Computer Modeling in Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2022.020976

ARTICLE

Systematic Approach for Web Protection Runtime Tools' Effectiveness Analysis

Tomás Sureda Riera[1,*], Juan Ramón Bermejo Higuera[2], Javier Bermejo Higuera[2], Juan Antonio Sicilia Montalvo[2] and José Javier Martínez Herráiz[1]

[1]Computer Science Department, University of Alcalá, Madrid, 28801, Spain

[2]Escuela Superior de Ingeniería y Tecnología, Universidad Internacional de La Rioja, Logroño, 26006, Spain

*Corresponding Author: Tomás Sureda Riera. Email: tomas.sureda@uah.es

Received: 22 December 2021 Accepted: 09 March 2022

ABSTRACT

Web applications represent one of the principal vehicles by which attackers gain access to an organization's network or resources. Thus, different approaches to protect web applications have been proposed to date. Of them, the two major approaches are Web Application Firewalls (WAF) and Runtime Application Self Protection (RASP). It is, thus, essential to understand the differences and relative effectiveness of both these approaches for effective decision-making regarding the security of web applications. Here we present a comparative study between WAF and RASP simulated settings, with the aim to compare their effectiveness and efficiency against different categories of attacks. For this, we used computation of different metrics and sorted their results using F-Score index. We found that RASP tools scored better than WAF tools. In this study, we also developed a new experimental methodology for the objective evaluation of web protection tools since, to the best of our knowledge, no method specifically evaluates web protection tools.

KEYWORDS

Web Application Firewall (WAF); Runtime Application Self Protection (RASP); F-Score; web attacks; experimental methodology

1 Introduction

Most computer security attacks against organizations involve an attempt to exploit a number of web application vulnerabilities [1].

As per the 2016 Internet Security Threat Report (ISTR) made by Symantec [2], 1.1 M of web application attacks are blocked every day and 78% of websites have some type of vulnerability, including 15% with critical vulnerabilities. The 2016 Web Applications Security Statistics Report published by WhiteHat Security [3] stated, "Web application attacks represent the greatest threat to an organization's security. Web app attacks represented 40% of breaches in 2015."

Attacks on web applications are significantly increasing year after year. The most prevalent attacks include SQL injection attacks (SQLi) and Cross Site Scripting (XSS) attacks [1].

The Open Web Application Security Project publishes the OWASP Top Ten document, currently in its 2021 version, which identifies the most critical risks in web applications of organizations. The top three in these lists are occupied by Broken authentication, Cryptographic Failures and Injection attacks [4].

Web Application Firewall (WAF) tools are one of the most widely used methods for web application protection. A WAF is deployed between the application and the requesting user to inspect the incoming traffic at the application layer of the OSI model and look for attack patterns, eventually blocking incoming malicious traffic. WAF devices perform a deep packet inspection of incoming network traffic and check the incoming traffic against a database of signatures or rules. These devices are independent of the programming language of the web application as they act before the malicious traffic can execute the code [5]. However, studies have proposed different ways to circumvent the protection provided by a WAF [6–8]: Normalization method, HTTP Parameter Pollution, HTTP Parameter Fragmentation, logical requests AND/OR, replacing SQL functions that get WAF signatures with their synonyms, using comments, case changing, and triggering a Buffer Overflow/WAF Crashing. Most of these techniques focus on *protocol-level evasion* that attemp to exploit the little differences in how WAFs see traffic and how backend web servers and applications see it.

Gartner [9] defined Runtime Application Self-Protection (RASP) tools as "a security technology that is built or linked into an application or application runtime environment, and is capable of controlling application execution and detecting and preventing real-time attacks." These tools combine real-time contextual awareness of the factors that have led to the application of current behavior [10,11]. These tools "embed" the security in the application server to be protected, intercepting all calls to the system to check whether they are secure. Thus, RASP tools depend entirely on the programming language of the application to protect.

On the other hand, Nicolett [12] reported that the WAF and RASP technologies play a fundamental role in protecting web applications, shielding the vulnerable parts of the application until they are mitigated through code correction by the developers' team.

Both technologies work in the application layer filtering the HTTP protocol. WAF is a black box technology. It intercepts the calls and responses to the logic of the application to be protected, performing a syntactic analysis to detect attacks. Thus, it does not require access to the logic of the application. On the other hand, RASP is a white box technology that installs an agent in the application server to examine the assignment of values to the variables in the process stack. The RASP may even have to examine the code of the application to decide if a payload is an attack attempt or not.

Referring to the principles of defense in depth, it is advisable to incorporate additional layers of defense into an organization's computer environment. Given the recent emergence of RASP tools, it is necessary to compare the two types of protection tools objectively. Such a comparison will allow us to determine whether these tools can be complementary and establish an additional defense layer, improving an organization's security.

However, to our best knowledge, there is no specific methodology that allows us to evaluate different web protection tools.

In this paper, we analyzed and evaluated the performance of different WAF and RASP tools in protecting a web application (simulated by a benchmark) against different types of attacks. We here aimed to determine the best suitable tool based on objective criteria through the collection and subsequent tabulation of false positives, false negatives, true positives and true negatives. The

different tools were sorted according to their effectiveness by using the F-Score index [13]. We have also developed a new experimental methodology to evaluate the different tools analyzed in this paper; the proposed methodology can be generalized for the evaluation of other tools that work in the network layer such as IDS and IPS. This methodology allows the use of any other automatic vulnerability scanning tool in addition to the one used in this work. It also includes different categories of vulnerabilities for analysis and different test benches in addition to those used in this work.

The main contributions of this study are:

- Analysis of the performance of different WAF and RASP tools protecting web applications.
- Using the F-Score indicator to present the results obtained by different web protection tools. The tools that show greater efficiency in the detection of attacks minimizing the presence of false positives obtain a higher score in the F-Score indicator.
- Development of a new experimental methodology to evaluate web protection tools.

2 Related Work

In Section 2.1, we present a review of the studies carried out on WAF technologies. In Section 2.2, we review studies on RASP technology, including studies on self-protecting, self-healing and self-adaptive systems. Finally, in Section 2.3, we present a review of the studies that evaluated and compared web protection tools.

2.1 Web Application Firewall

In this study, Byrne [14] highlighted the role played by the WAF in the in-depth defense of an IT system. He stated that the WAF detection methods evolved from a static to a dynamic approach because of their use of machine-learning techniques.

Schmitt et al. [15] pointed out that an attacker can remotely determine whether the request has been blocked (shorter response time) or not by a WAF by measuring the time used to return a response to a legitimate request and another malicious request. Based on this, the attacker can adjust the construction of the payloads to evade the protection of the WAF.

