Computers & Security 160 (2026) 104714

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Full Length Article

An empirical study on the evaluation and enhancement of OWASP CRS (Core Rule Set) in ModSecurity

Anuvarshini MK, Kommuri Sai Suhitha Bala, Sri Sai Tanvi Sonti, Jevitha KP *

TIFAC-CORE in Cyber Security, Amrita School of Engineering, Coimbatore, Amrita Vishwa Vidyapeetham, India

ARTICLE INFO

Keywords:
Web Application Firewall (WAF)
ModSecurity
OWASP CRS (Core Rule Set)
WAF Evaluation
WAF Rule Enhancement

ABSTRACT

The effectiveness of a Web Application Firewall is determined by their ability to accurately detect and block malicious payloads while allowing legitimate traffic without any interference. This research evaluates the effectiveness of the popular open-source OWASP CRS (Core Rule Set) with the ModSecurity web application firewall. The study analyzes the impact on performance metrics under different configurations of the OWASP CRS. This study also aims to evaluate the detection capabilities of the WAF in its strict configuration to uncover gaps in the existing rule coverage. The identified gaps were then improved through the creation of 146 new custom rules that were designed to recognize attack payloads that managed to evade all rules in the OWASP CRS. The implemented custom rules, which were developed in accordance with the gaps identified during the test, improved the detection precision from 60.54% to 97.46 % with no increase in false positives within our controlled test environment, thereby incrementally strengthening the security of the rule set by detecting threats that had previously escaped notice.

1. Introduction

Web applications are now an essential part of our everyday life, providing seamless access to various services such as banking, healthcare, e-commerce, communication,etc., which otherwise would be possible only through direct interaction. Their convenience and accessibility led to widespread usage that has not only transformed professional and personal interactions but has also led to an increasing dependence on digital platforms for critical operations. But with convenience comes risk; every time we use an online service, we share a piece of information such as our details, conversations, financial information, etc. If not properly protected, this information can be misused, causing security issues (Aïmeur, 2014).

The seamless integration of essential services into daily routines has been made possible by the advancement of web application architecture. In the beginning, web applications followed a simple client-server model where the users connected to a centralized server that handles requests and provided either static or dynamically created responses (Nyabuto et al., 2024). This architecture maintained a separation between the user and the server where the interactions were limited between them, and the security measures were also relatively simple as access was controlled at a single server level, and threats were primarily focused on direct attacks on the server (Sharma et al., 2019). However, with the growing demand from users and businesses for more functionality, accessibility, real-time interactions and updates, web applications have become highly sophisticated, introducing multiple layers to handle user interactions, process business logic, storage and retrieval of application data, security measures like authentication (Naresh and Jevitha, 2020) and authorization (Sharma and Jevitha, 2023), and external service integrations. These layers enable web applications to become more capable to provide services, but has also made them more complex and introduced new vulnerabilities, as each layer and external service could become potential new attack surfaces (Mouli and Jevitha, 2016).

Modern web applications aren't isolated systems anymore but they are built using the microservices architecture where an application is divided into small independent services. This modern architecture uses APIs to facilitate communication between services, allowing applications to extract data from external providers, integrate with third-party authentication services, and connect with other business systems. While APIs enhance functionality and simplify development, they also introduce new risks where each exposed API, if misconfigured or unprotected, can act as an entry point which can provide attackers with

* Corresponding author.
E-mail addresses: anuvarshini954@gmail.com (A. MK), suhithabala@gmail.com (K.S.S. Bala), sontisrisaitanvi@gmail.com (S.S.T. Sonti), kp_jevitha@cb.amrita.edu (J. KP).

https://doi.org/10.1016/j.cose.2025.104714
Received 30 April 2025; Received in revised form 23 September 2025; Accepted 16 October 2025
Available online 20 October 2025
0167-4048/© 2025 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

unauthorized access (Díaz-Rojas et al., 2021).

Additionally, modern web applications rely heavily on JavaScript frameworks so that sophisticated interactions can take place within the user's browser and replies can be generated faster by minimizing the number of requests sent to the server. Given that a lot of data and business logic is processed within the browser, securing the server alone is no longer adequate.

The greater burden is now placed on the client side to reduce the load on the server giving scope for attacks on the client's side (Vishnu and Jevitha, 2014). As these vulnerabilities are not only limited to the server side operations, security issues could arise from either side which broadens the attack surface.

Therefore, to enhance security, proactive efforts are required to make sure no aspect of the application is unattended by seeking to resolve both server and client side security issues. It is now evident that defending intricate web structures against emerging risks requires a multilayered security strategy. Web application firewalls (Razzaq et al., 2013) have emerged as a key component in preventing modern web attacks. As with most cybersecurity tools, WAFs work on a defensive model - they check the server's traffic in real time for potentially unsafe patterns, and if any are detected, it is blocked immediately.

WAFs not only protect against server side vulnerabilities such as SQL injection, but they also offer good security to client side vulnerabilities such as cross site scripting that attacks the user's browser instead of server (Ravindran et al., 2023). WAFs can keep such vulnerabilities at bay by filtering and analyzing the data from possibly malicious sources before it hits the server or the client. This kind of protection is very much needed in the current risk environment, where because of intricate web architecture, vulnerabilities are being exploited at both server and client side which demands for a strong solution that can secure both the server and users. This is where WAFs come into play to mitigate both types of vulnerabilities, minimizing the risk of security incidents.

There are multiple WAF solutions, but rule-based WAFs are deployed extensively because they are transparent about detection mechanisms and have controllable filtering policy control, allowing administrators to specify exact detection logic specific to their environment. ModSecurity (OWASP ModSecurity Team 2025) is a well-known open source rule-based WAF solution which is extensively used for its flexibility. It is a module for web servers like Apache (Apache 2025), NGINX (NGINX 2025) and IIS (IIS 2025), where security teams create custom rules for their threat environment. Modsecurity's greatest strength is that it can be integrated with the OWASP CRS (CRS) (OWASP CRS Team 2025), where the rules are specially crafted by the security experts of the OWASP organisation (OWASP Team 2025) to identify and counter a variety of web based attacks.

The OWASP CRS follows a negative security model, where known malicious patterns are used to identify and block potentially harmful requests. Hence, attackers are always testing new ways to hide using obfuscation, payload alteration, or encoding methods to evade these filters. While all HTTP requests undergo standard encoding during transmission, some evasion attempts involve additional or layered encodings, such as double or triple URL encoding or Base64. CRS does not perform extra decoding by default, as these cases are uncommon in typical applications and the rules are designed to remain generic. The OWASP CRS also provides an optional auto-decoding plugin for deployments that require additional decoding. Regular evaluation and rule revisions are therefore crucial (Kollepalli et al., 2024). Since a WAF can only be as effective as its adaptability, this study emphasizes how important it is to test detection rules often in order to keep up with new threats. The OWASP CRS may continue to be a potent defense against complex web threats by regularly analyzing the bypass attempts and enhancing the rule coverage.

This research focuses on evaluating and improving the detection capability of OWASP CRS integrated with the ModSecurity. An in-depth analysis is done to determine the impact of different configuration parameters of the anomaly scoring mode on the detection rate and permissiveness with the OWASP CRS. On the basis of this analysis, particular emphasis was placed on the false negative payloads that were able to successfully bypass the detection of the WAF. Since these bypasses are weaknesses in the current rule set, the discovery of these evaded payloads was prioritized because they had a strong potential to exploit real-world attacks. In the analysis of bypass patterns, we implemented new security rules to counter attack patterns that bypassed default detection techniques. This highlights the need for regular testing and improving the WAF rules to be ahead of evolving attack strategies. Our work should be viewed as an incremental strengthening of CRS which fills specific gaps identified during testing, rather than a broad general enhancement of WAF security. Web security is a continuous process, even if the once-avoidable threats were effectively banned by the new rules.

The remainder of this paper is structured as follows: Section 2 explores related work, summarizing significant studies in the field and highlighting the gaps that necessitate this research. Section 3 discusses technical concepts related to ModSecurity and the OWASP CRS. Section 4 describes the methodology for setup and configuration, analysis, and the inclusion of new security rules. Section 5 presents the results and discussion. Finally, Section 6 concludes the study by summarizing the key findings and outlining potential directions for future research.

2. Related work

In this section, we discuss the studies on Web Application Firewall with emphasis on ModSecurity and OWASP CRS. From the survey, ModSecurity can detect and block web attacks such as SQL Injection, Cross-Site Scripting (XSS), Command Injection. Our discussion also covers research that touches upon web application firewall, paranoia levels (described in detail in Section 3.3), performance trade-offs, and bypass techniques. In this overview of these researches, we emphasise broad areas where WAF security measures can be optimized further.

T. Rahmawati et al. (Rahmawati et al., 2023) aims to enhance web security by developing a proxy-based Web Application Firewall (WAF) combined with Security Information and Event Management (SIEM) systems. The aim is to avoid cyber-attacks by detecting the fundamental vulnerabilities enumerated by OWASP as Cross-Site Scripting (XSS), SQL Injection (SQLi), and Local File Inclusion (LFI). This study suggests a WordPress plugin to consolidate and visualize security information from the SIEM and WAF, with the use of Damn Vulnerable Web Application (DVWA) as the test environment. The results confirm the effectiveness of the suggested system, with the WAF providing a detection efficiency of 100 % for XSS, 97 % for SQLi, and 74 % for LFI attacks. The SIEM was effective in detecting ping attacks, DDoS attempts, and brute-force incursions, storing the information safely with Snorby (Snorby 2025) prior to encoding it into JSON format for visualization using the plugin. Performance tests of the system recorded an astonishing surge in the utilization of resources, with CPU utilization rising from 9 % to 50 % and memory utilization from 3.6 % to 59 % upon attacks. However, our research varies in a number of other important areas. We conduct a comprehensive study of rule improvement, developing rules to match patterns in payloads that bypass existing OWASP guidelines successfully. Our method explores paranoia levels and anomaly scores comprehensively to optimize the performance of the WAF, while this article focuses on the integration of SIEM and the implementation of WordPress plugins. We also employ GoTestWAF, a security test automation tool, to comprehensively test the threat detection capabilities in different environments. Our study provides an in-depth analysis of the enhancement of WAF security using baseline detection efficiency and customized implementation of rules.

