Computing (2019) 101:161–185
https://doi.org/10.1007/s00607-018-0664-z

An empirical study on combining diverse static analysis tools for web security vulnerabilities based on development scenarios

Paulo Nunes[1,4] · Ibéria Medeiros[2] · José Fonseca[1,4] · Nuno Neves[2] · Miguel Correia[3] · Marco Vieira[4]

Received: 30 September 2017 / Accepted: 15 September 2018 / Published online: 25 September 2018

Abstract
Automated Static Analysis Tool (ASATs) are one of the best ways to search for vulnerabilities in applications, so they are a resource widely used by developers to improve their applications. However, it is well-known that the performance of such tools is limited, and their detection capabilities may not meet the requirements of the project regarding the criticality of the application. Diversity is an obvious direction to take to improve the true positives, as different tools usually report distinct vulnerabilities, however with the cost of also increasing the false positives, which may be unacceptable in some scenarios. In this paper, we study the problem of combining diverse ASATs to improve the overall detection of vulnerabilities in web applications, considering four development scenarios with different criticality goals and constraints. These scenarios range from low budget to high-end (e.g., business critical) web applications. We tested with five ASATs under two datasets, one with real WordPress plugins and another with synthetic test cases. Our findings revealed that combining the outputs of several ASATs does not always improve the vulnerability detection performance over a single ASAT. By using our procedure a developer is able to choose which is the best combination of ASATs that fits better in the project requirements.

Keywords Static analysis · Vulnerability detection · XSS · SQLi

Mathematics Subject Classification 68M15 Reliability · 68M11 Internet topics

This work extends a preliminary version presented at the 13th European Dependable Computing Conference (EDCC 2017).

✉ Paulo Nunes
pnunes@ipg.pt

Extended author information available on the last page of the article

1 Introduction

Web applications have experienced a rapid growth over the last years and become one of the most prevalent technologies in the Web. They when installed in a single host server become immediately accessible to users anytime, anywhere via an Internet connection through a range of different devices.

A security vulnerability is a flaw or a weakness in system security procedures, design, implementation, or internal controls that can be exercised (accidentally triggered or intentionally exploited) and result in a security breach or a violation of the system's security policy. However, applications, even developed by skilled programmers, have security vulnerabilities [1]. In fact, the number of attacks reported by different entities exploiting vulnerabilities has been growing in the past years [19]. The attacks can have a devastating impact including financial cost and loss of market share [20].

ASATs are a key piece in the security chain as they inspect the source code of software, without executing it, to discover potential bugs early in the Software Development Life Cycle (SDLC) that lead to security vulnerabilities [24]. However, ASATs have limitations, such as missing some of the vulnerabilities [False Negative (FN)] and generating many false alarms [False Positive (FP)] [23]. These limitations arise from conceptual constraints of the Static Analysis (SA) process that mainly checks the structure of a program based on fixed rules and does not consider the user input data or the dynamic characteristics of the application [35]. Therefore, it is known that different ASATs report distinct sets of security vulnerabilities, with some overlap [14,26,30].

The state-of-the art of ASATs is, on average, able to detect about half of the existing security vulnerabilities [17]. To improve their overall detection capabilities, some researchers have proposed combining the results of diverse ASATs [27,33,36]. Of particular interest is the work of Diaz et al. [15], which compares the performance of nine ASATs, most of them commercial tools, against the Software Assurance Reference Dataset (SARD) from the Software Assurance Metrics and Tool Evaluation (SAMATE) [28] at NIST. Based on the results, the authors recommended the use of several tools with different detection algorithms/heuristics to improve the analysis results. On the other hand, Beller et al. [13] investigated how common is the use of ASATs in real-world, taking as reference the 122 most popular Open-Source Software projects. The results showed that a single ASAT was used in 41% of the projects, two ASATs in 22%, and three ASATs in 14% of the projects. This suggests that developers might not be aware of the benefits of using multiple tools and/or that the increase of FPs reported may lead developers to avoid using multiple ASATs [21]. However, existing works are limited in several aspects: the used datasets are synthetic or composed of small sets of applications (each work presents its own selection of test applications), the evaluation metrics used are too simple [e.g., number of True Positives (TPs)], and the analysis does not consider the specificities of the development scenarios were the tools can be used, which may vary both in terms of development time and resources.

In this paper, we argue that the use of multiple ASATs might be helpful, as more vulnerabilities may be reported, however, the drawback is that the number of FPs may at the same time increase. Furthermore, we also claim that the acceptable/expected outcome of the SA process (in terms of TPs and FPs) depends on the development scenario. A scenario is a realistic situation of vulnerability detection that depends on the criticality of the application being developed and on the security budget available. Thus, it is no longer evident that combining more ASATs is better in every case. Moreover, datasets composed of real world applications fully characterized in terms of vulnerabilities and non-vulnerabilities are missing.

The goal of our work is studying the potential of combining the outputs of multiple ASATs with a 1-out-of-n strategy as a way to improve the performance of vulnerability detection across different realistic development scenarios. In practice, we formulate the following seven hypotheses: $\mathbf{H}_1$: The number of vulnerabilities detected always increases as the number of combined ASATs increases. $\mathbf{H}_2$: The number of FPs always increases as the number of combined ASATs increases. $\mathbf{H}_3$: The best combination of ASATs is the same across development scenarios. $\mathbf{H}_4$: The best combination of ASATs is the same across different classes of vulnerabilities. $\mathbf{H}_5$: The best combination of ASATs is the same regardless the classes of vulnerabilities. $\mathbf{H}_6$: The methodology adopted can be used for other kind of applications. $\mathbf{H}_7$: The results for H_1 to H_5 are the same for any kind of application.

Although the response to the hypotheses H_1 to H_5 may seem obvious, empirical evidences are missing in the literature to better understand the advantages and limitations of different combinations of ASATs, when considering representative vulnerability detection scenarios. For example, a less informed researcher or developer could easily state that *the number of vulnerabilities detected increases as the number of combined ASATs increases*, however, knowing in which scenarios that is true, to which amount that happens for different types of vulnerabilities, and what is the impact in terms of FPs, are aspects that require more detailed studies, as the one presented in [31].

The contributions of this work are: (1) A general approach for creating datasets fully characterized in terms of Vulnerable LOCs (VLOCs) and Non-vulnerable LOCs (NVLOCs), which can be applied to any kind of applications (real or synthetic); (2) An approach for combining and ranking the results of diverse ASATs; (3) An experimental campaign with five free ASATs to detect vulnerabilities in about 20,000 synthetic test cases from the National Institute of Standards and Technology (NIST); (4) A comparative experimental study combining the outputs of multiple ASATs using datasets with real applications versus synthetic test cases.

The rest of the paper is organized as follows. The next section details our approach for generating datasets, combining and ranking the outputs of several ASATs. Sections 3 and 4 present two instances of our approach in different target application domains. Section 5 presents and discusses the results. Section 6 discusses related work and Sect. 7 concludes the paper.

