Efficiency and effectiveness of web application vulnerability detection approaches: A Minified Literature Review

Minified version of the paper by Bing Zhang, Jingyue Li, Jiadong Ren, and Guoyan Huang [23].

PEDER GRUNDVOLD, Norwegian University of Science and Technology, Norway

There are some existing studies reviewing different Web Application Vulnerability Detection (WAVD) approaches, but most of these focus on analysing a specific vulnerability or comparing a few related approaches. Because of this Bing Zhang et al. [23] found the need for a general systematic analysis reviewing all existing WAVD approaches.They reviewed 105 WAVD papers from January 2008 to June 2019, but since this is an area of science with a rapid progress we wanted to update their research with the most recent papers.

We will therefore use the same approach to review another, smaller set, of WAVD papers published between July 2019 and March 2020. In the selected papers we identified six of the nine categories of web attributes analyzed that were categorized by Bing Zhang et al. We also identified five of the nine categories of approaches for analyzing the web attributes from Bing Zhang et al. We then summarize the efficiency and effectiveness of different WAVD approaches on detecting several types of input validation vulnerabilities. Hopefully this can work as a supplement to the article by Bing Zhang et al. and also help both researchers and security engineers to better understand the current state of WAVD approaches.

1 INTRODUCTION

When analysing web applications security engineers often make use of web application vulnerability detection (WAVD) tools. There exists a lot of these tools, with different use cases as well as different approaches, e.g. machine learning [1, 8], dynamic taint analysis [4] or scan-and-inject [7]. The standard awareness document OWASP Top 10 [10] categorize the most common web vulnerabilities into injection, broken authentication, sensitive data exposure, XML external entities (XXE), broken access control, security misconfiguration, cross-site scripting (XSS), insecure decentralization, using components with known vulnerabilities and insufficient logging and monitoring. The existing WAVD tools tackle these vulnerabilities in different ways and with a different degree of success. While studies have reviewed WAVD approaches before, there are no studies that provide an empirical evaluation with regards to their efficiency and effectiveness.

To help in the future research by identifying gaps in the field, as well as providing an overview for security engineers, we summarize existing tools and evaluate their empirical results. The literature in review are articles published from July 2019 to March 2020. Our study attempted to answer three research questions, similar to the original study by Bing Zhang et al. [23]:

- RQ1: What kinds of web application attributes have been analyzed by the current WAVD approaches and how were the web attributes analyzed?

- RQ2: How well were the web attributes analyzed to detect web application vulnerabilities?
- RQ3: Which web applications and test suites have been used to provide empirical evaluation results?

The process consisted of first identifying ten articles through keyword searching using the Oria.no search engine [24] and Google Scholar. We also had to filter out articles without the necessary information needed for the evaluation.

The rest of this paper is organized in the following way. Section 2 will provide a brief summary of web application vulnerabilities. Then section 3 will explain the research design and implementation. Section 4 will present our results with regards to RQ1, RQ2 and RQ3. Section 5 discusses these results, and Section 6 concludes and gives recommendations for future work.

2 WEB APPLICATION VULNERABILITIES AND SECURITY ANALYSIS PROCESSES

OWASP [10] classifies web application vulnerabilities into ten categories. More generally it can ble classified into input validation-, session management- and application logic vulnerabilities [11, 12].

2.1 Web application vulnerabilities

Input validation vulnerability happens when web applications do not have sufficient control over user input. Input validation refers to how your application filters, scrubs, or rejects input before additional processing [13]. This sanitation is essential to security. Without this user interfaces like input fields can be exploited by injection malicious commands that in turn will be received by the server. Common injection attacks include XSS, SQLi, XMLi, XPathi, LDAPi and OS command injection [22].

Session management vulnerabilities arise when web applications do not handle user sessions correctly. Sessions allow for users to be authenticated once and then being able to freely browse the application with the authenticated privileges. This is achieved with session tokens (i.e. session_id). Common mistakes are putting session ID’s in the URL or faulty application session timeouts. Typical attacks include session fixation [14], session sniffing[14], CSRF [15], and clickjacking [16, 17].

Faulty authentication of users makes web applications vulnerable to application logic (AL) vulnerabilities which allows malicious users to access restricted information and get elevated privileges. Common attacks include parameter manipulation [18], weak access control [19, 20], and workflow bypass [21].

