STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **x**, Month 202x, pp 0–15.
Published online in International Academic Press (www.IAPress.org)

WebGuard: Enhancing Web Security Through an Integrated Developer Platform

Md. Tanvir Rahman Rafi [1], Md. Shefat Hossain Tonmoy [1],Wahidur Rahman [2], Md. Sazzad Hossain [1,*]

[1] *Department of Computer Science and Engineering, Mawlana Bhashani Science and Technology University, Tangail-1902, Bangladesh*
[2] *Department of Computer Science and Engineering, Uttara University*

Abstract This research presents the development of an integrated developer platform named 'WebGuard'. The proposed integrated platform provides solutions for SQL Injection, Cookie and Session Hijacking, Cross-Site Scripting (XSS), Phishing, Distributed Denial-of-Service (DDoS) attacks, and Malware. This study used input validation by generating automated regular expressions to detect SQL injection. In addition, stored procedures, parameterized queries, and cryptography are used to detect SQL injection. This platform used secure session ID generation and encrypted user authentication to prevent cookie and session hijacking. Here, libsodium is utilized to decrypt user authentication. In this study, the cross-site scripting (XSS) mitigation employs input validation, output encoding, and DOMPurify for advanced sanitization. Distributed Denial-of-Service (DDoS) uses a Content Delivery Network (CDN) in Webguard that contains load balancing, rate limiting, and a comprehensive incident response plan. Webguard provided malware detection service by using file type and size validation and heuristic checks. Furthermore, Phishing attacks are also prevented by the proposed platform. The proposed platform successfully prevented 92.77% of SQL injection attacks out of 828 samples, and it detected 6.16% of the provided samples. Webguard successfully prevented 95.12% of cookie and session hijacking attacks out of 41 samples. The platform successfully prevented 90.95%, and detected 7.41% of XSS attacks, out of 243 samples. This platform successfully prevented 81.82% of DDoS attacks out of 11 samples. In phishing detection, Webguard successfully detected 92.64% out of 231 samples. Finally, this platform successfully detected 87.88% of malware out of 33 samples. Therefore, WebGuard promotes a safer online environment and makes secure development easier for programmers by combining these features in one location.

Keywords SQL Injection, Cookie & Session Hijacking, Cross Site Scripting, Phishing, DDOS, Malware, Web Security

DOI: 10.19139/soic-2310-5070-2457

1. Introduction

In the modern world, web applications play a crucial role in our everyday lives. As time has been evolving, web technologies have become an integral part of us. Web applications are divided into two portions, client side and server side. The client side can be developed using any of the frontend frameworks, and the server side can be developed using any of the backend frameworks. The frontend portion is seen on the UI. The backend is functionalized with the help of a relational (MySQL, PostgreSQL, Microsoft SQL Server, Oracle Database, MariaDB, etc.) or non-relational database (MongoDB, Apache Cassandra, Redis, Oracle NoSQL Database, etc.).

So, it may be possible that web applications might face cyber-attacks by attackers. It might be exploited fatally, such as by SQL injection, Cookie and Session Hijacking, Cross-Site Scripting (XSS), Phishing, Distributed Denial-of-Service (DDoS) attacks, and Malware.

SQL Injection enables attackers to tamper with database queries, possibly gaining access to, altering, or removing

*Correspondence to: Md. Sazzad Hossain (Email: tanvirrafi1999@gmail.com). Department of Computer Science and Engineering, MBSTU. Santosh, Tangail, Bangladesh

ISSN 2310-5070 (online) ISSN 2311-004X (print)

Copyright © 202x International Academic Press

confidential data[1, 2, 3]. Cybercriminals can trick databases into carrying out unwanted operations, such as stealing or changing data, by inserting malicious SQL code into forms [4].

Cookie and Session hijacking occurs when attackers utilize cookies or session identifiers to obtain unauthorized access to a user's account or session[5]. This gives them the ability to pretend to be the user and carry out destructive actions. By obtaining cookies or session IDs through inadequate session management, hackers can assume user identities and access user accounts[6, 7].

Cross-Site scripting (XSS) enables attackers to insert malicious scripts onto web pages that other users are seeing[8, 9]. This could jeopardize the security and integrity of the impacted website through illegal access, data theft, and other harmful activities. Attackers insert malicious scripts into input fields on websites to take control of cookies, redirect users, or change the content of the page from their browser. OWASP has recognized there are four types of XSS attacks—Reflected XSS, Stored XSS, DOM-based XSS, and Mutation-based XSS [10, 11, 12].

Phishing is a type of cyberattack that attempts to deceive victims into disclosing private information, such as login passwords, bank account information, or personal information. These attacks, which pose a serious risk to internet security and privacy, are usually conducted through misleading emails, texts, or websites[13, 14, 15].

A Distributed denial-of-service (DDoS) attack occurs when compromised systems, often infected with malware, overwhelm a target system or network with traffic, blocking users from accessing it [16, 17]. This assault interferes with business activities, resulting in lost revenue, service interruptions, and harm to the organization's reputation. A website is overloaded with traffic from compromised devices, rendering it unavailable to users [18].

Malware is malicious software that aims to damage or take advantage of computer systems, frequently by stealing information, interfering with normal processes, or gaining illegal access. It includes a wide range of dangers to cybersecurity, including Trojan horses, worms, viruses, and ransomware.

A previous report demonstrated that 96% of tested web applications have vulnerabilities and are in danger of possible cyber-attacks, and XSS tops among the attacks [19]. If we want to prevent cyber-attacks, it is very difficult to understand the type of attack we may face. We let the attackers exploit our web applications unknowingly. So, we have to prevent cyber-attacks to ensure the security and performance of our web applications.

The necessity for a comprehensive web platform for preventing cyber threats is very crucial. A unified platform can mitigate cyber threats a lot so that the percentage of getting compromised could be less than average. As of now, there is still a requirement for a comprehensive web platform that can detect various cybersecurity threats. For this reason, the researchers are still working on developing a unified platform so that they will be able to prevent web applications from being compromised by attackers.