2 Methodology

In this section we propose a general approach for evaluating the performance of combining the results of several ASATs. Therefore, we need a dataset (i.e., applications) with the results of ASATs searching vulnerabilities in applications. For evaluating and comparing the effectiveness of the combinations of the ASATs, first we need to

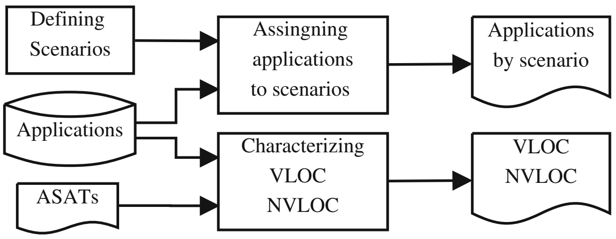

Fig. 1 Overall process for creating datasets characterized in term of VLOCs and NVLOCs

define a set of metrics and a method for combining the results, next computing the metrics and ranking the combination of ASATs. Moreover, to calculate the metrics, we have to know the VLOCs [i.e., Positive instances (P)] and the NVLOCs [i.e., Negative instances (N)] of the applications. However, such datasets are not publicly available. Therefore, our methodology includes a process for creating datasets (Sect. 2.1) and a method for combining and ranking the results of diverse ASATs (Sect. 2.3).

2.1 Creating datasets

Figure 1 illustrates our overall process for creating datasets organized in scenarios and fully characterized in term of VLOCs and NVLOCs. The process is composed by four phases: (1) *defining scenarios*, (2) *selecting applications*, (3) *assigning applications to scenarios*, and (4) *characterizing VLOCs and NVLOCs*.

2.1.1 Defining scenarios

Our scenarios are adapted from Antunes et al. [10]. They proposed four scenarios based on the business criticality of having applications with vulnerabilities and the impact severity that such vulnerabilities cause when exploited. To minimize such severity based on the criticality, they defined the four business scenarios with increasing requirements, and efforts needed to satisfy them (see Table 1). For example, in a business-critical scenario the objective is to detect as many vulnerabilities as possible, even if that requires spending budget analyzing FPs. Four criticality levels representing realistic scenarios were considered (we changed the names of the scenarios defined by them to better represent their requirements, but maintained their scope): (1) *Highest-quality*—every vulnerability missed may be a big problem due to the high criticality of the application; (2) *High-quality*—a few non-trivial vulnerabilities may be missed given that there are not many FPs; (3) *Medium-quality*—vulnerabilities may be missed at the cost of reducing the FPs; (4) *Low-quality*—every FP is an important cause of concern due to budget restrictions.

2.1.2 Selecting applications

The demand for new web applications will increase exponentially over the next decade [16]. In fact, most companies and millions of people relies on the Internet and web

Table 1 Goals of vulnerability detection by scenario [31]

Scenario	TP	FP	Resources to fix (manual effort)
1—Highest-quality	Highest	Many	All that are needed
2—High-quality	Highest	Not many	Balanced appropriately
3—Medium-quality	High	Low	Limit, redirected to fix
4—Low-quality	High	Lowest	Very low

applications. The complexity and interactivity of applications has been increased tremendous. Therefore, are emerging with great speed many tools, frameworks (e.g., Laravel) and Content Management Systems (CMS) [e.g., WordPress (WP)] for developing applications in a short frame time. The applications that compose the dataset have to be representative of all applications. Unfortunately, this is very hard to attain. To reduce this problem, the dataset is specifically built for a particular domain (e.g., WP plugins). However, the selection of a set of representative applications in a particular domain is yet a difficult task. Thus, our approach to select applications is based on vulnerability's repositories (e.g., National Vulnerability Database) that publicly confirmed vulnerabilities in real software. Thus, for the chosen domain, we collect applications with at least one vulnerability in the target classes of vulnerabilities [e.g., SQL Injection (SQLi), Cross-Site Scripting (XSS)]. The LOC where the vulnerabilities are located become our initial set of VLOCs.

2.1.3 Assigning applications to scenarios

To organize the dataset, we need to assign a representative group of applications to each scenario. Since this is very hard to achieve (e.g., real business-critical software is often kept secret) and has an associated level of subjectivity (e.g., there are different interpretations of what constitutes critical software), we propose a general procedure to classify applications based on code quality. Generically, the assumption is that scenarios that are more stringent normally the software developed is with better quality. Therefore, we should assign applications with better quality to scenarios with higher criticality. In practice, this means that a given set of applications can be used in a scenario if their source code has a sufficient level of quality and admissible artefacts.

Our process for assigning applications to scenarios has two steps [31]. The first is based on the approach proposed by Baggen et al. [12] for rating the maintainability of the source code of applications (from 0.5 to 5.5 stars). The Baggens approach uses a standardized measurement model based on the ISO/IEC 9126 definition of maintainability and a small set of Source Code Metrics (SCMs) (e.g., Cyclomatic Complexity Number (CCN) [32]). These SCMs are used to measure the SPPs (e.g., Unit Complexity) of the software. Table 2 shows the SPPs and their relationship with the sub-characteristics of maintainability and includes an example. The second step of the process is based on a simple schema for mapping the ratings in scenarios. We used the following mapping the rating to scenarios: [4.5, 5.5[—Scenario 1; [3.5, 4.5[—Scenario 2; [2.5, 3.5[—Scenario 3; and [0.5, 2.5[—Scenario 4. As shown, we used

Table 2 Mapping of SPPs to ISO/IEC sub-characteristics of maintainability and an example

	Software product properties								
Sub-characteristics	D	UC	US	MC	CC	UI	CIS	UT	Average
Ratings example	5.0	4.0	3.0	4.0	3.0	3.0	2.0	3.0	
Analyzability	×	×	×						4.0
Changeability	×	×		×	×				4.0
Stability						×	×	×	2.6
Testability		×	×	×				×	3.5
Maintainability rating (average: ⋆⋆⋆⋆)									3.5

D, duplication; UC, unit complexity; US, unit size; MC, module coupling; CC, class complexity; UI, unit interface; CIS, class interface size; UT, unit testing

intervals of 1 for mapping the ratings into the scenarios, trying to respect Baggens's approach, except for the less stringent scenario (4), which accommodates all the ratings below 2.5.

2.1.4 Characterizing VLOCs and NVLOCs

The applications selected for the dataset have at least one VLOC (Sect. 2.1.2). However, the real number of VLOCs is unknown. For large code bases, identifying all VLOCs is a hard task that requires a thorough review by security experts. The upper bound for the number of VLOCs for a given class of vulnerability in an application is limited to the number of Lines of Codes (LOCs) with a related security-critical function calls [i.e., Sensitive Sinks (SSs), e.g., `echo` for XSS and `mysql_query` for SQLi] with at least one variable [11]. In fact, these LOCs are potentially vulnerable because they are the gateways by which threats are manifested. In addition, a SS is considered a function from the language that can output an unexpected result if some of its parameters is malicious data, which was provided by an attacker through application Entry Points (EPs) (e.g., `$_GET` in PHP that receives user inputs). Therefore, a VLOC is a LOC with at least one SS for which there exists at least one data flow from untrusted data sources (EPs). On the other hand, we consider a NVLOC as being data flows that are not vulnerable, i.e., LOCs with all SSs function calls without any variable [11] or with variables or EPs previously sanitized by some Sanitization Function (SF) (e.g., `htmlentities` function to invalidate XSS vulnerabilities) that invalidates malicious data. The no 1 illustrates examples of VLOCs, one NVLOC, EPs, SFs and SSs.

