IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON EMERGING APPROACHES TO CYBER SECURITY

Received December 27, 2020, accepted January 6, 2021, date of publication January 11, 2021, date of current version January 20, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3050566

Effective Filter for Common Injection Attacks in Online Web Applications

SANTIAGO IBARRA-FIALLOS[1], JAVIER BERMEJO HIGUERA[1], MONSERRATE INTRIAGO-PAZMIÑO[2], JUAN RAMÓN BERMEJO HIGUERA[1], JUAN ANTONIO SICILIA MONTALVO[1], AND JAVIER CUBO[1]

[1]Escuela Superior de Ingeniería y Tecnología, Universidad Internacional de La Rioja, 26006 Logroño, Spain
[2]Departamento de Informática y Ciencias de la Computación, Escuela Politécnica Nacional, Quito 170450, Ecuador

Corresponding author: Javier Bermejo Higuera (javier.bermejo@unir.net)

ABSTRACT Injection attacks against web applications are still frequent, and organizations like OWASP places them within the Top Ten of security risks to web applications. The main goal of this work is to contribute to the community with the design of an effective protection of web applications against common injection attacks. Our proposal is a validation filter of input fields that is based on OWASP Stinger, a set of regular expressions, and a sanitization process. It validates both fundamental characters (letters, numbers, dot, dash, question marks, and exclamation point) and complex statements (JSON and XML files) for each field. The procedure of deploying the proposed filter is detailed, specifying the sections and contents of the configuration file. In addition, the infrastructure for running the tests is described, including the setting of an attack tool, and the implementation of a controller. The attack tool is used as a security scanner for common injection attacks, and the controller is developed for routing the requests in two steps; first a request is addressed to the filter, and if it is valid, it will redirect to the web application itself. The proposal filter has been tested on three public as well as on a real private web application. An accuracy of 98,4% and an average processing time of 50 ms are achieved, based on wich it is possible to conclude the proposed filter is highly reliable and does not require additional computational resources.

INDEX TERMS Information security, input validation, software security, regular expression, sanitization.

I. INTRODUCTION

In general, software development is challenging when it comes to creating secure and reliable software. As the programmer gains experience, he or she becomes aware of the value of secure programming, best practices, as well as the importance of protection against intrusions. For example, data manipulations, among others, who depend on the sensitivity of the information handled by an application, may have a greater or lesser impact.

According to the OWASP Top Ten Web Application Security Risks, the injection vulnerability is located at the beginning of the list, and it could be executed by internal and external users to an organization [1]. This establishes a point of reference and an important warning to software developers. While developing a web application programmers should change the tendency to be more concerned with functional validation and give less value to secure programming. It is also known that there are web applications that are vulnerable to the most common injection attacks. These applications can not be rewritten. However, to stay online, these applications should be protected.

The goal of this article is to contribute to reducing the injection attacks through the design of a new filter based on OWASP Stinger [2], and set as a module on a Jboss/Wildfly application server [3]. Consequently, it can be invoked by all the web applications deployed on the server.

The filter helps web applications to create at least one layer of protection against common injection attacks (SQL Injection (SQLi), Command Injection (CI), Cross Site Scripting (XSS), etc.), which occur when entries are not validated. Therefore, this article focuses on the analysis of vulnerabilities that can be mitigated by the validation and sanitization of input parameters in web applications. At the end of this proposal the generic filter in combination with OWASP framework could mitigate common injection attacks with 98.4% accuracy. Summing up, the main contribution of this paper is to design a simple, fast, and highly reliable filter

The associate editor coordinating the review of this manuscript and approving it for publication was Luis Javier Garcia Villalba.

for stopping common injection attacks in web applications through a set of rules based on several regexes and other decisive settings.

The remainder of the paper is organized as follows. Section II presents an analysis of related proposals. In Section III essential considerations about the filter design are described. Section IV details filter development and deployment. Section V provides testing, results, and discussion of the filter performance. Finally, some conclusions and future works are presented in Section VI.

II. RELATED WORKS

A. BACKGROUNG

Nowadays, web applications, given their exposure to the Internet, are frequently targeted by attackers exploiting critical vulnerabilities. This has increased efforts to mitigate vulnerability attacks. The work in [5] presents the classification of web applications attacks into six categories: command execution attack, information disclosure, logical attacks, authentic based attack, authorization based, and client based attack. Figure 1 shows the above-mentioned classification:

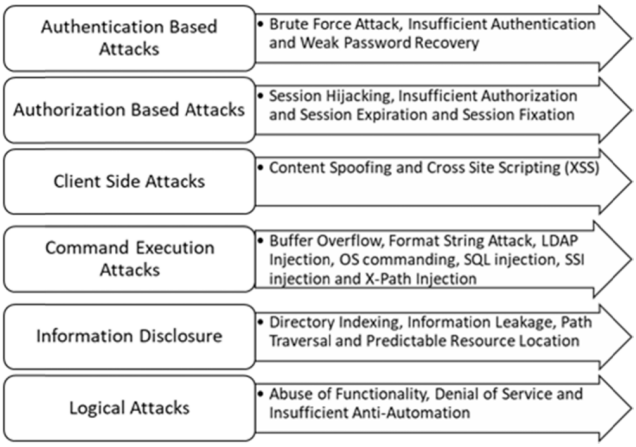


FIGURE 1. Classification of web attacks [5].

In the following, we present an analysis of related works. At the end of the section, we includes some comparison tables of our proposal as regards some of the analysed works.

According to Barabanov *et al.* [6], the most successful attacks are: XSS, Cross-site request forgery (CSRF), enhancement of privileges related to circumvention of security functions, denial of service attacks, disclosure of critical software information in error messages and SQLi.

In the reference document [7], an empirical study of more than 7000 vulnerabilities related to input validation is carried out depending on the programming language used for the web application development. In this experiment, seventy eight open source web application frameworks are analyzed for several web programming languages, including: PHP, Perl, Python, Ruby,.NET, and Java. The results show that almost 20% of the frameworks do not provide any validation functionality at all. In fact, only thirty seven studies provided support for complex input data validation.

When it comes to the validation, sanitization of the parameters and verification of the headers of entry in the web applications, guides that allow us to carry out this work in an effective way are available.

In their research work, Scott and Richard [8] propose a generic methodology for web application protection, composed of a descriptor for each one of the parameters sent from the client, whose control location must be between the client and the protection application. The theory presented in that work, is also applied as an additional protection guideline for each application in our research work. The main difference is in the proposal of a primary generic defense line not specific to an application, so, it is not necessary to define each of the parameters entered to achieve this.

Boyd and Keromytis [9], explain the concept of random instructions included in a proxy located before a database engine. This system, named SQLrand, has the functionality to change the SQL instructions generated in a random way (for instance, language), in standard SQL sentences for each one of the database engines. This randomness in language makes it impossible for attackers to clearly identify the attack vectors against the database engine, which prevents SQL injection problems. Using a randomized SQL query language it is possible to detect and abort queries that include injected code.

Wassermann and Su [10], present an input parameter static analyzer, which looks for attack patterns especially SQL injection. It has not yet been tested in the field but the first results have yielded good theoretical results.

Ismail *et al.* [11], propose having a local proxy on the client side, which allows intercepting requests and responses that are exchanged with the Web Server. This local proxy would be in charge of stopping XSS attacks, with the particularity that the load of this work would be assumed by the client computer and by the server side.

The work of Buehrer *et al.* [12], presents a validation option against SQL injection attacks. It consists of carrying out a syntactic analysis tree of each of the SQL sentences that are expected to be executed in a given application. Its structure is stored and compared with the resulting sentence after injecting the input parameters. In case the structure tree does not coincide with the stored one it is almost certain that an attack is being faced and therefore the query is rejected and will never be executed. The authors assert that ''it is more effective to measure the results of the input than to attempt to validate the input before inserting it into the proposed query.''

In their work, Liu and Tan [13] define the paths of the data input in the code through evaluation states called nodes, from which test cases can be generated, that can indicate if any input data might alter the normal running of a program. This research work is directed towards an early correction of vulnerabilities in the input data (during the implementation of the applications), to able to serve as a computer auditing tool. However, their research does not provide data on the

effectiveness of the proposed method for avoiding vulnerabilities in the input data.

