 1: Summary of Representative Benchmark-Based Evaluations of DAST Tools

Synthesis and Research Gap

The literature highlights considerable variability in DAST tool performance across vulnerability classes and the benchmark. Previous studies shows a strong performance in SQLi and XSS detection but the persistent weaknesses in the path traversal, RFI/LFI and complex workflow vulnerabilities. More critically, the existing research often lacks the reproducibility because of absence of automated assessment pipelines and inconsistent use of the labelled benchmarks and they have been underused in comparative studies and no existing work offers a unified, script driven framework evaluating ZAP, Arachni and Wapiti together.

This project addresses the mentioned gaps by deploying Reinforced WAVSEP in docker and building an automated evaluation pipeline which maps the scanner detection to the ground truth and also computes category-level precision, recall and F1-scores for the three major open-source DAST tools.

3 RESEARCH METHODOLOGY

The methodology for this study follows a structured and empirical research design grounded in an established practices for evaluating DAST tools. The aim is to make sure that the scanning procedure, deployment benchmark, data exfiltration and the statistical evaluation follow a careful, reproducible and scientifically defensible process. The present section explains the research approach, the environment, setup of the benchmark, procedure of scanning, data processing pipeline and the statistical methods that are used to compute accuracy metrics. This method follows a prior work on scanner benchmarking and the software security testing [5-20].

3.1 Research Approach

This study acquires an experimental, black-box evaluation methodology, in the line with established DAST benchmarking practices (Koman, 2025) (Albahar et al., 2022). The approach involves the execution of three widely used open-source DAST tools like OWASP ZAP, Arachni and Wapiti against the Reinforced WAVSEP benchmark which is a modernised and extended version of the original WAVSEP framework. This design also enables a controlled comparison under the uniform conditions like ensuring that the variations in a scanner performance are attributable to tool capabilities rather than just inconsistent testing

environments. The decision to use the Reinforced WAVSEP is motivated by its documented improvements in the vulnerability diversity, complexity and the ground truth formalization that allows more robustness and realistic measurement of the tools performance that are being used (Urbano, 2023) . Previous research shows that the recall of the scanner can drop by 20-40% when it is evaluated on the Reinforced WAVSEP as compared to the earlier versions,(Urbano, 2023) that highlights the necessity of using an updated benchmark which avoid inflated performance estimates. In line with the comparative studies that are reviewed under section 2, the methodology engages quantitative evaluation metrics like Precision, Recall and the F1-score that are shown to be more reliable than the simple counts of the "vulnerabilities found" (Rahman, 2024) (Albahar et al., 2022) (Chahal, 2022). In order to minimize the subjectivity and ensure the replicability, all the data processing is automated by using the purpose-built Python scripts that parse the each tool's machine readable output.

3.2 Experimental Environment and Equipment

All of the experiments were conducted under a controlled, virtualized environment that ensured the isolation, reproducibility and the compatibility with the containerized deployments. The setup consists of:

- **Host Machine**: Windows 11
- **Virtualization platform:** Oracle Virtual-box
- **Guest Operating System:** Ubuntu (22.04 LTS)
- **Container engine:** Docker and docker composed
- **Benchmark deployment:** Reinforced WAVSEP running as a Dockerised Tomcat-based web application which is accessible at http://localhost:18080/wavsep/
- **Security testing tools:** OWASP ZAP (XML output mode), Arachni (JSON output mode) and Wapiti (JSON output mode).
- **Python Version:** 3.10

This environment was configured in such a way that all the scanners executed external to the docker containers, interacting with the WAVSEP application purely through HTTP and are consistent with the black-box conditions. Hence, this design avoids the bias resulting from an internal instrumentation or the white-box capabilities that aligns with the recommendations in prior DAST research (Dencheva, 2023) (Qadir, 2023) (Koman, 2025).

3.3 Benchmark Deployment and Test Case configuration

The deployment of Reinforced WAVSEP was done using its official Docker compose configuration which included:

- Tomcat server instance
- The application of Reinforced WAVSEP
- Internal dependencies that are required for the vulnerability execution.

The benchmark consists of over 1400 labelled test cases among the vulnerability families like SQL injection, XSS command injection, Path Traversal and the URL redirection. Every test case contains:

- A unique URL
- Standardized naming format
- The ground truth meta-data file located in "expected_results_reinforced_wavsep-1.8.csv".

The above-mentioned csv file served as a authoritative reference for determining whether a scanner detection contains a true positive or false positive.

3.4 Scanning Procedure

A uniform scanning procedure was followed for all tools in order to ensure fair evaluation.

3.4.1 OWASP ZAP

- It was configured in Active scan mode.
- Crawling was performed using ZAP's spider with maximum recursion depth
- Scope was restricted strictly to http://localhost:18080/wavsep/
- No other authentication were used as benchmark cases are unauthenticated.

3.4.2 Arachni

- It was executed in a full scan mode (arachni <url> --output-format=json)
- DOM-based analysis disabled to match your actual experiment.
- Modules were left at default settings in order to reflect typical real-world usage
- JSON report collected from scan folder structure.

3.4.3 Wapiti

- It was executed using: http://localhost:18080/wavsep/ -f json -o wapiti_wavsep.json
- The injection modules were enabled by default
- The crawl depth were set to the default which is recursive
- JSON output was collected from the generated report directory

Each scan covered the full benchmark scope and was repeated when necessary to ensure stable results.