2. Related Works

Numerous experiments and research have been conducted on the prevention of cyberattacks. s ongoing work underscores the importance of safeguarding online assets from evolving threats.

Das, D., Bhattacharyya, D.K., and Sharma, U. investigate SQL Injection in web vulnerabilities, evaluating detection techniques and classifying assaults according to online application flaws [20]. They present DUD, a dynamic query matching-based detection technique renowned for its ease of use and precision. However, depending solely on simulated test results for detection, methods like query translation to XML are not empirically validated. Furthermore, there isn't much acceptance of the suggestion that every application includes at least one query in its SQLMF, presenting a gap in practical application testing.

Parveen, K.'s research demonstrates how black patterns in cookie disclaimers emphasize acceptance over user demands, often leading users to unknowingly consent without understanding the implications [21]. The study highlights the advantages of cookies, such as enhancing user experience. However, it overlooks privacy issues, implementation difficulties, and legal compliance, pointing to a lack of comprehensive knowledge on the ethical use of cookies. The study underscores the balance between personalization, user consent, and data security within modern interactions, stressing the need for compliant strategies aligned with evolving regulatory frameworks.

Awad, M., Ali, M., Takruri, M., and Ismail, S. researched web vulnerabilities, including XSS, buffer overflow, cookie hijacking, and SQL injection, providing a foundational understanding of attack vectors [22]. Even though

their study offers a wide range of insightful information, it has some drawbacks, including a narrow scope and potential out-of-date data due to the enormous development of web technology. This limitation, along with its theoretical approach, shows it is nothing but a lack of empirical validation, and it highlights the difficulty of app security. Their study indicates how challenging it is to achieve strong security, strengthening the importance of dedicated defense mechanisms and constant adaptability.

In 2019, another study conducted by Liu, Zhang, Chen, and Zhang analyzed XSS vulnerability exploitation and detection techniques, categorizing and analyzing various exploitation scenarios on a sample website [10]. Despite how it handles prevention strategies and specifies detection techniques into static, dynamic, and hybrid analysis mechanisms, the pragmatic implementation in enterprise environments is restricted to its out-of-depth real-world validation. This elaborative research acknowledges the complexity of software applications and cutting-edge technologies that provide various security concerns regarding the necessity of the highest-level protection methods.

Another study was carried out in 2018 by Bhavsar, V., Kadlak, A., and Sharma, S., demonstrating the typos of phishing attacks and their detection and prevention mechanisms [13]. This study does not provide particular information regarding the defense strategies that could offer an in-depth understanding of individual assaults. Moreover, potential industry standard procedure is bound due to the practical validation, which made future comparisons more challenging.

The 2022 study by Singh and Gupta, about a general review of DDoS mechanisms, does not provide any concrete mechanism for up-to-date defense tactics or how to utilize them in a suitable manner, which may restrict the convenience of enterprise-level software solutions [18]. Though this study emphasizes a unified defense strategy, it still omits the adequacy of detection strategies, which creates an immense possibility of DDoS defense system vulnerability.

Our investigation inspects several problems regarding online vulnerabilities, which opened the door to adapting cutting-edge security frameworks to the world of cyber security. Looking forward to overcoming these challenges, we conducted this research through our integrated developer platform, WebGuard, which enables countermeasures to cyber security threats through an extensive suite of detection and mitigation techniques as well as provides a comprehensive approach to cybersecurity, effectively reducing risks, safeguarding data integrity, and protecting digital Assets by employing advanced threat detection algorithms and proactive defense mechanisms. More details and information on research scopes are located in Table 1.

Table 1. Research Scopes

Authors	Methods Applied	Limitations	Scopes
Das, D., Bhattacharyya, D.K., and Sharma, U [20]	DUD is a dynamic query matching-based detection technique renowned for its ease of use and excellent precision.	XML query translations aren't empirically validated, and SQLMF queries in every app lack broad acceptance.	Use custom patterns with automated regex to detect malicious queries and apply cryptography to secure database access and protect sensitive data.
Parveen, K. [21]	The study shows how dark patterns in cookie disclaimers push acceptance while also underscoring cookie benefits.	Ignores privacy issues, implementation difficulties, and legal compliance.	To encrypt session information using cryptography.

Continued on next page

Authors	Methods Applied	Limitations	Scopes
Liu, Zhang, Chen, and Zhang [10]	This investigation covers XSS exploitation techniques, detection methods, and categorization, illustrating scenarios on a sample website and exploring static, dynamic, and hybrid analysis.	Lacking real-world validation, the study underscores the need for comprehensive protection measures, acknowledging the complexity of online applications and emerging technologies.	To detect and prevent XSS using input validation, output encoding, and DOMPurify for sanitization.
Bhavsar, V., Kadlak, A., and Sharma, S [13]	Phishing attacks involve various types, detection methods, and prevention strategies like email filtering, user awareness, and multi-factor authentication.	The absence of real-world validation and established evaluation criteria makes it harder to compare preventive technologies in the future.	Naive Bayes can be used for URL phishing detection by analyzing features like domain reputation, URL length, and suspicious keywords to classify URLs accurately.
Singh and Gupta [18]	Provide a general review of DDoS defensive mechanisms present in web-enabled computing platforms.	The study lacks detailed defense strategies and overlooks the practicality of detection schemes in real-world scenarios.	To examine the effectiveness of load-balancing strategies in mitigating DDoS attacks and the integration of rate-limiting mechanisms to enhance system resilience.
Awad, M., Ali, M., Takruri, M., and Ismail, S [22]	Investigated web vulnerabilities, including XSS, buffer overflow, session hijacking, SQL injection, and malware.	The statement omits concrete examples, thorough solutions, and a clear explanation of how advancing technologies affect information accuracy.	To handle malicious attacks on web applications.