Our approach to find more VLOCs in the dataset is based on running ASATs searching vulnerabilities in the dataset and on a manual review to confirm if each SS is a TP or a FP. The union of all TPs with the initial list of vulnerabilities (Sect. 2.1.2) become the list of VLOCs. The list of NVLOCs is obtained from all LOCs potentially vulnerable, i.e., with a SS but are FP.

```
$name = $_GET['name'];                                           // EP
    , User
print("<h1>Your search for: $name</h1>");                       // SS
    ,VLOC,XSS
$sql="SELECT * FROM Contacts where name like '%$name%'";
${\it result}={\it mysql_query}($sql);
          // SS,VLOC,SQLi
echo '<table><tr><th>Name</th><th>Phone</th></tr>';             // No
     SS
while($data = mysql_fetch_array($result)) {                     // EP
    , Database
  echo '<tr><td>'.$data[name].'</td>';                          // SS
    ,VLOC
  echo '<td>'.htmlspecialchars($data['phone'],ENT_QUOTES);      // SS
    ,NVLOC,XSS
  echo '</td><tr>';                                             // No
     SS
```

Listing 1 PHP vulnerable code (SQLi and XSS) for searching contacts in a database

Table 3 Summary of metrics by scenario [31]

Scenario	Antunes et al. scenario [10]	Main metric	Tiebreaker metric
1—Highest-quality	Business-critical	Recall	Precision
2—High-quality	Heightened-critical	Informedness	Recall
3—Medium-quality	Best-effort	F-measure	Recall
4—Low-quality	Min-effort	Markedness	Precision

2.2 Metrics

For evaluating the combinations of ASATs we propose the metrics defined by Antunes et al. [10]. Therefore, for each scenario there is one main metric to rank the combinations of ASATs and a tiebreaker metric used only when there is a tie among combinations of ASATs (see Table 3).

In practice, the metrics depend on the goals of the detection, which are related with the amount of available resources to fix the vulnerabilities (see Table 1). For example, for the highest-quality scenario the goal is to find the highest number of vulnerabilities at any cost. The metric recall is used to measure this global information. However, it ignores the precision (FPs) of the results. Thus, in the case of a tie among combinations of ASATs, the precision metric is used to rank first the combination reporting less FPs. The metrics *recall* $= TP/(TP + FN)$, *precision* $= TP/(TP + FP)$ and *F-Measure* $= 2 \times TP/(2 \times TP + FP + FN)$ are well known and widely used. Next, we define the remaining ones:

– *Informedness = Recall+Inverse Recall-1*. Requires knowing the overall number of P ($P = TP + FN$) and N ($N = FP + TN$) instances. Therefore, every TP increases the metric in the proportion 1/P and every FP decreases the metric in the proportion 1/N. Since the prevalence of P instances (for the plugins) is less than the prevalence of N instances, the metric prioritizes ASATs reporting more vulnerabilities and at the same time not too many FPs, which is the goal of the

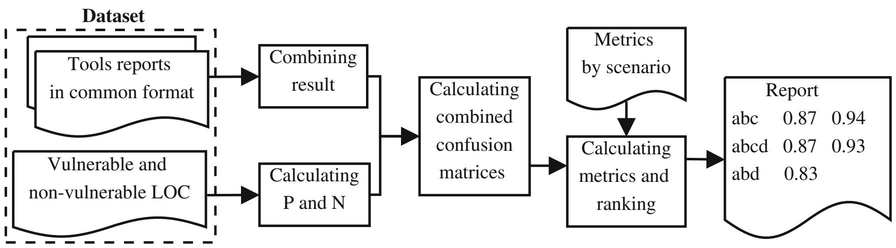
Dataset
Tools reports in common format
Vulnerable and non-vulnerable LOC
Combining result
Calculating P and N
Calculating combined confusion matrices
Metrics by scenario
Calculating metrics and ranking
Report
abc 0.87 0.94
abcd 0.87 0.93
abd 0.83

Fig. 2 Overall process of combining results of multiple tools

high-quality scenario.

$$informedness = TP/(TP + FN) + TN/(FP + TN) - 1 \tag{1}$$

– *Markedness = Precision+Inverse Precision-1*. Considers only the number of TPs and True Negatives (TNs) reported. The metric sums the proportions of the P and the N instances that are correctly identified as such. For the Precision (1st part of the formula 2) a TP increases the metric while a FP decreases the metric in the same proportion. For the Inverse Precision (2nd part of formula 2) a FP also decreases the metric but at the inverse proportion of the N instances reported (i.e., low value). Therefore, the metric portrays the required goal for the low-quality scenario because severely penalizes ASATs reporting FPs.

$$markedness = TP/(TP + FP) + TN/(FN + TN) - 1 \tag{2}$$

2.3 Combining the results of multiple ASAT

Our method for combining the results of the ASATs is based on all possible combinations of the ASATs. Multiple strategies (e.g., majority voting, N-out-of-N, 1-out-of-N) could be considered for combining the results of multiple ASATs. However, since the dataset is characterized in terms of VLOCs and NVLOCs, we use the 1-out-of-N strategy for combining the results of the ASATs. The process proposed to calculate the combined results for two or more tools is based on a set of automated steps (see Fig. 2):

1. *Calculate the number of P and N in the dataset*—using the lists of VLOCs, NVLOCs, and the list of the applications per scenario, calculate the values of P and N for each scenario and for each class of vulnerability.
2. *Combine results of ASATs*—for each scenario, class of vulnerability and combination of the ASATs, merge the TPs and the FPs of the tools.
3. *Calculate the combined confusion matrices*—with the outputs of steps 1 and 2, calculate, for each scenario, class of vulnerability and combination of ASATs, the corresponding confusion matrix (i.e., TP, FP, FN, and TN).
4. *Calculate the metrics and rank*—with the results from step 3, compute the metrics for each scenario (see Table 3) and rank the combinations of tools.

Table 4 Plugin background information [31]

Scenarios	OOP	POP	Total	%Tot.	Files	LLOC	%LLOC
1—Highest-quality	10	2	12	9.0	352	19,542	4.2
2—High-quality	39	17	56	41.8	1687	122,835	26.4
3—Medium-quality	40	11	51	38.1	2223	211,297	45.3
4—Low-quality	14	1	15	11.2	728	112,490	24.1
Total	103	31	134	100.0	4990	466,164	100.0

3 WordPress plugins dataset

In this section, we apply our process for creating datasets to the domain of WP plugins. WP is by far the most prominent CMS [9,19]. There are over 52,042 WP plugins that have been downloaded more than 1.3M times [6]. However, probably most of these plugins have vulnerabilities and, since a single plugin may be used in thousands of websites, they are an attractive target for hackers. For instance, Sucuri reported that, in Q1 2016, 89,000 WP sites were compromised by exploiting vulnerable plugins [5].

The target classes of vulnerabilities we analyzed are SQLi and XSS, and the target ASATs are free tools. In fact, according to WhiteSecurity statistics, from 42,000+ WP Websites in Alexa Top 1 Million, more than 73% of the installations have vulnerabilities, which could be potentially detected using free automated tools [8]. SQLi and XSS are vulnerabilities that are on the Open Web Application Security Project (OWASP) Top 10 [2], and are also quite common in WP plugins [4]. Both vulnerabilities occur when input data are not properly validated. A SQLi attack is based on the injection of code that changes the SQL query sent to the database and an XSS attack consists in the injection of malicious JavaScript in the input of a vulnerable web page.

3.1 Selecting the plugins

The online WordPress Vulnerability Database (WPVD) [4] provides a list of WP plugins with known vulnerabilities including proofs of concept (PoC) and other details (e.g., CVE identifier). From this list, we selected all the plugins with SQLi and/or XSS vulnerabilities, resulting in a list of 134 plugins with 152 SQLi (84 with PoC) and 67 XSS (13 with PoC) vulnerabilities registered. In practice, the dataset consists of 103 Object-Oriented Programming (OOP) plugins and 31 Procedure Oriented Programming (POP) plugins having a total of 4990 files and 1,023,081 LOC (57% is OOP, 32% is POP, and 11% are mix of both). The dataset contains 466,164 Logical Lines of Code (LLOC), where 39.5% are POP, 47.8% are OOP, and 12.7% is a mix of both (see Table 4).

