Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2023, 10(8):158-165**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Comparative Analysis of DAST, SAST, and IAST: A Comprehensive Study on Large-Scale Web Applications

Vivek Somi

Technical Account Manager at Amazon Web Services

Abstract: With large web applications, encountering an increasing number of threats, application security has become progressively important. Security issues including SQL injections, cross-site scripting (XSS), and data leaks seriously impact organizations. In response to these threats, different application security testing techniques, including Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), and Interactive Application Security Testing (IAST), have come into being. They deal with weaknesses at diverse points throughout the Software Development Lifecycle (SDLC) and vary in effectiveness, ease of implementation, and detailedness. This work offers an in-depth comparison between SAST, DAST, and IAST, zeroing in on their relevance to large web applications. We investigate their capabilities, restraints, and adaptability within today's Continuous Integration and Continuous Deployment (CI/CD) chains. With an analysis of the use cases, scope, and impact of both, this paper serves to offer an understanding of how organizations can improve application security testing for greater performance and risk mitigation.

Keywords: DAST, SAST, IAST, application security, web application security, CI/CD integration, SDLC, vulnerability testing

1. Introduction

As the digital world interface gets more complex web applications are left open to attacks. Since web applications, immense scale platforms such as online banking, e-commerce websites and social networking sites contain user information and cater for millions of users, they are easily attacked. Yet, it is found that 43% of cyber-attacks are aimed at web applications and it cost $ 4.45 million on average to have a data breach in 2023 based on IBM's Cost of a Data Breach Report. Large-scale web applications are more challenging to develop and maintain, which adds to the risk of creating exploits like SQL injections, cross-site scripting (XSS), and misconfigured security systems. To solve these problems, organizations need to implement sound approaches to security testing that would help uncover threats before they are exploited.

Three primary types to detect vulnerabilities are Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), and Interactive Application Security Testing (IAST). While SAST interprets the source code and provides no results of the execution; DAST, on the other hand, explores it and identifies several opportunities; IAST is a blend of both SAST and DAST. Despite their differing approaches, these testing methods share a common goal: protection of web applications from prospective risks. The objective of this paper is to serve as an informative source of information that will compare SAST, DAST, and IAST, with special emphasis on Large Scale Web Applications. We assess the effectiveness of both options, discuss their advantages, disadvantages, and applicability, as well as analyze their integration with CI/CD to achieve the best security results.

2. Main Body

Problem Statement

Businesses cannot underestimate the benefits of Web applications, they are essential; however, it must be noted that such applications are always attacked. One of the problems arising during web application security is the choice of proper methods of its vulnerability detection during development. As the applications scale up to the web area, the possible attack points are more, and the bandwidth becomes broader [1]. The risks may have their source in poorly coded code, in unknown and unverified third-party components, or in incorrect settings. As per the OWASP Top 10 list, applications face issues like broken access control, injection flaws, and security misconfigured [2]. Static and dynamic security testing methodologies as well as other traditional black-box testing techniques, can simply not match the current rapidity and intricate structure of Web application development. The practices of continuous integration and delivery demand security testing tools that do not hinder the speed of integration into the tools and processes used [3]. Nevertheless, several organizations continue to struggle to apply security, through appropriate CI/CD processes, hence, slow identification of risks and security weaknesses within production triplets.

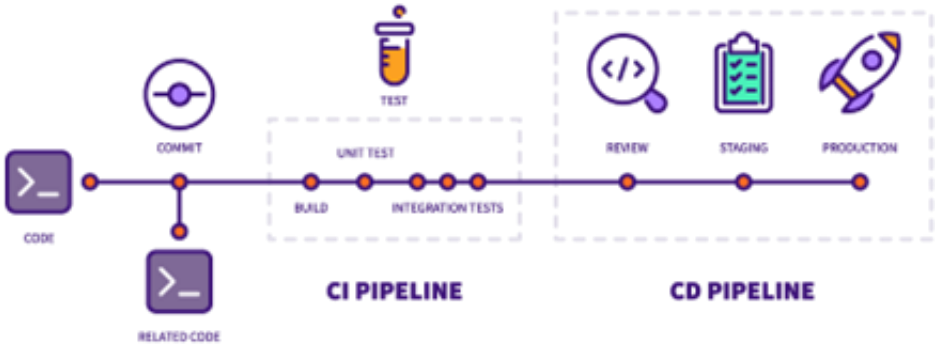

Figure 1: *Continuous Integration, Delivery, and Deployment Process*

Moreover, none of the security testing methods can be fully effective throughout all phases of application development SDLC. For instance, SAST can detect errors in the code that writes but may not detect those errors that appear when the program is running [4]. DAST is indeed good at identifying runtime failures, but it fails to identify code-based vulnerabilities. IAST tries to fill these gaps, but it is not devoid of problems associated with performance losses and system configuration. It is for this reason this paper seeks to identify the testing methodologies that should be used to support the other methods in various scenarios to enhance vulnerability testing and mitigation [5]. In this paper, an attempt will be made to understand how SAST, DAST, and IAST can help to overcome these challenges in large-scale web applications.

