International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-9 Issue-1, October, 2019

Evaluation of SQL Injection Vulnerability Detection Tools

Najla'aAteeq Mohammed Draib, Abu Bakar Md Sultan, Abdul Azim B Abd Ghani, HazuraZulzalil

Abstract: ***SQL injection vulnerabilities have been predominant on database-driven web applications since almost one decade. Exploiting such vulnerabilities enables attackers to gain unauthorized access to the back-end databases by altering the original SQL statements through manipulating user input. Testing web applications for identifying SQL injection vulnerabilities before deployment is essential to get rid of them. However, checking such vulnerabilities by hand is very tedious, difficult, and time-consuming. Web vulnerability static analysis tools are software tools for automatically identifying the root cause of SQL injection vulnerabilities in web applications source code. In this paper, we test and evaluate three free/open source static analysis tools using eight web applications with numerous known vulnerabilities, primarily for false negative rates. The evaluation results were compared and analysed, and they indicate a need to improve the tools.***

Keywords: Web Application, Static Analysis Tools, SQL Injection, Vulnerabilities

I.INTRODUCTION

Web applications are frequently deployed with different types of security vulnerabilities that can be uncovered and maliciously exploited by attackers Medeiros et al. (2016), Kieyzun et al. (2009), D. Kaur and Kaur (2016), Cova et al. (2007). Thereupon, due to the presence of vulnerabilities, their high global exposure, and the important assets they usually store, web applications are being attractive and ideal targets for security attacks.Among web application security attacks, Structured Query Language Injection Attacks (SQLIAs) have consistently been top-ranked for the past few years, as eventually specified by OWASP (2017), SANS Institute (2011), and TRUSTWAVE (2018). SQLIA is a notorious technique that exploits vulnerability and occurs in the database layer of a web application. Basically, it takes advantages of SQL Injection Vulnerabilities (SQLIVs) in the input validation of submitted requests in server-side program which interacts with the database server. In this type of attack the web application is designed in a way which consumes the user input data for generating dynamic SQL queries. SQL Injections (SQLIs) are introduced to vulnerable web application in two main types, based on input mechanism, namely; first-order SQLI and second-order SQLI. Learning about these types and how to detect and prevent them is very important to insure secure web applications. Brief explanation of SQL injection types and detection/and prevention methods can be found in these studies Shehu and Xhuvani (2014), Sharma and Jain (2014), William G. J. Halfond, Jeremy Viegas (2008), P. Kaur and Kour (2016), Gomaa et al. (2016).The main consequences of successful SQLIA include loss of confidentiality, authentication, and integrity of information in the database Mishra et al. (2014), Johari and Sharma (2012), Singh et al. (2016).Considering SQLI is one of the major threats today to web applications, getting tools that can identify them so audients/developers can remove or fix them early before web applications are deployed is one of essential ways to get rid of them.

Static analysis approaches are the most widely used techniques for identifying potential SQLIVs in web application source code. They, basically, scan the program texts to find the root cause of a vulnerability, early, during testing phase of web applications' development, even before the program is run for the first time. Static analysis techniques are usually white-box testing approaches, because they have access to the source code, so they identify vulnerabilities by tracking the flow of data coming from tainted sources throughout the code and detect if they reach security-critical function.

The beauty of static analysis techniques hides in automatically finding errors early in the development phase as it not only minimises the cost of fixing the errors, but also the coding approach of the developer is improved due to the quick feedback received. Another advantage of using static analysis is its ability to provide better coverage compared to dynamic analysis. A considerable amount of research has been conducted on detecting SQLIVs as an advance solution for mitigating the devastating problem of SQLIAs. However, static analysis vulnerability detection techniques are suffering from generating many false positives (i.e., not actual vulnerabilities) and, in some cases, they are not capable of detecting all types of SQLIVs that exist Draib et al. (2018). This signifies the urgent need for more research towards identifying the limitations of existing static analysis approaches for detecting SQLIVs which in turn would shed lights on any rooms for improving static analysis underling techniques.

