

## Detecting vulnerabilities in website using multiscale approaches: based on case study

Mudassor Ahmed Chowdhury<sup>1</sup>, Mushfiqur Rahman<sup>1</sup>, Sifatnur Rahman<sup>2</sup>

<sup>1</sup>Department of Computer Science Engineering, Daffodil International University, Dhaka, Bangladesh

<sup>2</sup>Department of Computer Science Engineering, Siddheswari College, Dhaka, Bangladesh

### Article Info

#### Article history:

Received Sep 18, 2023

Revised Jan 17, 2024

Accepted Jan 18, 2024

#### Keywords:

Comprehensive protection  
Distributed denial-of-service  
Multiscale approaches  
Python-based scanner  
Security misconfiguration  
Structured query language injection  
Web security

### ABSTRACT

In the realm of modern web applications, security stands as an utmost priority. To address this critical concern, we've developed a versatile Python script with the primary goal of proactively identifying vulnerabilities and thwarting transient attacks. Leveraging various libraries, this tool comprehensively covers a broad spectrum of threats, including SQL injection (SQLi), cross-site scripting (XSS), cross-site request forgery (CSRF), sensitive data leakage, security misconfiguration, distributed denial-of-service (DDoS) vulnerabilities, and secure socket layer (SSL) or transport layer security (TLS). This Python-based solution prioritizes adaptability, ensuring seamless integration of future updates to effectively combat evolving threats. Utilizing innovative methods such as SQLi and XSS payload injection, the script assesses the susceptibility of input fields. And addressing CSRF vulnerabilities, the script generates and validates tokens, fortifying defenses against unauthorized actions. Employing pattern analysis, it combats sensitive data exposure and security misconfigurations, adeptly identifying elements like credit card numbers, passwords, and headers. Furthermore, the script enhances overall security by scrutinizing SSL/TLS protocols and monitoring port accessibility. It reinforces DDoS detection by actively monitoring traffic patterns, identifying anomalies, and proactively averting disruptions.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



### Corresponding Author:

Mushfiqur Rahman

Department of Computer Science Engineering, Daffodil International University

Dhaka, Bangladesh

Email: mushfiqur.cse@diu.edu.bd

## 1. INTRODUCTION

In today's ever-evolving digital landscape, safeguarding the security of websites stands as a paramount necessity. The protection of sensitive data and prevention of unauthorized access are critical aspects of maintaining a secure online environment. The existence of vulnerabilities within websites introduces substantial risks, as malicious actors can exploit them to gain unauthorized access or compromise user information. Effectively detecting and mitigating these vulnerabilities is imperative for upholding a robust web security posture. This study centers on the development of a customizable Python script designed to thwart short-term attacks by adeptly identifying vulnerabilities within websites. The script utilizes established vulnerability detection libraries, employing a multiscale approach for identifying common vulnerabilities like SQL injection (SQLi), cross-site scripting (XSS), cross-site request forgery (CSRF), sensitive data leakage, security misconfiguration, and distributed denial-of-service (DDoS) issues.

## 2. KEY FEATURES

The scanner employs advanced multiscale methodologies, representing a paradigm shift that significantly bolsters precision and extends the scope of vulnerability detection. This innovative approach ensures a meticulous examination of web applications at multiple levels, thereby enhancing efficacy in identifying potential security threats. A notable strength of the scanner lies in its commitment to adaptability, acknowledging the dynamic nature of the web security landscape. Its proactive stance in staying ahead of emerging threats through continuous vigilance, maintenance, and updates makes it a reliable and forward-thinking solution. Apart from its versatile capabilities across various scales, the scanner demonstrates exceptional proficiency in specific resolution tasks, exhibiting unmatched efficacy in addressing significant threats.