In Park and Park [14], WAIDS (Web Application Intrusion Detection System) is presented. That system analyzes user entries, comparing them against predefined normal entry profiles and focusing on profiles different from those expected, whereby which detecting possible attacks and stopping them. The authors have managed to distinguish data entry profiles quite well to such an extent that false positives are significantly reduced.

In Aljawarneh *et al.* [15], a semantic data validator is created as a service for web applications, placing markers in the fields of the forms that define maximum length, minimum length, accepted characters and data type. Those markers are used as criteria to accept or reject the data entered in the form. This semantic data validator consists of an RDF (Resource Description Framework) for web page annotations, an HTTP interceptor, a data extraction RDF, a text analysis RDF, and a validator module. The results of this validator provide detection and prevent most web application attacks, much needed, according to the authors, to improve effectiveness and performance.

The authors of Scholte *et al.* [16], present another alternative for input validation, through a prototype developed in PHP, with principles of data type self-learning and known validation vector application, rendering 83% effectiveness against known SQL injection attacks and 65% in XSS attacks. That development was named as IPAAS (Input Parameter Analysis System). In order to fulfill its objective, it consists of 3 phases that are: extraction of the input parameters, ascertainment of the type of data within the identified parameters, and finally, application of known vectors against the 100 known vulnerabilities for SQL and XSS injection.

An analysis of web filtering rules is performed in [17]. Authors propose an improved method to detect only XSS vulnerabilities based on the attack vector designed and implemented in the application layer. However, that is not a protection filter.

A filtering API for XSS attacks based on white lists is proposed in Dalai *et al.* [18]. It filters server responses rather than user input, so it requires no modifications on the client side. Since it is implemented on the server side, it will only detect and block server side XSS attacks. It will not mitigate DOM-based XSS attacks.

The OWASP project [1], in its description of injection attacks, gives a general idea, and indicates that "injection prevention requires keeping data separate from commands and queries". This general idea encompasses the usage of Secure APIs, whitelisting, and other length controls and SQL syntax. Besides, there are tools on the market that offer configurations to create a filter, among them are:

- Datapower IBM Support [19], [20]: IBM SOA integrator hardware, which brings XML files with patterns against SQL injection, XSS, as well as protection against denial of service attacks. It can be worked with the default XSL filter or create one to measure, reading the XML patterns, as appropriate.
- PHP fllter_var Brady [21]: it uses a PHP filter_var function which, under the security parameters for input validation protection described above, allows web applications developed in PHP to have a means of protection against injection and XSS attacks.
- Spring 3 Validation Johnson *et al.* [22]: it is one of the most popular frameworks for developing Java language applications. It allows the validation of each of the parameters of a bean. These validations can be of length, type of data, patterns, etc., allowing an option for the validation of the input parameters for the created web applications based on Spring.

B. SIMILAR WORK COMPARISON

Tirronen [23] presents a proposal based on the analysis of structured data, in order to minimize attacks. The data is organized as an abstract syntax trees (ASTs). The performance of this proposal is not discussed quantitatively.

In [24] a study based on various machine learning techniques to identify SQL injections is presented. The study trains and tests with PHP code files. Two scenarios are provided using two feature data sets for the machine learning techniques. In terms of accuracy, the three best results are when using: Support Vector Machines (SVM), Multilayer Perceptron, and Convolutional Neural Network (CNN).

Aliero *et al.* in [25] proposes a method with the phases crawling, attacking analysis and reporting to stop SQL injections. It is tested in three web applications. This proposal obtains a precision equal to one.

At last, Venkatramulu and Guru [26] propose a Rule based PAttern Discovery (RPAD) for input type validation vulnerabilities. They propose to use it as a network IDS that allows the detection of vulnerabilities in the input validation. They carried out an experimental study with a dataset composed of 2,783 attack patterns extracted from NIST CVE entries and 512 normal patterns extracted from 7 real-time web applications. The precision obtained was 0.95.

Table 1 shows a comparison of the solutions described above as regards our proposal.

As it can be observed, in the previous table, the results obtained with the filter proposed in this article, based on the metrics (Precision, Recall and F-measure) are better than those obtained in the references [24] and [25] and similar to those obtained in RPAD [26].

C. WAF COMPARISON

Another different device, that is also used to defend against web applications attacks, is the Web Application Firewalls (WAF) [27]. It consists of a web server module or plugin that inspects HTTP/HTTPS traffic between a client and a server. It is based on a set of rules that allow detecting and blocking the most common attacks such as SQLi and XSS, input parameter manipulation, cookie hijacking, etc. It is indicated in this paper that the rule set of this type of

TABLE 1. Comparison of our filter with other similar works.

Paper	Proposal	Types of attack	Precision	Recall	F-measure
Proposed filter	Filter	Injection Attacks	**1**	**0,984**	**0,992**
[24]	SVM, Multilayer perceptron and CNN	SQLi	0,986 0,910 0,954	0,583 0,637 0,599	0,732 0,746 0,734
[25]	Filter	SQLi	**1**	0,500	0,67
[26]	Rules	SQLi, XSS	0.954	0.960	0.957

device requires a high maintenance cost, which is not an effective enough device to protect web applications and it should be complemented by a signature-based system. Since web applications are developed specifically for a particular organization, traditional signatures that detect attacks on web applications are not effective.

In the article by Razzaq *et al.* [28] a comparison of different WAF solutions, both commercial and open source, is presented (F5, Barracuda, Web Sniper, i-Sentry, Secure IIS, Easy Guard, Web Defend, Secure Sphere, Anchiva, Profanes, Citrix, WebApp secure, eServer Secure, Server defender AI and Mod Security). This comparison is made at the level of their specifications and most important characteristics, not providing real data on their effectiveness against different types of web applications attacks. That does not allow a precision[1] comparison between their results and those of ours.

The work of Holm and Ekstedt [29], present a study with real data on the effectiveness of various WAF devices (DenyAll rWeb, Imperva SecureSphere and Barracuda 660) against injection attacks. It was conducted by professional pentester using 16 different operational scenarios. The results of that study were reviewed and validated by a group of 49 experts, without using any objective metrics such as false positives, false negatives, etc. An accuracy of 80% was obtained with all this prevention countermeasures actives. If we compare it with the accuracy obtained with the filter implemented and described in this article, we observe that it is 98,4% much higher than the one indicated above.

An improvement in the effectiveness of WAF devices is the solution proposed in [30], called ANNbWA, based on the implementation of an Artificial Neural Network (ANN) to protect applications against SQLi and XSS attacks. The solution has an accuracy of 100% for XSS attacks and 98,5% for SQLi attacks. Compared with the filter proposed in this article, the solution has an accuracy of 100% for XSS attacks and 95% for SQLi attacks. The results are the same for XSS attacks and slightly worse for SQli attacks.

A similar work is carried out by Ito and Iyatomi [31], in which a WAF device based on a Character Level Convolutional Neural Network (CLCNN) is presented. The network was trained with a dataset that includes 36,000 normal traffic and over 25,000 malicious traffic. The accuracy obtained in this training was 99,8% compared to 98,4% given with the filter proposed in this article. However, that solution [31], has not been evaluated in a real situation unlike the filter that as indicated below is working in a company in the banking sector. The main drawback of that solution is that it only detects the attack but does not block it. This could be the cause of his low processing time.

In [32], a WAF is developed to detect new types of attacks that do not require signature updates, using a back-propagation neural network. That solution has an accuracy of 95%.

Last, Tekere and Bay [33] present a WAF that performs signature-based detection and prevention of Web attacks using a hybrid model that uses signature-based detection and anomaly-based detection implemented by a neural network. The results of accuracy against web application attacks obtained were 96,5%.

Table 2 shows a comparison of the accuracy of the solutions described above. It can be concluded that our filter presents a greater accuracy than some commercial WAFs. The same conclusion holds for similar to those of an experimental nature including techniques of artificial intelligence.

Based on all the above, we deduce that a generic filter to protect against injection attacks does not exist. This is due to the little or no validation of the imputs in the Web applications. However, there are directives offered by organizations dedicated to the defense of this type of application, as is the case of the OWASP and its entry validation reference sheet. Unlike the options that exist on the market, explained in this section, our research work presents a generic filter for any web application developed on any platform, with a generic configuration against common injection attacks and whose protection can be generalized (without depending on specific web application conditions). This means having a first line of defense that can be complemented with any of the solutions described above, as well as with secure programming practices. Therefore, we consider that our proposal contributes to the community in reducing injection attacks.

