ISSN: 2634 - 8853

Journal of Engineering and Applied Sciences Technology

SCIENTIFIC Research and Community

Review Article

Open Access

A Comparative Analysis and Benchmarking of Dynamic Application Security Testing (DAST) Tools

Vivek Somi

Technical Account Manager at Amazon Web Services, USA

ABSTRACT

Cybersecurity is crucial in today's era of advanced technology, rapidly developing scientific understanding, and a completely interconnected global society to guarantee high safety in all aspects of life. Furthermore, there is an ever-increasing number of difficulties and dangers to achieving security in cyberspace. One of the most basic and essential ways to avoid cybersecurity is to conduct security testing for vulnerabilities. In order to make the most of the potential synergies between various types of analysis tools, this paper combines static white box security analysis (SAST), dynamic black box security analysis (DAST), and interactive white box security analysis (IAST) in that order. This investigation aims to improve security vulnerability detection while decreasing false positives.

***Corresponding author**

Vivek Somi, Technical Account Manager at Amazon Web Services, USA.

Received: February 05, 2024; **Accepted:** February 12, 2024; **Published:** February 26, 2024

Keywords: Web Application, Security Vulnerability, Analysis Security Testing, Static Analysis Security Testing, Dynamic Analysis Security Testing, Interactive Analysis Security Testing, Assessment Methodology, False Positive, False Negative, Tools Combination

Introduction

In recent years, web apps have seen a surge in use across a wide range of industries and governmental agencies. In order to stay competitive, these apps need to be built frequently and as quickly as feasible. Consequently, programmers either intentionally include security holes in their code or rely on susceptible third-party modules or components. Sometimes, their financial resources are tight. Due to these instances, they often lose sight of security, a crucial part of the development life cycle.

Furthermore, businesses often discover that engineers need more security understanding, heightening the likelihood of creating unsafe software. When conducting a security analysis, the analyst must choose the most appropriate static (SAST), dynamic (DAST), and interactive (IAST) analysis security testing tools in tandem, bearing in mind the various forms of analysis security testing (AST) and the abundance of security vulnerabilities present in web applications' code and configurations [1]. The examined online apps failed the OWASP Top Ten project, as confirmed by many studies' findings [2-5]. “The most common and harmful vulnerabilities are still SQL injection (SQLI) and cross-site scripting (XSS).” Combining many similar technologies may improve performance in terms of both true and false positives [6-9]. Combining diverse strategies to utilise the synergies of different instruments may improve the ratio of true positives to false positives, according to many studies [10-12]. Both the number of false positives (vulnerabilities discovered when none existed) and the number of false negatives (vulnerabilities found) may be decreased by combining the technologies presented in these works. All security vulnerabilities reported by an AST tool, including those found via human assessments, must be double-checked, according to the reviewed literature. The security analyst can efficiently resolve false positives; thus, they pose no genuine threat. However, if the tool has not seen a false negative before, it can be hard to find-and that might lead to severe problems. Among these methods, you may find tools for static (SAST), dynamic (DAST), and interactive (IAST) white box security analysis. Time and highly trained personnel are needed for manual analysis. In order to conduct a thorough analysis of the security of a web application, it is essential to access all sections and levels of the program, covering the whole attack surface. Additionally, it is required to use technologies that automate security analysis to the maximum extent feasible.

When evaluating fully functional, live software, it is known as Dynamic Application Security evaluating (DAST), a subset of penetration testing. Both automatic and manual methods exist for executing DAST, which aims to identify security flaws in various applications, including web, mobile, IoT, and cloud. DAST is an essential component of software security assessments because it mimics an external attacker by investigating the system without knowing its underlying structure. Automated DAST systems also remove the need for developers or security staff to rely on their own knowledge and skills to mimic attacks and identify exploitable flaws. The likelihood of vulnerabilities in the final product may be reduced if developers adopt a security attitude from the start of the software development life cycle [8]. Before releasing a system to the public or a client, developers may use DAST to identify and fix any security flaws. This lessens the possibility of an assault succeeding and the harm it may do, monetarily and to the reputation of the company. Since every system operates differently, conducting optimum automated dynamic testing of the whole software might be challenging. Automated dynamic testing may still help with system security;

however, since DAST is application-agnostic, a single DAST tool can be modified to conduct security checks on many systems and apps. Web applications that use new techniques, such as AJAX technologies, increase the demands on the tools. "Studying DAST tools' behaviour and vulnerability-finding capabilities is crucial for ensuring high-security software systems and making them more challenging to attack, as these tools have diverse strengths and shortcomings."