3.2 Assigning plugins to scenarios

The results of applying our process for assigning applications to scenarios (see Sect. 2.1.3) are presented in Table 4, which shows the number of plugins that compose the dataset, distributed over the four scenarios. Scenarios 1 and 2 have lower number of plugins compared with the scenarios 2 and 3. This is realistic as finding code with very high quality is not trivial and due to the popularity of the plugins considered it is expected to not find plugins with low-quality. In terms of LLOCs, we observed a “normal” distribution around the medium-quality scenario with a small tail for the highest-quality scenario.

3.3 Characterizing VLOC and NVLOC for the WordPress plugins dataset

For detecting the SQLi and XSS vulnerabilities in the WordPress Plugins Dataset (WPD) we used the following ASATs: RIPS v0.55 [14], Pixy v3.03 [22], phpSAFE [30], WAP v2.0.1 [26], and WeVerca v20150804 [18]. RIPS performs static taint analysis and string analysis. RIPS and Pixy are the two most referenced PHP ASATs in the literature, but they are not ready for OOP analysis. RIPS has only been developed as open source until 2014. RIPS has recently released a commercial version able to fully analyze OOP code. WAP, phpSAFE, and WeVerca are recent tools under active development, and they are prepared for OOP code.

To obtain the list of VLOCs, we ran the five ASATs to search SQLi and XSS vulnerabilities in the dataset. All tools, but WeVerca, were configured by default for PHP EPs, SSs and SFs (e.g., `htmlentities` for preventing XSS and `mysql_real_escape_string` for preventing SQLi). WeVerca does not allow configuration and includes a list of EPs, SSs and SFs. Overall, phpSAFE was unable to analyze 18 plugins. WAP analyzed all plugins, but seven of them only partially. RIPS and Pixy analyzed partially 76 and 103 plugins, respectively. WeVerca was not able to analyze a total of 20 source files of 14 plugins. In practice, the tools could not to fully analyze some plugin/files, reporting runtime errors or taking a very long time without any results.

For gathering the list of NVLOCs, we developed a PHP script to select from the source files all LOCs with a SS function call with at least one variable. From this list, we removed the items that were already labeled as VLOCs.

Table 5 reports the results of characterizing VLOCs and NVLOCs of the WPD. The table depicts the number of TPs and FPs reported by the ASATs, followed by the number of vulnerabilities registered in the WPVD (column VD). The column NVLOC-FP represents the number of NVLOC not reviewed manually. The next four columns show the total number/percentage of P (the distinct VLOCs), and the total number of N (i.e., NVLOCs).

For the low-quality scenario, the ASATs reported fewer VLOCs than the high-quality and medium-quality scenarios. However, software with low-quality does not necessarily means more security vulnerabilities. We also observed that both the density of (P + N) per KLLOC of code and the percentage of files successfully analyzed (FSA) by all tools are low for this scenario, with which may limit the number of VLOCs found by the tools. In fact, the tools have more difficulties analyzing software with low quality.

Table 5 Distribution of VLOCs and NVLOCs by scenarios for the WordPress Dataset [31]

	Scenario	Tools		VD	NVLOC	Total				(P+N)/KLLOC	FSA
		TP	FP	P	- FP	P	N	P%	N%		%
SQLi	1—Highest-quality	65	5	17	82	75	87	46	54	0.16	8.3
	2—High-quality	318	62	35	1053	346	1115	24	76	1.46	6.2
	3—Medium-quality	251	163	22	2051	267	2214	11	89	2.48	5.2
	4—Low-quality	41	32	10	1105	50	1137	04	96	1.19	2.1
	Total	675	262	84	4291	738	4553	14	86	5.29	–
XSS	1—Highest-quality	165	45	3	945	168	990	15	85	1.16	17.7
	2—High-quality	1841	224	1	5601	1842	5825	24	76	7.67	16.8
	3—Medium-quality	2386	680	4	9289	2389	9969	19	81	12.36	17.1
	4—Low-quality	543	117	5	3477	545	3594	13	87	4.14	10.3
	Total	4935	1066	13	19,312	4944	20,378	20	80	25.32	–

Tools: phpSAFE,RIPS,WAP,Pixy,WeVerca. VD, number of VLOCs registered in the WPVD; FSA, files successfully analyzed by all five tools

Table 6 Summary of the generated synthetic test cases

Vulnerability	Safe				Unsafe				Total
	POP	OOP	Total	%Total	POP	OOP	Total	%Total	
SQLi	6336	2304	8640	90	684	228	912	10	9552
XSS	4296	1432	5728	57	3264	1088	4352	43	10,080

4 Synthetic dataset

The main goal of this experiment is to evaluate the applicability of our approach for creating Synthetic Datasets (SDs) using synthetic test cases. It involves three steps.

(1) *Selecting the synthetic test cases* The SARD at NIST provides a repository of test cases (datasets) with a set of known security vulnerabilities [7]. The most significant synthetic dataset for PHP was provided by Stivalet and Delaitre [34]. They proposed a generic approach for generating safe (i.e., non-vulnerable) and unsafe (i.e., vulnerable) test cases and developed a tool for generating PHP test cases. Therefore, we used the Stivalet's tool for generating the test cases for SQLi and XSS vulnerabilities (see Table 6).

(2) *Assigning test cases to scenarios* Table 7 lists the results of assigning the test cases to scenarios. Most of them were assigned to the highest-quality scenario where almost of the test cases are OOP. The high-quality scenario accounts for 10%, the medium-quality scenario 19%, and the low-quality scenario none. This distribution occurred because the test cases are very small programs (from 3 LLOC to 29 LLOC). Therefore, the values for some SPPs (e.g., Module Coupling) of the test cases are zero, meaning that for these SPPs the test cases have the maximum quality (5.5 stars).

(3) *Characterizing VLOCs and NVLOCs of synthetic test cases* An advantage of SDs is that the test cases indicate where vulnerabilities occur. Since, each test case has just one target SS, the number of NVLOC is equal to the number of safe test cases (column N of Table 7) and the number of VLOCs is equal to the number of unsafe test cases (column P of Table 7).

5 Results and discussion

For studying the potential of combining the outputs of diverse ASATs as a way to improve the performance of vulnerability detection, we used the results of five ASATs searching for vulnerabilities in two datasets: WPD and a SD. For the WPD we used the same results of characterizing the VLOCs and the NVLOCs of the dataset. For the SD we ran the same ASATs to detect vulnerabilities in the synthetic test cases with the same configurations used for detecting the vulnerabilities in the WPD, to avoid bias.