Holm et al. [16] estimated the effectiveness of web application firewalls in preventing of SQL injection attacks carried out by professional penetration testers. They considered the presence or absence of four conditions: the use of an experienced operator that monitors the WAF, tuning the WAF using an automated black box tool, using an experienced professional to tune the WAF, and spending significant effort to tune the WAF. They measured the effectiveness using 16 different operational scenarios that were judged by a panel of 49 experts. The authors did not use any objective measures like false positive, false negative, etc.

Different studies have attempted to improve the effectiveness of WAF solutions. Moosa [17] proposed a model based on an Artificial Neural Network (ANN) that can effectively protect against SQL injection attacks. This proposed model has a training phase and a working phase. In the training phase, the ANN was exposed to a set of normal and malicious data so as to train the ANN. In the working phase, the trained ANN was integrated into the WAF to protect the web application.

In an attempt to improve the effectiveness of the WAF rule sets, Auxilia et al. [18] proposed a Negative Security Model that monitors requests for anomalies, unusual behavior, and common web application attacks.

Applebaum et al. [19] conducted a survey showing the development of WAFs based on machine-learning methods.

2.2 RASP and Self-Protected Systems

Many studies examined self-protected, self-healing and self-adaptive systems.

Weyns et al. [20] conducted a literature review and showed that self-adaptive systems improved the flexibility, reliability and performance of the system. They also pointed out the lack of empirical systematic studies as well as the lack of industrial evidence in support of such systems.

Yuan et al. [21] proposed a taxonomy to classify self-protection systems and research approaches. The proposed taxonomy consisted of nine dimensions, which were grouped into two different categories. The first category, Research Positioning, characterized the objectives and the investigation of self-protection, i.e., the aspects of "WHAT." The second category, Technique Characterization, described the "HOW" aspects of self-protection research.

Yuan et al. [22] redefined this taxonomy was redefined. Their taxonomy contained 14 dimensions that were grouped into three categories: The first category, Approach Positioning, characterized the objectives and attempts of self-protection research: the "WHAT" aspects. The second category, Approach Characterization, classified the "HOW" aspects of self-protection research. The third category, Approach Quality, evaluated self-protection research.

Keromytis [23] characterized the general architecture of a self-healing system. In the proposed model, the system enters a self-diagnostic state on detecting anomalous behaviors in order to identify the fault and extract as much information as possible about the causes and impact on the system. Subsequently, the system attempts to self-adapt generating candidate fixes, which are tested in the self-testing phase in order to determine the fix that best suits the optimum state of the system; once the fix has been found, it is deployed on the protected system.

According to Lane [24], most RASP tools can learn from the applications they are protecting. The author differentiates between passive and active learning. The process helps to refine detection rules by adapting generic ones to match specific requests and add fraud detection capabilities. Another model used a security positive approach (whitelisting); this way, the RASP tool knows how API calls are made and what certain lines of code look like, blocking unknown patterns.

Taint analysis is one of the techniques used in RASP solutions. Haldar et al. [25] analyzed the externals to the web application data, marking them as untrusted (tainted data).

Zeller et al. [26] proposed a system with a self-protecting layer to protect it against attacks aimed at misusing business logic rules. Yin et al. [27] suggested a scheme that automatically added the injection filtering instruction into the existing web program using static code analysis technology.

Another study used RASP technology to design and implement the runtime self-protection framework of smart contracts [28].

2.3 Evaluation and Comparison of Protection Tools

The Web Application Firewall Evaluation Criteria [29] attempts to develop a standardized criterion for the evaluation of WAFs in order to provide users with a tool that allows an educated decision when selecting a WAF. However, it only describes a series of characteristics that must be taken into account in choosing a WAF.

The SANS Institute [30] compared the HP Application Defender's RASP solution and an unnamed WAF and reviewed their respective preventive and detective capabilities. However, this report lacked the metrics to establish an objective comparison between the two solutions compared within it.

Razzaq et al. [31] presented a critical analysis of different WAFs. However, their analysis was limited to "*defense mechanism*": security policy control, monitoring, blocking, response filtering, attack prevention, authentication, website cloaking, deep inspection, session protection and overall security performance. These criteria also included the management interface.

3 Materials and Methods

Section 3.1 presents the categories of vulnerabilities against which the various WAF/RASP protection tools will be analyzed. In Section 3.2, we present a review of the studies carried out on the generation of test cases capable of revealing vulnerabilities in a web application. In Section 3.3, we present the metrics used in our study and discuss the reasons for using this metrics in Section 3.4. In Section 3.5, we present the WAF and RASP tools that were evaluated and compared. In Section 3.6, we provide an explanation of how the different tools have been configured and the reason for doing it in that specific way. We also provide a detail of the categories of vulnerabilities covered by each tool. Finally, in Section 3.7, we explain the different attack detection methods used by each tool.

3.1 Categories of Vulnerabilities Analyzed

- Command Injection: Classified by Mitre with the code CWE-78 [32]. These attacks are produced usually by the lack of validation of user input data or calls to unsafe functions that execute commands from the operating system on which the web application is run. This vulnerability may allow an attacker to execute unexpected commands directly in the operating system.
- SQL Injection: Classified by Mitre with code CWE-89 [32]. This attack is caused by the lack of validation of the entries entered by the user, which are interpreted as part of the SQL query.
- Path Traversal: Classified by Mitre with code CWE-22 [32]. These vulnerabilities are exploited by entering user input of special characters such as ".." and "/". By not validating the entries, the use of these characters allows the application to escape from the restricted directory, where it is supposed to run, to other directories of the operating system.
- Cross-Site Scripting (XSS): Classified by Mitre with code CWE-79 [32]. This vulnerability occurs when the JavaScript code is injected, avoiding the Same Origin Policy, which states that scripts in a domain should not be able to access resources or execute code in a different domain. Code injection is possible due to the lack of validation of user inputs.
- Remote File Inclusion (RFI): Classified by Mitre with code CWE-98 [32]. This vulnerability allows the attacker to link to files located on other servers; thus, the attacker can obtain and run on untrusted code.
- Open Redirect: Classified by Mitre with code CWE-601 [32]. This vulnerability allows redirection to an external site through the data entered in a non-validated entry of a form, which can lead to different phishing attacks.

3.2 Test Cases

In the improvement of computer security research, it is essential to generate representative sets of test cases capable of efficiently revealing various vulnerabilities in an application. These sets of test

cases will typically be used in automated vulnerability scanning tools, allowing the generation and launch of a large number of attacks against a system to be audited. In this section, we will review different studies about the generation of test cases.

