The International Journal of Engineering and Information Technology

journal homepage:www.ijeit.misuratau.edu.ly

SAST Tools and Manual Testing to Improve the Methodology of Vulnerability Detection in Web Applications

Elrowayati Ali A.
College of Industrial Technology, Misurata, Libya.
elrowayati@yahoo.com

Fadeel Ammar M.
Misurata University,
Misurata, Libya.
Ammar1892000@yahoo.com

Abstract— **Statically analyzing code during development is a common process of the development process, using Static Application Security Testing tools. SAST analyzes code without its execution and is also very fast compared to dynamic means and therefore focuses on a certain program part. However, the results of static analysis tools are not always accurate, either missing vulnerabilities or reporting false positives. This paper considers an evaluation of several SAST tools and an analysis of student code samples with known vulnerabilities, comparing manual analysis with the results of SAST tools. The results confirmed that SAST tools properly identify critical vulnerabilities and provide errors. A tool has identified ShiftLeft as the most efficient tool; however, its findings overlapped with the results of other tools for some applications. In addition, an analysis of student projects showed the most frequent vulnerabilities as Cross-site Scripting (XSS), NoSQL/SQL Injection, and Server-Side Request Forgery (SSRF) which make up more than 52% of the found vulnerabilities.**

Index Terms— **static application security testing, Code review, vulnerability assessment, Web application security , OWASP.**

I. INTRODUCTION

Web applications have become a crucial component of daily existence, yet numerous applications are deployed with severe vulnerabilities that can be exploited with fatal consequences [1]. Therefore, the quality of software and web applications lies in the absence of vulnerabilities in them, and early detection of these vulnerabilities before they are published is very important. Usually, these vulnerabilities are weaknesses in the source code of these applications which can be resolved by source code review.

Source code review is an important part of ensuring code quality, which in effect is a form of manual static analysis of the code, which is important especially when it comes to finding weaknesses that hide in the code, which are more difficult to find by using dynamic analysis ways; for example, relating to non-functional requirements such as security.

Received 30 Apr , 2024; revised 06 May, 2024; accepted 15 Mar 2024. AVAILABLE ONLINE 08 AUG, 2024.

Nevertheless, code review poses a challenge as humans are prone to errors when performing repetitive and tedious tasks, such as reviewing extensive amounts of source code.

Therefore, some form of automated static code analysis is recommended. Static analysis tools offer several inherent benefits such as early detection of faults when applied in the early stages of development, not requiring fully functional or runnable code, and eliminating the need to develop test cases [2]. One of the most important static analysis tools is the Static Application Security Testing (SAST) tool. This tool, designed on certain rules, can detect various security vulnerabilities without the need to execute the source code.

While both SAST tools and manual code reviews aim to identify and fix vulnerabilities in web applications, SAST offers speed and ease of use. However, SAST has limitations. False negatives (missing real vulnerabilities) and false positives (flagging nonexistent issues) can create a false sense of security. Additionally, effectiveness varies based on the development environment and supported languages, hence making it challenging to choose the best SAST tool. This study has three main goals. First, it investigates the effectiveness of SAST tools in identifying vulnerabilities; that aim is to see how well these tools contribute to creating secure code with minimal weaknesses. Second, it explores which SAST tools are best at detecting exploitable vulnerabilities, with the goal of minimizing security risks and costs. Finally, by applying SAST tools to student code, the study identifies the most common vulnerabilities students make.

The rest of the study is organized as follows. Section 2 is a concept of static code review technologies, Section 3 reviews the related work about SAST tools for web applications security assessment, Section 4 describes the new proposed methodology approach, Section 5 is an analysis and results of the methodology, Section 6

evaluation and discussion, and Section 7 is conclusions and future work.

II. STATIC CODE REVIEW TECHNOLOGIES CONCEPT

A. SDLC AND CODE REVIEW

The Software Development Life Cycle (SDLC) acts as a roadmap for software projects, guiding creation, maintenance, and updates. This structured approach ensures consistent software quality [3]. Code review plays a vital role within the SDLC. It involves examining code to identify issues like bugs, inefficiency, or security weaknesses. Reviews can be manual or automated and occur throughout the development process. This practice offers several advantages: better code quality, stronger security, improved teamwork and knowledge sharing, and lower maintenance costs. By integrating code review into the SDLC, software teams can deliver high-quality code that meets stakeholder expectations [3].

