A Defensive Framework for Reflected XSS in Client-Side Applications

Khulud Fisal Alenzi[1] and Onytra Abbas Bashir Abbas[2,*]

[1]*Department of Information Technology, University of Tabuk, Kingdom of Saudi Arabia*
[2]*Department of Computer Science, University of Tabuk, Kingdom of Saudi Arabia*
E-mail: kookaaf@gmail.com; obashir@ut.edu.sa
[*]*Corresponding Author*

Received 08 February 2021; Accepted 24 August 2022;
Publication 28 December 2022

Abstract

Cross-site scripting attack (XSS) is a common vulnerability that is exploited in modern web applications by entering advanced HTML tags and Java Script functions. An attacker could potentially use this vulnerability to steal users' sensitive information, hijack user sessions or rewrite whole website contents displaying fake login forms. This class of attacks affects the client-side of a web application and is a critical vulnerability that is difficult to both detect and remediate for websites, often leading to insufficient server-side protection, which is why the end-users need an extra layer of protection at the client-side. In this paper, we analyze the best-known client-side XSS filters, study their mechanisms, structures and mentioned the advantages and disadvantages of each filter. This paper presents a novel XSS filtering model based on filtering rules, XSSFilter, uses Regular Expression in Xpath to detect reflected content, which makes it more robust for web sites that employ custom input sanitizations. We provide a detailed experimental evaluation to compare the four filters with respect to their usability and protection.

Keywords: Cross-site scripting, XSS, XSS filters, filtering rules, XSSFilter.

Journal of Web Engineering, Vol. 21_7, 2209–2230.
doi: 10.13052/jwe1540-9589.2179

1 Introduction

Cross-site scripting (XSS) is the most worrying security issue confronting web programmers (Wichers, 2013). XSS now tops buffer overruns as a type of reported vulnerability (Christey and Martin, 2007). Even though it is still difficult to fix XSS weakness yet fixing each XSS weakness in a huge site is a more troublesome assignment, an undertaking that many sites don't achieve.

Rather than waiting for each website to fix its XSS security gaps, web browsers can reduce some of the XSS vulnerability categories, protect platforms that have not yet, or may not, correct their vulnerabilities. Normally, it is easy to create XSS filters from the client-side. In a reflected XSS risk, the payload is injected in both the HTTP request to the server and the HTTP reply from the server. However, there are some problems to build a candidate without any false negatives, even for a limited set of security issues. XSS has for quite some time been through the best dangers to "web security" as characterized in few reports that contain definite data about the degree and danger of this class of weaknesses. One of these reports is the "Open Web Application Security Project (OWASP) Top 10 – 2017" report that contains a rundown of the main 10 most significant security chances for web applications (OWASP, 2017). Albeit cross-site scripting has tumbled to seventh in the "OWASP Top 10 – 2017" report (OWASP, 2017), cross-site programming stays one of the most perilous types of risk.

Another report, distributed every year in the course of recent years, by White-Hat Security, called the "2017 – The White-Hat Security Application Report" (Grossman, 2007), likewise demonstrates that cross-webpage scripting is among the two most significant shortcomings on the web, the stressing note in this report is that albeit cross-website scripting is one of the most genuine weaknesses, it isn't offered need to address it by most sites. The measurements being introduced recommend that the weaknesses getting the most consideration are weaknesses that are anything but difficult to fix. Therefore, it is recommended that associations must embrace a danger-based remediation measure, which implies that the most basic weaknesses should be organized first, similar to cross-site scripting.

One investigation from Hydara et al. (Hydara, Sultan, Zulzalil, and Admodiasaso, 2015). They inspected a sum of 115 examinations identified with cross-site scripting, presumed that XSS stays a major issue for web apps, notwithstanding all the submitted exploration also the arrangements being given up until now. As observed from the later statistics from OWASP, White Hat, and the BugCrowd, this end nods, that XSS vulnerabilities remains to be at large.

The internet browser works in the interface between the web application client and the webserver. The security of the request created here is dependent on the HTTP headers used by the web application developers as well as the policies the web application developers use. Therefore, it is necessary to focus on the client-side mechanism to provide stronger protection for the web application to store critical data from cybercriminals. There exist so many different types of attacks for targeting all kinds of vulnerabilities that are often contained in web applications.

