Evaluation of Software Static Analyzers

Lobna Khaled
Nile University
Sheikh Zayed
1150003060, 0020
l.khaled@nu.edu.eg

Nashwa Abdelbaki
Nile University
Sheikh Zayed
238541751, 0020
nabdelbaki@nu.edu.eg

ABSTRACT
With the massive increase of software applications and websites, testing has become a very important concern in the software development process. This is due to the spread of a large number of security flaws. Dynamic testing requires code execution to examine the functional and non-functional behavior of software systems. It requires more time and cost, and it finds fewer bugs. On the other hand, static testing is done before code deployment and without code execution. Additionally, it provides a comprehensive diagnostics of code and focuses more on defects prevention. This provides greater benefits and is more cost-effective. Several techniques exist to perform static testing. One of them is using static analyzers tools that locate vulnerabilities in code and identify potential security flaws. Furthermore, these tools offer solutions to avoid security breaches. This paper contributes to the field of software testing in many aspects, by introducing the recent research studies in static analyzers. We discuss the importance of static analyzers and their challenges. We provide useful guidelines for selecting test cases to evaluate different static analyzers tools. Additionally, we explain a list of software common weaknesses. Furthermore, we explore the current research trend in static analyzers tools and techniques. Finally, we perform a research study to examine the performance of five PHP static analyzers tools. We report their ability to detect five common vulnerabilities by using Software Assurance Reference Dataset (SARD) test suite.

CCS Concepts
• **Security and privacy→ Software security engineering. Security and privacy → Web application security.**

Keywords
Static analyzers, Software testing, Software common weakness, Static analyzers techniques, PHP static analyzers tools

1. INTRODUCTION
Nowadays software applications represent a significant role in our daily life. The past years showed an improvement in a variety of these applications, whether they are related to work, social media, education, or entertainment applications. Business companies are keen on introducing mobile and web applications that are reliable and secured to maintain their relationship with customers. Developing these applications follow a specific life cycle that includes requirements analysis, design, building, testing, and deployment. The sequence of these steps depends on the applied process model. Testing is considered an important activity in software development. It occurs at different stages during the software life cycle. There are a variety of process models that software engineers follow in developing these applications. Testing is considered a very important step during the development phase. If testing is done properly it can save time and effort. Figure 1 shows that the cost required for fixing bugs increases as the software is developed and has been released for use [1].

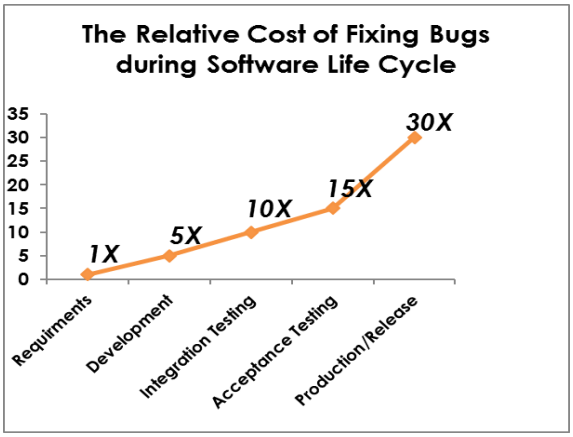

Figure 1. The relative cost of fixing bugs [1].

There are different testing types. One of them is static testing by using static analyzers tools. OWASP (Open Web Application Security Project) is an online community that provides useful information and acts as a benchmark for web application security. According to OWASP, static analysis refers to the running of static code analysis tools that attempt to highlight possible vulnerabilities within 'static' (non-running) source code, by using techniques such as Taint Analysis and Data Flow Analysis [2]. Testing engineers find difficulties in deciding which static analyzer tool to use in each project. Each tool excels in detecting specific problems. The objective of this paper is to highlight the importance of using static analyzers for testing software applications. We discuss the state of the art technologies, approaches, and tools used in static analyzers tools. We identify a set of criteria for selecting test cases to examine these tools. Furthermore, we explain some common software weaknesses. Finally, we perform a research study to evaluate five PHP static

ICSIE 2020, November 11–13, 2020, Cairo, Egypt

ACM ISBN 978-1-4503-7721-8/20/04...$15.00
https://doi.org/10.1145/3436829.3436835

analyzers tools. The rest of this paper is organized as follows. Section 2 introduces the literature review. Static analyzers importance, challenges, and alarm types are explained in Section 3. Guidelines for selecting test cases are presented in Section 4. A list of software common weaknesses is illustrated in section 5. The analysis and the research study section is highlighted in section 6. Finally, conclusion and future work are explained in section 7.