## 3. RELATED WORKS

Unauthorized access to systems is often achieved through techniques like SQL injection (SQLi) and cross-site scripting (XSS). SQLi exploits vulnerabilities in database queries, allowing attackers to manipulate or extract sensitive data. Prevention involves implementing parameterized queries and input validation. XSS, on the other hand, injects malicious scripts into web pages, with stored XSS being particularly potent. Protection measures include input validation, secure coding practices [1]. The proposed tool teaches for educational assessment on cyber security about how to check or detect vulnerabilities of web applications where focus on possible attacks like SQLi and XSS [2]. In the context of educational websites in Bangladesh, SQL injection (SQLi) vulnerabilities present significant risks. These include normal, error-based double query, and blind injection techniques, empowering attackers to manipulate database queries and potentially compromise sensitive data. It underscores the importance of robust security measures such as code audits, input validation, and the adoption of secure coding practices to mitigate these threats effectively [3], [4]. High detection accuracy machine language algorithms (MLA) can guarantee detection being achieved in real-time [5]. In Bangladesh, measuring vulnerabilities of government websites is in a critical state [6]. This proposed work aims to combat CSRF threats in web applications through real-time vulnerability scans using Python, addressing the significant concern of unauthorized actions on trusted websites as outlined in Open Worldwide Application Security Project (OWASP) top 10 web application attacks list by monitoring form counts within the targeted web application or local host addresses for organizations [7].

SQLi attacks are a significant security concern for web applications, enabling attackers to gain unrestricted database access and access sensitive data; existing solutions often fall short or have adoption limitations [8]–[10]. The study delves into web application vulnerabilities, emphasizing the importance of penetration testing, and offers a comprehensive review and comparison of penetration testing tools to assist testers in selecting the most suitable technology for enhancing web security [11], [12]. In response to the increasing importance of cybersecurity in the digital age, the project aims to create a web portal using Python and Flask to facilitate communication and develop automated tools for detecting SQLi, XSS, and content discovery, including the integration of skip fish for vulnerability scanning. Additionally, the project includes the creation of a dummy website susceptible to operating system (OS) command injection for ethical penetration testing, with detailed reports and user consent for patching vulnerabilities [12].

A tool designed for conducting controlled and scalable cybersecurity experiments, enabling the execution and logging of malicious actions, including software exploits and intrusion detection alerts, while staying updated with threat intelligence data [13]. XSS is a web application attack where malicious code is injected into the application's output, potentially transferring sensitive data to attackers. Traditional methods focus on server-side prevention but often leave vulnerabilities unresolved for extended periods. This paper proposes a client-side solution that tracks sensitive data within the browser, empowering users to make decisions about data transfer to third parties, thus adding an extra layer of protection during web browsing, independent of the application's security measures. This approach enhances user security while navigating the web [14]. The escalating importance of web security is driven by the surging prevalence of web applications and the concurrent increase in web-based attacks. It introduces an automated web vulnerability scanner capable of identifying SQLi and XSS vulnerabilities in web applications. And also, this study successfully identified numerous potentially vulnerable websites, including prominent organizations and government entities, prompting security improvements in response to the findings [9], [10]. The growing threat of malware to organizations and governments, highlighting its potential for data exposure, operational disruption, and severe consequences. It outlines various deployment methods for malware, with a focus on software injection, particularly SQLi, a persistent web application security risk. While secure development practices help mitigate risk, an accessible Python package, provides a solution for identifying and thwarting SQLi attacks. Leveraging state-of-the-art machine learning algorithms, enhancing web application security [15]. And the threat of SQLi and XSS attacks to modern websites, emphasizing their prevalence and potential

for serious damage. It presents an in-depth review of these attacks, their vulnerabilities, and prevention methods, while also discussing future countermeasure developments [16]–[18].

Metamorphic security testing for web-interactions (MST-wi) automates security testing in modern web systems, addressing the oracle problem by integrating input generation strategies and utilizing engineer-specified metamorphic relations. Evaluation on Jenkins and Joomla revealed an 85% detection rate for vulnerabilities, highlighting MST-wi's scalability and efficacy in automated web system security testing [18]. An overview of vulnerability assessment and penetration testing (VAPT) techniques to address the increasing web hacking activities and emphasizes the importance of cybersecurity awareness and measures for organizations to protect against cyber threats [19]–[21]. The ETSS detector tool efficiently identifies multiple vulnerabilities across various URLs simultaneously, enhancing its effectiveness in vulnerability detection of XSS [22].

