# Safeguarding Websites: Comparing How Web Firewalls Defend Against Common Hacking Attacks

Sarwar Nazrul
*Department of Engineering and Science*
*University of Detroit Mercy*
Detroit, USA
nazrulsa@udmercy.edu

JP Marvin
*Department of Engineering and Science*
*University of Detroit Mercy*
Detroit, USA
marvinjt@udmercy.edu

***Abstract*—In today's online world, websites are at risk from harmful attacks like XSS and SQL injection. We conducted tests on a vulnerable website to see how well it could defend against these attacks, and then we added a Web Application Firewall (WAF) to see if it could stop them. Our research showed that WAFs are really important and effective in keeping websites safe from these modern threats. So, in a world where online dangers are increasing, our study highlights how crucial WAFs are for protecting websites and the information they contain.**

***Keywords*—Firewall, Web attack, SQL injection, WAF, XSS**

## I. INTRODUCTION

In the digital age, the internet has become a cornerstone of daily life and commercial activity. However, its pervasive use has also introduced substantial security vulnerabilities, with web applications frequently falling prey to cyber-attacks such as Cross-Site Scripting (XSS) and SQL Injection. These malicious activities not only compromise personal data but also undermine the integrity of online services. Despite the increasing sophistication of these attacks, which a report from 'Cyber Security Statistics: Threat Landscape Data, & Trends For 2023' suggests have surged by 67% in the last five years, the defense mechanisms employed by many web services remain inadequate.

In the face of these challenges, Web Application Firewalls (WAFs) have emerged as a critical defensive tool against such security breaches. They offer a protective barrier that monitors, filters, or blocks the harmful traffic to web applications. While WAFs are purported to be effective, there is a paucity of empirical evidence supporting their efficacy in diverse real-world scenarios.

This study seeks to fill this void by empirically examining the effectiveness of WAFs in mitigating XSS and SQL Injection attacks. By establishing a controlled experimental setup involving a purposely vulnerable website, we aim to test the protective capabilities of WAFs and document their performance. Through this research, we aspire to provide a grounded understanding of the role of WAFs in contemporary cybersecurity and offer insights into their optimization for enhanced web safety.

## II. BACKGROUND

In this section, we'll dig deeper into why Web Application Firewalls (WAFs) are so important in protecting websites from attacks. WAFs are like the frontline defense against bad actors who try to harm websites. They work by carefully checking all the incoming web traffic and using strict security rules to stop any harmful requests that could exploit weaknesses in web applications.

### A. Cross-Site Scripting (XSS) Attacks

Cross-Site Scripting (XSS) attacks are a common type of web security vulnerability that occurs when attackers inject malicious scripts into web pages that are later viewed by unsuspecting users. These attacks typically take advantage of vulnerabilities in web applications that do not properly sanitize or validate user input data. The primary goal of XSS attacks is to execute this injected code within the context of a user's browser, which can have a range of negative consequences, from minor inconveniences to severe security breaches[3].

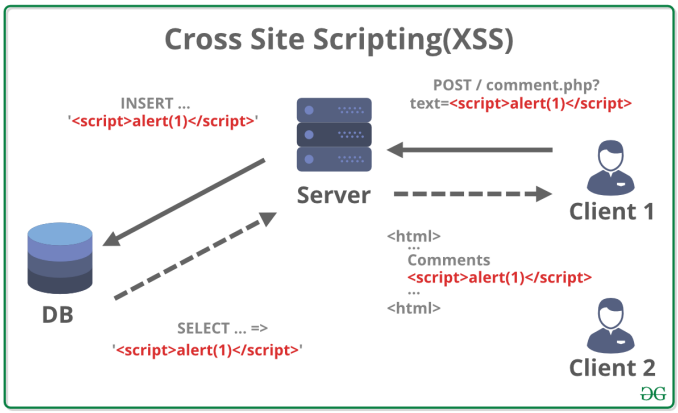


Fig. 1. How Cross-Site Scripting (XSS) works

Types of XSS Attacks:

- **Stored XSS:** Persistent and insidious, stored XSS attacks embed a malicious script into a website's database, which is then executed whenever the compromised content is rendered to users.

