Date of publication MAR–31, 2025, date of current version FEB–10, 2025.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727–6209
Online ISSN 2312–5381
DOI 10.47839/ijc.24.1.3888

Quantitative WEB Application Vulnerability Assessment using SAST Methodology

ANASTASIIA BRYHYNETS, HALYNA HAIDUR, SERGII GAKHOV, VITALII MARCHENKO
Department of Information and Cyber Security, State University of Information and Communication Technologies, Solomianska 7, Kyiv, Ukraine

Corresponding author: Anastasiia Bryhynets (e–mail: anastasiyka.br@gmail.com).

ABSTRACT This paper presents a study on Static Application Security Testing (SAST) with a focus on the Snyk Code tool. SAST enables early detection and remediation of security vulnerabilities during software development, improving overall system security. The research introduces the General Application Vulnerability Rate (GAVR) model, which quantifies vulnerability risks based on the CVSS 3.1 framework. A case study using Snyk Code demonstrates the identification and assessment of security flaws, such as XSS and certificate validation issues. The study highlights the need for an integrated approach to security testing, emphasizing automation and structured vulnerability assessment to enhance software security.

The GAVR model enhances traditional security evaluations by incorporating exploitability probabilities, offering a more dynamic risk assessment. The findings suggest that integrating SAST within the software development lifecycle significantly reduces security risks and improves remediation efficiency. By leveraging automation and systematic vulnerability quantification, this study underscores the importance of proactive security strategies to safeguard web applications against evolving threats.

KEYWORDS SAST; web application vulnerabilities; CVSS v.3.1; general application vulnerability score, cybersecurity.

I. INTRODUCTION

STATIC Application Security Testing (SAST) is a set of testing methods and techniques in which the component or system under test is not run (not executed) [7]. In essence, SAST is used as a 'white box' testing method that requires access to and understanding of the underlying code base and programming language.

SAST is designed to identify potential security vulnerabilities that may be present in the code. It does this by checking the code against a set of pre–defined rules or patterns associated with known security vulnerabilities. Some of the common security issues that SAST can detect are buffer overflows, insecure encryption algorithms and unhandled exceptions.

SAST testing allows the identification vulnerabilities in the source code of an application at an early stage of development. This is because it can be performed at the very beginning of the SDLC, a software development life cycle, a structure used in project management to describe the stages and tasks associated with each step of development. The SDLC model reflects the entire software development process from initial planning to maintenance and finally to retirement and replacement of the completed application [1]. This means: SAST allows developers to identify and fix security problems before the code is compiled. This saves time and effort and significantly reduces the costs associated with fixing security problems later.

II. PROBLEM DEFINITION

In the software development process, especially in the context of today's dynamic digital environment, early identification and mitigation of potential vulnerabilities is critical to ensuring a high level of web application security. While source code analysis using SAST techniques has proven effective, it faces a number of challenges, including the quantification and prioritisation of vulnerabilities.

According to a study conducted by Verizon, web applications are the target of 26% [2] of all cyber-attacks, making them the second most common category of attack in cyberspace. With approximately 58% of the world's organisations working remotely [3], there is a need to ensure that reliable security measures are in place to minimise the potential threats posed by remote access to corporate resources.

The primary goal of protecting against web threats is to reduce the risk of data breaches, which can have catastrophic consequences for organizations. The average cost of a data breach is estimated at $4.24 million [3], highlighting the financial impact of inadequate cyber security. However, organisations that implement improved security processes and

architecture can significantly reduce potential losses, demonstrating the importance of a proactive approach to information security.

III. SCIENTIFIC PUBLICATIONS ANALYSIS

Discussion of this topic in academic articles focuses on integrating security practices, such as SAST, into the software development lifecycle (SDLC) to proactively identify and remediate vulnerabilities. This focus correlates with the broader theme of improving web application security through early detection.

Paper [10] highlighted the importance of automating security measures within DevOps and DevSecOps methodologies to ensure the efficiency and security of software development and deployment processes. The author emphasized the usefulness of SAST in automating code quality checks to identify security vulnerabilities early in the development process. However, the author also focused on automation through Dynamic Application Security Testing (DAST), container scanning, software composition analysis and automated vulnerability scanning, presenting a comprehensive approach to protecting web applications from potential threats.

According to [12], an integrated approach to application security facilitated by automation reflects the need for preventive measures in vulnerability detection. It illustrates how automated tools, including SAST, are used not only to identify vulnerabilities but also to enforce security policies, thereby reducing the risk of human error and ensuring that critical security tasks such as regression testing are performed consistently.

In [15], a detailed overview of SAST was given with the explanation of its work as a 'white box' testing method that checks the source code of an application for security vulnerabilities. The author noted that the intervention early in the development lifecycle provides real–time feedback to developers, allowing them to identify vulnerabilities a priori before they escalate. The article highlighted the usefulness of SAST in quickly and efficiently scanning the entire code base of an application, as well as its ability to detect key vulnerabilities such as SQL injections and cross–site scripting, thereby improving code quality.