B. VULNERABILITY MANAGEMENT SYSTEM BASED ON SAST TOOLS

Vulnerability management is a critical aspect of any organization's security posture. It involves identifying, prioritizing, and remediating vulnerabilities across systems and applications. Traditional Vulnerability Assessments (VAs) rely on Vulnerability Management Systems (VMS) to scan a broad range of IT infrastructure for weaknesses. However, for application security, a more targeted approach is necessary. This is where SAST tools come into play. This type of assessment is great for showing how good an organization is at performing patching and deploying a secure configuration. The key here is that these types of assessments do not focus on gaining access to critical data from the perspective of a malicious actor, but instead, are related to finding vulnerabilities [4]. SAST tools are a specialized type of VA specifically designed to analyze application source code during development. Unlike VMS tools that scan deployed systems, SAST offers the advantage of early vulnerability detection. By integrating seamlessly into the development lifecycle, SAST helps developers identify and fix security flaws in the code itself, before the application is deployed and potentially exploited. In conclude, building a vulnerability management system based on SAST tools is a strategic approach to application security. By integrating security testing into the development lifecycle, organizations can significantly improve the security posture of their applications, reduce development costs, and achieve faster release cycles. This proactive approach ensures applications are built with security in mind, ultimately enhancing the overall cyber security posture of the organization. [5]

C. SAST TOOLS

SAST tools are a type of security analysis tool that assesses both source code and executable for potential security vulnerabilities. SAST tools are widely regarded as a crucial aspect of a Secure Software Development Life Cycle (SSDLC) [7]. These tools can detect various types of vulnerabilities, including buffer overflow, injection attacks, and cross-site scripting. However, the degree of user-friendliness of SAST tool interfaces can vary significantly, particularly about error tracing [6,7].

III. RELATED WORKS

In the literature, few research studies have evaluated different aspects of software security tools. In [1], the authors focused on evaluated web vulnerability scanners using multiple benchmarks, emphasizing the need for diverse evaluation criteria. In [8], the article specifically assessing the effectiveness of source code analysis tools in identifying memory corruption vulnerabilities, highlighting limitations in detecting such vulnerabilities; however, his study focused on desktop applications written in C++/C and Java only. Recently, in [9], the author examined sixteen SAST tools for identifying vulnerabilities in JavaScript and Python web applications, emphasizing the importance of multiple tools and evaluation strategies; However, some of the popular tools among developers have been overlooked.

IV. METHODOLOGY

As illustrated in Fig.1 below, the methodology for a code review process leverages automation to help identify potential issues in code. The methodology has been broken down into steps:

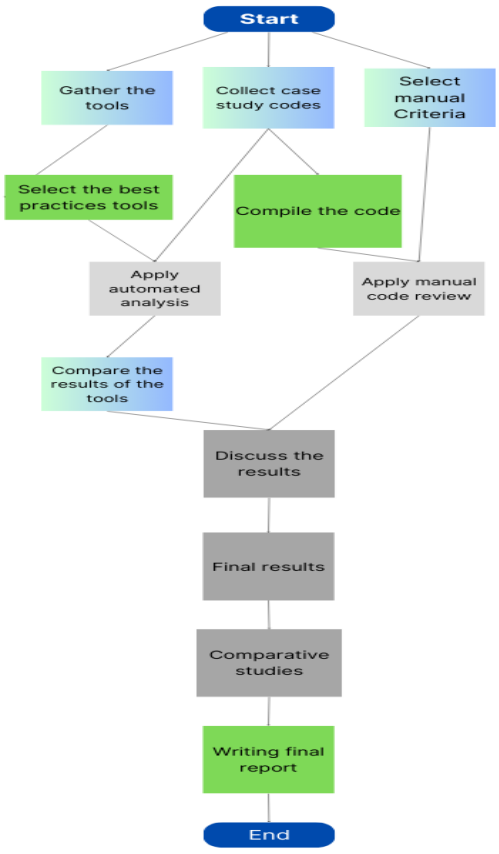

Figure 1. proposal methodology

1. **Select manual analysis criteria:** This research leveraged the "OWASP Code Review Guide Project" [11] to define criteria for manual code review. This approach aimed to streamline the process by focusing on security vulnerabilities within the source code.

2. **Collecting Case Study Code:** Evaluating static analysis tools requires testing them on case study source code. Here, the case studies were chosen by analysing their code and comparing the results. The case studies included :
   - **Vulnerable Code:**
     - Code obtained from GitHub with known vulnerabilities.
     - OWASP Juice Shop project source code.
   - **Real-World Code:**
     - Compiled code from student-designed applications.
     - A portion of source code from an actual project.
3. **Gathering Tools:** The collection of tools was based on the most popular and used in the developer environment, to be a study as close as possible to developers. In addition, to ensure the study reflected real-world developer environments, popular tools were chosen. The initial selection included :
   - SNYK Code So Now You Know (SNYK) Code
   - Horusec
   - Fluid Attack's Scanner
   - WhiteSource Bolt
   - Sonarcloud
   - ShiftLeft Scan
4. **Selecting the Best Tools:** Two key factors guided tool selection :
   - **Language Support:** The tools needed to support the targeted languages (JavaScript/TypeScript, PHP).
   - **Previous Evaluations:** Tools were evaluated based on similar studies using the OWASP Benchmark tool and their performance on case studies.

