Black-box Fuzzing Approaches to Secure Web Applications: Survey

Aseel Alsaedi[1], Abeer Alhuzali[2], Omaimah Bamasag[3]
Computer Science Department, Faculty of Computing and Information Technology
King Abdulaziz University, Jeddah, Saudi Arabia

***Abstract*—Web applications are increasingly important tools in our modern daily lives, such as in education, business transactions, and social media. Because of their prevalence, they are becoming more susceptible to different types of attacks that exploit security vulnerabilities. Exploiting these vulnerabilities may cause damage to the web applications as well as the end-users. Thus, web apps' developers should identify vulnerabilities and fix them before an attacker exploits them. Using black-box fuzzing techniques for vulnerability identification is very popular during the web apps' development life cycle. These techniques pledge to find vulnerabilities in web applications by constructing attacks without accessing their source codes. This survey explores the research that has been done in the black-box vulnerability finding and exploits construction in web applications and proposes future directions.**

***Keywords*—Black-box fuzzing; web application security; vulnerability scanning; automatic web app testing; vulnerability detection; survey**

I. Introduction

Web applications are significant components in various fields: commercial, banking, entertainment, education, healthcare, and social networking. To reach international markets, organizations have utilized web applications to promote their products/goals and provide end-user services. In recent years, web applications have evolved rapidly. This rapid development has led to the use of different technologies and libraries, which results in more complex and feature-full applications. Many application developers do not have the necessary security skills that prevent them from writing buggy code. That, in turn, allows attackers to exploit those vulnerabilities and may cause damage. Thus, the security of web applications is of paramount concern.

Verizon's 2019 report [1] stated that most data breaches occurred due to attacks on web applications. Two out of three of all the examined data breaches were in web applications. Another report on web application vulnerability [2] analyzed nearly 5,000 different web apps from March 2019 until February 2020 and found that 26% of these applications have critical vulnerabilities. The remaining apps had medium-level vulnerabilities. All these studies indicate that vulnerabilities are prevalent in web applications.

Typically, security staffs use manual analysis to identify vulnerabilities to fix. This process is time-consuming and error-prone, as it depends on the expertise level of the analyst, and it is challenging because of the increasing complexities of web applications. Additionally, manual analysis is difficult to perform in each new update on applications that can lead an attacker to exploit vulnerabilities.

Many automated techniques for vulnerability analysis and exploit constructions have been proposed. These approaches aim to reduce the cost, time, effort, and precision during testing. Due to these benefits, these types of approaches have become a hot topic in recent years, especially when applications tend to be complex. Broadly, these approaches can be categorized into white-box, and black-box fuzzing approaches. The white-box testing is based on examining the source code and the behavior of a web application to find security vulnerabilities. Several studies utilized this technique to identify critical vulnerabilities in web applications such as [3], [4], [5], [6], [7], [8], and [9]. Unfortunately, this type of approach cannot apply to a wide range of web apps; many times, it is specialized for a particular programming language. Additionally, the source code of the web app is not available at each time. On the other hand, the black-box fuzzing is a vulnerability-analysis and exploits construction approach that automatically discovers vulnerabilities and generates exploits without accessing the source code. Compared with white-box, the major benefit of using black-box fuzzing is that it is fast and efficient, and it can find security bugs in any web application, regardless of its implementation details. Thus, this technique applies to a wide range of web applications. Additionally, the black-box fuzzing approach is helpful for developers who have little or no experience in writing secure source code.

As a result of the pressing need to protect web applications without accessing the source code, a significant research effort has been geared towards developing many techniques for detecting web applications using the black-box fuzzing approach. Much of this research addresses a specific class of vulnerabilities or delivers fewer false positives, such as [10] and [11].

As such, the primary goal for this survey is to analyze the last ten years of existing bug-finding techniques in web applications , focusing on the black-box fuzzing approach. Further, this survey contributes towards identifying the challenges of the black-box fuzzing approaches, which aids the research community in determining where further research can be performed and provides valuable insights for improving the crawling module. Because of our goal, this survey intends to answer the following questions:

1) What are the techniques utilized by the approach?
2) Is the approach applicable to be used in modern web applications?
3) How does the approach construct benign inputs needed by web applications to explore further and test the application?