As discussed in [10] and [12], integrating SAST into the SDLC provides a proactive approach to security through automation and the use of comprehensive scanning tools such as SAST. This underscores the importance of using SAST–class solutions to develop secure Web applications by identifying and remediating vulnerabilities early in the lifecycle. This strategy not only increases the security of web applications, but also complies with industry standards and requirements, highlighting the importance of early automation of security processes in today's digital environment.

IV. THE CONCEPT OF SAST

In today’s landscape, automation plays a crucial role in various development activities, ranging from ensuring code quality and performing unit testing to managing deployments. Automating testing and deployment can bring many advantages to software teams, such as faster and more frequent delivery, higher quality and reliability, lower cost and resource consumption, and better collaboration and communication. Automation can speed up the testing and deployment cycles, allowing software teams to release software more often with fewer errors or bugs.

Thanks to the use of the SAST method, it is possible to conduct in–depth code analysis [11]. This is done as follows: the solution scans the entire code base, including the libraries and frameworks used, to identify any potential security flaws. This provides comprehensive coverage of all software components, including third–party libraries and packages.

With the advent of DevSecOps [12], security has become an integral part of the development lifecycle, and SAST is a good tool in this paradigm. Automating scanning with the SAST method will allow the development of secure software by ensuring security in the early cycles of application development.

In modern software development life cycle (SDLC), the security of an application is as important as its functionality [15, 16]. SAST helps development teams do this in several ways: the integration of SAST into the Software Development Life Cycle represents a pivotal shift in traditional development paradigms—a strategy known as "shift left." Basically, it assumes that the software development team may find bugs faster if they test their code as it is being written, rather than waiting until the end of the project [9]. This approach encourages the identification and mitigation of security vulnerabilities early in the development process. By embedding security measures from the onset, development teams can significantly reduce both the costs and the effort required for later remediation. This proactive detection and resolution not only streamline the development process but also enhance the overall quality and security of the software.

Further deepening its impact, SAST tools analyze the entire code base faster than manual review in a detailed way. This thorough investigation extends beyond mere surface checks, delving into complex code bases to uncover hidden or subtle patterns that might pose significant security risks. By identifying these vulnerabilities early in the development stages, SAST tools help prevent the progression of potentially exploitable code into later stages of production, thereby safeguarding the application from future security threats.

In addition to enhancing security protocols, SAST empowers developers directly. In the era of DevSecOps [27], where security is increasingly integrated with development and operational practices, SAST plays a crucial role. It equips developers with the necessary tools and knowledge to embed security into the very fabric of the application from the beginning. This empowerment enables developers to take ownership of the security aspects of their code, making security a fundamental component of the application development lifecycle rather than an afterthought. As expressed in a key observation, the developer analyzes a warning from a SAST solution and investigates the essence of the security defect to fix it. The tool documentation helps with this. Thus, the developer becomes more experienced in information security. This process not only mitigates risks early but also enhances the security skills of developers, integrating learning with development practices.

Lastly, the use of SAST, in conjunction with other security tools, facilitates a comprehensive protection strategy that extends across the entire software ecosystem [13, 14]. For example, combining both SAST and DAST offers a more holistic approach to security testing that enables continuous feedback across different stages of the development lifecycle. This holistic approach ensures that not only the application itself but all interconnected systems and software components are secured. By integrating SAST and other security

methodologies, teams can build a robust security posture that effectively shields the entire digital infrastructure from potential threats. This extensive coverage is crucial in today's interconnected and interdependent technological landscape, where vulnerabilities in one system can potentially compromise the integrity of others.

SAST scan unfolds through a meticulously structured sequence designed to scrutinize and methodically organize the security aspects of application code. Initially, the process commences with the comprehensive analysis of the code base, during which the entire source code of the application is examined to ascertain its structural and functional attributes. Subsequently, leveraging a robust database encompassing known vulnerability patterns, the scanning tool meticulously searches for the analyzed code to pinpoint patterns that may indicate potential security threats. Following the identification of possible vulnerabilities, these issues are systematically ranked or prioritized. This prioritization process is crucial and is based on factors such as the severity of the vulnerability, its potential impact on the system, and the ease with which it can be exploited. The final stage involves a feedback loop where developers are provided with real–time feedback on the vulnerabilities identified. This prompt feedback mechanism is essential as it facilitates immediate corrective actions, thereby enabling more rapid and effective remediation measures. This structured approach ensures that each aspect of the application's security is addressed comprehensively, enhancing the overall robustness of the software against potential cyber threats. The detailed algorithm of the application scanning process is shown in Figure 1.