III. FILTER DESIGN CONSIDERATIONS

The objective of this research work is to develop a generic filter, with effective protection (accuracy) of at least 98,4% against common injection attacks. In order to get it, our filter ues regular expression, sanitizing known words in injection attacks, whose combination used as described in this work allows it to be integrated into web applications in a transparent way to protect all their input fields and avoid injection vulnerabilities. The following considerations are taken into account for the fitler design and to fulfill our objective:

1. The first consideration is what we are going to protect within the web applications: we must return to the basic concept of security, essential for this research work

[1]Precision is defined by the ratio of true positives (The attack exists, and the request is stopped) to total number of attacks per 100 (%)

TABLE 2. Comparison of our filter with WAFs works.

Paper	Type of Device	Types of attack	Accuracy (%)	Observations
Proposed filter	Filter	Injection Attacks	98,4%	
[29]	Commercial WAF	Injection Attacks	80%	
[30]	WAF ANN Experimental	SQli XSS	98,5% 100%	The filter proposed is 100% effective for XSS attacks and 95% effective for SQLi attacks
[31]	AWL CLCNN Experimental	All Attacks	99,8%	In the solution the data are from training not from practical situations.
[32]	AWF ANN Back-Propagation Experimental	All Attacks	95%	
[33]	WAF hybrid model Experimental	All Attacks	95,5%	

concerning web applications, which is "do not trust any parameter sent from the client to the web server, as it is susceptible to alteration". From this concept, our proposed filter considers validating HTTP headers, cookies, parameters (forms and query string), JSON and XML. URL parameters are not included in our generalization as their structure is unique to each web application.

2. The second consideration is "?'What can we generalize?". This is a central question from the OWASP directives mention Wichers [34]: validation of data types, validation of length based on a minimum and maximum dimension, the definition of valid known strings, regular expressions to define allowed strings, validation of JSON and XML requests with schemas, among others. All the directives indicated in previous paragraphs are valid and essential. However, for the development of the filter there must be the validation JSON, XML through schemas, allowance for the definition of characters, through the use of regular expressions and sanitization of reserved words (commonly used in injection attacks), the rest will depend on the correct use of each application.
3. The third consideration is about Technologies: For the filter, Java (a widespread language for web development), for the framework for input protection, in this case, OWASP Stinger and finally for the server to mount the filter configured as a module, Jboss/Wildfy.
4. As a fourth consideration, all the data that comes in a web request should be considered as input. In our filter we detail the following ones:
   a. HTTP headers: User-Agent, Content-Type, Host, among others, each of them configured with the regular expressions that are identified as expected strings in each web application. A generic configuration is not placed in the filter, but support is given to perform the suggested configuration.
   b. Cookies: The Java object ServletRequest is where the requests will be captured. This object has the characteristic of omitting cookies whose values have blank spaces. If the value of the cookie is encrypted (encoded in base64 or hexadecimal, or any other), every application should be responsible for the verification of the expected values.
   c. Parameters of an HTTP request: In an ideal scenario, the information between client and server is transmitted with alphanumeric characters and basic punctuation marks. The permissibility of the rest of the characters must be specified in the fields of each application, and as well the validation about if the information is correct for one of these fields is delegated to each application.
   d. JSON/XML: Modern applications increasingly tend to handle an exchange of JSON messaging or XML. To validate everything that is transmitted with JSON/XML, we incorporate the schemes validation in our filter.

IV. FILTER DEVELOPMENT AND DEPLOYMENT

In this section, first, we present the details of the development and deployment of the proposed filter, and then, we show the analysis, tests, and the results of our proposal.

For the development of our generic filter, we have worked over the OWASP Stinger 2.x framework. Furthermore, as we mentioned before, we have extended the functionality supporting JSON and XML schemes' validation.

The proposed filter has an external configuration file, which is structured in six important sections: regex, cookie rule, parameter rule, header rule, JSON rule, and XML rule; and four concepts: SANITIZATION, FILTER SETTINGS, FILTER AS JBOSS/WILDFLY MODULE and CONTROLLER. In the following, we describe these sections.

A. REGEX SECTION

In this section, the regular expressions to be used in the validation of the entries in headers, parameters and cookies are configured. We use the PERL syntax that is compatible with Java package java:util:regex defining four general expressions.

1) SECURE TEXT

Describe the minimum necessary characters to transfer information between client and application web, covering english and spanish. Here it is relevant to mention that it started only with alphanumeric characters and spaces.

Regex 1.

^(|[a-zA-Z0-9ñÑáÁéÉíÍóÓúÚ._!?]+)$

2) RESERVED WORDS

Reserved words commonly used in SQL and XSS injection attacks are defined. Many of these words are also used in the component IBM DATAPOWER to filter this type of attack. These reserved words are: select, update, delete, insert, procedure, create, alter, analyze, call, commit, drop, grant, purge, revoke, execute, union, exec, exec sp, exec xp, or, and, like, javascript, and script. They are defined in Regex 2.

Regex 2.

```
(\s+([Ss][Ee][Ll][Ee][Cc][Tt]|[Uu][Pp][Dd][Aa][Tt]
[Ee]|[Dd][Ee][Ll][Ee][Tt][Ee]|[Ii][Nn][Ss][Ee][Rr]
[Tt]|[Pp][Rr][Oo][Cc][Ee][Dd][Uu][Rr][Ee]|[Cc][Rr]
[Ee][Aa][Tt][Ee]|[Aa][Ll][Tt][Ee][Rr]|[Aa][Nn][Aa]
[Ll][Yy][Zz][Ee]|[Cc][Aa][Ll][Ll]|[Cc][Oo][Mm][Mm]
[Ii][Tt]|[Dd][Rr][Oo][Pp]|[Gg][Rr][Aa][Nn][Tt]|[Pp]
[Uu][Rr][Gg][Ee]|[Rr][Ee][Vv][Oo][Kk][Ee]|[Ee][Xx]
[Ee][Cc][Uu][Tt][Ee]|[Uu][Nn][Ii][Oo][Nn]|[Ee][Xx]
[Ee][Cc]|[Ee][Xx][Ee][Cc]\s[Ss][Pp]|[Ee][Xx][Ee]
[Cc]\s[Xx][Pp]|[Oo][Rr]|[Aa][Nn][Dd]|[Ll][Ii][Kk][Ee])
\s+|\s*([Jj][Aa][Vv][Aa][Ss][Cc][Rr][Ii][Pp][Tt]|[Ss]
[Cc][Rr][Ii][Pp][Tt])\s*))
```

3) USER AGENT

Regex 3 is a generalized regular expression for defining user-agent header. It has been tested against a database of 9219 user-agent headers and represents the most common user-agent.

Regex 3.

```
^[a-zA-Z]+/[\d.]+( )[a-zA-Z0-9.\-:_/\[\]\(\),;\+]*$
```

4) COOKIE

All alphanumeric and encoding characters (base64, URL encode, hexadecimal, and encryption) are supported. This is represented in Regex 4.

Regex 4.

```
^[a-zA-Z0-9/\+ =._\-%]+$
```

B. COOKIE RULE SECTION

This section is built with a default validation called COOKIE ALL, in which once the entry is validated, it goes through the sanitization process based on the reserved words.

C. PARAMETER RULE SECTION

This section is modified by adding the flag sanitized, which is set to TRUE value. Therefore, the input parameters are sanitized after passing the regex validations.

D. HEADER RULE SECTION

This section has been created for the validation and sanitization of the values of the different HTTP headers to be used in web applications, under the same principle of allowed values, associated by name and URI. Within the XML configuration file for this section, the ''Missing'' and ''Malformed'' concepts of the Stinger implementation are handled.

E. JSON RULE SECTION (NEW)

The best strategy to validate JSON messaging under the OWASP security parameters (type, length, accepted string, etc.) is the use of schemas, which is currently in draft-07. The JSON Schema documentation mentions: "JSON Schema is a vocabulary that allows annotations and validations of JSON documents Wright and Andrews [35]. To use this feature in the proposed filter, the following is done:

1) JSON TAG

Define the JSON tag in the configuration XML file with the name JSON ALL.

2) CONTENT-TYPE HEADER

JSON messages arriving at the filter must contain the Content-type header with the value application/JSON.