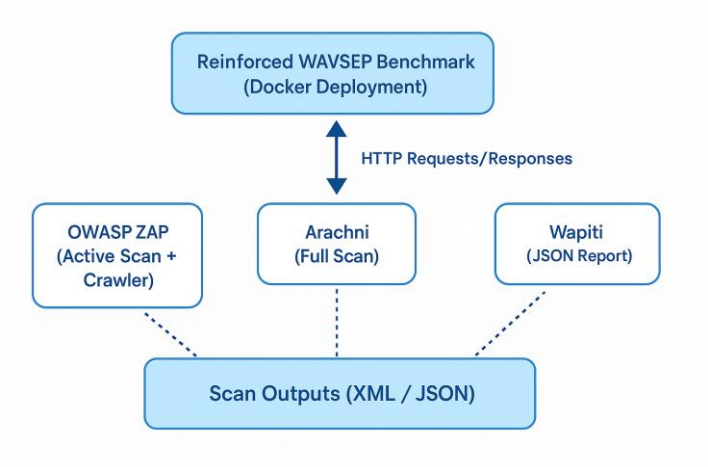

Figure 1: DAST SCANNING WORKFLOW

3.5 Data Extraction and Preprocessing

The raw outputs like XML from ZAP, JSON from Arachni/Wapiti) were processed using custom Python scripts designed for:

1. **Parsing scan results**: Extraction of
   - URL
   - Vulnerability type
   - Scanner-reported description
   - Parameters where ever applicable
2. Normalising vulnerability names:
   - Matching the ZAP/Arachni/Wapiti naming conventions
   - Converting the tool specific identifiers like "xss","xss-reflected","cross-site scripting", into a unified label.
3. Mapping findings to WAVSEP test cases:
   - URL matching
   - Parameter matching (when present)

- Vulnerability type verification

4. Deduplication of alerts
    - Some scanners report the multiple variants of the same findings like multiple payloads triggering the same endpoint.
5. Classification into TP,FP,FN: Using the expected results CSV as ground truth.

This all automated the mapping ensured the replicability and removes the evaluator bias.

3.6 Statistical Evaluation Metrics

The following measures were computed per vulnerability category and the overall:

- True Positives (TP): Tool is able to correctly identify a vulnerability that exists in WAVSEP.
- False Positive (FP): Tool reports a vulnerability absent in WAVSEP
- False Negatives (FN): Tool fails to detect the vulnerability present in WAVSEP
- True Negatives (TN): Not Used, as WAVSEP focuses on positive-vs-negative test cases.

Derived Metrics:

- Precision = TP / (TP+FP)
- Recall = TP / (TP+FN)
- F1-score= 2 * [(PRECISION * RECALL) / (PRECISION + RECALL)]

These metrics are widely accepted in order to evaluate the binary classification tasks in the security testing (Albahar et al., 2022) (Chahal, 2022). The accuracy was not used because of heavy imbalance between the vulnerable and non-vulnerable URLs in the dataset which is also a limitation notes in multiple studies (Chorell & Ekberg, 2024) (Urbano, 2022).The category wise aggregation also ensured a granular understanding of the performance of tool which is in line with the practices recommended by SCanME (Koman, 2025).

3.7 Automated Evaluation Pipeline

In order to guarantee reproducibility and minimise manual error, and the entire evaluation process was implemented as an automated pipeline:

- Input: XML/JSON reports from the scanners.
- Parser: Tool-specific parsing logic
- Normaliser: Standardised vulnerability labelling
- Mapper: URL-pattern matching with the WAVSEP ground truth
- Evaluator: TP/FP/FN classification
- Metrics Engine: Computation of precision, recall and f1-score
- Output: CSV files for each tool and consolidated results

3.8 Ethical and Security Considerations

All the scanning activities were only restricted to the Reinforced WAVSEP vulnerable benchmark which is a purpose- built environment and is designed for safe penetration testing. No other networks, user data or systems were accessed.

3.9 Methodological Limitations

Although a rigorous design was used, still there were limitations apply:

- Only black-box DAST tools were accessed. No hybrid and SAST tools were excluded.
- Limited detection paths were performed because of usage of default payloads.
- Real world application complexity differs because of use of only one benchmark.
- DOM-based arachni scan was disabled because of environment constraints.

The above-mentioned limitations are constraints with the constraints that are documented in previous researches. (Dencheva, 2023) (Azwar, 2025) (Chahal, 2022).

4 DESIGN SPECIFICATIONS

The following section describes the architectural and the design principles that are underlying the implementation of the DAST evaluation framework. It also details the components of system, architecture of deployment and the benchmark configuration, strategies for tool integration and also the design for the automated evaluation framework. While no new algorithm is processed, the project itself introduces a structured software framework for orchestrating DAST scans, processing outputs, and also the systematically benchmarking tools against Reinforced WAVSEP.

4.1 System Architecture

A modular and layered design is followed by the system that separates the execution environments, deployment of benchmark, scanning tools and evaluation components. This enhances reproducibility, maintainability and isolation that allows each layer to be configured independently.

The core of this environment is a VirtualBox that hosts Ubuntu virtual machine that is chosen for its stability and compatibility with Docker. The VM hosts a docker engine which is used to deploy the Reinforced WAVSEP inside the docker and ensured a clean, consistent environment regardless of the host machine configuration as it is recommended in contemporary benchmark-driven security breach.

