Improving the Effectiveness of Web Application Vulnerability Scanning

Marc Rennhard

School of Engineering
Zurich University of Applied Sciences
Winterthur, Switzerland
Email: rema@zhaw.ch

Damiano Esposito, Lukas Ruf, Arno Wagner

Consecom AG
Zurich, Switzerland
Email: Damiano.Esposito,Lukas.Ruf,
Arno.Wagner@consecom.com

***Abstract*—Using web application vulnerability scanners is very appealing as they promise to detect vulnerabilities with minimal configuration effort. However, using them effectively in practice is often difficult. Two of the main reasons for this are limitations with respect to crawling capabilities and problems to perform authenticated scans. In this paper, we present JARVIS, which provides technical solutions that can be applied to a wide range of vulnerability scanners to overcome these limitations and to significantly improve their effectiveness. To evaluate JARVIS, we applied it to five freely available vulnerability scanners and tested the vulnerability detection performance in the context of seven deliberately insecure web applications. A first general evaluation showed that by using the scanners with JARVIS, the number of detected vulnerabilities can be increased by more than 100% on average compared to using the scanners without JARVIS. A significant fraction of the additionally detected vulnerabilities is security-critical, which means that JARVIS provides a true security benefit. A second, more detailed evaluation focusing on SQL injection and cross-site scripting vulnerabilities revealed that JARVIS improves the vulnerability detection performance of the scanners by 167% on average, without increasing the fraction of reported false positives. This demonstrates that JARVIS not only manages to greatly improve the vulnerability detection rate of these two highly security-critical types of vulnerabilities, but also that JARVIS is very usable in practice by keeping the false positives reasonably low. Finally, as the configuration effort to use JARVIS is small and as the configuration is scanner-independent, JARVIS also supports using multiple scanners in parallel in an efficient way. In an additional evaluation, we therefore analyzed the potential and limitations of using multiple scanners in parallel. This revealed that using multiple scanners in a reasonable way is indeed beneficial as it further increases the number of detected vulnerabilities without a significant negative impact on the reported false positives.**

Keywords*–*Web Application Security; Vulnerability Scanning; Vulnerability Detection Performance; Authenticated Scanning; Combining Multiple Scanners.

I. Introduction

This paper is an extended and revised version of our conference paper [1] that was published at ICIMP 2018 (the thirteenth International Conference on Internet Monitoring and Protection). Compared to the original version, this paper contains a much more elaborate evaluation to further demonstrate the effectiveness and usefulness of the presented approach.

Security testing is of great importance to achieve security and trustworthiness of software and systems. Security testing can be performed in different ways, ranging from completely manual methods (e.g., manual source code analysis), to semi-automated methods (e.g., analyzing a web application using an interceptor proxy), to completely automated ways (e.g., analyzing a web service using a vulnerability scanner).

Ideally, at least parts of security testing should be automated. One reason for this is that it increases the efficiency of a security test and frees resources for those parts of a security test that cannot be easily automated. This includes, e.g., access control tests, which cannot really be automated as a testing tool does not understand which users or roles are allowed to perform what functions. Another reason is that automating security tests enables performing continuous and reproducible security tests, which is getting more and more important in light of short software development cycles.

There are different options to perform automated security testing. The most popular approaches include static and dynamic code analysis and vulnerability scanning. Vulnerability scanners test a running system "from the outside" by sending specifically crafted data to the system and by analyzing the received response. Among vulnerability scanners, web application vulnerability scanners are most popular, as web applications are very prevalent, are often vulnerable, and are frequently attacked [2]. Note also that web applications are not only used to provide typical services such as information portals, e-shops or access to social networks, but they are also very prevalent to configure all kinds of devices attached to the Internet, which includes, e.g., switches, routers and devices in the Internet of Things (IoT). This further underlines the importance of web application security testing.

At first glance, using web application vulnerability scanners seems to be easy as they claim to uncover many vulnerabilities with little configuration effort – as a minimum, they only require the base URL of the application to test as an input. However, the effective application of web application vulnerability scanners in practice is far from trivial. The following list summarizes some of the limitations:

1) The detection capabilities of a scanner are directly dependent on its crawling performance: If a scanner cannot find a specific resource in a web application, it cannot test it and will not find vulnerabilities associated with this resource. Previous work shows that the crawling performance of different scanners varies significantly [3], [4].
2) To test areas of a web application that are only reachable after successful user authentication, the scanners must authenticate themselves during crawling and testing. While most scanners can be configured so they can perform logins, they typically do not support

all authentication methods used by different web applications. Also, scanners sometimes log out themselves (e.g., by following a logout link) during testing and sometimes have problems to detect whether an authenticated session has been invalidated. Overall, this makes authenticated scans unreliable or even impossible in some cases.

3) To cope with these limitations, scanners usually provide configuration options, which can increase the number of detected vulnerabilities [5]. This includes, e.g., specifying additional URLs that can be used by the crawler as entry points, manually crawling the application while using the scanner as a proxy so it can learn the URLs, and specifying an authenticated session ID that can be used by the scanner to reach access-protected areas of the application if the authentication method used by the web application is not supported. However, using these options complicate the usage of the scanners and still does not always deliver the desired results.

4) With respect to the number and types of the reported findings, different vulnerability scanners perform differently depending on the application under test [6]–[10]. Therefore, when testing a specific web application, it is reasonable to use multiple scanners in parallel and combine their findings. However, the limitations described above make this cumbersome and difficult, as each scanner has to be configured and optimized differently.

In this paper, we present JARVIS, which provides technical solutions to overcome limitations 1 and 2 in the list above. Using JARVIS requires only minimal configuration, which overcomes limitation 3. And finally, JARVIS and its usage are independent of specific vulnerability scanners and can be applied to a wide range of scanners available today, which overcomes limitation 4 and which provides an important basis to use multiple scanners in parallel in an efficient way.

To demonstrate the effectiveness and usefulness of JARVIS, to quantify how much it can improve the vulnerability detection performance of scanners, and to learn more about the potential and limitations of combining multiple scanners, this paper also includes a detailed evaluation. In this evaluation, JARVIS was applied to the five freely available scanners listed to Table I.

TABLE I. ANALYZED WEB APPLICATION VULNERABILITY SCANNERS

Scanner	*Version/Commit*	*URL*
Arachni	1.5-0.5.11	http://www.arachni-scanner.com
OWASP ZAP	2.5.0	https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
Skipfish	2.10b	https://code.google.com/archive/p/skipfish/
Wapiti	r365	http://wapiti.sourceforge.net
w3af	cb8e91af9	https://github.com/andresriancho/w3af

The choice for using freely available scanners was mainly driven by the desire to evaluate the performance of using multiple scanners in parallel. This is a much more realistic scenario with freely available scanners as commercial ones often have a hefty price tag. Also, several previous works concluded that freely available scanners do not perform worse than commercial scanners [3], [4], [11], [12]. Arguments for using the scanners in Table I instead of using others include our previous experience with these scanners, that these scanners are among the most popular used scanners in practice, and that they perform well in general according to [4], [11], [12].

The main contributions of this paper are the following:

- Technical solutions to improve the crawling coverage and the reliability of authenticated scans of web application vulnerability scanners. In contrast to previous work (see Section II), our solutions cover both aspects, can easily be applied to a wide range of scanners available today, and require only minimal, scanner-independent configuration.
- A general evaluation that shows that by using these technical solutions, the vulnerability detection performance of the scanners included in the evaluation can be improved by more than 100% on average. Many of the additionally reported vulnerabilities are security-critical, which means that JARVIS provides a true security benefit.
- A more detailed evaluation focusing on SQL injection and cross-site scripting vulnerabilities that demonstrates that the vulnerability detection performance of the scanners with respect to these two types of highly relevant vulnerabilities can be increased by 167% on average, without increasing the fraction of reported false positives.
- A final evaluation that shows that using multiple scanners in a reasonable way is beneficial as it further increases the number of detected vulnerabilities without a significant negative impact on the reported false positives.

The remainder of this paper is organized as follows: Section II covers relevant related work. Section III describes the technical solutions to overcome the limitations of today's web application vulnerability scanners. Section IV contains the general evaluation results and Section V provides a more detailed evaluation focusing on SQL injection and cross-site scripting vulnerabilities. The final part of the evaluation is provided in Section VI, where the benefits and limitations of using multiple scanners in parallel are analyzed. Section VII concludes this work.

II. RELATED WORK

Several work has been published on the crawling coverage and detection performance of web application vulnerability scanners. In [3], more than ten scanners were compared, with the main results that good crawling coverage is paramount to detect many vulnerabilities and that freely available scanners perform as well as commercial ones. The same is confirmed in [4], which covers more than 50 free and commercial scanners. The works by Chen [11], which covers about 20 scanners and which is updated regularly, and by El Idrissi et al. [12], which includes 11 scanners in its evaluation, also result in the conclusion that free scanners perform as well as commercial ones. In [5], Suto concludes that when carefully training or configuring a scanner, detection performance is improved, but this also significantly increases the complexity and time effort needed to use a scanner. Furthermore, Bau et al. demonstrate that the eight scanners they used in their analysis have different strengths, i.e., they find different vulnerabilities [6]. The same is confirmed by Vega et al. [7], which in addition compare the vulnerabilities detected by the four scanners in their evaluation with alerts reported by the intrusion detection system (IDS) Snort [13]. Qasaimeh et al. conclude that the five scanners used in their evaluation not only perform differently with respect to the number of findings detected, but also with respect to the number of false positives [8]. Smaller studies by using two and three scanners were done in [9] and [10], respectively, which confirm that different scanners have different strengths with respect to detection capabilities.

Other work specifically aimed at improving the coverage of vulnerability scanning. In [14], it is demonstrated that by considering the state changes of a web application when it processes requests, crawling and therefore scanning performance can be improved. In [15], van Deursen et al. present a Selenium WebDriver-based crawler called Crawljax, which improves crawling of Ajax-based web applications. The same is achieved by Pellegrino et al. by dynamically analyzing JavaScript code in web pages [16]. In [17], Zulla discusses methods to improve web vulnerability scanning in general, including approaches to automatically detect login forms on web pages.

Our work presented in this paper builds upon this previous work, in particular on the observations that freely available scanners perform similarly as commercial ones, that different scanners have different strengths with respect to detection capabilities, and that good crawling coverage is paramount to detect many vulnerabilities. Besides this, however, our work goes significantly beyond existing work. First of all, the presented solution – JARVIS – not only addresses crawling coverage but also the reliability of authenticated scans, which has a significant impact on the number of vulnerabilities that can be detected. In addition, JARVIS is scanner-independent, which means it can easily be applied to most vulnerability scanners available today. Furthermore, we provide a detailed evaluation using several scanners and several test applications that truly demonstrates the benefits and practicability of our technical solutions. And finally, to our knowledge, our work is the first one to quantitatively evaluate the benefits and limitations when combining multiple scanners.

III. Technical Solutions to Improve Web Application Vulnerability Scanning

One way to improve the vulnerability detection performance of web application vulnerability scanners is to directly adapt one or more scanners that are available today. However, the main disadvantage of this approach is that this would only benefit one or a small set of scanners and would be restricted to scanners that are provided as open source software. Therefore, a proxy-based approach was chosen. The advantages of this approach are that it is independent of any specific scanner, that it does not require adaptation of any scanner, and that it can be used with many scanners that are available today and most likely also with scanners that will appear in the future. The basic idea of this proxy-based approach is illustrated in Figure 1.

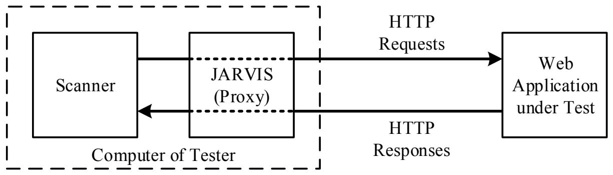


Figure 1. Proxy-based Approach of JARVIS.

A proxy-based approach means that JARVIS, which provides the technical solutions to overcome the limitations of web application vulnerability scanners, acts as a proxy between the scanner and the web application under test. This gives JARVIS access to all HTTP requests and responses exchanged between the scanner and the web application, which enables JARVIS to control the entire crawling and scanning process and to adapt requests or responses as needed. This proxy-based approach is possible because most scanners are proxy-aware, i.e., they support configuring a proxy through which communication with the web application takes place. Note that JARVIS can basically be located on any reachable host, but the typical scenario is using JARVIS on the same computer as the web application vulnerability scanner (e.g., on the computer of the tester).