Shahriar et al. [33] proposed *MUSIC*, a tool that automatically generates mutants for applications written in JSP. This mutation-based testing approach uses nine mutation operators that inject SQL Injection Vulnerabilities (SQLIV) into the application source code. This approach forces the generation of an adequate test data set containing effective test cases capable of revealing SQLIV.

Gu et al. [34] proposed the generation of test data, mutating captured protocol messages and introducing regular expressions into the approach for constructing test case templates.

3.3 Metrics

- F-Score: The F-Score is also called F-measure or F1-Score; as can be seen in Eq. (1). It is the weighted harmonic mean of two measures: precision (P) and recall (R) [35].

$$F = \frac{1}{\alpha \frac{1}{P} + (1+\alpha)\frac{1}{R}} \tag{1}$$

  where the weighting factor

  $\alpha \in [0, 1]$

  F-Score balanced, assigns the same weights to precision and recall, which means that

  $$\alpha = \frac{1}{2}$$

  Therefore, the balanced F-Score formula can be written as shown in Eq. (2).

$$F = 2\frac{PR}{P+R} \tag{2}$$

- Precision (P): It is the ratio of the number of malicious payloads correctly detected divided by the number of total malicious payloads detected. This measure is also known as Positive Predictive Value (PPV). See Eq. (3).

$$P = \frac{TP}{TP+FP} \tag{3}$$

- Recall (R): As can be seen in Eq. (4), is the ratio of the number of malicious payloads correctly detected divided by the number of known total malicious payloads. Also called True Positive Rate (TPR) or Sensitivity.

$$R = \frac{TP}{TP+FN} \tag{4}$$

  The F-Score value is a compromise between Recall and Precision. It assumes values between the interval [0, 1]. The value of Recall is 0 if no malicious payload has been detected and will be 1 if all detected payloads are malicious and all malicious payloads have been detected.

- False Positive Rate (FPR): It is the ratio of incorrectly detected payloads to exploit a specific vulnerability associated with a non-vulnerable test case, divided by the number of payloads incapable of exploiting non-vulnerable test cases. In other words, FPR is the probability that a non-malicious payload is incorrectly detected as malicious (probability of false alarms). See Eq. (5).

$$FPR = \frac{FP}{FP + TN} \tag{5}$$

- Negative Predictive Value (NPV): As it can be inferred from Eq. (6), NPV is the ratio of payloads correctly detected to be incapable of exploiting a specific vulnerability associated with a non-vulnerable test case, divided by the sum of the number of payloads incapable of exploiting non-vulnerable test cases and the number of payloads detected incorrectly as unable to exploit a vulnerable test case. That is the confidence that can be assigned to the absence of alerts before the generation of a specific attack.

$$NPV = \frac{TN}{TN + FN} \tag{6}$$

- Accuracy: It is the ratio of payloads correctly identified divided by the total generated payloads. See Eq. (7).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7}$$

3.4 Metrics Discussion

As we intend to compare an objective *intragroup* (each tool of a particular category, compared to the other tools of the same category) and an objective *intergroup* (each tool of a particular category, compared to the other tools of another category) of various web protection tools, it is essential to have indexes that allow numerical ordering of the efficiency of the various tools. Bermejo Higuera [13] and Diaz et al. [36] proposed the use of *recall*, *precision* and *F-Score* metrics to analyze the results. This proposed approach was a concrete methodology for their evaluation.

Accuracy is one of the simplest methods to evaluate a binary classification model. It classifies the web request as *attack* or *no attack* and has advantages such as easily interpretable. However, its disadvantage is that it is not robust when the data is unevenly distributed, or where there is a higher cost associated with a particular type of error.

On the other hand, the F-Score allows to take into account the costs associated with FP and FN detection as in the case of web attack detection in different criticality scenarios. As the level of criticality increases, the costs associated with FP detection will be significantly lower. In addition, F-Score allows the correction of Acuracy deviations due to class imbalance, as in the case of a dataset in which 90% of web requests are malicious. If a classifier classifies all web requests as attack, it will automatically get 90% accuracy, but it would be useless in a real scenario.

Both Precision and Recall must be examined to fully evaluate the effectiveness of a model. These metrics are closely related. When the classification threshold increases, the number of FP decreases but FN increases. In such a case, Precision increases while Recall decreases. If the classification threshold decreases, FP increases and FN decreases, and accordingly, Precision decreases while Recall increases.

The F-Score is used to combine the Precision and Recall measures into a single value so as to easily compare the combined performance of Precision and Recall between various solutions. F-Score gives equal importance to Precision and Recall, as is the *harmonic mean* of Precision and Recall. As can be seen in Eq. (8), for certain scenarios with different levels of criticality, the more general formula of F-Score F_β can be used, where β is chosen such that Recall is considered β times as important as Precision:

$$F_\beta = \left(1 + \beta^2\right) \cdot \frac{Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall} \tag{8}$$

A common value of beta is 2, where recall has a weight 2 times higher than precision: in a high criticality scenario, it is desirable to minimize FN (undetected attacks) at the cost of increasing FP (normal requests detected as attacks).

3.5 WAF/RASP Tools to be Evaluated

The tools that we propose to analyze are detailed in Table 1.

Table 1: WAF and RASP tools to be evaluated

Tool	Technology	Commercial/Open source
ModSecurity	WAF	Open source
Imperva	WAF	Commercial
Fortify App. Defender	RASP	Commercial
Contrast	RASP	Commercial

3.5.1 ModSecurity

ModSecurity is a hybrid WAF that delegates part of its work on the web server. In the case of Apache ModSecurity delegates some of its roles through its integration with modules [37]. Apache performs the following functions:

- Decrypts SSL traffic.
- Splits the incoming connection flow into HTTP requests.
- Partially parses HTTP requests.
- Invokes ModSecurity, depending on the correct configuration context.
- Decompresses the bodies of HTTP requests as needed.
- In case Apache is configured as a reverse-proxy, it forwards the requests to the backend servers.

The functionality provided by ModSecurity could be summarized in the following points:

- Traffic parsing: The data that compose the web requests are examined by security analyzers that extract bits of data and store them for later use by the rules.
- Buffering: The request and response bodies are buffered, so that ModSecurity has access to the complete requests before passing them to the web application for processing. ModSecurity also has access to the complete responses before they are sent to the client. In this way, a reliable lock can be provided in the case of malicious traffic, although it is disadvantage includes requirement of an additional RAM for the storage of the request and response bodies.
- Full transaction log or audit log: This feature allows it to log all full HTTP traffic instead of logging only partial access log information. It records the headers and body of the request, as well as the headers and body of the response.
- Rules engine: It evaluates the transaction and carries out the actions defined in the rules in case the evaluated traffic coincides with any of the patterns defined in any of those rules.