3) MISSING CONCEPT

The ''Missing'' concept (brought from Stinger implementation), serves to indicate what actions to take when a JSON is expected in the body of the petition, but this arrives empty.

4) PROCESSED CONCEPT

This concept allows configuring JSON's validation action against a predefined scheme for messages, whose location is fixed in a protected directory. The schema name is associated with the service name key in the root of the same JSON message. This parameter is also validated and sanitized before being used.

F. XML RULE SECTION (NEW)

The OWASP, in its validation sheet, recommends that the validation of XML messages should be done through schemas. To use this feature in this experimental prototype, the following is done:

- Define the XML tag in the configuration XML file with the name XML ALL.
- The XML messages that arrive at the filter must contain in the header Content-type with the pattern /XML.
- Concepts Missing and Processed are handled in a similar way for JSON Rule Section.

G. SANITIZATION

The process of sanitization of headers, cookies, and parameters, consists of the removal of reserved words from strings before passing them to the web application for processing. The regular expression that represents the reserved words is configurable according to the need if our proposal Regex 2 is not used.

H. FILTER SETTING

The initial configuration of the filter is required and included it within a web application. This configuration is done in the WEB.xml file, where the filter and URLs to be protected are defined. The filter definition accepts the following three parameters:

TABLE 3. Stinger module configuration modified as Jboss/Widlfy module.

```
<module xmlns="urn:jboss:module:1.1"
name="com.<company>.configuration">
  <resources>
    <resource-root path=" S-Stinger.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
    <module name="javax.servlet.api" optional="true"/>
          <system export="true">
              <paths>
                  <path name="sun/misc"/>
              </paths>
          </system>
  </dependencies>
</module>
```

1) CONFIG

Indicates the path and name of the XML file where the validation rules are configured.

2) ERROR PAGE

Indicates the name of the web page that will be redirected in case the filter detects an inconsistency.

3) RELOAD

It is a boolean parameter that indicates whether the XML configuration file is read in each request or not.

The filter has a customization section that can be adjusted to each product's reality. A validator can be configured to first allow decoding (either url encoding, base64, any type of encryption, etc.) and then apply data type validation, regular expression, and length. Therefore, it is not a problem for the filter to have it encoded.

I. FILTER AS JBOSS/WILDFY MODULE

The installation of the modified Stinger library is done in Jboss/Wildfy module mode. It can be available for an entire domain/host controllers infrastructure. In this way, it is avoided adding the library for each web application to be protected. Besides, it will have a single point of update of the library. To configure the Stinger library as a module in Jboss/Wildfy the next steps described below were followed:

1. Create the directory "../module/com/<yourcompany>/configuration/main/"
2. Create module.xml file with the structure and information shown in Table 3.
3. Paste the file *S-Stinger.jar* into the directory created in the first step.
4. Finally, place the jboss-deploymentstructure.xml file inside the WEB-INF folder, with the contents as in Table 4.

J. CONTROLLER

The component controller is a reverse Proxy Servlet. It was created for testing web applications programmed in any language. Its main function is to redirect the client's requests to the filter validator. If this request is valid, then it lets the request pass to the web application.

TABLE 4. Content of jboss-deployment-structure.xml.

```
<jboss-deployment-structure>
    <deployment>
        <dependencies>
            <module name="com.< your company >.configuration"/>
        </dependencies>
    </deployment>
</jboss-deployment-structure>
```

The controller is configured between the client and web applications. It receives all requests, and before passing them to the application, validates the entries against the security filter. If the filter detects an injection attempt, it rejects the request with a 400 HTTP response code, and returns an error message to the client. Show Figure 2.

All sections and concepts above described were the basis for experimentation in protecting against common web application injection attacks.

For a better understanding of the interaction between the filter components and their flow in Jboss/Wildfy the diagrams in Figures 3 and 4 are depicted.

V. RESULTS AND DISCUSSION

This section provides an evaluation of the implemented prototype of our filter, detailing the testing, results, and discussion of the filter performance.

The proposed filter is evaluated from two points of view:

1. Capability to detect and stop common injection attacks. The proposed filter has been tested in three public applications created for this type of test, and in an application of a private company in a real environment.
2. Performance in a concurrent environment. This point is important because the filter is designed for any web application, whether it is web banking, service, etc. applications. Therefore it is important that it does not degrade the response of customers using these applications even in times of high concurrence.

A. CAPABILITY TO DETECT AND STOP COMMON INJECTION ATTACKS

The capability of our proposal to detect and stop common injection attacks has been tested on three public and one private web applications. All three public applications are known to have these types of vulnerabilities. The private application, on the other hand, has been developed using good programming practices. To perform the tests on the public web applications, the OWASP ZAP tool has been used to automate hundreds of requests in a short period, and a dynamic analysis has been performed to capture the results. Meanwhile, the tests on the private web application have been performed in a production environment and for one year.

The process for testing the three public web applications was:

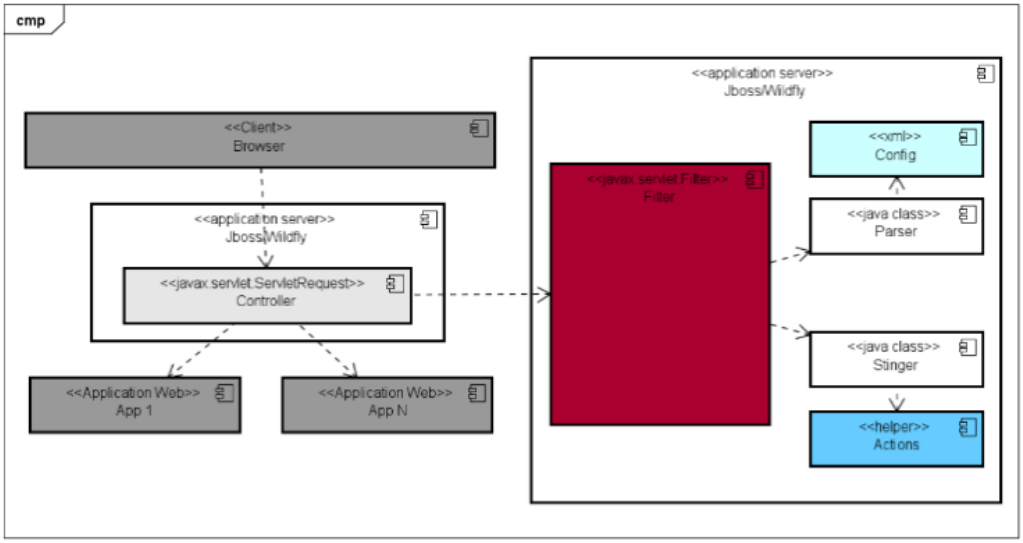


FIGURE 2. Component diagram – interaction controller.

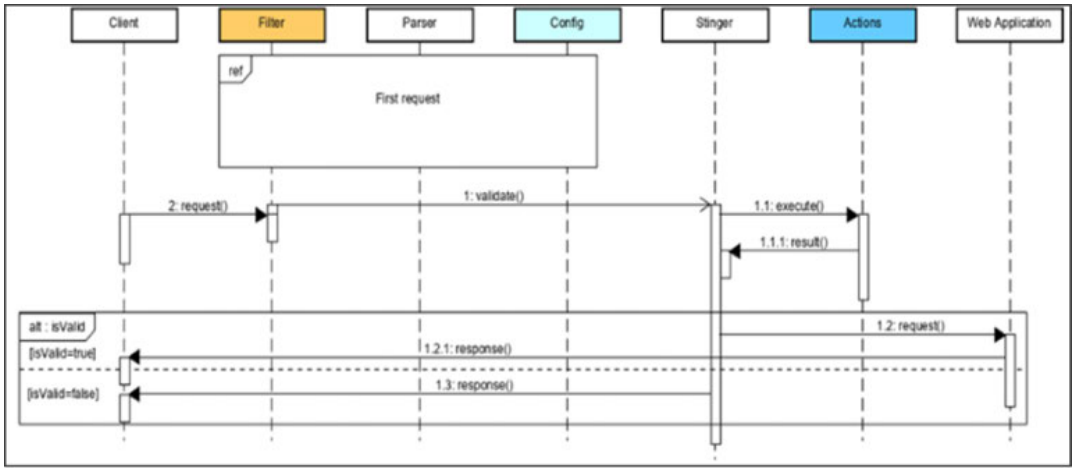


FIGURE 3. Diagram of sequence – interaction between client, filter and web application.