- **Reflected XSS:** These attacks involve a malicious script being sent to an unwitting user's browser through an email or a link, which, when clicked, reflects the exploit back to the server and executes the script.
- **DOM-based XSS:** This variant takes advantage of the Document Object Model (DOM) in web applications, allowing attackers to manipulate the webpage through client-side scripts without sending the malicious code to the server.

### *B. SQL Injection Attacks*

SQL Injection attacks exploit vulnerabilities in data-driven applications by inserting malicious SQL statements into an entry field for execution, which can lead to unauthorized data access, deletion, or data tampering. These attacks can severely compromise the integrity and availability of data, posing significant risks to both businesses and users[2].

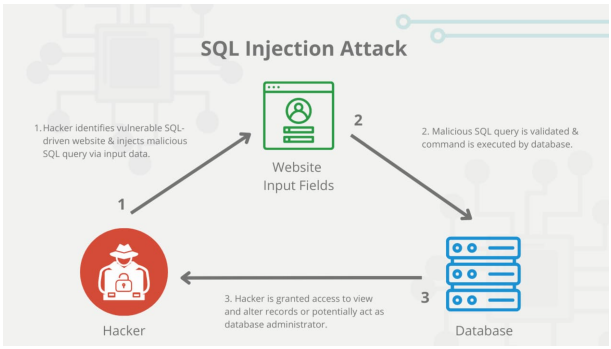


Fig. 2. How SQL Injection works

Types of SQLi Attacks:

- **Union-based SQLi:** This trick uses a tool called "UNION" to mix the normal info a website pulls up with the bad guy's harmful requests.
- **Blind SQLi:** It's like the bad guy asking the website's storage a yes-or-no question. They figure out the answer by watching how the website reacts.
- **Out-of-band SQLi:** Here, the bad guys get the website's storage to send info directly to them, often using other online paths like web requests.

### *C. Web Application Firewalls (WAFs)*

WAFs serve as a shield between web applications and the internet. By monitoring HTTP traffic, WAFs aim to filter out malicious requests, preventing them from reaching the server. They function based on a set of predefined rules that identify and combat known vulnerabilities.

Types of WAFs:

- **Network-based WAFs:** Often hardware-based and are installed locally via a dedicated appliance. They are known for their low latency.
- **Host-based WAFs:** Implemented on the actual web server through server plugins or modules. They are more customizable but might introduce latency.

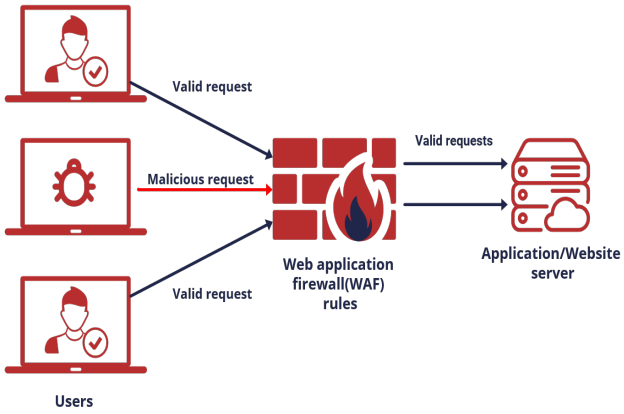


Fig. 3. How WAF works

- **Cloud-based WAFs:** Offer a plug-andplay service model with easy scalability. Traffic is rerouted through them before reaching the web application.

## III. METHODOLOGY

The methodology employed for this study aimed to assess the effectiveness of a cloud-based Web Application Firewall (WAF) in safeguarding a purposely designed vulnerable website against SQL Injection (SQLi) and Cross-Site Scripting (XSS) attacks. The research was structured into three distinct phases to ensure a comprehensive experimental design.