Apache can be reconfigured to send the ModSecurity's error logs through Syslog onto a remote server acting as a Security Information and Event Management (SIEM). However, forwarded data are only the short version of the ModSecurity alert messages that appear in the Apache error_log file. To conduct proper incident response, the ModSecurity audit log files needs to be accessed [38].

3.5.2 Imperva

This is a cloud-based solution which provides characteristics such as load balancing, failover, Content Delivery Network (CDN) and DDoS protection. Imperva works by a change in the DNS of the applications to be protected, so that the traffic passes through the Imperva servers. In this way, all traffic is analyzed by the integrated WAF in the product, providing protection against the categories of OWASP Top Ten attacks. To maintain a holistic view of the security events produced, it allows the integration with different SIEM tools. It has a series of integrated WAF rules for protection against different types of attacks, allowing the user to create new rules through an editor. Imperva also provides a two-factor authentication available for any website without any changes in the application code and can implement virtual patching on many common applications such as WordPress, Joomla, Drupal, and others, to protect against known vulnerabilities and misconfiguration issues.

3.5.3 Fortify Application Defender

Fortify Application Defender is available as a SaaS solution or as an on-premise solution for applications developed in Java or .NET. It installs an agent (a jar file in the case of applications developed in Java, or an installable file in the case of applications developed in .NET) that is integrated in the application server to be protected.

This agent analyzes the requests made to the application and the response generated by the application. Depending on the configuration of the agent, it can monitor or block malicious requests.

The data are sent in real time to the servers of Fortify App. Defender through a web interface. The reports on the attack attempts that the application undergoes, detailing the type of attack in question, the severity of the attack, day and time of the attack, the request path, source IP address, status (monitor or protected), etc., will allow a more in-depth analysis of any malicious payload detected. The different categories of protective vulnerabilities can be easily activated or deactivated by setting them in monitor or protection mode.

All log data can be communicated to SIEM applications or to logs of compliance. The contextual perspective from the data flow of the applications and the execution logic allows the reconstruction of the malicious request and the traceability of the stack.

3.5.4 Contrast

It works by installing a jar or .NET file that is started with the application server to be protected, communicating with the "TeamServer," which is responsible for collecting and processing all information received. The contrast solution is available as a SaaS product or an on-premise product. It provides information about attacks produced in a web interface. It also provides extensive information about detected vulnerabilities, libraries in use and installed versions of these libraries, acting as an Interactive Application Security Testing (IAST) solution. The IAST solution uses an agent running on an application server or a library built into the code at compile time to create an instrumented version of the software. This can then detect behavior, indicating a vulnerability or an attack [39]. When a vulnerability is detected, it indicates the line of code in which it is possible to exploit the vulnerability, as well as indications to remedy the vulnerability, stack of calls, etc. As a RASP solution, it reports

exhaustively on the attacks. It indicates the blocked attacks and those that, although they have been made, have not been able to exploit the vulnerability.

3.6 Tools Configuration and Vulnerability Coverage

The only tool that could be configured is ModSecurity, because the implementation of the others is done by redirecting traffic to the provider's servers (case of Imperva) or by installing an agent (a ∗.jar or ∗.net file) in the application that will intercept the requests and responses of the application to protect, based on the policy marked from the tool's control panel (Fortify App. Defender and Contrast). The control panels of Imperva, Fortify App. Defender and Contrast are cloud-based. Because the only possible configurations in these control panels are the tool configuration in "monitor mode" or "block mode" and the activation and deactivation of different categories of vulnerabilities to be protected (SQLi, XSS, etc.), it has been opted to leave the configuration of all the tools analyzed in their default values (modifying only some ModSecurity rules to allow analysis through an automatic vulnerability scanning tool).

3.7 Attack Detection Methods

WAF and RASP tools are based on pattern analysis. WAF tools use syntactic analysis of requests and responses to check whether they match the patterns defined in their rules. This analysis is done in the application layer, without direct access to the logic of the application. In comparison, the RASP tools install an agent in the application to be protected and, thus, have access to direct interception of calls and responses inside the application. When the logic of the application, calls and responses are know, they can be compared more effectively against a system of rules.

Contrast makes use of bytecode instrumentation, a feature in Java used to help integrate programs and application features during development. It embeds an agent into an application to be protected, which will, thereafter, be directly monitored and protected from the inside out. With that agent, Contrast can directly access the application requests and responses and can easily perceive exploits and unknown threats. Owing to the instrumentation approach, Contrast can observe how an application responds when it is free of attacks, being able to gather as much information as possible before deciding to block an attack. This amount of information makes a difference when accurately detecting the different types of attacks.

The use of advanced data labeling is possible due to the bytecode instrumentation [40]. This allows the tracing of all the data that enters the application from an untrusted source. The data in an HTTP header can be labeled as "untrusted," while the data in an HTTP parameter can be labeled "cross site" because an attacker can send it through domains. Initially, these labels are applied to the complete information. The tags are updated each time the data is modified or examined. This data flow tracking only works for the most common injection-type vulnerabilities, such as SQL injection, XSS, LDAP injection, XPath injection, command injection, etc.

A summary of each tool characteristics can be seen in Table 2.

Table 2: Tools characteristics

	WAF tools		RASP tools	
	ModSecurity	Imperva	Fortify App. Defender	Contrast
How traffic is analyzed	Reverse proxy	Redirecting traffic via DNS changes	Installing an agent in the application	Installing an agent in the application
Attack detection	Comparison of patterns against a system of rules.	Comparison of patterns against a system of rules.	Comparison of patterns against a system of rules.	Comparison of patterns against a system of rules.
Where traffic analysis is done	Applic. layer	Applic. layer	Inside the application	Inside the application
Has access to the application logic	No	No	Yes	Yes
Makes use of anomaly detection	No	No	Unknown	Yes. Through bytecode instrumentation
Advanced tagging	No	No	No	Yes. Through bytecode instrumentation

4 Experimental Methodology

Since no test suite is specifically designed for the analysis of WAF and RASP solutions, the following methodology has been developed to allow the analysis of the different tools evaluated in the present work:

(i) Selection of categories of vulnerabilities against which protection tools are to be evaluated.

(ii) Selection of a benchmark with different test cases that serves as reference for the evaluation of the tools. The OWASP Benchmark[1] project was chosen for evaluating tool performance in the following vulnerability categories: command injection, SQL injection, path traversal, cross-site scripting. For the RFI and open redirect categories, the Web Application Vulnerability Scanner Evaluation Project (WAVESEP)[2] project was selected for evaluating tool performance.

(iii) As discussed in Section 3.5, the ModSecurity, Imperva, Fortify App. Defender and Contrast tools were selected.