Tables 8 and 9 list the results organized by scenario, class of vulnerability and together SQLi and XSS vulnerabilities for the WPD and the SD, respectively. The tables only show the TOP 5 (of 31) combinations of ASATs for each scenario, due to space limitations. The tables for all scenarios include the metrics Recall and Precision

Table 7 Synthetic test cases background information by scenario

Scenario	SQLi						XSS					
	POP		OOP		Total		POP		OOP		Total	
	N	P	N	P	N	P	N	P	N	P	N	P
1—Highest-quality	1908	213	228	2208	2136	2421	3880	2940	1432	1088	5312	4028
2—High-quality	1143	195	0	96	1143	291	324	236	0	0	324	236
3—Medium-quality	3285	276	0	0	3285	276	92	88	0	0	92	88

Table 8 Best solutions for the WordPress plugins: SQLi, XSS and SQLi + XSS

SQLi							XSS							SQLi + XSS		
Tools	TP	FP	Pg	MM	TM	P/R	Tools	TP	FP	Pg	MM	TM	P/R	Tools	MM	TM
Highest-quality				Recall	Prec.		Recall					Prec.			Recall	Prec.
ac	65	5	9	0.867	0.929	–	ab	165	43	11	0.982	0.793	–	abc	0.947	0.821
ace	65	5	9	0.867	0.929	–	abe	165	43	11	0.982	0.793	–	abcd	0.947	0.821
abce	65	5	9	0.867	0.929	–	abc	165	45	11	0.982	0.786	–	abcde	0.947	0.821
acde	65	5	9	0.867	0.929	–	abd	165	45	11	0.982	0.786	–	abce	0.947	0.821
abc	65	5	9	0.867	0.929	–	abce	165	45	11	0.982	0.786	–	acde	0.844	0.869
c	49	4	7	0.653	0.925	–	b	113	29	10	0.673	0.796	–	a	0.539	0.851
a	29	5	5	0.387	0.853	–	a	102	18	8	0.607	0.850	–	b	0.465	0.796
e	0	0	0	0	–	–	d	69	14	7	0.411	0.831	–	c	0.296	0.878
b	0	0	0	0	–	–	e	44	5	7	0.262	0.898	–	d	0.284	0.831
d	0	0	0	0	–	–	c	23	6	3	0.137	0.793	–	e	0.181	0.898
High-quality		Informedness			Rec.	Prec.	Informedness					Rec.	Prec.		Informedness	Rec.
acde	318	59	36	0.866	0.919	0.844	abce	1841	224	51	0.961	10.0	0.892	abce	0.946	0.987
abce	318	60	36	0.865	0.919	0.841	abcde	1841	224	51	0.961	10.0	0.892	abcde	0.946	0.987
abcde	318	60	36	0.865	0.919	0.841	abe	1838	223	51	0.960	0.998	0.892	abde	0.930	0.971
ace	316	59	36	0.860	0.913	0.843	abde	1838	223	51	0.960	0.998	0.892	abe	0.930	0.971
acd	311	58	35	0.847	0.899	0.843	abc	1770	224	51	0.923	0.961	0.888	abc	0.910	0.951

Table 8 continued

High-quality		Informedness			Rec.	Prec.	Informedness					Rec.	Prec.		Informedness	Rec.
a	274	58	30	0.740	0.792	0.825	a	1164	90	46	0.617	0.632	0.928	a	0.636	0.657
c	44	4	12	0.124	0.127	0.917	b	1013	194	46	0.517	0.550	0.839	b	0.454	0.483
b	43	2	8	0.123	0.124	0.956	e	436	50	25	0.228	0.237	0.897	e	0.200	0.207
e	18	1	6	0.051	0.052	0.947	d	453	148	28	0.221	0.246	0.754	d	0.193	0.214
d	16	0	7	0.046	0.046	1.0	c	219	55	18	0.110	0.119	0.799	c	0.112	0.120
Medium-quality			F-measure		Rec.	Prec.	F-measure					Rec.	Prec.		F-measure	Rec.
abce	251	163	21	0.737	0.940	0.606	abce	2386	652	46	0.879	0.999	0.785	abce	0.863	0.993
abcde	251	163	21	0.737	0.940	0.606	abcde	2386	652	46	0.879	0.999	0.785	abcde	0.863	0.993
abc	250	163	21	0.735	0.936	0.605	abc	2383	652	46	0.879	0.998	0.785	abc	0.863	0.991
abcd	250	163	21	0.735	0.936	0.605	abcd	2383	652	46	0.879	0.998	0.785	abcd	0.863	0.991
abde	237	163	19	0.711	0.888	0.593	abde	2359	652	46	0.874	0.987	0.783	abde	0.856	0.977
b	153	113	6	0.574	0.573	0.575	b	1812	490	43	0.773	0.759	0.787	b	0.752	0.740
a	99	50	15	0.476	0.371	0.664	a	970	267	41	0.535	0.406	0.784	a	0.529	0.402
c	72	0	11	0.425	0.27	10.0	d	717	56	23	0.454	0.300	0.928	d	0.441	0.290
d	54	13	4	0.323	0.202	0.806	e	621	21	19	0.410	0.260	0.967	e	0.383	0.242
e	21	34	3	0.130	0.079	0.382	c	344	13	18	0.251	0.144	0.964	c	0.270	0.157
Low-quality		Markedness			Prec.	Rec.	Markedness					Prec.	Rec.		Markedness	Prec.
bc	6	0	2	0.963	1.0	0.120	c	62	3	6	0.835	0.954	0.114	c	0.857	0.957
bce	6	0	2	0.963	1.0	0.120	abce	543	117	12	0.822	0.823	0.996	cde	0.835	0.925
bcde	6	0	2	0.963	1.0	0.120	abcde	543	117	12	0.822	0.823	0.996	ce	0.832	0.925

Table 8 continued

Low-quality		Markedness			Prec.	Rec.	Markedness					Prec.	Rec.		Markedness	Prec.
bcd	6	0	2	0.963	1.0	0.120	abc	542	117	12	0.822	0.823	0.994	cd	0.819	0.916
c	5	0	2	0.962	1.0	0.100	abcd	542	117	12	0.822	0.823	994	de	0.815	0.911
c	5	0	2	0.962	1.0	0.100	c	62	3	6	0.835	0.954	0.114	c	0.857	0.957
b	1	0	1	0.959	1.0	0.020	a	244	33	10	0.803	0.881	0.448	e	0.802	0.901
a	36	32	7	0.517	0.529	0.720	e	73	8	7	0.785	0.901	0.134	d	0.776	0.879
e	0	0	0	–	–	0.000	b	377	91	10	0.760	0.806	0.692	b	0.761	0.806
d	0	0	0	–	–	0.000	d	51	7	9	0.758	0.879	0.094	a	0.748	0.812
abce∗	675	260	–	–	0.722^1	0.915^2	abce∗	4935	1038	–	–	0.826^1	0.998^2	abcde∗	0.812^1	0.987^2

Tools: a—phpSAFE, b—RIPS, c—WAP, d - Pixy, e—WeVerca.
MM—Main Metric, TM—Tiebreaker Metric, Rec.—Recall (R) (^{2}), Prec.(P) (^{1})—Precision.
Pg—Number of plugins where the combination of ASATs reports vulnerabilities.
∗Solution with best recall regardless the scenarios