This paper is organized as follows. Section 2 provides an overview of the web application and its common vulnerabilities. Section 2 describes a black-box fuzzing approach, and Section 4 analyzes and compares existing approaches. Section 5 discusses the results by identifying current techniques' main weaknesses and then identifying potential research areas. Finally, Section 6 concludes our review.

II. Web Applications' Architecture and Common Vulnerabilities

A. Web Application Architecture and Characteristics

The traditional web applications have a three-tier architecture: client-side, server-side, and back-end databases that provide and store the web application data. The client-side part is executed on the user's web browser, allowing the user to interact and communicate with the web application via user inputs, links, etc. The client-side code is written using different technologies such as HTML, JavaScript, and CSS. On the other hand, the server-side is executed on the web server, responding to the client messages and managing the business logic. Server-side code is commonly written in PHP, Java, and so on. The client-side and server-side communicate with each other through messages, which are HTTP requests and responses. To demonstrate how a web application works, Fig. 1 shows a high-level view of how a typical web application works when communicating with the server-side to register a new user. It can be summarized as follows:

1) The user opens the web page, which its browser can render, and fills out all the necessary inputs to complete the registration. Filling out inputs on web forms provides interactions between users and browsers that enable users to deal with different input types, such as numeric and text. Once the user has completed the fill-in and submitted the form (usually by clicking the submit button), the inputs are encapsulated into an HTTP request and sent to the server.
2) The server-side receives the client request and then processes it.
3) The server-side sends the web form data to the database as a query to add the user to the web application list.
4) The server-side replies to the user as an HTTP response to the present result.

Today, web applications are getting more complex and dynamic because of the adaptation of different technologies such as AJAX (**A**synchronous **J**avaScript **A**nd **X**ML). The web form in our example, as shown in Fig. 2 can dynamically update part of the page to prevent unwanted requests when the data involves errors. It can also change the page's display contents dynamically without waiting for the server-side to deliver a new HTML page.

B. Web Applications' Vulnerabilities

Web applications play critical roles in our lives and are used in various activities and services. Typically, web applications deal with private or sensitive data, which becomes a valuable target for attackers. As shown in Fig. 2, a web application accepts potentiality dangerous inputs since the entered user inputs may include malicious code that harms the application. There are many types of vulnerabilities; in the following, we will focus solely on the most common vulnerabilities.

1) Cross-Site Scripting: XSS is a vulnerability executed on the client-side of the web application. The XSS is a code injection that enables the attacker to execute a malicious script (e.g., JavaScript code) in the victim's browser. The exploitation of XSS vulnerability is very dangerous, according to OWASP [12], because the XSS enables the attacker to modify web pages and steal sensitive information such as cookies, session tokens, and users' credentials. There are two classes of XSS attacks: reflected and stored.

- **Reflected XSS** occurs when the attacker successfully injects a malicious script into an HTTP request. The victim browser receives an HTTP response, including a malicious script, and executes it.
- **Stored XSS** occurs when an attacker injects a malicious script in content such as post comments and store permanently, often on the database. Suppose that malicious content is retrieved from the database without filtering. In that case, the malicious code will be executed on the victim's browser at all times when any user visits the infected page.

2) SQL Injection: SQLI is ranked as the most common injection vulnerability on web applications according to OWASP [12]. It occurs when an attacker manipulates the original logic, semantic, or syntax of an SQL query by using specially designed inputs such as SQL keywords or operators into original queries, which aims to control the back-end databases of the web application. There are two types of SQLI attacks: first-order SQLI and second-order SQLI.

- **First-order SQLI** The malicious queries are loaded and executed directly on the database.
- **Second-order SQLI** The crafted input is inserted into the database without sensitization. After that, that malicious content is retrieved without sanitization, which will allow the execution of malicious content.

III. Black-Box Fuzzing Approach as Defensive Technique for Protecting Web Applications

A. Typical Scenario

The black-box scanner consists of three primary modules: the crawling module, attack module, and analysis module. Crawling is a fundamental component in web application scanners which explores the applications that determine the scanner's capability to identify vulnerabilities. If a vulnerability scanner can discover subtle vulnerabilities on the application's deep locations, this indicates that the scanner has an effective crawling component. To understand how black-box fuzzing works, we will use the following example.