Problem Statement

Vulnerabilities in online apps and APIs pose a growing hazard to organisations in the modern digital world. Exploiting these vulnerabilities may result in devastating data breaches, service interruptions, and monetary losses. It is now impossible to find security flaws in an application during runtime without automated Dynamic Application Security Testing (DAST) techniques that do not need access to the source code. Nevertheless, organisations are finding it more challenging to choose the best DAST technology to fulfil their security needs because of the wide variety of options and the quick development of online applications. In order to help organisations make educated choices, it would be helpful if top DAST products were thoroughly analysed and benchmarked. To assist organisations in selecting the best solution for application security, this research compares the leading automated DAST solutions now available and assesses their efficacy, efficiency, and usability.

Literature Review

The most critical types of vulnerabilities are included in the OWASP Top Ten project. The OWASP Top Ten project was not passed by the web apps that were assessed, according to many publications [2-4]. Web applications in organisations and companies connected through the Internet and intranets support various business functions. However, they are also susceptible to a wide range of attacks that aim to gain economic advantage, privileged information, denial of service, extortion, etc., by exploiting vulnerabilities in their design, implementation, or operation. "These vulnerabilities are part of the OWASP Top Ten project. Codes written in the.NET framework (C# or Visual Basic), iOS's Swift, or PHP are just a few examples of the many web programming languages available today". Based on much research, Java is the most often used language [13,14]. These days, many people choose Node.js, Python, and C++. Nowadays, web apps rely on technologies like AJAX, HTML5, flash, and Javascript frameworks like Angular, Vue, React, Jquery, Bootstrap, etc. [15,16]. "Vaadin is a framework for developing collaborative web apps using HTML5 UIs and Java backends. Developers should undergo secure code development training to avoid security holes in web application source code." Using secure languages that verify memory and type at build time is another way to avoid this. Java, C# and Rust are among these languages [1]. Every configuration of navigators, applications, and database servers must adhere to security standards, which designers and developers must follow. Installing a Web Application Firewall is just one piece of the puzzle regarding online security [17-19].

DAST tools are black box analysis tools that can attack all of a web application's external source inputs while running [20]. In the first stage, they attempt to crawl the online application to find all the potential inputs that may be used to attack it. "In addition to automated crawling that incorporates information about the online application, such as programming languages, application servers, database servers, authentication, and session methods, the human crawling phase must use the tool as an intercepting proxy." Following the crawling step, the tools launch a recursive assault on all identified web application source inputs, injecting malicious payloads at each stage. After that, a security vulnerability is checked by syntactically analysing each HTTP response.

Lastly, potential false negatives and false positives must be manually corrected in the vulnerability report. Here, unlike with white box tools, nobody knows where the app's code is hiding. The active web app's user interface is the test's target. In comparison to SAST techniques, DAST methods often find fewer true positives and fewer false positives [10,21]. Vulnerability scanning during software deployment is now possible with the help of DAST tools. In order to get analytical data, it is necessary to mimic the actions of an attacker.

Furthermore, these tools may be run apart from the application's programming language. The DAST tools constantly improve and add new features, such as JWT authentication, attack vectors (XML, JSON, etc.), and vulnerability detection techniques. One of their standout features is fuzzing, which involves testing the application to see if it fails, like changing form entries.