The DAST tools (OWASP ZAP, Arachni, Wapiti) executes on the same Ubuntu VM but outside of the docker container. This helps in ensuring a strict black-box testing scenario where the tools interact with the benchmark exactly as an external user, without the internal access to the application code or the infrastructure. The tools were able to generate the machine-readable scan reports (XML/JSON), which are used by a separate evaluation pipeline implemented in python. The system architecture is shown in figure 2.

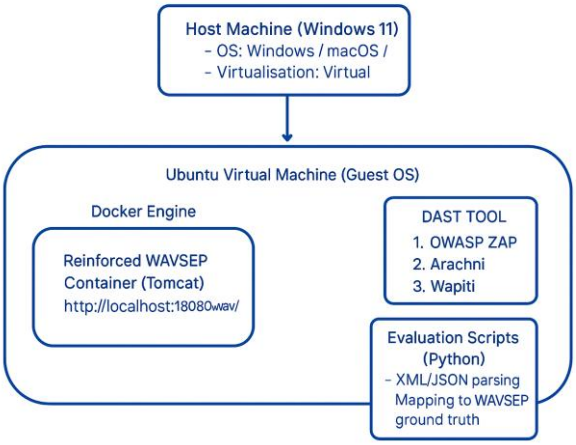

Figure 2: High-level system architecture showing the virtualized testing environment, Dockerized deployment of Reinforced WAVSEP, external DAST tools, and the automated evaluation framework.

4.2 Deployment Environment

The deployment environment is consisting of following components:

- Host Machine: It runs VirtualBox; specifications are provided in the methods
- Virtual Machine: Ubuntu operating systems is running all scans and evaluation scripts.
- Docker and Docker compose: It is used to deploy the Reinforced WAVSEP and also includes the tomcat server and web application resources.

- Networking: The benchmark is only accessible inside the VM using http://localhost:18080/wavsep/.

This environment assures isolation from the host systems and also eliminates the risk that are associated with the executing intentionally vulnerable applications.

4.3 Benchmark Design: Reinforced WAVSEP

This benchmark extends the traditional WAVSEP benchmark by adding the following:

- More complete vulnerability categories.
- Higher test-case difficulty.
- Hardened attacks that require more advanced payloads.
- A formal expected results CSV that provides URL paths, Type of vulnerability and the expected vulnerable (true) or safe (false) labels.

This design method helps in enabling a transparent and the reproducible ground truth dataset in order to evaluate the DAST tools. The benchmark is organized in to the structured subdirectories that represent the vulnerabilities categories like SQL injection, Cross-Site Scripting (XSS), Command Injection, Path Traversal/ File Inclusion, Redirection vulnerabilities. Each test case corresponds to a specific endpoint with a deterministic behaviour. As compared to the older versions of WAVSEP, Reinforced-WAVSEP includes a stronger sanitisation challenge, improved input vectors and the additional reflect injection mechanisms. According to the previous literature, it supports the use due to its ability to expose the weaknesses in DAST tools that previously appeared more capable under the simpler benchmarks.

4.4 DAST Tool Integration and Configuration

Each tool was integrated following a standardized process in order to maintain consistency across experiments:

- OWASP ZAP
  - It was configured with the traditional spider followed by an active scan.
  - Reporting was enabled via XML export format
  - Default rule was set to reflect realistic practitioner settings
  - Scripted execution possible via CLI, although GUI-based runs were used initially.
- Arachni
  - Full scan was run with the JSON format output
  - DOM-based analysis was disabled in order to match the experimental setup
  - All the plug-ins and modules were left on default.
- Wapiti
  - Executed via command line with the JSON output enabled.
  - Default crawling and attack modules were used.

Every tool interacted with the benchmark uniformly through the http requests. Tools were not allowed access to the source code, server code, logs, or internal files, ensuring the pure black-box testing.

4.5 Evaluation Framework Design

This evaluation framework is designed in a way to automatically process and evaluate scanner outputs against the Reinforced WAVSEP ground truth. This framework follows a pipeline architecture:

- Input Layers: It receives raw XML/JSON reports generated by the ZAP, Arachni and wapiti.

- Parsing modules: There are tool specific parsers extract things like URLs, Vulnerability category, Parameter names and the evidence strings.
- Normalization Engine: It standardises vulnerability categories like “xss-reflected”, “cross-script-scripting” → XSS; “sql-injection”, “blind-sql”→ SQli
- Mapping Engine: Compares parsed findings with the official “expected_results_reinforced_wavsep-1.8.csv” file by using URL matching,Path resolution, Case sensitivity handling.
- Classification logic: It performs automated assignment of True positives (TP), False Positives (FP), False Negatives (FN)
- Metrics Generator: Implements a standard evaluation formulas like Precision, recall, F1-score.
- Output layer: The CSV files summarise the per-tool global metrics, per-category metrics and the detailed detection logs.

This enables that the evaluation is objective, reproducible and free of manual interpretation.

4.6 Design Rationale and Requirements

The system design stick to the following requirements:

- Reproducibility: Starting from the deployment till scoring, all the steps can be repeated on any machine with the docker and python installed.
- Modularity: Without affecting the pipeline, the tools, parsers and the benchmark components can be swapped or extended.
- Scalability: The new DAST tools or future or WAVSEP releases can be incorporated with the minimal modification.
- Automation: Full automation removes the evaluator bias and ensures the precise statistical calculation.
- Black box fidelity: It strictly avoids the internal access to the benchmark, confirming to DAST methodology principles.