Revised Manuscript Received on October 30, 2019.
* Correspondence Author
Najla'a Ateeq Mohammed Draib, Dept. of Software Engineering and Information System, Faculty of Computer Science and Information Technology Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia
Abu Bakar Md Sultan, Dept. of Software Engineering and Information System, Faculty of Computer Science and Information Technology Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia
Abdul Azim B Abd Ghani, Dept. of Software Engineering and Information System, Faculty of Computer Science and Information Technology Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia
Hazura Zulzalil, Dept. of Software Engineering and Information System, Faculty of Computer Science and Information Technology Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia

Several studies have been done on evaluation of the web application vulnerability detection tools Makino and Klyuev (2015), Amankwah et al. (2017), H. Kaur and Jai (2015), Tajpour and Shooshtari (2010), Suteva et al. (2013). However, evaluation/comparing of open source web application vulnerability detection tools to identify their capabilities and limitations in detecting SQL injection vulnerabilities in web application source code have never been studied so far. There exist three open source static analysis tools called WAP Medeiros et al. (2016), RIPS Dahse (2010), and PIXY Jovanovic et al. (2006) to analyse PHP web applications' source code for SQLIVs. To the best of our knowledge, WAP, RIPS and PIXY are the most cited and effective open source PHP static analysis tools in the literature. RIPS has upgraded into a new version which is more precise and covers more types of SQL injection, but it is not freely available Dahse (2015). On many situations students and researchers cannot afford to buy the professional version of such tools. Moreover, using open source static analysis tools help developers in speeding up the development process and reducing its cost. Thus, it would be very helpful to find out how well the free versions detect SQLIVs.

In this paper, we evaluate these three static analysis tools. These are open source, automated and multi-platform software. Evaluation of these vulnerability detection tools is very key as it helpsreserchers to: (1) assess the tools' capabilities for detection of SQL injection vulnerabilities (2)evaluate the weakness and strength of the tools for detection (3) expand the overmentioned tools functionality. The rest of this paper is structured as follows. The comparison method is presented in Section I, including details of the tested web applications and the general characteristics of the considered tools. Section II describes the methodology, while Section III explains the results and discussion. Finally, Section IV provides concluding remarks and mentions directions of future research.

II.COMPERISON METHOD

Vulnerable web application

For evaluating the static analysis tools, eight open-source vulnerable web applications are tested against SQLIVs. The applications are namely DVWA 1.9, Schoolmate 1.4.5, WackoPicko, WebChess 1.9, FAQforge 1.3.2, myBloggie2.1.4, EVE 1.0 and Sqli-labs. Wackopicko and DVWA were run in the OWASP Broken Web Application Project virtual machine OWASP (2018), that has a collection of intentionally vulnerable web applications. WackoPicko and DVWA are developed with intended number of SQLIVs that have been determined and known to the public. For the rest of the selected applications, the vulnerabilities information is collected from their advisories as shown in Table 1. All these applications have been used in previous works in evaluating vulnerability detection tools and approaches Makino and Klyuev (2015), Medeiros et al. (2016), Beatriz (2014), Ping (2018). Table 1 presents more details about these applications.

Table. 1 Vulnerable open source web application used in our experiment

Application	Version	Total Files	Description	Security advisory
DVWA	1.9	734	Damn Vulnerable Web Application	https://www.slideshare.net/soham28/dvwadamn-vulnerabilities-web-application
Schoolmate	1.4.5	64	School admin system	groups.csail.mit.edu/pag/ardilla/CVE-2010-5011
WackoPicko	Master	97	Photo sharing and photo-purchasing site	https://github.com/adamdoupe/WackoPicko https://www.aldeid.com/wiki/WackoPicko
WebChess	0.9	35	Online chess game	Bugtraq ID:43895 http://groups.csail.mit.edu/pag/ardilla/
FAQforge	1.3.2	20	FAQ management tool	Bugtraq-43897 http://groups.csail.mit.edu/pag/ardilla/
EVE	1.0	9	Player activity tracker for an online game	Bugtraq ID:15389 http://groups.csail.mit.edu/pag/ardilla/
Sqli-labs	Master	302	A platform to learn SQLI	http://dummy2dummies.blogspot.com/2012/06/sqli-lab-series-part-1.html
Mybloggie	2.1.4	73	Weblog system	CVE-2006-4042