The focus of Kalle Rindell, Karin Bernsmed, and Martin Gilje Jaatun's efforts was software development security risk management as technical debt. Specifically, they identified four main categories of technical debt: requirements, architecture, code, and testing [8]. "Fully Secure (FS), Optimally Secure (OS), and Satisfactorily Secure (SS) are some of the security objectives that Neha Mahendra and Mohammad Muqeem used to structure their work. That got them very close to settling on success criteria" [9]. Fang You-yuan, Gu Tian-yang, and Shi Yinsheng looked at the various software security testing approaches and spoke about how software security testing is classified. The pros and cons of different approaches and the range of their applications are discussed in this paper's conclusion [10]. Richard Amankwah and Patrick Kwaku Kudjo developed a web vulnerability scanner to find vulnerabilities by combining the best features of previous methods [11]. In order to help developers and security test managers, Rajendra Gokhale and Susheel Kumar Sharma highlighted the difficulties associated with web application security testing [12]. Along with others, Atsuo Hazeyama has been working hard to build a knowledge foundation and methodology for safe software development. This is useful for evaluating the efficacy of the testing procedures [22]. Dheerendra Singh, Arunima Jaiswal, and Gaurav Raj catalogued the problems and difficulties associated with security testing. An in-depth examination of the changing difficulties in security testing has touched on topics such as cross-site scripting, SQL injection, cross-site request forgery, and XML injection [23]. Niek To make penetration testing more successful, Jan van den Hout worked on its implementation approaches and proposed a standardised methodology to get the best results [24]. D. Rajya Lakshmi and S. Suguna Mallika compiled a list of web application testing methods and their benefits and drawbacks and published it in [14]. Critical vulnerability to overall vulnerability count ratio estimation was a focus of Devanshu Bhatt's study. He discusses the application's potentially exploitable vulnerability access routes [15]. According to the research of Tosin Daniel Oyetoyan, Bisera Milosheska, Mari Grini, and Daniela Soares Cruzes, a mix of tools may be necessary for a more thorough security assessment when using SAST [16]. In order to show the potential uses of penetration testing, Daniel Dalalana Bertoglio and Avelino Francisco Zorzo conducted comprehensive mapping research [17]. The difficulties, effects, and remedies of SAST methods have been outlined by Jinqiu Yang, Lin Tan, John Peyton,

and Kristofer A. Duer. Developers may benefit more from the suggested methods for using SAST techniques [18]. Working on online testing methodologies, Kamran Ali and Xia Xiaoping categorise web testing into multiple types, each requiring a unique methodology [19]. Several prominent vulnerability scanners have been uncovered and evaluated by Supriya Gupta and Lalitsen Sharma [25]. Regarding software projects, Shafagat Mahmudova spoke about the dangers and how to analyse them to ensure they are secure [26]. Software development lifecycle (SDLC) security testing was the focus of research by Neha Mahendra and Suhel Ahmad Khan, who compiled a comprehensive, organised overview of relevant frameworks, approaches, and methodologies [27]. "Ina Schieferdecker, Juergen Grossmann, and Martin Schneider developed a model-based security testing procedure based on the SUT's architectural and functional models, threat, fault, risk models, and weaknesses and vulnerabilities" [28,29]. Shaikh Abdullah Al-Malaise Al-Ghamdi Security testing techniques were proposed in a survey on software testing methods. "These techniques included code reviews, static analysis, fuzz injection, source and binary code fault injection, risk analysis, vulnerability scanning, and penetration testing. Under perfect circumstances, Vidyabhushan Anantrao Upadhye and Shashank D Joshi were able to determine a vulnerability scanner's capabilities" [30].

Open Source Dast Tools

Open-source and commercial DAST tools abound, each with its own unique set of advantages and disadvantages in terms of architecture, functionality, and performance. This section introduces and describes some open-source tools that were considered for the assessment.