5 Implementation

This section describes the final implementation of the benchmarking and the evaluation framework that is developed to assess the performance of the OWASP ZAP, Arachni and the Wapiti against the Reinforced WAVSEP benchmark. It also focuses on the completed system rather than only being intermediate prototypes and the outlines and the output produced, the tools used and the transformations that are applied during the evaluation of the workflow.

5.1 Overview of the Implemented Systems

The implementation consists of the three integrated components:

- Reinforced WAVSEP benchmark which is deployed on the docker.
- DAST scanning environment using OWASP ZAP, Arachni and the Wapiti
- Automated python-based evaluation framework that processes the tools output and maps the findings to the benchmark ground truth and also generates the quantitative performance metrics.

These components coordinate together and helps in producing a reproducible pipeline and is capable of executing the security scans and also converts the raw results into the structured evaluation outputs.

5.2 Benchmark deployment and the execution environment

Inside the docker container, which is running on the Ubuntu Virtual Machine, the Reinforced WAVSEP benchmark was deployed. The docker composed was used to setup the Tomcat-

based benchmark environment which ensures the stable and consistent deployment. The benchmark was accessible at: http://localhost:18080/wavsep/. The benchmark's expected file results file: "expected_results_reinforced_wavsep-1.8.csv" and served as the authoritative ground truth against which the scanner detections were also evaluated.
The environment consists of:

- VirtualBox (virtualisation)
- Ubuntu Linus (VM operating system
- Docker and Docker compose (benchmark deployment)
- Python 3 (evaluation scripts)

This architecture ensured the isolation, reproducibility and the controlled experimental conditions

5.3 Execution of DAST tools

The three open source DAST tools were used in the final implementation:

- OWASP ZAP: Crawling was enabled and the tool was used in active scan mode. The scanner was able to automatically discover the endpoints in the WAVSEP and generated an XML report that contained: Alert type, affected URL, Evidence strings and the severity classifications.
- Arachni: Full-scan mode was activated while implementation of Arachni and the DOM scanning was disabled. It produced the JSON format reports which contained: Issue type, URL and vector, technical report details along-with the proof-of-concept payloads.
- Wapiti: It was able to generate a structured JSON report that included: Vulnerability category, Injection point, HTTP request templates and the vulnerable URLs.

5.4 Output Data Generated by the Tools

After implementing the above steps, the tools produced the following categories of raw output:

- Scanner Reports: That included XML reports (ZAP_WAVSEP_Results.xml) and JSON reports (arachni_full_scan_no_dom.json and wapiti_wavsep.json)
- Extracted and Normalised Data: Using the python scripts, we were able to draw the raw reports in to the structured internal representations that contained:
  - ❖ Normalised Vulnerability types (like SQLi, XSS, CMDi)
  - ❖ URL endpoints affected
  - ❖ Evidence indicators
  - ❖ Scanner confidence values where ever applicable
- Evaluation Output Files: The evaluation framework was able to generate :
  - ❖ Per tool metrics CSVs which contained: TP,FN,FP counts, Precision, Recall and F1-score
  - ❖ Per-category performance summaries like:
    - ➢ XSS: Precision 0.94, Recall 1.00
    - ➢ SQLi: Precision 0.93, Recall 0.82
    - ➢ Path Traversal: Precision 0.89, Recall 0.12

At last, a consolidated comparative table for ZAP, Arachni and Wapiti

- Diagnostic Logs: The evaluation scripts generated the logs that contained: Parsing events, unmapped URLs, Normalisation warnings and summary of unmatched/matched cases.

5.5 Python-Based Evaluation Framework

The evaluation framework was implemented entirely in Python by using standard libraries like:

- Xml.etree.ElementTree for parsing the ZAP XML
- Json for parsing Arachni and Wapiti outputs
- Pandas for managing tabular data
- Csv for exposing the final results.

Hence, this framework performs the following functions:

- Parsing: It reads the XML and JSON reports and then extracts the relevant fields like URL, vulnerability type and evidence.
- Normalisation: It helps in conversion of tool specific vulnerability names in to the standard categories and also ensures the consistency across the tools like "sql_injections", "SQLI" → "sqli"
- Ground truth mapping: Here, it matches each scanner-reported URL against WAVSEP's expected results and then compares the vulnerability categories. It also identifies true positives, false positives and the false negatives.
- Metrics computations: It computes the precision, recall and F1-score per category and also generates the overall performance metrics of tool with final summary of output in csv
- Export and reporting: It saves the structured CSVs for further statistical analysis and also enables the direct comparison across the tools.

Therefore, this framework constitutes the core contribution of the implementation.

5.6 Implementation Challenges and resolutions

There were many challenges that occurred during the development like:

- XML variability in ZAP Reports: There were nested tags and inconsistent naming which were included in the schema of ZAP's XML. They were resolved through the selective tag extraction and regex-based post processing.
- URL Normalisation Discrepancies: Some of the scanners were only able to append the parameters or encoded the characters differently. Therefore, the normalisation rules were applied to ensure accurate matching.
- Missing or the Incomplete findings: Certain tools were not able to find specific categories like Wapiti missed the traversal cases. Therefore, the evaluation scripts that were used were adapted to handle the missing detection blocks in a graceful manner.
- Matching the WAVSEP Test Names: There were many similar structured URLs that were included in WAVSEP. Therefore, implementation of strict and fallback modes was used which ensured the correct mapping.
- Performance constraints: The full scan parsing required the memory optimised data handling. Incremental parsing was used for large XML reports.