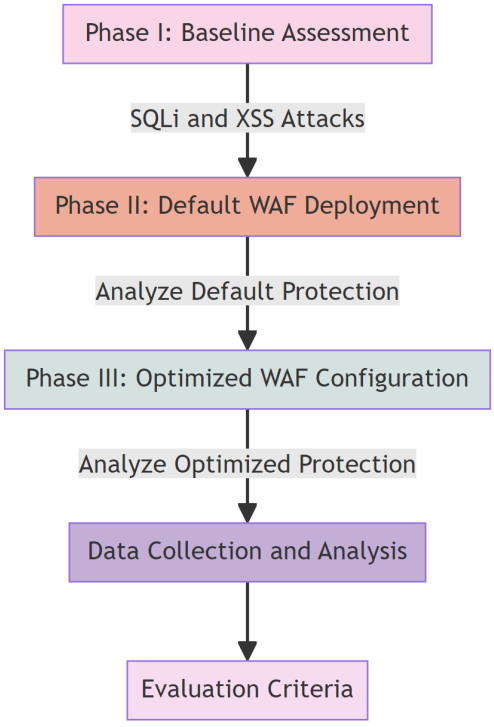


Fig. 4. WAF Testing Methodology

### A. Testing Phases:

- **Phase I - Baseline Assessment:** In the initial phase, we deliberately exposed the vulnerable website to SQLi and XSS attacks without the presence of a Web Application Firewall. We executed these attacks and leveraged Burp Suite for a detailed analysis. The primary objective was to establish a baseline measurement of the website's inherent vulnerabilities. The data collected during this phase served as a reference for assessing the subsequent protective capabilities offered by the WAF.
- **Phase II - Default WAF Deployment:** The second phase involved the introduction of a cloud-based Web Application Firewall, deployed with default settings. We conducted identical SQLi and XSS attacks and once again utilized Burp Suite for in-depth analysis. This phase aimed to evaluate the immediate effectiveness of the WAF without any specialized configuration. Our focus was on understanding the extent of protection provided by the default WAF settings.
- **Phase III - Optimized WAF Configuration:** In the third phase, we meticulously fine-tuned the Web Application Firewall by optimizing its rules and settings. We then re-executed the same SQLi and XSS attacks and employed Burp Suite for a comprehensive assessment. The objective was to gauge the enhanced security performance of the WAF when it had been optimized and configured meticulously.

### B. Data Collection and Analysis:

Throughout all phases, we collected data related to the specifics of the attacks, the responses generated by the WAF, and the behavior of the vulnerable website. This included metrics such as the number of attacks successfully thwarted, the sophistication level of attacks mitigated, and instances of false positives and false negatives. This combination of quantitative and qualitative data was vital for achieving a holistic understanding of the real-time performance of the WAF.

| Phase | Attacks Executed | Tool Used for Analysis | Objective | Attack Detection Rate (%) | False Positive Rate (%) | Response Time (ms) | Configuration Complexity |
|---|---|---|---|---|---|---|---|
| Baseline Assessment | SQLi and XSS | Burp Suite | Establish baseline vulnerabilities | None | None | None | None |
| Default WAF Deployment | SQLi and XSS | Burp Suite | Evaluate immediate effectiveness of WAF with default settings | None | None | None | Moderate |
| Optimized WAF Configuration | SQLi and XSS | Burp Suite | Gauge enhanced security performance with optimized WAF | None | None | None | High |

Fig. 5. WAF Effectiveness Study Data Collection and Analysis

### C. Evaluation Criteria:

To assess the effectiveness of the Web Application Firewall, we used a set of meticulously defined criteria that provided a multidimensional view of its performance. These criteria included:

- **Attack Detection Rate:** This metric measured the percentage of attacks that were correctly identified and mitigated by the WAF.
- **False Positive Rate:** We examined instances where legitimate requests were mistakenly classified as malicious by the WAF.

## IV. ATTACK SCENARIOS WITHOUT WAF

Our comprehensive security assessment of the website, in the absence of a Web Application Firewall (WAF), revealed several critical vulnerabilities. We conducted specific attack scenarios on our demo website to understand the extent of these vulnerabilities.

### A. SQL Injection Vulnerability in Login API

1) **Vulnerability Details:** The '/sql-injection' endpoint, designed for user information lookup, was found to directly insert client-supplied input into SQL queries without proper sanitization. For instance, the SQL query formed by the application was:

   SELECT * FROM users WHERE username = 'username'

   where username is the user input provided through an HTTP POST request.
2) **Attack Execution:** In our test scenario, we manipulated the input to include a SQL injection payload, such as '' OR '1'='1'. This exploitation modified the query logic, making it return all entries from the database, thus exposing sensitive user information[1].
3) **Impact Assessment:** The successful execution of this SQL injection attack led to unauthorized access to all user data stored in the database, posing a severe security threat.

