A High-Performance Algorithm and Tool for Detecting Critical Vulnerabilities

1Deshani Gaurav Kiribhai, 2Goda Ronak Jitendrabhai, 3Prof.Dr.C.K. Kumbharana
1Research Scholar., 2Research Scholar, 3Head of Department
1Department of Computer Science,
1Saurashtra University, Rajkot, India

Abstract: As web applications continue to advance and permeate various aspects of our lives, the prevalence of dangerous website security vulnerabilities, including SQL injection and cross-site scripting, poses a significant threat. These vulnerabilities create an enticing opportunity for hackers to exploit websites for personal gain, political motives, or to gain notoriety. While some research and commercial software have been developed to scan and identify these vulnerabilities, this paper introduces an efficient algorithmic approach and accompanying tool specifically designed for web security vulnerability detection. The proposed method addresses the need for accurate vulnerability detection by presenting a comprehensive study and tool. Through experimental evaluations, it is demonstrated that the algorithm achieves a high level of accuracy in detecting vulnerabilities. Moreover, the tool outperforms popular commercial software available in the market, boasting faster performance and the ability to identify less commonly known vulnerabilities such as shell injection or file inclusion.

Index Terms – Website vulnerability, SQL Injection, XSS hack, Advance algorithms, Tools, Statistics, Algorithm.

I. INTRODUCTION

The detection algorithm for web application vulnerabilities discussed in the paper aims to address the security concerns associated with web applications. The authors recognize that websites have become prime targets for hackers due to their increasing role as communication tools and providers of various services. The paper refers to the Symantec Corporation Internet Security Threat Report of 2017, which highlighted that a significant number of scanned websites (76%) contained security vulnerabilities, including high-risk vulnerabilities (9%). These vulnerabilities, such as SQL injection, cross-site scripting, buffer overflow, shell injection, and file inclusion, can be exploited by hackers for economic, reputational, or political gain. To mitigate these risks, the paper emphasizes the importance of website security requirements for programmers and administrators. It introduces a detection algorithm designed to identify web application vulnerabilities effectively. However, specific details about the algorithm and its methodology are not provided in the given excerpt. It is worth noting that the provided information represents a brief summary of the paper's introductory section. Without access to the complete paper, it is challenging to provide a more detailed analysis of the study or discuss its findings.

Based on the provided structure of the article, here are some suggestions for automated testing mechanisms and improvements to detect web application vulnerabilities such as SQLi, XSS, BoF, SI, and FI:

Automated Testing Mechanisms:
- **Static Code Analysis:** Develop algorithms that analyze the source code of web applications to identify potential vulnerabilities. This can involve pattern matching, syntax analysis, and data flow analysis to detect SQLi, XSS, and other common vulnerabilities.
- **Dynamic Application Security Testing (DAST):** Design algorithms that interact with web applications to identify vulnerabilities by sending various inputs and analyzing the responses. This can help detect issues like BoF, SI, and FI.
- **Fuzz Testing:** Utilize fuzzing techniques to generate invalid or unexpected inputs and monitor the application's behavior. This can uncover vulnerabilities by triggering unexpected program states or crashes.

Use of Machine Learning Techniques:
- **Keyword Optimization:** Employ machine learning algorithms to optimize the search keywords used for vulnerability detection. This can involve training models on known vulnerabilities and automatically generating relevant keywords for improved scanning accuracy.
- **Anomaly Detection:** Implement machine learning algorithms for anomaly detection to identify unusual or malicious behavior patterns that may indicate the presence of vulnerabilities.
- **Classification Models:** Train machine learning models to classify web requests or inputs as either safe or potentially vulnerable. This can help reduce false positives and improve the efficiency of the scanning process.
Techniques for Reducing Input Data Space:

- Input Data Filtering: Develop algorithms to filter out irrelevant or non-executable inputs, focusing only on inputs that have the potential to trigger vulnerabilities.
- Input Generation Optimization: Utilize techniques like genetic algorithms or constraint solving to generate a reduced and optimized set of inputs for vulnerability scanning, thus reducing the overall scanning time.