(iv) Selection of the vulnerability scanner from which automated attacks against the benchmark will be launched. We have chosen the OWASP ZAP[3] tool.

[1] https://owasp.org/www-project-benchmark/.
[2] https://github.com/sectooladdict/wavsep.
[3] https://owasp.org/www-project-zap/.

(v) Selection of vulnerable and non-vulnerable cases of the different vulnerability categories used to measure, respectively, True Positive (TP) and FP. A total of 238 vulnerable test cases and 134 non-vulnerable test cases were selected and broken down into six different vulnerability categories.

(vi) Generation of attacks from OWASP ZAP against the selected test cases of the two benchmarks, without any protection tool interposed.

(vii) Selection of payloads that generate an alert in OWASP ZAP (in the case of vulnerable test cases) and payloads that do not generate alerts in OWASP ZAP (in the case of non-vulnerable test cases). In this way, we identify payloads that generate alerts (TP) in vulnerable test cases and, on the other hand, payloads that do not generate alerts (True Negative-TN) in non-vulnerable test cases. In total, there were 1,341 payloads: 684 TP and 657 TN.

(viii) Interposition of the protection tool to be evaluated between the benchmark and OWASP ZAP.

(ix) Generation of attacks from OWASP ZAP against the selected test cases of the benchmark with the interposing tool.

(x) Revision of the alerts generated in OWASP ZAP and in the log's files of the protection tool in the same payloads selected in point (vii) and obtaining the indicators of TP, TN, FP and FN.

(xi) Tabulation of results and obtaining metrics, ranked using F-Score index.

An example of payload can be seen in Fig. 1, while a flowchart with the different steps to analyze the tools is shown in Fig. 2.

SQLi	POST	http://192.168.0.181/benchmark/sqli-00/BenchmarkTest00024	BenchmarkTest00024=bar%27%29+AND+3085%3DCAST%28%28CHR%2858%29%7C%7CCHR%28105%29%7C%7CCHR%28122%29%7C%7CCHR%28109%29%7C%7CCHR%2858%29%29%7C%7C%28SELECT+%28CASE+WHEN+%283085%3D3085%29+THEN+1+ELSE+0+END%29%29%3A%3Atext%7C%7C%28CHR%2858%29%7C%7CCHR%28110%29%7C%7CCHR%28114%29%7C%7CCHR%28117%29%7C%7CCHR%2858%29%29+AS+NUMERIC%29+AND+%28%27XUVt%27%3D%27XUVt&foo=bar

Figure 1: A payload example

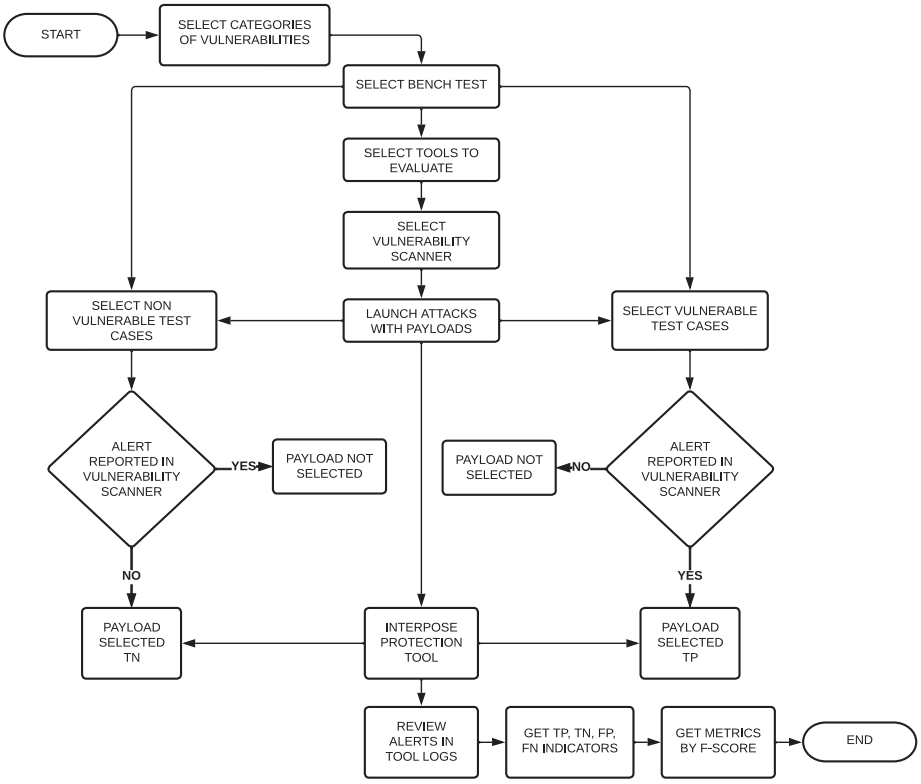

Figure 2: Steps to analyze the tools

5 Results

This section presents the results obtained by each of the evaluated solutions. The TOTAL row of each table was calculated based on the weighted average (excluding the extreme values[4]) of the results of each index of each category of vulnerabilities. Such an approach eliminated the influence caused by the different number of payloads of each category.

5.1 ModSecurity

As shown in Table 3, all categories of vulnerabilities show a high percentage of false positives, specifically in the SQL Injection and Cross-Site Scripting categories. ModSecurity systematically blocks all requests having the same number of TPs and FP, thereby causing the False Positive Rate (FPR) value to be 1. In the case of a high FN rate and a low TN rate, the value of the Negative Predictive Value is the lowest of all evaluated solutions (0.428). It also has a low level of global precision (0.507), indicative of the high number of FP. The Recall index is affected in different categories due to the number of FN, which affects the F-Score index, whose overall score is 0.541.

[4]Except in cases where it is impossible to eliminate the extremes due to the equality of results; in those cases, the weighted average will be calculated without the exclusion of extreme values.

Table 3: Results obtained by ModSecurity

Vuln.	#Payl.	#TP	#FP	#TN	#FN	Precision	Recall	FPR	NPV	Accuracy	F1-Score
Comm. injec.	156	52	52	26	26	0.5	0.667	0.667	0.5	0.5	0.571
SQLi	400	200	200	0	0	0.5	1	1	0	0.5	0.667
Path trav.	391	113	114	81	83	0.498	0.577	0.585	0.494	0.496	0.534
RFI	116	31	42	17	26	0.425	0.544	0.712	0.395	0.414	0.477
Redirect	156	33	28	36	59	0.541	0.359	0.438	0.379	0.442	0.431
XSS	122	61	61	0	0	0.5	1	1	0	0.5	0.667
Total	1341	490	497	160	194	0.507	0.655	0.684	0.428	0.487	0.541

5.2 Imperva

The results obtained by Imperva are detailed in Table 4. The Precision, Recall, FPR and F-Score in the Remote File Inclusion and Redirect categories were 0 as these categories did not contain any malicious payload. Specifically, the Precision index of this tool is the lowest of all evaluated, reaching an overall value of 0.501. The other categories contain a high number of FPs, and the categories of Path Traversal and Cross-Site Scripting have FPR as 1. The overall value of F-Score stands at 0.645.