Cross-site scripting attack (XSS) is a common vulnerability that is exploited in modern web applications by entering advanced HTML tags and Java Script functions. Little verification of entry on the web application can steal cookies from the web browser database. A cross-site scripting attack continuously leads lists of vulnerabilities in the most widespread web application (Wichers, 2013). Following are the types of cross-site scripting attacks: (al, 2010).

1.1 Persistent XSS (Stored Attack)

Is server-side attack, occurs when the injected script is stored on a publicly accessible area of a website. When a user visits one of these places, the browser will retrieve and present the data, which in turn will execute the XSS attack stored in the browser context. Figure 1 illustrates the flow of a typical Stored XSS attack.

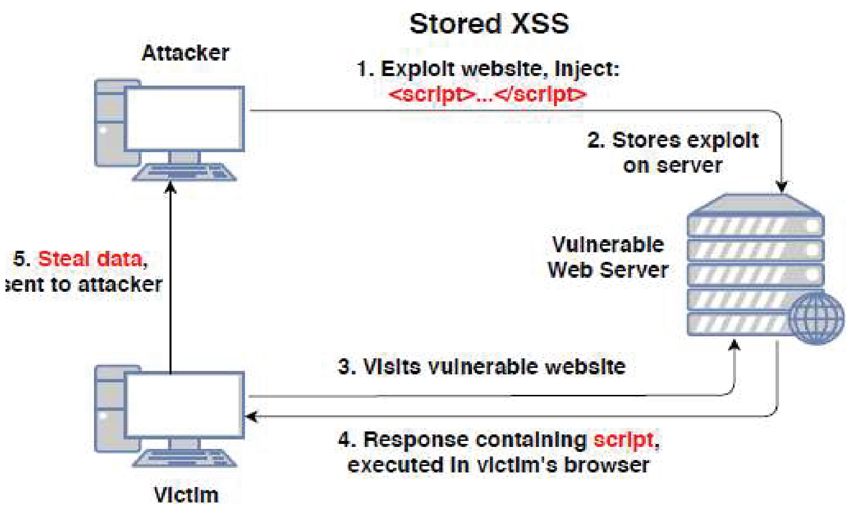

Stored XSS
Attacker
1. Exploit website, inject:
<script>...</script>
2. Stores exploit on server
Vulnerable Web Server
5. Steal data, sent to attacker
3. Visits vulnerable website
4. Response containing script, executed in victim's browser
Victim

Figure 1 Typical flow of stored XSS attack.

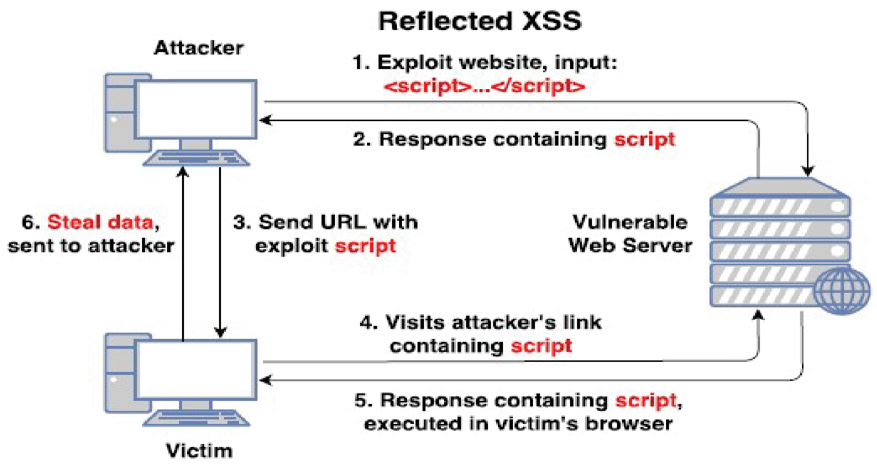

Reflected XSS
Attacker
1. Exploit website, input:
<script>...</script>
2. Response containing script
6. Steal data, sent to attacker
3. Send URL with exploit script
Vulnerable Web Server
4. Visits attacker's link containing script
5. Response containing script, executed in victim's browser
Victim

Figure 2 Typical flow of reflected attack.