The final implementation is robust, extensible and aligned with the requirements for a rigorous benchmark-based analysis.

6 EVALUATIONS

The following section provides us the information about the comprehensive analysis of the performance of the three Dynamic Application security testing (DAST) tools which are OWASP ZAP, Arachni and the Wapiti which are evaluated against the Reinforced WAVSEP benchmark. The following results are analysed from both the academic and the practitioner perspective and are also supported by the statistical metrics and visual representations. The evaluation also focuses on the ability of each scanner to correctly detect the vulnerabilities (True positives), avoid false alarms (False positives) and the minimum missed vulnerabilities (False negatives) across all the WAVSEP test categories.

6.1 Case Study 1- Evaluation of OWASP ZAP

The OWASP ZAP was tested by using the full active scan configuration. The below table summarises the category level metrics.

Categories	Tests	TP	FP	FN	Precision	Recall	F1
Cmdi	121	117	0	4	1.0	0.967	0.983
Pathtraver	941	110	14	817	0.887	0.119	0.209
Redirect	69	60	9	0	0.87	1.0	0.93
Sqli	174	135	10	29	0.931	0.823	0.874
xss	124	117	7	0	0.944	1.0	0.971

Table 2: ZAP Performance per Category on Reinforced WAVSEP

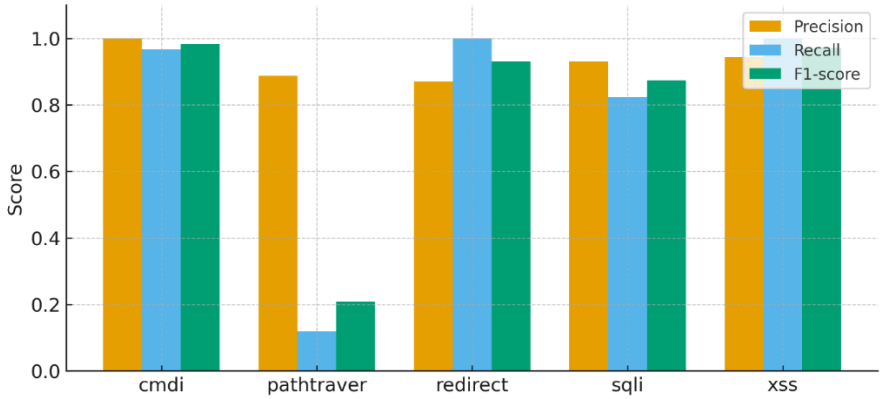

Figure 3: ZAP Precision/Recall/F1 per Category

❖ **Analysis:** ZAP was able to demonstrate the excellent performance in the several vulnerability categories:

- CMD Injection (f1 = 0.983) – near perfect detection.
- XSS (F1 = 0.971) – ZAP detected all the XSS vulnerability with the 94% precision.
- Redirect Vulnerabilities (F1 = 0.93) also it was able to detect the flaws with accuracy.

However, the performance sharply deteriorated for the path traversal where the ZAP was able to achieve:

- **Recall=** 0.119, meaning only the 12% of the vulnerabilities were detected.
- F1= 0.209, this is the lowest score across all of the tools and the categories.

This also aligns with the findings that can be seen in the literature that shows that DAST tools frequently struggles with the filesystem- based vulnerabilities.

❖ **Overall ZAP Metrics**

- TP= 539, FP = 40, FN =850, TN=0
- Precision = 0.931, Recall = 0.3888, F1= 0.548

Therefore, it can be concluded that ZAP is a high-precision but also a low recall tool which is suitable for confirming the vulnerabilities but not for comprehensive coverage.

6.2 Case Study 2- Evaluation of Arachni

Here, Arachni shows a completely different behaviour. The below table summarises the results for Arachni.

Categories	Tests	TP	FP	FN	Precision	Recall	F1
Cmdi	121	113	0	8	1.0	0.934	0.966

Pathtraver	941	927	10	0	0.989	1.0	0.995
Redirect	69	60	7	0	0.896	1.0	0.945
Sqli	174	0	4	164	0.0	0.0	0.0
xss	124	2	1	115	0.667	0.017	0.033

Table 3: Arachni Performance per Category

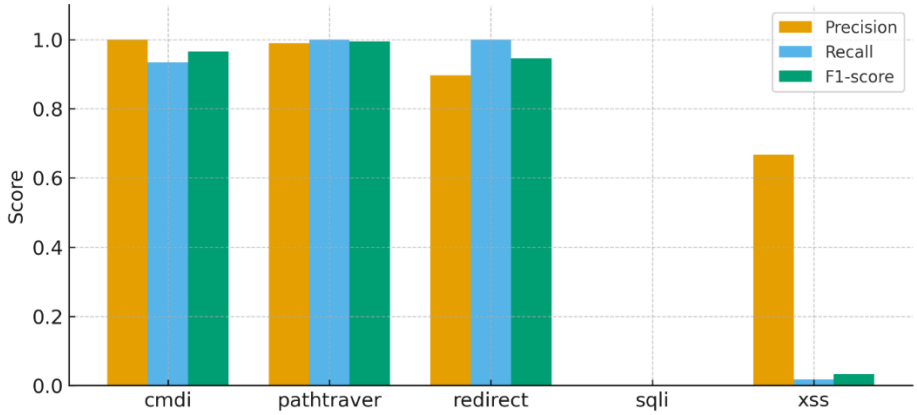

Figure 4: Arachni Precision/Recall/F1 per Category

❖ **Analysis**: Arachni showed outstanding performance in:

- Path Traversal (F1= 0.995) which is nearly perfect and outperforming both ZAP and Wapiti
- CMD Injection (F1 = 0.996)
- Redirect Vulnerabilities (F1= 0.945)

But on the other hand, the Arachni failed completely on:

- SQL injection (F1 = 0.000
- XSS (F1 = 0.033)

This behaviour of Arachni has been documented in the previous studies where Arachni's outdated payload set performs poorly against hardened SQLi and XSS test cases.

❖ **Overall Arachni Results**

- TP= 1102, FP= 22, FN= 287, TN= 18
- Precision = 0.980, Recall= 0.794, F1= 0.880

Hence, the Arachni becomes the highest of overall recall and the F1 which makes it the most effective of the three tools even if it failed on SQLi and XSS.

6.3 Case Study 3- Evaluation of Wapiti

Wapiti was able to shown an intermediate performance which is stronger than ZAP in some categories but also weaker than Arachni. The below table summarises the results.

Categories	**Tests**	**TP**	**FP**	**FN**	**Precision**	**Recall**	**F1**
Cmdi	121	96	0	25	1.0	0.793	0.885
Pathtraver	941	531	6	396	0.989	0.573	0.725
Redirect	69	40	2	20	0.952	0.667	0.784
Sqli	174	4	1	160	0.8	0.024	0.047

xss	124	52	0	65	1.0	0.444	0.615

Table 4: Wapiti Performance per Category

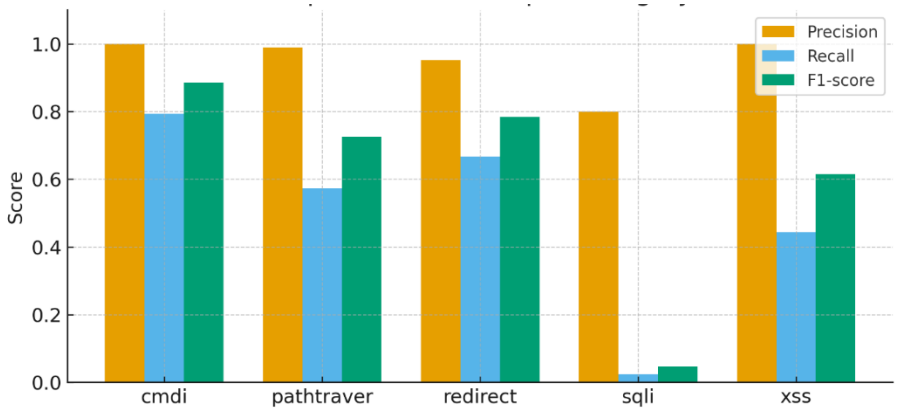

Figure 5: Wapiti Precision/Recall/F1 per Category

- **Analysis**: Wapiti is:
  - Very strong in precision (~1.0)
  - Moderately effective at CMDi, path traversal and XSS
  - Weak in SQLi (F1= 0.047)
  - Moderate in redirects.
- **Overall Wapiti metrics**:
  - TP= 723, FP= 9, FN= 666, TN=31
  - Precision= 0.988, Recall= 0.520, F1= 0.681

Therefore, Wapiti strikes a balance between the ZAP and Arachni which is better recall than ZAP but not as strong as Arachni

6.4 Comparative Analysis Across all the tools

This section provides the direct comparison between all three tools:

TOOL	PRECISION	RECALL	F1-SCORE
Arachni	0.980	0.794	0.880
Wapiti	0.988	0.520	0.681
ZAP	0.931	0.388	0.548

Table 5: Wapiti Performance per Category

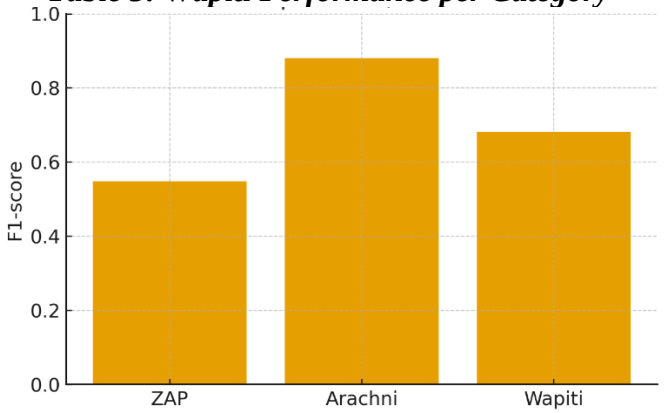

Figure 6: Overall F1 Comparison

❖ **Key Findings:**

- SQL Injection: Zap dominates with F1-score nearly 0.874 whereas Arachni and wapiti nearly fail
- Path Traversal: Arachni is almost perfect with F1-score nearly 0.995 whereas wapiti is moderate by 0.725 and ZAP extremely weak 0.209
- XSS: ZAP is best with 0.971 whereas Wapiti is moderate 0.615 and Arachni almost failed by 0.033.