R. A. Muzaki et al. (Muzaki et al., 2020) focuses on web application security with a Web Application Firewall (WAF) using ModSecurity alongside the Reverse Proxy method. The study targets the prevention of cyberattacks on web applications, including Cross-Site Scripting (XSS), SQL Injection, and Web Scanning for Unauthorised Vulnerability. The

study employs a method where ModSecurity is installed on a Reverse Proxy server and the OWASP CRS is used to identify and block malicious requests before reaching the web server. The results of the study show that the suggested security implementation considerably minimizes threats to web applications. The performance was compared in two scenarios: one without the WAF and the other with the WAF. Without the WAF, all three attacks succeeded, compromising the web application. Nonetheless, the use of ModSecurity and the Reverse Proxy effectively mitigated all three attacks. The XSS attack was thwarted by ModSecurity, blocking the execution of harmful scripts; the system responded with appropriate error codes, such as 403 Forbidden, and attack logs confirmed that ModSecurity correctly identified and rejected malicious requests based on predefined rules from the OWASP CRS. This confirms that combining ModSecurity with a Reverse Proxy is an effective strategy for safeguarding web applications against prevalent threats, offering a low-cost yet robust security enhancement. While this study focused on three prevalent attack types, our research expanded the scope by testing the WAF against a broader range of attacks and evaluating its performance across different configurations such as paranoia levels and threshold value. This approach allowed us to assess the WAF's effectiveness and identify areas for improvement.

K. Nagendran et al. (Nagendran et al., 2020) examine the techniques used by attackers to evade Web Application Firewalls (WAFs), detailing their intrinsic vulnerabilities and shortcomings. The aim of this research is to alert security professionals to these evasion methods, hence allowing them to build stronger firewall defenses. The examination addresses various evasion techniques used to neutralize attacks like SQL Injection (SQLi), Cross-Site Request Forgery (CSRF), Cross-Site Scripting (XSS), File Inclusion and Remote Code Execution (RCE). Attackers use evasion techniques to evade Web Application Firewalls (WAFs) using techniques like case manipulation, URL encoding, keyword filter evasion, buffer overflow, and payload obfuscation. The research also discusses negative (blacklist-based), positive (whitelist-based), and mixed firewall frameworks, adding that each WAF model requires different methodologies for potential evasion. Despite Web Application Firewalls (WAFs) being an essential element in security, they are not immune to exploitation. The majority rely on obsolete filtering techniques that can be circumvented by adversaries employing encoding tricks, character substitutions, or injection modifications. The research indicates that no WAF model is completely successful and that attackers continually adapt their strategies. Hence, this study highlights the importance of regularly testing WAF filters with various attack techniques to strengthen the security posture.

B. I. Mukhtar et al. (Mukhtar and Azer, 2020) examines the SQL Injection (SQLI) attacks, their detection and prevention methods, and evaluates the efficacy of the Modsecurity Web Application Firewall (WAF) in mitigating SQLI attacks. The DVWA web application was tested for SQL injection vulnerabilities via SQLMAP (SQLmap 2025). ModSecurity WAF was configured in embedded mode on the Apache2 web server. SQLMAP was utilised again with various evasion techniques to assess Modsecurity's efficacy in blocking SQL injection attempts. The testing phases comprised: Basic SQL Injection testing (without Web Application Firewall), SQL Injection testing post-Modsecurity installation, and Complex SQL Injection testing utilising evasion techniques. Preliminary assessments (without Modsecurity) confirmed SQL injection vulnerabilities in the DVWA application. Notwithstanding the utilisation of advanced SQL injection evasion techniques, the Web Application Firewall remained resilient, rejecting all attempted injections. The efficacy of Modsecurity depends on the regular upgrading of its rule sets to accommodate evolving SQLI techniques. The research exclusively evaluated SQL injection attacks and did not assess the firewall's efficacy against other web-based threats. Testing was conducted in a laboratory setting; more realistic scenarios with increased traffic were not evaluated. The article exclusively assesses SQL injection (SQLi) attacks, neglecting other significant web vulnerabilities such as Cross-Site Scripting (XSS), Remote Code Execution (RCE), NoSQL Injection, and API-based attacks. The paper fails to examine how attackers circumvent ModSecurity's security rules, rendering it inadequate for actual security risks. Attackers perpetually advance evasion strategies, necessitating the improvement of special rules.

J. J. Singh et al. (Singh et al., 2018) examines the impact of different paranoia levels in ModSecurity Web Application Firewall (WAF) on its effectiveness in detecting and preventing web-based threats. The study aims to evaluate if higher levels of paranoia improve security without significantly compromising system performance. A controlled testbed environment was created utilising vulnerable online apps. ModSecurity Web Application Firewall was installed in conjunction with the OWASP CRS. Different levels of paranoia (PL1 to PL4) were established. Various web attacks, including SQL Injection (SQLI), Cross-Site Scripting (XSS), and Command Injection, were conducted. Recorded attack attempts were analysed based on detection efficacy. The influence of paranoid levels on response time, and false positives was evaluated. Elevated paranoia levels (PL3 and PL4) successfully thwarted a greater number of attempts, whereas lower paranoia levels (PL1and PL2) failed to identify some sophisticated attack routes. Increased paranoia resulted in a rise in false positives, obstructing authentic requests and lower levels provided an ideal equilibrium between performance and security, albeit with diminished detection capacity. However, this paper presents the need of further research in exploring the relationship between paranoia levels and false positives, which has been evaluated in our paper for each paranoia level across varying threshold by classifying the performance metrics such as True Positives, True Negatives, False Positives and False Negatives. Also it emphasizes on testing the rule set for layered encoding technique which is commonly used to bypass the WAF. In our research the testing utilizes various encoding techniques to the payloads to determine the WAF's capability to precisely identify the obfuscated payloads. Additionally, this research utilised CRS 3, which was the latest version available at that time for research, in our research we utilised the latest available CRS version - CRS 4 to identify potential gaps in the upgraded rule set.

3. Background

3.1. WAF

Web Application Firewall (WAF) is a web security tool that protects the web application from threat actors by filtering requests and responses between the user and the web application. Conventional firewalls protect networks at the packet level, but WAFs inspect HTTP traffic and also HTTPS traffic by decrypting using the private key of SSL/TLS certificate. Hence they are extremely effective against web-based attacks such as SQL injection, cross-site scripting (XSS), Remote Code execution, etc. They take action based on the predefined security rules to decide if a request is malicious or not by intercepting and analyzing the web traffic, hence reducing the risk of exploitation.

3.2. ModSecurity

ModSecurity (OWASP ModSecurity Team 2025) is a popular open-source rule-based WAF that is supported by the Open Worldwide Application Security Project (OWASP) (OWASP Team 2025). OWASP is an open community that works towards providing guidelines and suggestions on best security practices, Also they list and frequently update the top 10 vulnerabilities that a web application can be exposed to. ModSecurity can be effortlessly integrated with the Apache web server framework. This is because Apache (Apache 2025) is a modular server, meaning the functionalities can be extended such as SSL/TLS services, allowed methods, authentication, caching, etc. apart from its core functioning capabilities. ModSecurity is designed to be one such module where the WAF functionality can be easily extended to the Apache server. When extended, apache inspects the requests based on the predefined rules. ModSecurity can also be used in other different

environments such as NGINX (NGINX 2025) and IIS (IIS 2025) servers, making it a flexible WAF solution.

3.3. OWASP CRS

The OWASP CRS (OWASP CRS Team 2025) is a generic attack detection rule that is compatible with ModSecurity or other similar firewalls based on SecRule and follows a negative security model. The CRS contains rules based on regular expression which has been proven as one of the effective signature-based technique for common web attacks (Joseph and Jevitha, 2015) such as SQL Injection (SQLi), cross-site scripting (XSS), Local File Inclusion (LFI), Remote File Inclusion (RFI), PHP Code Injection, Java Code Injection, HTTPoxy, Shellshock, Unix/Windows Shell Injection, Session Fixation, Scanner/Bot Detection and Metadata/Error Leakages (OWASP CRS Team 2025).

The OWASP CRS has two modes to inspect requests which is self-contained mode and anomaly scoring mode. The anomaly scoring mode is set as default compared to the other mode where in self-contained mode the action is directly triggered after a single rule match but in anomaly scoring mode it uses collaborative matching where a request is given a number by matching against a set of rules for a match to identify how anomalous the request is. This technique greatly eliminates false positives based on the cumulative score and not a single rule, hence lowering the false positives and keeping security and usability well balanced (OWASP CRS Team 2025).

The OWASP CRS employs a number of parameters in the process of anomaly scoring to calculate a numeric score for the request like paranoia level, threshold value and severity levels. The paranoia level specifies the amount of rules activated, where every higher level encompasses rules from its lower levels. This progressive structure allows configuring the WAF based on the security needs required (OWASP CRS Team 2025).