1.2 Non-Persistent (Reflected Attack)

Occurs when a user input file during a request is shipped to an internet site, which instantly returns data in response to the browser, without the web site first ensuring. For the reverse XSS attack to work, the attacker needs to make the victim somehow require a special query that contains the malicious script, if a user visits this specific URL, the attack code will be run and executed in the user's browser. Figure 2 illustrates typical XSS attack flow.

1.3 DOM-Based Vulnerabilities

The DOM allows the page to interact with JavaScript page code, making the page more dynamic. This also makes it possible for malicious code to change the page if the JavaScript entry is not handled properly. If a website includes some JavaScript code in the response that directly uses input from an input source, like the URL, a DOM Based XSS might be executed. Figure 3 shows the data flow of an XSS-based DOM attack. These attacks can actually be executed without even sending the attack script to the web server at all, using a special Hypertext Markup Language (HTML) character, in the URL.

CDOM-based XSS attack is the most advanced and not well-known type. Indeed, many of the security vulnerabilities in this type of attack stem from the inability of web application developers to fully understand how it works.

White Security report that it's being neither organized nor simple for sites to fix and remediate XSS, it turns out to be evident that the client needs a few

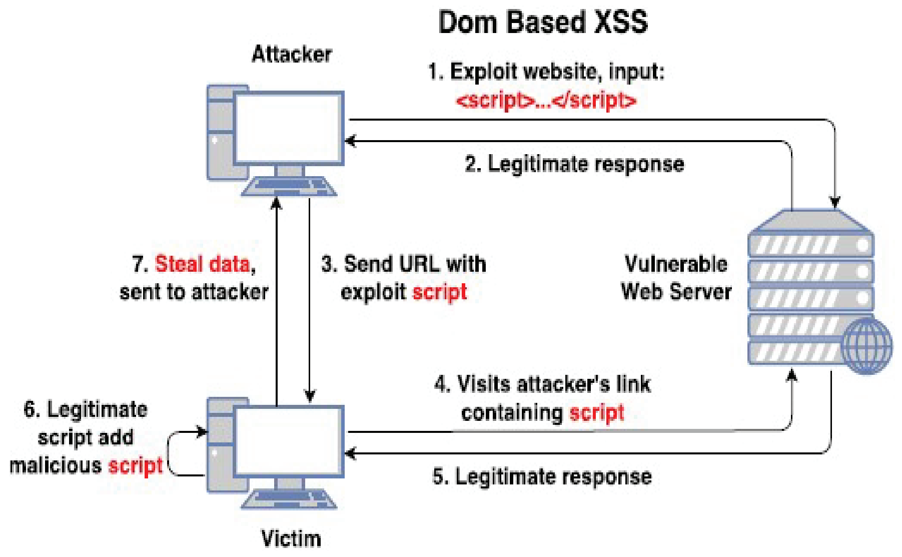

Dom Based XSS
Attacker
1. Exploit website, input: <script>...</script>
2. Legitimate response
7. Steal data, sent to attacker
3. Send URL with exploit script
Vulnerable Web Server
4. Visits attacker's link containing script
6. Legitimate script add malicious script
5. Legitimate response
Victim

Figure 3 The flow of DOM-base attack.

methods for securing themselves at the customer side since it's mostly the end-clients of weak web applications that are influenced by possible assaults. Among the absolute best 5 most utilized internet browsers (Internet, 2015) Mozilla Firefox is that the sole program that does exclude any very inherent sifting against cross-webpage scripting assaults, which may bargain clients inside the instance of a weak web application. Currently, XSS attacks are dealt with by fixing the server-side vulnerability, which is usually the result of improper input validation routines. Although it is a clear course of action, this approach leaves the user completely open to abuse if the compromised website is unwilling or unable to fix the security issue. A complementary approach is to protect the user's environment from XSS attacks. This requires methods to distinguish malicious JavaScript code downloaded from a trusted website from regular JavaScript code, or techniques to mitigate the impact of cross-site scripting attacks. (Rodríguez G. E., Torres, Flores, and Benavides, 2020).