2. LITERATURE REVIEW

Testing applications against security holes have become a big research topic as there are now a lot of security flaws and there are many techniques to breach important mobile and web applications. In this section, we will discuss how using static analyzers have been addressed by different scholars and in several contexts. In reference [3], the authors surveyed 42 developers to examine how testers adopt and configure static analyzers during the development life cycle. The questionnaire was useful to clarify the most used tools like find bugs and check style. Furthermore, the participants identified the main development context. They include static analyzers in their projects like local programming, code review, and continuous integration. Rahma Mahmood and Qusay H. Mahmoud evaluated a list of static analyzers for java source code which are Find bugs, PMD, LAPSE, and Yasca. In addition to other static analyzers for C/C++ source codes which are RATS, Flaw Finder and Yasca. They used the source code of the US Department of Homeland Security's Software Assurance Metrics and Tool Evaluation (SAMATE) Project website to test the above-mentioned tools. They analyzed the successful and unsuccessful attempts of each tool in detecting code vulnerabilities. After that, they reported the system requirement, ease of use and effectiveness of each tool. The study is considered very helpful as it introduces a list of static analyzers that are widely used in [4]. Reference [5] the authors proposed a machine learning technique that can eliminate missing important bugs. Additionally, it reduces false alarms by combining sound and unsound approaches of static analyzers. In reference [6], the authors developed a static analyzer for detecting errors in C. They compared their tool with two commonly used static analyzers. The result was that their tool was able to detect more errors than the other two tools (221 to 339 more bugs). A. Arusoaie, S. Ciobâca, V. Craciun, D. Gavrilut and D. Lucan evaluated several open-source static analyzers and their ability to detect various error types and subtypes. They used the Toyota ITC test suite which contains 650 test cases. It targets low-level defects like buffer overflow and memory allocation defects [7]. Reference [8] the authors mentioned the usefulness of using static analyzers to guarantee compliance with different coding standards like CWE, OWASP top ten, JPL and MISRA. They compared sixteen different static analyzers tools for C and C++. In [9], the authors stated that static analyzers are all common in analyzing the source code without actual execution. They can be divided into three types which are:

- Syntax Checkers: They focus only on examining program syntax. Most of the MISRA C rules can be examined by this type of analyzer.
- Unsound Semantic Analyzers: They extract semantic errors from programs like runtime error (division by zero and buffer overflow). Such analyzers are characterized by producing False Positive (generating alarm while there is no error) and False Negative errors (dismissing producing alarm while there is an error). Example Bug finder and Coverity.
- Sound Semantic Analyzers: This relies on abstract interpretation and formal methods to check the code. They offer an accurate and strict mathematical way for testing applications. Other authors compared sound static analyzers with unsound static analyzers [10]. They explained the three main components in static analyzers which are syntactic information, semantic information and set of diagnostic rules. They conducted a research study to show that using static analyzers with high diagnostic rules will increase the error detection rate in the development stage by 55%.

In [11] the authors proposed a static analysis tool for Python by the abstract interpretation which targets only some Python features. They used the Python Official Regression Tests Suite and the Python Performance Benchmark Suite to examine their framework. Using static analysis to detect vulnerabilities in android applications is discussed in [12]. The authors discussed static analysis techniques such as data flow and control flow. They mentioned famous security checking tools for android applications like Flowdroid and Amandroid. In [13], the authors proposed a novel model of static analyzers. It can learn from the data provided to it. Authors in [14] highlighted the importance of integrating security applications into agile software development activities. They interviewed 8 developers to know their expectations and potential challenges they may address while integrating the static analyzer tools in the development life cycle. In [15], authors performed a deep analysis of 8 static analyzers tools for C # and evaluated them to a predefined set of criteria. In reference [16], the authors suggested the formation of a multiple level static analyzer models. The first level checks program compliance with coding standards. The other levels focus on finding critical errors. In [17], the authors explained different approaches they used in building a static analyzer tool SharpChecker. It supports C# programs and can detect 30 different bugs by applying dataflow, context- and path-sensitive inter-procedural analysis. In reference [18] the authors showed how static analyzers can support fuzz testing. They performed a study to evaluate their approach. The study showed increasing test coverage by 10-15% over stand-alone fuzzers. This is in addition to reducing the time required for detecting vulnerabilities. In [19], authors mentioned different sources of unsafe data in software applications. After that, they introduced a new approach for detecting unsafe data in an embedded system by the use of static analyzers techniques like Abstract Syntax Tree (AST), data-flow analysis and control flow analysis. In [20], authors performed a study to explain the different types of False-Positive patterns generated by static analyzers. They examined their study on open source and commercial tools. Their contribution was generating a list of 14 patterns of False-Positive alarms. This helps developers know the root cause of these flaws and be more focused. Authors in [21] explained two security mechanisms which are the Vulnerability Prediction Models (VPMs) and the Optimum Checkpoint Recommendation (OCR). These mechanisms can ensure secure and reliable software. The authors discussed the role of static analyzers to support and enhance the use of these mechanisms. In [22] the authors introduced a technique for repositioning and grouping similar alarm types. The technique produces a single alarm instead of several similar alarms to reduce the large number of alarms generated by static analyzers.