Arachni

One robust Ruby framework for evaluating the safety of online apps is Arachni, which Tasos Laskos first created [31]. Windows, Mac OS X, and Linux users may use the tool's web-based graphical user interface (GUI) in addition to its command line interface (CLI). Because it uses a meta-analysis to assess the findings and features a self-training mechanism during scanning, Arachni is an intelligent tool. "Versatility for complicated applications using technologies like JavaScript, HTML5, DOM manipulation, and AJAX is made possible by its integrated browser environment, which permits analysis of client-side code". This allows the system to handle input vectors that are invisible by conventional scanners effectively and to provide excellent coverage to contemporary online apps. Arachni is well-documented and supports proxy. At the time of writing, Arachni was on the verge of obsolescence; Ecsypno, the business that developed it, created a commercial tool called Codename SCNR, its replacement.

Black Widow/Black Ostrich

The academic proof of concept tool Black Widow was created in 2021 by Eriksson et al. In order to overcome significant obstacles in scanning contemporary online applications, the authors developed a crawler that looks for links and relationships between various components of an application [32]. Eriksson et al. have created its successor, Black Ostrich, which aims to address a significant problem with current crawlers: failing input validation [33]. A new capability it has is the ability to exploit validation regular expression patterns to generate malicious input that manages to evade validation. Due to their exclusive focus on cross-site scripting (XSS) vulnerabilities, Black Widow and Black Ostrich have significant limitations.

Nikto

Open source and written in Perl, Nikto is a web server scanner that supports Windows, MacOS, and Linux. Chris Sullo and David Lodge designed it. The program dynamically scans for common web server vulnerabilities, obsolete versions, and misconfigurations using a signature-based approach [34]. It also looks for these issues. Because it lacks stealth functionality, it will run a web server test as quickly as possible, drawing attention to itself in log files and maybe even an intrusion detection system. In addition to proxy support, Nikto's simple design includes a command line interface (CLI) and a collection of plugins. The tool and its capabilities are also well-documented.

Nuclei

A cyber security firm called Project Discovery created an open-source Nuclei project [35]. Web application vulnerability scanning is only one of its many uses; it can also explore infrastructure, cloud platforms, web servers, and networks for exploitable vulnerabilities and help fix them. Its template-based approach greatly enhances the tool's versatility in handling different testing situations. Users may build and distribute YAML-based templates, which form the tool's basis. Specific security flaws are identified and addressed using the procedures outlined in the templates. These procedures include outlining potential attack paths, identifying the vulnerability, its severity, priority, and, if applicable, associated exploits. A command line interface (CLI) and comprehensive documentation are among Nuclei's many features.

OpenVAS

Greenbone AG's Open Vulnerability Assessment System (OpenVAS) is a C and Rust-based open-source project that evolved from the Nessus project [36]. Among its contents are the network vulnerability scanner OpenVAS and several industrial and Internet protocols at low and high levels. The tool sends and generates the vulnerability detection tests and threat information via a feed. It checks the systems for a number of characteristics and may scan firewalls, switches, and servers for any security holes. These features include operating systems, open ports, program installations, user accounts, file system layouts, and customizations. "The OpenVAS platform offers comprehensive reporting and documentation, a command line interface, and a web-based graphical user interface."

W3af

The Python web security scanner W3af created by Andres Riancho is modular, with two primary sections: the core and the plugins [37]. The plugins count on the core for functionality and for organizing the scan process. Though a host of plugins exists, the main features of W3af are the crawler, audit, and assault. The auditing plugin obtains the injection points and URLs from the crawler plugin to find vulnerabilities through the transmission of custom data. At last, the attack plugin intends to exploit the security vulnerabilities found by the audit plugin. The program provides a command line interface (CLI) and a graphical user interface (GUI). But the tool is not updated in the recent times.