According to [9], the top two performing tools (SNYK Code and Horusec) were chosen. Additionally, ShiftLeft Scan, a popular developer tool not included in [9], was added.

5. **Compile code:** In order to prepare code for analysis, work was done on collecting various source codes to ensure comprehensive analysis. The focus was on combining code with known vulnerabilities and student-developed projects from Misurata University. The collected source code falls into these categories:
   - GitHub Code: A collection of source codes containing different security vulnerabilities.
   - OWASP Juice Shop: The source code of a vulnerable website designed for security training purposes.
   - Student Projects:
     - Project 1: Source code for a student-developed website, potentially deployed publicly.
     - Project 2: Source code for a student-developed web application, likely used for a student event and not publicly accessible.
6. **Applying Manual and Automated Analysis:** After completing the code preparation phase, we conducted manual code review and then applied SAST tools on the target code samples. To evaluate the effectiveness of the SAST tools in identifying real vulnerabilities, we employed a vulnerability classification process (detailed in Fig. 2). The results of this process will be discussed in the analysis and results section.

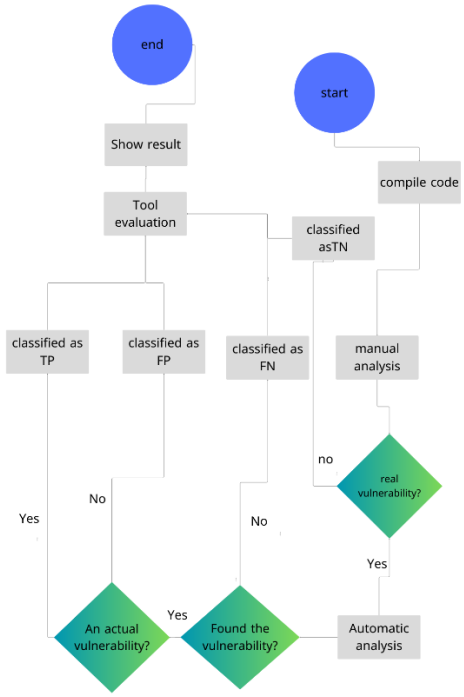

Figure 2. the vulnerability classification process

7. **Comparing the Results of the Tools:** To identify the most effective SAST tool in terms of security vulnerability detection, we compared the results of the SAST code analysis on the target codes. The comparison will be based on standard performance metrics: precision, recall, and F-measure. The equations for these metrics are provided below:

$$Precision = {TP}/{(TP + FP)} \quad (1)$$

$$Recall = {TP}/{(TP + FN)} \quad (2)$$

$$F - measure = 2 \times Precision \times recall/{(precision + recall)} \quad (3)$$

Where:

TP (True Positive): This refers to a case where the tool correctly identifies a real security vulnerability in the code. It's the ideal scenario where the tool detects a genuine problem. [11]

FP (False Positive): This occurs when the tool identifies a potential vulnerability but it's actually a harmless issue or a coding practice that doesn't pose a real security risk. It's a false alarm triggered by the tool. [11]

FN (False Negative): This happens when the tool misses a real security vulnerability in the code. It's a critical error where the tool fails to detect a genuine problem, leaving the code susceptible to attacks. [11]

8. **Final Results and Discussion**
   Following the data analysis stage, the final results from both manual and automated analyses will be presented. These results will be thoroughly evaluated and discussed to identify key findings and draw meaningful conclusions that contribute to the study's objectives. The details of this analysis and discussion will be presented in section F (Evaluation and Discussion).
9. **Comparative Study and Final Report**
   After presenting and discussing the analysis results, we will conduct a comparative study. This involves comparing our findings on SAST tool performance with those of a similar study by reference [9]. This comparison aims to identify the most significant similarities and differences between the two studies. By doing so, we can gain valuable insights into the broader context of SAST tool evaluation and further strengthen the validity of our own results by situating them within a broader context. The findings of this comparative study will also be included in the final report (section F).