3. Solution

Static Application Security Testing (SAST)

Static analysis is a White-Box testing technique that implies using an application's source, object or Byte code to help identify vulnerabilities. It is normally conducted at the initial part of the SDLC, preferably during the development or build phase. SAST is better than DAST because to use it you do not need to run any application and thus it identifies flaws such as SQL injection, cross-site scripting (XSS) and buffer overflow in the source code before the application is run [6]. Veracode's study demonstrates that applications tested under SAST have an average of 56% lower density of vulnerability compared to applications that were not tested with SAST [7]. Such reduction highlights the need and significance of considering security issues as early as the system development life cycle.

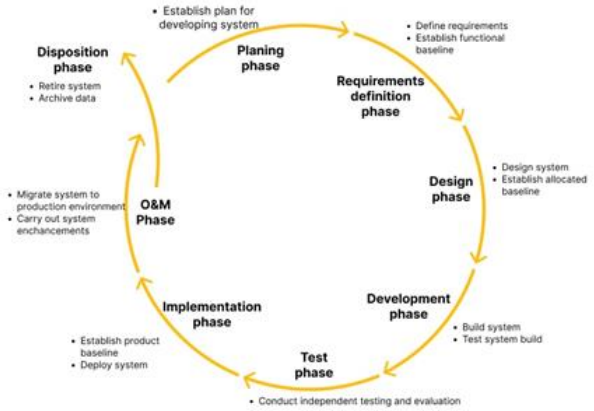

Figure 2: *System Development Life Cycle*

Static Application Security Testing (SAST) has unique benefits for the identification of vulnerabilities at the initial stages of the software development life cycle. However, because SAST runs by inspecting the source code without running the application, it is very helpful in identifying security flaws within the early stages of software development [8]. This is important for dropping the costs of correcting security problems, which are often higher in the later stages of a product's lifecycle than during the development phase. SAST gives first-rate breadth analysis, which means all the possible code paths are checked within a program [9]. Here, such strengths as good coverage of input validation flaws, insecure handling of cryptographic data, and hard-coded credentials are identified. Nevertheless, SAST is the perfect compatibility of the approach with contemporary development platforms, such as IDEs and CI/CD [10]. This integration allows performing security testing with each push of the code-to-code repository, thus strengthening the security from the development phase.

Figure 3: *Static Application Security System*

One of the biggest concerns of the tool is that it has many false positives. The Synopsys report reveals that SAST tools can produce up to 70% false positives when they send out an alert [11, 12]. This large number of alerts can result in alert fatigue whereby in their pursuit to ignore the numerous fake alarms they may end up missing real vulnerabilities. Moreover, SAST has no runtime information, and it cannot identify a range of issues that is, such as memory leaks or inept authentication settings [13]. There are two major drawbacks of full codebase scans: time consumption, which increases with the increased size of a program and its use of extensions; The long scan times are quite a drawback in this form of development since this might slow down development which is quite contrary to the agile development system [14]. Nevertheless, there are major challenges, letting SAST stay as an important layer in security testing of early stages.