3. IMPORTANCE AND CHALLENGES OF STATIC ANALYZERS

Static analyzers enhance software security. As with the massive increase of software, security flaws have also been increased. They target vulnerable software. Consequently, it can result in

getting user information and gain full control over the software. Static analyzers examine the code and detect security wholes. They ensure that the applications are secured and safe for use. Static analyzers also ensure compliance with coding standards. Since organizations that develop important applications like automotive or banking applications must follow coding standards. Static analyzers can be considered the best way to examine compliance with coding standards. Furthermore, static analyzers are the most forward way that lets developers learn from their mistakes. They introduce instant solutions and best practices. Also, they reduce time and effort. If developers know the best static analyzers that are compatible with their developing environment, they don't have to wait until the end of important milestones in their projects to know their errors. They can use it while the project is still under the development process [4][5][6][23].

Figure 2. Importance of static analyzers.

Although static analyzers are very useful in detecting errors, many software testers find difficulties in using them. This is due to generating too many false alarms. Also, testers find difficulty in choosing the best static analyzer tool that will be compatible with the work environment. They also may not have enough time to learn how the tool should work. Testers are uncertain regarding tools outcomes. Some tools poorly explain the generated alarms and may not provide solutions for fixing bugs [7][8].

3.1 Alarms Types Generated by Static Analyzers

False Positive: if the code is bug-free, and the tool reports bugs. False Negative: if the tool ignores true bugs, and reports bug free code. True Positive: if the code has bugs, and the tool recognized these bugs. True Negative: if the code doesn't have bugs, and the tool doesn't report any bugs.

4. Guidelines for Selecting Test Cases

For quantitatively evaluating different tools, test cases used for evaluation should follow design guidelines. In this section, we will highlight some important guidelines for selecting test cases.

4.1 Run the Tools on Test Cases with Known Defects in Advance

Defects that will be evaluated should be known in advance. It is preferred to run the static analyzers on determined test cases. With specified bugs for each test case. It is not recommended to use big projects in evaluating different tools. For simplicity and for getting accurate results: use small projects with well-defined bugs in advance.

4.2 Categorize Defects Types

Different defect types should be sorted. Then grouped and prioritized to know the strengths and weaknesses of each tool.

4.3 Appropriate Selection of Test Cases and Analysis Tools to Ensure Wide Coverage of Code Defects

Test cases should not cover only code quality. But they should also focus on security flaws. Each test case should target certain bugs or security flaws to examine the tool's ability in detecting bugs. Tools selection should be examined properly as some tools don't make a deep analysis and just focus on quality checking. Besides, it is preferred to select the tools that suggest solutions for preventing bugs.

4.4 Test Cases Should State All Requirements in Advance

All assumptions should be clearly defined in advance. If there are test cases designed for 16-bit machines or 32-bit machines, it should be declared in advance to avoid misleading evaluation of static analyzers [10].

5. SOFTWARE COMMON WEAKNESSES

This section shall explain a list of software security weaknesses, according to Common Weakness Enumeration (CWE). It acts as a benchmark for security weakness identification. It is considered a reference for evaluating security tools.

5.1 Improper Neutralization of Input during Web Page Generation ('Cross- Site Scripting') (CWE-79)

It is a web security vulnerability. An attacker can inject malicious code into a vulnerable web application that does not sanitize the user's input. The attack happens when a victim visits a web application which runs the malicious code. The webpage then transfers the script to the victim browser. This enables the attacker to do any action on behalf of the original user. The attacker can gain access to users' passwords and steal the victim's cookies. Attackers can inject malicious code through applications enabling comments or forums. Cross-site scripting can be used also to hack the website itself, by changing the web contents or redirecting the user to another website that is malicious [24].

5.2 Missing Authorization (CWE-862)

Occurs when a user tries to access software and the software doesn't apply any required authorization check to identify the user's identity. It allows users to access restricted data. And they may also apply malicious actions over this data. This can lead to a wide range of problems, including information exposures, and denial of service [25].

5.3 Improper Neutralization of Special Elements used in a SQL Command ('SQL Injection')(CWE- 89)

It is a code injection attack that targets data-driven applications by placing malicious SQL code into a user input field like user name or user password. The attacker injects a SQL statement that reveals confidential information by manipulating the SQL query. It can be used also to make changes in the application database (adding, modifying, deleting) records [26].