3 RESEARCH DESIGN AND IMPLEMENTATION

To compare the efficiency and effectiveness of the different WAVD approaches we classified them by investigating their processes (steps) and the web attributes they used. Then we analyzed the WAVD approaches, and the vulnerabilities they focused on. After this we compare each fine-grained category of WAVD approaches on different kinds of vulnerabilities. Finally, we identified the possible resources for evaluating the efficiency and effectiveness of the WAVD approaches. The papers covered are WAVD related articles published between August 2019 and April 2020.

3.1 Searching and filtering the primary studies

We started searching using the Oria.no search engine [24] with different search strings. After this we searched using Google Scholar, but this is harder since many of the articles found here are protected behind payment walls that NTNU have no access to. Most papers originate from popular scientific publication databases like IEEE, Elsevier and SpringerLink. The search strings are listed in Table 1. After sorting out articles that did not meet the inclusion criteria and removing others based on the exclusion criteria (see Table 2) we obtained **6** papers. Since the publishing period we are interested in is so small we did not use a backward snowballing search. After trying this on the first papers we found that almost all papers in the reference lists were older than August 2019 so this was not effective. However applying a forward snowballing search was successful. We reviewed all articles citing the first obtained papers using "Google scholar-search within citing articles" in a recursive way and managed to find **2** more articles.

Table 1: Search string

Key words	Search string
web, website, web application web application vulnerability, web vulnerability, vulnerability detecting, scanning, detection, autonomous, discovery, tool	(web OR website OR web application) AND (web application vulnerability OR web vulnerability OR vulnerability) AND (detecting OR scanning OR detection OR autonomous OR discovery OR tool)

Table 2: Inclusion Criteria and Exclusion Criteria

Inclusion Criteria	Exclusion Criteria
1. Paper related to WAVD; 2. Papers published between August 2019 and April 2020 (some papers originally published earlier are included since they are part of a later journal issue [2, 7, 27]); 3. Paper written in English; 4. Paper comes from peer-reviewed journals or conferences.	1. Secondary studies; 2. Duplicated studies; 3. Paper with many versions (We choose only the latest one and excluded early version); 4. Papers we cannot find the full text or papers was withdrawn; 5. Papers are relevant to vulnerability detection but not about web applications; 6. Papers we cannot extract the needed information to answer our research questions (e.g. analysis objects and analysis). 7. Papers behind payment walls that NTNU does not have access to

4 RESULTS

4.1 Basic information of the primary studies

Seven of the ten identified primary papers are from journals, the rest is conference papers. Most of the WAVD approaches were based on PHP and/or Java, while some used Python for machine learning.

4.2 Results of RQ1

Before presenting the results of RQ1, we introduce the different attribute categories and WAVD approaches. Similar to the research questions, these are directly copied from Bing Zhang et al. [23]. However, some categories have been left out since they were not represented in the papers we reviewed.

The classification of the web attributes the WAVD analyses. Five articles focused on analysing the source code or intermediate data derived from source code. They can be classified into four categories.

- **S1. Models derived from source code.** Some WAVD analyzed the control flow models (e.g., Control Flow Graph (CFG), Call Graph (CG), and Data Dependence Graph (DDG)), data flow models (e.g., Data flow (DF) and Information flow (IF)), or syntax model (e.g., Abstract syntax tree (AST)) of the web application. For example Guo et al. [30] search for exploit points in the source code and then build CFG and DFG which in turn help generating vulnerable code slices. As part of their solution Nguyen et al. [27] use PHP-Parser [33] to generate an AST.

- **S3. Application entry points identified from crawling the source code.** Some vulnerability scanners, such as [16], crawl all websites first to identify all possible application entry points (AEPs, such as inputs and SQL statements) and then send normal, malicious or incorrect parameter values to the AEPs to detect vulnerabilities. Aliero et al. [7] uses a HTTP fetcher to download all pages related to a specific URL and then crawls these to identify SQL Injection Vulnerabilities (SQLIV).

- **S4. Patterns or rules derived from secure code.** Some WAVD approaches generate patterns or rules from a secure web application or legitimate use of the web application and then identify vulnerabilities by checking whether the new or updated web applications or malicious inputs violate the patterns or rules. For example Stallenberg et al. [26] made use of legitimate XML messages, extracted from functional tests or during software execution in the web application fields, to generate test objects (TO) that were later used in generated JUnit (executable) tests exposing XMLi vulnerabilities.