#### 4. DETECTING METHODOLOGY

Figure 1 showed how detecting code methodology work which will be effective for new organization or any personal user to scanning the vulnerabilities. The provided diagram and accompanying description outline a methodology for conducting web application security testing, with a focus on various security vulnerabilities. Figure 1 showed scanning or testing methodology, the “cybersecurity user” represents the individual or team responsible for initiating the security testing process, while the “Web Application” is the target under scrutiny. The process begins with the cybersecurity user supplying a list of URLs for assessment. This user-initiated approach allows for flexibility and customization, ensuring that the security testing aligns with specific application requirements and potential vulnerabilities. For each uniform resource locator (URL) provided, a series of security checks are systematically executed. The first function called is *check\_sqli*, which is designed to probe for SQLi vulnerabilities.

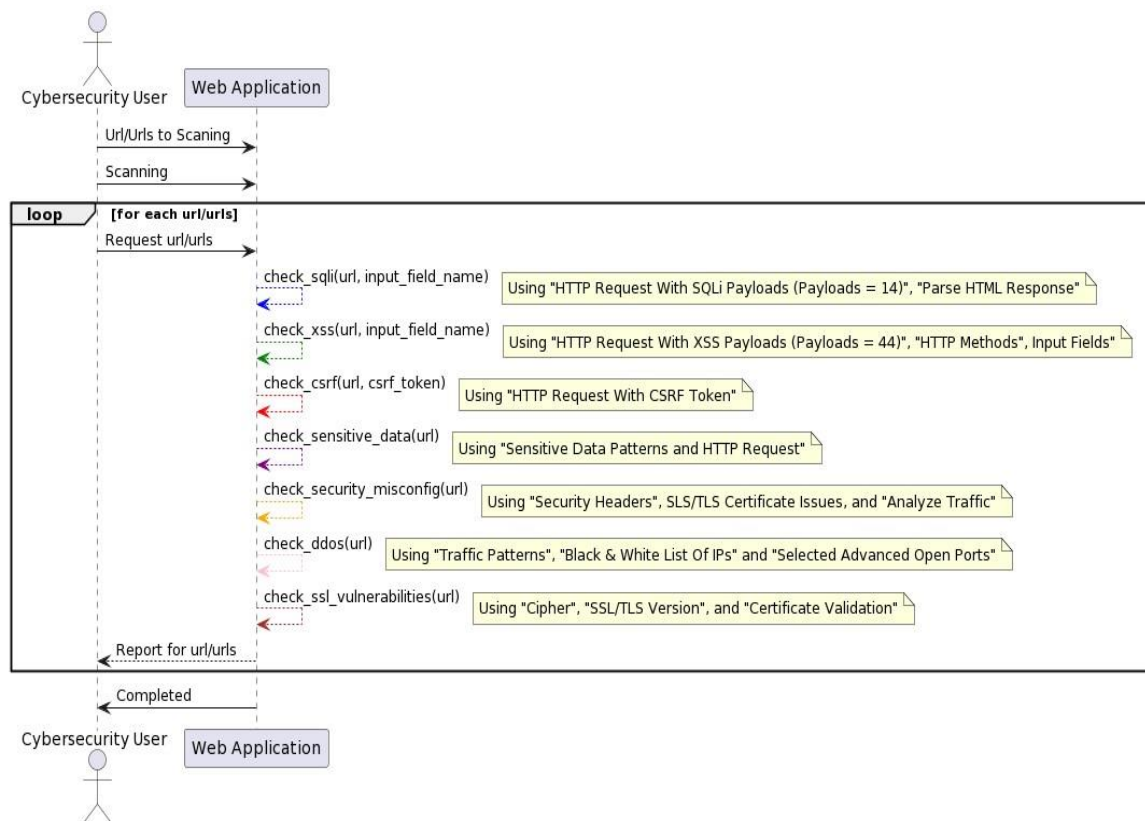


Figure 1. Process of vulnerability scanning

In the contemporary digital landscape, the imperative of securing websites is paramount to shield sensitive data and thwart unauthorized access. This study introduces a meticulously developed Python script