Envision a simple web application with a home page, registration page, and course-view page. As shown in Fig. 3, the user can access the course-view page only after completing the registration process. The crawler starts with a seed URL and extracts all reachable pages. The crawler identifies entry points and analyzes web forms to assign input values

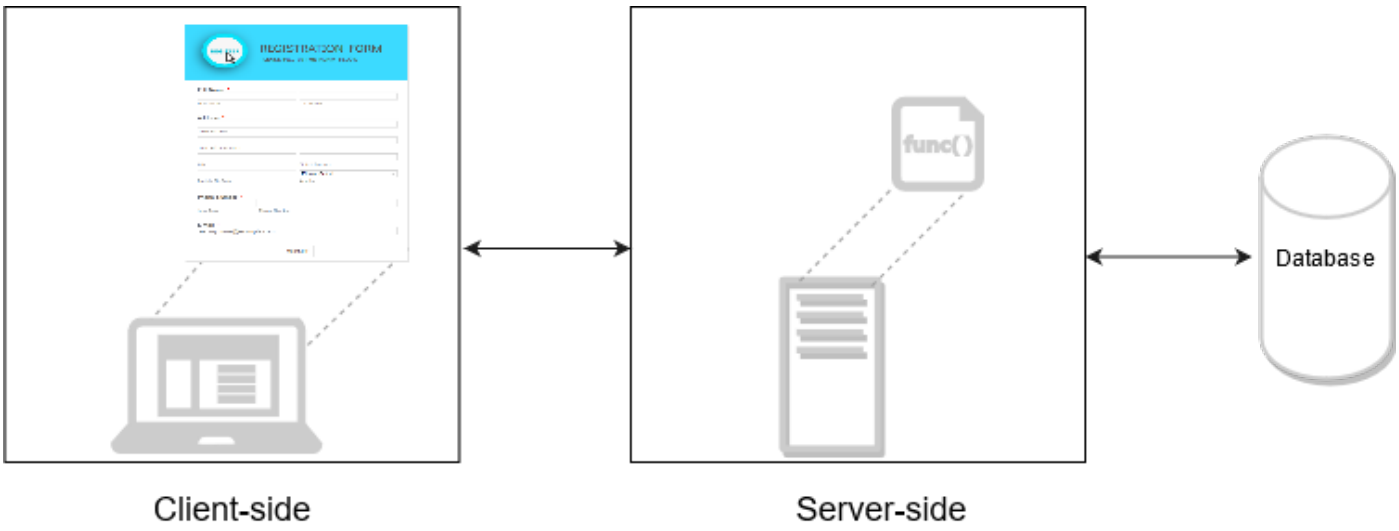

Fig. 1. Web Applications Architecture.

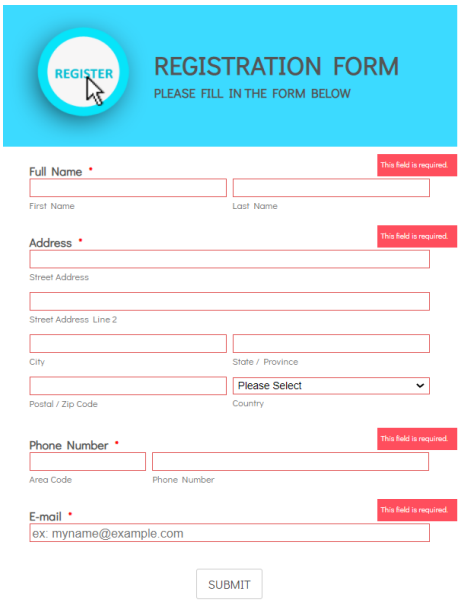

Fig. 2. Web form Input Validation.

Fig. 3. Describes the Navigation Graph of the Example.