5.4 Information Exposure through an Error Message (CWE-209)

This weakness occurs when the application shows an error message that reveals important information regarding system configuration, users, used technology and confidential data. The generated data may be used by the attackers. Or it might help them to run a more sophisticated attack. For example, if a SQL error is caught by exception handling, the error message might reveal important information regarding SQL query structure and other important information. Another common error that occurs when requesting a non-existent URL is HTTP 404 not found. It can show information about the webserver and its specifications. Additionally database systems may return error code if there is a connection problem or there is a query problem. That also may provide important information about the database type, the database server, and the login details [27].

5.5 Path Traversal Attack (CWE-22)

This type of attack is used by attackers to access restricted directories and execute commands outside of the web server's root directory. It may be used to disclose information from a web server or a web application. Attackers use special elements like the (../) sequence to go up a directory folder and reach other files outside the restricted locations. This attack is used to get important information like passwords and to gain access to confidential folders. It may be also used to identify different security mechanisms on the server or the web application to bypass them. Furthermore, it can be used to delete important files that affect the application's availability [28].

5.6 Deserialization of Untrusted Data (CWE-502)

Serialization is the process of transforming an object into a binary or textual form that can be persistent in a storage medium or transferred through a network. Deserialization is the opposite process. It converts the streamed data to an object. The flaw is initiated when de-serializing malicious user input without appropriate verification that the data is not tainted. This can lead to Denial-of-Service (DoS) attacks [29].

6. ANALYSIS and RESEARCH STUDY

By probing into the latest research topics in static analyzers, we can summarize that the research topics are mainly concerned with:

- Evaluating tools against benchmarked test suites
- Developing new tools to reduce false-positive rates
- Conducting surveys and questionnaire with developers and testers to understand their perspective toward static analyzers
- Integrating machine learning algorithms to introduce new features for static analyzers.
- Highlighting the benefits of integrating static analyzers to security mechanisms
- Explaining the role of static analyzers in detecting vulnerabilities in embedded systems and mobile applications

Also, we can note that the main focus is on the C, C++ and Java languages. Future research direction is required to examine other tools that support different programming languages like PHP, Ruby, Scala, and Python. Using other test suites rather than Juliet and Toyota is urgently needed for research. There are different techniques for building static analyzers tools like Abstract Syntax Tree (AST), Data Flow Analysis (DFA) and Control Flow Analysis (CFA).

6.1 Abstract Syntax Tree

AST represents the syntax of source code abstractly as a hierarchical tree structure and ignores unnecessary details. The example 1 + 4 * 3 + 2 * 6 is presented in Figure 3[30].

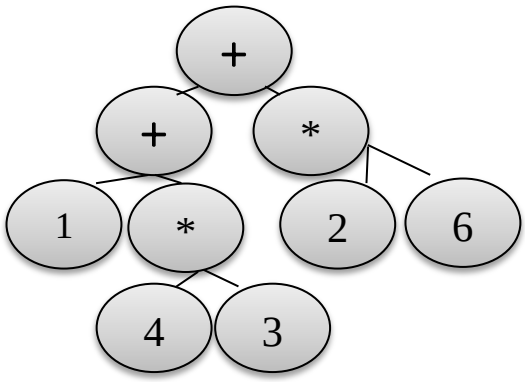

Figure 3. Abstract Syntax Tree

6.2 Data Flow Analysis

It is concerned with gathering information about the definition, use, and dependencies of program variables as shown in Figure 4 [31].

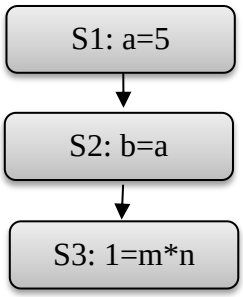

Figure 4. Data flow graph.

6.3 Control Flow Analysis

It is concerned with identifying the order of program instructions, to determine all possible execution paths, a loop structure, and dependencies. Example: if –else shown in figure 5 [32].

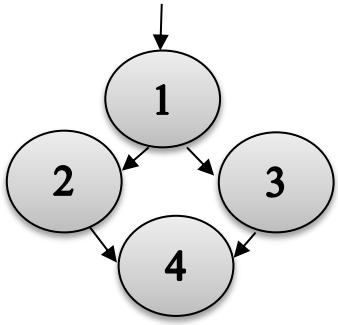

Figure 5. Control flow graph

There is a variety of analysis techniques adopted by different tools. Each tool is based on an analytical framework that comprises certain techniques. What makes tools differ is the method of applying these techniques by different algorithms and approaches. The current research papers focus on only evaluating a limited set of vulnerabilities, and neglect covering recent OWASP top ten and CWE top 25 vulnerabilities. An interesting field that needs further research is using machine learning, data and text mining algorithms to develop static analyzers tools. Static analyzers should be widely used for all software applications to avoid security breaches.