The CRS has four paranoia levels:

- Paranoia Level 1: provides baseline security for anyone running an HTTP server on the internet with a minimal need to tune away false positives
- Paranoia Level 2: suitable when real user data is involved such as an online shop where some false positives are expected
- Paranoia Level 3: protects high-security services such as online banking, where more false positives can occur than in previous levels
- Paranoia Level 4: designed for very high-security requirements, where the strictness is very high by executing all the rules in the CRS, resulting in a high number of false positives compared to its lower levels

The false positives in the CRS can be reduced by adding rule exclusions by observing the kind of requests expected for that particular application (OWASP CRS Team 2025).

Another parameter in anomaly scoring mode is the *threshold* value. The OWASP CRS evaluates the inbound anomaly score for a request and the outbound anomaly score for a response separately, ensuring each is assessed separately for potential security risks. First, all request rules are executed, and the inbound anomaly score is calculated based on each matched rule's severity level to decide if the request should be blocked. Once the response is processed, all the response rules are executed, and the outbound anomaly score is checked against the threshold for a blocking decision. The threshold value is the accumulated score for an inbound request or an outbound response, and it determines if a request should be blocked or allowed. Increasing the threshold value would require more rule matches for a request/response to determine its maliciousness, hence increasing the threshold makes CRS less sensitive (OWASP CRS Team 2025).

In anomaly scoring mode, every rule in the CRS is assigned a severity level such as critical, error, warning and notice. Each assigned level carries a specific weight - Critical (5), Error (4), Warning(3) and Notice (2). When a rule is triggered, its severity weight is added to the anomaly score, indicating the seriousness of the detected threat. Rules with critical or error severity contribute more to the overall score, indicating high risk attacks, while warning and notice help identify less critical issues. This severity based scoring mechanism allows the WAF to classify and prioritise threats based on their potential security risk (OWASP CRS Team 2025).

The default OWASP CRS configuration is paranoia level-1 (PL-1) and anomaly threshold - 5 which provides baseline security, and threshold value 5 ensures if a single rule with a severity level of *critical* gets triggered then the request/response is blocked. The OWASP CRS can be configured in a strict setting by using paranoia level-4 (PL-4) the anomaly threshold - 5, where in PL-4 all the security rules in OWASP CRS are activated. This configuration allows deeper inspection and is capable of catching advanced or rare attack patterns that might bypass lower paranoia levels.

4. Methodology

4.1. Setup and configuration

For the purpose of this study, ModSecurity was installed as the Web Application Firewall (WAF) with the OWASP CRS on an Apache web server running in a controlled testing environment. The test environment was set up within a virtual machine running Kali Linux 2024.1 with 4 GB of RAM, which offers a stable environment for security testing. ModSecurity 2.9.8–1 was used as the web application firewall that was set up as an extended module to the base Apache 2.4 web server. The OWASP CRS 4.8.0-dev was combined with ModSecurity to introduce a predefined rule set to identify and prevent web attacks. The Damn Vulnerable Web Application (DVWA) was installed as the test application to mimic real world attack situations to enable controlled execution of various attack payloads.

The ModSecurity was set up in anomaly scoring mode, as the rule set utilized within our configuration is based on this mode as this reduces false positives, the detections were taken from the anomaly score and not from individual rule matches. The default OWASP CRS configuration employs paranoia level (PL-1) and anomaly threshold - 5 in the *crs-setup.conf* . This file is used to customize CRS settings like paranoia levels, anomaly scoring thresholds, severity level scores, rule exclusions, etc. The *modsec.conf* is the primary configuration file of ModSecurity which contains settings that define how ModSecurity should operate. This file is included in the web server's configuration to activate ModSecurity. For this study, debug log was enabled in the *modsec.conf* for easy examination of how ModSecurity handles various requests in order to get a better understanding of the rule behavior to enhance detection logic. Also, JSON processing was made explicit in *modsec.conf* to more easily examine structured JSON payloads, which was critical when testing new web attacks.

4.2. Testing methodology

A systematic test has been conducted after establishing the testing environment to test and observe how effectively ModSecurity configured with the OWASP CRS was able to identify and react to different kinds of web attacks. The testing was conducted to achieve the following objectives:

1. To analyse how the parameters in anomaly scoring mode in CRS impacted the performance metrics using the GoTestWAF testing tool, as it provides a comprehensive test report to classify these values.
2. To evaluate the effectiveness of the OWASP CRS rule set in detecting malicious payloads without any additional/layered encoding beyond the standard HTTP transmission encoding.

3. To evaluate the effectiveness of the OWASP CRS rule set in detecting malicious payloads when additional or layered encodings are applied beyond the standard HTTP transmission encoding.

The initial phase of the test was conducted by utilizing the GoTestWAF v0.5.6 test tool which automates sending both malicious and legitimate requests. This tool came in handy as it performs thorough testing through simulating requests that mimics the real world environment. Traditional penetration testing tools that concentrate on only vulnerability exploitation were not suited for this job, as GoTestWAF enabled us to see how well ModSecurity could differentiate between real threats and legitimate user activities.

The OWASP CRS uses the default configuration values which is paranoia level-1 and inbound threshold value of 5 as specified in the crs-setup.conf. But to learn more about detection behavior with various configurations, the test was performed on all paranoia levels (PL-1 to PL-4) with the threshold being 5, 8, 10, 13, 15, 18 and 20 for each level of paranoia by modifying the crs-setup.conf. The threshold value can be any positive integer, but to catch threats such as rare protocol attack where only a single specific *critical* rule type is defined in the OWASP CRS, hence the value 5 was chosen as the initial value (OWASP CRS Team 2025), then it was gradually increased with the mentioned values till 20, to observe how ModSecurity behaves with increasing threshold. The intention behind choosing this range of values is to observe how detection capability and permissiveness of the WAF changes as the threshold value and paranoia level increases.

This helped us realize how variations in the parameters of configuration influenced detection accuracy, especially when it came to true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). The objective of this test was to see how ModSecurity behaves when it is operating in anomaly scoring mode and observe how it could block attacks but not legitimate traffic.

The second phase of the test was conducted after setting the OWASP CRS in its strict configuration which is paranoia level 4 (PL-4) with an anomaly threshold of 5. This phase of testing focused on to evaluate ModSecurity's effectiveness in detecting and blocking various attack attempts. To test the basic detection capabilities of the WAF using the direct non-encoded payloads, the nemesida WAF bypass tool (Nemesida-WAF 2025) was chosen. This testing tool simulated 6813 payloads over 17 attack types such as API testing payloads (API), Custom HTTP Method payloads (CM), GraphQL testing payloads (GraphQL), LDAP injection (LDAP), Local File Inclusion (LFI), multipart/form-data payloads (MFD), NoSQL Injection payloads (NoSQLi), Open Redirect payloads (OR), Remote Code Execution payloads (RCE), Remote File Inclusion payloads (RFI), SQL Injection payloads (SQLi), Server-Side Includes payloads (SSI), Server-Side Request Forgery payloads (SSRF), Server-Side Template Injection payloads (SSTI), Unwanted Access payloads (UWA) and Cross-Site Scripting payloads(XSS).

To further challenge the detection capabilities of the WAF, GoTestWAF fuzzing tool (Wallarm 2025) was chosen due to its capability of using advanced evasion techniques such as using encoding techniques and placing the encoded payloads in different HTTP positions such as URLPath, URLParameter, HTMLForm, JSON bodies, XML payloads, and HTML MultipartForm to circumvent the WAF detection. The GoTestWAF simulated a wide range of attacks. These included CRLF injection, LDAP injection, mail injection, NoSQL injection, path traversal, remote code execution (RCE), shell injection, SQL injection, server-side include (SSI) injection, server-side template injection (SSTI), XML injection, and cross-site scripting (XSS) and API attacks. Additionally, community-contributed attack payloads were also included to assess how well ModSecurity could handle emerging threats beyond the standard OWASP CRS rules.

This phase of testing was done to identify payloads that successfully evaded detection even with the strict security settings. By analysing these bypasses, the detection gaps of the OWASP CRS were identified and addressed by rule enhancements to improve detection. After the rule improvement, the new configuration was retested to confirm the efficacy of the new rules.

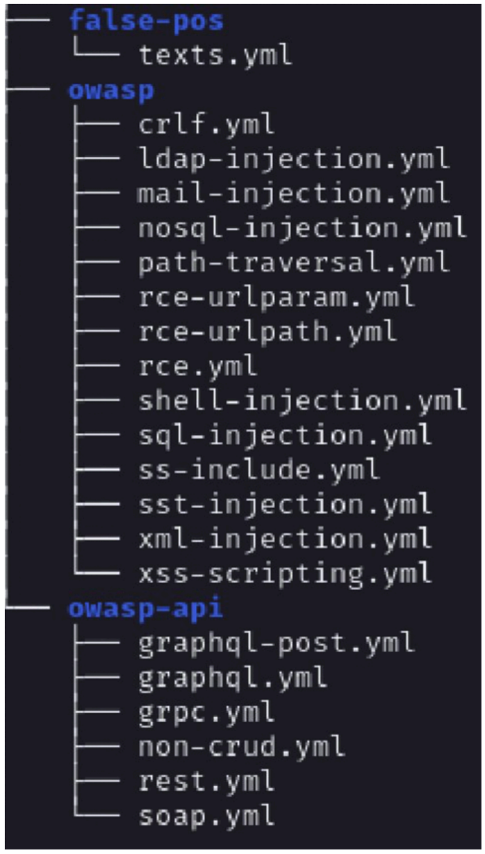

Fig. 1. Folder Structure of Payload Sets used by GoTestWAF.