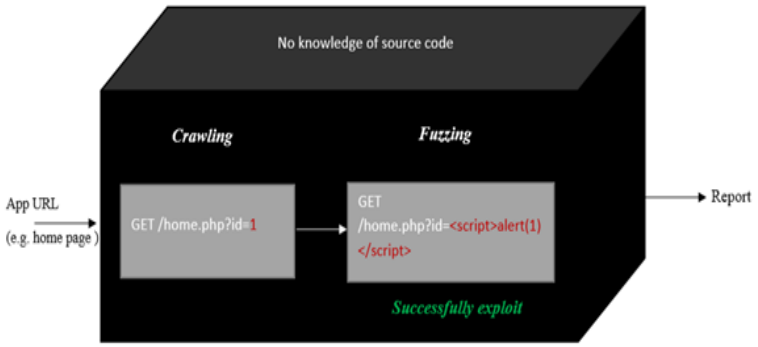

Fig. 4. Black-box Fuzzing Approach.

to discover several links on the application [13]. Then, the attack module fuzzes the application using information learned from the crawler. In other words, the attack module produces attack strings for each input or entry points to fuzz the web application. Finally, the analysis module analyzes the response page after launching the attack to determine where the attack string is reflected in the web application and reports the vulnerabilities [13]. In Fig. 4, the analysis module checks if the script code of the XSS attack is executed.

B. Automated Tools

Many black-box vulnerability scanners, commercial and open-source, are available, all of which have unique characteristics. Here, we review some black-box scanners.

Acunetix [14] is a commercial tool for automatic scanning web applications. It crawls web applications, even the AJAX-heavy ones. It provides different technologies, such as AcuSensor, to increase coverage and obtain higher accuracy for some vulnerabilities, such as accessing the application's source code. It discovers a wide range of vulnerabilities, such as XSS, SQLI, and so on.

AppSpider [15] is a commercial tool to find security bugs automatically. It complies with sophisticated technologies of modern web applications such as AJAX. It has discovered more than 95 attacks, including the OWASP Top 10 vulnerabilities [12]. It can construct values for web forms based on test cases and modify them to belong to different languages and more values.

Burp Suite Professional [16] is a commercial tool that web application scanners provide as part of their features. The scanner covers the most vulnerabilities on OWASP [12], such as XSS, SQLI, XPath, and so on. It has an advanced crawler that discovers vulnerabilities behind advanced JavaScript and dynamic features. It also constructs inputs for forms randomly and uses an intelligent manner to determine what fields are encountered on the page. It provides the ability to tune the configuration of the security test.

Nessus Professional [17] is a commercial vulnerabilities scanner. It can identify vulnerabilities accurately because of its high crawling coverage adapting different technologies to support modern web applications, including Ajax. Additionally, it scans at high speed and has very low false positives.

Netsparker [18] is a commercial web-vulnerability scanner that aims to identify different types of vulnerabilities such as XSS, SQLI, and so on. The crawler can handle different technologies, regardless of their complexity, platform, or architecture. Hence, it can be used for scanning modern web applications. The power of the tool is developed to identify vulnerabilities and gain nearly zero false positives precisely.

Grabber [19] is a free and open-source web-application-vulnerability scanner. It discovers security bugs involving XSS, SQLI, file inclusion, backup files, simple Ajax, and JS source-code analyzer. It scans the web application automatically by identifying the application's entry points into which the data can be injected. It is suitable to use for scanning small web applications.

Vega [20] is a free and open-source web-application scanner. It discovers many vulnerabilities, including SQLI, XSS, remote file inclusion, shell injection, and so on. The Vega crawler can extract URLs and forms and automatically constructs inputs for web applications. Further, it gives the user an additional feature to work as a proxy and be semi-automated to maximize the crawling coverage of web applications.

W3AF (**W**eb **A**pplication **A**ttack and **A**udit **F**ramework) [21] is a free and open-source vulnerability scanner. It discovers different types of vulnerabilities, including XSS and SQLI. Further, the crawler module extracts URLs and forms, and it uses an intelligent manner to give a fake value based on developers' knowledge of the tool.

Wapiti [22] is a free and open-source web-vulnerability scanner. It identifies various vulnerabilities, such as XSS, XXE, XPath, SQLI, and so on. The tool works in this way: the crawler extracts the URLs and forms from HTML, Flash, and basic JavaScript. After that, it launches a payload of attacks on the scripts and forms gained from the crawler to determine vulnerable locations. Finally, it generates reports to show the vulnerabilities and their locations. The crawler support fills form through some placeholder values that are consistent with most policies for web applications.