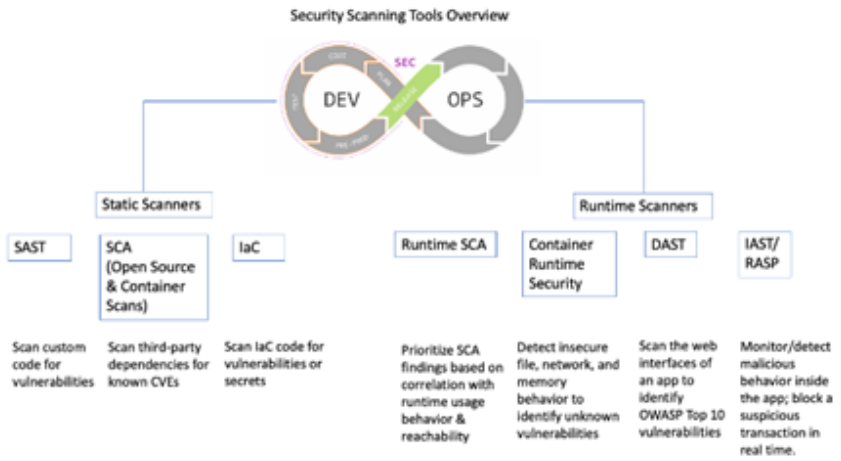

Figure 4: *Security Scanning Tools and Processes*

4. Dynamic Application Security Testing (DAST)

DAST- Dynamic Application Security Testing- is a type of black-box testing where the objective is to check security flaws in an active application through its interaction with the external interfaces. DAST, unlike SAAS (Static Application Security Testing), does not require the target application's source code; this makes DAST suitable for testing third-party or externally accessible parts of an application [15]. This approach proves to be more beneficial in the identification of runtime vulnerabilities such as; SQL injections, cross-site scripting (XSS) and broken authentication schemes.

The major benefits linked with DAST include; The possibility of identifying issues which are actual only when the application is functioning. This real-time analysis highlights the areas of vulnerability that such traditional testing methodologies may not be able to identify. For example, when compared to other methodologies such as DAST it can provide an understanding of how the data is processed through external interfaces and helps one to get a better view of how this particular application behaves in conditions of real-life usage [16]. Second, DAST does not require access to the source code since it tests for the presence of vulnerabilities, and this is another benefit – one can employ this type of testing if the source code is unavailable, for example, when testing third-party components, or microservices architecture. This flexibility makes it a prime weapon in today's flexible, componentized app environments [17]. DAST is also preferable to SAST because it provides fewer incidents of false positives. This is in contrast with AST which tests an existing application against a fake copy since DAST communicates with a live application then the vulnerabilities revealed are real security threats. For instance, DAST would only report cross-site scripting (XSS) vulnerability after demonstrating the ability to inject a script and watch how it is executed, which makes the type of false alarm noise produced by automated security testing [18].

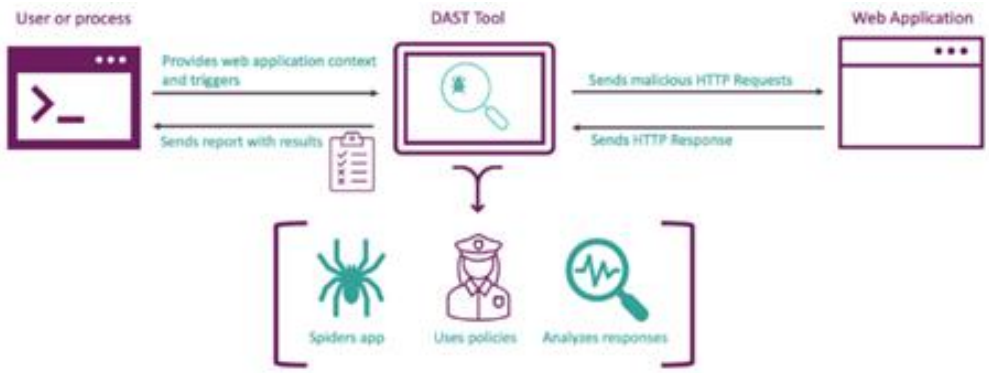

Figure 5: *Dynamic Application Security Testing*

However, it is still necessary to note that DAST also has some disadvantages. The major disadvantage is that it identifies a small percentage of all the code in the code base. Again, what DAST can do is check black-box pieces of the application that it can interact with, usually executable components such as APIs and inputs [19]. This means that failsafe internal keys like business logic or back-end processing could be kept from running. Therefore, DAST should be utilized in combination with such testing approaches as SAST, which can identify a range of risks across the whole codebase. The problem with DAST is that it is difficult to determine the time when vulnerabilities are exposed. DAST works based on having a fully functional and running application which is why the problems are observed later in SDLC and the cost of fixing them is high [20]. While SAST is capable of detecting problems at this stage, it is done in advance, and developers can solve problems before the application is released.

Figure 6: *DAST Working Process*

Some aspects of DAST tools need tuning to generate positive outcomes. This is a common problem with any complex application because most automated scans need to be configured properly to identify all the vulnerabilities [21]. For instance, some applications may need particular recommendations for the authentication particularities or the handling of the sessions, all these aspects need to be taken into consideration during the scan. By far, DAST is an effective functional technique for vulnerability probing in online applications. It still has its drawbacks, such as lower code coverage and later detection, but the ability to find real vulnerabilities is critical for overall security when used in combination with the SAST method.