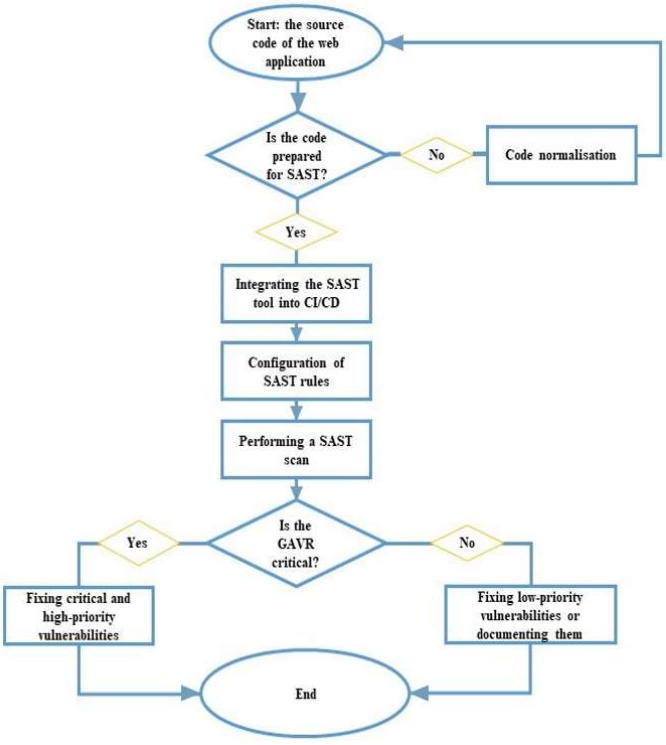


Figure 1. Algorithm for scanning a web application using the SAST method.

In conclusion, the integration of SAST into the DevSecOps framework significantly enhances the security posture of software development. By incorporating SAST early in the development lifecycle, teams can detect and address vulnerabilities at the initial stages, substantially reducing the risks and costs associated with later remediation. This proactive approach not only streamlines the security process but also embeds a security mindset directly within the development phase, empowering developers to produce more secure code from the outset. Ultimately, SAST provides a thorough and effective means of safeguarding against potential security threats, ensuring that both the application and its broader software ecosystem are robustly protected. This integration, therefore, represents a critical step forward in the evolution of secure software development practices.

VI. GENERAL APPLICATION VULNERABILITY RATE

The web application vulnerability assessment process uses a mathematical approach to systematically evaluate and quantify the security risks posed by identified vulnerabilities. This approach uses the Common Vulnerability Scoring System (CVSS) version 3.1, a widely accepted standard for assessing the severity of security vulnerabilities. It is a free and open industry standard for assessing the severity of computer system security vulnerabilities and attempts to assign severity scores to vulnerabilities, allowing responders to prioritize responses and resources according to the threat. Scores are calculated based on a formula that depends on several metrics that approximate ease of exploit and the impact of exploit [8].

In this paper we formalize the assessment process and present an improved methodology for calculating the General Application Vulnerability Rate (GAVR).

Each identified vulnerability is assigned a comprehensive rating based on the CVSS 3.1 system. The CVSS rating consists of a base score and a base severity, which are supplemented by additional scores that reflect the specific characteristics and potential impact of the vulnerability [20-22].

The Base metric group represents the intrinsic characteristics of a vulnerability that are constant over time and across user environments. It is composed of two sets of metrics: the Exploitability metrics and the Impact metrics [23].

The Temporal metric group reflects the characteristics of a vulnerability that may change over time but not across user environments. For example, the presence of a simple-to-use exploit kit would increase the CVSS score, while the creation of an official patch would decrease it.

The Environmental metric group represents the characteristics of a vulnerability that are relevant and unique to a particular user’s environment. Considerations include the presence of security controls which may mitigate some or all consequences of a successful attack, and the relative importance of a vulnerable system within a technology infrastructure.

Unlike traditional CVSS-based assessments, GAVR incorporates exploitability probability for a more comprehensive security evaluation. The model provides a structured method for prioritizing vulnerabilities, optimizing remediation efforts.

By integrating exploitability data, GAVR offers a dynamic assessment that adapts to real-world attack scenarios. This approach ensures that organizations can allocate resources more efficiently, mitigating the most critical threats first. Additionally, the model helps security teams identify patterns in vulnerabilities, improving long-term defensive strategies.

To quantify the severity and potential impact of each vulnerability accurately, these metrics are integrated into a structured formula. This calculation not only reflects the

inherent risk posed by the vulnerability but also accommodates temporal changes and environmental differences that could alter its severity. The formula for calculating the rating of an individual vulnerability V_i (1) is as follows:

$$V_i = BaseScore_i + TemporalScore_i + EnvironmentalScore_i, \quad (1)$$

where:

V_i is the rating of the i-th vulnerability.

$BaseScore_i$ is the CVSS base score for the i–th vulnerability, reflecting its inherent characteristics.

$TemporalScore_i$ – reflects the characteristics of the vulnerability that may change over time but do not depend on a particular user's environment. This score adjusts the baseline score to account for factors that may affect the severity of the vulnerability over time.

$EnvironmentalScore_i$ – modifies the baseline and temporal scores to tailor the CVSS score to a specific organizational context, reflecting the impact of the vulnerability on a particular environment. Environmental factors are derived from impact assessments of affected systems, considering industry-defined security controls and mitigation measures.

GAVR (2) is a metric designed to provide a holistic assessment of the security posture of web applications by aggregating the ratings of all identified vulnerabilities, weighted by their likelihood of exploitation. Mathematically it is expressed as:

$$GAVR = \frac{\sum_{i=1}^{N}(V_i * P_i)}{N}, \quad (2)$$

where:

$GAVR$ is a general application vulnerability rating.

N is the total number of vulnerabilities detected in the application.

V_i is the rating of the i–th vulnerability determined earlier.