WFuzz (the **W**eb **Fuzz**er) [23] is a free and open-source web-application-vulnerability scanner. It is designed to perform brute-force attacks against web applications. It can find the essential web pages that are difficult to reach through normal browsing. It also discovers XSS, SQLI, and XXE attacks on GET and POST parameters. Table I shows an overview of the existing web application scanners in the black-box fashion.

TABLE I. SUMMARY OF THE CHARACTERISTICS OF BLACK-BOX WEB APPLICATION SCANNERS

Name	License	Price starts at	Dynamic features
Acunetix [14]	commercial	$4,500	✓
AppSpider [15]	commercial	$2000	✓
Burp Suite Pro. [16]	commercial	$399	✓
Nessus Pro. [17]	commercial	$3,438.50	✓
Netsparker [18]	commercial	•	✓
Grabber [19]	open-source	N/A	⋆
Vega [20]	open-source	N/A	×
w3af [21]	open-source	N/A	×
Wapiti [22]	open-source	N/A	×
WFuzz [23]	open-source	N/A	×
• Not Provided by the Vendor ⋆ Partially Support			

IV. BLACK-BOX FUZZING RESEARCH APPROACHES

To develop an effective black-box fuzzing approach, the increase in crawling coverage has become significant. Thus, this section analyzes the existing primary research in this domain and answers the previously mentioned questions.

Question 1: What are the techniques utilized by the approach?

Many approaches have been used to improve the coverage of the crawler to detect web applications in a black-box fashion. By answering this question, it will be possible to identify the approach and determine if there are any contributions or limitations of these approaches, as the following:

Bisht et al. [24] proposed an approach to detect server-side parameter tampering vulnerabilities in web applications. It is based on extracting constraints from the web forms (i.e., client-side) and uses constraint solving technology to generate test cases that expose the parameter tampering opportunities.

Doupé *et al.* [25], these authors proposed an approach that aimed to enhance the crawling of black-box scanners to discover a wide range of vulnerabilities. Their approach is based on capturing the changes in the application states and using the changes discovered to enhance the crawling coverage. However, their approach does not handle dynamic contents implemented by AJAX. Additionally, it does not support enhanced form submission.

Djuric [26] developed a tool called SQLIVDT to generate SQLI against web applications. His approach uses two types of crawling, which are automated and manual, to identify all the gateways to web applications and can be used to execute SQLI. However, this approach handles dynamic features manually via proxy.

Li and Xue [27] capture web applications' behavior as a finite state machine (FSM) and discover logic-flow vulnerabilities from the difference between the FSM of the expected behavior of an application without bugs and the actual FSM. However, their approach cannot handle dynamic contents.

Pellegrino *et al.* [28] proposed a semi-automated scanner that aims to expand the code coverage of web applications. It uses dynamic analysis on the client-side code to handle JavaScript-based web applications. As a result of this work,

the author improved the crawling coverage by more than 86% compared to other tools. However, this research's primary goal is to analyze the client-side of dynamic web applications without adept form submission. Moreover, this approach is limited to finding XSS vulnerabilities.

Muñoz *et al.* [29] proposed a novel approach to maximize the crawler's code coverage. Their approach is based on analyzing web forms to extract fields that fill with appropriate values from external sources. In general, this approach cannot guarantee that the application's server-side will accept the constructed input values.

Deepa *et al.* [30] proposed *DetLogic* to improve the crawler by discovering the logic vulnerabilities in web applications. *DetLogic* acts as a proxy between the client-side and server-side of the applications that utilizes the information coming from the server to model web application behavior as a finite state machine (FSM). Logical constraints are then constructed from the FSMs to launch attacks. However, their approach is more applicable to static web applications rather than modern web applications. Additionally, this approach does not handle modern web forms that include certain restrictions on the form inputs.