5. Interactive Application Security Testing (IAST)

IAST is the combination of the best aspects of both SAST and DAST, as it analyses the applications when they are running, but at the same time provides a detailed overview of their security conditions. IAST in this context deploys sensors or agents to gather data from the two contexts – application source and execution [22]. Another advantage of IAST is that it provides constant feedback on vulnerabilities. During development when the application is run and tested the IAST is constantly reviewing how the code behaves with the externally sourced inputs and immediately after a vulnerability is unmasked an alert is given [23]. Real-time feedback, or insects, makes it much faster to track down and address these weaknesses, freeing up programmers from time-consuming security troubleshooting. IAST tools help to lower the time needed to find issues to a maximum of 30% if compared to standalone testing tools and that's why they are perfect for today's rapid development circles [24].

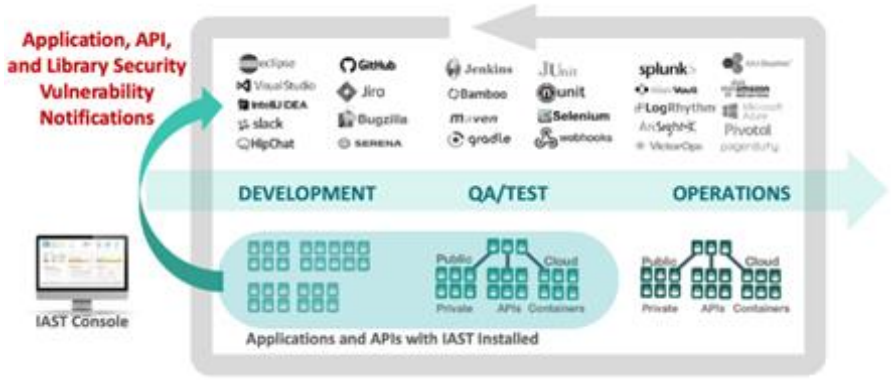

***Figure 7**: Interactive Application Security Testing*

This makes IAST complement well CI/CD pipelines which are commonplace in agile environments where developers often rewrite, and fresh features are deployed. IAST Secures Software by Identifying Them Early IAST integrates security testing into a development cycle, meaning that no security bugs ever get to production-level code [25]. A problem of the proposed approach is based on the fact that the use of agents or sensors can introduce a performance overhead when applications are busy with traffic. Secondly, the implementation and configuration of IAST tools may also be quite challenging especially when dealing with large code bases. Last but not least, IAST strictly relies on its agents, and even minor misconfiguration can result in either partial or erroneous testing outcomes.

6. Uses, Impact, And Scope

DAST, SAST and IAST have their inherently important tasks to perform to safeguard large-scale web applications throughout the development phase. First-generation instruments, such as Static Application Security Testing (SAST), are most effective in the earliest stages of the SDLC when the code repository is most dynamic [26]. Static Application Security Testing (SAST) makes it possible for developers to find code-based weaknesses, such as buffer overflows and input validation errors, before deployment thus saving a developer a lot of money in terms of fixing a security bug as estimated by IBM, to be as high as 75 percent [27]. SAST tools are most used with the capability of scanning complete source code and it is most efficient in organisations where operational teams are allowed full access to the code base. DAST, therefore, is best carried out in a pre-production or pre-staging environment where the live application test may be done. It effectively recognises runtime vulnerabilities such as SQL injections and lacking authentication during runtime thus helping in the minimization of risks posed by external threats [2, 26]. However, the disadvantage of DAST is its inability to navigate through the inner pathways of the code or otherwise stay as being optimal for assessing the outer parts of an application.

IAST kind of bridges between SAST and DAST by providing feedback about many of both the static and dynamic code vulnerabilities in real-time throughout the SDLC. IAST, integrated into the application, works very well with the CI/CD pipeline, reporting on security issues in real-time as the code is written. The real-time notification helps the teams quickly remediate most of the vulnerabilities, meaning that security issues do not make it to the production level. IAST is the most comprehensive technique, but it is efficiency widely depends on how it is embodied and used correctly to scan the maximum amount of code and runtime environments.