designed for the proactive detection of vulnerabilities in websites, thus fortifying defenses against potential short-term attacks. In this Figure 1 the script's methodology revolves around sending a series of Hypertext Transfer Protocol (HTTP) Power on Self-Test (POST) requests to a specified URL, each carrying a distinct SQL injection (SQLi) payload. Parsing the HTML response using BeautifulSoup, the script checks for the presence of the payload in the response text. The identification of SQLi payloads in the response signals a potential vulnerability in the URL's input fields. SQL injection, a malicious technique involving the insertion of harmful SQL statements, poses a severe threat to web application databases. As this attack occurs at the application level, traditional network layer defenses prove ineffective. Therefore, the script's proactive approach plays a pivotal role in identifying and mitigating this critical security threat. It accomplishes this by sending a series of HTTP POST requests to the specified URL, each containing a different SQLi payload. After each request, the hypertext markup language (HTML) response is parsed using BeautifulSoup, and the code checks if the payload is present in the response text. If any SQLi payloads are detected in the response, it indicates a potential SQLi vulnerability in the URL's input fields. SQL injection is a code injection technique where an attacker inserts some malicious SQL statements to break the barrier of accessing the databases of the web applications [8], [23], [24]. It can be detected using various methods, including manual code review and automated vulnerability scanners. SQL injection attack is executed at application level, where normal firewall and intrusion detection at network layer have no defense against of the attack [25]. Once the SQLi payload is complete, the go through or shifts to the *check\_xss* function, a vital component specifically tailored to unveil cross-site scripting (XSS) vulnerabilities, a prevalent security issue in web applications. When starting the *check\_xss* function then it plays a crucial role in our defense strategy. It meticulously crafts an extensive set of XSS payloads, encompassing both server-side and client-side variants. These payloads are methodically injected into diverse input fields, employing various HTTP methods and input field types. It targets server side and client side both of those code to rigorous testing; we ensure comprehensive coverage. In addition, our code diligently examines the implementation of encoding or sanitization functions, adding an extra layer of defense to the application's security posture. This proactive stance towards identifying and mitigating XSS vulnerabilities is not just a security measure; it is a commitment to preserving the integrity of user data and fostering trust in our digital environment. As we know the process of script is circle process that is why the next step is CSRF, the *check\_csrf* function takes center stage. This function is intricately designed to uncover CSRF issues lurking within web applications. CSRF attacks, wherein unauthorized actions are triggered on behalf of authenticated users, pose a significant threat. The *check\_csrf* function generates a random CSRF token with a defined expiration time. This token is then seamlessly integrated into an HTTP request sent to the specified URL, incorporating it in the request data, headers, and cookies. Detection of a successful form submission without proper CSRF token validation becomes a red flag, signaling the presence of a CSRF vulnerability.

The *check\_sensitive\_data* function stands as a pivotal element in our methodology. In the ever-evolving digital landscape, where inadvertent exposure of critical information—ranging from credit card numbers to email addresses and social security numbers—poses a significant risk, this function plays a critical role in our proactive security measures. To uncover these potential risks, the code executes a methodical process. It initiates an HTTP request to the specified URL, retrieves the page's HTML content, and meticulously scans for patterns indicative of various types of sensitive information. Notably, the search extends beyond the webpage itself to encompass linked files, including JavaScript and CSS. This thorough examination of script ensures that sensitive data remains shielded, even in associated resources.

Now the *check\_security\_misconfig* function, which main focus in the proactive security testing of web applications. It dedicates attention to HTTP response headers, conducting thorough checks for the presence and security of headers such as "Server," "X-Powered-By," and others. Additionally, it extends its scrutiny to SSL/TLS certificates, common vulnerabilities like directory listing, and potential issues related to default credentials. And its distinctive advantage lies in its immediacy, delivering real-time feedback about security misconfigurations. This capability empowers developers to swiftly identify, address, and rectify vulnerabilities, thereby enhancing the overall security posture of the application.

The *check\_ddos* also perform proactive security testing of web applications as like *check\_security\_misconfig* function. But the checks are different like the process begins with monitoring incoming traffic, swiftly raising flags when the volume surpasses a predefined threshold and also identifying unusual request patterns associated with DDoS attacks. Moreover, the code incorporates proactive measures, including rate limiting, to address suspicious IP addresses, providing an additional layer of protection against potential attackers. The last function of the script is *check\_ssl\_vulnerabilities* and the process of this function is extracting the hostname from the provided URL and subsequently establishing an SSL/TLS connection to the server. During this connection, the function meticulously checks for weak cipher suites, including those considered vulnerable. If any weak cipher suite is detected, then it is showed in console of the script. And also, if the function identifies 'TLSv1' or 'SSLv3', it reports a vulnerable protocol version.