Deepa *et al.* [31] proposed an approach to detect XQuery and parameter-tampering vulnerabilities in XML-database-based applications. Their approach aims to reduce false alarms and expand the coverage of crawlers. To achieve more web exploration, it analyzes the client-side code to handle the constraints of the web forms. However, their approach is limited to find few types of vulnerabilities.

Koswara *et al.* [32] developed *W3AF+* that extends their basic functionality from *W3AF*[21]. The authors developed a traditional crawler to adapt their method to handle dynamic web applications (e.g., Ajax applications). Their method depends on recording changes of the inner states on the applications. They handle the dynamic application by extending their crawler via capturing the changes caused by the event generator and saving the new state of the resulting DOM in the original state machine when it differs from the current state. However, their method is restricted to *on-click* event.

Liu *et al.* [33] developed a tool to increase the crawler coverage of the scanner via filling the required information, such as login forms, through giving correct values by the user when the crawler gives the user an instruction to complete. However, this approach cannot guarantee that the application's server-side code will accept the constructed input values.

Aliero *et al.* [34] proposed an SQLIVS tool that aims to maximize the coverage of crawlers to find subtle vulnerabilities and minimizes false alarms. Their approach is based on analyzing different HTTP responses to determine the presence of SQLI vulnerabilities.

Eriksson *et al.* [35] proposed a tool, *Black Widow*, to maximize the code coverage of black-box-vulnerability scanners in web applications. Their approach is based on capturing the application's inner state to identify the sinks and sources and enhancing the navigation model to navigate more dynamic workflows. However, their approach is limited to finding XSS vulnerability and construct inputs on forms randomly.

Question 2: Is the approach applicable to be used in modern web applications?

Generally, modern web applications have many dynamic features that appear when rendering applications at runtime. These features can generate a change in the pages' DOM, which includes generating contents dynamically, such as forms, links, and so on. Consequently, these DOM changes can affect the navigation graph of the web application, which may impact finding the vulnerable path and, therefore, lead to miss vulnerabilities. Table II shows a comparison of the approaches regarding the adaptation of dynamic features on their crawling.

TABLE II. COMPARISON OF THE APPROACHES REGARDING ADAPTATION DYNAMIC FEATURES

Approach	Dynamic features
[24]	×
[25]	×
[26]	✓
[27]	×
[28]	✓
[29]	✓
[30]	×
[31]	×
[32]	✓
[33]	✓
[34]	×
[35]	✓

Question 3: How does the approach construct benign inputs needed by web applications to explore further and test the application?

Traditionally, constructing inputs for web forms that mimic user interaction with web applications has been done mostly manually or randomly. Filling forms enable users to deal with different types of inputs, such as numeric, text, etc. This variety of input restrictions makes the automatic construction of the correct inputs challenging. Thus, web application analyses infer the required inputs to be generated to explore the application's workflow and find more vulnerabilities. Table III shows a comparison of the approaches regarding constructing inputs on their crawling.

TABLE III. COMPARISON OF THE APPROACHES REGARDING CONSTRUCT INPUTS ON THEIR CRAWLING

Approach	Input generation
[24]	constraint solver
[25]	database (external sources)
[26]	user
[27]	N/A
[28]	N/A
[29]	external sources
[30]	constraint solver
[31]	constraint solver
[32]	N/A
[33]	user
[34]	N/A
[35]	random

V. Discussion

Many researchers focus their efforts on discovering security bugs on the web app before attackers exploit them. In Section IV, we discussed several research efforts tackling the challenge of code coverage on the black-box fuzzing approach. We found that the crawling module of the black-box fuzzing approach is a significant challenge in modern web applications. The first limitation is that many existing research systems still suffer from shallow coverage of the black-box fuzzing approach due to difficulties in handling the dynamic features of modern web apps. Unless the crawling module of the black-box fuzzing approach can support dynamic features such as JavaScript, it is difficult for that fuzzer to analyze most modern web applications that may include critical vulnerabilities behind elements generated on the fly. As shown in Table II, the results indicate that there are no significant differences between studies on supporting dynamic features. In fact, most of the previous studies that handle contents generated dynamically are limited to specific types of events. The second limitation that has hampered crawling involves the design of web forms. Input validation with its restrictions makes it difficult for an automatic generation to fill inputs correctly and submit them to the server-side code. Due to this limitation, the scanners fail to find vulnerabilities in deep locations in web application structures. As shown in Table III, several techniques used to construct input on forms, which do not guarantee that the values will be accepted by the server code, for all previous techniques except [24], [30] and [31]. To precisely infer valid inputs, the study [24], [30], [31], [8], and [9] use constraint solver to construct inputs on the forms by deriving constraints of HTML form inputs.