V. ANALYSIS AND RESULTS

This section presents the key findings obtained during the different stages of the study, ultimately addressing the goal of evaluating the SAST tools. Here, we will analyze and compare the results from both manual code review and automated analysis using the selected SAST tools.

A. Manual Code Review Results

We begin by reviewing the results of the manual code review process. These results will be presented in a summarized format using a statistical chart for a high-level overview. Fig. 3 depicts the number of security vulnerabilities identified in the target source codes based on the predefined criteria. It's important to note that the manual analysis revealed a total of 209 vulnerabilities.

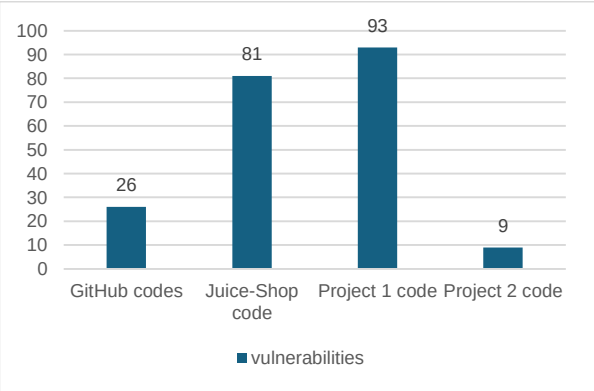

Figure 3 the results of the manual analysis

B. Automated Analysis Results using SAST Tools

Next, we will analyse the results obtained from the automated analysis using the selected SAST tools. Here, we will present the total number of vulnerabilities discovered by each tool within the target source code. The vulnerabilities shown in Fig.4 encompass all potential vulnerabilities detected by the SAST tools, and might include some false positives.

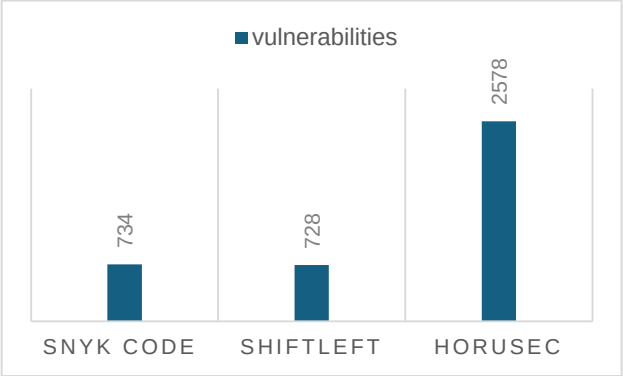

Figure 4 the results of the automated analysis

VI. EVALUATION AND DISCUSSION

A. Discussion of Results

Comparing the manual results with the automatic results here, we compare the results obtained from manual code review with the findings from automated analysis using SAST tools. This comparison aims to assess the effectiveness of the tools in detecting vulnerabilities compared to human expertise. True Positives, False Positives, and False Negatives: We will present the results in table1, focusing on True Positives (TP), False Positives (FP), and False Negatives (FN). Note: Since we analyzed real code with existing vulnerabilities, True Negatives (TN) are not applicable. Performance Metrics: As mentioned earlier, Precision, Recall, and F-measure are crucial metrics for evaluating the performance of classification models like SAST tools. These metrics were defined previously.

Table 1. comparison results between manual and automatic analysis on all target codes

Tools	TP	FP	FN	Recall	precision	F-Measure
SNKY code	174	560	35	0.833	0.237	0.369
ShiftLeft	176	552	33	0.842	0.241	0.375
Horusec	176	2402	33	0.842	0.068	0.126

By comparing the performance of the SAST tools across different project environments and security vulnerabilities, it's evident that their effectiveness can vary. While ShiftLeft emerged as the tool with the highest F-measure, indicating a good balance between vulnerability detection and minimizing false positives, our study emphasizes the value of using multiple tools for a more comprehensive code review. Furthermore, combining manual code review with automated analysis provides the most robust approach to identifying security vulnerabilities. Notably, Cross-site Scripting (XSS) and NoSQL/SQL Injection vulnerabilities were frequently detected, highlighting their prevalence and criticality in web applications.

B. Comparative studies

Comparing our findings with previous studies helps validate our results, identify potential limitations, and potentially offer new recommendations. In this section, we will compare the performance of the SAST tools used in our study with the findings of a similar study in [9].

This prior study evaluated these same tools on a set of 16 applications. Furthermore, we focused on two key comparisons:

Overall Tool Performance: This comparison will analyze the F-measure scores achieved by each tool in both studies across all applications (including those written in languages besides JavaScript). The results are presented in Table 2.

Table 2. comparing the results of the tools analysis of the target codes with the results of the previous study [9].