7. Conclusion

In today's fast evolutionary world of the development of web applications, the security of the system is always a major concern. Specifically, web applications with many users and high availability are exposed to a growing number of complex threats, so the selection of the appropriate security testing approaches is critical. SAST, DAST and IAST are good tools but have some demerits and the merits or effectiveness of these methodologies mainly depends on the type of environment in which have been applied.

SAST is particularly useful for the kind of code analysis applied at the initial stage of a software development project, and it might be an effective way to detect vulnerabilities that have not yet been included in production

environments. They found out that DAST is important to find runtime vulnerabilities but it is more effective when in combination with other testing techniques. IAST gives the best coverage and is equipped to give feedback as often as possible during the phases of the SDLC.

Presumably, the best way of securing large web applications is to implement all three methodologies of application security although wearing different layers. In the same manner, new methods such as Feedback-Based Application Security Testing (FAST) hold for the future possibility of improving the vulnerability spotlights, especially in complicated application structures. Integrated security at the source, during the development process, in the application and during the final stages can be used to develop strong security prospects that greatly minimise chances of data violation and security of sensitive details.

References

[1]. S. Velliangiri and H. M. Pandey, "Fuzzy-Taylor-elephant herd optimization inspired Deep Belief Network for DDoS attack detection and comparison with state-of-the-arts algorithms," Future Generation Computer Systems, vol. 110, pp. 80–90, Sep. 2020, doi: https://doi.org/10.1016/j.future.2020.03.049.

[2]. OWASP, "A01 Broken Access Control - OWASP Top 10:2021," owasp.org, 2021. https://owasp.org/Top10/A01_2021-Broken_Access_Control/ (accessed Jan. 15, 2023).

[3]. S. Mojtaba, M. Ali Babar, and L. Zhu, "'Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices.,'" Ieee.org, 2020. https://ieeexplore.ieee.org/iel7/6287639/6514899/07884954.pdf (accessed Feb. 12, 2023).

[4]. J. Schmitt, "SAST vs DAST: what they are and when to use them," CircleCI, Apr. 15, 2022. https://circleci.com/blog/sast-vs-dast-when-to-use-them/ (accessed Apr. 05, 2023).

[5]. m3ntat, "'System.OutOfMemoryException' was thrown when there is still plenty of memory free," Stack Overflow, 2024. https://stackoverflow.com/questions/1153702/system-outofmemoryexception-was-thrown-when-there-is-still-plenty-of-memory-fr (accessed Apr. 05, 2023).

[6]. M. Shen, B. Yuan, A. Pillai, X. Zhang, J. Davis, and A. Machiry, "Usage and Effectiveness of Static Analysis in Open-Source Embedded Software: CodeQL Finds Hundreds of Defects," 2020. Accessed: May 10, 2023. [Online]. Available: https://davisjam.github.io/files/publications/ShenYuanPillaiZhangDavisMachiry-CodeQLEMBOSS-arxiv.pdf

[7]. A. Nguyen-Duc, M. V. Do, Q. Luong Hong, K. Nguyen Khac, and A. Nguyen Quang, "On the adoption of static analysis for software security assessment–A case study of an open-source e-government project," Computers & Security, vol. 111, p. 102470, Dec. 2021, doi: https://doi.org/10.1016/j.cose.2021.102470.

[8]. LucaCompagna, "Developers, beware of the tarpits for SAST in your code," SAP Community, May 23, 2022. https://community.sap.com/t5/application-development-blog-posts/developers-beware-of-the-tarpits-for-sast-in-your-code/ba-p/13528505 (accessed May 13, 2023).

[9]. T. Mohammad and P. Sainio, "A Framework of DevSecOps for Software Development Teams," 2023. Accessed: Jun. 27, 2023. [Online]. Available: https://www.utupub.fi/bitstream/handle/10024/175669/Sapkota_Dinesh_Thesis.pdf?sequence=1

[10]. S. Elder et al., "An empirical case study comparing vulnerability detection techniques on a Java application," 2022. Accessed: May 13, 2023. [Online]. Available: https://arxiv.org/pdf/2208.01595

[11]. J. R. Bermejo Higuera, J. Bermejo Higuera, J. A. Sicilia Montalvo, J. Cubo Villalba, and J. José Nombela Pérez, "Benchmarking Approach to Compare Web Applications Static Analysis Tools Detecting OWASP Top Ten Security Vulnerabilities," Computers, Materials & Continua, vol. 64, no. 3, pp. 1555–1577, 2020, doi: https://doi.org/10.32604/cmc.2020.010885.

[12]. H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A Survey on Ethereum Systems Security," ACM Computing Surveys, vol. 53, no. 3, pp. 1–43, Jul. 2020, doi: https://doi.org/10.1145/3391195.