- **S5. The fingerprint derived from vulnerable code.** Some WAVD approaches, especially vulnerability scanners, search file names, tokens, and function names, are known to be vulnerable. Nguyen et al. [27] proposes a white-box approach that first scans the entire source code and then uses taint analysis in generated AST to detect a wide array of web application vulnerabilities.

Five articles focused on analysing the web application execution or intermediate data derived from running the application. This can be classified into two categories:

.

- **B2. Web page content dynamically generated during application execution.** Some approaches focused on the script, iframe, CSS style, or image tags in the generated web pages to identify possible vulnerabilities inserted into the pages. Fang et al. [8] proposed a reinforced learning approach where training XSS samples are analysed for the specific tags source, svg, img and iframe.

- **B4. Patterns or rules derived from HTTP traffic.** Some studies focused on analyzing the HTTP traffic (such as page input fields, form fields, login input, HTTP GET/POST parameters, HTTP cookies, HTTP user-agents, and referrer header values) between client and server. Calzavara et al. [1] use supervised learning to classify sensitive HTTP requests and then test them for CSRF vulnerabilities.

The classifications of WAVD approaches. Based on what web attributes the WAVD approaches analyze and the detailed analysis process, Bing Zhang et al. [23] classify the WAVD approaches into nine categories. As was the case above, some categories have also here been left out since they were not represented in the papers we reviewed.

- *Match Fingerprint Using Elements Extracted from a Model (MFM).* WAVD methods in this category usually begin by deriving models, e.g., CFG, DDG, AST, browsing behavior models, navigation graphs, and navigation paths. The WAVD approaches then traverse the model to extract code elements to compare with known fingerprints. For example Le et al. [27] propose the vulnerability scanner E-THAPS which uses taint analysis in generated AST to detect web application vulnerabilities.

- **Verify Constraints or Patterns Using Information Extracted from a Model (VCPM).** The methods in this category typically compare the expected behavior (i.e., constraints) with the information of possible behavior that can be extracted from models.

- **Classify Using Property Extracted from a Model (CCPM).** These methods build models first and then extract properties from models to perform classification. To generate suspicious code slices Guo et al. [30] use models like CFG and DFG. These are generated from scanning the web applications for specific code (HTML tags).

- **Generate Attack from Model (GAM).** These methods usually use information in the models to generate or guide the generation of attacks to test the web application. Jan et al. [4] test for XMLi

vulnerabilities with a set of test objectives (TOs) used by generic algorithms to create malformed XML messages.

- **Generate Attack from Constraints or Patterns (GA).** These methods generate attacks or penetration test cases from constraints or attack patterns to determine whether the web application violates the constraints or the attacks succeed. Stallenberg et al. [26] proposed automatic tool with two main phases. The first phase aims to generate malicious XML TOs that, if generated by the front-end web application, can compromise the web server. In the second phase, search algorithms are used to search for user inputs that, upon validation and sanitization, results in malicious XML messages.

4.3 Results of RQ2

The ten studies we covered mainly focused on input validation vulnerabilities. When possible we collected evaluation information like True Positive Rate (TPR), False Positive Rate (FPR), False Negative Rate (FNR) and precision. Getting all these measures are not always easy. For example Calzavara et al. [1] points out that getting a correct FNR would require the knowledge of all vulnerabilities on the tested websites. We also collected information about time- and memory consumption and whether the need of human interaction was necessary or if the process was fully automated. Some papers also had additional effectiveness information like f-measure [3, 7, 25] and code coverage [4].

The efficiency and effectiveness of approaches detecting IPV vulnerabilities. Injection vulnerabilities are top listed in OWASP 2013 and OWASP 2017 and all of our reviewed papers are focusing on this. The detailed efficiency and effectiveness data of various approaches for detecting injection vulnerabilities are included in Table A.1 in Appendix A. The data can be summarized in the following way.