P_i is the probability of exploitation of the i–th vulnerability, normalized on a scale from 0 to 1. This probability is then converted to a CVSS–equivalent scale from 0 to 10 to be consistent with the CVSS scoring mechanism (Fig. 2). The qualitative assessment on this scale is expressed as follows:

- low level of vulnerability – from 0.1 to 3.9;
- medium level of vulnerability – from 4.0 to 6.9;
- high level of vulnerability – from 7.0 to 8.9;
- critical level – from 9.0 to 10.0.

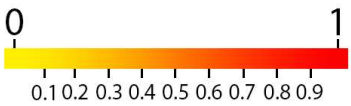


Figure 2. Scale for assessing the likelihood of vulnerabilities.

The weight-setting method is based on empirical security analysis data, considering exploitability trends observed in real-world attack scenarios. The probability values are influenced by publicly available threat intelligence reports, CVE exploitation frequency, and expert judgment in risk assessment methodologies. To ensure reproducibility, all factors used in weighting are sourced from standardized frameworks such as NIST, MITRE ATT&CK, and OWASP Top 10 rankings.

The proposed methodology for calculating the GAVR effectively integrates the severity and probability of exploitation of each vulnerability, providing a comprehensive assessment of application security vulnerabilities. This metric facilitates the prioritization of remediation efforts based on the aggregated risk profile, thereby increasing the effectiveness of the security measures implemented in the application.

This scientific approach to quantifying web application vulnerabilities using GAVR provides a systematic and objective basis for assessing security risks. By using the CVSS 3.1 scoring system in combination with the Probability of Exploitation factor, security analysts and web application developers can gain a detailed understanding of the application's vulnerability landscape and make strategic decisions about the secure development of web applications in the early stages of development.

Consider an example of a web application GAVR assessment. As an example of such an application, we have chosen a task from the HackTheBox learning platform called 'Proxy as a Service'.

There are various SAST tools available, each offering unique features and advantages. Selecting the best tool depends on multiple factors, such as integration capabilities, vulnerability detection accuracy, and ease of use. To determine the most effective solutions, we analyze three leaders from Gartner's Magic Quadrant for Application Security Testing: Veracode, Checkmarx, and Synopsys (Figure 3).

Figure 3. Magic Quadrant for Application Security Testing [10]

Veracode provides an advanced SAST solution with seamless integration into popular IDEs (Visual Studio, IntelliJ, Eclipse) and CI/CD pipelines, allowing developers to receive automated security feedback during coding. Its policy-driven scanning before deployment ensures clear guidance for prioritizing and addressing security issues across an organization's software ecosystem.

Checkmarx offers extensive programming language support and highly customizable scanning rules, making it adaptable to various enterprise security policies. Its deep code analysis capabilities and integration with CI/CD environments provide robust security assurance.

Synopsys delivers a comprehensive SAST solution tailored for complex enterprise environments. Its ability to scan

applications of all sizes efficiently, combined with in-depth vulnerability detection and remediation guidance, enhances security at scale.

Snyk Code, which we utilized in our case study, distinguishes itself with real-time IDE integration, AI-enhanced threat intelligence, and a developer-friendly approach. It prioritizes vulnerabilities based on contextual risk, ensuring efficient security management throughout development. One of the advantages of using Snyk Code is that it offers a free version, making it accessible for researchers and developers to conduct security testing without financial constraints. This allows our experiment to be easily replicated by other researchers, ensuring the reproducibility of findings. By leveraging a widely available tool, future studies can expand on our work, comparing results across different environments and applications. This accessibility promotes further research and improvements in SAST methodologies.

A comparative analysis of these tools reveals their strengths in different use cases (Table 1). Veracode excels in enterprise-wide risk visibility, Checkmarx provides highly configurable security policies, Synopsys ensures comprehensive vulnerability coverage, and Snyk Code emphasizes real-time, developer-centric security.

Table 1. Comparative analysis of SAST tools

Feature	Snyk Code	Checkmarx	Synopsys	Veracode
Integration	IDEs, CI/CD, real-time scanning	IDEs, CI/CD, customizable rules	IDEs, CI/CD, enterprise-focused	IDEs, CI/CD, policy-driven scanning
Programming Support	Wide language support	Extensive language support	Broad coverage	Multiple languages
Customization	AI-enhanced recommendations	Highly customizable	Enterprise-level configurations	Policy-driven
Detection Accuracy	Context-aware prioritization	High	Comprehensive	High
Ease of Use	Developer-friendly, real-time fixes	Configurable security policies	Complex but detailed analysis	Enterprise risk visibility
Unique Strengths	Free version, AI-enhanced insights	Customizable scanning rules	Deep code analysis for large orgs	Strong enterprise security features

This information is derived from publicly available reports, including Gartner's Magic Quadrant for Application Security Testing, vendor documentation, and independent research studies on SAST tool capabilities. Future research should extend testing to include multiple SAST tools across a diverse range of web applications, ensuring broader applicability of our methodology.

In order to enhance application security and integrate best practices within the development lifecycle, we have chosen to deploy Snyk's product [24, 25] for SAST tool.