Tools	Our study	[7]
SNKY code	0.369	0.7
Horusec	0.126	0.73
ShiftLeft	0.375	-

JavaScript-Specific Performance: Here, we will compare the F-measure scores for each tool specifically on JavaScript applications within both studies. The results are presented in Table 3.

Table 3 comparing the results of the tools analysis of the target codes with the results of the previous study [9] based on JavaScript codes

Tools	Our study	[7]
SNKY code	0.369	0.712
Horusec	0.126	0.706
ShiftLeft	0.375	-

As shown in Table 2, there are significant differences in the overall F-measure scores between our study and [9]. This can be attributed to variations in the target application environments. Our study included applications written in multiple languages, whereas [9] focused solely on JavaScript. Table 3 highlights similar trends for JavaScript-specific performance. Here, while Snyk Code and Horusec achieve lower F-measure scores compared to [9], it's important to consider that ShiftLeft, which achieved the highest F-measure in our study, wasn't included in the previous researches such as [9].

VII. CONCLUSIONS AND FUTURE WORK

A. Conclusion

We have conducted an empirical study regarding the effectiveness of Static Application Security Testing (SAST) tools on detecting security vulnerabilities in source code by comparing the results of manual code review to SAST analysis for various case studies. We have shown that SAST tools effectively detected a noticeable proportion of security vulnerabilities. Notably, ShiftLeft was the most effective tool for the evaluated JavaScript/TypeScript and PHP web applications. We identified Cross-site Scripting as the most common vulnerability in student-developed web applications. Most importantly, some SAST tools also detected and provided alerting to the developers regarding this critical vulnerability. Tools such as SNYK Code even gave remediation recommendations for students to build more secure applications. In conclusion, this study met its objectives effectively as SAST tools are a great complement to manual code review to enhance the security of web applications in mitigating potential security risks.

Future work

This research paves the way for several compelling areas of future exploration:

- Development of effective developer education. Further studies could be undertaken in order to develop effective educational programs that will stimulate the adoption of SAST tools among developers and students in large numbers. This would significantly contribute to a more secure software development landscape.
- Integrating code review practices: Further research is necessary to develop strategies to help integrate code review practices into the workflow of a developer or a student. The ability to share and check on code is one of the best ways to ensure application security and safety.
- OWASP Benchmark Project: This research finds this very valuable and practical for inquiring into how well the OWASP Benchmark project is practically deployed and exploited. The study offers a rich set of evaluation tests that developers and students can exploit to find the most effective tools for detecting and remedying vulnerabilities within their applications well before deployment.

ACKNOWLEDGMENT

This work is partly supported by faculty of IT, Misurata University, Libya. Our gratitude is also extended to the National Information Security and Safety Authority (NISSA)for its cooperation and the assistance and valuable information it provided for this study.

REFERENCES

[1] Mburano, Balume, and Weisheng Si. "Evaluation of web vulnerability scanners based on owasp benchmark." 2018 26th International Conference on Systems Engineering (ICSEng). IEEE, 2018.

[2] Axelsson, Stefan, et al. "Detecting defects with an interactive code review tool based on visualisation and machine learning." the 21st International Conference on Software Engineering and Knowledge Engineering (SEKE 2009). 2009.

[3] Winters, Titus, Tom Manshreck, and Hyrum Wright. Software engineering at google: Lessons learned from programming over time. O'Reilly Media, 2020.

[4] Duffy, Christopher, et al. Python: Penetration Testing for Developers. Packt Publishing Ltd, 2016.

[5] Roytman, Michael, and Ed Bellis. Modern Vulnerability Management: Predictive Cybersecurity. Artech House, 2023.

[6] OWASP website, OWASP Code Review Guide, access date 5/7/2023 https://owasp.org/www-project-code-review-guide/ .

[7] Hsu, Tony Hsiang-Chih. Practical security automation and testing: tools and techniques for automated security scanning and testing in devsecops. Packt Publishing Ltd, 2019.

[8] Tejning, Johan. "Vulnerability assessment of source codeanalysis tools for memory corruptionvulnerabilities a comparative study." (2021).

[9] Di Stasio, Vincenzo. Evaluation of Static Security Analysis Tools on Open Source Distributed Applications. Diss. Politecnico di Torino, 2022.

[10] Higuera, Juan R. Bermejo, et al. "Benchmarking Approach to Compare Web Applications Static Analysis Tools Detecting OWASP Top Ten Security Vulnerabilities." Computers, Materials & Continua 64.3 (2020)..

[11] Fisher, Derek. Application Security Program Handbook. Simon and Schuster, 2023.