4.3. Analysis

The analysis of the initial test to extract performance metrics values was done using the CSV report generated by the GoTestWAF as shown in Fig. 2, which holds the information of test results to reveal the WAF behaviour upon different attack simulations to examine its detection capabilities. Each entry in the CSV report consists:

- *'payload'* - denotes the payload used for the test
- *'check status'* - indicates whether the WAF blocked or allowed the request by categorising it as 'Blocked' or 'Passed'
- *'Response code'* - indicates the HTTP status code returned by the server
- *'encoder'* - denotes the encoding technique used on the payload such as Base64 or URL encoding, before placing it in the HTTP request component to test the WAF's ability to decode and detect obfuscated threats
- *'placeholder'* - specifies where the payload was injected in the HTTP request such as headers, body, or URL parameters to identify if a particular placeholder was vulnerable
- *'set'* - denotes the test case category that the payloads belong to, such as OWASP, OWASP-API, community or false positive as shown in Fig. 1
- *'case'* - defines the type under the *set* category such as SQL Injection, XSS scripting belong under the OWASP set or benign payloads defined as 'texts' belong to false positive *set* as shown in Fig. 1
- *'test result'* - summarises the WAF's response for performance assessment across various attack scenarios

Table 1
Performance Metrics and the condition used to extract from the GoTestWAF CSV Report.

Performance Metrics	Condition
True Positives (TP) - actual attack payloads identified and blocked by the WAF	'case' is not of 'texts' type and the 'check status' is 'Blocked'
False Positives (FP) - are cases where the WAF blocks legitimate traffic	'case' is of 'texts' type and the 'check status' is 'Blocked'
False Negatives (FN) is the attack payloads that might pass through the WAF detection	'case' is not of 'texts' type and the 'check status' is 'Passed'
True Negatives (TN) are legitimate traffic passed through the WAF without unnecessary blocking	'case' is of 'texts' type and the 'check status' is 'Passed'.

The effectiveness of the WAF was examined by pulling out critical performance values from the CSV report using certain conditions as shown in Table 1 to correctly recognize and categorize these values from the report. Values of these metrics were gathered for each unique configuration of the OWASP CRS by repeating the WAF testing by different threshold values - 5, 8, 10, 13, 15, 18 and 20 for each paranoia level. These measures served as the basis for examining the detection accuracy of the WAF which indicated its areas of strength and weaknesses.

The analysis of the second phase of the test involved observing the test results of the nemesida WAF bypass tool to determine the WAF's efficiency in its strict setting in identifying web threats without any obfuscation techniques. The test result of this tool is displayed only in the terminal, providing a straightforward approach for the analysis. The analysis involved checking the blocked and bypass rate of the False negative test section for every attack type involved in the testing, and test summary which highlights the overall blocked and bypass rate and the test final result. This analysis was helpful in determining the basic detection capabilities of the WAF setup.

The final analysis involved observing the test results of the GoTestWAF to determine the WAF's efficiency in identifying advanced threats that use encoding techniques to circumvent the WAF detection. The GoTestWAF's test result is displayed in the terminal, also it provides a comprehensive report by generating the test result in HTML and CSV format for detailed analysis and documentation of the test outcome. The CSV report was analysed by applying necessary conditions as mentioned in the Table 1 to extract False negative instances to observe if the WAF setup can identify obfuscated threats.

The identified payloads that bypassed this strict security configuration were further analysed to identify gaps in the rule set and trends in evasion techniques such as attack types, specific encoding techniques or the request component where these payloads were injected. This analysis gave valuable insights on the detection capabilities of the OWASP CRS by emphasising its strengths and weaknesses. The results from this analysis formed a foundation for proposing rule enhancements to address the limitations and improve the overall efficiency of the rule set.

4.4. Custom rule implementation

The custom rule implementation was done based on the findings from the GoTestWAF analysis, which showed major bypasses that weren't effectively detected by the existing OWASP CRS rules. While the WAF Bypass tool by Nemesida helped in evaluating the basic detection capabilities of the WAF, it was the more complex evasion techniques used by GoTestWAF such as encoding and obfuscation techniques that highlighted the actual gaps in the rule coverage. To address these gaps, a total of 146 new custom rules were implemented. The rule implementation is based on the SecRule directive of ModSecurity that follows the syntax:

SecRule VARIABLES "OPERATOR" "TRANSFORMATIONS,ACTIONS"

where *Variables* instructs ModSecurity where to target, *Operators* determine when to trigger a match, *Transformations* specify how it should normalize variable data and *Actions* instructs what to do if a rule matches (OWASP CRS Team 2025).

The custom rule implementation was done in three distinct phases where:

1. Extract the payloads using the appropriate ModSecurity *variable* as shown in Table A.1 from the placeholders such as URLPath, HTMLForm, URLParam, etc., where bypasses were observed.
2. Based on the analysis of encoding techniques used in bypassed payloads, it was observed that Base64 and URL encodings were predominantly used, hence the extracted payloads were decoded accordingly. Upon the observation of the debug log of ModSecurity, Base64 encoded payloads required additional processing after extraction in all the request components; they had to be decoded using a separate chained rule before evaluation. URL encoded payloads were decoded automatically at the time of capture in some placeholders like URLPath, URLParam and HTMLForm, but certain request components like HTMLMultipartForm, JSONRequest and Headers required an additional chained rule for URL decoding. However, we also observed that capturing only the payload after the last slash in the URLPath and decoding it was not possible using the existing transformation functions or the auto-decode plugin, as they decode the entire URLPath instead of just the segment after the last slash. This limitation required the use of custom rules.This tailored approach to handle decoding in different request components was crucial to effectively detect attacks that exploited encoding techniques to evade the OWASP CRS evaluation. The detailed breakdown of the rule designed to handle encoding cases provided in Appendix C.
3. After designing rules for each placeholder, the attack patterns for each attack type were implemented. This was done by extracting bypassed payloads of PL-4 for each attack type from the CSV report generated by the testing tool. The bypasses for an attack type were then categorised based on their associated HTTP request components by filtering the 'placeholder' column. For each identified placeholder, Since the payloads were recorded in an encoded format in the CSV report as shown in Fig. 2, the payloads were decoded according to the encoder used to restore their original structure. Once decoded, regular expressions were manually derived to match patterns observed in the bypassed payloads. For example, the regular expression for XSS was derived by analysing the bypassed payloads to capture the observed evasion patterns in the payload structure.

```
Payload,Check Status,Response Code,Placeholder,Encoder,Set,Case,Test Result
ax--exec=`id`--remote=origin,blocked,403,JSONRequest,Plain,owasp,rce,passed
cmd=127.0.0.1 && ls /etc,blocked,403,JSONRequest,Plain,owasp,rce,passed
; cat /et'c/pa'ss'wd,blocked,403,JSONRequest,Plain,owasp,rce,passed
ax--exec=`id`--remote=origin,blocked,403,Header,Plain,owasp,rce,passed
; cat /et'c/pa'ss'wd,blocked,403,Header,Plain,owasp,rce,passed
cmd=127.0.0.1 && ls /etc,passed,200,Header,Plain,owasp,rce,failed
```

Fig. 2. CSV report generated by GoTestWAF.