Snyk offers a comprehensive suite of products designed to enhance security at every stage of the software development lifecycle. For example, it has Snyk Open Source that provides advanced software composition analysis (SCA) backed by industry-leading security and application intelligence or Snyk Container which helps developers and DevOps find, prioritize, and fix vulnerabilities throughout the SDLC. In case of static testing, the best product to deploy is Snyk Code, a solution that keeps pace with modern development, analyzing source code precisely across the software development lifecycle.

Snyk Code is a security tool that is fast and accurate and produces fewer false positives, making it easier for developers to remediate issues and build secure software. As an advantage, using Snyk Command Line interface, the development team can run Snyk Code tests locally or incorporate them into CI/CD pipeline to scan source code for security vulnerabilities, which makes the process automated and fast-pacing.

Let us scan the web application with the command "snyk code test". As a result of the scan, the solution returns 2 critical vulnerabilities (Fig. 4).

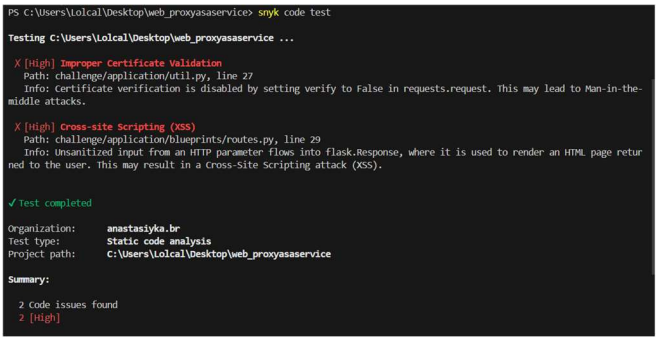


Figure 4. Results of the source code scanning with Snyk Code SAST solution.

The vulnerabilities found include XSS and incorrect certificate validation (Figures 5–6).

A cross–site scripting attack occurs when an attacker tricks a legitimate web application or website into accepting a request as coming from a trusted source [6]. This is done by leaving the context of the web application; the application then passes this data to its users along with other trusted dynamic content without checking it. The browser unwittingly executes a malicious client–side script (using client–side languages, usually JavaScript or HTML) to perform actions that would otherwise be blocked by the same browser's origin policy.

Regarding the second vulnerability, communication over encrypted TLS/SSL protocols can only take place if a server has a valid certificate that associates that server with a valid public key identifier issued by a third–party authority [18, 19].

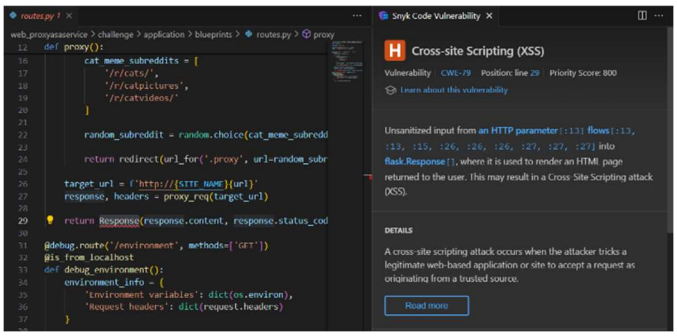


Figure 5. XSS vulnerability found with SAST solution.

If the certificate validation is cursory or incomplete, this creates a vulnerability that could allow an attacker to spoof one or more certificate details (such as the expiration date) and gain unauthorized access to sensitive data and privileged actions.

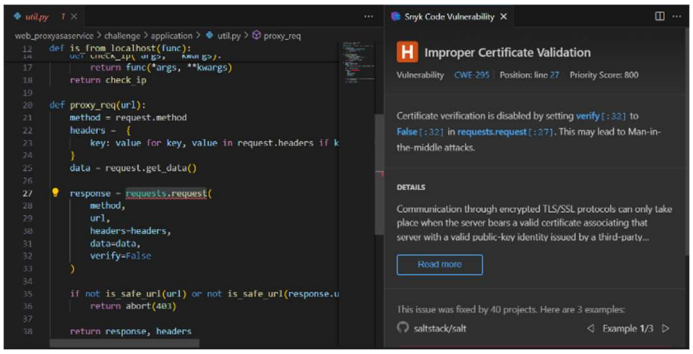


Figure 6. Improper certificate validation vulnerability found with SAST solution.

One of the key benefits of using Snyk Code's SAST solution is its prioritization, which is directly proportional to the level of vulnerability exploitation. In the case of this particular web application, the probability of the vulnerability being exploited is calculated based on empirical data and expressed as $P_i = 0.8$, corresponding to a CVSS-equivalent score of 8.

Based on the information in CWE and [17], let us calculate [26] the rate of the XSS vulnerability V_{xss} (3) using the formula of V_i (1). So, for the above runtime environment:

$$V_{xss} = AV{:}N/AC{:}L/PR{:}N/UI{:}N/S{:}C/C{:}H/I{:}L/A{:}N = \\ = 9.3\,, \quad (3)$$

where:
$AV{:}N$ – Attack Vector: Network;
$AC{:}L$ – Attack Complexity: Low;
$PR{:}N$ – Privileges Required: None;
$UI{:}N$ – User Interaction: None;
$S{:}C$ – Scope: Changed;
$C{:}H$ – Confidentiality: High;
$I{:}L$ – Integrity: Low;
$A{:}N$ – Availability: None.