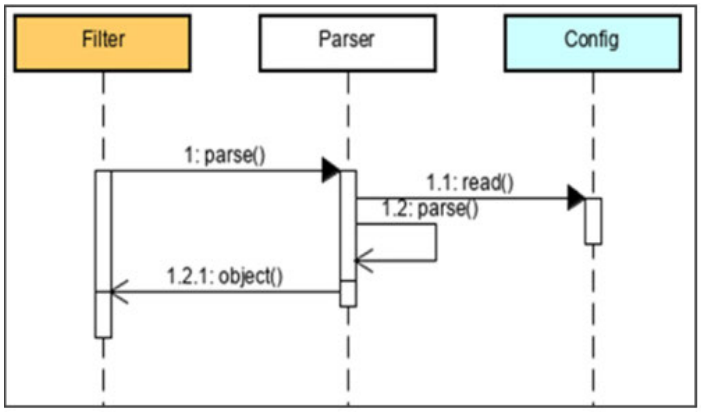


FIGURE 4. Diagram of sequence – "first request" square in Fig. 2.

1. Choose three web applications that are vulnerable to injection attacks.
2. Configure the OWASP ZAP tool [36], setting the context (allows automating the scanning of the internal pages after login), and policy scan (it's a set of real and common injection attacks with the default threshold, strength and quality).
3. Register the dynamic analysis (DAST) of these applications without the protection filter. Then, configure the filter like a jboss/wildfy module in each web

application, as it was described in the prior section, and register DAST with the protection filter.

1) WEB APPLICATIONS FOR TESTING

Three web applications for testing were selected, considering to be applications that have injection vulnerabilities, and they are chosen at random. The selected applications were: Webgoat OWASP [37], Badstore Roemer [38], and bWap Mesellem [39].

2) CONTEXTS IN ZAP TOOL OWASP

Use the OWASP ZAP tool for security tests [36]. The steps for the tests to be performed with this tool are the following:

- Configure the ZAP tool as a proxy on a free port (for example 9191).
- The default scan policies are used in ZAP tool, since they include all injection types.
  - Threshold equal to Default: the default value is ''medium''. It is acceptable because if we set a low level, we could have many false positives, and a high level could cause many vulnerabilities not to be reported.
  - Strength equal to Default: this setting determines the number of attacks that the ZAP tool must perform to
  - determine vulnerability. The default value is the mean value (approximately 12).
  - Quality equal to Release: this parameter indicates how mature the rule for determining a vulnerability is. The release represents the rules that have been extensively tested and mature, which are precisely the ones that fit the purpose of our experiment.
- Activate proxy through browser settings.
- Configure Servlet Reverse Proxy (with/without activating filter).
- Place the Servlet Reverse Proxy URL and start capturing the authentication process in each of the selected applications. The authentication of each application consists of a login form URL, a validation request, and a final page once authenticated. With the authentication requests, a context for each web application is defined.
- Explore applications for all pages that are known to have injection vulnerabilities beforehand so that the ZAP tool can save it and then target attacks and exploit these vulnerabilities.Select the context and startup request for each application and start the attacks: Spider, AJAX Spider, Active Scan and in some cases fuzz (under analysis).
- The previous step is performed, for each application with and without filter activated in our Reverse Proxy Servlet.

3) DYNAMIC ANALYSIS WITH/WITHOUT THE FILTER PROTECTION

A success attack tested without using the filter is considered an existing attack. Therefore, the vulnerabilities are known, and the attacks tested with the filter can be classified according to a confusion matrix [40]:

TABLE 5. WebGoat application results.

Vulnerability	No filter	With filter	Discarded	FN	TP
SQL Inyection (H)	263	10	10	0	253
SQL Injection – Hypersonic	7	0	0	0	7
SQL Path Traversal	5	3	0	3	2
Cross Site Scripting (Reflected)	6	0	0	0	6
Format String Error	29	0	0	0	29
Parameter Tampering	3	0	0	0	3
TOTAL	313	13	10	3	300

a) **True Positive (TP):** Right. The attack exists, and the request is stopped.
b) **True Negative (TN):** Successful. The attack does not exist, and the request is not stopped.
c) **False Positive (FP):** Leftover. The attack does not exist, but, the request is stopped.
d) **False Negative (FN):** Fault. The attack exists, but the request is not stopped.

Tables 5, 6, and 7 show the evaluation of the three selected applications: Webgoat, Badstore, and bWap, respectively. It is reported: the number of successful attacks when the web application is configured without the filter protection (Without filter); the number of successful attacks when the web application is configured with the filter protection (With filter); the number of discarded successful attacks (Discarded); False-Negative (FN); and True-Positive (TP) taking off discarded cases.

Successful attacks using the filter are discarded with the following criteria:

- The initial filter settings include User-Agent header protection, form parameters, query string parameters, cookies, JSON and XML messages. All other cases (e.g., URL parameters), require criteria of each application. Therefore, all attacks that could not be stopped, but do not enter inside the scope of the filter, will be discarded.
- Attacks that included blank spaces in the cookie values are also discarded, as these values are omitted by the javax.servlet.ServletRequest object, which handles the requests in this prototype. In other words, the web application simply did not receive the manipulated cookie as described at this point. If this cookie is not used in the web application, it returns a valid response, and ZAP takes it as a successful attack.

Concerning the results when testing the WebGoat web application, as shown in Table 5, there were 313 successful attacks without the filter activated, 300 rejected by the filter, and 13 successful; 10 discarded, and only 3 false negatives.

TABLE 6. bWAPP application results.

Vulnerability	No filter	With filter	Discarded	FN	TP
SQL Inyection (H)	128	11	3	8	117
SQL Injection – Hypersonic	90	3	3	0	87
SQL Path Traversal	2	0	0	0	2
Cross Site Scripting (Reflected)	3	0	0	0	3
Format String Error	1	0	0	0	1
Parameter Tampering	5	0	0	0	5
TOTAL	229	14	6	8	215

TABLE 7. Badstore application results.

Vulnerability	No filter	With filter	Discarded	FN	TP
SQL Inyection (H)	34	0	0	0	34
Cross Site Scripting (Reflected)	17	0	0	0	17
Cross Site Scripting (Persistent)	3	0	0	0	3
TOTAL	54	0	0	0	54

Table 6. shows the results when testing with the bWAPP web application. A total of 229 successful attacks to the bWAPP web application without the filter activated, 215 rejected by the filter, 14 successful; 6 discards and 8 false negatives.

Table 7. shows the results when testing with the Badstore web application. The vulnerability found without the filter is registered. A total of 54 successful attacks were recorded without the filter protection, and all the events were rejected by the filter.

To assess the effectiveness of the filter against injection attacks in the input of a web application, we also apply the metrics Precision [40], AcurRecall [40], F-measure [40] and Defensive Efficiency [41].

The recall is the hit rate for filtering vulnerabilities. It is given by the ratio of correctly detected attacks and the total number of known attacks. It is also known like True Positive Rate (TPR) or Sensitivity. This relationship is represented by the mathematical formula in (1).

$$Recall = \frac{TP}{TP + FN} \tag{1}$$

Accuracy is given by the relationship in the proportion of correct detected attacks (both true positives and true negatives) among the total number of attacks. This relationship is represented by the following mathematical formula in (2).

$$Accuracy = (TP+TN)/(TP+FN+TN+FP) \times 100 \tag{2}$$

The value of 1 in accuracy indicates that we do not have false positives, but there exists the risk of passing many vulnerabilities undiscovered.

Precision is the rate of correct detection of attacks. It is given by the relationship between correctly detected attacks and the number of total attacks and the number of total attacks detected. It is also known like Positive Predictive Value (PPV). This relationship is represented by the following mathematical formula in (3).

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

In work [37], F-measure is defined as "a harmonic measurement of precision and recall values." The harmonic average is more intuitive than the arithmetic average when calculating an average of ratios. This relationship is represented by the mathematical formula in (4).

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{4}$$

Defensive Efficiency is the rate with which the filter managed to stop common injection [38]. It is calculated with the formula in (5).

$$DE = \frac{attacks - successattacks}{defensive} \times 100\%, \tag{5}$$

where: "attacks" denotes the total of observed attacks, "successattacks" refers to the total of observed successful attacks, and "defensive" means the total of successful defensive actions.

