Applying Fuzzing in Software Testing: A Case Study on Kawn Subscriptions Manager

Arnaldo Marulitua Sinaga, Ivanowsky Fernandes Habeahan, Riyanthi Angrainy Sianturi, Willy Susilo, and Yohanssen Pratama

***Abstract*— Fuzzing is an automated black box testing method that evaluates abnormal inputs to trace targeted vulnerabilities. In this research, fuzzing is implemented on the Kawn Subscriptions Manager application. Further, we use the mutation testing method to assess the ability and the success of fuzzing in finding vulnerabilities in the application. The web fuzzer used in fuzzing is FFUF, and the input or payload tested is generated based on the word list required to test each function. A total of 4 mutants were generated and by performing mutation testing, those 4 mutants were successfully killed. Therefore, a 100% mutation score is obtained. It means that the fuzzing method using the FFUF web fuzzer successfully found vulnerabilities in software applications. In addition, it was found that the Django application has implemented strict security against the POST request method. Based on the research findings, we suggest fuzzing all functions in future research. In addition, it is necessary to compare fuzzing with other similar methods to identify the capability and reliability of fuzzing. In addition, our research highlights the importance of integrating comprehensive security measures and testing frameworks in the web application development lifecycle. By using FFUF, we demonstrate an efficient approach to identify and mitigate potential security threats, ensuring robust protection against cyberattacks.**

***Index Terms*— Fuzzing, FFUF, Mutation testing, Payload, Software security.**

I. INTRODUCTION

FUZZ testing is a promising technique that has been used to uncover many bugs and vulnerabilities [1]. This technique involves creating many test cases to repeatedly evaluate target programs while observing any exceptions that occur. These exceptions serve as signals of potential security issues. Typically, fuzz testing uses a queue of seeds, which are particularly interesting inputs, and new inputs are continuously generated by mutating these seeds in an endless loop [2]. Compared to other methods, fuzzing requires little knowledge of the target. It can be easily implemented in large applications and thus has become the most popular vulnerability discovery solution, especially in the industry [3].

In this research, the web application that will use the fuzzing method is Kawn Subscriptions Manager. This web-based application is designed to manage subscription packages for F&B (Food and Beverage) businesses on the Kawn application. Software testing with fuzzing methods is done in real execution; therefore, fuzzing obtains a high level of accuracy. One tool that can be used in this fuzz method is the FFUF tool. FFUF, or "Fuzz Faster You Fool", is a fast web fuzzer written in Go that allows typical directory discovery, virtual host discovery (without DNS records), and GET and POST parameter fuzzing.

In Kawn Subscriptions Manager, there is also data on clients, namely F&B businesspeople, essential to keeping safe. For this reason, it is vital to apply fuzzing to the Kawn Subscriptions Manager. However, fuzzing may not be effective in testing different web applications because fuzzing requires sufficient resources, such as time and computing power, to generate extensive and diverse inputs [3]. Therefore, a fault-based testing method with a mutation analysis approach is used to measure the effectiveness of fuzzing testing in identifying vulnerabilities in web applications or software [4].

Mutation-based fuzzing is a quality assurance tool that is becoming increasingly popular for its ability to uncover critical bugs and security vulnerabilities in widely used software systems [5]. Mutation testing plays a crucial role in this context by replacing or modifying existing parts of the source code, resulting in code variants known as 'mutants'. This approach is not just about identifying bug disclosure results generated by fuzzer, but also about providing mutation scores as additional feedback to evaluate the fuzzing results contained in the Kawn Subscriptions Manager. Its inclusion in the testing process ensures a comprehensive examination of the software's vulnerabilities, instilling confidence in the security professionals and researchers. According to Jia and Harman [6], mutation testing enhances the fault detection capability of testing strategies, making it an invaluable tool in software security assessments [7]. By employing mutation testing, we ensure that the vulnerabilities are not only detected but also that the detection mechanisms are rigorously validated, thereby reinforcing the overall security posture of the application [8].

A. Sinaga is a with the Faculty of Vocational Studies at Institut Teknologi Del (IT Del), Indonesia (e-mail: aldo@del.ac.id*).

I. Habeahan is with the Faculty of Vocational Studies at Institut Teknologi Del (IT Del), Indonesia (e-mail: ivanowskyfernandez@gmail.com).