Tested Static Security Analysis Tools

Table 2 lists the general characteristics of the static analysis tools used in this study. All these tools are open source, fully automated, and cross-platform software. These vulnerability detection tools are obtained from GetHup and SourceForge. WAP and PIXY are written in Java whereas Rips is written in PHP. All have no graphical user interface except RIPS which comes with a graphical interface that provides users with more options and information. WAP is based on combined taint analysis with data mining to reduce the rate of false positives. It uses the taint flow analysis to track the spread of untrusted input through an application. It then uses data mining to predict the existence of false positives within results Medeiros et al. (2016). RIPS builds control flow graphs and then creates block and function

summaries by simulating the data flow for each basic block, which allows to conduct a precise taint analysis. PIXY performs tainted and alias analysis, it takes a PHP program as input and creates a report that lists possible vulnerable points in the program, together with additional information for understanding the vulnerability (e.g. dependent graphs and control flow graphs of the detected paths). PIXY and RIPS are not prepared for OOP code analysis.

Table. 2 General characteristics of the evaluated tools

	RIPS	PIXY	WAP
Company/ Creator	Rips technology	The Secure Systems Lab at the Technical University of Vienna	OWASP
Version	V0.55	V3.0.3	V2.1
License	GPL v3	GPL V3	GNU Affero General Public License 3.0
Operating system	Windows/Linux/Mac	Windows/Linux	Linux/Macintosh/Windows.
PHP version supported	V3-4, NO OOP	V4-5, NO OOP	V4 or higher
Supported input vectors	$_GET, $_POST, $_COOKIES, $_REQUEST	$_GET, $_POST, $_COOKIES	$_GET, $_POST, $_COOKIE, $_REQUEST
Requirement	Web server, PHP, browser	JRE	JRE
Source	https://sourceforge.net/projects/rips-scanner/files/	https://github.com/oliverklee/pixy	http://awap.sourceforge.net/

III.METHODOLOGY

We tested the aforementioned static analysis tools against the given above vulnerable web applications for detecting SQLIVs. Input arguments for WAP and RIPS were the whole path of the vulnerable web application. PIXY accepts one file to be analysed thus we used the index.php file for every web application, for sqli-labs we analysed the lessons separately.

The main goal of this experiment is to evaluate static analysis tools and identify their limitations in detecting these vulnerabilities. We concentrate on false negative (i.e., missed vulnerabilities detection) rate to analyse the capabilities of the tools. However, considering false positive rate was challenged due to the limited details about the real SQLIVs in the applications as previous works only reported the number of detected vulnerabilities, but not the vulnerabilities as such.

In order to identify false negative rates and number of known vulnerabilities that are successfully detected by the examined tools, we compared the results obtained from the experiment with those reported before by the security advisory of the vulnerable applications.

IV.RESULTS AND DISCUSSION

The numbers of known SQLIVs found and the numbers of false negatives are given in Table 3. All examined tools have reported very high false negative rates. Running from 35.2% for PIXY and 44% for WAP to 54.1% for RIPS. PIXY and RIPS suffer from an overhead performance, compared to WAP. Evedently, PIXY have showed a low overhead performance to scan all applications but it requires 38 seconds for scanning Mybloggie 2.1.4. This inconsistency is due to using object-oriented programing while developing the application. In fact, PIXY does not support PHP 5.0 object-oriented features, thus it did not manage to process DVWA and WackoPicko. Likewise, RIPS attached a message “Code is object-oriented. This is not supported yet and can lead to false negatives” with the results of scanning Mybloggie, WebChess, DVWA, and WackoPicko. WAP missed some vulnerabilities where $_SESSION array is used, this is could be because WAP does not consider input vectors passed to the query through $_SESSION array. Compared with WAP and PIXY, RIPS shows more informative results which spares out the need of reviewing the whole code, especially in penetration testing context.

Regarding stored SQL injection, all examined tools failed to detect them as they do not track tainted data through to the database. This signifies the need to formulate static analysis mechanism for identifying second-order SQLIVs.