Benchmarking Results

Reviewing several application security tools reveals beneficial information about their usefulness, effectiveness, and suitability for improved software application security [38]. Despite their crucial function in quickly identifying vulnerabilities, SAST tools have specific limitations. Analyses demonstrate that SAST tools can help discover common vulnerabilities but usually generate many false positives. As a result of this problem, teams might

face development delays while they take the time to validate and troubleshoot these reports. In addition, SAST's focus on source code makes it possible to miss runtime vulnerabilities and environmental configuration issues that can only arise during operational use. An analysis of SonarQube and Fortify Static Code Analyzer took place. Results showed that both tools feature extensive detection mechanisms; however, they vary in how easy they are to integrate and their effect on the development cycle. The interface of SonarQube became known for being friendly to developers, combined with a lower rate of false positives compared to Fortify, which is more comprehensive but takes additional time to lower its false positive rate. DAST tools enable an outside perspective on applications that are running, identifying vulnerabilities exploitable in an application when deployed. The finding showed that DAST tools work well at modelling external attacks and identifying runtime issues, but they are limited in their applicability. Being external tools, DASTs might miss vulnerabilities within an application's internal framework, especially those that do not expose themselves through external interfaces. OWASP ZAP and Burp Suite are renowned for their success in simulating attacks. OWASP ZAP received recognition for its simple design and ease of use, which makes it appropriate for inclusion in the development cycle. Because of its extensive analysis features, Burp Suite was better suited to those in the security profession. RASP tools introduce a new method by merging defence with application functionality and instantly reacting to real-time attacks. Investigation results indicated that RASP tools deal effectively with vulnerabilities by monitoring application behaviour and blocking exploitation attempts. The barriers related to performance impact and the notoriously tricky integration of these tools within current application designs hamper the uptake of RASP technology. Contrast Security and Imperva RASP confirmed their significant resources for preventing threats from detected and undetected vulnerabilities. Contrast Security stood out for its methods, which were conducive to developers and subtly affected application performance. While delivering solid defences, Imperva RASP needs more careful tuning and configuration to achieve optimal performance.

Table 1: Comparison of SAST, DAST, and RASP

Feature	SAST	DAST	RASP
Method of Operation	Examines source code without executing it	Evaluates running applications by simulating attacks	Integrates into the application to detect and defend against attacks in real-time
Stage in SDLC	Early in the development cycle	Post-development, preproduction	Production/runtime
Type of Issues Detected	Syntax errors, security flaws such as buffer overflows, SQL injections	Runtime issues, authentication/ authorization errors, session management issues	Real-time attacks, malicious inputs, runtime vulnerabilities
Integration	Integrated into the development process	Part of the broader AST strategy used in staging environments	Embedded within the application, it operates in a production environment
Granularity	Examines code at a granular level	Evaluates the application as a whole	Monitors and protects at runtime, providing contextual insights
Remediation Assistance	Provides early detection and fixing of issues	Identifies issues in a running state, providing context for runtime vulnerabilities	Provides immediate protection and mitigation
Coverage	Source code, configuration files	HTTP requests, responses, session data	Data flow, control flow, internal connection information
False Positives	This can be higher due to a lack of runtime context	Generally lower as it evaluates actual runtime behaviour	Low, as it operates in the actual runtime environment
Advantages	Early detection of wide range of detectable issues	Effective in identifying runtime-specific vulnerabilities	Immediate and continuous protection, context-aware defence
Disadvantages	May miss runtime-specific issues	Requires a running application, potential for environmental dependencies	Overhead on application performance, complexity in integration

Scope

This study concentrates on analyzing and benchmarking open-source DAST tools, such as Arachni, Black Widow, Nikto, OpenVAS, W3af and Nuclei. The range includes assessing tools on principal performance metrics such as the accuracy of vulnerability detection, the occurrence of false positives, integration ease with CI/CD pipelines, usability, reporting capabilities, and compliance assessments. The study will likewise analyse the limitations of DAST tools, such as their incapacity to reveal business logic flaws or vulnerabilities that do not become visible during runtime. To complete a thorough understanding of every tool's function, benchmarking will vary across application types, covering both web applications and APIs. The study aims to deliver organizations practical insights and recommendations for selecting the most suitable DAST tool based on their identified needs and security contexts. This study also provides a comparison of SAST, DAST and RASP testing methodologies.