As a basis for JARVIS, the community edition version 1.7.19 of Burp Suite [18] is used. Burp Suite is a tool that is intended to assist a tester during web application security testing. It is usually used as a proxy between the browser of the tester and the web application under test and supports recording, intercepting, analyzing, modifying and replaying HTTP requests and responses. Therefore, Burp Suite already provides many basic functions that are required to implement JARVIS. In addition, Burp Suite provides an application programming interface (API) so it can be extended and JARVIS makes use of this API.

JARVIS consist of two main components. The first is described in Section III-A and aims at improving the test coverage of scanners. This component should especially help scanners that have a poor crawling performance. The second component, described in Section III-B, aims at improving the reliability of authenticated scans and should assist scanners that have limitations in this area. Finally, Section III-C gives a configuration example when using JARVIS to demonstrate that the configuration effort is small.

A. Improving Test Coverage

Improving test coverage could be done by replacing the existing crawler components of the scanners with a better one (see, e.g., [14]–[16]). While this may be helpful for some scanners, it may actually be harmful for others, in particular if the integrated crawler works well. Therefore, an approach was chosen that does not replace but that assists the crawling components that are integrated in the different scanners. The idea is to supplement the crawlers with additional URLs (beyond the base URL) of the web application under test. These additional URLs are named *seeds* as they are used to seed the crawler components of the scanners. Intuitively, this should significantly improve crawling coverage, in particular if the integrated crawler is not very effective. To get the additional URLs of a web application, two different approaches are used: endpoint extraction from the source code of web applications and using the detected URLs of the best available crawler(s).

Endpoint extraction means searching the source code (including configuration files) of the web application under test for URLs and parameters. The important benefits of this approach are that it can detect URLs that are hard to find by any crawler and that it can uncover hidden parameters of requests (e.g., debugging parameters). To extract the endpoints, ThreadFix endpoint CLI [19] was used, which supports many common web application frameworks (e.g., JSP, Ruby on Rails, Spring MVC, Struts, .NET MVC and ASP.NET Web Forms). In addition, further potential endpoints are constructed by appending all directories and files under the root directory of the source code to the base URL that is used by the web application under test. This is particularly effective when scanning web applications based on PHP.

Obviously, endpoint extraction is only possible if the source code of the application under test is available. If that is not the case, the second approach comes into play. The idea here is to use the best available crawler(s) to gather additional URLs. As will be shown later, the scanner Arachni provides good crawling performance in general, so Arachni is a good starting point as a tool for this task. Of course, it is also possible to combine both approaches to determine the seeds: extract the endpoints from the source code (if available) *and* get URLs with the best available crawler(s).

Once the seeds have been derived, they must be injected into the crawler component of the scanners. To do this, most scanners provide a configuration option. However, this approach has its limitations as such an option is not always available and usually only supports GET requests but no POST requests. Therefore, the seeds are injected by JARVIS. To do this, four different approaches were implemented based on *robots.txt*, *sitemap.xml*, a landing page, and the index page.

Using *robots.txt* and *sitemap.xml* is straightforward. These files are intended to provide search engine crawlers with information about the target web site and are also evaluated by most crawler components of scanners. When the crawler component of a scanner requests such a file, JARVIS supplements the original file received from the web application with the seeds (or generates a new file with the seeds in case the web application does not contain the file at all). Both approaches work well but are limited to GET request.

The other two approaches are more powerful as they also support POST request. The landing page-based approach places all seeds as links or forms into a separate web page (named *landing.page*) and the scanner is configured to use this page as the base URL of the web application under test (e.g., http://www.example.site/landing.page instead of http://www.example.site). When the crawler requests the page, JARVIS delivers the landing page, from which the crawler learns all the seeds and uses them during the remainder of the crawling process. One limitation of this approach is that the altered base URL is sometimes interpreted as a directory by the crawler component of the scanners, which means the crawler does not request the landing page itself but tries to fetch resources below it. This is where the fourth approach comes into play. The index page-based approach injects seeds directly into the first page received from the web application (e.g., just before the </body> tag of the page *index.html*). Overall, these four approaches made it possible to successfully seed all scanners in Table I when used to test the web applications in the test set (see Section IV-A).

As an example, the effectiveness of the landing page-based approach is demonstrated. To do this, WIVET version 4 [20] is used, which is a benchmarking project to assess crawling coverage. Table II shows the crawling coverage that can be achieved with OWASP ZAP (in headless mode) and Wapiti when they are seeded with the crawling results of Arachni via a landing page.

TABLE II. CRAWLING COVERAGE

Scanner	*Raw Coverage*	*Coverage when Seeded with the Crawling Results of Arachni*
Arachni	92.86%	
OWASP ZAP	14.29%	96.43%
Wapiti	48.21%	96.43%

Table II shows that the raw crawling coverage of Arachni is already very good (92.86%), while Wapiti only finds about half of all resources and OWASP ZAP only a small fraction. By seeding OWASP ZAP and Wapiti with the crawling results of Arachni, their coverage can be improved drastically to 96.43%. This demonstrates that seeding via a landing page indeed works very well.

B. Improving Authenticated Scans

Performing authenticated scans in a reliable way is challenging for multiple reasons. This includes coping with various authentication methods, prevention of logouts during the scans, and performing re-authentication when this is needed (e.g., when a web application with integrated protection mechanisms invalidates the authenticated session when being scanned) to name a few. It is therefore not surprising that many scanners have difficulties to perform authenticated scans reliably.

To deal with these challenges, several modules were implemented in JARVIS. The first one serves to handle various authentication methods, including modern methods based on HTTP headers (e.g., OAuth 2.0). The module provides a wizard to configure authentication requests, can submit the corresponding requests, stores the authenticated cookies received from the web applications, and injects them into subsequent requests from the scanner to make sure the requests are interpreted as authenticated requests by the web application. The main advantages of this module are that it enables authenticated scans even if a scanner does not support the authentication method and that it provides a consistent way to configure authentication independent of a particular scanner.

Furthermore, a logout prevention module was implemented to make sure a scanner is not doing a logout by following links or performing actions which most likely invalidate the current session (e.g., change password or logout links). This is configured by specifying a set of corresponding URLs that should be avoided during the scan. When the proxy detects such a request, it blocks the request and generates a response with HTTP status code 200 and an empty message body. In addition, a flexible re-authentication module was developed. Re-authentication is triggered based on matches of configurable literal strings or regular expressions with HTTP response headers (e.g., the *location* header in a redirection response) or with the message body of an HTTP response (e.g., the occurrence of a keyword such as *login*).

C. Configuration Example

To give an impression of the configuration effort needed when using JARVIS, Table III lists the parameters that must be configured when scanning the test application BodgeIt (see Section IV-A). In this example, the seeds are extracted from the source code.

TABLE III. Example Configuration when Scanning BodgeIt

Parameter	*Value(s)*
Base URL	http://bodgeit/
Source code	~/bodgeit/
Authentication mode	POST
Authentication URL	http://bodgeit/login.jsp
Authentication parameters	username=test@test.test password=password
Out of scope	http://bodgeit/password.jsp http://bodgeit/register.jsp http://bodgeit/logout.jsp
Re-auth. search scope	HTTP response body
Re-auth. keywords	Login, Guest, user
Re-auth. keyword interpretation	Literal string(s)
Re-auth. case-sensitive	True
Re-auth. match indicates	Invalid session
Seeding approach(es)	Landing page, robots.txt, sitemap.xml

The entries in Table III are self-explanatory and show that the configuration effort is rather small. In particular, the configuration is independent of the actual scanner, which implies that when using multiple scanners in parallel (see Section VI), this configuration must only be done once and not once per scanner.

IV. General Evaluation

This section starts with a description of the evaluation setup. Then, it is analyzed how many vulnerabilities are reported when the scanners are used with and without the technical improvements described in Section III. Next, these vulnerabilities are analyzed in more detail to check how many unique vulnerabilities are detected and how severe they are. Finally, it is analyzed whether all vulnerabilities that can be detected by the scanners without using JARVIS are always also detected when JARVIS is used.

A. Evaluation Setup

Table IV lists the web applications that were used to evaluate the scanners (Cyclone Transfers and WackoPicko do not use explicit versioning).

TABLE IV. Web Applications used for the Evaluation

Application	*Version*	*URL*
BodgeIt	1.4.0	https://github.com/psiinon/bodgeit
Cyclone Transfers	–	https://github.com/thedeadrobots/bwa_cyclone_transfers
InsecureWebApp	1.0	https://www.owasp.org/index.php/Category:OWASP_Insecure_Web_App_Project
Juice Shop	2.17.0	https://github.com/bkimminich/juice-shop
NodeGoat	1.1	https://github.com/OWASP/NodeGoat
Peruggia	1.2	https://sourceforge.net/projects/peruggia/
WackoPicko	–	https://github.com/adamdoupe/WackoPicko

All these applications are deliberately insecure and well suited for security training and to test vulnerability scanners. The main reason why the applications in Table IV were chosen is because they cover various technologies, including Java, PHP, Node.js and Ruby on Rails.

The evaluation uses four different configurations that are identified as *-/-*, *S/-*, *-/A* and *S/A*. Basically, *S* indicates that seeding is used and *A* indicates that authenticated scans are used. The four configurations are described in more detail in Table V.

TABLE V. Configurations used during the Evaluation

Config.	JARVIS is Used	*The Scans are Executed...*
-/-	No	...without seeding and non-authenticated (i.e., using the basic configuration of the scanners by setting only the base URL)
S/-	Yes	...with seeding but non-authenticated (i.e., using the technical solution described in Section III-A)
-/A	Yes	...authenticated but without seeding (i.e., using the technical solution described in Section III-B)
S/A	Yes	...with seeding and authenticated (i.e., using both technical solutions described in Sections III-A and III-B)

As the source code of all the test applications is available, the endpoint extraction approach described in Section III-A is used for seeding in configurations *S/-* and *S/A*.

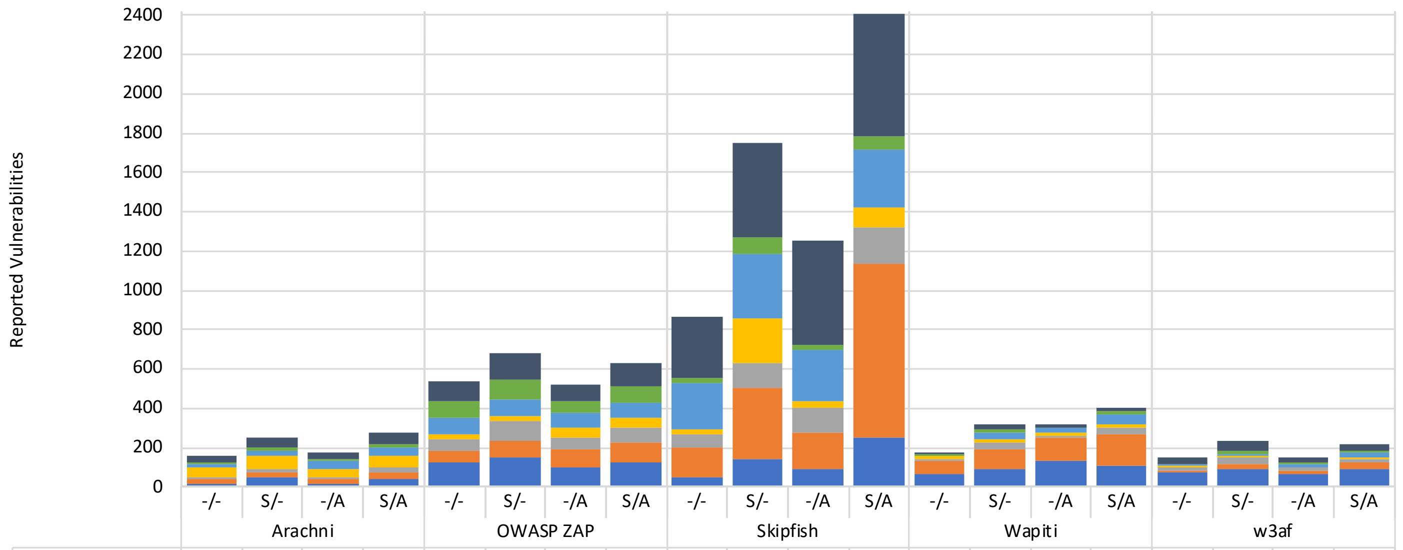
The test applications were run in a virtual environment that was reset to its initial state before each test run to make sure that every run is done under the same conditions and is not influenced by any of the other scans.