Table 9 Best solutions for the synthetic dataset: SQLi/XSS

SQLi						XSS						SQLi + XSS		
Tools Highest-quality	TP	FP	MM Recall	TM Prec.	P/R	Tools	TP	FP	MM Recall	TM Prec.	P/R	Tools	MM Recall	TM Prec.
bde	355	2072	0.805	0.146	–	abde	3290	4192	0.817	0.440	–	abde	0.816	0.368
abde	355	2072	0.805	0.146	–	abcde	3290	4232	0.817	0.437	–	abcde	0.816	0.364
bcde	355	2132	0.805	0.143	–	ade	3286	4142	0.816	0.442	–	ade	0.812	0.378
abcde	355	2132	0.805	0.143	–	acde	3286	4182	0.816	0.440	–	acde	0.812	0.374
de	342	1607	0.776	0.176		bde	3226	4176	0.801	0.436	–	bde	0.801	0.364
e	234	1086	0.531	0.177	–	e	2336	3209	0.580	0.421	–	e	0.575	0.374
d	156	609	0.354	0.204	–	d	1958	1783	0.486	0.523	–	d	0.473	0.469
b	126	885	0.286	0.125	–	b	1048	1436	0.260	0.422	–	b	0.263	0.336
a	75	468	0.170	0.138	–	a	656	864	0.163	0.432	–	a	0.164	0.354
c	63	430	0.143	0.128	–	c	408	712	0.101	0.364	–	c	0.105	0.292
High-quality		Informedness		Rec.	Prec.				Informedness	Rec.	Prec.		Informdness	Prec.
d	150	222	0.590	0.769	0.403	d	121	107	0.183	0.513	0.531	d	0.418	0.629
de	190	498	0.572	0.974	0.276	cd	121	107	0.183	0.513	0.531	de	0.387	0.877
bd	183	504	0.532	0.939	0.266	ad	121	107	0.183	0.513	0.531	bd	0.367	0.861
cde	190	597	0.493	0.974	0.241	acd	121	107	0.183	0.513	0.531	bcd	0.332	0.861
bcd	183	558	0.488	0.939	0.247	c	0	0	0.000	0.000	–	cd	0.332	0.629

Table 9 continued

High-quality		Informedness		Rec.	Prec.				Informedness	Rec.	Prec.		Informdness	Prec.
d	150	222	0.590	0.769	0.403	d	121	107	0.183	0.513	0.531	d	0.418	0.629
e	128	376	0.353	0.656	0.254	a	0	0	0.000	0.000	–	e	0.260	0.657
b	81	372	0.115	0.415	0.179	c	0	0	0.000	0.000	–	b	0.215	0.624
a	81	438	0.062	0.415	0.156	b	188	268	−0.031	0.797	0.412	c	−0.037	0.084
c	36	188	0.033	0.185	0.161	e	155	244	−0.096	0.657	0.388	a	−0.092	0.188
Medium-quality		F-measure		Rec.	Prec.				F-measure	Rec.	Prec.		F-measure	Rec.
e	132	882	0.205	0.478	0.130	d	81	90	0.626	0.921	0.474	e	0.243	0.484
ce	132	882	0.205	0.478	0.130	ad	81	90	0.626	0.921	0.474	ce	0.243	0.484
ae	132	948	0.195	0.478	0.122	bd	81	90	0.626	0.921	0.474	ace	0.232	0.484
ace	132	948	0.195	0.478	0.122	cd	81	90	0.626	0.921	0.474	ae	0.232	0.484
de	265	2588	0.169	0.96	0.093	de	81	90	0.626	0.921	0.474	cde	0.204	0.951
e	132	882	0.205	0.478	0.130	d	81	90	0.626	0.921	0.474	e	0.243	0.484
d	225	2310	0.160	0.815	0.089	e	44	28	0.550	0.500	0.611	d	0.199	0.841
b	105	1155	0.137	0.380	0.083	a	0	0	0.000	0.000	0.000	b	0.129	0.288
a	0	66	0.000	0.000	0.000	b	0	0	0.000	0.000	0.000	c	–	0.000
c	0	0	0.000	0.000	–	c	0	0	0.000	0.000	0.000	a	–	0.000
∗bde	824	5621	–	0.128^1	0.904^2	∗abde	3559	4588	–	0.437^1	0.818^2	∗de	0.447^1	0.573^2

MM, main metric; TM, tiebreaker metric; Rec., recall (R) (^{2}); Prec. (P) (^{1}), precision
Tools: a—phpSAFE, b—RIPS, c—WAP,d—Pixy,e—WeVerca. Inf. - Informedness. ∗See Table 8

to facilitate the determination of the scenario-specific impact. The tables also include the ranking of the individual ASATs, as reference. The results of all combinations can be consulted online at [29].

5.1 Comparing the results of the WPD and the SD

For both datasets and considering the SQLi and XSS vulnerabilities together, the best solution includes the ASATs of the best solution for each class of vulnerability, with two exceptions. First, for the WPD and for the low-quality scenario, the best solution excludes RIPS because it reported many FPs for XSS and only one TP for SQLi. Second, for the SD and for the medium-quality scenario the best solution is the same as for XSS which excludes the ASAT of the best solution for SQLi.

The best solutions for both datasets are very different. For the WPD most of the solutions are composed by several tools, while for the SD most of the solutions consist of a single tool, excepted for the highest-quality scenario.

The overall results show that the values of the main metrics of all best solutions for the SD are lower than for the WPD, meaning that the individual effectiveness of the ASATs is much lower for the SD.

For instance, in several cases the worst individual tool (e.g., Pixy) for the WPD is the best individual tool for the SD. The inverse occurred for other tools (e.g., WAP for the highest-quality scenario). Pixy tool is old but was included in five out of six best solutions using the SD. Moreover, the tool is the best solution for the SD in three cases. As stated before, the tool is limited to analyzing OOP. Therefore, the good results of Pixy for these three cases are due to the presence of POP in most of the test cases. It means that for the SD plugins, Pixy has a good effectiveness analyzing POP test cases and the other tools have low effectiveness because they fail analyzing these sample POP test cases. For example, phpSAFE, RIPS and WAP did not report any vulnerability in several cases (see Table 9). For the WPD, Pixy was only included in one out of eight best solutions. This occurred because in this scenario about 1/3 of the plugins are POP and 49% of the LLOC are POP coming both from the POP plugins and OOP with POP code.

For the WPD, WAP reports the fewest FPs (high precision) and is included in seven out of eight solutions. However, for the SD, WAP has low precision and is never included in the best solutions. A possible reason is that WAP uses data mining to identify FPs using a machine learning classifier. Perhaps, the classifier has to be better trained for SDs.

For the WPD, WeVerca is ranked in the middle or below for all cases. However, for the SD the tool was ranked first or second for all cases, except for the XSS and for the high-quality scenario where the tool was ranked in the last position. The tool reported both the highest number of TPs and FPs for most of the scenarios and class of vulnerability. Therefore, the tool needs to be improved in order to report less FPs.

For the highest-quality scenario the values of the main metrics are similar for both datasets. In contrast, the values for the tiebreaker metrics are very low for the SD (e.g., 0.146 for SQLi) and high for the WPD (e.g., 0.929 for SQLi). Therefore, the tools have better effectiveness for the WPD than for the SD.

5.2 Testing the hypotheses

Based on our findings, the first five hypotheses stated in the introduction are false. Hypothesis H_1 is false because we found many cases where adding a ASAT to an existing combination of ASATs, does not increase the number of vulnerabilities found (e.g., for the highest-quality scenario and XSS: *ab*, *abe*, *abce*). On the other hand, we also observed that the number of FPs does not always increase with the number of ASATs in a combination (e.g., for the plugins, the medium-quality scenario and SQLi: *ab*, *abe*, *abde*). The best solution for vulnerability detection depends on the chosen scenario and on class of vulnerability. Therefore, hypotheses H_3 and H_4 are both false. In fact, the detection capabilities of the ASATs are not uniform across the two classes of vulnerabilities. The same occurs for combinations of ASATs. Moreover, in almost all cases the values of the metrics for XSS vulnerabilities are better than for the SQLi vulnerabilities [31]. The best combination of ASATs regardless the classes of vulnerabilities is different in several cases. Therefore, the hypothesis H_5 is false. The approach was successfully applied to other kind of dataset with similar number of P and N instances but with applications (test cases) with very small sizes (i.e., LLOC). Therefore, the hypothesis H_6 is not false. The results for the SD show that we can derive similar conclusions for the hypotheses H_1 to H_5. Therefore, the hypothesis H_7 (The results for the hypotheses H_1 to H_5 are the same for other kind of applications) is not false.