R. A. Sianturi is with the Faculty of Vocational Studies at Institut Teknologi Del (IT Del), Indonesia (e-mail: riyanthi@del.ac.id).

W. Susilo is with the Faculty of Engineering and Information Sciences at the University of Wollongong, Australia (e-mail: willy_susilo@uow.edu.au).

Y. Pratama is with the Graduate School of Science and Technology at Nara Institute of Science and Technology, Japan (e-mail: yohanssen.pratama.yl0@is.naist.jp).

This paper consists of five sections: section 2 explains the studied methods, namely Fuzzing and Mutation Testing. Section 3 explains the studied object and conducted experiment. Section 4 explains the results obtained from the experiment. Section 5 describes the conclusions and potential further research.

II. Procedure for Paper Submission

A. Software Security Testing

Software systems and applications are frequently released with numerous features and settings [9]. These elements cater to users and the underlying platforms for various purposes, including architectural configurations, virtualization, performance optimization, security and access control, privacy, and system-level interactions [10]. Software testing is a solution to verify whether the built application is by the expected requirements [11]. Testing is more than just debugging. Testing is not only used to find defects and fix them. It is also used in validation, verification processes, and reliability measurements. Therefore, software testing is essential to ensure no errors in the application [12].

Software testing aims to identify the mistakes and features or functions that do not match the expected requirements so that they can be corrected immediately. Properly tested software products can ensure quality, security, and reliability. Testing is a viable approach to detecting implementation bugs that have a security impact, aka vulnerability [13]. A software vulnerability is a security flaw, glitch, or weakness in software code that an attacker could exploit to harm the stakeholders of a software system [14]. Therefore, it can be beneficial in terms of cost efficiency, time savings, and, most importantly, customer satisfaction.

Software security is the idea of designing software to continue functioning correctly in the face of malicious attacks [7]. Web application security is a significant part of any web-based online business. The widely accessible nature of the Internet exposes web assets to possible attacks from multiple locations with varying degrees of scope and sophistication. Web application security is about the security of web applications, websites, and web services such as APIs. It also aims to address vulnerabilities [15].

Ensuring robust web application security involves implementing various practices and technologies designed to detect, prevent, and mitigate potential threats. This includes the use of secure coding practices, regular security testing, and the adoption of frameworks and tools that provide built-in security features [14]. Additionally, it is essential to stay updated on the latest security threats and trends, such as SQL injection, cross-site scripting (XSS), and distributed denial-of-service (DDoS) attacks, which can exploit vulnerabilities in web applications. By regularly conducting vulnerability assessments and penetration testing, organizations can identify and rectify weaknesses before they are exploited by malicious actors. Furthermore, integrating security into the software development lifecycle (SDLC) ensures that security considerations are addressed at every stage of development, thereby enhancing the overall resilience of web applications against cyber threats [16].

Security testing is a process carried out to find security vulnerabilities in software or applications [13]. It will have various tests to ensure that the developed system is fully protected against multiple threats of cyberattacks. The purpose of this testing is to find loopholes and weaknesses in the system that can lead to loss of data or company reputation. One type of software security testing is vulnerability scanning. Vulnerability Scanning (Vuln Scan) is an automated data security test. Software scans for system vulnerabilities such as cross-site scripting, SQL injection, command injection, path traversal, and insecure server configuration. This tool is often referred to as part of Dynamic Application Security Testing (DAST). DAST tools dynamically analyze a running application's responses to various inputs, simulating potential attack scenarios in real-time [16]. According to OWASP [17], DAST is crucial for detecting vulnerabilities that occur only during runtime, making it an essential component of a comprehensive security testing strategy. By combining DAST with other testing methods such as fuzzing and static analysis, organizations can achieve a multi-layered defense mechanism, thereby significantly enhancing the security posture of their applications [18].

Vulnerability itself is a deficiency or defect in a computer system that can be found in software, hardware, protocols, and even in security policies. Vulnerability is an application error that will eventually cause failure to violate security properties that should constantly be monitored [13]. Vulnerability is what can allow attacks that disrupt the system in terms of confidentiality, integrity, and availability. The vulnerability causes severe damage to information systems and software. Therefore, many efforts have been mobilized to overcome these vulnerabilities. One of the right solutions to overcome this problem is the fuzzing method. Fuzzing, by systematically injecting malformed inputs and monitoring for unexpected behaviors, offers a proactive approach to uncovering hidden flaws. As noted by Tsankov et al. [18], the effectiveness of fuzzing in revealing critical vulnerabilities has made it a cornerstone in contemporary cybersecurity strategies. By identifying and addressing these vulnerabilities early, organizations can significantly reduce the risk of exploitation and enhance the robustness of their systems [19].