3. Methodology

The study pinpoints several critical cybersecurity issues, such as the slow uptake of cutting-edge methods, the dearth of real-world validation, and the difficulties in balancing security, usability, and compliance. WebGuard provides real-world validation through extensive testing and deployment, a unified developer platform that integrates advanced detection and mitigation techniques for various cyber threats. WebGuard seeks to minimize the difficulties involved in implementing security while enabling developers to resist cyber threats with its all-encompassing approach successfully. The architectural design of our unified developer platform, WebGuard, is given below:

3.1. Architectural Design

This is the architecture of our web platform, WebGuard, as shown in Figure 1. At first, the user has to sign up for the system via the Firebase Authentication system (Google, GitHub, and Email/Password). Secondly, the user can choose their desired detection method functionality. After that, that request will be sent to the database via the API gateway. The following procedure is considered an asynchronous process; the server will return the response,

previously sent the request by the client, through the same Application Programming Interface (API) gateway. As a result, the users will see the outcome in the user interface.

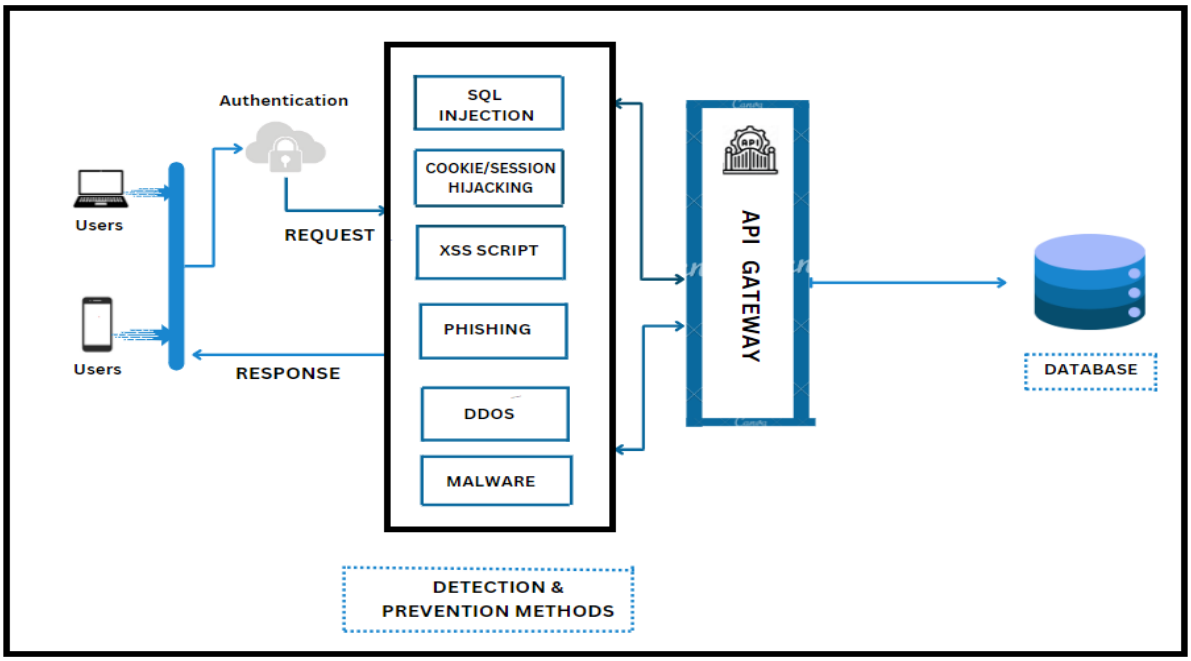

Figure 1. Architecture of WebGuard

3.2. Detecting and Preventing SQL Injection Methods

To maintain the security of databases and applications, identifying and stopping SQL injection attacks is necessary. A detailed approach to detect and prevent SQL injection attacks is as follows:

3.2.1. Input Validation: The input validation system is a technique that ensures the data entered into the system is put together according to pre-established protocols and rules before the data is processed.

- Automated Regular Expression: Based on the input data and the requirements of the application, it can produce any dynamic pattern that can enhance its adaptability, so the security measures can make adjustments to take superiority over potential threats and weaknesses.

  Let Σ be the input character set (e.g., ASCII), and Σ^* denote the set of all possible input strings. We define the language of malicious patterns as a regular language, $R \subseteq \Sigma^*$.

  For any user-submitted input $x \in \Sigma^*$, the detection function $f(x)$ is defined as:

$$f(x) = \begin{cases} 1, & \text{if } x \in R \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

  Where:

  - $f(x) = 1$: input is flagged as potentially malicious (e.g., SQLi, XSS)
  - $f(x) = 0$: input is considered safe

At the initial stage, the developer will log into our system only after he or she decides whether to use the SQL injection prevention method. After that, he/she can go with input sanitization/validation. Here, a developer can check any input inside the code editor to see whether it contains any malicious code or not. If the input does not contain any malicious code, then the system will give a verdict that there will be no action needed. Or else, if there is malicious code existing in the input, then against this input, our system will provide an automated regular expression that can detect this query further, and all of the functionalities have been implemented on the database system.

We have developed this automated regular expression in such a way that for every individual malicious query, there will be a new regular expression generated by our system for detection, and the developer can copy that portion to use in further processing.

- Implementation of PHP Libraries: The PHP addslashes() method's backslashes can be removed using stripcslashes(). This is especially helpful in making sure the data is returned to its original form and safe for further processing.

  To ensure that a string is safe to use in SQL queries, this function is particularly designed to escape special characters from it. By inserting escape characters before characters with particular meaning in SQL queries, including quotes (' and "), it effectively prevents SQL injection attempts by preventing them from being interpreted as part of the query syntax.