Almost 25% of web applications are susceptible to XSS as mentioned in (Mewara, Bairwa, and Gajrani, 2014) (vulnerability report, 2014). Moreover, XSS within the top position among all other security vulnerabilities. Several anti-XSS defense techniques are proposed (Vogt, Nentwich, Jovanovic, and Kirda, 2007) (Introducing Content Security Policy, 2013), among these defenses, client-side filters are designed to supply browsers the power to mitigate various categories of XSS. Within the event of a reflected XSS attack, both the HTTP request and therefore the response from the server contain the corresponding attack payload. Filters that are implemented in browsers got to define and block the attack. Although in some browsers, client-side known XSS filters are employed by default. Unfortunately, they need their limitations to alleviate several XSS vulnerabilities categories. Within the worst case, these filters are often easily bypassed. Hence, we try to build a robust, effective, client-based security mechanism to protect web applications from cross-site scripting vulnerabilities; find the best filter that achieves the best security for our browsers.

In the next section we review several related works and make compression between the best filter that defense against XSS attacks. In Section 3 we describe our approach, and Section 4 evaluates the design, both in terms of correctness and performance. Section 5 contains our concluding remarks.

2 Related Works

Before going through this work, it is important to look at the existing related works, specifically paying attention to the current client-side filtering status of cross-site scripting attacks, which contains vulnerabilities and suggestions for further improvements. Based on this, "XSS Firefox Client-Side Filter" is the first client-side proposed solution for XSS (Vikne and Ellingsen, 2018).

In the paper (Bates, Barth, and Jackson, 2010) analyses existing client-side filters and techniques for cross-site scripting, before presenting a better solution based on a different design.

The analyzed filters use regular expressions to filter them, which the paper concludes are either unacceptably slow or easily circumvented. Then they introduce a new filter design that achieves high performance and high accuracy, by implementing the filter to block scripts after the HTML parser but before executing the script, and by using string match instead of regular expressions. The filter, XSS Auditor, was first implemented in the Web Kit rendering engine and is now enabled by default in Google Chrome.

Noxes (Vigna, Jovanovic, Kirda, Kruegel, Vigna, and Jovanovic, 2006). The first web intermediary was executed as a client-side answer for moderate XSS risks. It deciphers all solicitations and won't permit any solicitations to be sent across areas aside from some whitelisted URLs. This will guarantee that no touchy client data is shipped off some other zones. In any case, this methodology experiences countless bogus positives and controls the use of numerous kindhearted activities the same number of locales profit cross-area demands.

NoScript (Maone, 2012) is primarily supposed to be a tool, which interrupts execution of malicious scripts from maximum web pages. Though, obstructing the execution of scripts is not a genuine choice for web applications. For resolving this issue, NoScript also acts as a client-side XSS filter, which sanitizes the HTTP request transmitted via web browser.

Pelizzi and Sekar (Pelizzi and Sekar, 2012) analyzed some of the serious performance issues in the XSS-Auditor (al, 2010) and NoScript (Maone, 2012) and proposed a new XSS filter, that utilizes an approximate string matching algorithm for detecting the whole and partial-script injections. The algorithm checks the length of the parameters and the script of the web page. If the parameters are longer than the script, it searches within the parameters for the script for a whole script injection. If the script is longer than the parameter, it searches the script for the parameters for partial-script injection.

Many propose client-side defenses against XSS attacks and demonstrate how client-side XSS filters can reduce the impact of Cross-Site Scripting Attacks. Several client-side XSS codes try to mitigate XSS vulnerabilities by preventing an attacker from extracting that information to the server. The goal of these codes is to monitor the flow of information within the website's JavaScript environment. Because of the XSS attack, it is highly recommended to use both server and client-side filtering, so that you can protect against all types of XSS attacks and achieve a good defense in depth strategy. This paper focuses on client-side liquidation, which includes a discussion of the various current solutions presented in this section.

Client-side filters are in the client which typically would be the web browser used to access web applications. Client-side filtering may be able to detect DOM Based XSS attacks, providing the extra protection server-side filters are missing. However, even though client-side filters could possibly detect all types of XSS attacks, it should not be used alone, without server-side filters by placing the filter on the client-side.