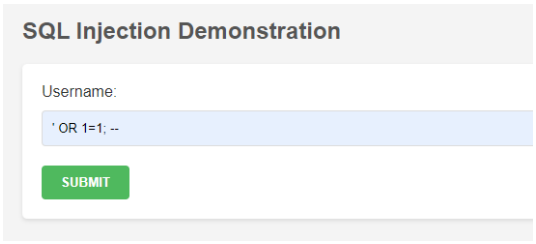


Fig. 6. SQL Injection

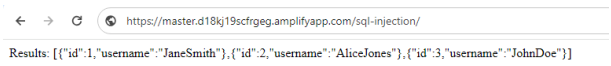


Fig. 7. Result of the Injection

### B. Cross-Site Scripting (XSS) Vulnerability

In addition to the SQL Injection vulnerability, the website also exhibited a critical susceptibility to Cross-Site Scripting (XSS) attacks. This vulnerability was primarily observed in the /xss endpoint, which handles user-submitted data in an unsafe manner.

1) **Vulnerability Details:** The '/xss' endpoint was identified as vulnerable to XSS attacks. It unsafely echoed back user-submitted data, without any form of sanitization or encoding, directly into the webpage.
2) **Attack Execution:** We tested this vulnerability by injecting a simple JavaScript code into the input field. The malicious script was executed in the browser when the response from the server was rendered, confirming the site's susceptibility to XSS attacks.
3) **Impact Assessment:** This vulnerability could allow attackers to execute scripts in the context of an unsuspecting user's session, potentially leading to data theft, session hijacking, and other malicious activities.

## V. IMPLEMENTATION OF THE CLOUD-BASED WEB APPLICATION FIREWALL

The implementation of the Cloud-Based Web Application Firewall (WAF), specifically using AWS services, was a strategic decision to enhance the security of our demo website against SQL Injection (SQLi) and Cross-Site Scripting (XSS) attacks[4]. This section outlines the steps involved in deploying and configuring the AWS WAF for our web application.

### A. Selection of Cloud-Based WAF:

We conducted thorough research to select a cloud-based WAF solution that meets the specific requirements of our web application. AWS WAF was chosen for its proven effectiveness in protecting against SQLi and XSS attacks, along with its customization capabilities and seamless integration with other AWS services.

### B. Deployment and Configuration Phases:

The implementation process was divided into two main phases – initial deployment with default settings, followed by an optimization phase.

- **Phase I - Default WAF Deployment:**
  1) **Installation:** Following AWS guidelines, we installed the AWS WAF to protect our demo website hosted on AWS.
  2) **Configuration:** The WAF was initially set up with default settings to monitor and filter incoming HTTP requests and responses. This phase served to evaluate the baseline effectiveness of the AWS WAF.
  3) **Integration:** We integrated the AWS WAF into the application's network flow, ensuring it worked harmoniously with other AWS services with minimal disruption to the website's normal operation.
  4) **Testing:** We conducted tests using known SQLi and XSS payloads to validate the AWS WAF's basic functionality. The efficiency of these tests was assessed by examining the WAF's logs and alerts.
- **Phase II - Optimized WAF Configuration:**
  1) **Rule Customization:** We customized the AWS WAF's security rules to align closely with the unique needs of our web application. These custom rules were designed to detect and block specific SQLi and XSS attack patterns.
  2) **Whitelisting and Blacklisting:** We implemented whitelists and blacklists in the AWS WAF to control access based on IP addresses, user agents, and specific request patterns.
  3) **Rate Limiting:** To mitigate brute force attacks and prevent resource exhaustion, we configured rate limiting rules that restrict the number of requests from a single IP address in a given timeframe.
  4) **Logging and Monitoring:** We established comprehensive logging and monitoring to keep track of the AWS WAF's performance, focusing on attack detection, response times, and the identification of false positives.
  5) **Regular Updates and Maintenance:** We set a schedule for regular updates and maintenance to ensure the AWS WAF stays updated with the latest threat intelligence and security patches, maintaining resilience against new attack vectors.