Scanning Tool Development:

- Integration of Algorithms: Develop a comprehensive scanning tool that integrates the proposed algorithms for vulnerability detection, incorporating the strengths of each technique.
- Real-Time Scanning: Design the tool to provide real-time scanning capabilities, allowing continuous monitoring and detection of vulnerabilities as the web application evolves.
- Reporting and Remediation: Enhance the tool's functionality by including reporting features that provide detailed vulnerability analysis and recommendations for remediation.

These suggestions aim to improve the accuracy, efficiency, and effectiveness of vulnerability detection in web applications, utilizing both automated testing mechanisms and machine learning techniques.

II. CONSIDERED WEBSITE VULNERABILITY

- **SQL Injection**

  SQL injection (SQLi) is a security vulnerability that arises from insufficient control over user input, enabling hackers to execute arbitrary commands (Ali, 2018). By exploiting this vulnerability, attackers can insert malicious queries to manipulate the intended SQL query, resulting in data theft or database destruction. This vulnerability is prevalent among websites and is ranked as the top error in the OWASP's list of the ten most common errors (Open Web Application Security Project, 2016). SQLi poses significant risks as it can expose sensitive database information, enable unauthorized data modifications, or even lead to complete data loss.

  **Gupta et al. (2018)**: The study focuses on detecting and debugging DOM-based XSS on mobile cloud-based social networks. The authors present a runtime Document Object Model (DOM) tree generator and nested context-aware sanitization-based framework to mitigate DOM-based XSS vulnerabilities on mobile cloud-based OSNs. The tool works in dual modes: offline mode captures web application module traces and converts them into static DOM trees, while the online mode detects unreliable script injection in the dynamically generated DOM tree at runtime. The evaluation results show the effectiveness of the tool in detecting unreliable/malicious script injections with low false-positive and false-negative rates and acceptable cost performance index.

- **Cross Site Scripting**

  Cross-Site Scripting (XSS) is a widespread vulnerability that enables hackers to inject malicious scripts into the source code of web applications. Typically, XSS attacks aim to circumvent access controls and impersonate users to carry out malicious activities (Antunes and Vieira, 2015). There are three common variants of XSS attacks: reflected XSS, stored XSS, and DOM-based XSS (Deepa et al., 2018a). XSS also holds a prominent position on OWASP's list of common vulnerabilities in web applications (Open Web Application Security Project, 2016).

  **Goswami et al. (2017)**: This study proposes an intuitive approach to XSS attack detection, with a focus on client-server load balancing. The method performs initial client-side checks for vulnerabilities using divergence measures. If the level of doubt surpasses the threshold, the request is canceled; otherwise, it is forwarded to the proxy for further processing. The authors also present an attribute-clustering method supported by the rank aggregation technique to detect confounded JavaScripts. The approach is validated using actual data and aims to detect and prevent various malicious actions like stealing cookies, distributing malware, and collecting user information.

  **Mohammadi et al. (2017)**: The authors suggest applying an encoder to eliminate unreliable input data as the best approach to prevent XSS attacks. They emphasize balancing security and functionality and highlight the importance of using the right encoder to fit the web context, such as HTML and JavaScript contents. The study introduces a security unit testing approach for detecting XSS vulnerabilities arising from improper coding of suspicious data. The unit tests for XSS vulnerabilities are automatically created on each website and evaluated, demonstrating the capability to detect zero-day XSS vulnerabilities with low error rates and better test coverage than existing methods.