Client-side XSS filters are an important second line of defense against XSS attacks. We warn web developers against relying on client-side XSS filters as a primary defense of security vulnerabilities in their applications, but we recommend that each browser includes an XSS filter to help protect its users from unpatched XSS vulnerabilities. Instead of using regular expressions to simulate the HTML parser, client-side XSS filters should integrate with the display pipeline and examine the response after analyzing

Table 1 Compares the latest client-side technologies to filter browser based on seven key factors

Related Work/ Key Factors	XSSFilt	XSSAuditor	IE 8.0	XSS-immune
Methodology	Approximate String matching	Exact String matching	Regular Expression based	String comparison and sanitization based
Web browser utilized for deployment	Firefox	Google Chrome 4	Internet Explorer 8, 9	Google Chrome
Category of an XSS worm Detected	Reflected, stored	Reflected, DOM-based	Reflected	Reflected, stored, DOM-based
Partial script injection Detection	Yes	No	No	Yes
Regular expression Utilized	No	No	Yes	No
Context-aware sanitization	No	No	No	Yes
False-positive rate	Medium	Medium	Medium	Low

it. Client-side filters are the best method to defense against XSS attacks. A lot of filters have been designed to protect the client against XSS attack. We will discuss some of these filters in this section.

A performance comparison mentioned in (Gupta, 2016). between IE8, NoScript, XSSAuditor, and XSS-immune shows that XSS-immune is the best XSS filter. See Table 1.

3 Proposed XSS Filter Methodology

In this section we describe the proposed XSS filtering algorithm (XSSFilter), how it is implemented and integrated into our system, also containing details about every part of the filtering process. The Algorithm aims to create XSS filtering model based on the filtering rules, that compares and test every script returned in the http response with every potentially dangerous script from the request. That helps achieve better, more accurate results and effective performance in relieving XSS attacks.

The XSSFilter focuses on primarily stopping Reflected XSS attacks via conduct a Context analysis to interpret and translate the content in the html code by building a set of rules to detect XSS vulnerabilities. These are the rules work through compare the contents of the request with the response to detect the Reflected XSS Vulnerability. It works by building an assortment of rules for XSS weakness detection:

1. Selected risk vectors are dispatched against contributions of the web application. Those risk selected attack vectors are launched against inputs of the web application. Those attack vectors are generally injected in a HTTP request as parameters or as fields in a web form.
2. The vulnerability filter receives the responses to the requests which contained the injected code.
3. The vulnerability filter checks for the presence of injected script in the received responses. If affirmative, XSS attack is considered successful and a vulnerability of the scanned web application has been discovered.
4. When discover the reflected XSS attacker the filter will be closed and block the website.

The proposed filter applies numerous techniques for dealing with the various stages required in the test and filtering process measure. This process contains a series of different tasks that are performed in a particular order, before concluding whether there exists a cross-site scripting injection or not. It includes methods for fetching the input from the request to different methods for comparing this data with either inline script, external script, or on-event handlers, all of which need to be processed differently and it proxies the HTTP(s) requests from a browser sends them to different modules/plug-ins for vulnerability scanning. The figure below shows the main improvement in our method:

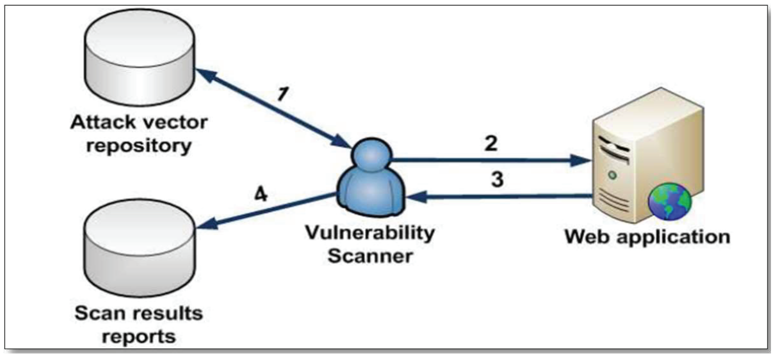


Figure 4 Current reflected-XSS detection system.

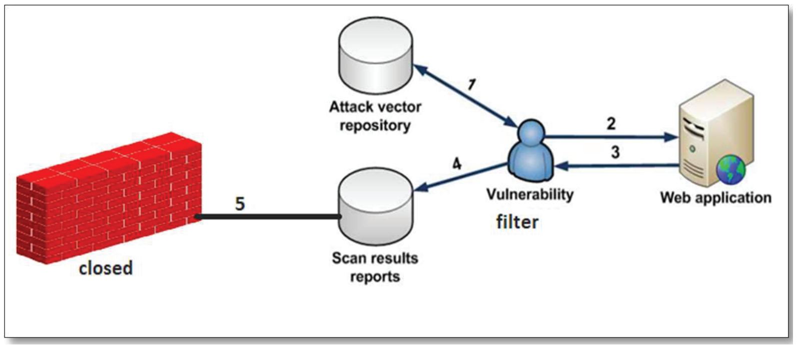


Figure 5 The proposed reflected-XSS filter detection system.