## 5. REQUIREMENTS

The requirements are consisted of Hardware and Software:

- Hardware: Processor with at least a dual-core architecture (e.g., Intel Core i3, AMD Ryzen 3), RAM at least 4 GB of RAM is recommended to handle the Python script. And its dependencies, (Storage) the storage requirements for the Python script itself are minimal. In summary, a modern dual-core processor, 4 GB of RAM, and minimal storage capacity are sufficient hardware specifications for running the Python script. These hardware requirements are well within the capabilities of most contemporary computers, making the script accessible for a wide range of users and systems. Then the workstation will easy to use for run this script.
- Software: (Operating System) The script should work on any major operating system, including Windows, macOS, and Linux. (Python Interpreter) The script is written in Python, so need to have Python 3.x installed on the system. The script is compatible with Python 3 x versions, (Python Libraries). The script relies on various Python libraries for different functionalities, such as '*secrets*', '*requests*', '*time*', '*re*', '*bs4*' (Beautiful Soup), '*socket*', '*ssl*', '*cryptography*', and '*urllib.parse*'. Ensure that these libraries are installed on Python environment. If need then find this script on GitHub (send a GitHub username to your-username via [email/other communication method]).

## 6. RESULTS

In Table 1 comprising 28 website addresses, a comprehensive vulnerability detection methodology was applied. Each web address underwent scrutiny through various vulnerability functions, systematically assessing potential security weaknesses. This iterative process continued until every website in the list underwent thorough vulnerability checks. The details of the vulnerability assessment results are presented in Table 1, outlining the security posture of each website in the context of the applied methodology.

Table 1. The result of vulnerabilities

SL. No	Web Address	SQLI	XSS	CSRF	Sensitive Data	Security Misconfiguration	DDOS	SLS/TLS
1	<a href="https://p*****f.org/c*****s/">https://p*****f.org/c*****s/</a>	No	No	Yes	No	Yes	No	Yes
2	<a href="https://www.b*.*.*.bd/en/index.**p">https://www.b*.*.*.bd/en/index.**p</a>	Yes	NO	Yes	NO	NO	No	No
3	<a href="https://www.s*****nk.com.*d/">https://www.s*****nk.com.*d/</a>	NO	Yes	Yes	NO	Yes	No	No
4	<a href="http://g*****o.**v.*d/">http://g*****o.**v.*d/</a>	Yes	NO	Yes	NO	NO	No	Yes
5	<a href="https://i**d.**v.*d/">https://i**d.**v.*d/</a>	Yes	NO	Yes	NO	NO	No	No
6	<a href="https://www.n*****.com/">https://www.n*****.com/</a>	NO	Yes	Yes	NO	Yes	No	No
7	<a href="https://www.f*****k.com/">https://www.f*****k.com/</a>	Yes	NO	Yes	NO	Yes	No	No
8	<a href="https://www.al*****s.com/">https://www.al*****s.com/</a>	NO	NO	Yes	NO	NO	No	No
9	<a href="https://www.b****.com/news">https://www.b****.com/news</a>	NO	Yes	Yes	NO	Yes	No	No
10	<a href="https://www.w***.int/">https://www.w***.int/</a>	Yes	Yes	Yes	NO	Yes	No	No
11	<a href="https://www.*****a.gov/">https://www.*****a.gov/</a>	NO	NO	Yes	NO	NO	No	No
12	<a href="https://c*****l.com/">https://c*****l.com/</a>	Yes	NO	Yes	NO	Yes	No	No
13	<a href="https://www.d*****.com.**/">https://www.d*****.com.**/</a>	Yes	NO	Yes	NO	NO	No	No
14	<a href="https://www.r***i.com/">https://www.r***i.com/</a>	Yes	NO	Yes	NO	Yes	No	No
15	<a href="https://b*****y.com/">https://b*****y.com/</a>	Yes	Yes	Yes	NO	Yes	No	No
16	<a href="https://www.te*hor.com/">https://www.te*hor.com/</a>	Yes	NO	Yes	NO	Yes	No	No
17	<a href="https://www.bio*****e.com/">https://www.bio*****e.com/</a>	NO	NO	Yes	NO	Yes	No	No
18	<a href="https://*****d.com.bd/">https://*****d.com.bd/</a>	Yes	NO	Yes	NO	Yes	No	No
18	<a href="https://www.p*****o.com/">https://www.p*****o.com/</a>	NO	Yes	Yes	NO	Yes	No	No
20	<a href="https://www.sq*****.com/">https://www.sq*****.com/</a>	Yes	NO	Yes	NO	NO	No	No
21	<a href="https://www.cy*****s.org/">https://www.cy*****s.org/</a>	Yes	Yes	Yes	NO	Yes	No	Yes
22	<a href="https://www.hackt*****e.org/">https://www.hackt*****e.org/</a>	Yes	NO	Yes	NO	Yes	No	No
23	<a href="https://tr**me.com/">https://tr**me.com/</a>	Yes	NO	Yes	NO	NO	No	No
24	<a href="www.****oo.com">www.****oo.com</a>	Yes	Yes	Yes	NO	Yes	No	No
25	<a href="www.****ook.com">www.****ook.com</a>	Yes	NO	Yes	NO	Yes	No	No
26	<a href="www.*m**l.com">www.*m**l.com</a>	Yes	NO	Yes	NO	Yes	No	No
27	<a href="www.h***il.com">www.h***il.com</a>	NO	NO	Yes	NO	Yes	No	No
28	<a href="https://secur***st.com/">https://secur***st.com/</a>	Yes	Yes	Yes	NO	Yes	No	No