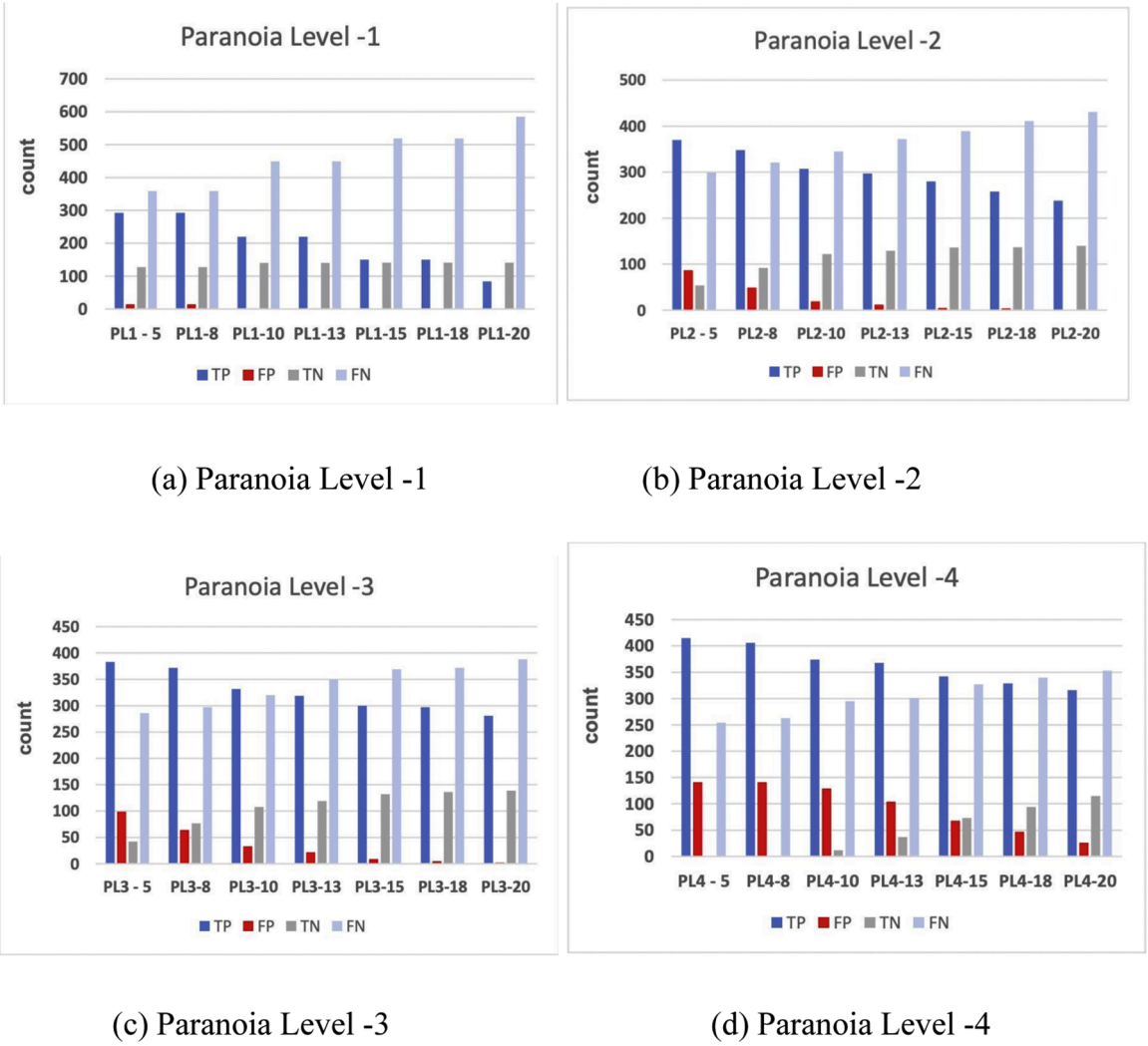

Fig. 3. Performance metrics across various paranoia and threshold values.

Patterns like <script[^>]*>.*?</script> and <script[^>]*src\s*=\s*['\"]?.*?['\"]?.*?> were used to detect both inline and externally sourced ⟨script⟩ tags. Event handler abuse was addressed using two overlapping components: <[^>]*on\w+\s*=\s*['\"]?.*?['\"]? detects handlers within HTML tags, while \bon\w+\s*=\s*['\"]?.*?['\"]? ensures detection of standalone JavaScript event assignments. The use of the *javascript: protocol* is detected through the pattern javascript:[^>]*, while SVG-based payloads such as ⟨svg onload=…⟩ are matched using <svg[^>]*>.*?</svg>. Any attempts to inject through media or script sources are captured via <[^>]*src\s*=\s*['\"]?.*?['\"]?. Direct invocation of common JavaScript functions such as alert(), confirm(), prompt(), eval(), and setInterval() is captured through the respective components alert\s*\(.*?\), confirm\s*\(.*?\), prompt\s*\(.*?\), eval\s*\(.*?\), and setInterval\s*\(.*?\). In addition, object references and advanced execution methods including document.domain, window.location, constructor(), apply(), and call() are explicitly matched using document\.domain, window\.location, constructor\s*\(.*?\), apply\s*\(.*?\), and call\s*\(.*?\). The pattern [\w\s]*=[^>]*\b(alert|prompt|confirm|eval|setInterval)\b.*? covers suspicious assignments, where malicious functions are bound to variables or properties without immediate invocation. Similarly the regular expressions were manually derived for other attack types as well by observing the payload patterns.

After deriving the regular expression, it was integrated into the appropriate placeholder's rule structure by modifying the rule ID to avoid rules with similar rule IDs. Since the testing tool fuzzes payloads across multiple placeholders, similar payloads were observed in different request components for the same attack type. As a result, the same regular expression was applied across other placeholders for an attack type as shown in Table C.1. The list of newly added rules to the existing OWASP CRS is provided*[1]

5. Results and discussions

5.1. Impact on performance metrics by varying CRS parameters

The performance evaluation of the OWASP CRS with ModSecurity was conducted for testing key performance parameters—True Positives (Malicious request correctly identified and blocked), False Positives (legitimate request incorrectly identified as malicious and blocked), True Negatives (legitimate request correctly identified and allowed), and False Negatives (Malicious request incorrectly identified as benign and allowed)—against four specified paranoia levels (PL-1, PL-2, PL-3, and PL-4) and threshold parameters at 5, 8, 10, 13, 15, 18 and 20 as mentioned in Section 4.2

[1] * The extended CRS ruleset is publicly available at: https://github.com/Anuvarshini09/coreruleset-with-custom-rules [Accessed 12 Aug 2025]

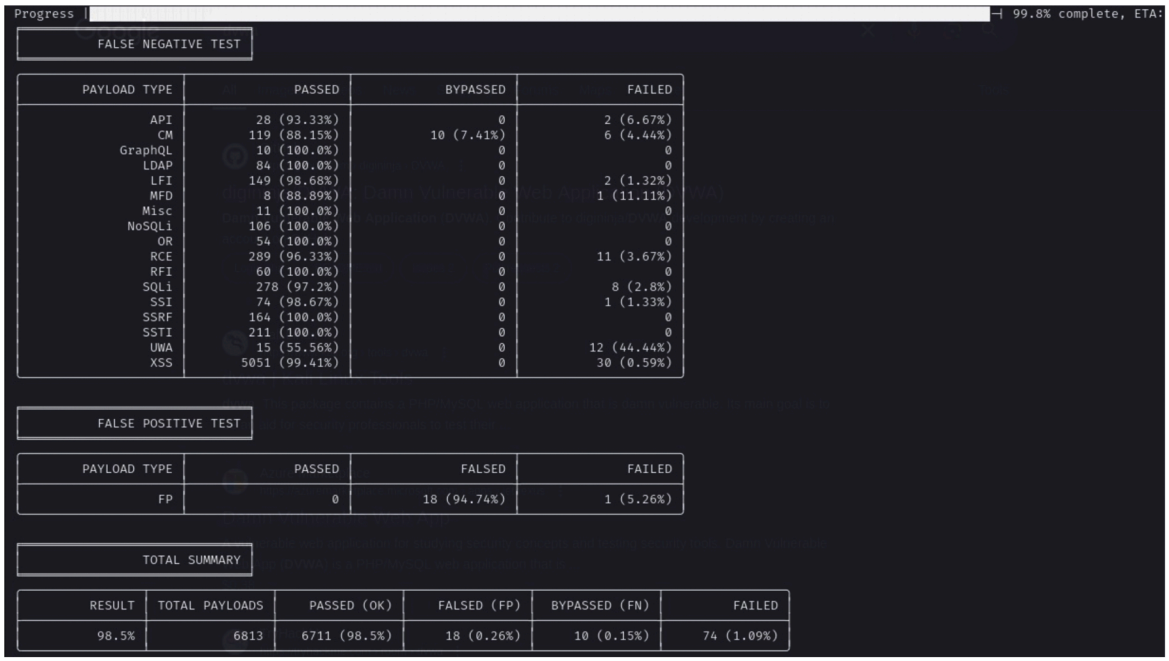

Fig. 4. Test Results of WAF bypass tool.

Fig. 3 represents bar graphs across different paranoia levels to represent the performance trends across paranoia levels, and each bar graph represents the performance across increasing threshold values for each paranoia level, illustrating the trade-off between detection effectiveness and usability. It is evident from Fig. 3, when we observe with a fixed threshold value 5, Paranoia Level 1 (PL1 - 5) had the lowest detection rate, and there were fewer false positives but at the cost of accepting a greater number of false negatives, with some sophisticated attack payloads remaining undetected. Conversely, when the paranoia level increases, i.e., PL2–5, PL3–5, PL4–5, had increasing detection, successfully reporting a greater number of malicious payloads, as reflected by the increasing true positive (TP) values across the figures. But increased TP came with increased FP, where legitimate traffic was improperly blocked. The performance variation of an increasing threshold value within a paranoia level, For example, In Fig. 3b at a minimum value of 5 (P2–5), the anomaly scoring mechanism could block more malicious payloads and reach greater TP value. The threshold value 5, however, rejected more legitimate payloads resulting in higher FP rates particularly at higher paranoia levels. However, increasing the threshold value to 20 (P2–20) as seen in Fig. 3b conversely decreased the FP rate but resulted in many malicious payloads passing through the WAF undetected, resulting in an increase in FN cases and a decrease in overall detection effectiveness

5.2. *Detection results from WAF bypass tool testing*

Fig. 4 below shows the detection rate of Modsecurity with OWASP CRS in its strict setting. The results demonstrate that ModSecurity,

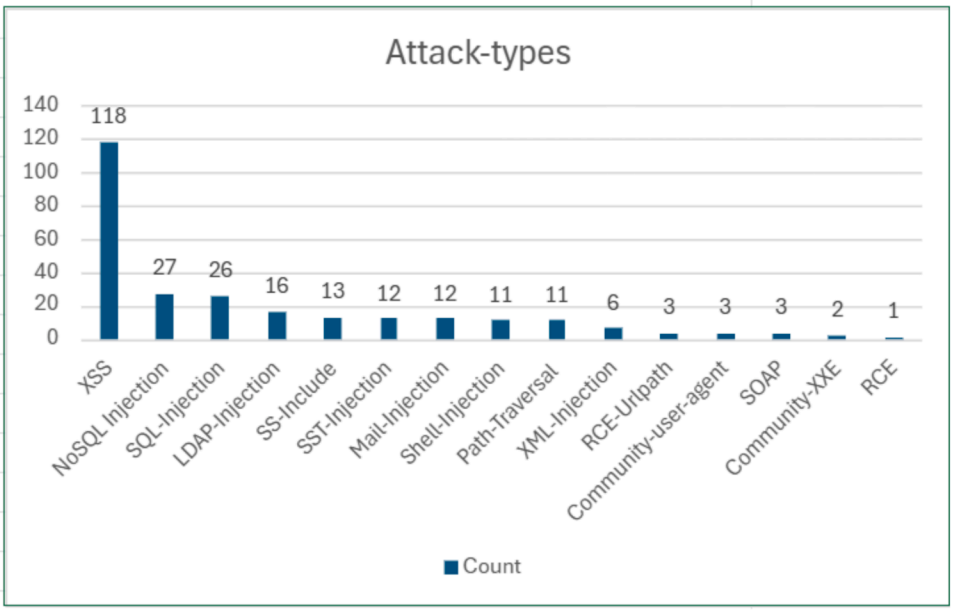

Fig. 5. Observed distribution of bypassed attack types.