Table. 3 Number of SQLIVs of the evaluated static analysis tools detected in the applications

	RIPS				WAP				PIXY			
	SQLI	KVD	FN	AT (seconds)	SQLI	KVD	FN	AT (seconds)	SQLI	KVD	FN	AT (seconds)
Schoolmate1.4.5	186	5	1	1.892	69	3	3	0.389	170	6	0	05
DVWA-1.9	5	3	3	20.789	2	2	4	02	-	-	-	-
Wackopicko	3	1	1	1.477	0	0	2	0.562	-	-	-	-
Webchess	29	5	7	5.379	3	3	9	0.418	0	0	12	0
Faqforg	0	0	1	0.256	0	0	1	0.273	1	1	0	0
Sql-labs	44	28	41	0.716	54	47	22	0.635	52	51	18	-
EVE 1.0	8	2	0	0.634	1	1	1	01	6	1	1	0
Mybloggie 2.1.4	1	0	1	5.109	0	0	1	0.460	1	0	1	38
Total	276	44	55	36.252	129	56	44	5.737	230	59	32	43

KVD: Detected Known Vulnerabilities,
FN: False Negative,
AT: Analyse Time

V.CONCLUSION

In this paper, we evaluated and compared three open source static analysis tools in terms of their ability to detect SQLIVs. Regarding the results, WAP is superior over RIPS and PIXY with less overhead performance. Wherease, Pixy have the lowest number of false negatives. However, it should be noted that all the examined tools are not perfect in detecting SQLIVs. The tools ability should be improved for detecting all types of SQLIVs. Moreover, we realize that there is a need for a standard benchmark with complete information about the real existing/known vulnerabilities and their exploits. Indeed, because of the lack of information about the real vulnerabilities we did not give the false positive rates of the tools, we only focus on false negative rates.In our future work we plan to expand the tool set that we investigate and compare to involve both open-source and commercial static analysis tools.

REFERENCES

1. Amankwah, R., Kwaku, P., & Yeboah, S. (2017). Evaluation of Software Vulnerability Detection Methods and Tools: A Review. *International Journal of Computer Applications*, *169*(8), 22–27. https://doi.org/10.5120/ijca2017914750
2. Beatriz, M. F. (2014). Automatic Detection of Vulnerabilities in Web Applications using Fuzzing. *Instituto Superior Tecnico*, 1–10. Retrieved from https://fenix.tecnico.ulisboa.pt/downloadFile/563345090413029/ExtendedAbstract-MEICA-67039-MiguelBeatriz.pdf
3. Cova, M., Felmetsger, V., & Vigna, G. (2007). Vulnerability analysis of Web-based applications. *Test and Analysis of Web Services*, 363–394. https://doi.org/10.1007/978-3-540-72912-9_13
4. Dahse, J. (2010). RIPS-A static source code analyser for vulnerabilities in PHP scripts. *Retrieved: February*, *28*, 2012. Retrieved from https://www.nds.ruhr-uni-bochum.de/media/nds/attachments/files/2010/09/rips-paper.pdf
5. Dahse, J. (2015). RIPS - free PHP security scanner using static code analysis. Retrieved April 8, 2017, from Rips-scanner.sourceforge.net website: http://rips-scanner.sourceforge.net/#download
6. Draib, N. A. M., Sultan, A. B. M., Ghani, A. A. B. A., & Zulzalil, H. (2018). Comparison of Security Testing Approaches for Detection of SQL Injection Vulnerabilities. *International Journal of Engineering & Technology*, *7*(4.1), 14–17. https://doi.org/10.14419/ijet.v7i4.1.19483
7. Gomaa, Y., Ahmed, A. E. A., Mahmood, M. A., & Hefny, H. (2016). Survey on securing a querying process by blocking SQL injection. *Proceedings of 2015 IEEE World Conference on Complex Systems, WCCS 2015*, 1–7. https://doi.org/10.1109/ICoCS.2015.7483271
8. Johari, R., & Sharma, P. (2012). A survey on web application vulnerabilities (SQLIA, XSS) exploitation and security engine for SQL injection. *Proceedings - International Conference on Communication Systems and Network Technologies, CSNT 2012*, (May 2012), 453–458. https://doi.org/10.1109/CSNT.2012.104
9. Jovanovic, N., Kruegel, C., & Kirda, E. (2006). Pixy: A static analysis tool for detecting web application vulnerabilities (Short paper). *Proceedings - IEEE Symposium on Security and Privacy*, *2006*, 258–263. https://doi.org/10.1109/SP.2006.29
10. Kaur, D., & Kaur, P. (2016). Empirical Analysis of Web Attacks. *Physics Procedia*, *78*(December 2015), 298–306. https://doi.org/10.1016/j.procs.2016.02.057
11. Kaur, H., & Jai, P. (2015). Comparing Detection Ratio of Three Static Analysis Tools. *International Journal of Computer Applications*, *124*(13), 35–40. https://doi.org/10.5120/ijca2015905749
12. Kaur, P., & Kour, K. P. (2016). SQL injection: Study and augmentation. *Proceedings of 2015 International Conference on Signal Processing, Computing and Control, ISPCC 2015*, 102–107. https://doi.org/10.1109/ISPCC.2015.7375006
13. Kieyzun, A., Guo, P. J., Jayaraman, K., & Ernst, M. D. (2009). Automatic creation of SQL Injection and cross-site scripting attacks. *2009 IEEE 31st International Conference on Software Engineering*, 199–209. https://doi.org/10.1109/ICSE.2009.5070521
14. Makino, Y., & Klyuev, V. (2015). Evaluation of web vulnerability scanners. *Proceedings of the 2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2015*, *1*(September), 399–402. https://doi.org/10.1109/IDAACS.2015.7340766
15. Medeiros, I., Neves, N., & Correia, M. (2016). Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining. *IEEE Transactions on Reliability*, *65*(1), 54–69. https://doi.org/10.1109/TR.2015.2457411
16. Mishra, N., Chaturvedi, S., Sharma, A. K., & Choudhary, S. (2014). XML-based authentication to handle SQL injection. *Advances in Intelligent Systems and Computing*, *236*, 739–749 https://doi.org/10.1007/978-81-322-1602-5_79
17. OWASP. (2017). Top 10-2017 Top 10 - OWASP. Retrieved February 25, 2017, from https://www.owasp.org/index.php/Top_10-2017_Top_10
18. OWASP. (2018). OWASP Broken Web Applications Project - OWASP. Retrieved March 23, 2019, from https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project#tab=Main
19. Ping, C. (2018). A second-order SQL injection detection method. *Proceedings of the 2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference, ITNEC 2017*, *2018-Janua*, 1792–1796. https://doi.org/10.1109/ITNEC.2017.8285104
20. SANS Institute. (2011). CWE/SANS TOP 25 Most Dangerous Software Errors. Retrieved February 1, 2019, from https://www.sans.org/top25-software-errors/
21. Sharma, C., & Jain, S. C. (2014). Analysis and classification of SQL injection vulnerabilities and attacks on web applications. *2014 International Conference on Advances in Engineering and Technology*