XSSFilter utilized in this strategy is Regular Expression in Xpath is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. in XSS filter takes at URL filter By adding specific URLs with patterns containing text and regular expressions can allow, block, exempt, and monitor web pages matching any specified URLs or patterns. In this section, we have decided to discover reflected XSS weaknesses.

3.1 XSS Filter Modules

Now, we will start by a brief review of the Algorithm implementation, we divided it into the following four components.

- Raw HTTP demand parser
- Initial prober
- Context analyzer
- Payload generator

3.2 Raw HTTP Request Parser

The first step is related to the raw HTTP request parser that takes input from a file and converts it into a request object. HTTP is one of several protocols (communication strategies) used to transfer data from one device to another over the Internet. It is a protocol that browsers mainly use to communicate with websites.

The HTTP message handling framework is designed to be expressive and flexible while maintaining memory efficiency and speed. It uses the same

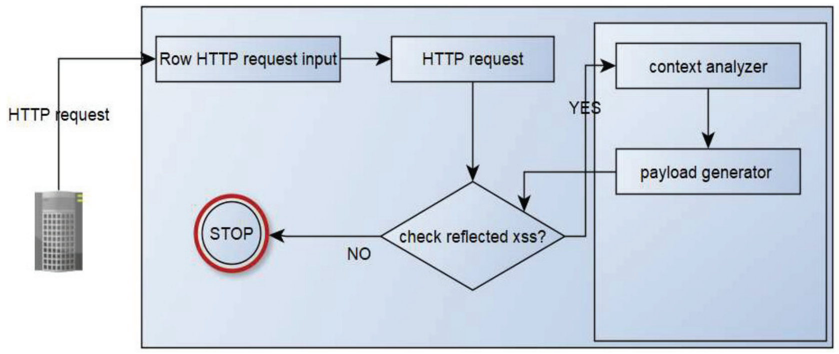


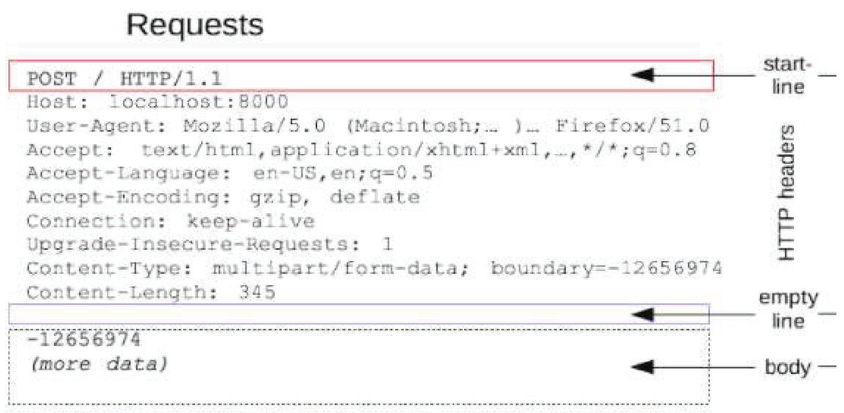
Figure 6 XSSFilter architecture.

API to parse, format, and implement HTTP messages, which helps ensure consistent behavior of HTTP services regardless of the I/O model. Analyze the HTTP request message and create an object model for it that can be used both on the server and the client. In this step, we have set up our environment. Here, our project method implements the raw HTTP request, let the Flow builder control all aspects of the HTTP request. For example, this allows us to connect to XSS services, connect to HTML services, or for many other advanced uses.

This step aims to build an environment based on passing HTTP requests to take the inputs from the file, in short, an environment that receives the input. This means to start by fetching all the input data to the website in form of GET and POST parameters, before checking each of these parameters if they contain any potentially malicious code that can be used for executing a cross-site scripting attack. Raw HTTP request parser would be a raw HTTP request parser that takes input from a file and converts it into a request object. Then we convert the body and request parameters into a DICT so that we can easily parse and add our payloads.