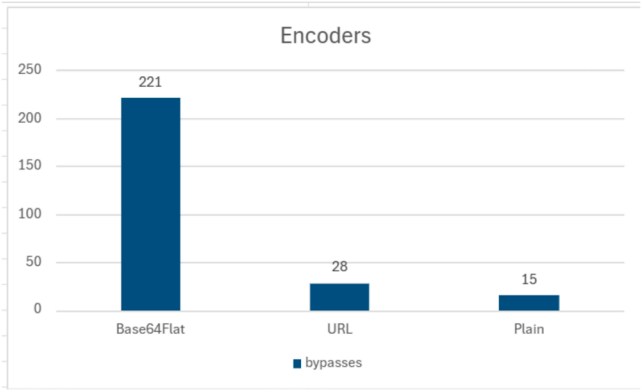

Fig. 6. Encoding techniques observed in bypassed payloads.

configured with the OWASP CRS, performed effectively against direct non-encoded attack payloads, with a detection accuracy of 98.5 %. The efficacy in blocking these types of threats proves that OWASP CRS guarantees its strength over traditional attacks, as the majority of straight injection and execution attempts were indeed blocked. These findings serve as a baseline for the next evaluation, the WAF response to obfuscated and encoded payloads, a more advanced evasion technique often utilized by attackers to bypass detection mechanisms.

5.3. Detection results from GoTestWAF testing

A total of 810 payloads were tested by GoTestWAF to observe the response of ModSecurity with OWASP CRS in its strict setting against obfuscated and encoded payloads. Among the total payloads 669 marked as malicious and 141 marked as legitimate. The CSV test results of GoTestWAF when analyzed shows that the WAF was only catching about 60 % of malicious requests out of 669 malicious attempts, it blocked 405 attack payloads but missed 264 attack payloads.

In observation of these 264 bypassed payloads, vulnerabilities in the detection strength of the OWASP CRS were identified. The bar graph in Fig. 5 illustrates the attack types identified among the 264 bypasses during GoTestWAF testing. Cross-Site Scripting (XSS) attacks were the most frequent, with 118 instances. After XSS, NoSQL Injection and SQL Injection appeared 27 and 26 times, respectively. Other types of attacks are also observed such as LDAP Injection with 16 bypasses, Server-Side Include (SSI) with 13 bypasses, and Server-Side Template Injection (SSTI) with 12 bypasses, Mail-Injection with 12 bypasses, Shell-Injection with 11 bypasses, Path-Traversal attack with 11 bypasses, XML-Injection with 6 bypasses, RCE - Url Path with 3 bypasses and 3 Community contributed bypasses for user-agent field and few bypasses were observed for SOAP endpoint, Community contributed XXE attacks and RCE. Although some of these attacks are less common, they still were also major security risks. It's important to address these issues to prevent possible breaches.

Encoding techniques are key to staying undetected. According to the bar graph in Fig. 6, Among 246 bypassed instances, 221 instances used Base64 encoding, 28 used URL encoding, and 15 used plain text with no encoding.The results indicate that CRS ruleset improvements are needed in order to effectively counter encoding-based evasion techniques.

The bar graph in Fig. 7 illustrates the distribution of request parameters associated with the 264 bypasses identified in the GoTestWAF results. A significant number of bypasses were linked to URL paths (72 instances), followed closely by URL parameters and HTML forms, each contributing 52 instances.HTML multipart forms accounted for 51 instances, JSON requests for 18, XML bodies for 8, headers for 5, while user-agent and SOAP bodies were involved in 3 cases.

Attack vector analysis shows that attackers frequently use encoding techniques such as Base64 encoding and URL encoding, but bypass strategies also extend beyond encoding where attackers take advantage of the HTTP request components such as URLPath, URLParam, HTMLForm, etc., to craft payloads to evade detection. In conclusion from the above results of the detection gaps of OWASP CRS, it was acknowledged that to improve security rules to catch more attacks while still letting legitimate users access the system without problems.

Table 2
Comparison of bypasses before and after custom rule inclusion for various attack types.

Attack Type	Bypasses before rule inclusion	Bypasses after rule inclusion
XSS	118	11
NoSQL Injection	27	0
SQL-Injection	26	2
LDAP-Injection	16	0
SS- include	13	1
SST-Injection	12	0
Mail-Injection	12	0
Shell-Injection	11	0
Path-Traversal	11	1
XML-Injection	6	0
RCE - Urlpath	3	2
Community-user-agent	3	0
SOAP	3	0
Community-XXE	2	0
RCE	1	0
Total	264	17

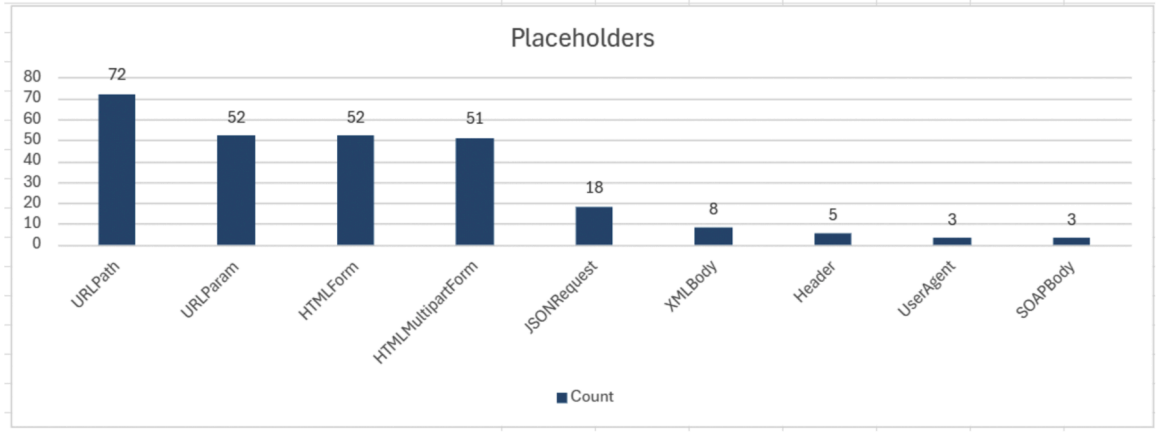

Fig. 7. Request components targeted in bypassed payloads.

Table 3
Comparison of false positives before and after custom rule inclusion.

False Positives before rule inclusion	False Positives after rule inclusion
141	141

5.4. Effectiveness of custom rule implementation

Implemented 146 new custom rules that made a huge difference in the rule set's ability to identify attacks that were initially bypassed by the tester. The detection rate rose from about 63% to nearly 97.5%, implying that the results went from 264 bypasses to only 17 bypasses out of 669 attempts. As shown in Table 2 below, Cross-Site Scripting (XSS) attacks significantly reduced successful attacks from 118 to just 11. SQL Injection attacks were also much better controlled, dropping from 26 successful attacks to only 2. Addition of new custom rules have completely stopped all NoSQL Injection attacks, going from 27 successful attacks to zero, similarly improvement for other attack types is also noticed. This implies that the new rules are good at stopping attacks that use encoding techniques. Base64 encoding, which was previously a major weakness with 233 successful attacks, is now well-defended. It also significantly improved protection for vulnerable areas like URL paths and HTML forms and other placeholders where bypasses were observed.

Looking at our overall security metrics, improvements were observed in catching real attacks (True Positives) while reducing missed attacks (False Negatives). Also, with the addition of new security rules we didn't see an increase in False Positives within our controlled test environment as shown in Table 3. While our findings show significant improvements in detection, however, these findings are based on a controlled lab environment using synthetic traffic. In practical deployments, the false positive rate may vary due to application behavior, traffic complexity, and deployment conditions. While the current results are encouraging, appropriate rule exclusions may be necessary in real-world scenarios to account for such variations and keep false positives minimal.

6. Conclusion

This research shows the significant impact of CRS configurations, including paranoia levels as well as threshold values on ModSecurity performance metrics. High paranoia levels indicate better detection rates (higher true positives) as a result of stricter and more aggressive rules enforcement. However, at the expense of increased false positives, for which application-specific exclusion rules must be established to counter the disruption of legitimate traffic. Conversely, lower paranoia levels reduce false positives, making the system less restrictive and minimizing interference with normal traffic. The adjustment simultaneously reduces detection effectiveness, hence increasing the chance of undetected attacks. Security mechanism tuning and user accessibility demand a perfect balance between paranoia levels and application-specific adjustment.

Within the same paranoia level adjustments to threshold values affect sensitivity-permissiveness balance. Lower thresholds improve security with increased blocking behavior, which further increases true positive rates, while higher thresholds reduce false positives with the risk of unintentionally admitting some malicious payloads to the system. This indicates the necessity to tune CRS configurations so that they may be optimized to suit individual application requirements.