The paper is totally depending on the script which this study made, so the script performs web application network security testing for a list of URLs. It checks for various vulnerabilities such as SQLi, XSS, CSRF, sensitive data exposure, security misconfigurations, DDoS, and SLS/TLS. In addition, to address privacy concerns, it has refrained from explicitly mentioning the names of the specific websites in question. However, for the purpose of academic research and to establish the authenticity of the work described in this methodology, it has maintained a record of the addresses of these websites. These addresses

are available and can be provided as needed to substantiate the research paper's claims and findings. Additionally, to ensure transparency and accessibility, the complete code related to this methodology has been uploaded to GitHub repository if need to know or more study about this code then connected on LinkedIn or Email. By maintaining a record of the website addresses and making the code available on GitHub (if collaborate then send a GitHub username to your-username via [email/other communication method]), it aims to strike a balance between privacy considerations and the need to validate and replicate the research work. From, this table it finds out the percentage of vulnerabilities in each category and identify the most vulnerable category, this study will calculate the total vulnerabilities and the percentage of vulnerabilities for each category. Based on this table collect 28 random data (Educational Purpose) and 7 categories of vulnerability, as well as determine which categories are major or minor, it need to count the number of vulnerabilities in each category and calculate their percentages as shown in Figure 2.

- SQLi: Vulnerabilities found 19 in 28 websites. So that, the calculation will be  $(19/28) * 100 = 67.86\%$  where the vulnerability status “Major”.
- XSS: Vulnerabilities found 9 in 28 websites. So that, the calculation will be  $(9/28) * 100 = 32.14\%$  where the vulnerability status “Major”.
- CSRF: Vulnerabilities found 28 in 28 websites. So that, the calculation will be  $(28/28) * 100 = 100\%$  where the vulnerability status “Major”.
- Sensitive data exposure: Vulnerabilities found 0 in 28 websites. So that, the calculation will be  $(0/28) * 100 = 0\%$  where the vulnerability status “Minor”.
- Security misconfiguration: Vulnerabilities found 20 in 28 websites. So that, the calculation will be  $(20/28) * 100 = 71.43\%$  where the vulnerability status “Major”.
- DDoS: Vulnerabilities found 0 in 28 websites. So that, the calculation will be  $(0/28) * 100 = 0\%$  where the vulnerability status “Minor”.
- SSL/TLS: Vulnerabilities found 2 in 28 websites. So that, the calculation will be  $(2/28) * 100 = 10.71\%$  where the vulnerability status “Minor”.