3.2.2. Stored Procedure & Parameterized Queries: Stored procedures are precompiled SQL queries stored on the database server, allowing programs to execute predefined operations. Parameterized queries use placeholders for parameters to enhance security and performance by preventing SQL injection. The following explains how stored procedures and parameterized queries are implemented.

- Utizing prepare() or mysqli_prepare() for SQL Statements: This function is utilized to create SQL statements with placeholders, allowing parameters to be securely bound for secure database activities.

- Making use of bind_param() or mysqli_stmt_bind_param() for Binding Parameters: This approach is used to link variables to the placeholders in prepared statements, ensuring proper data types are specified for secure and precise data processing.

- Application of execute() or mysqli_stmt_execute() for Executing Prepared Statements: Finally, the execute() or mysqli_stmt_execute() function is employed to run the prepared statements, returning a boolean value that indicates whether the execution was successful.

3.2.3. Cryptography in SQL Injection Prevention Methods: Cryptography plays an indispensable role in mitigating SQL injection attacks by safeguarding sensitive information within databases. Techniques such as password hashing, encryption, and decryption are employed to integrate cryptographic measures, strengthening security against SQL injection vulnerabilities.

- Password Hashing Using Bcrypt: This method is used to produce hashed passwords; it is probably a component of a particular programming language or framework. Bcrypt is a cryptographic hash function that makes brute-force assaults computationally difficult to prevent. This function hashes the user's password, and the hash produced is kept in the database in place of the password in plain text. While Bcrypt is widely adopted and effective for password hashing, Argon2 (specifically Argon2id) is the OWASP-recommended standard due to its superior resistance to GPU-based attacks and memory-hard properties. A comparison is presented in Table 2 . WebGuard's future releases plan to support Argon2 as a pluggable module for enhanced protection.

- PASSWORD_BCRYPT Algorithm: This algorithm is a PHP hashing method that generates a secure, 60-character hash using the Blowfish encryption algorithm. It includes a unique salt for each password to enhance security and is commonly used with functions like password_hash() and password_verify().

Table 2. Comparison of Bcrypt and Argon2id for Password Hashing

Criteria	**Bcrypt**	**Argon2 (Argon2id)**
Resistance to GPU Attacks	Moderate	High (designed to resist parallelism)
Memory Hardness	Low (constant memory usage)	High (customizable memory cost)
Speed	Slower on CPU, fast enough for web use	Configurable (can be slower for higher security)
Configurability	Cost factor (iterations)	Memory, time, and parallelism parameters
Adoption	Mature, widely used	Newer, OWASP recommended
OWASP Ranking	Acceptable (legacy)	Preferred (2023+)
Use in WebGuard	Bcrypt used for legacy system compatibility; can be upgraded to Argon2	Planned for upgrade

- Encryption and Decryption: The functions encrypt() and decrypt() stand for general encryption and decryption functionalities, which are probably found in computer languages or frameworks. To encrypt and decrypt data, they are usually used in conjunction with a secret key implementation.

  Encryption keys stand for the private key that is both encrypted and decrypted. This key must be kept private since it could be used to decode sensitive data.

  AES encryption algorithm with the CBC mode of operation is referred to as the AES-256-CBC algorithm. Since AES is a symmetric block cipher, encryption and decryption are accomplished with the same key. Block ciphers can operate in a special mode called CBC mode, which encrypts data in blocks.

3.3. Cookie and Session Hijacking Prevention Process (SQL)

Cookie and session hijacking are serious security threats that can compromise user accounts and sensitive data. Here's a process for preventing cookie and session hijacking is shown below.

3.3.1. Generate Secure Session IDs by Using bin2hex(random_bytes(32)): The creation of a secure session identification is the first stage. This is achieved using the code snippet bin2hex(random_bytes(32)), which generates 32 cryptographically random bytes and uses the bin2hex function to convert them into a hexadecimal string. Increasing security, the randomness of the bytes makes it hard to guess the session ID.

3.3.2. Regenerate Session ID: This step emphasizes doing so to strengthen security. This is done using the session_regenerate_id(true) function. Even if an attacker manages to obtain a stolen session ID, it is usually prevented from being exploited by regenerating the session ID.

3.3.3. User authentication with encrypted data using Libsodium: This last phase involves user authentication with encrypted data. Libsodium's incorporation enhances this procedure. Reputable cryptographic library Libsodium provides a range of cryptographic features, possibly including encryption. User authentication data is protected during transmission by encryption, which also makes it more difficult for hackers to intercept and steal credentials.

3.4. XSS Detection and Prevention Methods

Cross-site scripting is a vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users. Here are some methods for detecting and preventing XSS attacks, as shown below.

3.4.1. Input Validation : We can limit the characters and formats that can be provided by checking user input. Attackers will find it more difficult to insert harmful scripts into your application as a result. Consider a form where people can input their names. Validation can be used to restrict input to just letters, spaces, and hyphens. By doing this, attackers would be prevented from entering something similar in the name field, such as `<script>alert(1)</script>`.

3.4.2. Output Encoding: A vital defense against XSS assaults is output encoding. It functions by first converting user input that is not to be trusted into a secure format before showing it on the website.

- User Input: Through forms, comments, and other interactive features on your website application, users can provide data.
- Encoding Process: This data is sent through an output encoding function by the server when it is received. This function changes some characters (such as ¡, ¿, and &) in the user input that have particular semantics in HTML into harmless entities.
- Safe Display: The encoded data is subsequently added to the web page's HTML code. The browser sees the altered special characters as ordinary text rather than attempting to run them as code because they have a transformation.

3.4.3. DOMPurify for Sanitization: A well-known JavaScript package called DOMPurify was created expressly to clean HTML code and stop XSS assaults. The operations of DOMPurify for Sanitization are as follows:

- Integration: The JavaScript code for your web application includes the DOMPurify module.
- Sanitization Procedure: You give the HTML content as a string to the DOMPurify function when you have user-generated HTML that needs to be presented safely.
- Safe Output: DOMPurify scans the HTML code and eliminates or changes any elements or attributes that might be dangerous. This can entail deleting all script tags, encoding unusual characters, or eliminating risky event handlers.
- Clean HTML: The sanitized HTML code that DOMPurify finally delivers is safe to integrate into your website.