Conclusion

Overall, this intensive evaluation of application security tools points out the necessity of using a varied strategy to secure software applications in the digital landscape. The paper examines static analysis techniques, dynamic analysis tools, and runtime protection systems to highlight their inherent strengths and limitations. Tools for Static Application Security Testing (SAST) are vital for quick vulnerability detection. However, they have their downsides, notably the high incidence of false positives and the unsuitability for detecting runtime problems. Dynamic Application Security

Testing (DAST) tools deliver helpful information about runtime vulnerabilities, although they do so with the constraint of limited access to the application's internal workings. Runtime Application Self-Protection (RASP) technologies have become a hopeful answer for addressing real-time threat mitigation; however, challenges surrounding integration and performance deserve attention. The research supports a layered security methodology by exploiting the cooperative strengths of SAST, DAST, and RASP tools to form a powerful defence against various cyber threats. This integral approach improves applications' security posture and corresponds with the changing dynamics of cyber risks and the rising complexity of cyber attackers.

References

1. Felderer M, Büchler M, Johns M, Brucker AD, Breu R, et al. (2016) Security Testing: A Survey. In Advances in Computers. Elsevier: Cambridge, MA, USA 101: 1-51.
2. Homaei H, Shahriari HR (2017) Seven Years of Software Vulnerabilities: The Ebb and Flow. IEEE Secur. Priv. Mag 15: 58-65.
3. Barabanov A, Markov A, Tsirlov V (2017) Statistics of software vulnerability detection in certification testing. In International Conference Information Technologies in Business and Industry 2018; IOP Publishing: Tomsk, Russia 1-8.
4. Sołtysik-Piorunkiewicz A, Krysiak M (2020) The Cyber Threats Analysis for Web Applications Security in Industry 4.0. In Towards Industry 4.0—Current Challenges in Information Systems; Studies in Computational Intelligence; Springer: Cham, Switzerland https://link.springer.com/chapter/10.1007/978-3-030-40417-8_8.
5. OWASP Foundation (2023) OWASP Top Ten 2017 https://owasp.org/index.php/Top_10_%202017-Top_10.
6. Algaith A, Nunes P, Fonseca J, Gashi I, Viera M (2018) Finding SQL injection and cross site scripting vulnerabilities with diverse static analysis tools. In Proceedings of the 14th European Dependable Computing Conference, IEEE Computer Society, Iasi, Romania 10-14.
7. Nunes P, Medeiros I, Fonseca JC, Neves N, Correia M, et al. (2018) An empirical study on combining diverse static analysis tools for web security vulnerabilities based on development scenarios. Computing 101: 161-185.
8. Bermejo JR, Bermejo J, Sicilia JA, Cubo J, Nombela JJ (2020) Benchmarking Approach to Compare Web Applications Static Analysis Tools Detecting OWASP Top Ten Security Vulnerabilities. Comput. Mater. Contin 64: 1555-1577.
9. Nunes P, Medeiros I, Fonseca JC, Neves N, Correia M, et al. (2018) Benchmarking Static Analysis Tools for Web Security. IEEE Trans. Reliab 67: 1159-1175.
10. Antunes N, Vieira M (2015) Assessing and Comparing Vulnerability Detection Tools for Web Services: Benchmarking Approach and Examples. IEEE Trans. Serv. Comput 8: 269-283.
11. Monga M, Paleari R, Passerini E (2009) A hybrid analysis framework for detecting web application vulnerabilities. In Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems, Vancouver, BC, Canada 25-32.
12. Higuera JB, Aramburu CA, Higuera JRB, Sicilia MA, Montalvo JAS (2020) Systematic Approach to Malware Analysis (SAMA). Appl. Sci 10: 1360.
13. Nanz S, Furia CA (2015) A comparative study of programming languages in rosetta code. In Proceedings of the 37th International Conference on Software Engineering 2015, Florence 1: 16-24.