Figure 2. Percentage of vulnerabilities in each category

## 7. CONCLUSION





This study or research paper presented Python-based scanner which represents a transformative advancement in web security, setting a new standard for vulnerability detection and mitigation. Its multiscale approaches and robust capabilities make it an indispensable tool for safeguarding digital assets in the ever-evolving landscape of web security. The commitment to ongoing adaptability ensures the scanner's enduring relevance and effectiveness against emerging threats. This research not only showcases the scanner's remarkable capabilities but also contributes valuable insights into the dynamic evolution of web security practices.

## REFERENCES





- [1] A. K. Baranwal, "Approaches to detect SQL injection and XSS in web applications," *Eece 571B, Term Survey Paper*. EECE 571B, The University of British Columbia, 2012.
- [2] M. Dua and H. Singh, "Detection and prevention of website vulnerabilities: Current scenario and future trends," in *2017 2nd International Conference on Communication and Electronics Systems*, 2017, pp. 429–435, doi: 10.1109/CESYS.2017.8321315.
- [3] M. Noman, M. Iqbal, and A. Manzoor, "A survey on detection and prevention of web vulnerabilities," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 6, pp. 521–540, 2020, doi: 10.14569/IJACSA.2020.0110665.
- [4] D. Alam, T. Bhuiyan, M. A. Kabir, and T. Farah, "SQLi vulnerability in education sector websites of Bangladesh," in *2015 Second International Conference on Information Security and Cyber Forensics*, 2015, pp. 152–157, doi: 10.1109/InfoSec.2015.7435521.
- [5] S. Rajesh, M. Clement, S. S. B., A. S. S. H., and J. Johnson, "Real-Time DDoS attack detection based on machine learning algorithms," *SSRN Electronic Journal*, 2021, doi: 10.2139/ssrn.3974241.
- [6] M. Moniruzzaman, F. Chowdhury, and M. S. Ferdous, "Measuring vulnerabilities of Bangladeshi websites," in *Proceedings of the 2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, 2019, doi: 10.1109/ECACE.2019.8679426.
- [7] G. Parimala, M. Sangeetha, and R. Andalpriyadharsini, "Efficient web vulnerability detection tool for sleeping giant-cross site request forgery," *Journal of Physics: Conference Series*, vol. 1000, no. 1, Apr. 2018, doi: 10.1088/1742-6596/1000/1/012125.
- [8] W. G. J. Halfond, J. Viegas, and A. Orso, "A classification of SQL injection attacks and countermeasures," in *Preventing SQL Code Injection by Combining Static and Runtime Analysis*, 2008.
- [9] S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic, "SecuBat: a web vulnerability scanner," in *Proceedings of the 15th international conference on World Wide Web*, May 2006, pp. 247–256, doi: 10.1145/1135777.1135817.
- [10] W. Wang, F. Dumont, N. Niu, and G. Horton, "Detecting software security vulnerabilities via requirements dependency analysis," *IEEE Transactions on Software Engineering*, vol. 48, no. 5, pp. 1665–1675, May 2022, doi: 10.1109/TSE.2020.3030745.
- [11] E. A. Altulaihan, A. Alismail, and M. Frikha, "A survey on web application penetration testing," *Electronics (Switzerland)*, vol. 12, no. 5, Mar. 2023, doi: 10.3390/electronics12051229.
- [12] R. Karayat, M. Jadhav, L. S. Kondaka, and A. Nambiar, "Web application penetration testing and patch development using kali Linux," in *2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS)*, Mar. 2022, pp. 1392–1397, doi: 10.1109/ICACCS54159.2022.9785232.
- [13] H. Holm and T. Sommestad, "SVED: scanning, vulnerabilities, exploits and detection," in *2016 IEEE Military Communications Conference*, Nov. 2016, pp. 976–981, doi: 10.1109/MILCOM.2016.7795457.
- [14] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, "Cross-site scripting prevention with dynamic data tainting and static analysis," *Proceedings of the Network and Distributed System Security Symposium, NDSS 2007*, San Diego, California, USA, 2007.
- [15] B. Reeves, "Protego: a python package for SQL injection detection," in *Symposium of University Research and Creative Expression (SOURCE)*, 2022, vol. 88.
- [16] R. Johari and P. Sharma, "A survey on web application vulnerabilities (SQLIA, XSS) exploitation and security engine for SQL injection," in *2012 International Conference on Communication Systems and Network Technologies*, May 2012, pp. 453–458, doi: 10.1109/CSNT.2012.104.
- [17] I. Balasundaram and E. Ramaraj, "An authentication scheme for preventing SQL injection attack using hybrid encryption (PSQLIA-HBE)," *European Journal of Scientific Research*, vol. 53, no. 3, pp. 359–368, 2011.
- [18] N. B. Chaleshtari, F. Pastore, A. Goknil, and L. C. Briand, "Metamorphic testing for web system security," *IEEE Transactions on Software Engineering*, vol. 49, no. 6, pp. 3430–3471, 2023, doi: 10.1109/TSE.2023.3256322.
- [19] P. S. Shinde and S. B. Ardhapurkar, "Cyber security analysis using vulnerability assessment and penetration testing," in *2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave)*, Feb. 2016, pp. 1–5, doi: 10.1109/STARTUP.2016.7583912.
- [20] M. E. Ruse and S. Basu, "Detecting cross-site scripting vulnerability using concolic testing," in *2013 10th International Conference on Information Technology: New Generations*, Apr. 2013, pp. 633–638, doi: 10.1109/ITNG.2013.97.
- [21] T. Singh, "Detecting and prevention cross –site scripting techniques," *IOSR Journal of Engineering*, vol. 02, no. 04, pp. 854–857, Apr. 2012, doi: 10.9790/3021-0204854857.
- [22] T. S. Rocha and E. Souto, "ETSSDetector: a tool to automatically detect cross-site scripting vulnerabilities," in *2014 IEEE 13th International Symposium on Network Computing and Applications*, Aug. 2014, pp. 306–309, doi: 10.1109/NCA.2014.53.
- [23] R. Abirami, D. C. J. W. Wise, R. Jeeva, and S. Sanjay, "Detecting security vulnerabilities in website using python," in *Proceedings of the International Conference on Electronics and Sustainable Communication Systems, ICESC 2020*, Jul. 2020, pp. 844–846, doi: 10.1109/ICESC48915.2020.9155908.
- [24] X.-Y. Hou, X.-L. Zhao, M.-J. Wu, R. Ma, and Y.-P. Chen, "A dynamic detection technique for XSS vulnerabilities," in *2018 4th Annual International Conference on Network and Information Systems for Computers (ICNISC)*, Apr. 2018, pp. 34–43, doi: 10.1109/ICNISC.2018.00016.
- [25] C. Anley, "Advanced SQL injection in SQL server applications," *NGSSoftware Insight Security Research*, 2002.