As a result, the black-box fuzzing approach can achieve better performance when considering the inner state of applications, dynamic features, and input generation. An intriguing research direction is exploring how to combine existing techniques to overcome these limitations and produce a fully automatic scanner in a black-box fashion.

VI. Conclusion

This paper reviews recent and existing techniques regarding black-box fuzzing to discover vulnerabilities. Our survey shows that new technologies and programming capabilities have accelerated the evolution and the complexity of web applications. As a result, automatically analyzing web apps for security purposes becomes more challenging. Additionally, we found that the crawling module of black-box fuzzing approaches still suffers from shallow coverage, which affects identify vulnerabilities on dynamic web applications. Improved crawling module of the black-box fuzzing approach to include dynamic features is necessary to assist the crawler in discovering more links and, therefore, find more vulnerabilities. Further, it is necessary to consider the practical approaches for input generation that can analyze web forms and infer their restrictions to generate correct input values automatically.

As far as these issues, the central problem of the black-box fuzzing approaches is how to deal with these challenges to enhance performance, which leads us to propose another area where black-box scanners should be improved.

References

[1] Verizon, "White paper: 2019 data breach investigations report," Verizon Business, Tech. Rep., 2019. [Online]. Available: https://enterprise.verizon.com/resources/reports/2019-data-breach-investigations-report.pdf

[2] Acunetix, "White paper: Acunetix web application vulnerability report 2020," Acunetix, Tech. Rep., 2020. [Online]. Available: https://www.acunetix.com/acunetix-web-application-vulnerability-report/

[3] M. C. Martin and M. S. Lam, "Automatic generation of xss and sql injection attacks with goal-directed model checking." in *17th {USENIX} Security Symposium*, 2008, pp. 31–44.

[4] G. Wassermann, D. Yu, A. Chander, D. Dhurjati, H. Inamura, and Z. Su, "Dynamic test input generation for web applications," in *Proceedings of the 2008 international symposium on Software testing and analysis*, 2008, pp. 249–260.

[5] A. Kieyzun, P. J. Guo, K. Jayaraman, and M. D. Ernst, "Automatic creation of sql injection and cross-site scripting attacks," in *2009 IEEE 31st international conference on software engineering*. IEEE, 2009, pp. 199–209.

[6] P. Bisht, T. Hinrichs, N. Skrupsky, and V. N. Venkatakrishnan, "Waptec: Whitebox analysis of web applications for parameter tampering exploit construction," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 575–586. [Online]. Available: https://doi.org/10.1145/2046707.2046774

[7] S.-K. Huang, H.-L. Lu, W.-M. Leong, and H. Liu, "Craxweb: Automatic web application testing and attack generation," in *2013 IEEE 7th International Conference on Software Security and Reliability*. IEEE, 2013, pp. 208–217.

[8] A. Alhuzali, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "Chainsaw: Chained automated workflow-based exploit generation," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 641–652.

[9] A. Alhuzali, R. Gjomemo, B. Eshete, and V. Venkatakrishnan, "{NAVEX}: Precise and scalable exploit generation for dynamic web applications," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 377–392.

[10] M. N. khalid, M. Iqbal, M. T. Alam, V. Jain, H. Mirza, and K. Rasheed, "Web unique method (wum): An open source blackbox scanner for detecting web vulnerabilities," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 12, 2017. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2017.081254

[11] F. Duchene, S. Rawat, J.-L. Richier, and R. Groz, "Kameleonfuzz: evolutionary fuzzing for black-box xss detection," in *Proceedings of the 4th ACM conference on Data and application security and privacy*, 2014, pp. 37–48.