B. Total Number of Reported Vulnerabilities

The first evaluation analyzes the total number of vulnerabilities that are reported by the scanners when using the four different configurations described in Table V. Figure 2 illustrates the evaluation results. The height of the bars represents the number of vulnerabilities reported over all seven test applications and the different colors of the bars represent the number of reported vulnerabilities per test application. The table in the lower part of the figure also contains the number of vulnerabilities reported per test application.

The first observation when looking at Figure 2 is that some scanners identify many more vulnerabilities than others. For example, Skipfish reports about ten times as many findings as Arachni or w3af. However, this does not mean that Skipfish is the best scanner, because Figure 2 depicts the "raw number of vulnerabilities" reported by the scanners and does not consider whether the vulnerabilities include false positives or duplicate findings, or how severe the findings are. For instance, as will be seen in Section IV-C, about 75% of the vulnerabilities reported by Skipfish are rated as *info* or *low* (meaning they have only little security impact in practice), while the other scanners report a much smaller fraction of such findings.

More importantly, Figure 2 allows to do a first assessment about the impact of using JARVIS. By comparing the total number reported vulnerabilities in configuration *S/-* with the one in configuration *-/-*, it can easily be seen that the technical solution to improve test coverage works well with all scanners: With every scanner, the number is always higher when seeding is used. For instance, when adding up the reported vulnerabilities of all test applications, Arachni reports 254 findings in configuration *S/-* compared to only 162 in configuration *-/-*. The same behavior can also be observed with the other four scanners. In addition, the benefit of seeding is not only obvious when looking at the combined results of all test applications, but also when looking at individual test applications: The number of vulnerabilities reported when seeding is used



	Arachni				OWASP ZAP				Skipfish				Wapiti				w3af			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
⬛ WackoPicko	39	50	36	54	101	137	79	117	313	484	526	617	14	21	14	20	28	55	26	34
🟩 Peruggia	4	20	4	20	79	98	62	78	25	81	27	69	1	17	2	18	2	14	2	7
🟦 NodeGoat	22	24	44	42	83	84	79	80	235	327	262	293	3	32	23	49	9	14	19	25
🟨 Juice Shop	47	64	47	60	29	29	49	49	20	229	33	104	19	19	19	19	4	7	4	8
⬜ InsecureWebApp	11	24	7	26	59	102	58	75	66	128	130	183	9	36	8	31	19	30	15	22
🟧 Cyclone Transfers	20	23	28	32	58	85	90	103	154	359	183	886	62	102	119	158	11	25	12	31
🟦 BodgeIt	19	49	12	39	126	149	102	125	51	145	90	252	70	89	134	110	74	94	70	92

Figure 2. Total Number of Reported Vulnerabilities per Scanner and Test Application.

is nearly always higher than without seeding. For instance, Arachni reports 64 vulnerabilities in Juice Shop in configuration *S/-* compared to 47 in configuration *-/-*. Among all 35 combinations of the five scanners and seven test applications, there is an improvement in 33 cases and overall, there are just two exceptions where the number of reported vulnerabilities is not increased and remains unchanged (OWASP ZAP and Wapiti when scanning Juice Shop).

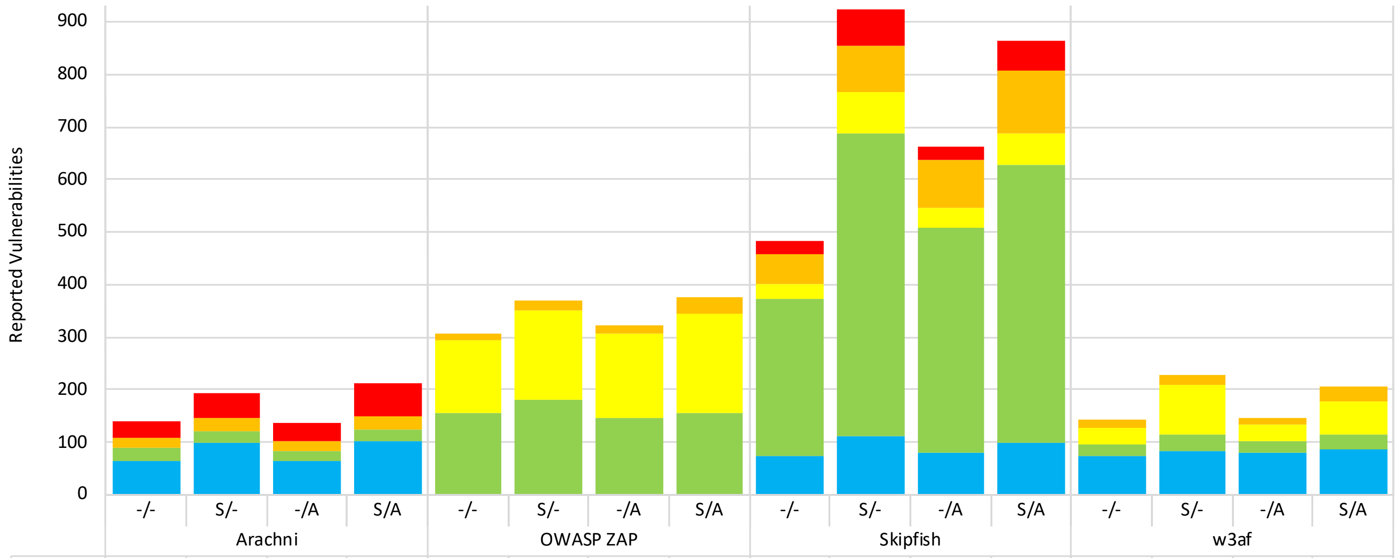
The benefit of the technical solution to improve authenticated scans is less obvious from the results in Figure 2. Using again Arachni as an example, the 178 vulnerabilities reported over all test applications in configuration *-/A* are only a small improvement compared to the 162 vulnerabilities reported in configuration *-/-*. With Wapiti, the results are much better with an improvement from 178 to 319 reported vulnerabilities. But in the case of OWASP ZAP, the numbers even get slightly lower when authenticated scans are used, from 535 to 519. When looking at individual test applications, the results vary as well. For instance, when scanning Cyclone Transfer, Wapiti reports 62 findings in configuration *-/-* and 119 findings in configuration *-/A*, which is significant improvement. But when scanning Peruggia with OWASP ZAP, 79 findings are reported in configuration *-/-*, which drops to 62 in configuration *-/A*. In general, more analysis is required to assess the impact of the technical solution to improve authenticated scans, which will follow in Sections IV-C and IV-D.

Furthermore, Figure 2 provides insights into the benefit of using both technical solutions at the same time (configuration *S/A*). Intuitively, one would expect this configuration to deliver clearly the highest number of vulnerabilities with all scanners, but this is not the case. With OWASP ZAP and w3af, the number of reported vulnerabilities over all test applications is slightly lower than in configuration *S/-*, with Arachni it is almost the same as in configuration *S/-*, and only Skipfish and Wapiti report clearly the highest number of vulnerabilities in configuration *S/A*. So just like when using only the solution to improve authenticates scans (see above), this result is currently non-conclusive and more analysis is required.

Note that to make sure that authenticated scans were carried out reliably, the involved requests and responses were analyzed after each scan. This showed that it was indeed possible to maintain authentication during all these scans, which confirms that the technical solution to improve authenticated scans is sound and works well in practice.

C. Reported Unique Vulnerabilities and Severity of Vulnerabilities

The previous evaluation in Section IV-B demonstrates that when considering just the raw number of reported vulnerabilities, JARVIS works well, in particular with respect to the technical solution to improve test coverage. However, it is not clear whether there is a true benefit in practice because it may be that the additionally found vulnerabilities are mainly duplicates of vulnerabilities that are already found in the basic configuration *-/-*, or are mainly non-critical issues. For instance, it could be that the increased number of reported vulnerabilities is mainly because the scanners report a higher number of issues related to missing HTTP response headers (e.g., missing X-Frame-Options headers), which are sometimes reported once for every requested URL (which implies many duplicate findings) and which are usually not very security-relevant.



	Arachni -/-	Arachni S/-	Arachni -/A	Arachni S/A	OWASP ZAP -/-	OWASP ZAP S/-	OWASP ZAP -/A	OWASP ZAP S/A	Skipfish -/-	Skipfish S/-	Skipfish -/A	Skipfish S/A	w3af -/-	w3af S/-	w3af -/A	w3af S/A
🟥 Critical	33	47	36	65	0	0	0	0	23	70	25	58	0	0	0	0
🟧 High	19	27	16	25	13	20	16	30	58	88	92	118	14	19	11	27
🟨 Medium	0	0	0	0	138	169	162	188	27	78	38	60	33	94	31	63
🟩 Low	25	21	20	21	155	181	146	157	299	575	427	529	21	30	24	29
🟦 Info	64	99	65	103	0	0	0	0	75	113	80	99	75	84	79	87

Figure 3. Reported Unique Vulnerabilities per Scanner over all Test Applications, according to Severity.

To analyze this in more detail, the reports of the scanners were processed with ThreadFix [21]. ThreadFix provides the functionality to normalize reports of different scanners, to eliminate duplicate findings, and to compare the results of different scanners or different runs by the same scanner. Eliminating duplicate findings means that if a specific vulnerability such as a missing HTTP response header is reported, e.g., ten times by a scanner, then it will be only included as one vulnerability in the output generated by ThreadFix. In addition, ThreadFix maps the severity levels of vulnerabilities reported by different scanners to five standard severity levels: *critical*, *high*, *medium*, *low* and *info*. The results of this processing with ThreadFix are illustrated in Figure 3. For each scanner, it shows the number of reported unique vulnerabilities (i.e., without duplicates) over all test applications when using the four different configurations. In addition, the number of vulnerabilities is separated according to the standard severity levels.

Note that Figure 3 and also the remainder of Section IV do not include the scanner Wapiti, because at the time of writing, Wapiti was not supported by ThreadFix. In addition, not every scanner uses all five standard severity levels from *critical* to *info*, as this depends on the scanner-specific severity mappings done by ThreadFix. Specifically, ThreadFix maps the severity levels of Arachni to the standard severity levels *critical*, *high*, *low* and *info* (without using *medium*), in the case of OWASP ZAP, only three levels *high*, *medium* and *low* are used (so *critical* and *info* are not used), and in the case of w3af, level *critical* is not used. The only scanner in Figure 3 that uses all five standard levels is Skipfish.

When comparing Figure 3 with Figure 2, one can see that the absolute heights of the bars, i.e., the total number of reported vulnerabilities, are lower in Figure 3. For instance, in the case of OWASP ZAP, the total number of reported vulnerabilities went down from 535 to 306 in the basic configuration *-/-* and from 684 to 370 in configuration *S/-*. This is not surprising as duplicate findings (in this case 229 and 314, respectively) were eliminated by ThreadFix. As all the bars got lower, this also shows that all scanners tend towards reporting duplicate vulnerabilities, no matter whether JARVIS is used or not. However, the more important result that can be seen from Figure 3 is that for each scanner, the relative heights of the bars when using different configurations are still very similar as in Figure 2, which means that many of the additional vulnerabilities that are reported when JARVIS is used are indeed new vulnerabilities, and not just duplicates of vulnerabilities detected in the basic configuration *-/-*. As a side note, Figure 3 also puts the high number of vulnerabilities reported by Skipfish into perspective, as a significant portion of them have severity *low* and *info* and which are therefore typically not very security-critical.

To quantify the benefit of JARVIS in more detail, Table VI contains the numbers of reported unique vulnerabilities and the relative improvements when using JARVIS.