Request.txt: is a text file consists of the needed instructions to achieve the first step; HTTP exchanged between a server and a client. HTTP requests are messages sent by the client to initiate an action on the server. They have three components: HTTP method, HTTP headers, and Body as shown in Figure 7.

After we have done this step passing the input HTTP request and transform it to inputs proportional to the environment. Once we are done parsing, we need to find a way to insert a probe in request params as well as the post body and check whether if it is reflecting in the response or not. To do this



```
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;… )… Firefox/51.0
Accept: text/html,application/xhtml+xml,…,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345

-12656974
(more data)
```

INITIAL PROBER

Figure 7

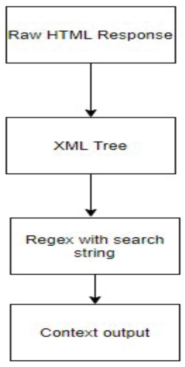


Figure 8 Steps to context analyzer by LXML.

we are going to use the requests package in python to parses params and body to create a list of requests objects with probes as a payload. The next step will send each request and check which parameter value is reflected in the response. We mean it takes a copy of the request that contains all the request contents, and we will need to find a way to insert a probe in request parameters and send it to the server. The risk-filter checks for the presence of infused content in the got reactions. On the off chance that positive, the XSS risk is viewed as effective and weakness of the examined web application has been found. Means if a file reflects in the HTTP response, we can conduct a Context analysis to interpret and translate the content in the HTML code, and if there is no reflection in the response, the scanning process stops.

3.3 Context Analyzer

Context Analyzer is one of the most important stages in this algorithm and building a context analyzer is the most difficult part. To write a context analyzer we use LXML package that parses HTML into an XML tree. It takes the HTML code and probe as inputs, and then returns a list of contexts in which XSS vulnerability may occur, because XSS vulnerability appears in many HTML contexts (HTML Tags-HTML Attribute Name-HTML Attribute Value-HTML Text Node-HTML Comments-Style-Style Attribute-Href Attribute-JS node). We Analyze it through XPath regular expression in LXML XPath. Figure below shows the steps.

Our goal here is to download the HTML contents of the main pages. Regular Expression in Xpath helps with using part of a locator attribute that stays constant to identify a web element on a web page. Sometimes the values within the HTML code for the attributes change. For instance, attributes would change every time the web page you are working on is refreshed as well as the contents of all hyperlinks on the main page. after a raw HTML string is parsed and converted into an XML tree. Then we will search the XML tree using RegEx to find the context in which the string has been reflected. If a string is found, then we extract all the codes and perform analysis and translation of the content in the code to detect the XSS vulnerability. The benefit of setting the context is in the next step of the algorithm.

3.4 Payload Generator

After a raw HTML string is parsed and converted into an XML tree, then scanning the XML tree using RegEx to find context in which the string has been reflected and successfully created a context analyzer. What is left is to create payloads based on the context and confirm those payloads. For this part, we created payload generator function that takes contexts and returns a list with RegEx to find in the XML tree and the payload. The importance of this stage is that it takes the context that contains injections to make corrections, modifications, and substitutions contexts with other codes, the test process is repeated to confirm the vulnerability is addressed. The last step is creating a new HTTP request does not contain XSS vulnerabilities.

If vulnerability is discovered, the filter will prevent it from being executed within the user's browser by creating a new modified request without injection. This request contains the modifications that were implemented in the previous stages. The new result is displayed by displaying the modified requests.

3.5 XSS Filter Implementation

The above four components implemented in four modules coded in python utilizing LXML and solicitations libraries and developed under Windows 10 Pro operating system – PyCharm (IDE).

Module 1: This step aims to build an environment based on passing HTTP request to take the inputs from the file and when using the filter, and Before checking each of these parameters if they contain any possible malicious code that can be used to conduct a cross-site scripting hacking, it starts by fetching all input data to the website in the form of GET- and POST-parameters. Then the filter will continue its examination of all input in the HTTP request.

Module 2: We Pass the input of the HTTP Request and convert it into inputs appropriate to the environment. This is considered the objectives of this stage. Also, we need to find a way to insert a probe in request parameters as well as the post body and check whether if it is reflecting in the response or not.