14. Aruoba SB, Fernández-Villaverde J (2015) A comparison of programming languages in macroeconomics. J. Econ. Dyn. Control 58: 265-273.
15. Beasley RE (2020) Ajax Programming. In Essential ASP. NET Web Forms Development; Apress: Berkeley, CA, USA https://link.springer.com/book/10.1007/978-1-4842-5784-5.
16. Moeller JP (2016) Security for Web Developers: Using Javascript, HTML and CSS; O'Reilly Media: Sebastopol, Russia https://www.abebooks.com/9781491928646/Security-Web-Developers-Using-JavaScript-1491928646/plp.
17. Razzaq A, Hur A, Shahbaz S, Masood M, Ahmad HF, et al. (2013) Critical analysis on web application firewall solutions. In Proceedings of the 2013 IEEE Eleventh International Symposium on Autonomous Decentralized Systems (ISADS), Mexico City, Mexico 1-6.
18. Holm H, Ekstedt M (2013) Estimates on the effectiveness of web application firewalls against targeted attacks. Inf. Manag. Comput. Secur 21: 250-265.
19. Tekerek A, Bay O (2019) Design and implementation of an artificial intelligence-based web application firewall model. Neural Netw. World 29: 189-206.
20. Mirjalili M, Nowroozi A, Alidoosti M (2014) A survey on web penetration test. Adv. Comput. Sci 3: 107-121.
21. Vega EAAA, Orozco LS, Villalba LJG (2017) Benchmarking of Pentesting Tools. Int. J. Comput. Electr. Autom. Control Inf. Eng 11: 602-605.
22. Mohino JDV, Higuera JB, Higuera J-RB, Montalvo JAS, Higuera B, et al. (2019) The Application of a New Secure Software Development Life Cycle (S-SDLC) with Agile Methodologies. Electronics 8: 1218.
23. OWASP Foundation (2023) OWASP Benchmark Project https://www.owasp.org/index.php/ Benchmark.
24. Nanz S, Furia CA (2015) A comparative study of programming languages in rosetta code. In Proceedings of the 37th International Conference on Software Engineering 2015, Florence, Italy 1: 16-24.
25. OWASP Foundation (2023) OWASP Testing Guide, 2020. Available online: https://owasp.org/www-project-websecurity-testing-guide/.
26. Huth M, Nielsen F (2019) Static Analysis for Proactive Security. Computing and Software Science. Lecture Notes in Computer Science; Springer: Cham, Switzerland 374-392.
27. Al-Amin S, Ajmeri N, Du H, Berglund EZ, Singh MP (2018) Toward effective adoption of secure software development practices. Simul. Model. Pr. Theory 85: 33-46.
28. Sipser M (2006) Introduction to the Theory of Computation, 2nd ed.; Thomson Course Technology: Boston, MA, USA.
29. Singh D, Sekar VR, Stolee KT, Johnson B (2017) Evaluating How Static Analysis Tools Can Reduce Code Review Effort. In Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, Raleigh, NC, USA 11-14.
30. Yang J, Tan L, Peyton J, Duer KA (2019) Towards better utilizing static application security testing. In Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, Montreal, QC, Canada 25-31.
31. Tasos Laskos (2023) Arachni - Web Application Security Scanner Framework https:// github.com/Arachni/arachni.
32. Benjamin Eriksson, Giancarlo Pellegrino, Andrei Sabelfeld (2021) Black Widow: Blackbox Data-driven Web Scanning. In: 2021 IEEE Symposium on Security and Privacy (SP) 1125-1142.
33. Benjamin Eriksson, Amanda Stjerna, Riccardo De Masellis,

Philipp Rüemmer, Andrei Sabelfeld (2023) Black Ostrich: Web Application Scanning with String Solvers. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. CCS '23. Association for Computing Machinery 549-563.

34. Chris Sullo, David Lodge (2023) Nikto 2.5 https://cirt.net/Nikto2.
35. Project Discovery (2023) Nuclei https://github.com/projectdiscovery/nuclei
36. Greenbone (2023) Greenbone OpenVAS https://www.openvas.org/.
37. Andres Riancho (2023) W3af https://github.com/andresriancho/w3af.
38. Seth A (2022) Comparing effectiveness and efficiency of interactive application security testing (IAST) and runtime application self-protection (RASP) tools. North Carolina State University 31.