First, the improvement that can be achieved with the technical solution to increase test coverage is analyzed. For instance, Table VI shows that when using Arachni, 141 unique vulnerabilities are reported in configuration *-/-*, which is increased to 194 vulnerabilities in configuration *S/-*. This corresponds to an improvement of reported vulnerabilities of 38%. With the other three scanners, the improvements are 21%, 92% and 59%. In the last row of Table VI, the reported vulnerabilities of the four scanners are added up. This shows that on average, the number of reported unique vulnerabilities is increased by 60% when moving from configuration *-/-* to *S/-*, which demonstrates that the technical solution to increase

TABLE VI. Reported Unique Vulnerabilities per Scanner, and Improvement by using JARVIS

Scanner	*Config.*	*Reported Unique Vulnerabilities*	*Improvement by using JARVIS*
Arachni	-/-	141	
	S/-	194	38%
	-/A	137	-3%
	S/A	214	52%
OWASP ZAP	-/-	306	
	S/-	370	21%
	-/A	324	6%
	S/A	375	23%
Skipfish	-/-	482	
	S/-	924	92%
	-/A	662	37%
	S/A	864	79%
w3af	-/-	143	
	S/-	227	59%
	-/A	145	1%
	S/A	206	42%
All four Scanners	-/-	1'072	
	S/-	1'715	**60%**
	-/A	1'268	**18%**
	S/A	1'659	**55%**

test coverage works very well.

Next, the improvement of the technical solution to improve authenticated scans is analyzed. As already seen in Section IV-B, the improvement is much smaller. For instance, Table VI shows that when using Arachni and when using configuration *-/A* instead of configuration *-/-*, the number of reported unique vulnerabilities actually goes down, from 141 to 137, which is a reduction of 3%. With the other scanners, the improvements are 6%, 37% and 1%, and adding up the reported vulnerabilities of the four scanners shows that on average, the number of reported unique vulnerabilities is improved by 18% when moving from configuration *-/-* to *-/A*. It therefore can be concluded that the solution to improve authenticated scans results in a significantly smaller improvement with respect to the absolute number of reported unique vulnerabilities than the solution to increase test coverage, which confirms the observation made in Section IV-B.

Finally, the combined effect of using both technical solutions is analyzed, i.e., configuration *S/A*. For the four scanners, this results in improvements between 23% and 79% and on average, an improvement of 55% can be achieved compared to configuration *-/-*. As these numbers are quite similar as the numbers that can be achieved in configuration *S/-*, i.e., when using only the technical solution to increase test coverage, and as the improvement that can be achieved in configuration *-/A* (see above) is relatively small, this further underlines that the effect of the technical solution to improve authenticated scans only has a relatively small effect on the absolute number of reported unique vulnerabilities.

Another important result that can be seen by looking at Figure 3 is that the increased number of vulnerabilities when using JARVIS is not just because several additional non-security-critical issues were detected (i.e., severity levels *low* and *info*). Instead, for each of the four scanners in Figure 3, the distribution of the different severity levels appears to be more or less constant, independent of the configuration that is used. To quantify this in more detail, Table VII contains the numbers of reported unique vulnerabilities and the absolute and relative number of security-critical vulnerabilities among them. For simplicity, we assume that severity levels *critical*, *high* and *medium* are considered security-critical, while levels *low* and *info* are considered non-security-critical.

TABLE VII. Reported Unique Vulnerabilities per Scanner and Configuration, and Fraction of Security-Critical Vulnerabilities

Scanner	*Config.*	*Reported Unique Vulnerabilities*	*Number of Security-critical Vulnerabilities*	*Fraction of Security-critical Vulnerabilities*
Arachni	-/-	141	52	37%
	S/-	194	74	38%
	-/A	137	52	38%
	S/A	214	90	42%
OWASP ZAP	-/-	306	151	49%
	S/-	370	189	51%
	-/A	324	178	55%
	S/A	375	218	58%
Skipfish	-/-	482	108	22%
	S/-	924	236	26%
	-/A	662	155	23%
	S/A	864	236	27%
w3af	-/-	143	47	33%
	S/-	227	113	50%
	-/A	145	42	29%
	S/A	206	90	44%
All four Scanners	-/-	1'072	358	**33%**
	S/-	1'715	612	**36%**
	-/A	1'268	427	**34%**
	S/A	1'659	634	**38%**

To explain Table VII, the numbers of scanner Arachni are discussed in detail. For instance, in configuration *-/-*, Arachni reports 141 unique vulnerabilities, 52 of them are security-critical (i.e., severity levels *critical*, *high* or *medium*). This corresponds to a fraction of 37%. In configuration *S/-*, 74 of the 194 reported vulnerabilities are security-critical, which corresponds to 38%. In the other two configurations *-/A* and *S/A*, these fractions are 38% and 42%, respectively. This shows that in the case of Arachni, the fraction of security-critical vulnerabilities is approximately the same for all four configurations. The same can be observed for the other scanners in Table VII, with the exception of w3af, where the fractions vary a bit more. The last row in the table contains the added up numbers of all four scanners, which shows a fraction of 33% security-critical vulnerabilities in configuration *-/-* and slightly higher fractions of 36%, 34% and 38% in the other three configurations, i.e., when using JARVIS. This demonstrates that on average, JARVIS not only increases the number of reported unique vulnerabilities, but that many of the additionally reported vulnerabilities are security-critical, which means that JARVIS provides a true security benefit.

To summarize this subsection, the following can be concluded:

- The technical solution to increase test coverage significantly increases the absolute number of reported unique vulnerabilities. On average, the number of reported vulnerabilities is improved by 60% when moving from configuration *-/-* to *S/-*.
- The technical solution to improve authenticated scans only has a small positive impact on the absolute number of reported unique vulnerabilities. On average, the number of reported vulnerabilities is improved by 18% when moving from configuration *-/-* to *-/A*.
- Using both technical solutions at the same time also significantly increases the absolute number of reported unique vulnerabilities. On average, the number of

reported vulnerabilities is improved by 55% when moving from configuration *-/-* to *S/A*. As this number is similar to what is achieved in configuration *S/-*, i.e., when using only the technical solution to increase test coverage, this further underlines that the effect of the technical solution to improve authenticated scans only has a relatively small effect on the absolute number of reported unique vulnerabilities.

- JARVIS slightly improves the fraction of security-critical vulnerabilities among all reported vulnerabilities. This means the practical benefit of JARVIS is even slightly better than the figures above. So, for instance, when the number of reported vulnerabilities can be improved by 60% when moving from configuration *-/-* to *S/-* (see above), then the improvement of security-critical vulnerabilities is even a bit higher than 60%.

For completeness, Figure 4 shows the number of unique vulnerabilities reported per scanner and test application when using the four different configurations, again separated according to the severity levels. Without going into the details, Figure 4 confirms that the conclusions of this subsection are also valid when considering the test applications individually: Using JARVIS results in a higher number of detected unique vulnerabilities and the distribution of the different severity levels per scanner and test application is more or less constant, independent of the configuration that is used.

D. Re-Detection of Vulnerabilities in Advanced Configurations

Intuitively, additionally seeding a scanner and/or performing authenticated scans (i.e., using configurations *S/-*, *-/A* and *S/A*) should always also report all vulnerabilities that are detected when scanning without additional seeding and without using authentication (i.e., in configuration *-/-*). However, this is not the case. To demonstrate this, Figure 5 illustrates how many of the vulnerabilities reported in the basic configuration are also found when scanning in the other three configurations. Just like in Section IV-C, this analysis is also based on the vulnerabilities after they have been processed with ThreadFix, which means that the scanner Wapiti is again not included and which implies that the heights of the bars (i.e., the total number of reported unique vulnerabilities) are exactly the same as in Figure 3.

Once more, the results of scanner Arachni are used to explain Figure 5 in details. The leftmost bar shows that in configuration *-/-*, Arachni reports 141 unique vulnerabilities. When using configuration *S/-*, then 194 findings are reported in total. Of these 194 findings, 128 are "new" findings compared to configuration *-/-* (indicated by the green part of the bar), and 66 are "old" findings compared to configuration *-/-* (indicated by the gray part of the bar), i.e., findings that were already detected in configuration *-/-*. This means that only 66 of the 141 vulnerabilities reported in configuration *-/-* are detected again in configuration *S/-* while 75 of the 141 vulnerabilities are missing, i.e., are not detected in configuration *S/-*. The same can be observed with all scanners and with all configurations: Whenever configurations *S/-*, *-/A* or *S/A* are used, a significant portion of the vulnerabilities detected in the basic configuration *-/-* are no longer detected. This means that in general, using JARVIS delivers a significant number of new findings, but also misses several of the findings that are reported when JARVIS is not used.

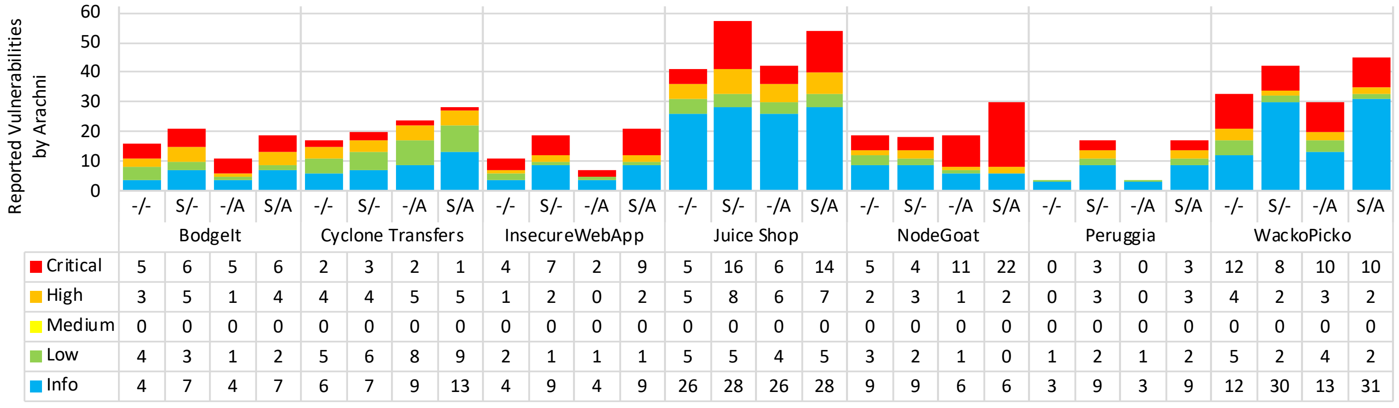
The direct consequence of this observation is that the increase of newly detected vulnerabilities is significantly higher than the increase of the absolute number of detected vulnerabilities as discussed in Section IV-C. To analyze this in detail, the relevant numbers are included in Table VIII.