## BIOGRAPHIES OF AUTHORS







**Mudassar Ahmed Chowdhury**     completed his Bachelor of Computer Science Engineering by education from Daffodil International University in the year of 2023. He specialist in software quality assurance with experience on software development life cycle; software testing life cycle and also hands on experience including multiple test cases and plans, "API Testing" using Postman, "Load Testing" using JMeter, "Automation Testing" using Java (Selenium Framework) and knowledge of "Cyber Security" concept along with hands on experience in "Nessus" and "Brup Suite" tools. He is reachable via LinkedIn and email: amudassor@gmail.com.



**Mushfiqur Rahman**     is a faculty member in the Department of Computer Science and Engineering at Daffodil International University. He is a research enthusiast interested in cyber security, and IoT. He has made significant contributions to numerous productive research articles as either the primary author or a co-author. His background encompasses research proficiency and expertise in software application development. He can be contacted through email: [mushfiqur.cse@diu.edu.bd](mailto:mushfiqur.cse@diu.edu.bd).



**Sifatnur Rahman**     is faculty member of Siddheswari College, Dhaka. She has completed his Bachelor in CSE (Engg.) and Masters in CS respectively in the year of 2017 and 2021. She has research interest in cyber security and software testing. She has participated in several national and international journals and conferences as an author and co-author. She can be reach at [sifatrahman.cse@gmail.com](mailto:sifatrahman.cse@gmail.com).