  **Dong et al. (2014)**: This study evaluates XSS errors in the HTML5 web standard. The researchers identify 14 XSS attack vectors associated with HTML5 and develop an XSS detection tool specifically focusing on webmail systems. The tool is applied to popular webmail systems, leading to the discovery of seven exploitable XSS vulnerabilities. The research highlights the importance of assessing security in the context of evolving web standards like HTML5. Each of these studies contributes valuable insights to the ongoing efforts to detect, prevent, and mitigate XSS vulnerabilities in web applications. They showcase different approaches and techniques to enhance web security and protect users from malicious attacks.
III. RELATED STUDY

There are two primary approaches to testing web applications for vulnerabilities: the "white box" and "black box" methods.

In the "white box" approach, the source code of the website is analyzed either manually or using code analysis tools such as FORTIFY, Ounce, or Pixy (Fonseca et al., 2007). This method involves a thorough examination of the code to identify potential vulnerabilities. However, when dealing with complex code, detecting errors can be challenging.

On the other hand, the "black box" approach is more effective in such cases. It involves conducting repeated attack tests on an active website, observing and recording the responses to identify and assess vulnerabilities. This approach, also known as test-based detection, does not rely on access to the source code. Black box testing is widely employed and has proven to be a common and effective method for vulnerability detection in web applications.

In recent years, there have been significant advancements in the identification and detection of vulnerabilities on websites. The latest research in this area can be categorized into three groups:

- **Studies on SQL Injection Bugs**: Researchers have focused on investigating SQL injection vulnerabilities, exploring new techniques to identify and mitigate these issues. They have examined different attack vectors, proposed improved detection methods, and developed tools to detect and prevent SQL injection attacks more effectively.

- **Cross-Site Scripting Bugs**: Extensive research has been conducted on cross-site scripting (XSS) vulnerabilities, aiming to enhance detection and mitigation techniques. Researchers have explored various types of XSS attacks, such as reflected XSS, stored XSS, and DOM-based XSS, and have proposed innovative approaches to identifying and mitigating these vulnerabilities.

- **Other Bug Categories**: In addition to SQL injection and cross-site scripting, researchers have investigated various other types of vulnerabilities. This includes but is not limited to vulnerabilities like cross-site request forgery (CSRF), remote code execution (RCE), server-side request forgery (SSRF), XML external entity (XXE) attacks, and insecure direct object references (IDOR). The research in this group focuses on identifying, understanding, and developing effective countermeasures for these vulnerabilities.

These research efforts aim to enhance website security by advancing the understanding of vulnerabilities, developing improved detection techniques, and proposing effective mitigation strategies.

IV. PROPOSED AND IMPROVED ALGORITHMS

Our system is designed to build web crawler algorithms for data collection from websites. The input data consists of a list of source URL addresses. The crawler scans these addresses, identifies sub-URLs, and adds them to the browsing list. This process is repeated iteratively until all the links within the website are listed and stored.

During the crawling process, our algorithm collects various information about the website and its contents. This includes the website URL, title, meta tags, page content, and links. The collected data is then marked and organized by the search engine. The extracted links are subsequently used as input for the vulnerability detection process.

For each type of vulnerability, we have developed a specific detection algorithm, which is explained in detail in the following section. Our approach follows a test-and-feedback methodology, interacting with the web server. Figure 1 illustrates the general workflow of our approach.

Starting with a website as input, we use the crawler process to collect information about the URLs. The corresponding URL filters are created based on each vulnerability type. These filters utilize information from fields, forms, parameters, and variables that users interact with within the application. The filters categorize and reduce the input data URLs.

Our automatic framework employs an HTTP request and HTTP response-based approach to interact with the web application. Incoming requests perform queries and retrieve feedback from various components such as the web server, SQL server, or FTP server. The interactive processes are captured, saved, and analyzed. The framework automatically conducts tests, receives and compares results, and returns values to detect the presence of vulnerabilities. The results are recorded and reported along with relevant information.

By integrating crawling, data collection, vulnerability detection, and reporting, our framework streamlines the process of identifying and assessing security vulnerabilities in web applications.
V. ALGORITHM FOR DETECTING XSS

Based on the characteristics of the XSS, we propose a new algorithm for detecting them. Algorithm 1 shows the operational details of the detecting XSS vulnerabilities algorithm.