TABLE VIII. REPORTED UNIQUE NEW VULNERABILITIES PER SCANNER, AND IMPROVEMENT BY USING JARVIS

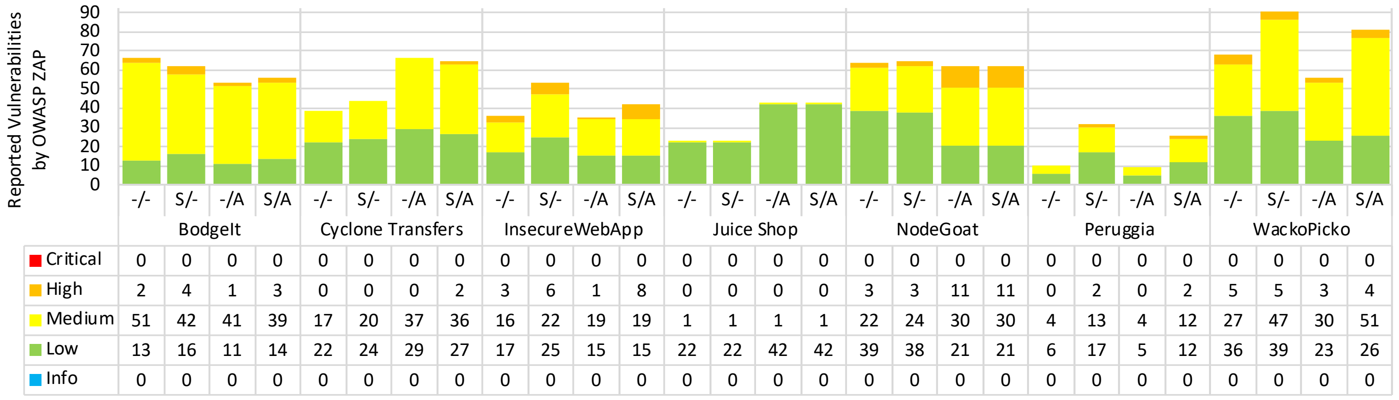
Scanner	*Config.*	*Reported Unique New Vulnerabilities*	*Improvement by using JARVIS*
Arachni	-/-	141	
	S/-	128	91%
	-/A	59	42%
	S/A	161	114%
OWASP ZAP	-/-	306	
	S/-	116	38%
	-/A	126	41%
	S/A	183	60%
Skipfish	-/-	482	
	S/-	611	127%
	-/A	419	87%
	S/A	606	126%
w3af	-/-	143	
	S/-	153	107%
	-/A	85	59%
	S/A	144	101%
All four Scanners	-/-	1'072	
	S/-	1'008	**94%**
	-/A	689	**64%**
	S/A	1'094	**102%**

The third column in Table VIII contains the number of reported new vulnerabilities per scanner and configuration and directly correspond to the "New Vulnerabilities" numbers in Figure 5. Looking at the numbers of Arachni, one can see that in configuration *S/-*, 128 new vulnerabilities are detected. Compared to the 141 vulnerabilities detected in configuration *-/-*, this corresponds to an increase of newly detected vulnerabilities of 91%. Likewise, 59 new vulnerabilities are detected in configuration *-/A*, an increase of 42% compared to configuration *-/-*. And finally, configuration *S/A* yields an increase of 114% compared to the basic configuration *-/-*. Similar results can be observed for the other three scanners. Adding up the numbers of all four scanners result in an increase of 94% in configuration *S/-*, 64% in configuration *-/A*, and 102% in configuration *S/A*.

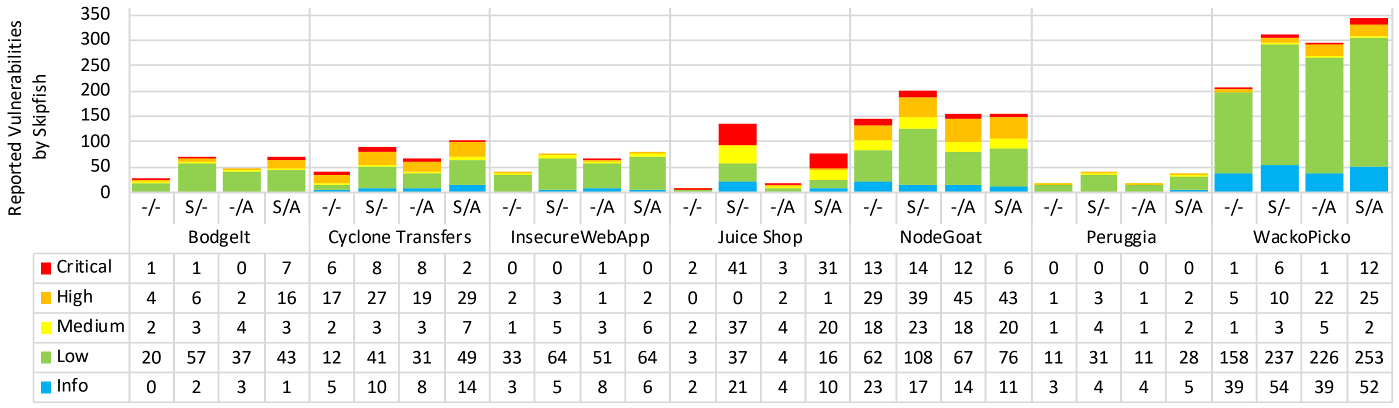
This analysis clearly shows that the effective benefit of JARVIS, i.e., the increase of new vulnerabilities that can be detected by using JARVIS, is significantly higher than the increase of the absolute number of detected vulnerabilities that was discussed in Section IV-C and listed in Table VI. For instance, in configuration *S/-*, there is an improvement of 60% on average with respect to the absolute number of vulnerabilities (see Table VI), but there is an improvement of 94% on average with respect to newly detected vulnerabilities. In addition, this analysis puts into perspective the previous conclusion that the technical solution to improve authenticated scans has only a relatively small positive impact. While the increase of the absolute number of vulnerabilities is indeed relatively small (18% on average, see Table VI), the increase of newly detected vulnerabilities is 64% on average, which is significantly higher. And in configuration *S/A*, the newly detected vulnerabilities can be increased by 102% on average, whereas the absolute increase according to Table VI is only



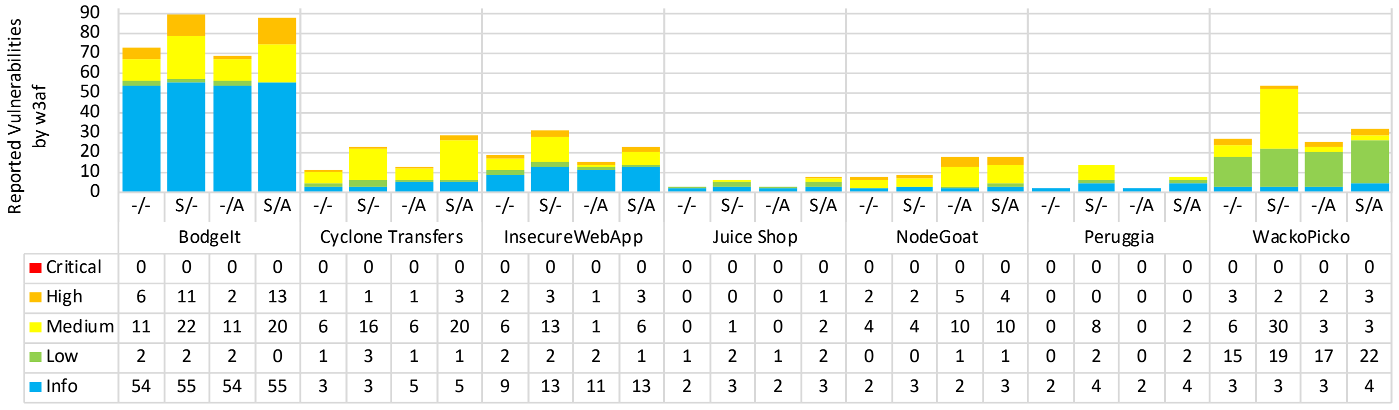
	BodgeIt				Cyclone Transfers				InsecureWebApp				Juice Shop				NodeGoat				Peruggia				WackoPicko			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟥 Critical	5	6	5	6	2	3	2	1	4	7	2	9	5	16	6	14	5	4	11	22	0	3	0	3	12	8	10	10
🟧 High	3	5	1	4	4	4	5	5	1	2	0	2	5	8	6	7	2	3	1	2	0	3	0	3	4	2	3	2
🟨 Medium	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
🟩 Low	4	3	1	2	5	6	8	9	2	1	1	1	5	5	4	5	3	2	1	0	1	2	1	2	5	2	4	2
🟦 Info	4	7	4	7	6	7	9	13	4	9	4	9	26	28	26	28	9	9	6	6	3	9	3	9	12	30	13	31



	BodgeIt				Cyclone Transfers				InsecureWebApp				Juice Shop				NodeGoat				Peruggia				WackoPicko			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟥 Critical	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
🟧 High	2	4	1	3	0	0	0	2	3	6	1	8	0	0	0	0	3	3	11	11	0	2	0	2	5	5	3	4
🟨 Medium	51	42	41	39	17	20	37	36	16	22	19	19	1	1	1	1	22	24	30	30	4	13	4	12	27	47	30	51
🟩 Low	13	16	11	14	22	24	29	27	17	25	15	15	22	22	42	42	39	38	21	21	6	17	5	12	36	39	23	26
🟦 Info	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

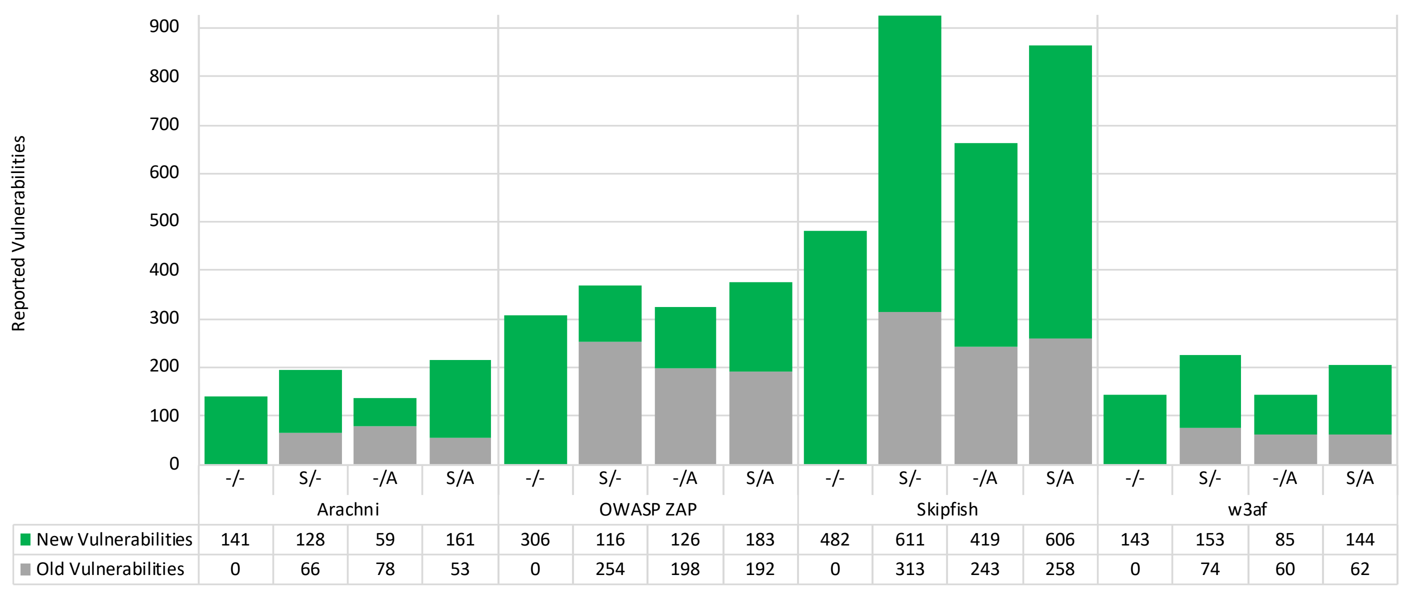


	BodgeIt				Cyclone Transfers				InsecureWebApp				Juice Shop				NodeGoat				Peruggia				WackoPicko			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟥 Critical	1	1	0	7	6	8	8	2	0	0	1	0	2	41	3	31	13	14	12	6	0	0	0	0	1	6	1	12
🟧 High	4	6	2	16	17	27	19	29	2	3	1	2	0	0	2	1	29	39	45	43	1	3	1	2	5	10	22	25
🟨 Medium	2	3	4	3	2	3	3	7	1	5	3	6	2	37	4	20	18	23	18	20	1	4	1	2	1	3	5	2
🟩 Low	20	57	37	43	12	41	31	49	33	64	51	64	3	37	4	16	62	108	67	76	11	31	11	28	158	237	226	253
🟦 Info	0	2	3	1	5	10	8	14	3	5	8	6	2	21	4	10	23	17	14	11	3	4	4	5	39	54	39	52



	BodgeIt				Cyclone Transfers				InsecureWebApp				Juice Shop				NodeGoat				Peruggia				WackoPicko			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟥 Critical	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
🟧 High	6	11	2	13	1	1	1	3	2	3	1	3	0	0	0	1	2	2	5	4	0	0	0	0	3	2	2	3
🟨 Medium	11	22	11	20	6	16	6	20	6	13	1	6	0	1	0	2	4	4	10	10	0	8	0	2	6	30	3	3
🟩 Low	2	2	2	0	1	3	1	1	2	2	2	1	1	2	1	2	0	0	1	1	0	2	0	2	15	19	17	22
🟦 Info	54	55	54	55	3	3	5	5	9	13	11	13	2	3	2	3	2	3	2	3	2	4	2	4	3	3	3	4

Figure 4. Reported Unique Vulnerabilities per Scanner and Test Application, according to Severity.