When all attacks are successful, the defensive efficiency is zero. For priority defenses, efficiency ranges from 0% to 100%. For preventive defenses, the efficiency tends to be equal or greater than 100% (the minimum occurs when the probability of attack prevention is 100%).

As we mentioned, the proposed filter has been tested on a private web application that provides banking transaction services, for a year. The web application has a high transactionality, 213,989,225 requests have been registered from 2016 to 2020, during this period 245 injection attacks were detected and stopped by the filter. Therefore, the high transaction rate was composed of valid requests that the filter recognized as such, without causing a denial of service. In terms of true and false positive cases (TP, FP, respectively), we had TP = 1, and FP = 0. Based on which, the filter performance in the private application is also computed.

Table 8 presents the filter performance according to each web application and average. It should be clarified that due to the minimal number of attempted attacks on the private application, the performance of the private application has not been considered for the calculation of the average filter performance.

TABLE 8. Filter performance. Average does not take account private rate.

Web application	Precision	Accuracy (%)	Recall	F-measure	Defensive Efficiency
Private	1	100.0%	1	1	100.0%
Webgoat	1	99.1%	0.990	0.995	105.3%
bWAPP	1	96.4%	0.964	0.982	102.79%
Badstore	1	100.0%	1	1	100.0%
Average	1	98.47%	0.984	0.992	102.71%

Finally, we include a series of limitations regarding the solution implemented and presented in this article:

- Filtering using regular white list expressions can prohibit benign strings if they have not been included in the list. This problem can be solved by observing false positives and debugging the corresponding expressions.
- The inclusion of validation of XML and JSON objects through their schemas may be insufficient if the schemas themselves do not include proper validation of the data with regular expressions. Another limitation is that a schema may or may not include blacklisted or white-listed regular expressions for input and output data validation.

B. FILTER LOAD TESTS

In order to complete this experimental prototype, it is important to perform load tests to analyze the supported concurrency. The load tests help to determine the maximum number of concurrent requests supported by the filter. The results can help in making infrastructure decisions (have one or more customer service lines) depending on each web application.

In our case, we will test load for requests with parameters, JSON, and XML content, under the following computer configurations: Processor Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz, 16 GB of RAM, 512 GB of solid-state disk, Windows 10 Pro 64-bit, and Jboss/Wildfy with 2GB of allocated memory. A Jmeter apache script is created, with three requests (with parameters, with JSON content and the last one with XML content).

1) TEST 1

In the first test, 1000 concurrent threads were uploaded in 5 seconds (200 per second), and the scheduler was set to 300 seconds (5 minutes), to maintain the concurrency of the 1000 threads. The response time is plotted in Figure 5. The aggregate report is given in Table 9.

2) TEST 2

In the second test configuration, 500 concurrent threads were uploaded in a period of 5 seconds (100 per second), and the scheduler was set 300 seconds (5 minutes), to maintain the concurrency of the 500 threads. The response time is plotted in Figure 6. The aggregate report is given in Table 10.

3) TEST 3

In the third test configuration, 250 concurrent threads were uploaded in 5 seconds (50 per second), and the scheduler was set 300 seconds (5 minutes), to maintain the concurrency of the 250 threads. The response time is plotted in Figure 7. The aggregate report is given in Table 11.

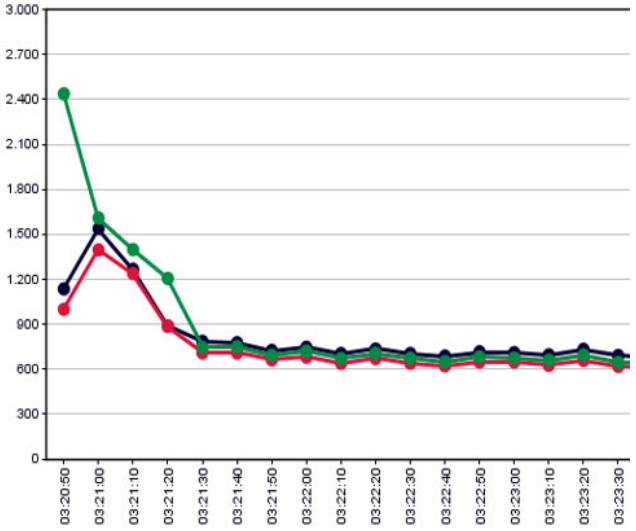


FIGURE 5. Response time, test with 1000 concurrent threads. The Y-axis unit is milliseconds. The X-axis is a timeline. Blue line: request of http-form. Red line: request of HTTP-JSON. Green line: request of HTP-XML.

TABLE 9. Aggregate report with 1000 concurrent threads.

	HTTP-Form	HTTP-JSON	HTTP-XML	Total
# Samples	136293	135978	135665	407936
Samples	758	691	737	729
Median	705	643	681	674
90% Line	1008	871	925	942
95% Line	1171	1011	1093	1109
99% Line	1650	1510	1842	1672
Minimum	12	258	281	12
Maximum	4538	3202	11043	11043
Error %	0.00%	0.00%	0.00%	0.00%
Performance	453.3/sec	452.8/sec	453.3/sec	1356.9/sec
Kb/sec	64.19	64.12	64.19	192.14
Sent KB/sec	263.86	219.79	332.01	813.90

TABLE 10. Aggregate report with 500 concurrent threads.

	HTTP-Form	HTTP-JSON	HTTP-XML	Total
# Samples	180365	180146	180013	540524
Samples	308	244	270	274
Median	292	223	251	244
90% Line	461	345	370	389
95% Line	519	374	409	449
99% Line	657	457	504	576
Minimum	1	0	2	0
Maximum	1838	1369	1412	1838
Error %	0.00%	0.00%	0.00%	0.00%
Performance	600.7/sec	600.0/sec	599.5/sec	1800.1/sec
Kb/sec	85.06	84.96	84.89	254.90
Sent KB/sec	349.61	291.19	439.11	1079.87

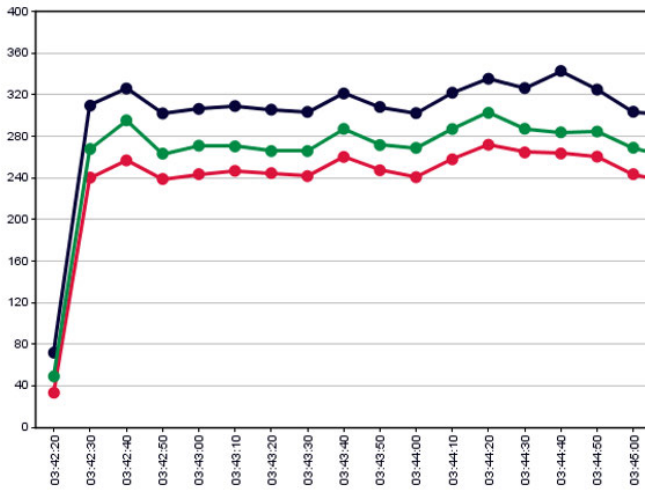


FIGURE 6. Response time, test with 500 concurrent threads. The Y-axis unit is milliseconds. The X-axis is a timeline. Blue line: request of http-form. Red line: request of HTTP-JSON. Green line: request of HTP-XML.

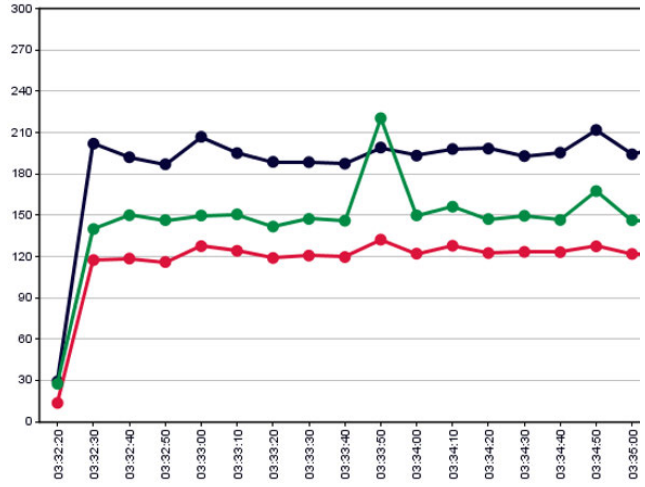


FIGURE 7. Response time, test with 250 concurrent threads. The Y-axis unit is milliseconds. The X-axis is a timeline. Blue line: request of http-form. Red line: request of HTTP-JSON. Green line: request of HTP-XML.