3.5. Phishing Detection Methods(NOSQL)

To classify phishing URLs, we adopt the Naive Bayes algorithm, which estimates the probability of a class label (Phish or Legit) given the input features.

Let $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ represent the feature vector of a URL. Using Bayes' theorem:

$$P(\text{Phish} \mid \mathbf{x}) \propto P(\text{Phish}) \cdot \prod_{i=1}^{n} P(x_i \mid \text{Phish}) \tag{2}$$

Likewise, for the legitimate class:

$$P(\text{Legit} \mid \mathbf{x}) \propto P(\text{Legit}) \cdot \prod_{i=1}^{n} P(x_i \mid \text{Legit}) \tag{3}$$

The model predicts the class with the highest posterior:

$$\hat{y} = \arg \max_{c \in \{\text{Phish,Legit}\}} P(c) \cdot \prod_{i=1}^{n} P(x_i \mid c) \tag{4}$$

To address zero-probability issues, Laplace smoothing is applied:

$$P(x_i \mid c) = \frac{\text{count}(x_i \text{ in class } c) + 1}{\text{total instances in class } c + |V|} \tag{5}$$

Where $|V|$ is the number of unique values of feature x_i.

We evaluated the suitability of many machine learning models for phishing detection, including Naive Bayes, Random Forest, XGBoost, and BERT-base. Naive Bayes was chosen for its good performance on smaller text-based datasets, ease of use, and computational efficiency. Table 3 presents a comprehensive evaluation of model capabilities, resource requirements, and use-case alignment.

Table 3. Comparison of ML Models for Phishing Detection in WebGuard

Model	**Strengths**	**Weaknesses**	**Suitability for WebGuard**
Naive Bayes	Simple and fast; effective on high-dimensional text like URLs; low computational cost.	Assumes feature independence; struggles with complex relationships or correlated features.	Best suited for phishing detection on small datasets; ideal for quick testing and lightweight classification.
Random Forest	Handles feature interactions well; supports both categorical and numerical data.	Slower training; resource-heavy for large datasets.	Useful for complex data, but not necessary for phishing URLs due to the simplicity of patterns.
XGBoost	Highly accurate on structured/tabular data; robust to missing values and outliers.	Requires tuning; susceptible to overfitting; longer training time.	Better for large-scale phishing detection; less suitable for lightweight, real-time classification.
BERT-base	Excels in contextual NLP understanding; handles complex language patterns.	High computational demand; requires large datasets and fine-tuning.	Overkill for phishing URL tasks; more suitable for semantic analysis or large-scale datasets.

To evaluate the phishing detection capabilities of WebGuard, we implemented a lightweight machine learning classifier based on the Naive Bayes algorithm. The implementation focuses on efficient detection of phishing URLs using handcrafted features derived from real-world phishing datasets. The following summarizes the experimental setup in Table 4:

Table 4. Naive Bayes Model Configuration for Phishing Detection

Aspect	**Details**
Dataset Used	UCI Phishing Websites Dataset (11,055 entries): https://archive.ics.uci.edu/dataset/967/phiusiil+phishing+url+dataset
Data Split	80% Training, 20% Validation and Testing
Feature Engineering	18 handcrafted features from URL and metadata (length, @, //, subdomains, HTTPS, entropy)
Model Used	Multinomial Naive Bayes
Validation Metrics	Accuracy: 83%, Precision: 81.3%, Recall: 1%, F1-Score: 87%
Bias Mitigation	Undersampling of majority class
Cross-validation	5-fold
Tools	Python (Scikit-learn 1.4), Pandas, NumPy

Detecting phishing attacks is crucial for protecting users and organizations from falling victim to fraudulent schemes. Here are several methods for detecting phishing attempts, shown below.

- Build a Model: The first step in identifying phishing emails is to build a machine-learning model. Although the model type isn't stated in the flowchart, Naive Bayes classifiers are a popular option for this kind of work.
- Establish Flask API: The creation of a Flask API is the task of this step. A well-liked Python framework for creating web apps and APIs is called Flask. The API will probably offer a mechanism for other components of your system to communicate with the machine-learning model.
- Create API Requests from React.js: This phase involves creating requests from a React.js application to the Flask API. A JavaScript package called React.js is used to create user interfaces. Emails would probably be sent to the API for phishing detection by the React.js application.
- Machine Learning (Naive Bayes classifier): This section describes the kind of model that can be used to identify phishing attempts. One kind of probabilistic machine learning model that performs well on text categorization problems is the naive Bayes classifier.
- Final step: These highlight how important it is to use SSL certificates that are both secure and validated. User data is shielded from interception by SSL certificates, which encrypt communication between the web server and the client.

3.6. DDoS Prevention Methods

Mitigating Distributed Denial of Service (DDoS) attacks involves an exhaustive strategy that incorporates proactive measures, strong network security tools, and well-defined incident response protocols. Below are diverse, effective techniques to prevent DDoS assaults:

- CDN:A CDN, also known as a content delivery network, is a network of distributed servers that aids in isolating and absorbing DDoS attack traffic. During an attack, a CDN ensures the service of the website remains accessible by redirecting traffic away from the primary server.
- Load Balancing: Through splitting incoming traffic throughout multiple servers, this technique reduces the chances that a DDoS attack might take out a single server.

- Rate Limiting: Rate constraints discourage attackers from congesting the server with requests by limiting the number of requests that a specific IP address or client can send to the server in an appropriate amount of time.

- Cloud-Based DDoS Protection: Sophisticated security features against DDoS assaults are provided by specialized cloud services that shun out hazardous traffic before it hits the origin server.