After studying the current research study in static analyzers, we decided to examine static analyzers tools that support PHP language due to the lack of research in evaluating PHP static analyzers, and the widespread of using PHP languages in several web projects nowadays. This is in addition to the security flaws still found in PHP.

We also selected the Software Assurance Reference Dataset (SARD) test suite which is considered one of the successful contributions of SAMATE projects. SARD is considered a test set for common security flaws. It allows researchers and testers to evaluate several tools against well-established standards. The SARD data set supports several languages, platforms, and applications. Currently, SARD contains 42 test suites. Our research study will focus on the PHP test suite. It contains 42212 test cases that reflect different security vulnerabilities. We selected five static analyzers tools for evaluation which are: Visual Code Grepper, RIPS, Sparrow Scan, Smart Dec and Dev Bugs. Consequently, we examined the tools' performance regarding detecting certain vulnerabilities which are: cross-site scripting, missing authorization, SQL injection, and information exposure through an error message. For each vulnerability, we selected a list of good and bad test cases from the SARD test suite to guarantee fair evaluation. We selected a total of fifteen good test cases and a total of fifteen bad test cases that reflect the prescribed four common weaknesses. The tools evaluation was measured by False Positive (FP), False Negative (FN), True Positive (TP) and True Negative rates (TN) as shown in Figures 6, 7, 8 and 9.

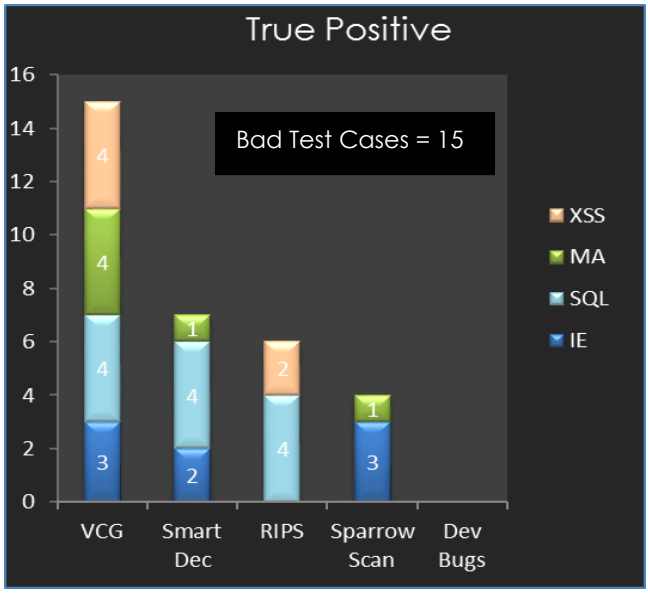

Figure 6. Tools' detection of True Positive alarms.

Figure 6 shows that the Visual Code Grepper tool detected all the positive alarms. Smart Dec, RIPS, and Sparrow scan tools detected some of the true alarms. Finally, the Dev Bug tool failed to detect any positive alarm.

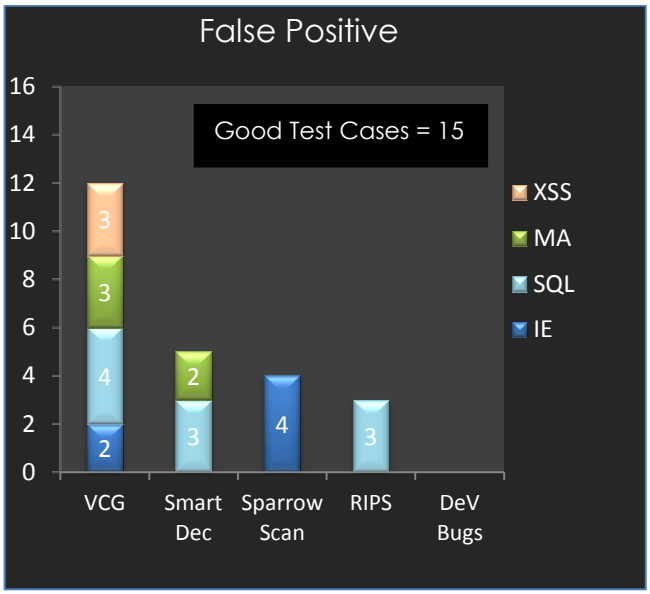

Figure 7. Tools' detection of False Positive alarms.

The above graph, Figure 7, shows that although the Visual Code Grepper tool detected most of the true positive alarms, its False Positive rate is considered the highest. This means that it reports wrong alarms while there are no positive vulnerabilities. It was followed by Smart Dec and Sparrow Scan.