TABLE 11. Aggregate report with 250 concurrent threads.

	HTTP-Form	HTTP-JSON	HTTP-XML	Total
# Samples	157382	157293	157216	471891
Samples	194	121	150	155
Median	172	101	126	123
90% Line	361	236	262	281
95% Line	425	267	296	342
99% Line	584	338	407	492
Minimum	0	0	2	0
Maximum	1642	1262	2451	2451
Error %	0.00%	0.00%	0.00%	0.00%
Performance	524.4/sec	524.1/sec	523.9/sec	1572.4/sec
Kb/sec	74.26	74.22	74.18	222.65
Sent KB/sec	305.22	254.39	383.69	943.28

The following should be noted:

a. All three tests have 0% error, which means that the web server never stopped responding
b. From the analysis of tests 1 to 3 we can say that we can use our filter with 500 concurrent requests without passing the half second increase in processing.
c. The processing of each request in a sample of 100 individual non-concurrent requests takes an average of 50 ms.

VI. CONCLUSION AND FUTURE WORKS

This section concludes the paper and list some future works.

A. CONCLUSION

Considering the objective of this wok, we have achieved a relevant contribution, the design of a filter for stopping injection attacks in web applications through a set of rules based on regexes and other decisive settings. Thus, by analyzing the results of this work we can conclude:

- It is possible to create a filter that is effective (98,4% accuracy) against common injection attacks in web applications. It has been done through the strategy of validating all the data that comes in an HTTP request (headers, parameters), with fair regular expressions for each field (only what is expected) and sanitizing common words used in injection attacks. Most of the related studies are focused on protecting a specific type of attack, while in our work we have shown that with our strategy we can effectively prevent most of the common injection attacks.
- On the one hand, the proposed filter presents a better accuracy than some commercial WAFs and similar accuracy those of experimental character that include techniques of artificial intelligence. On the other hand, the proposed filter, compared to other filter-based solutions, shows better results than those obtained in the references [24] and [25] and close results to those obtained in [26].
- The average filter processing time is less than half a second, allowing web applications to improve security without affecting response times to end-users. This has been another important advantage to using it in a private high transactionality web application of banking services.
- Simple solutions applied in the appropriate way can allow correcting big security problems on web applications. The basic strategy proposed in this work can lead to getting anapplication on-line with the confidence that they will be effectively protected against common injection attacks.

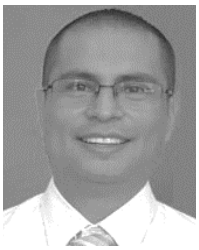
B. FUTURE WORKS

This section proposes improvements to the present work, with regards to the initial results of which are quite promising. Among what is expected to advance, the following points are being considered:

- The advance of the Web applications, as now it includes URL parameters, and with the time it is necessary to complement the development of the filter to protect this type of parameters under the methodology raised in the present work.
- Compute the size of the sample of web applications to be tested, in order to generalize the basic configuration of protection against injection attacks (starting point).