Now, based on the information in CWE, we will similarly calculate the rate of improper certificate validation (ICV) vulnerability V_{icv} (4) using formula (1). So, for the above runtime environment:

$$V_{icv} = AV{:}N/AC{:}L/PR{:}N/UI{:}N/S{:}U/C{:}H/I{:}H/ \\ A{:}N = 9.1, \quad (4)$$

where:
$AV{:}N$ – Attack Vector: Network;
$AC{:}L$ – Attack Complexity: Low;
$PR{:}N$ – Privileges Required: None;
$UI{:}N$ – User Interaction: None;
$S{:}U$ – Scope: Unchanged;
$C{:}H$ – Confidentiality: High;
$I{:}H$ – Integrity: High;
$A{:}N$ – Availability: None.

At this point, it is possible to calculate the General Application Vulnerability Rate (5):

$$GAVR = \frac{(V_{xss} * P_{xss}) + (V_{icv} * P_{icv})}{2} = \\ = \frac{(9{,}3 * 0{,}8) + (9{,}1 * 0{,}8)}{2} = 7{,}36, \quad (5)$$

where:
V_{xss} – rate of XSS vulnerability;
P_{xss} – probability of realization of XSS vulnerability;
V_{icv} – rate of improper certificate validation vulnerability;
P_{icv} – probability of realization of improper certificate validation vulnerability.

Therefore, the general application vulnerability rate is 7.36. Considering the scale in Figure 2, according to the qualitative scale, this vulnerability level can be rated as “high”, so the web application cannot be considered secure. In this case it is recommended to fix the existing vulnerabilities in the code and perform a second scan.

One of the tools for fixing the detected vulnerabilities is the built-in functionality of the Snyk Code solution, which allows viewing possible options for fixing the vulnerability (Fig. 7).

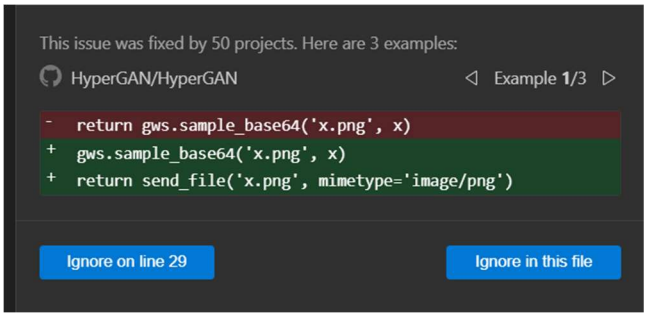


Figure 7. Recommendations for remediation of the vulnerability based on the other projects.

In summary, the enhanced methodology for calculating the General Application Vulnerability Rate (GAVR) presented in this paper offers a refined and quantifiable approach to assessing security risks in web applications. By utilizing the Common Vulnerability Scoring System (CVSS) version 3.1 as a foundation, and augmenting it with a mathematical model that incorporates the probability of exploitation, our methodology provides a robust mechanism for determining the overall security posture of applications.

The calculation of GAVR encompasses an aggregate of scores derived from the CVSS base, temporal, and environmental metrics, tailored to each identified vulnerability within the application's ecosystem. This holistic view allows for a nuanced understanding of potential security threats, considering both their inherent risks and the specific contextual factors of the operating environment. The integration of the probability of exploitation further enhances this model, ensuring that the severity of vulnerabilities reflects not only their potential impact but also their likelihood of being exploited.

Ultimately, this refined approach aids in prioritizing remediation efforts more effectively, directing resources towards the most significant vulnerabilities, and enhancing the security measures within the application development lifecycle. This not only mitigates the risks of potential breaches but also aligns with best practices in secure software development, promoting a proactive stance in addressing security from the earliest stages of application design and development. By adopting such methodologies, organizations can significantly bolster their defense mechanisms against the evolving landscape of cyber threats, ensuring that their applications remain robust and resilient against attacks.

VII. DISCUSSION

The results of our study underline the significant impact of incorporating SAST into the Software Development Life Cycle (SDLC) for enhancing web application security. The identification of critical vulnerabilities such as cross-site scripting (XSS) and improper certificate validation emphasizes the necessity of early detection and mitigation of security risks.

The primary hypothesis of this study was that the use of SAST, particularly with the Snyk Code tool, would provide a comprehensive and effective means of identifying and quantifying web application vulnerabilities. Our findings support this hypothesis, demonstrating that SAST can uncover severe security issues early in the development process, thereby reducing the potential for exploitation and enhancing overall application security.

The approach to quantifying vulnerabilities using the General Application Vulnerability Rate (GAVR) model, which integrates the Common Vulnerability Scoring System (CVSS) with the probability of exploitation, offers a balanced and objective method for prioritizing security patches. The GAVR model's ability to provide a holistic assessment of application security facilitates strategic decision-making in the remediation process. The identification of a high GAVR score in this test case highlights the model's effectiveness in pinpointing areas requiring immediate attention.