- **XMLi**
  - **GAM.** [4] defines four types of XML attacks, namely deforming, random closing tags, replicating and replacing. They then propose a black-box, co-evolutionary algorithm (COMIX) to search for multiple of these vulnerabilities at the same time. They report a TO code coverage of 100% but have no information about TPR, FNR or FPR.
  - **GA.** [26] introduces JCOMIX, a penetration testing tool that generates XMLi test cases to expose vulnerabilities. It implements various search algorithms, including fuzzing, (GAs), and the co-evolutionary algorithm designed for XMLi testing (COMIX) [4].
- **SQLi**
  - **VCPM.** [2] propose SEPTIC, a system for identifying SQLi- and stored injection attacks. After a training phase it compares incoming queries with the trained query model and achieves 0% FPR and 0% FNR.

  - **GAM.** [3] improved test efficiency of SQLi vulnerabilities by using a random search (fuzz testing). They achieved "more than 26% in reducing the number of SQLi attempts before accomplishing a successful injection" and a precision of 74%.
  - **GA.** [7] propose an algorithm used for black-box testing that crawls all pages of an application to identify SQLi vulnerabilities. It achieves a TPR of 71% and a 100% precision.
- **XSS**
  - **GAM.** [1] trained a model using supervised learning to classify sensitive HTTP requests and then test them for CSRF vulnerabilities. Testing showed a precision of 74%. They also detail vulnerabilities the model was able to find in major production web applications like bombas.com and indeed.com.
  - **AM+CCPM.** Another machine learning approach to detect XSS attacks is proposed by [8]. They alternate between training two models, one adversarial model and one detection model. In this way the system is able to generate its own training data by "fighting against itself". The detection model achieves a 97.90% TPR and 99.50% precision.
- **SQLi and XSS**
  - **MFM.** [27] propose a white-box detection platform where the vulnerability scanner is based on taint analysis and the generation of AST. It will first scan the entire source code and then use taint analysis to detect a wide array of web application vulnerabilities, mainly SQLi and XSS. It achieves a FNR of 11.54%, a TPR of 88.46% and a precision of 90.20%.
  - **CCPM.** Another vulnerability detection system is proposed by [30]. They introduce VulHunter which is a system using deep learning on bytecode to detect vulnerabilities in PHP software. Experiments show a 6.26% FPR, a 2.01% FNR, a 97.99% TPR and a precision of 86.76%.
- **Time consumption.** Only three of the ten studies reported a time consumption. However, since these studies got their empirical results from completely different test suites, no effective comparison would be possible.

- **Effectiveness.** We consider studies effective if they reported exact FPR/FNR values for a given vulnerability detection and also had these values lower than 10%, as they do in Bing Zhang et al. [23]. Only two studies actually reported these numbers. [2] used VCMP with a method that is trained by forcing calls to all queries in an application and achieved a zero percent FPR and FNR. [30] used CCPM and deep learning to get effective FPR and FNR.

Insights into the evaluation metrics and completeness. A recurrent problem with almost every paper we studied is a lack of detailed evaluations. Five out of the ten studies we reviewed had no data about either FPR, FNR or TPR. Again, only five studies reported a precision for their proposed solution. Furthermore, several of the evaluation numbers we use here had to be calculated from data in the papers, they were not actually given.

4.4 Results of RQ3

We will now make a summary of the datasets, web applications and test suites used for the empirical evaluation in the primary studies. We will also briefly look at which vulnerabilities the given datasets have. Five of the ten primary studies had detailed information about what they used for testing. Two of these used web applications; [1] tested on live production applications and [4] tested on a vulnerable-by-design application from github. Two others used vulnerability databases, for example [8] used the XXSed database, created in 2007 and according to their webpage the biggest archive for XSS vulnerabilities [34]. Le et al. [27] used *"...a series of web application challenges which belong to the BKAV WhiteHat Contests"*, which they claimed cover all classical cases of vulnerability types.

Five primary studies did not have detailed information about their testing datasets. Several of these vaguely referred to some type of application. For example Stallenberg et al. [26] tested on what they called SBANK. This was also used by Jan et al. [4] where they described it as an *"XML-based web application interacting with a real-world bank card processing system of a credit card processing company"*. However, in contrast to Stallenberg et al., Jan et al. also had other test datasets.

5 DISCUSSION

For a discussion regarding related studies see Bing Zhang et al. [23], which this is a minified version of.