REFERENCES

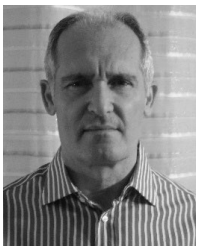
[1] OWASP. (2017). *Top 10-2017 A1-injection. 2017*. [Online]. Available: https://www.owasp.org/index.php/Top_10-2017_A1-Injection
[2] OWASP. *OWASP Stinger Version 2*. Accessed: Jan. 10, 2021. [Online]. Available: https://www.owasp.org/640index.php/OWASP_Stinger_Version_2
[3] N. Salnikov-Tarnovski. (2017). *Most Popular Java Application Servers: 2017 Edition*. [Online]. Available: https://dzone.com/articles/most-popular-java-application-servers-2017-edition
[4] J. D. V. Mohino, B. Higuera, B. Higuera, and S. Montalvo, "The application of a new secure software development life cycle (S-SDLC) with agile methodologies," *Electronics*, vol. 8, no. 11, p. 1218, Oct. 2019, doi: 10.3390/electronics8111218.
[5] H. Homaei and H. R. Shahriari, "Seven years of software vulnerabilities: The Ebb and flow," *IEEE Secur. Privacy*, vol. 15, no. 1, pp. 58–65, Jan. 2017.
[6] A. Barabanov, A. Markov, and V. Tsirlov, "Statistics of software vulnerability detection in certification testing," in *Proc. Int. Conf. Inf. Technol. Bus. Ind.*, Tomsk, Russia, 2018, vol. 1015, no. 4, pp. 1–9.
[7] T. Scholte, W. Robertson, D. Balzarotti, and E. Kirda, "An empirical analysis of input validation mechanisms in Web applications and languages," in *Proc. 27th Annu. ACM Symp. Appl. Comput. (SAC)*. New York, NY, USA: Association Computing Machinery, 2012, pp. 1419–1426, doi: 10.1145/2245276.2232004.
[8] D. Scott and R. Sharp, "Abstracting application-level Web security," in *Proc. 11th Int. Conf. World Wide Web (WWW)*, New York, NY, USA, 2002, pp. 396–407, doi: 10.1145/511446.511498.
[9] S. W. Boyd and A. D. Keromytis, "Sqlrand: Preventing SQL injection attacks," in *Applied Cryptography and Network Security*, M. Jakobsson, M. Yung, and J. Zhou, Eds. Berlin, Germany: Springer, 2004, pp. 292–302, doi: 10.1007/978-3-540-24852-1_21.
[10] G. Wassermann and Z. Su, "Sound and precise analysis of Web applications for injection vulnerabilities," in *Proc. ACM SIGPLAN Conf. Program. Lang. Design Implement. (PLDI)*. New York, NY, USA: Association Computing Machinery, 2007, pp. 32–41, doi: 10.1145/1250734.1250739.
[11] O. Ismail, M. Etoh, and Y. Kadobayashi, "A proposal and implementation of automatic detection/collection system for cross-site scripting vulnerability," in *Proc. 18th Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, vol. 1, 2004, pp. 145–151, doi: 10.1109/AINA.2004.1283902.
[12] G. T. Buehrer, B. W. Weide, and P. A. G. Sivilotti, "Using parse tree validation to prevent SQL injection attacks," in *Proc. 5th Int. Workshop Softw. Eng. Middleware (SEM)*. New York, NY, USA: ACM, 2005, pp. 106–113, doi: 10.1145/1108473.1108496.
[13] H. Liu and H. B. K. Tan, "Automated verification and test case generation for input validation," in *Proc. Int. Workshop Autom. Softw. Test (AST)*. New York, NY, USA: ACM, 2006, pp. 29–35, doi: 10.1145/1138929.1138936.
[14] Y. Park and J. Park, "Web application intrusion detection system for input validation attack," in *Proc. 3rd Int. Conf. Converg. Hybrid Inf. Technol.*, vol. 2, Nov. 2008, pp. 498–504, doi: 10.1109/ICCIT.2008.338.
[15] S. Aljawarneh, F. Alkhateeb, and E. Al Maghayreh, "A semantic data validation service for Web applications," *J. Theor. Appl. Electron. Commerce Res.*, vol. 5, no. 1, pp. 39–55, Apr. 2010, doi: 10.4067/S0718-18762010000100005.
[16] T. Scholte, W. Robertson, D. Balzarotti, and E. Kirda, "Preventing input validation vulnerabilities in Web applications through automated type analysis," in *Proc. IEEE 36th Annu. Comput. Softw. Appl. Conf.*, Jul. 2012, pp. 233–243, doi: 10.1109/COMPSAC.2012.34.
[17] J. Liu and Y. Ou, "An improved XSS vulnerability detection method based on attack vector," in *Proc. Int. Conf. Modeling, Simulation Anal. (ICMSA)*, 2018, doi: 10.12783/dtcse/icmsa2018/23251.
[18] A. K. Dalai, S. D. Ankusha, and S. K. Jena, "XSS attack prevention using DOM-based filter," in *Progress in Intelligent Computing Techniques: Theory, Practice, and Applications* (Advances in Intelligent Systems and Computing), vol. 719, P. Sa, M. Sahoo, M. Murugappan, Y. Wu, and B. Majhi, Eds. Singapore: Springer, 2018, pp. 227–234, doi: 10.1007/978-981-10-3376-6_25.
[19] IBM Support. (2019). *Configuring the Cross-Site Scripting (XSS) Fllter on the IBM Websphere Datapower SOA Appliance*. [Online]. Available: http://www01.ibm.com/support/docview.wss?uid=swg21456694
[20] IBM Support. (2019). *Customizing Default SQL Injection Protection on the IBM Websphere Datapower SOA Appliance*. [Online]. Available: http://www-01.ibm.com/support/docview.wss?uid=swg21444739
[21] P. Brady. (2017). *Input Validation*. [Online]. Available: https://phpsecurity.readthedocs.io/en/latest/Input-Validation.html
[22] R. Johnson *et al.* (2009). *Spring 3 Validation*. [Online]. Available: https://docs.spring.io/spring/docs/3.0.0.RC2/reference/html/ch05s07.html
[23] V. Tirronen, "Stopping injection attacks with code and structured data," in *Cyber Security: Power and Technology* (Intelligent Systems, Control and Automation: Science and Engineering), vol. 93, M. Lehto and P. Neittaanmäki, Eds. Cham, Switzerland: Springer, vol. 93, 2018, pp. 219–231, doi: 10.1007/978-3-319-75307-2_13.
[24] K. Zhang, "A machine learning based approach to identify SQL injection vulnerabilities," in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, San Diego, CA, USA, Nov. 2019, pp. 1286–1288, doi: 10.1109/ASE.2019.00164.
[25] M. S. Aliero, I. Ghani, K. N. Qureshi, and M. F. Rohani, "An algorithm for detecting SQL injection vulnerability using black-box testing," *J. Ambient Intell. Humanized Comput.*, vol. 11, no. 1, pp. 249–266, Jan. 2020, doi: 10.1007/s12652-019-01235-z.
[26] S. Venkatramulu and R. Guru, "RPAD: Rule based pattern discovery for input type validation vulnerabilities detection & prevention of HTTP requests," *Int. J. Appl. Eng. Res.*, vol. 12, no. 24, pp. 14033–14039, 2017.
[27] S. Khandelwal, P. Shah, M. K. Bhavsar, and D. S. Gandhi, "Frontline techniques to prevent Web application vulnerability," *Int. J. Adv. Res. Comput. Sci. Electron. Eng.*, vol. 2, no. 2, p. 208, 2013.
[28] A. Razzaq, A. Hur, S. Shahbaz, M. Masood, and H. F. Ahmad, "Critical analysis on Web application firewall solutions," in *Proc. IEEE 11th Int. Symp. Auto. Decentralized Syst. (ISADS)*, Mexico City, Mexico, Mar. 2013, pp. 1–6, doi: 10.1109/ISADS.2013.6513431.
[29] H. Holm and M. Ekstedt, "Estimates on the effectiveness of Web application firewalls against targeted attacks," *Inf. Manage. Comput. Secur.*, vol. 21, no. 4, p. 250, 2013, doi 10.1108/IMCS-11-2012-0064.
[30] A. Moosa, "Artificial neural network based Web application firewall for SQL injection," *Int. J. Comput., Elect., Automat., Control Inf. Eng.*, vol. 4, no. 4, pp. 610–619, 2010.
[31] M. Ito and H. Iyatomi, "Web application firewall using character-level convolutional neural network," in *Proc. IEEE 14th Int. Colloq. Signal Process. Appl. (CSPA)*, Batu Feringghi, Malaysia, Mar. 2018, pp. 103–106, doi: 10.1109/CSPA.2018.8368694.
[32] J. J. Stephan, S. D. Mohammed, and M. K. Abbas, "Neural network approach to Web application protection," *Int. J. Inf. Edu. Technol.*, vol. 5, no. 2, p. 150, 2015.
[33] A. Tekerek and O. F. Bay, "Design and implementation of an artificial intelligence-based Web application firewall model," *Neural Netw. World*, vol. 29, no. 4, pp. 189–206, 2019, doi: 10.14311/NNW.2019.29.013.
[34] D. Wichers. (2019). *Input Validation Cheat Sheet*. [Online]. Available: https://github.com/OWASP/CheatSheetSeries
[35] A. Wright and H. Andrews. (2019). *JSON Schema*. Accessed: Mar. 20, 2020. [Online]. Available: http://json-schema.org/
[36] OWASP. (2019). *OWASP Zed Attack Proxy Project*. [Online]. Available: https://owasp.org/www-project-zap/
[37] OWASP. (2018). *OWASP WebGoat Project*. [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project
[38] K. R. Roemer. (2018). *BadStore*. [Online]. Available: https://download.vulnhub.com/badstore/
[39] C. M. Mesellem. *BWAPP an Extremely Buggy Web App*. Accessed: Jan. 10, 2021. [Online]. Available: http://www.itsecgames.com/
[40] G. Díaz and J. R. Bermejo, "Static analysis of source code security: Assessment of tools against SAMATE tests," *Inf. Softw. Technol.*, vol. 55, no. 8, pp. 1462–1476, Aug. 2013, doi: 10.1016/j.infsof.2013.02.005.
[41] J. E. Sandoval and S. P. Hassell, "Measurement, identification and calculation of cyber defense metrics," in *Proc. MILCOM Mil. Commun. Conf.*, San Jose, CA, USA, Oct. 2010, pp. 2174–2179, doi: 10.1109/MILCOM.2010.5680489.



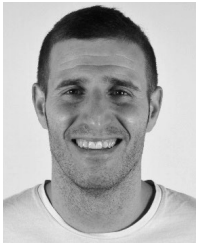
SANTIAGO IBARRA-FIALLOS received the B.S. degree from the National Polytechnic School, Quito, Ecuador, in 2007, and the master's degree from the International University of Rioja, La Rioja, Spain, in 2018. He is currently a Researcher and a Senior Software Developer of Web and mobile bank applications. His interests include the fields of software security.

JAVIER BERMEJO HIGUERA received the B.S. degree from Alcala University and the Ph.D. degree from the Army Polytechnic School. He is currently a Professor with the Escuela Superior de Ingeniería y Tecnología, Universidad Internacional de La Rioja. He has authored several publications. His research interests include the fields of software security, cybersecurity, and malware analysis.

MONSERRATE INTRIAGO-PAZMIÑO received the B.S. degree in computer science engineering from the National Polytechnic School, Quito, Ecuador, in 2007, and the M.S. degree in computer science from the Technical University of Madrid, Madrid, Spain, in 2012. She is currently pursuing the Ph.D. degree in computer science with the Technical University of Madrid. She is currently an Associate Professor with the Department of Informatics and Computer Science, National Polytechnic School. Her research interests include software quality and machine learning.



JUAN RAMÓN BERMEJO HIGUERA received the M.Sc. degree in computer engineering and the Ph.D. degree from the Distance Education National University, Spain, in 1998 and 2014, respectively. He is currently the Chief of the Cybersecurity Unit, National Institute of Aerospace Techniques, Spain. He is also a Professor of applications security in the International University of La Rioja and also an Associate Professor of the Ciberdefence Master Science degree with the Alcala de Henares University. He has authored several publications. His research interests include cybersecurity and cyber defense.



JUAN ANTONIO SICILIA MONTALVO received the B.S. and Ph.D. degrees from the Universidad de Zaragoza. He is currently a Professor with the Escuela Superior de Ingeniería y Tecnología, Universidad Internacional de La Rioja. His research interests include combinatorial optimization, computer security, software development based on mathematical algorithms, numerical methods, and heuristic techniques for solving engineering problems.

JAVIER CUBO received the B.S. and Ph.D. degrees in computer science with the Universidad de Málaga. He has worked both in academia and industry. He is currently an Academic Coordinator and the Head of the Software Engineering and Security Research Group, Escuela Superior de Ingeniería y Tecnología, Universidad Internacional de La Rioja. He received two scholarships as a Visiting Research Ph.D. Student with the University College London, and a Postdoctoral Researcher with the University of Pisa. He has participated in numerous research projects and conferences. He has authored or coauthored in journals and international workshops and conferences. His research is focused on the development of models, methodologies, architectures in software engineering, in the paradigms of service-oriented computing, cloud computing, big data, the Internet of Things and services, smart cities technologies, and marketing digital aspects. He is a member of organizing and program committees and a reviewer of journals over the last years.