In summary, the main advantage of combining the results of several ASATs is the identification of more vulnerabilities. In fact, for several cases there are ASATs that individually did not find any vulnerabilities or found few vulnerabilities in many plugins. Moreover, even using all the ASATs some vulnerabilities remain undetected. However combining many tools can be counterproductive in some cases as that will not lead to the detection of more vulnerabilities, but will increase the number of FPs reported, which then need to be verified manually by the developers. Finally, identifying the strengths and limitations of ASATs, helps developers to determinate how such tools can be combined to provide a more thorough analysis of the software depending on the specificities of the scenario and on the class of vulnerability being analyzed [31].

5.3 Threats to validity

(1) *Datasets* The datasets across the various scenarios are unbalanced, which may affect the results in some cases. For example, the low-quality scenario for the SD has no test cases. In the same scenario, for the WPDs and for SQLi, only one ASAT reported vulnerabilities, which may limit our study. Works using other tools are needed for improving the characterization of the vulnerable/non-vulnerable LOCs in the dataset.
(2) *Vulnerabilities* There are limitations regarding the scope of the datasets in this experiment, since it considers only WordPress plugins and one test suite of synthetic test cases, SQLi, and XSS vulnerabilities.
(3) *Characterizing VLOCs and NVLOCs* The number of VLOCs for the WPD is based on the results of the ASATs and manual inspection in order to be classified as VLOC

or NVLOC. As with any classification there is a potential for misclassification, that could significantly affect the reliability of the results. The use of more ASATs and a larger list of CVE vulnerabilities may reduce this threat to validity. However, even using several state-of-the-art and widely used commercial ASATs there are missed vulnerabilities [17].
(4) *Free ASATs* All ASATs used in this study are free. Pixy is not updated since 2007 and RIPS has only been developed as open source until 2014. On the other hand, WAP, phpSAFE, and WeVerca are recent tools prepared do analyze OOP code. There are several commercial and other free ASATs, thus, the results of this study are only valid for the tools used.
(5) *Tools configuration* The dataset used in this study was collected with all tools configured by default for PHP entry points, sensitive sinks and sanitization functions. The results of the tools may be improved (+TP and −FP) by adjusting their configuration settings for WordPress built-in database functions, sanitization and escaping routines.
(6) *Language domains* Both datasets and the tools are for PHP language. Our choice was deliberate because PHP powers over 82% of web applications [9]. Works using other languages such as ASP.NET and Java are need.

6 Related work

Rutar et al. [33] studied five well-known ASATs on a small set of Java programs with different sizes from various domains. They concluded that the results of each tool are highly correlated with the techniques used for finding bugs, and that no single tool can be considered the best to detect defects. They proposed a meta-tool for automatically combining and correlating their outputs. This meta-tool is based on a set of scripts that combine the results of the various tools in a common format. The bugs found are not manually reviewed, thus, there is no distinction between TP and FP. The metric used to evaluate and compare the tools was the number of bugs found by each tool.

Meng et al. [27] proposed an approach to merge the results of multiple ASATs. The user specifies the programs to be analyzed and chooses the classes of bugs to be scanned. After determined which tools can search for the specified class of bugs, generated the necessary configurations to run the tools, run the tools, combined the outputs in a single report, and applied two prioritizing policies to rank the results. Meng et al. used their approach to conclude that developers could benefit from more than one ASAT. The results were not classified as TP and FP and the authors did not propose any metric to evaluate the approach. The dataset was composed by a small Java program that is not representative of real applications. Therefore, with such limited validation it is very difficult to assess the strength and drawbacks of the solution.

Wang et al. [36] proposed an approach that combines multiple ASATs in a simple Web Service. The user has the possibility to choose the classes of bugs to scan and upload the source code and auxiliary information such as the programming language and the classes of bugs to be scanned. The tools perform the analysis of the source code and results are merged in a way that the same defect is only reported once. The combined results are sent back to the user. The approach was evaluated in terms of running time when combining two ASATs, but the experiments were quite limited, having just a single Java test case. Therefore, the solution lacks the validation of the effectiveness of the vulnerability detection when using a combination of ASATs.

The National Security Agency (NSA) Center for Assured Software (CAS) specified a methodology, the CAS Static Analysis Tool Study Methodology, that measures and rates the effectiveness of ASATs in a standard and repeatable manner [25]. The metrics used are precision, recall, F-score (i.e., F-measure), and discrimination rate (DR). A discrimination occurs if a ASAT reports a vulnerability in the vulnerable test case (TP) and keeps quiet in the non-vulnerable test case (TN). The CAS has created a collection over 81,000 synthetic C/C++ and Java programs with known flaws, which is called Juliet Test Suite [7]. Each test case is a slice of artificial code having exactly one flaw and at least one non-flaw construct similar to the vulnerability. In 2011, the CAS conducted a study with the purpose of determining the capabilities of five ASATs for C/C++ and Java [3]. In this study, they proposed the combination of two ASATs to show that adding a second ASAT might complement the first one. However, the evaluation of the combinations is limited because it is based on the metrics recall and DR. The metric recall does not consider the number of FPs reported, and the DR severely penalizes ASATs that report many vulnerabilities but also reports FPs. Furthermore, they also evaluated the overall coverage (recall) of four combinations of ASATs. The ASATs were labeled with a number from 1 to 5. Then, the combination of ASATs: 12, 123, 1234, and 12345, were evaluated across all the test cases. They concluded that the recall increases as the number of tools increases. However, this evaluation is limited as there are many combinations that were not considered.

Unlike the approaches above, that use a small dataset or synthetic simple test cases, our approach is based on a considerable number of real plugins and four representative vulnerability detection scenarios. Moreover, the dataset is characterized in terms of vulnerable and non-vulnerable LOCs for a more precise classification of the results produced by the tools with respect to TP and FP. Another difference is that our study uses a single main metric to evaluate the combinations of ASATs in each scenario that takes into consideration the goals of the vulnerability detection in that scenario. In fact, the approaches previously referred use a simple metric such as the number of bugs that each tool founds or several metrics. In this case, the user has the task of choosing the metric that seem most appropriate and use it for evaluating a single ASAT or combinations of ASATs, without any further guidance.

7 Conclusion

In this work, we addressed the problem of combining the output of several ASATs searching for SQLi and XSS vulnerabilities in two different datasets, one with WordPress Plugins and another with PHP synthetic test cases. We proposed a generic methodology, which can be used with any dataset and free or commercial tools. The dataset is organized in four scenarios of increasing criticality and each scenario uses different metrics to rank the tools.

Our findings revealed that combining the outputs of several free ASATs do not always improve the vulnerability detection performance. Thus, the best solution can be a single tool or a combination of tools that may not include all the tools under evaluation. Combining multiple ASATs has benefits due to the complementarity of the produced results. However, for solutions including ASATs that report many FPs the overall performance is worse in some scenarios. Our results highlighted a considerable variance on the rates of TPs and FPs among the ASATs and the datasets. This means that, overall, the best combination of ASATs is highly dependable on the specific situation, and it should be selected after a properly targeted benchmarking procedure, such as ours. These results are very useful for software engineers choosing an ASAT solution for a concrete project with particular criticality and for ASAT developers improving their tools.