The Nemesida WAF Bypass Tool test revealed that ModSecurity properly identified 6795 out of 6813 payloads, with a 98.5 % detection rate across various categories of attacks, indicating that the WAF has good detection capabilities with direct payloads. However, when tested with GoTestWAF which applies advanced bypass techniques such as encoding the payload, reveals the WAF is more vulnerable to evasion. This shows that while OWASP CRS performs well against simple attacks, its detection drops when handling encoded payloads. The findings highlight the need for better rules that can handle encoding to further strengthen the rule set's defenses against new evasion methods.

Adding 146 custom rules has introduced improvements in the effectiveness of the WAF setup, with the detection rate raised from 63 % to 97.5 % in GoTestWAF. These rules were specifically designed to address bypass techniques observed from our test results, such as handling multiple encodings, payload decoding after the last slash in the URLPath, and other evasive patterns. The CRS auto-decoding plugin was not used since it decodes the entire URLPath rather than allowing targeted decoding for specific components, for which our approach required. The improvements have drastically reduced successful evasion attempts involving encoding attacks, thus offering improved protection against different encoding-based bypasses, and other exploitation types, especially Cross-Site Scripting (XSS), SQL Injection attacks, NoSQL Injection attacks, etc. The observation point is that attackers typically go around default CRS rules using advanced encoding methods and input manipulation. The research shows that addition of custom rules significantly reduces false negatives with no increase in false positives within the scope of the test environment. The improvements are tied to the evasion patterns observed in this study and represent an incremental approach rather than a complete solution. The findings highlight that gaps still exist in CRS and require continuous refinement, as no single test or deployment can uncover all possible bypass techniques. Hence, our contribution should be viewed as an incremental improvement of the CRS, rather than a complete redesign of WAF detection. Since the enhancements were designed in direct response to observed gaps in OWASP CRS, the approach is naturally more aligned with CRS than with a generalised WAF framework. While this limits generalisation across diverse WAF ecosystems, the approach is still likely to remain relevant for future versions of CRS, provided the underlying architecture continues to follow SecLang.

Future activities can involve testing the new custom rules on various environments using the CRS configuration parameters to further evaluate their effectiveness and explore advanced bypass techniques beyond the scope of this testing tool. The evaluation can be extended by testing the ruleset in production-like environments with more diverse web applications and traffic patterns. This would help assess how the rules perform under more realistic and complex conditions, beyond controlled lab setups. By strengthening detection mechanisms and systematically addressing known bypass techniques, this research contributes to enhancing the security posture of web applications, ensuring robust protection against evolving threats while preserving operational functionality.

CRediT authorship contribution statement

Anuvarshini MK: Writing – review & editing, Writing – original draft, Methodology, Formal analysis, Conceptualization. **Kommuri Sai Suhitha Bala:** Writing – original draft, Formal analysis, Conceptualization. **Sri Sai Tanvi Sonti:** Writing – original draft, Formal analysis. **Jevitha KP:** Writing – review & editing, Validation, Supervision, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

At the very outset, we would like to offer our heartfelt gratitude to the Almighty, whose blessings gave us the strength, wisdom, and perseverance to complete this research. We express our sincere thanks to Mr. Sitaram Chamarty, Professor of Practice, TIFAC—CORE in Cyber Security, Amrita Vishwa Vidyapeetham, for his invaluable insights and

constructive feedback, which greatly enhanced the quality of our work. We would also like to express our heartfelt thanks to Prof. M. Sethumadhavan, Professor and Head, TIFAC—CORE in Cyber Security, for his continuous support and for facilitating the necessary resources and environment that enabled us to carry out our research smoothly.

Appendix A. Variables involved in ModSecurity rule processing

Table A.1
Payload extraction using ModSecurity variables.

Placeholder	Variable and Regular expression to extract payload
URLPath	The ModSecurity variable REQUEST_URI is used to target the URL component of the Request. A regular expression: ' ^.*/([^/]+)$ ' is then evaluated on the captured URL to extract the payload after the last slash.
URLParam	The ModSecurity variable ARGS_GET targets key-value pairs from the URL query string of an incoming HTTP Request. A regular expression: '.+' is then evaluated to extract the key-value pairs.
HTMLForm, HTMLMultipartform	The ModSecurity variable ARGS_POST targets key-value pairs from the POST request body of an incoming HTTP Request. A regular expression: ".+'' is then evaluated to extract the key-value pairs.
JSONRequest	The ModSecurity variable ARGS in the rule targets key-value pairs from the GET/POST request of an incoming HTTP Request. Since JSON data is structured, an additional request body processor for JSON is enabled in modsecurity.conf to parse and extract values in key-value format before pattern matching. A regular expression: '.+' is then evaluated to extract the key-value pairs
Headers	The ModSecurity variable REQUEST_HEADERS targets all HTTP request headers as key-value pairs. A regular expression: '.+' is then evaluated to extract the matched headers.
REQUEST_HEADERS:User-Agent	The ModSecurity variable REQUEST_HEADERS targets only User-Agent as key-value pairs. A regular expression: '.+' is then evaluated to extract the matched content.

Appendix B. Rule design for decoding of encoded payloads for different request placeholders

i) URLPath, URLParam and HTMLForm

The rule id 100,001 captures the payload from the request component then stores it in a variable ''tx.raw_value' for further processing. As discussed above, these placeholders automatically decode URL encoded payloads at the time of capture, hence the captured payload is directly evaluated against the attack pattern in rule id 100,003. The rule id 100,002 is a chained rule that handles the case of base64 encoded payloads, where it applies Base64 decoding to the payload captured in the rule id 100,001 and then finally evaluated against the attack payload in the chained rule id 100,003.

SecRule REQUEST_URI|ARGS_GET|ARGS_POST "@rx {appropriate regex as shown in Table 2}" \

```
"id:100,001,phase:2,log,auditlog,pass,capture,\
setvar:'tx.raw_value= %{MATCHED_VAR}',\
msg:'Captured value: %{TX.raw_value}'"
```

SecRule TX:raw_value "@rx .+" \

```
"id:100,002,phase:2,log,auditlog,pass,t:base64Decode,capture,\
setvar:'tx.base64_decoded_value= %{MATCHED_VAR}',\
msg:'Base64-decoded value: %{TX.base64_decoded_value}'"
```

SecRule TX:raw_value|TX:base64_decoded_value "@rx {attack pattern}" \

```
"id:100,003,phase:2,log,auditlog,block,\
msg:'Detected attack pattern',severity:CRITICAL,\
tag:'paranoia-level/4',\
setvar:'tx.xss_score=+ %{tx.critical_anomaly_score}',\
setvar:'tx.inbound_anomaly_score_pl4=+ %{tx.critical_anomaly_score}'"
```

ii) HTMLMultipartForm, JSONRequest, Headers and User-Agent

The rule id 100,004 captures the payload from the request component then store it in a variable ''tx.raw_value' for further processing. As discussed above, these placeholders does not automatically decode URL encoded payloads at the time of capture, hence the captured payload is URL decoded separately in the rule id 100,006, like how the Base64 payloads are decoded separately after the capture in the rule id 100,005 . Once the payload is decoded, it is evaluated against the attack pattern in rule id 100,007.

SecRule ARGS_GET|ARGS|HEADERS|HEADERS:USER-AGENT "@rx {appropriate regex as shown in Table 2} " \

```
"id:100,004,phase:2,log,auditlog,pass,capture,\
setvar:'tx.raw_value= %{MATCHED_VAR}',\
msg:'Captured value: %{TX.raw_value}'"
```

SecRule TX:raw_value "@rx .+" \

```
"id:100,005,phase:2,log,auditlog,pass,t:base64Decode,capture,\
setvar:'tx.base64_decoded_value= %{MATCHED_VAR}',\
msg:'Base64-decoded value: %{TX.base64_decoded_value}'"
```

SecRule TX:raw_value "@rx .+" \

```
"id:100,006,phase:2,log,auditlog,pass,t:urlDecodeUni,capture,\
setvar:'tx.url_decoded_value= %{MATCHED_VAR}',\
msg:'URL-decoded value: %{TX.url_decoded_value}'"
```

SecRule TX:base64_decoded_value|TX:url_decoded_value "@rx {attack pattern}" \

```
"id:100,007,phase:2,log,auditlog,block,\
msg:'Decoded payload contains malicious pattern',severity:CRITICAL,\
tag:'paranoia-level/4',\
setvar:'tx.xss_score=+ %{tx.critical_anomaly_score}',\
setvar:'tx.inbound_anomaly_score_pl4=+ %{tx.critical_anomaly_score}'"
```

Appendix C. Regular expression derived for various attack types

Table C.1
Attack types with associated bypass placeholders and regular expression derived.