Table 4: Results obtained by imperva

Vuln.	#Payl.	#TP	#FP	#TN	#FN	Precison	Recall	FPR	NPV	Accuracy	F1-Score
Comm. injec.	156	52	52	26	26	0.5	0.667	0.667	0.5	0.5	0.571
SQLi	400	191	190	10	9	0.501	0.955	0.95	0.526	0.503	0.657
Path trav.	391	196	195	0	0	0.501	1	1	0	0.501	0.668
RFI	116	0	0	59	57	0	0	0	0.509	0.509	0
Redirect	156	0	0	64	92	0	0	0	0.41	0.41	0
XSS	122	61	61	0	0	0.5	1	1	0	0.5	0.667
Total	1341	500	498	159	184	0.501	0.897	0.894	0.47	0.48	0.645

5.3 Fortify Application Defender

Table 5 presents the results obtained by Fortify Application Defender. No attacks were detected in the categories of Command Injection, Path Traversal and RFI. The overall Precision index of this tool was at 0.825. In the SQL Injection category, the correct detection rate TP and TN was 100%, reaching the value 1 in the Precision, Recall, Negative Predictive Value, Accuracy and F1-Score indices. It reaches a high number of TN; however, the high number of FN detected penalizes its score in the Negative Predictive Value index, which was 0.595. Its percentage of successes (Accuracy) was 0.582, and its F-Score was 0.777.

Table 5: Results obtained by fortify app. defender

Vuln.	#Payl.	#TP	#FP	#TN	#FN	Precision	Recall	FPR	NPV	Accuracy	F1-Score
Comm. injec.	156	0	0	78	78	0	0	0	0.5	0.5	0

(Continued)

Table 5 (continued)

Vuln.	#Payl.	#TP	#FP	#TN	#FN	Precision	Recall	FPR	NPV	Accuracy	F1-Score
SQLi	400	200	0	200	0	1	1	0	1	1	1
Path trav.	391	0	0	195	196	0	0	0	0.499	0.499	0
RFI	116	0	0	59	57	0	0	0	0.509	0.509	0
Redirect	156	62	0	64	30	1	0.674	0	0.681	0.808	0.805
XSS	122	59	39	22	2	0.602	0.967	0.639	0.917	0.664	0.742
Total	1341	321	39	618	363	0.825	0.803	0.058	0.595	0.582	0.777

5.4 Contrast

The results obtained by Contrast are detailed in Table 6. Due to the complete absence of FP, its Precision index was set to 1 and the False Positive Rate value was 0. In assition, the Recall index was 0.706 and its F-Score was 0.810 because of the relatively low number of FN and the high number of TPs. The Negative Predictive Value reaches the value of 0.764 due to the correct identification of TN and the low number of FN. The percentage of hits is around 83%, while the accuracy is at 0.832.

Table 6: Results obtained by contrast

Vuln.	#Payl.	#TP	#FP	#TN	#FN	Precision	Recall	FPR	NPV	Accuracy	F1-Score
Comm. injec.	156	13	0	78	65	1	0.167	0	0.545	0.583	0.286
SQLi	400	183	0	200	17	1	0.915	0	0.922	0.958	0.956
Path trav.	391	187	0	195	9	1	0.954	0	0.956	0.977	0.977
RFI	116	16	0	59	41	1	0.281	0	0.590	0.647	0.438
Redirect	156	46	0	64	46	1	0.5	0	0.582	0.705	0.667
XSS	122	42	0	61	19	1	0.689	0	0.763	0.844	0.816
Total	1341	487	0	657	197	1	0.706	0	0.764	0.832	0.81

5.5 Summary of Results and Comparison of Tools

Table 7 shows the different solutions sorted in descending order according to the score obtained by each of them in the F-Score index. In this index, the ordering was chosen because it indicates the relationship between Precision and Sensitivity (Recall) such that it objectively shows the relationship between TP, FP and TN.

This table also represents other indexes that provide additional information on the behavior of each solution. However, the FPR index has been modified in a way that its interpretation in the set of other indexes presented is more understandable by the reader. The FPR indicates the probability of false alarms so that the smaller its value is, the better is the performance of the solution evaluated, i.e., the value of the index is inversely proportional to the performance of the solution. Since the other five indexes presented have a directly proportional relationship between the value of each of the indexes and the performance of the solution, and that the possible values of each of them are in the interval [0, 1], the value of the complement of 1 according to the FPR was chosen to represent it in the tables. If, for example, the actual value of the FPR is 0.059, the value will be $1-0.059 = 0.941$.

Table 7: Results obtained by the different tools evaluated sorted by F-Score

Tool	F-Score	Precision	Recall	FPR	NPV	Accuracy
Contrast	0.810	1	0.706	1	0.764	0.832
Fortify App. Defender	0.777	0.825	0.803	0.942	0.595	0.582
Imperva	0.645	0.501	0.897	0.106	0.47	0.484
ModSecurity	0.541	0.507	0.655	0.316	0.428	0.487

ModSecurity and Imperva present the same results in several of the analyzed indexes (Precision, Accuracy and NPV), probably because request is intercepted before its arrival to the application. The comparison of the request against a set of established rules and standards, without the possibility of analyzing the calls to the system made by the application, may cause it to generate a high number of FP, which penalizes its score in Precision and, therefore, in F-Score. Imperva obtains a better score on the F-Score index (0.645) because of its high score on the Recall index (0.897). Obviously, the results obtained by WAF solutions will improve once the specific configuration applied to each solution has been modified depending on the environment and the application to be protected.

Apart from the ease of use, configuration and complementary services offered by the Imperva solution (load balancing, request accelerator, etc.), no significant differences are seen between Imperva and ModSecurity.

Contrast achieves a score of 1 in Precision, highlighting in all other indexes with respect to the other evaluated solutions, obtaining the highest score in F-Score (0.81).

Fortify App. Defender has high values in Precision (0.825) and FPR (0.942) but its score in the NPV and Accuracy indexes is penalized due to the high number of FN. However, it is still above the scores obtained by the WAF solutions. Fortify App. Defender can be considered a good option in the protection of specific vulnerabilities like SQLi, Redirect and XSS.