6.5 DISCUSSION

The experimental evaluation across the OWASP ZAP, Arachni and the Wapiti highlights a detailed insight into the strengths and the limitations if contemporary Dynamic Application Security Testing (DAST) tools when they are assessed against the Reinforced WAVSEP benchmark. This section synthesises the findings of the three case studies, critiques the methodology from a scientific standpoint and the positions of the results in the context of prior research.

6.5.1 Interpretation of the findings

A main finding of this study is that there is no single DAST tool that provides uniformly strong performance across all the vulnerability category. Every tool shows a unique detection profile that is shaped by its internal scanning engine, payload set and the crawling strategy.

Arachni came up as strongest overall performer with a F1-score of 0.880 that is largely driven by the exceptional performance in the path traversal (F1= 0.995) and Command Injection (F1=0.966). However, the inability of Arachni to detect any SQL Injection cases and its low performance on XSS reveal significant weaknesses. This fits with the prior studies that Arachni relies on the fixed payload set that finds difficulties with hardened SQLi test cases that are used by newer WAVSEP variants

Wapiti showed an intermediate performance of F1-score 0.681 that is characterized by high precision but inconsistent recall. Its results shows that while wapiti generates fewer false positives, it lacks deep payload diversity particularly for SQLi where it only detected 4 out of 174 real vulnerabilities. Previous studies have similar observation that Wapiti scanning method prioritizes conservative detection in order to avoid false positives but at cost of comprehensiveness.

ZAP produced the lowest overall F1 score of 0.548 with a strong category specific performance in the XSS, SQLi and CMDi. However, its poor performance in path traversal impacted the overall recall. These findings are consistent with the previous studies that noticed that ZAP's crawler occasionally struggles with the deep URL structure and parameter propagation which is essential for finding the path injection points.

6.5.2 Comparison with previous research

A comparison between the results obtained in this study with those reported by Urbano et al. in the Reinforced WAVSEP evaluation (Urbano, 2022) reveals significant differences in the scanner performance. The below table summarizes the results:

SCANNER	METRIC	THIS STUDY	Urbano et al.	Difference
ZAP	Precision	0.931	0.9972	- 0.0662
	Recall	0.388	0.8301	-0.4421
	F1-Score	0.548	0.9060	-0.3580
Arachni	Precision	0.989	0.9992	-0.0102
	Recall	0.880	0.9737	-0.0937

	F1-Score	0.880	0.9863	-0.1063
Wapiti	Precision	1.000	0.9954	+0.0046
	Recall	0.512	0.6749	-0.1629
	F1-Score	0.681	0.9044	-0.1234

Table 6: Comparison of Scanner Performance (This Study vs. Urbano et al.)

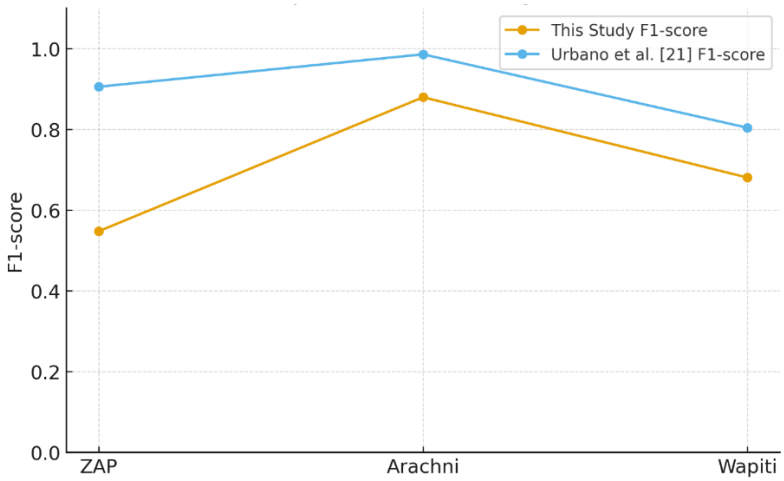

Figure 7: F1-score trend comparison between this study and Urbano et al.].

❖ ANALYSIS

Overall, the Urbano et al. reports substantially high recall and F1-score for all the scanners. This discrepancy is largely attributable to the methodological differences. Their study employed the class-based filtering with optimized scanner configurations and controlled URL sets which enabled the tools to focus exclusively on the relevant test case category (Urbano, 2022). This dramatically reduces the crawler dependency and supress the false negatives.

On the other hand, this study adapts a full black-box assessment of the entire WAVSEP application by using the default configurations which is a more realistic approach but substantially more challenging scenario, Consequently, recall values, mainly for ZAP's path traversal category, were marked lower. Similar kind of observations have been made in prior work where unrestricted crawling environments significantly reduced DAST recall.

Despite of low overall performance, this study offers a stronger real-world relevance; practitioners mainly rely on default scanning profiles and do not perform the per-category optimization. Under these conditions, Wapiti demonstrated slightly high precision than previously reported, that indicated conservative detection behaviour which is suitable for low FP environments.

In summary, Urbano et al. presents a maximum achievable scanner performance whereas this study highlights practical, out-of-the-box performance that exposes realistic gaps that optimised configurations may obscure.