[13]. P. Thomson, "Static Analysis: An Introduction," Queue, vol. 19, no. 4, pp. 29–41, Aug. 2021, doi: https://doi.org/10.1145/3487019.3487021.

[14]. T. Rangnau, R. v. Buijtenen, F. Fransen, and F. Turkmen, "Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines," 2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC), Oct. 2020, doi: https://doi.org/10.1109/edoc49727.2020.00026.

[15]. Amazon Web Services, "What is an API? - API Beginner's Guide - AWS," Amazon Web Services, Inc., 2022. https://aws.amazon.com/what-is/api/

[16]. M. Casanova Páez, S. Pau, D. Canto, R. Victor, and G. Font, "Application Security Testing Tools Study and Proposal," 2020. Accessed: Jun. 02, 2023. [Online]. Available: https://openaccess.uoc.edu/bitstream/10609/126750/1/mcasanovapaezTFM0121.pdf

[17]. F. Mateo Tudela, J.-R. Bermejo Higuera, J. Bermejo Higuera, J.-A. Sicilia Montalvo, and M. I. Argyros, "On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications," Applied Sciences, vol. 10, no. 24, p. 9119, Dec. 2020, doi: https://doi.org/10.3390/app10249119.

[18]. A. Gupta, "An Integrated Framework for DevSecOps Adoption," International Journal of Computer Trends and Technology, vol. 70, no. 6, pp. 19–23, Jul. 2022, doi: https://doi.org/10.14445/22312803/ijctt-v70i6p102.

[19]. S. Mukherjee, S. Roy, and R. Bose, "Defining an appropriate trade-off to overcome the challenges and limitations in Software Security Testing," Journal of Xidian University, vol. 14, no. 7, Jul. 2020, doi: https://doi.org/10.37896/jxu14.7/166.

[20]. Z. A. Maher, "Challenges and limitations in secure software development adoption - A qualitative analysis in Malaysian software industry prospect," Indian Journal of Science and Technology, vol. 13, no. 26, pp. 2601–2608, Jul. 2020, doi: https://doi.org/10.17485/ijst/v13i26.848.

[21]. A. Munoz-Arcentales, S. López-Pernas, J. Conde, Á. Alonso, J. Salvachúa, and J. J. Hierro, "Enabling Context-Aware Data Analytics in Smart Environments: An Open Source Reference Implementation," Sensors, vol. 21, no. 21, p. 7095, Oct. 2021, doi: https://doi.org/10.3390/s21217095.

[22]. Josu Díaz-de-Arcaya, Raúl Miñón, A. I. Torre-Bastida, Javier Del Ser, and A. Almeida, "PADL: A Modeling and Deployment Language for Advanced Analytical Services," Sensors, vol. 20, no. 23, pp. 6712–6712, Nov. 2020, doi: https://doi.org/10.3390/s20236712.

[23]. OWASP, "4.0 Testing Guide," 2014. Available: https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Testing_Guide_v4.pdf

[24]. A. Alti, A. Lakehal, and P. Roose, "A decentralizedagent-basedsemantic service control andself-adaptationin smart health mobile applications," Concurrency and Computation: Practice and Experience, vol. 34, no. 5, Nov. 2021, doi: https://doi.org/10.1002/cpe.6697.

[25]. R. A. Correa, J. Ramon Bermejo Higuera, J. Bermejo Higuera, J. Antonio SiciliaMontalvo, M. Sanchez Rubio, and A. A. Magre´n˜an, "Hybrid Security AssessmentMethodology forWeb Applications," Computer Modeling in Engineering & Sciences, vol. 126, no. 1, pp. 89–124, 2021, doi: https://doi.org/10.32604/cmes.2021.010700.

[26]. M. Monaghan, "Comparative analysis of Static application security testing (SAST) and Dynamic application security testing (DAST) by using open-source web application penetration testing tools MSc Internship MSc in Cyber Security Lyubka Dencheva Student ID: x20195907 School of Computing National College of Ireland," 2022. Accessed: Mar. 17, 2023. [Online]. Available: https://norma.ncirl.ie/5956/1/lyubkadencheva.pdf

[27]. L. Noonan, "5 Damaging Consequences of a Data Breach," MetaCompliance, Feb. 25, 2020. https://www.metacompliance.com/blog/data-breaches/5-damaging-consequences-of-a-data-breach (accessed Apr. 14, 2023).