5.1 Implications of the results for practitioners and researchers

- **For practitioners.** From this study it is clear that there are several new, emerging WAVD tools, especially in regards to input validation. Some of these report improvements on the more well-known tools like Sqlmap and Nikto [2, 7]. Practitioners could use our results from RQ1 and RQ2 to be updated on the latest tools, maybe some are ready to be used in real-life cases.
- **For researchers.** One clear take-away from this paper is the predominance of input validation WAVD papers. Even though, as mentioned earlier, this is the most frequent vulnerability researchers should still diversify their research. This paper can therefore help researchers better identify the gaps in literature. Our results from RQ3 also show a general lack of documentation with regards to test- applications and datasets. Future studies should be aware of this issue.

5.2 Threats to validity. Since the time period for the primary papers used in this article is so short there was a big issue finding enough relevant studies. The primary papers needed to be relevant to WAVD, be from the correct time period and have enough detailed information for us to evaluate. One can even argue that some of the primary papers presented here did not meet that last criteria, but we could not be too picky. Because of this there might just not be enough data to make any scientific conclusions. Furthermore there is also the threat that relevant articles were missed as searching for relevant papers is a both tedious and hard task to do.

6 CONCLUSION AND RECOMMENDATION FOR FURTHER WORK

In today's society the importance of IT security is becoming ever more apparent. Web applications have an obligation to protect their userdata as well as to protect the application itself. Due to this the needs for efficient ways to find web application vulnerabilities are also increasing. This is why we need WAVD tools and in the recent years several new approaches have appeared. In the same way as the original paper by Bing Zhang et al. [23], we have systematically reviewed some of these approaches using the same classification and methods.

Using the primary articles we classified six categories describing what web attributes the different WAVD approaches analyse. From there we also categorized five approaches to WAVD analysis from the primary papers. The results show several different methods, based on a wide range of web attributes.

As with the studie from Bing Zhang et al. [23] we see that there is a lack of proper benchmarking web applications and test datasets to evaluate different approaches. More research should go into developing clear guidelines for WAVD evaluation and a set of web applications/test suites that are suitable for testing different approaches against each other with empirical results.

A SUPPLEMENTARY MATERIALS

See the contents of Appendix A.

REFERENCES

[1] Stefano Calzavara, Mauro Conti, Riccardo Focardi, Alvise Rabitti, and Gabriele Tolomei. 2020. Machine Learning for Web Vulnerability Detection: The Case of Cross-Site Request Forgery. *IEEE Security & Privacy* (2020), 0–0. DOI:http://dx.doi.org/10.1109/msec.2019.2961649

[2] Iberia Medeiros, Miguel Beatriz, Nuno Neves, and Miguel Correia. 2019. SEPTIC: Detecting Injection Attacks and Vulnerabilities Inside the DBMS. *IEEE Transactions on Reliability* 68, 3 (2019), 1168–1188. DOI:http://dx.doi.org/10.1109/tr.2019.2900007

[3] Long Zhang, Donghong Zhang, Chenghong Wang, Jing Zhao, and Zhenyu Zhang. 2019. ART4SQLi: The ART of SQL Injection Vulnerability Discovery. *IEEE Transactions on Reliability* 68, 4 (2019), 1470–1489. DOI:http://dx.doi.org/10.1109/tr.2019.2910285

[4] Sadeeq Jan, Annibale Panichella, Andrea Arcuri, and Lionel Briand. 2019. Search-based multi-vulnerability testing of XML injections in web applications. *Empirical Software Engineering* 24, 6 (2019), 3696–3729. DOI:http://dx.doi.org/10.1007/s10664-019-09707-8

[5] SpiderLabs. 2016. SpiderLabs/MCIR. (May 2016). Retrieved April 27, 2020 from https://github.com/SpiderLabs/MCIr

[6] X.SSed Staff. Cross Site Scripting (XSS) attacks information and archive. Retrieved April 27, 2020 from http://www.xssed.com/about

[7] Muhammad Saidu Aliero, Imran Ghani, Kashif Naseer Qureshi, and Mohd Fo'Ad Rohani. 2019. An algorithm for detecting SQL injection vulnerability using black-box testing. *Journal of Ambient Intelligence and Humanized Computing* 11, 1 (July 2019), 249–266. DOI:http://dx.doi.org/10.1007/s12652-019-01235-z