Attack type	Placeholder	Regular expression
1. Cross-site scripting (XSS)	URLPath, URLParam, HTMLForm, HTMLMultipartform	(?i)(<script[^>]*>.*?<\/script>|<script[^>]*src\s*=\s*['\"]?.*?['\"]?.*?>|<[^>]*on\w+\s*=\s*['\"]?.*?['\"]?|javascript:[^>]*|<svg[^>]*>.*?<\/svg>|<[^>]*src\s*=\s*['\"]?.*?['\"]?|alert\s*\(.*?\)|confirm\s*\(.*?\)|prompt\s*\(.*?\)|eval\s*\(.*?\)|setInterval\s*\(.*?\)|document\.domain|window\.location|constructor\s*\(.*?\)|apply\s*\(.*?\)|call\s*\(.*?\)|[\w\s]*=[^>]*\b(alert|prompt|confirm|eval|setInterval)\b.*?|\bon\w+\s*=\s*['\"]?.*?['\"]?)
2. NoSQL Injection	URLPath, URLParam, HTMLForm, HTMLMultipartform, JSONRequest	(?:\bnew\s+Date\b|do\s*{|\bwhile\b|db\.injection\.insert\b|true\b|\$where\b|\bvar\s+\w+\s*=\s*[^;]+;|\d+\s*==\s*\d+|;|\$or\b)
3. SQL Injection	URLPath, URLParam, HTMLForm, HTMLMultipartform, JSONRequest and Header	\bselect\s*\(\s*0\s*\)\s*from\s*\(select\s*\(sleep\s(\d+\))\)\s*v|EXEC\s+Master\.dbo\.xp_cmdshell|JSON_EXTRACT\(\s*''{''aKER'':\s*\d+}''\s*,\s*''\$.aKER''\)\s*=\s*\d+|JSON_DEPTH\(\s*''\{\}''\s*\)\s*!=\s*\d+
4. LDAP Injection	URLParam, HTMLForm, HTMLMultipartform, JSONRequest	(?i)*\s*\(|\(objectclass=*\)\)|\(&\(uid=admin\)\(\!\(&\(1 = 0\)\(userPassword=.*\)\)\)|userPassword\s*:\s*2\.5\.0\.13\.0\.18\s*:=\s*\d+
5. SS-Include	URLPath, URLParam, HTMLForm, HTMLMultipartform	<!–#exec\s+cmd\s*=\s*['\"]?(wget\s+http:[^\s]+[^'\"]*\s*|?\s*rename\s+[^'\"]+\s+[^'\"]+|ls|dir)['\"]?\s*–>
6. SST-Injection	URLPath, URLParam, HTMLForm, HTMLMultipartform	(?i)(%2b|\+| %27|')|#[^}]|{[^}]}|<#assign\s+\w+\s*=.?>\s\${[^}]*}
7. Mail-Injection	URLPath, URLParam, HTMLForm, HTMLMultipartform	(?i)(\r?\n)*V\d{3}\s+CAPABILITY(\r?\n)*|(\r?\n)*QUIT(\r?\n)*|(\r?\n)*RCPT\s+TO:\s*[^\s@]+@[^\s@]+\.[^\s@]+(\r?\n)*
8. Shell–Injection	URLParam, HTMLForm, HTMLMultipartform, JSONRequest	(?i)[;|]\s*(getent\s*\$IFS\$9hosts|\$IFS\$9\$\(\d+*\d+\)|wget\s+https?:\/\/[^\s]+|set\s*\/a\s*\d+*\d+)
9. Path traversal	URLParam, URLPath, HTMLForm, HTMLMultipartform, JSONRequest	(?i)(?:\.\./|\.\.\\\| %2e %2e| %2e %2e %2f| %2e %2e %5c| %2e %2e %2f| %2e %2e %5c)(?:etc.|windows|system32|root|var|home|user|default|config|passwd|shadow|hosts)|(?:[a-zA-Z]:\\\|\\\\)(?:c\\\|\\\\)(?:users|windows|system32|default|admin|public|documents|downloads)\\[^\\]*\.dat
10. RCE–URLPath	URLPath	(?i)\bexec=`[^`]+`|–remote=origin
11. User Agent	Header:User-Agent	(http:\/\/[a-zA-Z0–9.-]+\.burpcollaborator\.net\/[0–9]+\.[0–9]+\.[0–9]+\.RELEASE\s*iPhone[0–9,]+.*\)\)|Mozilla\/5\.0.0\s+\(Windows NT\s+[0–9.]+;\s*WOW64\)\s*AppleWebKit\/[0–9.]+\s*\(KHTML,\s+like\s+Gecko\)\s*Chrome\/[0–9.]+\s*Safari\/[0–9.]+\s*root@[a-zA-Z0–9.-]+\.interact\.sh|mercuryboard_user_agent_sql_injection\.nasl')
12. RCE	Header	(?i)cmd\s*=\s*\d{1,3}(\.\d{1,3}){3}\s*&&\s*ls\s+\/[^\s]+

Data availability

Insights drawn were based on the testing tool results and the testing tools are open source

References

Aïmeur, E., 2014. Online privacy: risks, challenges, and new trends. In: International Conference on Risks and Security of Internet and Systems. Springer, pp. 263–266.

Apache, Apache web server official site, accessed: 15-Mar-2025. URL https://apache.org/.

Díaz-Rojas, J.A., Ocharán-Hernández, J.O., Pérez-Arriaga, J.C., Limón, X., 2021. Web api security vulnerabilities and mitigation mechanisms: a systematic mapping study. In: 2021 9th International Conference in Software Engineering Research and Innovation (CONISOFT). IEEE, pp. 207–218.

IIS, IIS web server official site, accessed: 17-Mar-2025. URL https://www.iis.net/.

Joseph, S., Jevitha, K.P., 2015. Evaluating the effectiveness of conventional fixes for sql injection vulnerability. In: Proceedings of 3rd International Conference on Advanced Computing, Networking and Informatics: ICACNI 2015, 2. Springer, pp. 417–426.

Kollepalli, R.P.K., Reddy, M.J.S., Sai, B.L., Natarajan, A., Mathi, S., Ramalingam, V., 2024. An experimental study on detecting and mitigating vulnerabilities in web applications. Int. J. Saf. Secur. Eng. 14 (2).

Mouli, V.R., Jevitha, K.P., 2016. Web services attacks and security-a systematic literature review. Procedia Comput. Sci. 93, 870–877.

Mukhtar, B.I., Azer, M.A., 2020. Evaluating the modsecurity web application firewall against sql injection attacks. In: 2020 15th International Conference on Computer Engineering and Systems (ICCES). IEEE, pp. 1–6.

Muzaki, R.A., Briliyant, O.C., Hasditama, M.A., Ritchi, H., 2020. Improving security of web-based application using modsecurity and reverse proxy in web application firewall. In: 2020 International Workshop on Big Data and Information Security (IWBIS). IEEE, pp. 85–90.

Nagendran, K., Balaji, S., Raj, B.A., Chanthrika, P., Amirthaa, R., 2020. Web application firewall evasion techniques. In: 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS). IEEE, pp. 194–199.

Naresh, S., Jevitha, K.P., 2020. Formal analysis of openid connect protocol using tamarin prover. In: International Conference on Advances in Electrical and Computer Technologies. Springer, pp. 297–309.

Nemesida-WAF, WAF bypass tool, accessed: 17-Mar-2025. URL https://github.com/nemesida-waf/waf-bypass.

NGINX, NGINX web server official site, accessed: 15-Mar-2025. URL https://nginx.org/.

Nyabuto, M.G.M., Mony, V., Mbugua, S., 2024. Architectural review of client-server models. Int. J. Sci. Res. Eng. Trends 10 (1), 139–143.

OWASP CRS Team, OWASP CRS main site, accessed: 17-Mar-2025. URL https://coreruleset.org/.

OWASP ModSecurity Team, Open source web application firewall,accessed: 15-Mar-2025. URL https://modsecurity.org/.

OWASP Team, OWASP official site, accessed: 17-Mar-2025. URL https://owasp.org/.

Rahmawati, T., Shiddiq, R.W., Sumpena, M.R., Setiawan, S., Karna, N., Hertiana, S.N., 2023. Web application firewall using proxy and security information and event management (siem) for owasp cyber attack detection. In: 2023 IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS). IEEE, pp. 280–285.

Ravindran, Rahulkrishnan, Abhishek, S., Anjali, T., Shenoi, Ashwin, 2023. Fortifying web applications: advanced XSS and SQLi payload constructor for enhanced security. In: International Conference on Information and Communication Technology for Competitive Strategies. Singapore. Springer Nature Singapore, pp. 421–429.

Razzaq, A., Hur, A., Shahbaz, S., Masood, M., Ahmad, H.F., 2013. Critical analysis on web application firewall solutions. In: 2013 IEEE Eleventh International Symposium on Autonomous Decentralized Systems (ISADS). IEEE, pp. 1–6.

Sharma, A., Verma, R., Nahar, O., 2019. Managing security in client-server network infrastructure. In: Proceedings of International Conference on Sustainable Computing in Science, Technology and Management (SUS-COM). Jaipur-India. Amity University Rajasthan.

Sharma, S., Jevitha, K.P., 2023. Security analysis of oauth 2.0 implementation. In: 2023 Innovations in Power and Advanced Computing Technologies (i-PACT). IEEE, pp. 1–8.

Singh, J.J., Samuel, H., Zavarsky, P., 2018. Impact of paranoia levels on the effectiveness of the modsecurity web application firewall. In: 2018 1st International Conference on Data Intelligence and Security (ICDIS). IEEE, pp. 141–144.

Snorby, Ruby on rails application for network security monitoring, accessed: 17-Mar-2025. URL https://github.com/Snorby/snorby.

SQLmap, Automatic SQL injection and database takeover tool, accessed: 17-Mar-2025. URL https://github.com/sqlmapproject/sqlmap.

Vishnu, B., Jevitha, K.P., 2014. Prediction of cross-site scripting attack using machine learning algorithms. In: Proceedings of the 2014 International Conference on Interdisciplinary Advances in Applied Computing, pp. 1–5.

Wallarm, GoTestWAF github repository, accessed: 17-Mar-2025. URL https://github.com/wallarm/gotestwaf.