	Arachni				OWASP ZAP				Skipfish				w3af			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟩 New Vulnerabilities	141	128	59	161	306	116	126	183	482	611	419	606	143	153	85	144
▨ Old Vulnerabilities	0	66	78	53	0	254	198	192	0	313	243	258	0	74	60	62

Figure 5. Reported Unique Vulnerabilities per Scanner, according to New and Old Vulnerabilities.

55% on average. To summarize, this analysis demonstrates that not only the technical solution to increase test coverage, but also the technical solution to improve authenticated scans significantly helps to uncover vulnerabilities that would not be found otherwise and therefore, it can be concluded that both technical solutions integrated in JARVIS provide a major benefit to increase the number of detected vulnerabilities.

Determining the exact reasons why several of the vulnerabilities found in configuration *-/-* are no longer detected when using the advanced configurations would require a detailed analysis of the crawling components of the scanners, of the specific behavior of the scanners when carrying out the vulnerability tests, and of the web applications in the test set, which is beyond the scope of this work. Nevertheless, it is certainly possible to give some arguments that explain that the observed behavior is reasonable:

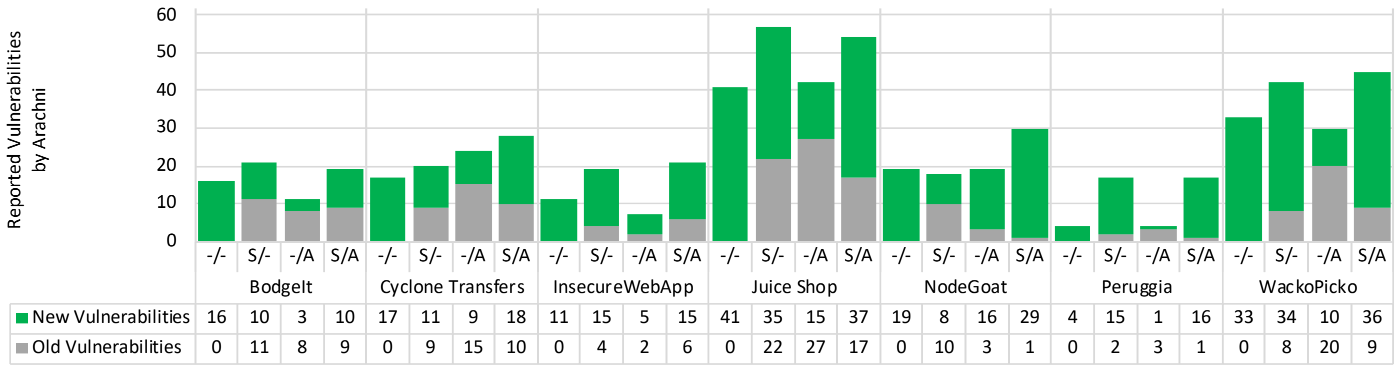
- Providing the crawler component of a scanner with additional seeds has a direct impact on the order in which the pages are requested. A different order implies different internal state changes within the web application under test [14], which typically leads to a different behavior of the web application both during crawling and during testing, and therefore to different findings.
- When doing authenticated scans, some of the resources that do not require authentication are often no longer reachable, e.g., registration, login and forgotten password pages. As deliberately insecure web applications often use such resources to place common vulnerabilities and as the evaluation of JARVIS is based on deliberately insecure applications (see Section IV-A), this most likely has a noticeable impact on the evaluation results.

An important consequence of the observation that not all vulnerabilities found in the basic configuration *-/-* are also found when using the three advanced configurations is that when testing a web application, a scanner should be used in all four configurations to maximize the total number of reported unique vulnerabilities (this will be analyzed in more detail in Section V). And obviously, although this was not analyzed in detail, an application that provides different protected areas for different roles should be scanned with users of all roles, i.e., configurations *-/A* and *S/A* should be used once per role.

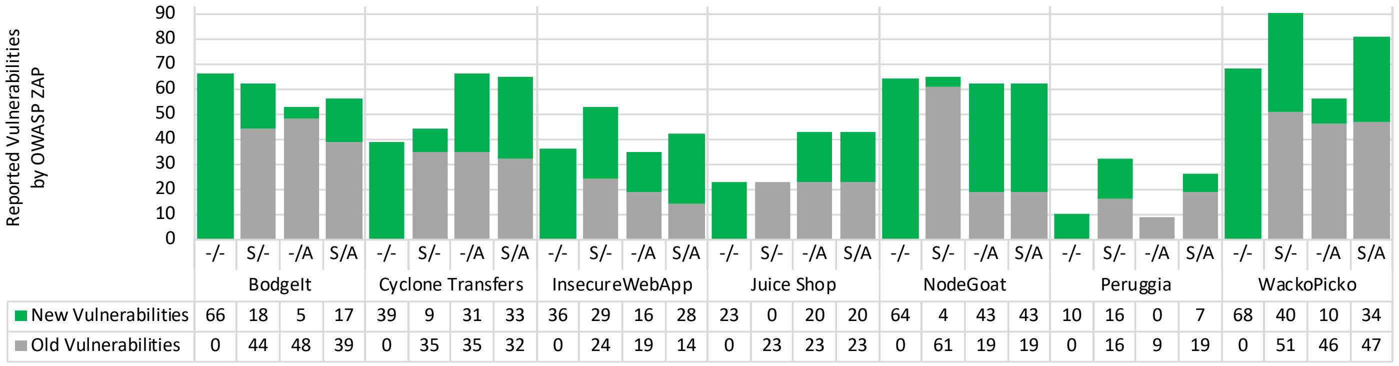
For completeness, Figure 6 shows how many of the vulnerabilities reported in the basic configuration are also found when scanning in other configurations, this time separated per scanner and per test application. Without going into the details, Figure 6 confirms that the conclusions made above are also valid when considering the test applications individually: When using the advanced configurations, several new vulnerabilities are reported and at the same time, several of the findings detected in the basic configuration are no longer reported.

V. Detailed Evaluation Focusing on SQL Injection and Cross-Site Scripting Vulnerabilities

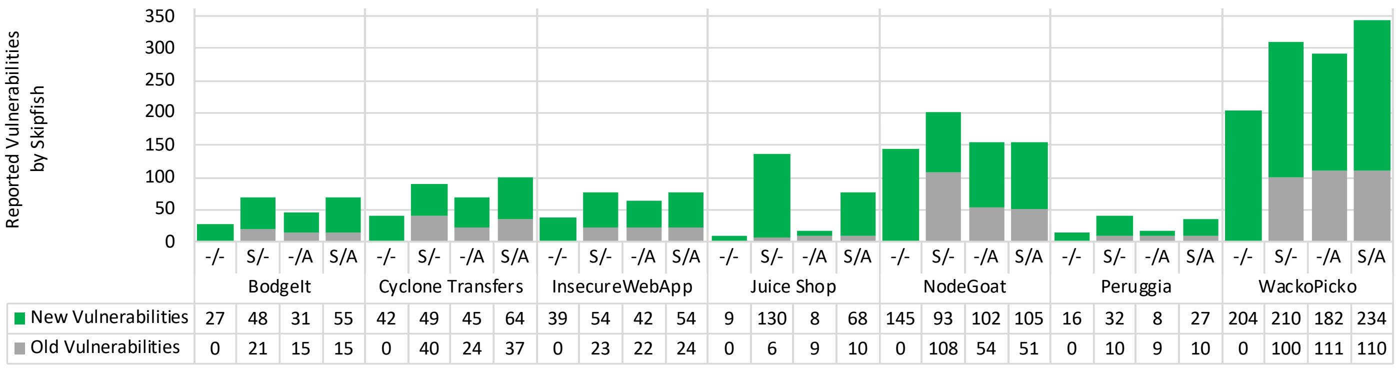
The evaluations done in Section IV demonstrate that JARVIS works very well to increase the number of detected vulnerabilities in the sense that in the advanced configurations, many additional vulnerabilities are detected and a significant fraction of them are security-critical. Two questions are still open, however. The first one is whether the additionally detected vulnerabilities are true vulnerabilities or merely false positives. In the context of a web application vulnerability scanner, a false positive is a vulnerability that is reported by the scanner, but that does not actually exist in the application under test. Conversely, a true positive is a vulnerability that is correctly identified by the scanner, i.e., one that is truly present in the tested application. The second question is whether it is indeed true that a scanner should always be used in all four configurations to maximize the total number of reported unique vulnerabilities. Based on the observations made in Section IV-D, this is most likely the case, but it should nevertheless be verified and quantified.



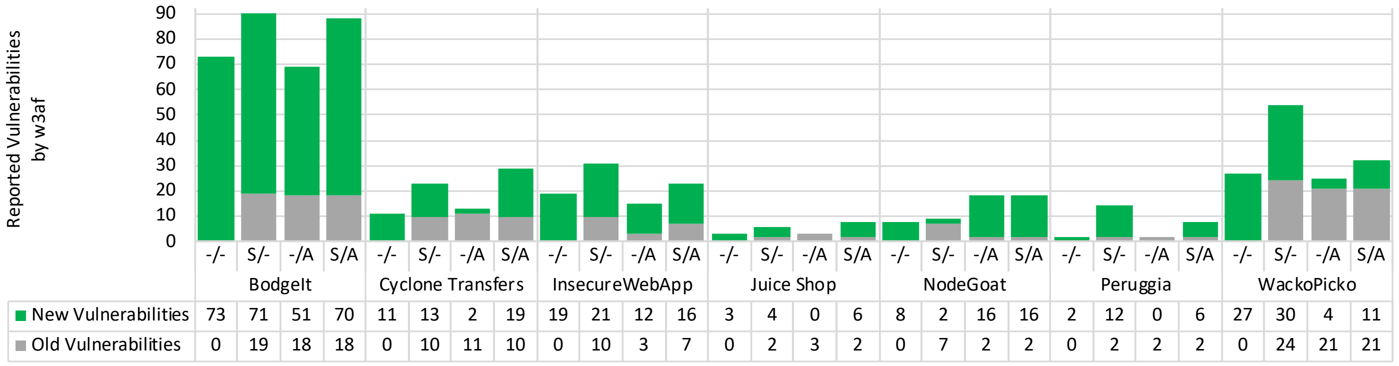
	BodgeIt				Cyclone Transfers				InsecureWebApp				Juice Shop				NodeGoat				Peruggia				WackoPicko			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟩 New Vulnerabilities	16	10	3	10	17	11	9	18	11	15	5	15	41	35	15	37	19	8	16	29	4	15	1	16	33	34	10	36
⬜ Old Vulnerabilities	0	11	8	9	0	9	15	10	0	4	2	6	0	22	27	17	0	10	3	1	0	2	3	1	0	8	20	9



	BodgeIt				Cyclone Transfers				InsecureWebApp				Juice Shop				NodeGoat				Peruggia				WackoPicko			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟩 New Vulnerabilities	66	18	5	17	39	9	31	33	36	29	16	28	23	0	20	20	64	4	43	43	10	16	0	7	68	40	10	34
⬜ Old Vulnerabilities	0	44	48	39	0	35	35	32	0	24	19	14	0	23	23	23	0	61	19	19	0	16	9	19	0	51	46	47