[8] Yong Fang, Cheng Huang, Yijia Xu, and Yang Li. 2019. RLXSS: Optimizing XSS Detection Model to Defend Against Adversarial Attacks Based on Reinforcement Learning. *Future Internet* 11, 8 (2019), 177. DOI:http://dx.doi.org/10.3390/fi11080177

[9] Xss and X.SSed Staff. The XSSed Project Database. Retrieved April 27, 2020 from http://www.xssed.com/
[10] OWASP Top Ten. Retrieved April 27, 2020 from https://owasp.org/www-project-top-ten/
[11] Xiaowei Li and Yuan Xue. 2014. A survey on server-side approaches to securing web applications. ACM Comput. Surv. 46, 4, Article 54 (March 2014), 29 pages. DOI: http://dx.doi.org/10.1145/2541315
[12] G. Deepa and P. S. Thilagam. 2016. Securing web applications from injection and logic vulnerabilities: Approaches and challenges. Information and Software Technology. 74 (June 2016), 160-180.
[13] Anon. Input Validation Vulnerabilities in Web Applications. Retrieved April 27, 2020 from https://scialert.net/fulltextmobile/?doi=jse.2014.116.126
[14] C. Visaggio. 2010. Session management vulnerabilities in today's web. In Proceedings of IEEE Security Privacy. 8, 5 (October 2010), 48-56.
[15] M. Balduzzi, C. Gimenez, D. Balzarotti and E. Kirda. 2011. Automated Discovery of Parameter Pollution Vulnerabilities in Web Applications. NDSS Symposium.
[16] P. Jyotiyana, S. Maheshwari. 2018. Techniques to Detect Clickjacking Vulnerability in Web Pages. In Optical and Wireless Technologies. Lecture Notes in Electrical Engineering, Springer, Singapore, 615-624.
[17] D. Kavitha, S. Chandrasekaran, and S. K. Rani. 2016. HDTCV: Hybrid Detection Technique for Clickjacking Vulnerability. Advances in Intelligent Systems and Computing, 607-620.
[18] X. Li, and Y. Xue. 2013. LogicScope: automatic discovery of logic vulnerabilities within web applications. In Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security, Hangzhou, China, 481-486.
[19] S. Wen, Y. Xue and J. Xu, et al. 2016. Toward Exploiting Access Control Vulnerabilities within MongoDB Backend Web Applications. In Proceedings of 40th Annual Computer Software and Applications Conference. Vol. 1. IEEE, 143-153.
[20] X. Li, X. Si, and Y. Xue. 2014. Automated black-box detection of access control vulnerabilities in web applications. In Proceedings of the 4th ACM conference on Data and application security and privacy, San Antonio, Texas, USA, 49-60.
[21] X. Li, and Y. Xue. 2011. BLOCK: a black-box approach for detection of state violation attacks towards web applications. In Proceedings of the 27th Annual Computer Security Applications Conference, Orlando, Florida, USA, 247-256.
[22] Julian Thomé, Lwin Khin Shar, Domenico Bianculli, and Lionel Briand. 2018. Security slicing for auditing common injection vulnerabilities. *Journal of Systems and Software* 137 (2018), 766–783. DOI:http://dx.doi.org/10.1016/j.jss.2017.02.040
[23] Bing Zhang, Jingyue Li, Jiadong Ren, and Guoyan Huang. 2018. Efficiency and effectiveness of web application vulnerability detection approaches: A Systematic Literature Review. J. ACM 37, 4, Article 1 (August 2018), 35 pages. https://doi.org/10.1145/1122445.1122456
[24] Oria Søketjeneste. Retrieved April 27, 2020 from https://bibsys-almaprimo.hosted.exlibrisgroup.com/
[25] Jing Zhao, Tianran Dong, Yang Cheng, and Yanbin Wang. 2020. CMM: A Combination-Based Mutation Method for SQL Injection. *Structured Object-Oriented Formal Language and Method Lecture Notes in Computer Science* (2020), 345–361. DOI:http://dx.doi.org/10.1007/978-3-030-41418-4_23
[26] Dimitri Michel Stallenberg and Annibale Panichella. 2019. JCOMIX: a search-based tool to detect XML injection vulnerabilities in web applications. *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2019* (2019). DOI:http://dx.doi.org/10.1145/3338906.3341178