Compared to most security tools that suffer from an excess of FP and must be manually reviewed to rule out or confirm an attempt to exploit a particular vulnerability, RASP solutions present a clear superiority in Precision and almost total absence of FP. Hence, RASP solutions have a high degree of accuracy.

Fortify App. Defender and Contrast limit the creation of customized advanced rules by the user. Imperva allows the creation of custom rules through a proprietary scripting language. ModSecurity, on the other hand, offers the most configuration options since it allows access to the rules files, creation of new rules, etc. Obviously, the detection ratio of ModSecurity and Imperva would improve significantly when custom rules are added to these tools to adapt the execution environment of the application to protect. However, it would be a disadvantage for the other two tools to be evaluated, for this reason. Thus, it has been decided to leave all the tools in their default configuration.

By installing the Contrast agent in the application to be protected, instrumentation is added to identify vulnerabilities in the source code and subsequently detect attacks and block them from within the applications. With the interception of calls and responses from within the application, the correct attack detection rate increases significantly. RASP technology has a detailed view into the actions of the system and can help improve security accuracy. For example, RASP has insight into application logic, configuration, and data and event flows, which means it can detect attacks with high accuracy.

Contrast differentiates between attacks capable of exploiting a vulnerability (blocked) and attacks that have not been able to exploit a vulnerability (ineffective).

The tools as well as their results could be improved using machine-learning, decision trees, etc. in order to provide feedback to them according to the execution environment, thus, minimizing the detection rate of FP and FN increasing their efficiency. With ModSecurity, it is relatively easy to implement this improvement; regarding the other tools analyzed, it would be necessary for the manufacturer to provide more configuration options to the user, as well as the possibility of providing feedback to the tool (ideally through machine-learning techniques but also allowing the user to manually mark an event as FP or FN).

In Table 8, the ease of use, configurability, services offered and other features of each tool are compared.

Table 8: Tools' features

	WAF tools		RASP tools	
	ModSecurity	Imperva	Fortify App. Defender	Contrast
Installation	Hard. Manual installation	Ease. Changing DNS in order to redirect traffic to Imperva servers	Medium. Need to install java/net agent	Medium. Need to install java/net agent
Configurability	Full. Through configuration files	Limited. Through control panel	Limited. Through control panel	Limited. Through control panel
Control panel	No	Cloud	Cloud	Cloud
Tool in monitor/block mode	Modifying configuration file	Through control panel	Through control panel	Through control panel
Advanced custom rules	Through rules files. Sec-Rules language. Full control	Imperva rules proprietary scripting language. Advanced control	Point-wise protection. Limited control	Virtual patches. Limited control
Categories of vulnerabilities to protect	Modifying configuration file	Through control panel	Through control panel	Through control panel
Events analysis	Hard. No web interface for analysis	Easy. Through control panel	Easy. Through control panel	Easy. Through control panel
Common vulnerabilities and exposures (CVE) shields	No	No	No	Yes

(Continued)

Table 8 (continued)

	WAF tools		RASP tools	
	ModSecurity	Imperva	Fortify App. Defender	Contrast
Interactive application security testing (IAST)	No	No	No	Yes
Load balancer	No	Yes	No	No
Content delivery network (CDN)	No	Yes	No	No

6 Conclusions and Future Work

This paper compared different WAF and RASP protection solutions, using the OWASP ZAP vulnerability scanner against several test cases selected from two web applications. These two applications were considered benchmarks (OWASP Benchmark and Wavsep), interposing each of the solutions to be evaluated, in order to analyze their results. No additional configuration were made to the evaluated WAF-RASP solutions, except for the purpose of benchmark analysis by the vulnerability analysis tool OWASP ZAP.

The results and their analysis confirm the real degree of efficiency of the tools, obtaining precise TP, FP and FN ratios. Complementary metrics such as Accuracy, F-Score, NPV and ACC were considered to establish different rankings of the tools: TP, FP, TN, FN together. More specifically, the FPR ratio of each tool showed that the RASP tools have hardly any false positives, while the WAF tools obtain higher results between 0.68 and 0.89. Thus, the RASP tools are confirmed to be more accurate due to their white box nature.

Since there is a lack of an established methodology that specifies the steps to be followed to evaluate web application protection tools, the proposed methodology described in Section 4 can be considered for use in future research works. This methodology can be adapted on new requirements for evaluation of any protection tool against any benchmark.

It is necessary to analyze the performance of new web protection tools (e.g., AppWall) against new vulnerabilities and constantly appearing attacks (e.g., log4j). The use of the web tool evaluation methodology, defined in this work, is suitable as a starting point to achieve this goal.

The development of a feedback system for ModSecurity should be considered as part of future research work. There is evidence of a project called ModProfiler, currently discontinued and with very few references, which could be a starting point for it [41].