Research, ICAETR 2014. https://doi.org/10.1109/ICAETR.2014.7012815

22. Shehu, B., & Xhuvani, A. (2014). A Literature Review and Comparative Analyses on SQL Injection : Vulnerabilities , Attacks and their Prevention and Detection Techniques. *IJCSI International Journal of Computer Science Issues*, *11*(4), 28–37.
23. Singh, N., Dayal, M., Raw, R. S., & Kumar, S. (2016). SQL Injection: Types, Methodology, Attack Queries and Prevention. *2016 International Conference on Computing for Sustainable Global Development (INDIACom)*, 2872–2876.
24. Suteva, N., Zlatkovski, D., & Mileva, A. (2013). Evaluation and Testing of Several Free / Open Source Web. *The 10th Conference for Informatics and Information Technology*, (Ciit), 221–224.
25. Tajpour, A., & Shooshtari, M. J. Z. (2010). Evaluation of SQL injection detection and prevention techniques. *Proceedings - 2nd International Conference on Computational Intelligence, Communication Systems and Networks, CICSyN 2010*, (July 2010), 216–221. https://doi.org/10.1109/CICSyN.2010.55
26. TRUSTWAVE. (2018). *Trustwave Global Security Report Introduction the State of Security*. Retrieved from https://www2.trustwave.com/rs/815-RFM-693/images/Trustwave_2018-GSR_20180329_Interactive.pdf
27. William G. J. Halfond, Jeremy Viegas, R. O. (2008). A Classification of SQL Injection Attacks and Countermeasures. *Preventing Sql Code Injection By Combining Static and Runtime Analysis*, 53. https://doi.org/doi=10.1.1.95.2968