Future work will focus on three main directions. First, we plan to improve and expand the datasets to include other kinds of applications and classes of vulnerabilities. Second, we are going to research novel ways to combine the results regardless classes of vulnerabilities. Finally, we intend to investigate different strategies of combining the ASATs such as intersection, majority voting and N-out-of-N.

References

1. https://freeformdynamics.com/wp-content/uploads/legacy-pdfs/pdf/insidetrack/2017/17-03-Managing_Application_Security_Risk.pdf. Accessed 17 Mar 2017
2. https://www.owasp.org/index.php/Top_10_2017-Top_10. Accessed 20 Mar 2017
3. https://media.blackhat.com/bh-us-11/Willis/BH_US_11_WillisBritton_Analyzing_Static_Analysis_Tools_WP.pdf (2011). Accessed 6 Apr 2017
4. WPScan Vulnerability Database. https://wpvulndb.com/. Accessed 26 Oct 2015
5. Website hacked trend report 2016-Q1 (2016) https://sucuri.net/website-security/Reports/Sucuri-Website-Hacked-Report-2016Q1.pdf. Accessed 6 Apr 2017
6. Wordpress plugin directory. https://wordpress.org/plugins/. Accessed 29 Dec 2016
7. NIST SARD Project. http://samate.nist.gov/SRD. Accessed 23 Feb 2017
8. https://colorlib.com/wp/is-wordpress-websites-secure/. Accessed 09 March 2017
9. https://w3techs.com/technologies. Accessed March 2018
10. Antunes N, Vieira M (2015) On the metrics for benchmarking vulnerability detection tools. In: 2015 45th Annual IEEE/IFIP international conference on dependable systems and networks, pp 505–516
11. Backes M, Rieck K, Skoruppa M, Stock B, Yamaguchi F (2017) Efficient and flexible discovery of PHP application vulnerabilities. In: 2017 IEEE european symposium on security and privacy (EuroS&P), pp 334–349. IEEE. https://doi.org/10.1109/EuroSP.2017.14. http://ieeexplore.ieee.org/document/7961989/
12. Baggen R, Correia JP, Schill K, Visser J (2012) Standardized code quality benchmarking for improving software maintainability. Softw Qual J 20(2):287–307
13. Beller M, Bholanath R, McIntosh S, Zaidman A (2016) Analyzing the state of static analysis: a large-scale evaluation in open source software. In: 2016 IEEE 23rd international conference on software analysis, evolution, and reengineering, vol 1, pp 470–481
14. Dahse J, Holz T (2014) Simulation of built-in PHP features for precise static code analysis. In: Proceedings 2014 network and distributed system security symposium. Internet Society, Reston, VA
15. Díaz G, Bermejo JR (2013) Static analysis of source code security: assessment of tools against SAMATE tests. Inf Softw Technol 55(8):1462–1476
16. Forbes: will the demand for developers continue to increase? https://forbes.com/sites/quora/2017/01/20/will-the-demand-for-developers-continue-to-increase/#7e502b681c3f. Accessed 15 May 2017
17. Goseva-Popstojanova K, Perhinschi A (2015) On the capability of static code analysis to detect security vulnerabilities. Inf Softw Technol 68:18–33
18. Hauzar D, Kofron J (2015) Framework for Static Analysis of PHP Applications. In: Boyland JT (ed) 29th European conference on object-oriented programming (ECOOP 2015), Leibniz international

proceedings in informatics (LIPIcs), vol 37, pp 689–711. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany

19. Imperva: Imperva web application attack report (WAAR). http://www.imperva.com/download.asp?id=509 (2015). Accessed 22 May 2017
20. Institute P (2015) Annual consumer studies. http://www.ponemon.org/. Accessed 22 May 2017
21. Johnson B, Song Y, Murphy-Hill E, Bowdidge R (2013) Why don't software developers use static analysis tools to find bugs? In: 35th International conference on software engineering. IEEE, pp 672–681
22. Jovanovic N, Kruegel C, Kirda E (2006) Pixy: a static analysis tool for detecting web application vulnerabilities. In: 2006 IEEE symposium on security and privacy, pp 6–263
23. Landi W (1992) Undecidability of static analysis. ACM Lett Program Lang Syst 1(4):323–337
24. Livshits VB, Lam MS (2005) Finding security vulnerabilities in java applications with static analysis. In: Proceedings of the 14th conference on USENIX security symposium, vol 14, SSYM'05. USENIX Association, Berkeley, CA, USA, pp 18–18
25. Meade FG. https://samate.nist.gov/docs/CAS%202012%20Static%20Analysis%20Tool%20Study%20Methodology.pdf. Accessed 5 May 2017
26. Medeiros I, Neves NF, Correia M (2014) Automatic detection and correction of web application vulnerabilities using data mining to predict false positives. In: Proceedings of the 23rd international conference on world wide web, WWW '14. ACM, NY, USA, pp 63–74
27. Meng N, Wang Q, Wu Q, Mei H (2008) An approach to merge results of multiple static analysis tools (short paper). In: 2008 The eighth international conference on quality software, pp 169–174
28. NIST: Software assurance metrics and tool evaluation. http://samate.nist.gov/. Accessed 28 Nov 2016
29. Nunes P. https://github.com/pjcnunes/Computing2018. Accessed 15 July 2018
30. Nunes P, Fonseca J, Vieira M (2015) phpSAFE: a security analysis tool for OOP web application plugins. In: 45th Annual IEEE/IFIP international conference on dependable systems and networks, DSN 2015, Rio de Janeiro, Brazil, June 22–25, 2015, pp 299–306
31. Nunes P, Medeiros I, Fonseca J, Neves N, Correia M, Vieira M (2017) On combining diverse static analysis tools for web security: an empirical study. In: 2017 13th European dependable computing conference (EDCC), pp 121–128
32. Pichler M. PHP depend. https://pdepend.org/. Accessed 03 Nov 2016
33. Rutar N, Almazan CB, Foster JS (2004) A comparison of bug finding tools for java. In: Proceedings of the 15th international symposium on software reliability engineering, ISSRE '04. IEEE Computer Society, Washington, DC, USA, pp 245–256
34. Stivalet B, Fong E (2016) Large scale generation of complex and faulty PHP test cases. In: 2016 IEEE International conference on software testing, verification and validation (ICST), pp 409–415
35. Vogt P, Nentwich F, Jovanovic N, Kirda E, Kruegel C, Vigna G (2007) Cross site scripting prevention with dynamic data tainting and static analysis. In: NDSS, vol 2007, p 12
36. Wang Q, Meng N, Zhou Z, Li J, Mei H (2008) Towards SOA-based code defect analysis. In: IEEE international symposium on service-oriented system engineering, 2008. SOSE '08, pp 269–274

Affiliations

Paulo Nunes[1,4] · Ibéria Medeiros[2] · José Fonseca[1,4] · Nuno Neves[2] · Miguel Correia[3] · Marco Vieira[4]

Ibéria Medeiros
imedeiros@di.fc.ul.pt

José Fonseca
josefonseca@ipg.pt

Nuno Neves
nuno@di.fc.ul.pt

Miguel Correia
miguel.p.correia@ist.utl.pt

Marco Vieira
mvieira@dei.uc.pt

[1] Unidade de Investigação para o Desenvolvimento do Interior, Guarda, Portugal

[2] LASIGE, Faculdade de Ciências, Universidade de Lisboa, Lisbon, Portugal

[3] INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal

[4] CISUC, University of Coimbra, Coimbra, Portugal