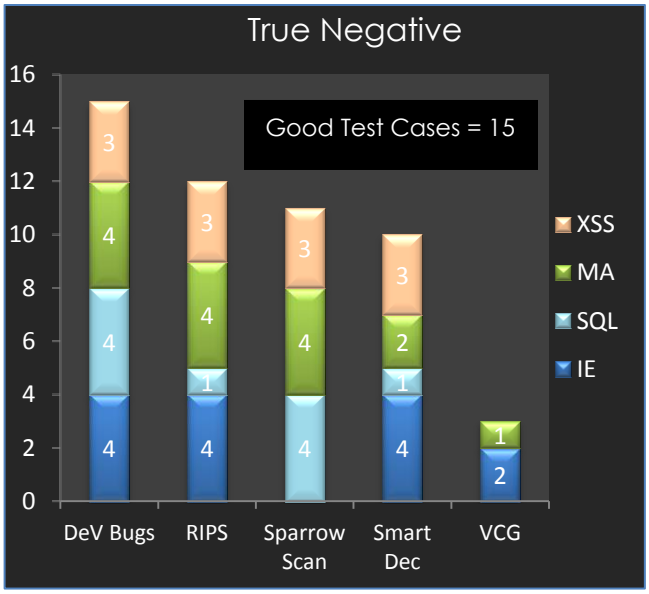

Figure 8. Tools' detection of True Negative alarms.

Figure 8 shows that the Dev Bugs and RIPS tools have the highest true negative rate. This means that they correctly don't report nonexistent alarms.

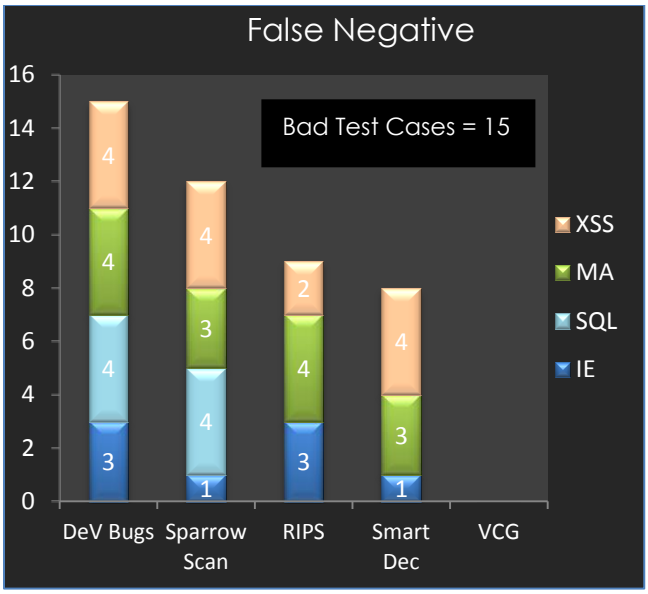

Figure 9. Tools' detection of False Negative alarms.

The above graph shows that the Dev Bugs tool ignored all the true vulnerabilities and failed to report any. On the other hand, the False-Negative rate of the visual code grepper tool is zero. This means it detected all the vulnerabilities, but sometimes it named them differently.

After evaluating the five tools we can conclude that VCG, RIPS, and Smart Dec excel in detecting SQL injection flaws. XSS is better detected by VCG and RIPS tools. Besides VCG, Smart Dec and Sparrow Scan tools can discover missing authorization errors more successfully. Finally, VCG, Sparrow Scan, and Smart Dec excel in discovering information exposure through an error message weakness. Using static analyzers for enhancing software security is not an easy process. It requires time and effort to select the best tools that can generate useful results. Testers must have a good experience in understanding the structure of these vulnerabilities. Some tools can successfully catch the flaws but name them differently. RIPS and Smart Dec tools can generate useful reports that explain each error and provide solutions to fix these flaws. SARD test suite covers only a limited list of flaws mapped to CWE or OWASP top ten.

7. CONCLUSION AND FUTURE WORK

In this paper, we introduced our study that evaluates PHP static analyzers tools and reported our findings. The main target of this effort is to introduce the different types of static analysis tools and techniques for more secure web applications. Some static analyzers can be integrated into the development environment for quality assurance. Automatic static analyzers tools are not a replacement to code reviews by security experts. It is an aiding tool to find code defects and to introduce solutions for resolving security flaws. Developers must carefully choose the appropriate tools that support the application's language and that match the application's vulnerabilities. The availability of standardized testing suites for several languages is very important to evaluate such tools. In addition, we discussed the difference between dynamic testing and static testing. We highlighted the importance of static analyzers and their challenges. After that, we discussed software common weaknesses. Our future work is to develop a framework that integrates with different tools. The suggested framework enables testers to first select what vulnerabilities they need to test. After that, the framework will select certain tools that better detect potential flaws. It will generate one consolidated report that explains the detected vulnerabilities. Also, it provides the best practices and suggested solutions to avoid found flaws. The main benefit of this framework is to avoid using multiple tools and generating too many alarms. Additionally, is to gain the maximum benefits of multiple tools, since each tool excels in detecting certain vulnerabilities.