Compared to previous studies that primarily focused on qualitative assessments or isolated quantitative measures, our integrated approach provides a more comprehensive understanding of the vulnerability landscape. The inclusion of both severity and exploitability factors in our model aligns with the findings of studies emphasizing the importance of early and detailed vulnerability detection in the SDLC.

The practical implications of our findings are substantial. By employing SAST tools like Snyk Code, developers can receive real-time feedback on security issues, enabling faster and more efficient remediation. This proactive approach not only reduces the risk of security breaches but also aligns with industry best practices for secure software development. Additionally, the ability to provide specific recommendations for fixing identified vulnerabilities demonstrates the practical utility of our approach in real-world scenarios.

While this study provides a robust framework for web application security assessment, it is not without limitations. The reliance on a single SAST tool and the specific test cases used may limit the generalizability of our findings. Future research could explore the integration of multiple security testing tools and a broader range of applications to validate and refine the proposed methodology. Additionally, examining the long-term effectiveness of the implemented security measures through repeated scans could provide deeper insights into the sustainability of our approach.

In summary, this study highlights the critical role of SAST in the early detection and mitigation of web application vulnerabilities. The developed GAVR model offers a valuable tool for quantifying and prioritizing security risks, thereby enhancing the overall security posture of web applications. By integrating these practices into the SDLC, developers can create more secure applications, ultimately protecting sensitive data and maintaining user trust.

VIII. CONCLUSION

Therefore, the study proposed the use of the SAST method using the Snyk Code tool to assess the security of a web application. The developed algorithm for scanning a web application using the SAST method allowed us to identify critical vulnerabilities, namely cross-site scripting (XSS) and incorrect certificate validation. The use of the GAVR comprehensive security assessment of the web application showed the level of criticality of the web application.

Based on the results, recommendations are made to fix the identified vulnerabilities and, in accordance with the developed methodology, to perform a rescan to verify the effectiveness of the measures taken. This approach helps to improve the security of the web application, minimizing potential risks to end users and the system as a whole.

The results of the study demonstrate the need for an integrated approach to security that includes regular testing and analysis of software security. For example, the algorithm developed to scan a web application using the SAST method demonstrates the importance and effectiveness of integrating security measures in the early stages of software development.

References

[1] A. S. Gillis, "What is the software development lifecycle (SDLC)? A definition from WhatIs.com," *Search Software Quality*. [Online]. Available: https://www.techtarget.com/searchsoftwarequality/definition/software-development-life-cycle-SDLC.

[2] Verizon Business, "2024 Data Breach Investigations Report," 2024. [Online]. Available at: https://www.verizon.com/business/resources/reports/dbir/.

[3] C. Harris, "50 Web Security Stats You Should Know In 2025," *Expert Insights*, Jan. 10, 2024. [Online]. Available at: https://expertinsights.com/insights/50-web-security-stats-you-should-know/.

[4] E. Moyle, "5 ways to automate security testing in DevSecOps | TechTarget," *Search Security*. [Online]. Available: https://www.techtarget.com/searchsecurity/tip/5-ways-to-automate-security-testing-in-DevSecOps.

[5] K. Brush, "What is static application security testing (SAST)? Definition from WhatIs.com," *Search Software Quality*. [Online]. Available: https://www.techtarget.com/searchsoftwarequality/definition/static-application-security-testing-SAST. [Accessed: Mar. 27, 2025].

[6] F. T. Alssir and M. Ahmed, "Web security testing approaches: comparison framework," in *Proc. 2nd Int. Cong. Comput. Applications Computational Science*, Springer, 2012, pp. 225–238. https://doi.org/10.1007/978-3-642-28314-7_23.

[7] A. Horváth, P. M. Erdősi, and F. Kiss, "The Common Vulnerability Scoring System (CVSS) generations – usefulness and deficiencies," in *IT és hálózati sérülékenységek társadalmi-gazdasági hatásai*, F. Kiss and A. Horváth, Eds., Infota, 2016, pp. 137–153.

[8] S. A. Vaddadi, R. Thatikonda, A. Padthe, and P. R. R. Arnepalli, "Shift-left testing paradigm process implementation for quality of software based on fuzzy," *Soft Computing*, Art. no. 87, 2023. https://doi.org/10.1007/s00500-023-08741-5.

[9] Synopsys, "2023 Gartner Magic Quadrant for Application Security Testing," [Online]. Available at: https://www.synopsys.com/software-integrity/resources/analyst-reports/gartner-magic-quadrant-appsec.html.

[10] O. Trofymenko, A. Dyka, and Y. Loboda, "Analysis of web application testing tools," *Cybersecurity: Education, Science, Technique*, vol. 4, no. 20, pp. 62–71, 2023. https://doi.org/10.28925/2663-4023.2023.20.6271.

[11] V. Susukailo, "Using the devsecops approach to analyze current information security threats," *Cybersecurity: Education, Science, Technology*, vol. 2, no. 14, pp. 26–35, 2021.

[12] F. A. Alaba, M. Othman, I. A. T. Hashem, and F. Alotaibi, "Internet of Things security: A survey," *J. Network Comput. Appl.*, vol. 88, pp. 10–28, 2017. https://doi.org/10.1016/j.jnca.2017.04.002.