Algorithm 1 Detecting XSS vulnerabilities

Step: 1 Input: URLs from the website, XSS payload.

Step: 2 Output: Set of URLs containing XSS vulnerabilities.

Step: 3 Get URLs from the website through the crawler process and data filter.

Step: 4 Initialize a list of payload checking XSS from the XSS bug test function, number the payload, from payload1 to payload.

Step: 5 Resend the requests to server with URLNew = URLOld + payload.

Step: 6 Check whether the server responds to the requests. If yes, this means that the XSS vulnerability exists at the initial URL, record the result and end the checking process. If not, continue to increase i = i + 1; go back to 5 and make all the payloads in the list.

VI. ALGORITHM FOR DETECTING SQL INJECTION

The algorithm used for detecting SQL injection (SQLi) vulnerabilities is also applicable to detect other types of injection vulnerabilities, including Server-Side Includes (SSI), where attackers use ‘injection’ techniques to insert special codes to exploit databases. Algorithm 2 outlines the specific operational steps involved in identifying SQLi vulnerabilities within the system.

Step: 1 Input: URLs from the website, list of special characters.

Step: 2 Output: Set of URLs containing SQLi vulnerabilities.

Step: 3 Get URLs from the website through the crawler process and filter input URL.

Step: 4 Initialize a list of special characters and scripts that test SQLi from the SQLi error handling function; number the characters, from i = 1 to n.

Step: 5 Send the requests to server with URLNEW = URLOLD + Character.

Step: 6 If the server responds to the requests, it means that there is an SQLi vulnerability at the initial URL, record the result and end the process. If not, continue to increase i = i + 1; go back to Step:5 and browse through all the characters in the list.
VII. APPLICATION OF MACHINE LEARNING TECHNIQUES TO IMPROVE VULNERABILITY DETECTION.

Using machine learning techniques, we have implemented database optimization for enhancing database efficiency. The database contains various elements like special characters, payload segments, and strings used for exploiting vulnerabilities, collectively referred to as "payload." These payloads are stored in an array of data, which serves as a parameter for vulnerability detection. Through our optimization efforts, we have streamlined this database, enabling the tools to achieve enhanced time efficiency as outlined below:

**Step 1** Calling the payload in the database (array of data) \( a_i \) in the array with \( n \) payload elements.

**Step 2** Each \( a_i \) is given a priority weight of \( k \). The larger the \( k \) is, the more common and preferably used the payload/character is in the types of attack; the array is arranged in descending order of weight \( k \). The grading of \( a_i \) elements has been made based on an experimental process with a lot of flawed websites, from which we use supervised machine learning techniques to help the machine evaluate what are the most used \( a_i \); those unused \( a_i \) will be eliminated from the array (or list).

**Step 3** Take out the element \( a_i \) for input in step 3 by sequential search algorithm, \( a_i \) with large \( k \) will take precedence. The sequential search algorithm used here is really effective because \( a_i \) has been weighted \( k \) and sorted in descending order according to \( k \).

**Step 4** Update the database.

The application of machine learning, in this case, is conducted based on the scan results, and we always improve and optimize the database. The algorithm will find out which payloads are mostly used, which are most likely to detect vulnerabilities, updating the database by examining which payload has been used in the vulnerability detection algorithm to put the higher weight on that An efficient algorithm and tool 91 payload. The payloads which are less often made the input of the algorithm will be put a lower weight; also, add other payloads. By updating the database as above, our database is not big, but it is highly effective in detecting the vulnerability, saving scanning time, helping to detect the vulnerability more quickly and accurately. Table 1 describes a part of the optimized database: nearly 20 MB database, including sets of the database for detecting SQLi, XSS… each set is approximately 2 MB, composed of more than 500 elements \( a_i \) saved as a file *.txt. The database is the libraries used in the ‘brute force of web vulnerabilities’ that have been optimized by machine learning. The illustrations are shown in Table 1.