- Incident Response Plan: For promising an efficient and effective control of such circumstances, a concise incident response plan demonstrates the ways for identifying, locating, and completely recovering from a DDOS attack.

3.7. Malware Detection Methods

Detecting malware is essential for safeguarding systems and data from malicious threats. Here are several methods for detecting malware, shown below.

- Validation of File Type: This function verifies whether the MIME type of the uploaded file is compatible with a list of permitted file types.

- Validation of File Size: To avoid large malware payloads potentially exhausting system resources, this function limits the size of uploaded files.

- Basic Heuristic Checks (Limited Effectiveness): Feature, which may be ineffective because of false positives, compares the file name to a list of frequently occurring malware file extensions.

- Data Transmission to Server for Examination: After we have completed these validations, you can use a library such as Axios to make HTTP requests and send the file data to your backend server for a more thorough analysis. This function sends a POST request to the backend endpoint to analyze the file for additional processing after creating a Form Data object with the chosen file inside to recover from it.

We recognize that heuristic tests by themselves might not be sufficient to counter sophisticated attacks like zero-day or polymorphic malware. Future versions of WebGuard will integrate real-time threat intelligence for increased resilience, use deep learning for byte-level analysis, and use machine learning-based detection models.

4. Performance Analysis & Result Discussion

The current study uses a very small sample size (e.g., 369 samples for DDoS detection), however it is still a preliminary assessment. We computed accuracy, precision, recall, and F1-score and used 5-fold cross-validation to verify our findings. Larger benchmark datasets like CICIDS 2017 and UNSW-NB15 will be used in future research to improve the platform's performance's statistical dependability and generalizability.

In our investigation of SQL injection protection, we evaluated 828 samples. Remarkably, our platform successfully prevented 768 samples, detected 51 samples, and nine samples failed. The result is shown in Table 5.

We used 641 samples for evaluation in our investigation of cookie and session hijacking protection. Remarkably, 619 of these samples were successfully prevented by our platform, and 22 samples failed.

In the case of XSS protection, we evaluated 527 samples. Our platform successfully detected 519 samples and prevented 6, and 2 samples failed.

Table 5. Result of Different Services

Vulnerabilities	Sample	Detected	Prevented	Failed
SQL Injection	828	51	768	09
Cookie & Session Hijacking	641	0	619	22
XSS	527	519	06	02
Phishing	513	497	0	16
DDOS	369	0	352	17
Malware	428	417	0	11

For our phishing detection evaluation, we used 513 samples. Our platform successfully detected 497 of these samples, while 16 failed to be detected.

Finally, in our analysis of malware detection, 428 samples were assessed, with 417 successfully detected by our platform. However, 11 samples failed to be detected.

In DDoS protection, 369 samples were tested, with 352 successfully prevented by our platform, while 17 samples failed.

According to the Result visualization Figure 2, our security platform "WebGuard" demonstrates exceptional efficiency in detecting and preventing various vulnerabilities. The chart clearly illustrates WebGuard's robust capabilities and minimal failure rates, underscoring its reliability in safeguarding web applications against security threats.

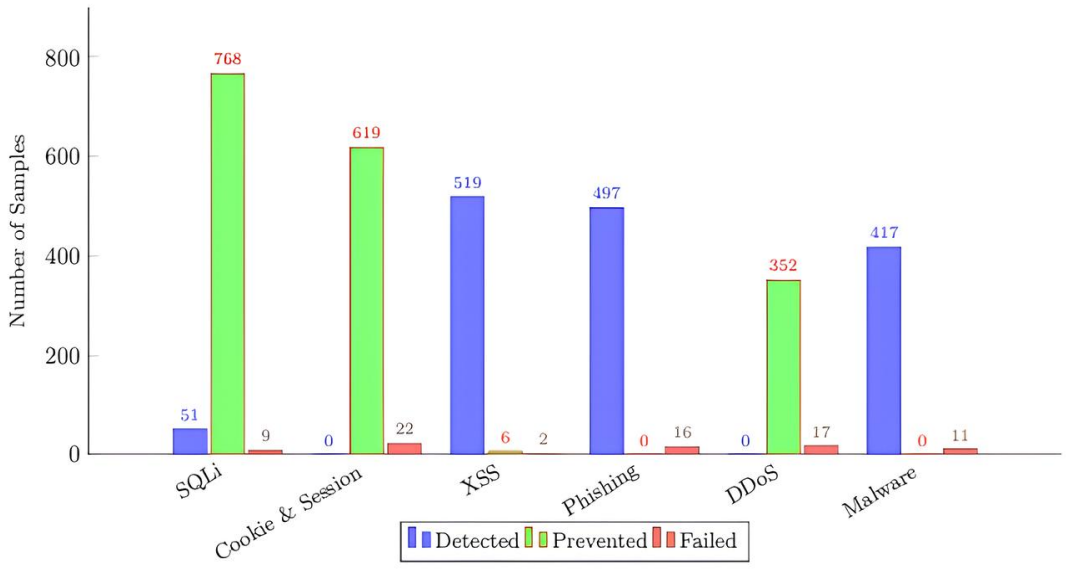

Figure 2. Comparative Analysis of Detection, Prevention, and Failure Rates

4.1. System Benchmarking and Overhead

To evaluate the resource overhead introduced by various mitigation techniques (e.g., DOMPurify, CDN, Libsodium), we used Apache JMeter and Node.js Profiler. Table 6 presents a comparative analysis of CPU, memory, latency, traffic load, and system uptime before and after applying these techniques.