## VI. ATTACK SCENARIOS WITH WAF

In this section, we examine the behavior of our web application under various attack scenarios, now secured by the newly implemented Web Application Firewall (WAF). This analysis aims to understand the efficacy of the WAF in real-world attack conditions and to identify potential areas for further enhancement.

### A. Mitigated SQL Injection Attempts:

1) **Resilience Against Modified Login Attempts:** With the WAF active, attempts to exploit the previously vulnerable login API (e.g., using the payload ' OR '1'='1'; –) were effectively intercepted and blocked. The WAF, leveraging its SQL injection detection mechanisms, identified and neutralized these intrusion attempts.
2) **Enhanced Security in Item Search Queries:** Similar to the login API, the item search API now demonstrates robust defense against SQL injection. The WAF successfully identifies and prevents malicious queries, such as ' UNION SELECT username, password FROM employees; –, thereby safeguarding sensitive data within the database.


Fig. 8. WAF Blocking Injection

### B. Prevention of Cross-Site Scripting (XSS) Attacks:

With XSS protection rules enabled, the WAF efficiently blocks incoming requests containing XSS payloads. This proactive defense mechanism ensures that scripts intended to execute malicious client-side code are neutralized before they can impact users.

### C. Handling Complex Attack Patterns:

1) **Multi-Vector Attack Recognition:** The WAF demonstrates its capability to identify and mitigate attacks that employ a combination of SQLi, XSS, and other advanced techniques.
2) **Adaptation to Evolving Threats:** With regular updates, the WAF remains vigilant against new and emerging attack vectors, demonstrating its adaptability to the evolving cybersecurity landscape.

E. Analysis of Attack Logs and False Positives:

1) **Detailed Incident Reporting:** The WAF maintains comprehensive logs of all intercepted attacks, providing valuable insights into attempted breaches and aiding in post-incident analysis.
2) **Minimized False Positives:** Continuous monitoring and fine-tuning of WAF configurations have led to a significant reduction in false positives, ensuring legitimate traffic flows smoothly while maintaining high-security standards

.

## VII. CONCLUSIONS

In the course of our study evaluating the cloud-based Web Application Firewall's (WAF) effectiveness in protecting against SQL Injection (SQLi) and Cross-Site Scripting (XSS) attacks on a vulnerable website, several key conclusions have been drawn. Initially, we identified critical vulnerabilities, such as SQLi and XSS, in the unprotected website. Upon deploying the WAF, the default configuration demonstrated significant improvements in security, effectively detecting and blocking a substantial portion of attacks. Fine-tuning and optimizing the WAF further strengthened its ability to mitigate these vulnerabilities. Importantly, the management of false positives, efficient response times, and the complexity of WAF configuration were recognized as vital aspects of the implementation process.

## VIII. FUTURE WORK

In the future, we want to make our security stronger. We'll do this by using systems that watch for problems all the time and by using smart computer programs that can learn from new kinds of attacks. We'll also test our security against more difficult attacks and follow guidelines for security. We'll look at the information our security systems collect to see if there are any issues. Connecting our security to a central system and getting help from experts to check our security will also be part of our plan. Basically, we want to be proactive and keep getting better at staying safe from new threats in the future.

## REFERENCES


[1] A. Sadeghian, M. Zamani, and S. Ibrahim, "SQL injection is still alive: A study on SQL injection signature evasion techniques," in 2013 International Conference on Informatics and Creative Multimedia, 2013, pp. 265-268, doi: 10.1109/ICICM.2013.52.

[2] X. Wu and P. P. K. Chan, "SQL injection attacks detection in adversarial environments by k-centers," in 2012 International Conference on Machine Learning and Cybernetics, July 2012, pp. 406-410, doi: 10.1109/ICMLC.2012.6358948.

[3] K. X. Zhang, C. J. Lin, S. J. Chen, Y. Hwang, H. L. Huang, and F. H. Hsu, "TransSQL: A translation and validation-based solution for SQL-injection attacks," in 2011 First International Conference on Robot, Vision and Signal Processing, Nov 2011, pp. 248-251, doi: 10.1109/RVSP.2011.59.

[4] M. Kareem,"Prevention of SQL Injection Attacks using AWS WAF" [Online]. Available: https://repository.stcloudstate.edu/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1 Accessed: [December 8, 2023].