Acknowledgement: The authors wish to express their appreciation to the reviewers for their helpful suggestions which greatly improved the presentation of this paper.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Imperva (2015). WAAR 2015. Technical Report. Imperva. https://www.imperva.com/docs/HII_Web_Application_Attack_Report_Ed6.pdf.
2. Symantec (2016). Internet Security Threat Report. Technical Report. Symantec. https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf.
3. WhiteHat Security (2016). Web Applications Security Statistics Report 2016. Technical Report. WhiteHat Security. https://info.whitehatsec.com/rs/675-YBI-674/images/WH-2016-Stats-Report-FINAL.pdf.
4. OWASP (2021). OWASP Top 10 2021. https://owasp.org/Top10/.
5. Clincy, V., Shahriar, H. (2018). Web application firewall: Network security models and configuration. *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, pp. 835–836. Tokyo, Japan.
6. Appelt, D., Nguyen, C. D., Panichella, A., Briand, L. C. (2018). A Machine-learning-driven evolutionary approach for testing web application firewalls. *IEEE Transactions on Reliability, 67(3),* 733–757. DOI 10.1109/TR.24.
7. OWASP (2022). SQL Injection Bypassing WAF-OWASP. https://www.owasp.org/index.php/SQL_Injection_Bypassing_WAF.
8. Garn, B., Sebastian Lang, D., Leithner, M., Richard Kuhn, D., Kacker, R. et al. (2021). Combinatorially xssing web application firewalls. *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 85–94. Porto de Galinhas, Brazil.
9. Gartner (2022). Runtime Application Self-Protection (RASP)-Gartner IT Glossary. http://www.gartner.com/it-glossary/runtime-application-self-protection-rasp.
10. Dubey, S. K. (2016). An evaluation of java applications using security requirements. *International Journal of Recent Trends in Engineering & Research Issue, 2(8),* 2455–1457.
11. Steiner, S., de Leon, D. C., Alves-Foss, J. (2017). A structured analysis of SQL injection runtime mitigation techniques. *Proceedings of the Annual Hawaii International Conference on System Sciences,* vol. 2017, pp. 2887–2895. Hawaii.
12. Nicolett, M. (2005). How to Develop an Effective Vulnerability Management Process. Technical Report ID Number: G00124126. Gartner. https://docplayer.net/5943695-How-to-develop-an-effective-vulnerability-management-process.html.
13. Bermejo Higuera, J. R. (2013). *Metodología de evaluación de herramientas de análisis automático de seguridad de aplicaciones web para su adaptación en el ciclo de vida de desarrollo* (*Ph.D. thesis*). Universidad Nacional Educación a Distancia (UNED). http://e-spacio.uned.es/fez/eserv/tesisuned:IngInd-Jrbermejo/BERMEJO_HIGUERA_Juan_Ramon_Tesis.pdf.
14. Byrne, P. (2006). Application firewalls in a defence-in-depth design. *Network Security, 2006(9),* 9–11. DOI 10.1016/S1353-4858(06)70422-6.
15. Schmitt, I., Schinzel, S. (2012). *Waffle: Fingerprinting filter rules of web application firewalls. 6th USENIX Workshop on Offensive Technologies (WOOT 12)*, Bellevue, WA, USENIX Association.
16. Holm, H., Ekstedt, M. (2013). Estimates on the effectiveness of web application firewalls against targeted attacks. *Information Management & Computer Security, 21(4),* 250–265. DOI 10.1108/IMCS-11-2012-0064.
17. Moosa, A. (2010). Artificial neural network based web application firewall for sql injection. *International Journal of Computer, Electrical, Automation, Control and Information Engineering, 4(4),* 12–21.
18. Auxilia, M., Tamilselvan, D. (2010). Anomaly detection using negative security model in web application. *2010 International Conference on Computer Information Systems and Industrial Management Applications (CISIM)*, pp. 481–486. Krakow, Poland.
19. Applebaum, S., Gaber, T., Ahmed, A. (2021). Signature-based and machine-learning-based web application firewalls: A short survey. *Procedia Computer Science, 189,* 359–367. DOI 10.1016/j.procs.2021.05.105.

20. Weyns, D., Iftikhar, M. U., Malek, S., Andersson, J. (2012). Claims and supporting evidence for self-adaptive systems: A literature study. *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pp. 89–98. Zurich, Switzerland.
21. Yuan, E., Malek, S. (2012). A taxonomy and survey of self-protecting software systems. *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 109–118. Zurich, Switzerland.
22. Yuan, E., Esfahani, N., Malek, S. (2014). A systematic survey of self-protecting software systems. *ACM Transactions on Autonomous and Adaptive Systems, 8(4),* 17. DOI 10.1145/2555611.
23. Keromytis, A. D. (2013). Characterizing software self-healing systems. *Communications in Computer and Information Science, 374,* 22–33.
24. Lane, A. (2016). Understanding and Selecting RASP: Technology Overview. https://securosis.com/blog/understanding-and-selecting-rasp-technology-overview.
25. Haldar, V., Chandra, D., Franz, M. (2005). Dynamic taint propagation for java. *21st Annual Computer Security Applications Conference (ACSAC'05)*, vol. 2005, pp. 303–311. Tucson, Arizona, IEEE.
26. Zeller, S., Khakpour, N., Weyns, D., Deogun, D. (2020). Self-protection against business logic vulnerabilities. *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 174–180. New York, NY, USA, Association for Computing Machinery.
27. Yin, Z., Li, Z., Cao, Y. (2018). *A web application runtime application self-protection scheme against script injection attacks*. In: Sun, X., Pan, Z., Bertino, E. (Eds.), *Cloud computing and security*, pp. 566–577. Cham: Springer International Publishing.
28. Yang, W., Peng, J. (2020). Research on evm-based smart contract runtime self-protection technology framework. In: Barolli, L., Amato, F., Moscato, F., Enokido, T., Takizawa, M. (Eds.), *Web, artificial intelligence and network applications*, pp. 617–627. Cham: Springer International Publishing.
29. Web Application Security Consortium (2022). The Web Application Security Consortium/Web Application Firewall Evaluation Criteria. http://projects.webappsec.org/w/page/13246985/WebApplicationFirewallEvaluationCriteria.
30. Williams, J. (2015). Protection from the Inside: Application Security Methodologies Compared. Technical Report. SANS Institute. https://techbeacon.com/sites/default/files/gated_asset/sans-application-security-methodologies-compared.pdf.
31. Razzaq, A., Hur, A., Shahbaz, S., Masood, M., Ahmad, H. F. (2013). Critical analysis on web application firewall solutions. *2013 IEEE Eleventh International Symposium on Autonomous Decentralized Systems (ISADS)*, pp. 1–6. Mexico City, Mexico.
32. Mitre (2022). CWE-Common Weakness Enumeration. https://cwe.mitre.org/index.html.
33. Shahriar, H., Zulkernine, M. (2008). Music: Mutation-based SQL injection vulnerability checking. *The Eighth International Conference on Quality Software*, pp. 77–86. Oxford, UK.
34. Gu, S., Li, W., Zhao, X. (2011). Brute force vulnerability testing technology based on data mutation. *2011 IEEE Vehicular Technology Conference (VTC Fall)*, pp. 1–6. San Francisco, CA, USA.
35. Zhang, E., Zhang, Y. (2009). F-measure. *Encyclopedia of Database Systems, 1147*. DOI 10.1007/978-0-387-39940-9_483.
36. Díaz, G., Bermejo, J. R. (2013). Static analysis of source code security: Assessment of tools against samate tests. *Information and Software Technology, 55(8),* 1462–1476. DOI 10.1016/j.infsof.2013.02.005.
37. Ristic, I. (2010). *ModSecurity handbook*. UK: Feisty Duck.
38. ModSecurity Project (2022). ModSecurity Frequently Asked Questions (FAQ). https://github.com/SpiderLabs/ModSecurity/wiki/ModSecurity-Frequently-Asked-Questions-(FAQ).
39. Lemos, R. (2016). 4 fundamental application security testing tools best practices. https://techbeacon.com/-4-best-practices-application-security-testing-tools.

40. Contrast Security (2013). Better Application Vulnerability Detection with Advanced Data Tagging. https://www.contrastsecurity.com/security-influencers/better-application-vulnerability-detection-with-contrasts-advanced-data-tagging.
41. Ristic, I. (2008). Modprofiler: Defending Web Applications from 0-day Attacks. https://www.blackhat.com/presentations/bh-usa-08/Ristic_Shezaf/BH_US_08_No_More_Signatures_Ivan_Ristic_Ofer_Shezaf.pdf.