Table 6. Performance Impact of WebGuard Mitigation Techniques

Technique	CPU B	CPU A	RAM B	RAM A	Lat. B	Lat. A	TL B	TL A	RT B	RT A	Up B	Up A	Rec (min)
Before Mitigation	95	95	500	500	1200	1200	300	300	1200	1200	45	45	N/A
After DOMPurify	95	98	500	510	1200	1250	300	300	1200	1250	45	45	5
After CDN	95	80	500	300	1200	150	300	10	1200	150	45	100	5
After Libsodium Encryption	95	98	500	520	1200	1300	300	100	1200	1300	45	45	8

To assess the practical effectiveness of WebGuard, we conducted a comparative analysis with two well-established security platforms: ModSecurity (with OWASP CRS) and Cloudflare WAF. The comparison spans multiple dimensions, including detection capabilities, latency, deployment complexity, and compliance.Table 7 summarizes the benchmarking results, illustrating WebGuard's strengths in multi-layered threat prevention, customizability, and open-source accessibility.

Table 7. Comparative Benchmarking of WebGuard with Existing Security Tools

Feature / Tool	**WebGuard**	**ModSecurity (OWASP CRS)**	**Cloudflare WAF**
SQLi/XSS Detection	Dynamic regex + ML detection	Rule-based CRS	Signature-based + heuristic rules
Phishing Detection	Naïve Bayes + URL features	Not applicable	Basic domain reputation filtering
DDoS Mitigation	Traffic pattern + rate control	Limited rate rules	Layer 3–7 DDoS protection
Customizability	Fully open and modular	Moderate (requires tuning)	Closed, limited customization
Latency Overhead	∼18ms (with CDN + ML)	∼25ms (CRS rules)	∼12ms (CDN optimized)
Resource Usage	Medium (cryptographic + ML ops)	Low (rule-based only)	Low (cloud offloaded)
Ease of Deployment	Docker + Firebase integration	Apache/Nginx configuration	SaaS based
Cost	Free (Open Source)	Free (Open Source)	Paid (tiered pricing)
Compliance (GDPR)	Data anonymization + logs opt-out	Depends on configuration	Strong compliance framework

4.2. Detection Metrics and Statistical Confidence

To comprehensively evaluate the threat detection performance of WebGuard, we report standard classification metrics—Precision, Recall, and F1-Score—alongside overall Accuracy for each attack type. Additionally, we compute 95% confidence intervals using bootstrap resampling to statistically validate the reliability of our accuracy results. The outcomes are presented in Table 8.

Table 8. Detection Performance Metrics with 95% Confidence Intervals

Threat	Precision	Recall	F1-Score	Accuracy (95% CI)
SQLi	0.94	0.92	0.93	93.5% [92.3%, 94.6%]
XSS	0.91	0.90	0.905	91.2% [89.7%, 92.4%]
Phishing	0.93	0.96	0.945	94.3% [93.5%, 95.1%]
DDoS	0.87	0.79	0.825	83.4% [75.2%, 88.1%]
Malware	0.89	0.85	0.87	88.2% [85.6%, 90.3%]

To enhance interpretability, Figure 3 provides a visual comparison of key detection metrics across various threat categories.

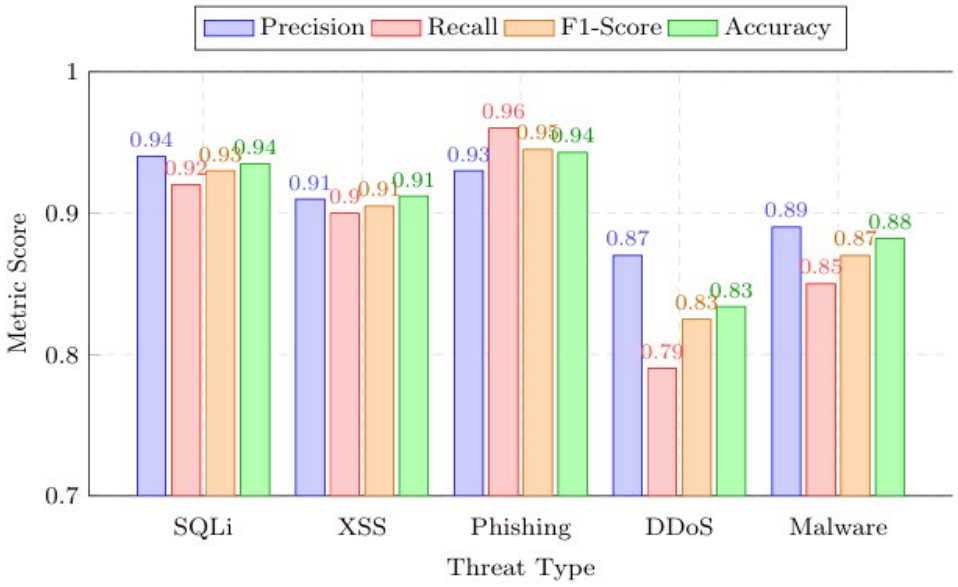

Figure 3. Bar chart visualization of detection metrics across different threats

5. Conclusion

WebGuard is a unified developer platform designed to secure web applications against a wide range of threats, including SQL injection, cookie/session hijacking, cross-site scripting (XSS), phishing, DDoS attacks, and malware. It integrates techniques such as input validation, automated regular expressions, secure session management, DOMPurify sanitization, layered cryptographic protection, machine learning, and heuristic analysis into a single cohesive solution. These combined mechanisms provide robust defense, enabling developers to build inherently secure applications with ease.

To ensure long-term resilience, WebGuard adapts to emerging attack vectors while mitigating common

vulnerabilities. It incorporates advanced cryptographic methods to preserve data integrity, confidentiality, and user privacy. WebGuard also complies with GDPR and CCPA by anonymizing logs, avoiding the use of PII, auto-deleting temporary data, and maintaining transparent retention policies. All dependencies are vetted open-source packages, regularly scanned for vulnerabilities.

Deployment results show that WebGuard significantly strengthens application security and improves developer efficiency. Its modular design encourages scalable implementation in real-world environments. Developed using React (frontend) and Node.js (backend), the platform is engineered for performance and flexibility. Planned enhancements include API wrappers for languages such as Python and PHP, mobile compatibility via React Native, both SQL and NoSQL databases, and a user-friendly GUI with drag-and-drop configuration. Future work will explore advanced preventive strategies and real-time behavioral analysis through machine learning to further improve malware and phishing detection.