B. Fuzzing

It is also essential to understand that internet security testing is not only about testing security functions such as authentication and authorization that can be implemented in applications. It is equally important to test the secure implementation of other functions (e.g., using business logic, correct input validation, and output coding) [7]. Properly tested software products can ensure quality, safety, and reliability [13]. Fuzzing or fuzz testing is one of the black box testing methods performed in automation, which evaluates abnormal inputs to trigger targeted vulnerabilities [20]. Barton Miller coined the term fuzzing, referred to as "the act of software torture". Fuzz testing is a promising technique that

has been used to uncover many bugs and vulnerabilities [1]. Compared to other methods, fuzzing requires little knowledge of the target. It can be easily implemented in large applications and thus has become the most popular vulnerability discovery solution, especially in the industry. This method is used by major industries such as Google, Microsoft, Amazon, Meta, and others [7].

In software testing, fuzzing stands out for its unique approach. Unlike other methods that use predefined test scenarios, fuzzing relies on random or arbitrary inputs to test the application. This approach is driven by the main purpose of fuzzing, which is to find vulnerabilities or bugs in the application by testing how it handles invalid or unexpected inputs. This distinct approach sets fuzzing apart from other testing methods.

Fuzzing, by its nature, does not have a definite 'expected result' like other testing methods. Its focus is on studying the system by providing unexpected inputs to see how the application or system responds. The goal of fuzzing is to uncover unexpected or unwanted conditions that can lead to bugs or threaten the system's security. This goal underscores the unique value of fuzzing in software testing.

One of the buzzers that can be used in fuzzing is FFUF. FFUF, or "Fuzz Faster You Fool", is a fast web fuzzer written in Go that allows typical directory discovery, virtual host discovery (without DNS records), and GET and POST parameter fuzzing. FFUF is inspired by Wfuzz, an older but very similar web fuzzer [13]. The main advantage of FFUF is in terms of performance over other web fuzzers such as WFUZZ. The way FFUF works is by compiling a list of words that will be used as input for the "fuzzed requests" that are executed.

Despite the advantages, fuzzing alone cannot address all potential vulnerabilities. A combined approach utilizing static and dynamic analysis methods can offer a more comprehensive security assessment. As mentioned by McGraw [14], integrating static code analysis with fuzzing can uncover vulnerabilities that might be missed when these techniques are used in isolation. Static analysis examines the code structure and logic without executing the program, identifying potential security flaws early in the development cycle. This preemptive strategy, coupled with the reactive nature of fuzzing, creates a robust defense mechanism against a wide range of security threats. Additionally, continuous monitoring and updating of security policies, as emphasized by Viega and McGraw, are crucial to adapting to the evolving threat landscape and ensuring sustained protection against new vulnerabilities [21].

C. Mutation Testing

Mutation testing was initially proposed by DeMillo as a method to evaluate the effectiveness of a test suite and to identify areas needing further testing [22]. This process involves repeatedly introducing artificial bugs (mutations) into the software to see if any test cases fail as a result. If at least one test case detects the mutation and fails, the mutant is considered "killed," indicating the test suite's effectiveness for that mutant. Conversely, if no test cases fail, the mutant "survives," which can highlight a weakness in the test suite, unless the mutant is equivalent and cannot be detected. Mutation testing is a testing technique that modifies the program by inserting faults into it to create new versions called mutants [23]. The original program modification process is done by changing the syntax in the program with the mutation operator. Mutation testing is conducted to measure the adequacy of the generated test suite. Ultimately, a mutation score is calculated as the percentage of killed mutants out of all non-equivalent mutants tested. A higher mutation score indicates a more effective test suite, while a lower score suggests less effectiveness.