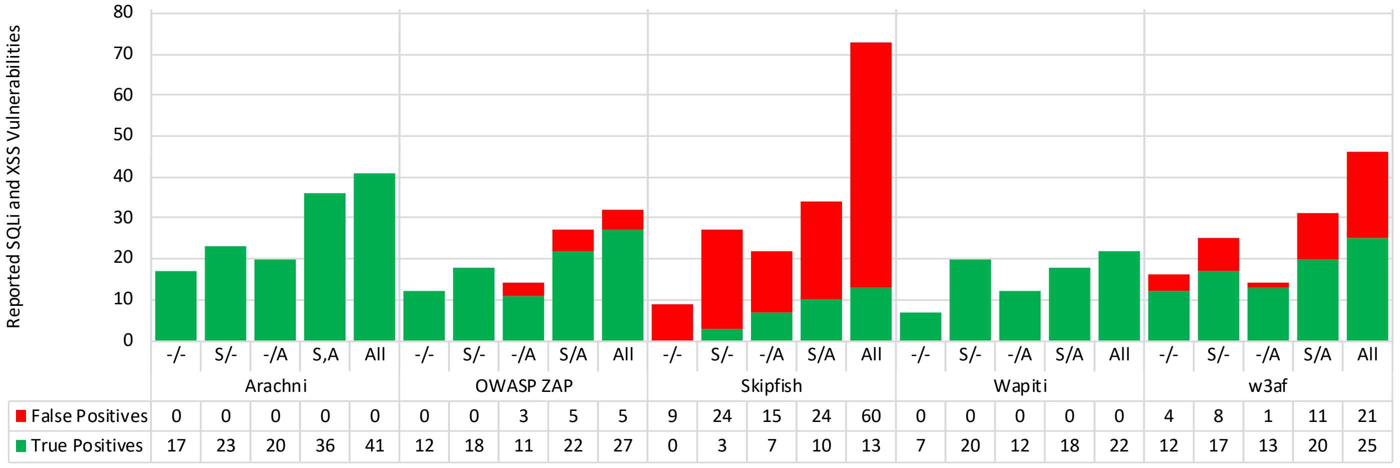


	BodgeIt				Cyclone Transfers				InsecureWebApp				Juice Shop				NodeGoat				Peruggia				WackoPicko			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟩 New Vulnerabilities	27	48	31	55	42	49	45	64	39	54	42	54	9	130	8	68	145	93	102	105	16	32	8	27	204	210	182	234
⬜ Old Vulnerabilities	0	21	15	15	0	40	24	37	0	23	22	24	0	6	9	10	0	108	54	51	0	10	9	10	0	100	111	110



	BodgeIt				Cyclone Transfers				InsecureWebApp				Juice Shop				NodeGoat				Peruggia				WackoPicko			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟩 New Vulnerabilities	73	71	51	70	11	13	2	19	19	21	12	16	3	4	0	6	8	2	16	16	2	12	0	6	27	30	4	11
⬜ Old Vulnerabilities	0	19	18	18	0	10	11	10	0	10	3	7	0	2	3	2	0	7	2	2	0	2	2	2	0	24	21	21

Figure 6. Reported Unique Vulnerabilities per Scanner and Test Application, according to New and Old Vulnerabilities.



	Arachni					OWASP ZAP					Skipfish					Wapiti					w3af				
	-/-	S/-	-/A	S,A	All	-/-	S/-	-/A	S/A	All	-/-	S/-	-/A	S/A	All	-/-	S/-	-/A	S/A	All	-/-	S/-	-/A	S/A	All
🟥 False Positives	0	0	0	0	0	0	0	3	5	5	9	24	15	24	60	0	0	0	0	0	4	8	1	11	21
🟩 True Positives	17	23	20	36	41	12	18	11	22	27	0	3	7	10	13	7	20	12	18	22	12	17	13	20	25

Figure 7. Reported Unique SQLi and XSS Vulnerabilities per Scanner, over all Test Applications, according to True and False Positives.

To answer these final two questions, a more detailed analysis focusing on SQL injection (SQLi) and cross-site scripting (XSS) vulnerabilities was done. To do this, all vulnerabilities of these types were first extracted from the original reports of the scanners. Then, the vulnerabilities were manually verified to identify them as either true or false positives. This required a lot of effort, which is the main reason why the focus was set on these two types. Nevertheless, this serves well to answer the two open questions and also to evaluate the true potential of JARVIS in general as both vulnerabilities are highly relevant in practice and highly security-critical. In addition, the test applications contain several of them, which means SQLi and XSS vulnerabilities represent a meaningful sample size. The results are illustrated in Figure 7, for each scanner and over all test applications. The green parts of the bars correspond to true positives (true vulnerabilities) and the red parts correspond to false positives (incorrectly reported vulnerabilities). Duplicates were manually removed, so the bars represent the number of unique vulnerabilities that were reported. In addition, Figure 7 not only shows the number of reported unique vulnerabilities per configuration, but also the total number of reported unique vulnerabilities when the findings of all four configurations are combined (this is identified as configuration *All*).

The first observation when analyzing Figure 7 is that the conclusions made in Section IV-C are still valid in the sense that for each scanner, the number of reported unique SQLi and XSS vulnerabilities is significantly increased when using the advanced configurations compared to the basic configuration *-/-* . This is not very surprising based on the analyses that were done so far, but it demonstrates that JARVIS not only improves the vulnerability detection performance when considering all reported vulnerabilities, but also when focusing on specific and highly relevant SQLi and XSS vulnerabilities.

In addition, Figure 7 delivers the answer to the first of the final two questions. Looking at the bars in the figure, it can be seen that using JARVIS does not have a significant impact on the number of false positives that are reported. For instance, Arachni, OWASP ZAP and Wapiti all produce no false positives when used in the basic configuration *-/-*. When using the advanced configurations, Arachni and Wapiti still do not report any false positives, while OWASP ZAP produces a relatively small fraction of false positives in configurations *-/A* and *S/A*. On the other hand, scanners that report false positives in the basic configurations (w3af and especially Skipfish, which does not report a single true positive in the basic configuration) also do so in the advanced configurations, but overall, the fraction of false positives reported in the advanced configurations remains in a similar order as in the basic configuration and is not significantly increased. As an example, the fractions of false positives reported by w3af are 25% in configuration *-/-*, 32% in configuration *S/-*, 7% in configuration *-/A*, and 35% in configuration *S/A*, so the fraction of false positives reported in any of the advanced configurations is not significantly higher than the 25% reported in configuration *-/-*. The same is true in the case of Skipfish, with the difference that the fraction of reported false positives is very high in general. Overall, the conclusion therefore is that JARVIS does not have a negative impact on the fraction of reported false positives. This is a very important finding because if using JARVIS resulted in a significantly increased fraction of reported false positives, then the value in practice would be very limited, even if the absolute number of true positives were also increased.

Furthermore, Figure 7 also answers the second open question and confirms what was already stated in Section IV-D: It is important to perform scans in all four configurations and to combine the detected vulnerabilities to maximize the number of reported unique vulnerabilities. This can easily be seen by comparing the heights of the bars: For each scanner, the height of the bar labeled with *All* is always greater than any of the other four bars, which means that the sum of the vulnerabilities detected in the four configuration (i.e., configuration *All*) is always higher than the number of vulnerabilities detected in any of the individual configurations (i.e., configurations *-/-*, *S/-*, *-/A* and *S/A*). For instance, in the case of OWASP ZAP, the four individual configurations report 12, 18, 14 and 27 unique vulnerabilities, and combining all these vulnerabilities results in 32 unique vulnerabilities, which is more than what was detected in any of the individual configurations. This is not only true when considering all vulnerabilities, i.e., true and false positives combined, but also when just considering the true positives. To analyze this in more detail, Table IX is used, which is based on the numbers in Figure 7, but which only considers the true positive vulnerabilities.

TABLE IX. Reported Unique True Positive SQLi and XSS Vulnerabilities per Scanner, and Improvement by using JARVIS

Scanner	*Config.*	*Reported SQLi and XSS Vulnerabilities*	*Improvement by using JARVIS*
Arachni	-/-	17	
	S/-	23	35%
	-/A	20	18%
	S/A	36	112%
	All	41	141%
OWASP ZAP	-/-	12	
	S/-	18	50%
	-/A	11	-8%
	S/A	22	83%
	All	27	125%
Skipfish	-/-	0	
	S/-	3	–%
	-/A	7	–%
	S/A	10	–%
	All	13	–%
Wapiti	-/-	7	
	S/-	20	186%
	-/A	12	71%
	S/A	18	157%
	All	22	214%
w3af	-/-	12	
	S/-	17	42%
	-/A	13	8%
	S/A	20	67%
	All	25	108%
All five Scanners	-/-	48	
	S/-	81	**69%**
	-/A	63	**31%**
	S/A	106	**121%**
	All	128	**167%**

From Table IX, it can be seen that for each of the five scanners, combining the results of all configurations delivers more true positives than are reported in any individual configuration. With Arachni, for instance, the best individual configuration (*S/A*) reports 36 findings, but when combining all four configurations, 41 findings are detected. The same observation can be made for the other scanners, which demonstrates that combining the vulnerabilities reported in all four configurations always results in the highest number of unique true positive vulnerabilities.

Compared to the basic configuration *-/-*, using configuration *All* more than doubles the number of reported unique true positive SQLi and XSS with every scanner. The smallest improvement is achieved with w3af, where the number of vulnerabilities is increased from 12 to 25 (a plus of 108%), followed by OWASP ZAP (125%), then Arachni (141%), then Wapiti (214%), and in the case of Skipfish, where not a single vulnerability (true positive) could be detected in the basic configuration, using JARVIS manages to detect 13 vulnerabilities (no %-benefit is included in Table IX with Skipfish as configuration *-/-* reports 0 true positives). Combining the numbers of all five scanners (see final row of Table IX) shows that on average and by combining the vulnerabilities reported in any of the four configurations, JARVIS manages to increase the number of reported true positive SQLi and XSS vulnerabilities by 167% compared to using the scanners without JARVIS.

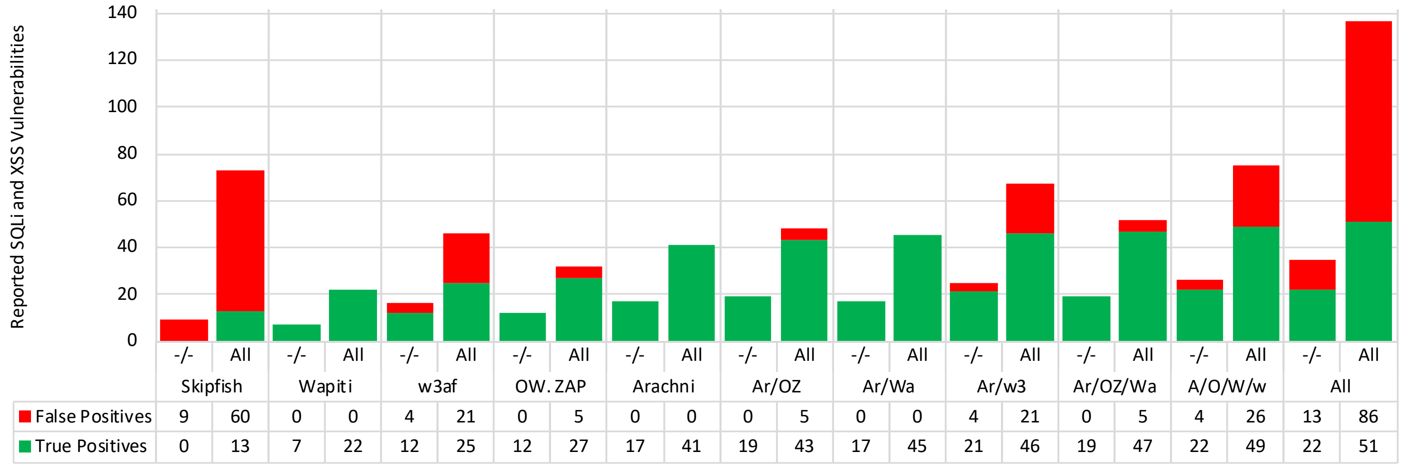
Finally, Figure 7 and Table IX also make it possible to compare the scanners. In particular, based on the test applications used in the evaluation and focusing on SQLi and XSS vulnerabilities, it shows that Arachni performs best as it finds the highest number of vulnerabilities without producing a single false positive, followed by OWASP ZAP and Wapiti. OWASP ZAP finds more true vulnerabilities than Wapiti, but also reports a few false positives. Next, there is w3af, which already reports a considerable fraction of false positives and finally, there is Skipfish, which performs quite poorly, not only with respect to true positives but especially also with respect to false positives. This once more puts into perspective the results of the first evaluation (see Figure 2), where Skipfish reported many more vulnerabilities than the other scanners.