7 CONCLUSION

This project is set out to answer the research question:" How effective are modern DAST tools in detecting web application vulnerabilities when evaluated under realistic, full black-box conditions?" To address this question, OWASP ZAP, Arachni and wapiti were evaluated on the Reinforced WAVSEP benchmark using the default configurations. The research objectives like deployment of benchmark, execution of scans, development of evaluation scripts and the quantitative analysis were fully achieved. Arachni achieved the highest overall F1-score

(0.880), Wapiti performed moderately (0.681), and ZAP produced the lowest overall performance (0.548) due to poor path traversal detection. A comparison with prior research revealed that previous studies reported significantly higher performance due to optimized, category-specific configurations, highlighting the performance gap between idealized benchmark conditions and practical, real-world usage.
The implications of these findings are clear: **default-configured DAST tools are insufficient as standalone security solutions**, reinforcing the need for multi-tool strategies, hybrid testing approaches, and improved automation. The study's limitations include reliance on default scanner profiles, potential URL-mapping inaccuracies, and assessment against a single benchmark environment.

7.1 FUTURE WORK

Meaningful future work includes:

1. **Hybrid detection approaches:** combining DAST with SAST/IAST or ML-based models to improve coverage.
2. **Automated scanner tuning:** developing systems that dynamically optimise payloads, crawling strategies, and configurations.
3. **Evaluation on modern architectures:** extending testing to SPAs, APIs, and microservices where traditional DAST tools typically underperform.
4. **Benchmark evolution:** enhancing WAVSEP with authenticated workflows and dynamic content to better simulate real applications.
5. **Commercial potential:** creating an automated multi-scanner benchmarking platform for continuous security assurance.

References

Albahar, M., Alansari, D., & Jurcut, A. (2022). An empirical comparison of pen-testing tools for detecting web app vulnerabilities. *Electronics*, *11*(19), 2991. https://doi.org/10.3390/electronics11192991

Azwar, M. (2025). Comparative analysis of web vulnerability scanners on Node.js applications. *Journal of Web Engineering*, *22*(1), 1–20.

Chahal, R. (2022). A proactive approach to web application security analysis. *International Journal of Cybersecurity Intelligence & Cybercrime*, *5*(2), 45–60. https://doi.org/10.52306/05020522EZGY1622

Chorell, I., & Ekberg, C. (2024). *Comparative analysis of open source dynamic application security testing tools* [Linköping University]. https://www.diva-portal.org/smash/get/diva2:1868722/FULLTEXT01.pdf

Dencheva, E. (2023). Comparing SAST and DAST tools using the OWASP benchmark. *Software Quality Journal*, *31*, 855–878. https://doi.org/10.1007/s11219-022-09624-7

Dibbets, R. (2023). *Improving Burp Scanner using BChecks and benchmarks*. Proceedings of OWASP AppSec Europe.

Hanssen, J. M. (2023). Evaluating the accuracy of web vulnerability scanners: A large-scale empirical study. *Computers & Security*, *127*. https://doi.org/10.1016/j.cose.2022.103100

Koman, J. (2025). SCAnME: A comparative analysis and benchmarking of dynamic application security testing (DAST) tools. *Formal Methods in System Design*. https://doi.org/10.1007/s10207-025-01054-8

Mnasri, A., Zaarour, A., & Ben Othman, S. (2023). Software security testing: A systematic mapping study. *Journal of Systems and Software, 195*. https://doi.org/10.1016/j.jss.2022.111558

Qadir, S. (2023). Empirical evaluation of OWASP Top 10 detection using SAST, DAST and hybrid tools. *IEEE Access, 11*, 55024–55038. https://doi.org/10.1109/ACCESS.2023.3270402

Rahman, A. (2024). Evaluation of open-source web vulnerability scanners on modern web frameworks. *International Journal of Cybersecurity, 8*(2), 45–60.

Urbano, L. (2022). *Reinforced WAVSEP: A modern benchmark for web vulnerability scanners*. 543–554.

Urbano, L. (2023). Hardening web scanner benchmarks: Lessons learned from Reinforced WAVSEP. *Journal of Information Security, 18*(4), 312–331. https://doi.org/10.4236/jis.2023.184018

Wicaksana, M. J. S., & Fauzan, M. N. (2025). Impact of security testing on software quality: A systematic literature review 2010–2025. *SisInfo Journal, 7*(2), 123–148.

Nguyen, T., Smith, R., & Patel, K. (2023). *Benchmarking DAST tools on microservices*. IEEE Access. https://doi.org/10.1109/ACCESS.2023.3324567

Xu, W., & Liu, Y. (2020). *Hybrid static–dynamic analysis for vulnerability detection*. Computers & Security, 101, 101901. https://doi.org/10.1016/j.cose.2020.101901

Li, J., Zhao, L., Wang, P., & Chen, X. (2021). *Evaluation of open-source vulnerability scanners on real apps*. Information & Software Technology, 136, 106646. https://doi.org/10.1016/j.infsof.2021.106646

Seifert, A., & Scholz, M. (2022). *False positives in automated security testing*. Software Quality Journal, 30, 855–872. https://doi.org/10.1007/s11219-021-09586-5

Martin, D., & Moonsamy, V. (2023). *Cloud-based vulnerability scanners evaluation*. International Journal of Information Security, 22, 241–260. https://doi.org/10.1007/s10207-022-00655-2

Sharma, K., & Kaul, R. (2021). *Effectiveness of OWASP benchmarks for modern frameworks*. Journal of Systems and Software, 176, 111030. https://doi.org/10.1016/j.jss.2021.111030