In generating mutants, the program code can be modified by inserting faults into the program. Program modification is done using mutation operators by changing the syntax in the program. Some mutation operators that can be implemented in Python programs include [7]:

- AOD - Arithmetic Operator Deletion
- AOR - Arithmetic Operator Replacement
- ASR - Assignment Operator Replacement
- COD - Conditional Operator Deletion
- COI - Conditional Operator Insertion
- CRP - Constant Replacement
- ROR - Relational Operator Replacement
- IOD - Overriding Method Deletion
- IOP - Overridden Method Calling Position Change
- SCD - Super Calling Deletion
- SCI - Super Calling Insertion
- DDL - Decorator Deletion
- SDL - Statement Deletion

The measurement used for mutation testing is named mutation score. The mutation score is calculated by using the following formula [1].

$$score = \frac{mutk}{muts+mutk} \times 100\% \quad (1)$$

The mutation score is the ratio of killed mutants ($mutk$) divided by all mutants (the sum of killed and survived mutants). Suppose the execution of mutants with test inputs given PASS or the execution results of mutant test inputs are different from the original. In that case, it means that the defect represented by this mutant is detected in such a way that the mutant is killed. Otherwise, the mutant survives, which means that the given input is not able to detect such a defect [7].

Generally, mutation scores can be used to reflect the ability of a given input to detect bugs. Higher mutation scores indicate that the test inputs are effective in identifying and killing mutants, which correlates with a higher likelihood of detecting real-world defects. As Offutt et al. [24] suggest, leveraging mutation scores provides a quantitative measure of test effectiveness, thereby offering a robust metric for assessing the quality of fuzzing efforts and the overall reliability of the testing process [7].

III. The Experiments

A. Kawn Subscription Manager

Kawn Subscription Manager is a web-based system designed to manage subscription packages for F&B (Food and Beverage) businesses using the Kawn application. It can help manage the subscription process of F&B merchants who subscribe to the Kawn application. This system can also facilitate the determination of the subscription period, the number of subscribers, and the subscription renewal process.

The development of Kawn Subscription Manager is due to the increasing number of F&B businesspeople who use the Kawn application by subscription. Therefore, the company needs help managing the subscription process, extending the subscription period, activating, and deactivating subscriptions, and monitoring subscription packages that F&B businesspeople have purchased. In Kawn Subscriptions Manager, there are three modules: Clients, Users, and Subscriptions. The features that will be used in fuzzing testing are as follows:

- Authentication
- Create outlets
- Create a subscription plan
- Create subscription

B. Test Case Analysis

In software testing, fuzzing does not use test scenarios. Instead, it relies on random or arbitrary inputs to test the application. The random or arbitrary inputs are called test cases in fuzzing [13]. Test cases in fuzzing are different from test cases in general software testing, which usually include predefined inputs and expectations of desired results.

The test scenario for fuzzing with the ffuf buzzer is the function of ffuf itself, namely:

- Finding pages and directory
- Virtual host discovery
- Fuzzing parameter

C. The Experiment

The stages that are conducted in this experiment are depicted in Fig. 1.

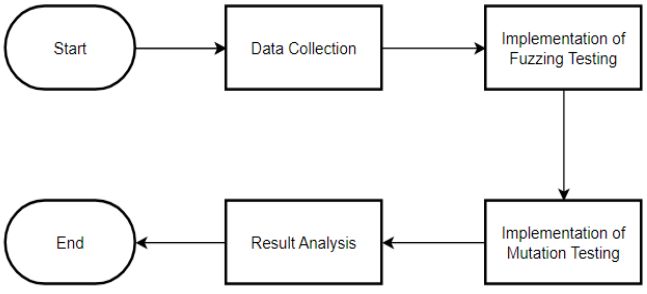

Fig. 1. Research Design

The first step in testing Kawn Subscription Manager is data collection. This process involves collecting information, facts, or data relevant to and necessary for the research. The second stage is applying fuzzing testing to the application. The stages carried out in fuzzing testing include target identification, generating fuzzed data, and test execution.

The third stage is done by applying mutation testing to measure the ability and success of fuzzing in finding vulnerabilities in the application. Stages in mutation testing include generating mutants and mutation testing. The fourth stage is result analysis. This stage is carried out by analyzing the test results that have been obtained and writing them into a test report containing the testing activities that have been carried out, test results, and conclusions.

The details of the testing stages with fuzzing testing can be seen in Fig. 2.