8. REFERENCES

[1] NIST-relative-cost-to-fix-a-flaw, NowSecure. 09-May-2017. [Online]. Available: https://www.nowsecure.com/blog/2017/05/10/level-up-mobile-app-security-metrics-to-measure-success/nist-relative-cost-to-fix-a-flaw/. [Accessed:18-Jul-2019].

[2] Source code analysis tools - OWASP, Owasp.org. 2019. [Online].Available:https://www.owasp.org/index.php/Source_Code_Analysis_Tools. [Accessed: 09- Jul- 2019]

[3] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, A. Zaidman and H. C. Gall. Context is king: The developer perspective on the usage of static analysis tools. *2018* IEEE *25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Campobasso, 2018, pp. 38-49. DOI=10.1109/SANER.2018.8330195

[4] R. Mahmood and Q. Mahmoud. Evaluation of static analysis tools for finding vulnerabilities in java and C/C++ source code. *2018 CoRR Journal*, vol: abs/1805.09040.*DOI*=http://arxiv.org/abs/1805.09040

[5] K. Heo, H. Oh and K. Yi. Machine-learning-guided selectively unsound static analysis. *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), Buenos Aires*, 2017, pp. 519-529. DOI=10.1109/ICSE.2017.54

[6] Chi Li, Min Zhou, Zuxing Gu, Guang Chen, Yuexing Wang, Jiecheng Wu, Ming Gu. VBSAC: A value-based static analyzer for C. *2019 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '19)*, ACM, New York, NY, USA, 382-385. DOI= https://doi.org/10.1145/3293882.3338998

[7] A. Arusoaie, S. Ciobâca, V. Craciun, D. Gavrilut and D. Lucanu. A comparison of open-source static analysis tools for vulnerability detection in C/C++ code.*2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC). Timisoara, 2017*, pp. 161-168. DOI=10.1109/SYNASC.2017.00035

[8] A. Fatima, S. Bibi, and R. Hanif. Comparative study on static code analysis tools for C/C++. 2018 15th International Bhurban Conference on Applied Sciences and Technology (IBCAST), Islamabad, 2018, pp. 465-469. DOI= 10.1109/IBCAST.2018.8312265

[9] Herter J., Kästner D., Mallon C., Wilhelm R. Benchmarking static code analyzers.2017 Tonetta S., Schoitsch E., Bitsch F. (eds) Computer Safety, Reliability, and Security. SAFECOMP 2017. Lecture Notes in Computer Science, vol 10488. Springer, Cham. DOI=https://doi.org/10.1007/978-3-319-66266-4_13

[10] A. S. Novikov, A. N. Ivutin, A. G. Troshina and S. N. Vasiliev. The approach to finding errors in program code based on static analysis methodology.*2017 6th*

Mediterranean Conference on Embedded Computing (MECO), Bar, 2017, pp. 1-4. DOI= 10.1109/MECO.2017.7977127

[11] Fromherz A., Ouadjaout A., Miné A. Static value analysis of python programs by abstract interpretation. *2018 Dutle A., Muñoz C., Narkawicz A. (eds) NASA Formal Methods. NFM. Lecture Notes in Computer Science, vol 10811*. Springer, Cham. DOI=https://doi.org/10.1007/978-3-319-77935-5_14

[12] Roy S., Chaulagain D., Bhusal S. Static Analysis for security vetting of android apps. 2018 Samarati P., Ray I., Ray I. (eds) From Database to Cyber Security. Lecture Notes in Computer Science, vol 11170. Springer, Cham. DOI=https://doi.org/10.1007/978-3-030-04834-1_19

[13] Bielik P., Raychev V., Vechev M. Learning a static analyzer from data. 2017 Majumdar R., Kunčak V. (eds) Computer Aided Verification. CAV 2017. Lecture Notes in Computer Science, vol 10426. Springer, Cham. DOI=https://doi.org/10.1007/978-3-319-63387-9_12

[14] Oyetoyan T.D., Milosheska B., Grini M., Soares Cruzes D. Myths and Facts about Static Application Security Testing Tools: An Action Research at Telenor Digital. *Agile Processes in Software Engineering and Extreme Programming. XP 2018. Lecture Notes in Business Information Processing, vol 314*. Springer, Cham. *DOI*=https://doi.org/10.1007/978-3-319-91602-6_6

[15] R. Shaukat, A. Shahoor and A. Urooj. Probing into code analysis tools: a comparison of C# supporting static code analyzers.*2018 15th International Bhurban Conference on Applied Sciences and Technology (IBCAST), Islamabad, 2018,* pp. 455-464. DOI=10.1109/IBCAST.2018.8312264