<table>
<thead>
<tr>
<th>( k )</th>
<th>( a_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>506</td>
<td>or 2=2</td>
</tr>
<tr>
<td>505</td>
<td>or 2=2--</td>
</tr>
<tr>
<td>504</td>
<td>or 2=2#</td>
</tr>
<tr>
<td>503</td>
<td>or 2=2/*</td>
</tr>
<tr>
<td>502</td>
<td>hacker’ --</td>
</tr>
<tr>
<td>501</td>
<td>hacker’ #</td>
</tr>
<tr>
<td>500</td>
<td>hacker’/*</td>
</tr>
<tr>
<td>499</td>
<td>hacker’or ‘2’='2'</td>
</tr>
<tr>
<td>498</td>
<td>hacker’or ‘2’='2'--</td>
</tr>
<tr>
<td>497</td>
<td>hacker’or ‘2’='2'#</td>
</tr>
<tr>
<td>496</td>
<td>hacker’or ‘2’='2’/*</td>
</tr>
<tr>
<td>495</td>
<td>hacker’or 2=2 or ‘’=’</td>
</tr>
</tbody>
</table>
VIII. SCANNING TOOL

After enhancing the algorithm, we successfully developed a web security vulnerability detection tool known as GKDWebScanner. This scanning tool works as follows:

IX. SYSTEMATIC FLOW OF GKDWEBSITE SCANNER

The system architecture comprises four key components: crawler, website attack, vulnerability analysis, and notification, as described below:

1. Crawler: This component is responsible for fetching the entire content of a website, including all links present on that site and those specified in the "robots.txt" file. The crawler generates a detailed sitemap of the website.

2. Website Attack: Within this section, there are various attack functions designed to target the website system and identify potential vulnerabilities.

3. Vulnerability Analysis: The vulnerability analysis component categorizes identified vulnerabilities into four hazard groups: high, medium, low, and informational levels. It also calculates the total number of vulnerabilities falling into each level.

The criteria used for classifying web vulnerabilities closely align with those utilized by other security tools, as follows:

- **High**: This level represents vulnerabilities through which an attacker can fully compromise the confidentiality, integrity, or availability of the target system without requiring specialized access or user interaction.

- **Medium**: At this level, the attacker can partially compromise the confidentiality, integrity, or availability of the target system without needing specialized access or user interaction.

- **Low**: This level indicates vulnerabilities that allow the attacker to compromise the confidentiality, integrity, or availability of the target system to a limited extent without specialized access or user interaction.

- **Info**: Vulnerabilities categorized as "info" provide information about the system but do not possess the capability to significantly impact the system.

Notification: The notification section contains "*.htm" files that furnish information about detected vulnerabilities, including their locations, and provides recommendations for patching these vulnerabilities. This segment serves as a reference for administrators or users to take appropriate actions in response to the identified vulnerabilities.

X. CONCLUSION

This article discusses common security vulnerabilities found on the web, including SQL injection, XSS. It proposes an algorithm and improvements aimed at enhancing the efficiency of web application vulnerability detection. The scanning tool employed in the study is called GKDWebScanner, which was used to perform tests and compare its performance with commercial software programs such as Acunetix, Nessus, and Arachni using standard datasets. The results indicate that GKDWebScanner exhibits a high rate of error detection for web applications and outperforms the other tools in terms of scanning speed.

To further enhance the tool's capabilities, the authors plan to develop its database and leverage machine learning techniques to boost scanning performance. Additionally, they aim to implement predictive capabilities to identify the likely location of vulnerabilities, thus streamlining the search process.

XI. ACKNOWLEDGMENT

We express our heartfelt gratitude to all those who supported and contributed to this research. Their guidance, resources, and encouragement were instrumental in our success. We are sincerely thankful to each and every individual involved.