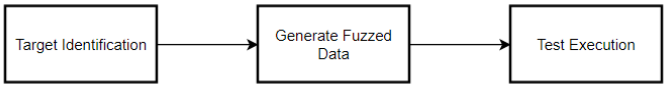

Fig. 2. Fuzzing Testing Stages

The first step in testing Kawn Subscription Manager is to identify the target or test object, namely Kawn Subscription Manager, by diagnosing what pages and fields will be tested. By identifying the target, we will get information on critical components and aspects of Kawn Subscription Manager, such as pages, APIs, or functions that receive input from users.

The second stage is to generate fuzzed data. This stage is done by compiling a wordlist composed of relevant words according to the function being tested. This wordlist will be used as a payload. The words contained in this wordlist are called test cases in fuzzing. This stage helps in trying out various possible inputs that may not be covered in regular test cases.

The third stage is test execution. This stage is a stage carried out by executing each word or test case that has been generated previously with the fuzzing method. This test execution process is done automatically with the ffuf web fuzzer. The details of the testing stages with mutation testing can be seen in Fig. 3.

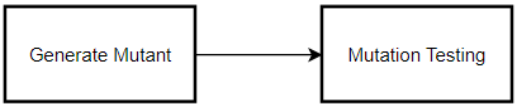

Fig. 3. Mutation Testing Stages

The first stage is generating mutants. At this stage, a mutant program is created. Mutant programs are generated by modifying the feature program code that has been tested in the previous stage. The program code modification will be done manually.

The second stage is mutation testing. This stage involves testing test cases against the mutant program that has been created. The test cases used are the same ones used to test the original program. At this stage, the mutation score calculation process is carried out.

Implementation of testing with fuzzing and experimentation with mutation testing on the Kawn Subscriptions Manager is summarized in the following steps: (i) Implementation of Testing with Fuzzing:

a. Fuzzed data creation is done by compiling a wordlist for relevant inputs in testing.
b. Fuzzed data is used to test finding pages and directories, virtual host discovery, and fuzzing parameters (GET and POST).

The testing is conducted by executing commands through the command prompt.

(ii) Experimentation with Mutation Testing:

c. Creating mutant programs is done by modifying the code of previously tested features.
d. Mutant programs are created for the functions of finding pages and directories, fuzzing parameters (GET and POST).

Mutation testing is carried out by running test cases on the mutant programs and comparing the results with the original program. The goal is to check whether the created test cases can kill the mutant programs that have been made.

IV. EXPERIMENTAL RESULT

This research obtained the results of the fuzzing implementation on two functions of the Kawn Subscriptions Manager application and the adequacy calculation of the resulting test inputs. The results obtained in the finding pages and directories test can be seen in Table 1.

TABLE I
TEST RESULTS GET PARAMETER FUZZING

Payload	Expected Output (Status Code)	Actual Output (Status Code)	Result
name	200	200	PASS
address	200	200	PASS
ads123	200	200	PASS
city_read	200	200	PASS
province_read	200	200	PASS
@sad#$	200	200	PASS
subscriptionplan	200	200	PASS
expires	200	200	PASS
active	200	200	PASS
trial_unit	200	200	PASS
recurrence_unit	200	200	PASS
is_active	200	200	PASS

Based on the results of the GET parameter fuzzing test in Table 1, it was found that all payloads were successfully executed. From the results obtained, it can also be concluded that there is no vulnerability in the GET parameter fuzzing test. Even though all payloads get a status code 200, which means OK where the server accepts and processes the payload request sent, the GET parameter is not vulnerable because it is tested on parameters for data filters.

TABLE II
MUTATION TESTING RESULT GETS PARAMETER FUZZING

Payload	Original	Mutant 1	Mutant 2	Mutant 3	Mutant 4
name	PASS	FAIL	FAIL	PASS	PASS
address	PASS	FAIL	PASS	PASS	PASS
ads123	PASS	PASS	PASS	PASS	PASS
city_read	PASS	PASS	FAIL	PASS	PASS
province_read	PASS	PASS	FAIL	PASS	PASS
@sad#$	PASS	PASS	PASS	PASS	PASS
subscriptionplan	PASS	PASS	PASS	PASS	FAIL
expires	PASS	PASS	PASS	PASS	FAIL
active	PASS	PASS	PASS	PASS	FAIL
trial_unit	PASS	PASS	PASS	FAIL	PASS
recurrence_unit	PASS	PASS	PASS	FAIL	PASS
is_active	PASS	PASS	PASS	PASS	PASS