[12] OWASP TOP 10, "Owasp top ten web application security risks — owasp," 2021. [Online]. Available: https://owasp.org/www-project-top-ten/

[13] A. Doupé, M. Cova, and G. Vigna, "Why johnny can't pentest: An analysis of black-box web vulnerability scanners," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2010, pp. 111–131.

[14] acunetix. Acunetix web vulnerability scanner. [Online]. Available: https://www.acunetix.com/

[15] Rapid7. Appspider - application scanner. [Online]. Available: https://www.rapid7.com/products/appspider/

[16] PortSwigger. Burp suite - application security testing software. [Online]. Available: https://portswigger.net/burp

[17] Tenable. Nessus professional vulnerability assessment. [Online]. Available: https://www.tenable.com/products/nessus

[18] Netsparker. Netsparker - web vulnerability scanner. [Online]. Available: https://www.netsparker.com/

[19] R. Gaucher. Grabber. [Online]. Available: http://rgaucher.info/beta/grabber/

[20] Subgraph. Vega vulnerability scanner. [Online]. Available: https://subgraph.com/vega/index.en.html

[21] w3af. w3af - web application attack and audit framework. [Online]. Available: http://w3af.org/

[22] N. Surribas. Wapiti web application scanner. [Online]. Available: https://wapiti.sourceforge.io/

[23] C. Martorella, C. del ojo, and X. Mendez. Wfuzz - the web fuzzer. [Online]. Available: https://wfuzz.readthedocs.io/en/latest/index.html

[24] P. Bisht, T. Hinrichs, N. Skrupsky, R. Bobrowicz, and V. Venkatakrishnan, "Notamper: automatic blackbox detection of parameter tampering opportunities in web applications," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 607–618.

[25] A. Doupé, L. Cavedon, C. Kruegel, and G. Vigna, "Enemy of the state: A state-aware black-box web vulnerability scanner," in *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12)*, 2012, pp. 523–538.

[26] Z. Djuric, "A black-box testing tool for detecting sql injection vulnerabilities," in *2013 Second International Conference on Informatics & Applications (ICIA)*. IEEE, 2013, pp. 216–221.

[27] X. Li and Y. Xue, "Logicscope: automatic discovery of logic vulnerabilities within web applications," in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, 2013, pp. 481–486.

[28] G. Pellegrino, C. Tschürtz, E. Bodden, and C. Rossow, "jäk: Using dynamic analysis to crawl and test modern web applications," in *International Symposium on Recent Advances in Intrusion Detection*. Springer, 2015, pp. 295–316.

[29] F. R. Muñoz and L. J. G. Villalba, "Web from preprocessor for crawling," *Multimedia Tools and Applications*, vol. 74, no. 19, pp. 8559–8570, 2015.

[30] G. Deepa, P. S. Thilagam, A. Praseed, and A. R. Pais, "Detlogic: A black-box approach for detecting logic vulnerabilities in web applications," *Journal of Network and Computer Applications*, vol. 109, pp. 89–109, 2018.

[31] G. Deepa, P. S. Thilagam, F. A. Khan, A. Praseed, A. R. Pais, and N. Palsetia, "Black-box detection of xquery injection and parameter tampering vulnerabilities in web applications," *International Journal of Information Security*, vol. 17, no. 1, pp. 105–120, 2018.

[32] K. J. Koswara and Y. D. W. Asnar, "Improving vulnerability scanner performance in detecting ajax application vulnerabilities," in *2019 International Conference on Data and Software Engineering (ICoDSE)*. IEEE, 2019, pp. 1–5.

[33] C.-H. Liu, W.-K. Chen, and C.-C. Sun, "Guide: an interactive and incremental approach for crawling web applications," *The Journal of Supercomputing*, vol. 76, no. 3, pp. 1562–1584, 2020.

[34] M. S. Aliero, I. Ghani, K. N. Qureshi, and M. F. Rohani, "An algorithm for detecting sql injection vulnerability using black-box testing," *Journal of Ambient Intelligence and Humanized Computing*, vol. 11, no. 1, pp. 249–266, 2020.

[35] B. Eriksson, G. Pellegrino, and A. Sabelfeld, "Black widow: Blackbox data-driven web scanning," *proceedings of IEEE SSP*, 2021.