[27] Van-Giap Le, Huu-Tung Nguyen, Duy-Phuc Pham, Van-On Phung, and Ngoc-Hoa Nguyen. 2018. GuruWS: A Hybrid Platform for Detecting Malicious Web Shells and Web Application Vulnerabilities. *Transactions on Computational Collective Intelligence XXXII Lecture Notes in Computer Science* (2018), 184–208. DOI:http://dx.doi.org/10.1007/978-3-662-58611-2_5

[28] CTFtime.org / WhiteHat Contest. Retrieved April 27, 2020 from https://ctftime.org/ctf/112

[29] Gutenberg Team, WordPress Contributors, and BuddyPress Community. WordPress.org. Retrieved April 27, 2020 from https://wordpress.org/plugins/browse/featured/

[30] Ning Guo, Xiaoyong Li, Hui Yin, and Yali Gao. 2020. VulHunter: An Automated Vulnerability Detection System Based on Deep Learning and Bytecode. *Information and Communications Security Lecture Notes in Computer Science* (2020), 199–218. DOI:http://dx.doi.org/10.1007/978-3-030-41579-2_12

[31] 2020. National Institute of Standards and Technology. (April 2020). Retrieved April 27, 2020 from https://www.nist.gov/

[32] Welcome to the NIST Software Assurance Reference Dataset Project. Retrieved April 27, 2020 from https://samate.nist.gov/SRD/index.php

[33] Nikic. 2020. nikic/PHP-Parser. (April 2020). Retrieved April 27, 2020 from https://github.com/nikic/PHP-Parser

SUPPLEMENTARY MATERIALS

Appendix A

Table A.1

THE EFFICIENCY AND EFFECTIVENESS OF WAVD METHODS ON INPUT VALIDATION VULNERABILITIES

ID	Type	AO	VL	Lang	Efficiency (TCM/MC/Auto)	FPR	FNR	TPR	Precision	Evaluation	Ref.
1	GAM	B4	SQLi	PHP	18.98s-208.17s					3 OSS VL simulation benchmark. F-measure 257.33.	[3]
2	VCPM	B4	SQLi and stored injection attacks	PHP, Java, Visual Basic	2.2% Latency Overhead	0.00 %	0.00%			Insert VLs in an application and compare with 4 anti-SQL tools and 11 real world applications	[2]
3	GAM	B4	XSS	JavaScript, Python					74%	20 websites from the Alexa Top 10k ranking	[1]
4	GAM	S1, S5	XMLi	Java, PHP	180s-1380s					A vulnerable-by-design, open-source application named XMLMao[5] Code coverage: 100%	[4]
5	GA	S3	SQLi	Java				71.00 %	100.00%	F-measure: 0.82.	[7]
6	AM+CCPM	B2	XSS	Python				97.90 %	99.50%	33,426 samples from the XSSed database[9].	[8]
7	GAM	B4	SQLi	PHP	6.52% effective test cases					3 OSS VL simulation benchmark. F-measure 8*	[25]
8	GA	S4	XMLi	Java	300s					45 XMLi VL detected in SBANK [4].	[26]
9	MFM	S1, S5	SQLi, XSS, remote code execution	PHP			11.54 %	88.46 %	90.20%	20 test cases from BKAV WhiteHat Contests [28] and 4 “safe” featured plugins of WordPress [29].	[27]
10	CCPM	S1	SQLi, XSS	PHP		6.26 %	2.01%	97.99 %	85.76%	18,989 programs from NVD [31] and SARD [32].	[30]

Table A.2

WEB APPLICATIONS/DATASETS USED IN PRIMARY STUDIES

Frequency of use	App. or test dataset	URL	Vulnerabilities	Ref.
1	Alexa top 10,000 websites	https://www.alexa.com/topsites	Not known	[1]
1	XMLMao	https://github.com/SpiderLabs/XMLmao	XMLi	[4]
1	The XSSed project DB	http://www.xssed.com/archive	XSS	[8]
1	WhiteHat Contest	https://ctftime.org/ctf/112	Command Injection, Object Injection, File Inclusion, Arbitrary Eval Code Injection, XSS, SQLi	[27]
1	National Vulnerability Database	https://www.nist.gov	SQLi, XSS	[30]