The execution of mutants with test inputs given PASS or the results of the execution of mutant test inputs is different from the original, meaning that the defect represented by this mutant is detected in such a way that the mutant is killed. Otherwise, the mutant survives [7]. So, based on Table 2 mutation testing results on GET parameter fuzzing, it can be obtained that all four mutants that have been created can be killed by at least one test input. This Parameter Fuzzing means that all faults injected in the four mutant programs can be detected as illustrated in Fig. 4.

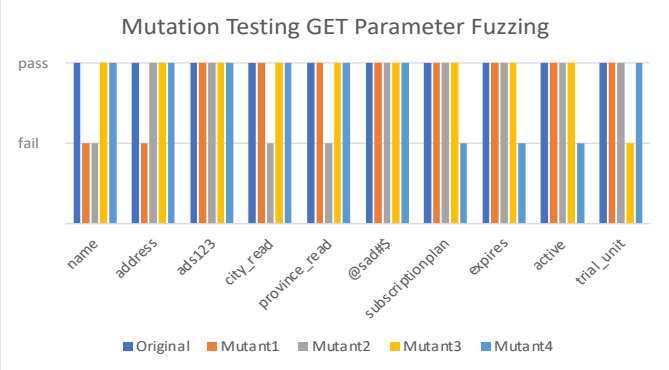

Fig. 4. Mutation Testing Get Parameter Fuzzing

Based on the mutation testing results of GET parameter fuzzing, we can then calculate the mutation score with mutk about 4 and muts about 0 such as the score is about 100%.

Based on Table 3, the functions that have mutation scores calculated are finding pages and directories and GET parameter fuzzing. In addition, there are virtual host discovery and POST data fuzzing functions. However, the reason the virtual host discovery function is not shown is that the function test did not get any results due to Kawn Subscription Manager being run locally on localhost. POST data fuzzing is not shown because fuzzing cannot penetrate Django's POST security, which means Django's security is very good [25]. According to the Django Documentation (2023), Django has built-in protection against most types of CSRF (Cross-Site Request Forgery) attacks. CSRF protection works by checking the credentials in each POST request. By way of explanation, every POST request sent in Django will generate the csrf token used. Through that csrf token, Django will check whether the source that generated the csrf token is the same as the source that made the POST request. If not, then Django will block the request.

TABLE III
TESTING RESULT

Function	Payload	Mutant	Kill Mutant	Survived Mutant	Mutation Score

Finding pages and directories	10	2	2	0	100%
GET parameter fuzzing	12	4	4	0	100%

Both functions in Table 3 have the same mutation score. The result of the mutation score on both functions is 100%. From the mutation score results of each function tested using mutation testing. Based on the results of calculating the average mutation score, the average mutation score on the application of fuzzing in testing the Kawn Subscription Manager Application measured using mutation testing is 100%.

V. CONCLUSION

The conclusions obtained from the research with the title Applying Fuzzing in Software Testing: A Case Study on Kawn Subscriptions Manager are fuzzing testing using the FFUF web fuzzer was successfully applied to the Kawn Subscriptions Manager test. Testing with fuzzing against the original program in Kawn Subscriptions Manager shows that all expected results match the actual results. Not all functions of fuzzing, namely POST data fuzzing, can be implemented in Django applications. Fuzzing is unable to attack Django's POST method; Django has built-in protection against most types of CSRF (Cross-Site Request Forgery) attacks. Therefore, Django's POST security is perfect. The ability and success of fuzzing in finding vulnerabilities in the Kawn Subscriptions Manager were successfully evaluated using the mutation analysis method. They obtained results showing that the test inputs tested with the application of fuzzing have a mutation score level of 100%, which means that the fuzzing method using the web fuzzer ffuf, is capable and successful in finding vulnerabilities in software applications with the mutation testing method. However, this study should be continued with further research. It is important to compare fuzzing testing with other testing methods. This can help identify the advantages and disadvantages of each method.