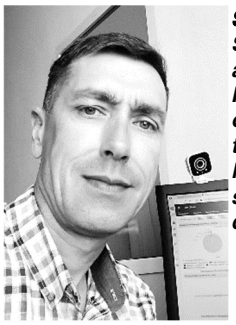
[13] A. O. Gapon, V. M. Fedorchenko, and O. V. Sievierinov, "Methods and means of static and dynamic code analysis," *Radiotekhnika*, vol. 212, pp. 7–13, 2023. https://doi.org/10.30837/rt.2023.1.212.01.

[14] Q. Yas, A. Alazzawi, and B. Rahmatullah, "A Comprehensive Review of Software Development Life Cycle methodologies: Pros, Cons, and Future Directions," *Iraqi J. Comput. Sci. Math.*, pp. 173–190, 2023. https://doi.org/10.52866/ijcsm.2023.04.04.014.

[15] H. S. Lisda, M. Y. R. Madhika, and E. Bayunanda, "Systematic literature review SDLC in software engineering," *Int. J. Comput. Inf. Technol.*, vol. 12, no. 1, 2023.
[16] H. D. Jayawardana, M. I. Uyanahewa, V. Hapugala, and T. Thilakarathne, "An analysis of XSS vulnerabilities and prevention of XSS attacks in web applications," *Int. J. Comput. Appl.*, vol. 182, no. 20, pp. 1–8, 2023.
[17] K. Wang, Y. Zheng, Q. Zhang, G. Bai, M. Qin, D. Zhang, and J. S. Dong, "Assessing certificate validation user interfaces of WPA supplicants," in *Proc. ACM MobiCom '22: 28th Annu. Int. Conf. Mobile Comput. Networking*, 2022. https://doi.org/10.1145/3495243.3517026.
[18] M. Luo, B. Feng, L. Lu, and K. Ren, "On the complexity of the Web's PKI: Evaluating certificate validation of mobile browsers," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 1, pp. 1–14, 2023. https://doi.org/10.1109/TDSC.2023.3255869.
[19] H. Howland, "CVSS: Ubiquitous and Broken," *Digital Threats: Res. Pract.*, 2021. https://doi.org/10.1145/3491263.
[20] M. Walkowski, M. Krakowiak, M. Jaroszewski, J. Oko, and S. Sujecki, "Automatic CVSS-based vulnerability prioritization and response with context information," in *2021 Int. Conf. Soft., Telecommun. Comput. Networks (SoftCOM)*, 2021. https://doi.org/10.23919/SoftCOM52868.2021.9559094.
[21] J. Franklin, C. Wergin, and H. Booth, "CVSS implementation guidance," National Institute of Standards and Technology, 2014. https://doi.org/10.6028/NIST.IR.7946.
[22] A. Balsam, M. Nowak, M. Walkowski, J. Oko, and S. Sujecki, "Analysis of CVSS vulnerability base scores in the context of exploits' availability," in *Proceedings of the 2023 23rd Int. Conf. Transparent Optical Networks (ICTON)*, 2023. https://doi.org/10.1109/ICTON59386.2023.10207394.
[23] K. Kuszczyński and M. Walkowski, "Comparative analysis of open-source tools for conducting static code analysis," *Sensors*, vol. 23, no. 18, Art. no. 7978, 2023. https://doi.org/10.3390/s23187978.
[24] A. War, A. Habib, A. Diallo, J. Klein, and T. F. Bissyandé, "Security vulnerabilities in Infrastructure as Code: What, how many, and who?" *Res. Square*, 2023. https://doi.org/10.21203/rs.3.rs-3600645/v1.
[25] A. G. Korchenko, B. B. Akhmetov, S. V. Kazmirchuk, and E. A. Chasnovskyi, "Information security risk assessment system – 'RISK-CALCULATOR'," *Ukrainian Sci. J. Inf. Sec.*, vol. 23, no. 2, 2017. https://doi.org/10.18372/2225-5036.23.11824.
[26] P. Maslianko and I. Savchuk, "DevOps – concept and structural representation," *KPI Sci. News*, no. 4, pp. 39–51, 2022. https://doi.org/10.20535/kpisn.2021.4.261938.

Anastasiia BRYHYNETS *Bachelor of cybersecurity, lead engineer of cryptographic laboratory. Field of scientific interests: application of machine learning methods in cyber security systems, vulnerability assessment methodologies, information security compliance, incident response and protection of confidential data systems.*

Halyna HAIDUR *Doctor of Technical Sciences, Professor, Head of Department of Information and Cyber Security. Field of scientific interests: cyber security technologies, application of machine learning methods in cyber security systems, methods and means of detecting anomalies in network traffic.*



Sergii GAKHOV *Candidate of Military Sciences, an Associated professor, associate professor of Department of Information and Cyber Security. Field of scientific interests: cyber security technologies, application of machine learning methods in cyber security systems, methods and means of detecting malicious processes.*

Vitalii MARCHENKO *PhD, Associate professor of Department of Information and Cyber Security Field of scientific interests: cyber security technologies, malware analysis, cryptanalysis, ML in cyber security systems, steganography.*