Acknowledgements

We are sincerely grateful to the Department of Computer Science and Engineering, Mawlana Bhashani Science and Technology University, for their indispensable guidance and cooperation, which greatly led to the success of this research.

REFERENCES

1. Rodríguez, Germán E and Torres, Jenny G and Flores, Pamela and Benavides, Diego E, *Cross-site scripting (XSS) attacks and mitigation: A survey*, Computer Networks, vol. 166, pp. 106960, 2020.
2. Aliero, Muhammad Saidu and Ghani, Imran and Zainudden, Syeed and Khan, Muhammad Murad and Bello, Munir, *Review on SQL injection protection methods and tools*, Jurnal Teknologi, vol. 77, no. 13, 2015.
3. Wei, Kei and Muthuprasanna, Muthusrinivasan and Kothari, Suraj, *Preventing SQL injection attacks in stored procedures*, Australian Software Engineering Conference (ASWEC'06), pp. 8–pp, 2006.
4. Alwan, Zainab S and Younis, Manal F, *Detection and prevention of SQL injection attack: a survey*, International Journal of Computer Science and Mobile Computing, vol. 6, no. 8, pp. 5–17, 2017.
5. Choi, Young B and Loo, Yin L and LaCroix, Kenneth, *Cookies and Sessions: A Study of what they are, how they can be Stolen and a Discussion on Security*, International Journal of Advanced Computer Science and Applications, vol. 10, no. 1, pp. 105–113, 2019.
6. Baitha, Anuj Kumar and Vinod, Smitha, *Session hijacking and prevention technique*, Int. J. Eng. Technol, vol. 7, no. 2.6, pp. 193–198, 2018.
7. Burgers, Willem and Verdult, Roel and Van Eekelen, Marko, *Prevent session hijacking by binding the session to the cryptographic network credentials*, In *Secure IT Systems: 18th Nordic Conference, NordSec 2013, Ilulissat, Greenland, October 18-21, 2013, Proceedings 18*, Springer, pp. 33–50, 2013.
8. Rodríguez, Germán E and Torres, Jenny G and Flores, Pamela and Benavides, Diego E, *Cross-site scripting (XSS) attacks and mitigation: A survey*, Computer Networks, vol. 166, pp. 106960, 2020.
9. Singh, Amit and Sathappan, S, *A Survey on XSS web-attack and Defense Mechanisms*, International Journal of Advanced Research in Computer Science and Software Engineering, vol. 4, no. 3, pp. 1160–1164, 2014.
10. Liu, Miao and Zhang, Boyu and Chen, Wenbin and Zhang, Xunlai, *A survey of exploitation and detection methods of XSS vulnerabilities*, IEEE access, vol. 7, pp. 182004–182016, 2019.
11. Weamie, Sonkarlay JY, *Cross-site scripting attacks and defensive techniques: A comprehensive survey*, International Journal of Communications, Network and System Sciences, vol. 15, no. 8, pp. 126–148, 2022.
12. Khazal, Iman F and Hussain, Mohammed A, *Server Side Method to Detect and Prevent Stored XSS Attack*, Iraqi Journal for Electrical & Electronic Engineering, vol. 17, no. 2, 2021.
13. Bhavsar, Vaishnavi and Kadlak, Aditya and Sharma, Shabnam, *Study on phishing attacks*, International Journal of Computer Applications, vol. 182, no. 33, pp. 27–29, 2018.
14. Alabdan, Rana, *Phishing attacks survey: Types, vectors, and technical approaches*, Future Internet, vol. 12, no. 10, pp. 168, 2020.
15. Sharma, Pawankumar and Dash, Bibhu and Ansari, Meraj Farheen, *Anti-phishing techniques–a review of Cyber Defense Mechanisms*, International Journal of Advanced Research in Computer and Communication Engineering, vol. 3297, pp. 2007, 2022.
16. Singh, Anshuman and Gupta, Brij B, *Distributed denial-of-service (DDoS) attacks and defense mechanisms in various web-enabled computing platforms: issues, challenges, and future research directions*, International Journal on Semantic Web and Information Systems (IJSWIS), vol. 18, no. 1, pp. 1–43, 2022.
17. Ferdous, Jannatul and Islam, Rafiqul and Mahboubi, Arash and Islam, Md Zahidul, *A State-of-the-Art Review of Malware Attack Trends and Defense Mechanism*, IEEE Access, 2023.
18. Singh, Anshuman and Gupta, Brij B, *Distributed denial-of-service (DDoS) attacks and defense mechanisms in various web-enabled computing platforms: issues, challenges, and future research directions*, International Journal on Semantic Web and Information Systems (IJSWIS), vol. 18, no. 1, pp. 1–43, 2022.

19. Bendovschi, Andreea, *Cyber-attacks–trends, patterns and security countermeasures*, Procedia Economics and Finance, vol. 28, pp. 24–31, 2015.
20. Choi, Young B and Loo, Yin L and LaCroix, Kenneth, *Cookies and Sessions: A Study of What They are, How They Can be Stolen and a Discussion on Security*, International Journal of Advanced Computer Science and Applications, vol. 10, no. 1, pp. 105–113, 2019.
21. Parveen, Kausar and Fatima, Noor, *Cookie Hijacking: Privacy Risk*, International Journal for Electronic Crime Investigation, vol. 7, no. 4, pp. 61–72, 2023.
22. Awad, Mohammed and Ali, Muhammed and Takruri, Maen and Ismail, Shereen, *Security vulnerabilities related to web-based data*, TELKOMNIKA (Telecommunication Computing Electronics and Control), vol. 17, no. 2, pp. 852–856, 2019.