[16] Belevantsev A., Avetisyan A. Multi-level Static Analysis for Finding Error Patterns and Defects in Source Code. 2018 Petrenko A., Voronkov A. (eds) Perspectives of System Informatics. PSI 2017. Lecture Notes in Computer Science, vol 10742. Springer, Cham. DOI=https://doi.org/10.1007/978-3-319-74313-4_3

[17] Koshelev, V.K., Ignatiev, V.N., Borzilov, A.I., et al. SharpChecker: Static analysis tool for C# programs. *2017 Program Comput Soft, vol 43, pp. 268-276.* Springer, Cham. *DOI=* https://doi.org/10.1134/S0361768817040041

[18] Shastry B. et al. Static program analysis as a fuzzing aid.2017 Dacier M., Bailey M., Polychronakis M., Antonakakis M. (eds) Research in Attacks, Intrusions, and Defenses. RAID 2017. Lecture Notes in Computer Science, vol 10453. Springer, Cham. DOI=https://doi.org/10.1007/978-3-319-66332-6_2

[19] A. S. Novikov, A. N. Ivutin, A. G. Troshina and S. N. Vasiliev. Detecting the use of unsafe data in software of embedded systems by means of static analysis methodology. *2018 7th Mediterranean Conference on Embedded Computing (MECO), Budva, 2018,* pp. 1-4. DOI=10.1109/MECO.2018.8406025

[20] Z. P. Reynolds, A. B. Jayanth, U. Koc, A. A. Porter, R. R. Raje, and J. H. Hill. Identifying and documenting false positive patterns generated by static code analysis tools. 2017 IEEE/ACM 4th International Workshop on Software Engineering Research and Industrial Practice (SER&IP), Buenos Aires, 2017, pp. 55-61.DOI=10.1109/SER-IP.2017.20

[21] Siavvas M., Gelenbe E., Kehagias D., Tzovaras D. Static analysis-based approaches for secure software development. Security in Computer and Information Sciences. Euro-CYBERSEC 2018. Communications in Computer and Information Science, vol 821. Springer, Cham. DOI= https://doi.org/10.1007/978-3-319-95189-8_13

[22] Tukaram Muske, Rohith Talluri, and Alexander Serebrenik. Repositioning of static analysis alarms. *27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2018).* ACM, New York, NY, *USA,* 187-197. DOI=https://doi.org/10.1145/3213846.3213850

[23] Eric Bodden. 2018. Self-adaptive static analysis. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results* (ICSE-NIER '18). ACM, New York, NY, USA, 45-48. DOI=https://doi.org/10.1145/3183399.3183401

[24] CWE - CWE-79: Improper neutralization of input during web page generation ('Cross-site Scripting') (3.3). Cwe.mitre.org, 2019. [Online]. Available: https://cwe.mitre.org/data/definitions/79.html. [Accessed: 09- Jul- 2019]

[25] CWE - CWE-862: Missing Authorization (4.2)", Cwe.mitre.org, 2020. [Online]. Available: https://cwe.mitre.org/data/definitions/862.html. [Accessed: 09- Dec- 2020].

[26] CWE - CWE-89: Improper neutralization of special elements used in an SQL command ('SQL Injection') (3.3).Cwe.mitre.org, 2019. [Online]. Available: http://cwe.mitre.org/data/definitions/89.html. [Accessed: 09- Jul- 2019]

[27] CWE - CWE-209: Information Exposure through an Error Message (3.3). Cwe.mitre.org, 2019. [Online]. Available: https://cwe.mitre.org/data/definitions/209.html. [Accessed: 09- Jul- 2019]

[28] Path traversal vulnerability | CWE-22 Weakness | Exploitation and remediation. Immuniweb.com, 2019. [Online]. Available: https://www.immuniweb.com/vulnerability/path-traversal.html. [Accessed: 09- Jul- 2019]

[29] CWE - CWE-502: Deserialization of untrusted data (3.4.1).Cwe.mitre.org, 2019. [Online]. Available: https://cwe.mitre.org/data/definitions/502.html. [Accessed: 25- Sep- 2019]

[30] Using static analysis in program development.Viva64.com, 2019. [Online]. Available: https://www.viva64.com/en/a/0017/. [Accessed: 10- Oct- 2019].

[31] Data flow analysis in compiler – GeeksforGeeks. GeeksforGeeks. 2019. [Online]. Available: https://www.geeksforgeeks.org/data-flow-analysis-compiler/. [Accessed: 10- Oct- 2019].

[32] Software Engineering | Control flow graph (CFG) – GeeksforGeeks. GeeksforGeeks. 2019. [Online]. Available: https://www.geeksforgeeks.org/software-engineering-control-flow-graph-cfg/. [Accessed: 10- Oct- 2019].