VI. Evaluation of Combining Multiple Scanners

As the configuration effort to use JARVIS is small and the configurations are scanner-independent (see Section III-C), JARVIS makes it possible to use multiple scanners in parallel in an efficient way. Therefore, in a final evaluation, the benefits and limitations of using multiple scanners in parallel are analyzed. To do this, the same vulnerabilities as in the previous subsection are used, i.e., only SQLi and XSS vulnerabilities are considered, which makes it possible to precisely analyze the impact of using multiple scanners on the reported true and false positives. Figure 8 shows the reported unique true and false positive vulnerabilities when using individual scanners and different combinations of multiple scanners and when using the scanners in the basic configuration *-/-* and when combining the results of all four configurations (i.e., configuration *All*). The results are ranked from left to right in ascending order according to the number of true positives that are identified in configuration *All*.

Looking at the results in configuration *All*, the rightmost bar combines the results of all five scanners, which obviously delivers most true positives (51), but which also delivers most false positives (86). The results also show that in this test setting, Arachni performs very well on its own, as it finds 41 true positives (without a single false positive), which means that the other four scanners combined can only detect 10 true positives that are not found by Arachni. Looking at combinations of scanners, then the combination of Arachni & Wapiti (Ar/Wa) performs well and manages to identify 45 of the 51 true positives without any false positives. Combining Arachni, OWASP ZAP & Wapiti (Ar/OZ/Wa) is also a good choice as it finds 47 true positives with only a few false positives. This demonstrates that combining multiple scanners is indeed beneficial to increase the number of detected true positives without a significant negative impact on the number of reported false positives. However, blindly combining as many scanners as possible (e.g., all five scanners used here) is not a good idea in general because although this results in most true positives, it also maximizes the number of reported false positives. Finally, comparing the results in configuration *All* with the ones in configuration *-/-* demonstrates that even when combining multiple scanners, configuration *All* increases the number of detected true positives by more than 100% in every single case, which again underlines the benefits of JARVIS.

Note that since seven test web applications that cover several technologies were used in this evaluation, the results are at least an indication that the suitable combinations of scanners identified above (Arachni & Wapiti and Arachni, OWASP ZAP & Wapiti) should perform well in many scenarios. However, this is certainly no proof and it may be that other combinations of scanners are better suited depending on the web application under test. This means that in practice, one has to experiment



	-/-	All	-/-	All	-/-	All	-/-	All	-/-	All	-/-	All	-/-	All	-/-	All	-/-	All	-/-	All	-/-	All
	Skipfish		Wapiti		w3af		OW. ZAP		Arachni		Ar/OZ		Ar/Wa		Ar/w3		Ar/OZ/Wa		A/O/W/w		All	
🟥 False Positives	9	60	0	0	4	21	0	5	0	0	0	5	0	0	4	21	0	5	4	26	13	86
🟩 True Positives	0	13	7	22	12	25	12	27	17	41	19	43	17	45	21	46	19	47	22	49	22	51

Figure 8. Reported Unique SQLi and XSS Vulnerabilities using different Scanner Combinations, over all Test Applications.

with different scanner combinations to determine the one that is best suited in a specific scenario.

VII. Conclusion

In this paper, we presented JARVIS, which provides technical solutions to overcome some of the limitations – notably crawling coverage and reliability of authenticated scans – of web application vulnerability scanners. As JARVIS is independent of specific scanners and implemented as a proxy, it can be applied to a wide range of existing vulnerability scanners. The evaluation based on five freely available scanners and seven test web applications covering various technologies demonstrates that JARVIS works well in practice. In particular, JARVIS manages to significantly improve the number of reported vulnerabilities without increasing the fraction of false positives, and many of the additionally found vulnerabilities are security-critical. The most relevant evaluation results are summarized in the following list:

- The technical solution to increase test coverage has a major positive impact on the number of detected vulnerabilities. Compared to using the scanners without JARVIS (i.e., in the basic configuration *-/-*), the absolute number of reported unique vulnerabilities can be increased by 60% on average in configuration *S/-*. When only considering newly detected vulnerabilities, i.e., vulnerabilities that are not detected in the basic configuration *-/-*, the increase is 94% on average.
- The technical solution to improve authenticated scans has a relatively small impact on the absolute number of reported unique vulnerabilities. On average, the absolute number of reported vulnerabilities is increased by 18% when moving from configuration *-/-* to *-/A*. However, when considering the newly detected vulnerabilities, the improvement is 64% on average, which means the technical solution to improve authenticated scans also has a significant positive impact on the number of detected vulnerabilities.
- Using both technical solutions, i.e., when using configuration *S/A* instead of configuration *-/-*, the absolute number of reported vulnerabilities is increased by 55% and the number of newly detected vulnerabilities is increased by 102% on average. This means that on average, using JARVIS with both technical solutions more than doubles the newly detected vulnerabilities compared to scanning without using JARVIS.
- JARVIS slightly improves the fraction of security-critical vulnerabilities among all reported vulnerabilities. This underlines the practical benefit of JARVIS as it does not just report many additional irrelevant findings, but truly increases the number of security-critical issues that can be found
- A significant portion of the vulnerabilities that are detected when a scanner is used without JARVIS (i.e., in the basic configuration *-/-*) are not detected again when the scanner is used with JARVIS (i.e., in the advanced configurations *S/-*, *-/A* and *S/A*). A direct consequence of this observation is that the scanners should always be used in all four configurations, i.e., in configuration *-/-* without using JARVIS and in configurations *S/-*, *-/A* and *S/A* with using JARVIS to maximize the total number of detected vulnerabilities.
- A detailed analysis using SQLi and XSS vulnerabilities showed that JARVIS does not have a negative impact on the fraction of false positives that are reported. Scanners that report no false positives in configuration *-/-* deliver no or only very few vulnerabilities when using JARVIS. And scanners that report some false positives in the basic configuration also do so in the advanced configurations, but overall, the fraction of false positives remains more or less constant, independent of the configuration. This result is highly relevant for the applicability of JARVIS in practice, as otherwise, the practical benefit would be very limited.
- The same analysis demonstrated that it is indeed important to perform scans in all four configurations and to combine the detected vulnerabilities, as the sum of the vulnerabilities that are detected in the four different configurations is always greater than the number of vulnerabilities detected in any of the individual configurations. Also, this analysis showed that by using JARVIS, the effectiveness of each of the

five scanners used in the evaluation could be more than doubled and on average, the number of detected true positive SQLi and XSS vulnerabilities could be increased by 167%. This underlines that JARVIS is both an effective and truly scanner-independent solution to increase the number of detected security-critical vulnerabilities.

The configuration effort to use JARVIS is small and the configurations are scanner-independent. Therefore, JARVIS also provides an important basis to use multiple scanners in parallel in an efficient way. The provided analysis shows that combining multiple scanners is indeed beneficial as it increases the number of true positives, which is not surprising as different scanners detect different vulnerabilities. However, it was also demonstrated that blindly combining as many scanners as possible is not a good idea in general because although this results in most true positives, it also delivers the sum of all false positives reported by the scanners. In the evaluation, the combination of Arachni & Wapiti or Arachni, OWASP ZAP & Wapiti yielded the best compromise between a high rate of true positives and a low rate of false positives. As a representative set of web application technologies was used in the evaluation, it can be expected that these combinations work well in many scenarios, but this is no proof and in practice, one has to experiment with different combinations to determine the one that is best suited in a specific scenario.

Acknowledgment

This work was partly funded by the Swiss Confederation's innovation promotion agency CTI (project 18876.1 PFES-ES).

References

[1] D. Esposito, M. Rennhard, L. Ruf, and A. Wagner, "Exploiting the Potential of Web Application Vulnerability Scanning," in Proceedings of the 13th International Conference on Internet Monitoring and Protection (ICIMP). Barcelona, Spain: IARIA, 2018, pp. 22–29.

[2] WhiteHat Security, "2018 Application Security Statistics Report," Tech. Rep., 2018, URL: https://www.whitehatsec.com/blog/2018-whitehat-app-sec-statistics-report/ [accessed: 2019-05-03].

[3] A. Doupé, M. Cova, and G. Vigna, "Why Johnny can't Pentest: An Analysis of Black-Box Web Vulnerability Scanners," in Proceedings of the 7th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, ser. DIMVA'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 111–131.

[4] S. Chen, "SECTOOL Market," 2016, URL: http://www.sectoolmarket.com/price-and-feature-comparison-of-web-application-scanners-unified-list.html [accessed: 2019-05-03].

[5] L. Suto, "Analyzing the Accuracy and Time Costs of Web Application Security Scanners," Tech. Rep., 2010, URL: http://www.think-secure.nl/pdf/Accuracy_and_Time_Costs_of_Web_App_Scanners.pdf [accessed: 2019-05-03].

[6] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, "State of the Art: Automated Black-Box Web Application Vulnerability Testing," in Proceedings of the 2010 IEEE Symposium on Security and Privacy, 2010, pp. 332–345.

[7] E. A. A. Vega, A. L. S. Orozco, and L. J. G. Villalba, "Benchmarking of Pentesting Tools," International Journal of Computer, Electrical, Automation, Control and Information Engineering, vol. 11, no. 5, 2017, pp. 602–605.

[8] M. Qasaimeh, A. Shamlawi, and T. Khairallah, "Black Box Evaluation of Web Application Scanners: Standards Mapping Approach," Journal of Theoretical and Applied Information Technology, vol. 96, no. 14, 2018, pp. 4584–4596.

[9] Y. Makino and V. Klyuev, "Evaluation of Web Vulnerability Scanners," in Proceedings of the IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), vol. 1, Warsaw, Poland, 2015, pp. 399–402.

[10] N. I. Daud, K. A. A. Bakar, and M. S. M. Hasan, "A Case Study on Web Application Vulnerability Scanning Tools," in 2014 Science and Information Conference, London, UK, 2014, pp. 595–600.

[11] S. Chen, "Security Tools Benchmarking: WAVSEP 2017/2018 - Evaluating DAST against PT/SDL Challenges," 2017, URL: http://sectooladdict.blogspot.com/2017/11/wavsep-2017-evaluating-dast-against.html [accessed: 2019-05-03].

[12] S. El Idrissi, N. Berbiche, F. Guerouate, and S. Mohamed, "Performance Evaluation of Web Application Security Scanners for Prevention and Protection against Vulnerabilities," International Journal of Applied Engineering Research, vol. 12, no. 21, 2017, pp. 11 068–11 076.

[13] SNORT, "Network Intrusion and Prevention System," URL: https://www.snort.org [accessed: 2019-05-03].

[14] A. Doupé, L. Cavedon, C. Kruegel, and G. Vigna, "Enemy of the State: A State-aware Black-Box Web Vulnerability Scanner," in Proceedings of the 21st USENIX Security Symposium (USENIX Security 12). Bellevue, WA: USENIX, 2012, pp. 523–538.

[15] A. v. Deursen, A. Mesbah, and A. Nederlof, "Crawl-based Analysis of Web Applications: Prospects and Challenges," Science of Computer Programming, vol. 97, 2015, pp. 173–180.

[16] G. Pellegrino, C. Tschürtz, E. Bodden, and C. Rossow, "jäk: Using Dynamic Analysis to Crawl and Test Modern Web Applications," in Research in Attacks, Intrusions, and Defenses, H. Bos, F. Monrose, and G. Blanc, Eds. Cham: Springer International Publishing, 2015, pp. 295–316.

[17] D. Zulla, "Improving Web Vulnerability Scanning," DEF CON, 2012, URL: https://www.defcon.org/images/defcon-20/dc-20-presentations/Zulla/DEFCON-20-Zulla-Improving-Web-Vulnerability-Scanning.pdf [accessed: 2019-05-03].

[18] PortSwigger, "Burp Suite," URL: https://portswigger.net/burp [accessed: 2019-05-03].

[19] ThreadFix, "ThreadFix Endpoint CLI," URL: https://github.com/denimgroup/threadfix/tree/master/archived/threadfix-cli-endpoints [accessed: 2019-05-03].

[20] B. Urgun, "WIVET: Web Input Vector Extractor Teaser," URL: https://github.com/bedirhan/wivet [accessed: 2019-05-03].

[21] ThreadFix, "Application Vulnerability Correlation with ThreadFix," URL: https://threadfix.it [accessed: 2019-05-03].