ACKNOWLEDGMENT

This research is fully supported by the Del Institute of Technology, Indonesia in providing the necessary facilities**.**

REFERENCES

[1] G. Klees, A. Ruef, B. Cooper, S. Wei, and M. Hicks, "Evaluating fuzz testing," in *Proceedings of the ACM Conference on Computer and Communications Security*, Association for Computing Machinery, Oct. 2018, pp. 2123–2138. doi: 10.1145/3243734.3243804.

[2] X. Zhu, S. Wen, S. Camtepe, and Y. Xiang, "Fuzzing: A Survey for Roadmap," *ACM Comput Surv*, vol. 54, no. 11s, Sep. 2022, doi: 10.1145/3512345.

[3] J. Li, B. Zhao, and C. Zhang, "Fuzzing: a survey," *Cybersecurity*, vol. 1, no. 1, Dec. 2018, doi: 10.1186/s42400-018-0002-y.

[4] X. Zuo, X. Yang, Z. Dou, and J. R. Wen, "RUCIR at TREC 2019: Conversational Assistance Track," in *28th Text REtrieval Conference, TREC 2019 - Proceedings*, National Institute of Standards and Technology (NIST), 2019. doi: 10.1145/1122445.1122456.

[5] "05 µ2- Using Mutation Analysis to Guide Mutation-Based Fuzzing".

[6] Y. Jia and M. Harman, "An Analysis and Survey of the Development of Mutation Testing," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649–678, 2011. doi: 10.1109/TSE.2010.62.

[7] I. Synopsys, "What Is Fuzz Testing and How Does It Work," Synopsys, Inc.

[8] Feradhita, "Security Testing," https://www.logique.co.id/blog/2021/03/02/security-testing/.

[9] T. Hamilton, "What is Software Testing," 2022.

[10] S. Dass and A. S. Namin, "Vulnerability coverage for adequacy security testing," in *Proceedings of the ACM Symposium on Applied Computing*, Association for Computing Machinery, Mar. 2020, pp. 540–543. doi: 10.1145/3341105.3374099.

[11] D. Gollmann, *Computer Security*. Wiley, 2011.

[12] J. Viega and G. McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way. Addison-Wesley Professional*. Addison-Wesley Professional, 2001.

[13] M. Mattsson, "A comparison of FFUF and Wfuzz for fuzz testing web applications."

[14] G. Mcgraw, "Building Security In," 2004. [Online]. Available: www.computer.org/security/

[15] F. Duchene, "Detection of Web Vulnerabilities via Model Inference assisted Evolutionary Fuzzing," 2014. [Online]. Available: https://hal.archives-ouvertes.fr/tel-01102325

[16] OWASP Foundation, "OWASP Testing Guide," https://owasp.org/www-project-web-security-testing-guide/latest/.

[17] OWASP, "OWASP Top Ten," https://owasp.org/www-project-top-ten/.

[18] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. Vechev, "Securify: Practical Security Analysis of Smart Contracts," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, in CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 67–82. doi: 10.1145/3243734.3243780.

[19] Django, "Django Documentation," https://buildmedia.readthedocs.org/media/pdf/django/4.2.x/django.pdf.

[20] M. R. Sampurna, "NetPLG Journal of Network and Computer Applications Implementasi Hydra, FFUF, dan WFUZZ dalam Brute Force DVWA," vol. 1, no. 2, 2022. [Online]. Available: https://jurnal.netplg.com/

[21] G. Mcgraw, "Software Security." [Online]. Available: www.cigital.com

[22] H. Du, V. K. Palepu, and J. A. Jones, "To Kill a Mutant: An Empirical Study of Mutation Testing Kills," in *ISSTA 2023 - Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, Association for Computing Machinery, Inc, Jul. 2023, pp. 715–726. doi: 10.1145/3597926.3598090.

[23] R. Gopinath, P. Görz, and A. Groce, "Mutation Analysis: Answering the Fuzzing Challenge," Jan. 2022, [Online]. Available: http://arxiv.org/abs/2201.11303

[24] P. Ammann and J. Offutt, *Introduction to Software Testing*. Cambridge University Press, 2016. doi: 10.1017/9781316771273.

[25] A. Derezińska and K. Hałas, "Analysis of mutation operators for the Python language," in *Advances in Intelligent Systems and Computing*, Springer Verlag, pp. 155–164, 2014. doi: 10.1007/978-3-319-07013-1_15.