Module 3: first we use regular expression in LXML XPath to implementation the filter. In this step our goal is to download the HTML contents of the main page, as well as the contents of all hyperlinks on the main page using the LXML tree, which supports the simple path syntax of the find. a context analyzer is going to use a package called LXML that parses HTML into an XML tree. We will search the XML tree using regex to find the context in which the string has been reflected. If finds for a string. The path always collects all results before returning them.

Module 4: The most important in these Modules is repeating the request in the browser to confirm the vulnerability. In this XSSfilter, we will work to repeat the request in the browser to confirm the vulnerability, and in the event, takes the context that it contains injections. If the vulnerability is discovered the filter will prevent it from being executed within the user's browser, and the new result is displayed by displaying the modified requests.

4 Results and Discussion

Client-side XSS filters do not need absolute correctness to be of use. However, the effectiveness of the filter depends on the percentage of vulnerabilities protected by the filter, and the rate of false-positive and false-negative. Our system comes with built-in XSS protection. XSS Filter uses the idea of matching inputs to outputs to detect XSS vulnerabilities. In practice, when testing the implemented filter, simple cross-site scripting attacks were successfully detected and blocked by scanning all request inputs and comparing them with the response and evaluating the context in which the injection was made. In our system, we use this website to test the code, the website contains reflected XSS injection. (testPHP)

The filter puts a specific word in the request and sends more than one request for one site and then searches for it in the result. In case there is a discovered vulnerability, it prints the result. If the vulnerability is not discovered, it prints a message that this site does not contain XSS vulnerability. The filter works to test each input to the site separately by analyzing the link, meaning that for each element in the perimeters it sends a request modified by this perimeter, and also for each element in the body sends a new request, and so on. In the event of success in penetration, it should be printed that this perimeter is permeable.

5 Evaluation

In this sub-section, we assess the accuracy and efficiency of our client-side XSS filter and we present our evaluation of the XSS Filter. We first define our criteria and metrics for evaluation using the threefold criterion, these are Performance, Protection Effectiveness and Usability. we will present a thorough evaluation of XSS Filter. Finally, we compare XSS Filter with other XSS approaches.

5.1 Performance

Performance is a crucial factor in determining the utility of a client-side XSS filter. Browser vendors are reluctant to deploy features that slow down key browser benchmarks, including JavaScript performance and page load time. By providing the interface between the browser's HTML parser and JavaScript engine, XSS Auditor achieves high performance and high reliability. The semantics of an HTTP response are analyzed by Post-parser. Blocks suspicious attacks avoid the passing of the inserted script to the JavaScript engine instead of risking changes in the HTML code. In XSSFilt, however there are two logical steps involved in XSSFilt operations.

(a) Realizing script content on the victim page.
(b) The identification of whether this code derives from requested data.

While the completeness of their regular expressions depends on IE and NoScript, they require to strike a complex balance between usability and security, but in XSS-immune, the highest observed value of XSS-immune performance on these web applications is almost 98 percent. In comparison with other filters, there was the lowest number of false positives and enough false negatives are observed in all the platforms of web applications.

As for our approach, it is like the IE 8 filter to make a comparison between the request and the response, but the difference here is in this approach. Our system sends request according to perimeter and puts in each perimeter a test word (teyascan) then compare it with the response. If it comes back in the response, it prints that perimeter is hackable.

To calculate the efficiency of the filter, the filter should be able to detect and avoid the execution of the inserted script. This is being tested by an automated test consisting of several separate script injections. in XSS Filter we tested our system by Vulnerable test websites for Acunetix Web Vulnerability Scanner (acunetix).

Xspear is a scanner for web vulnerabilities. It is a full web application security testing solution that can be used in both stand-alone and complex environments. Xspear is a powerful XSS scanning and parameter analysis tool on ruby gems, capable of both static and dynamic XSS vulnerability analysis. Therefore, it can scan, detect, and analyze potential XSS vulnerabilities on web applications. From XSpear features is Testing request/response for XSS protection bypass and reflected (or all) params – Reflected Params and Filtered test event handler HTML tag Special Char Useful code – Testing custom payload.

To evaluate the testing of XSSFilter to make sure the modified browser ran the code for our implemented filter, we used the search function on each of the websites and tested with two different parameters, one safe and one unsafe to compare the load time between the modified browser and original browser. We found that our system was able to detect 3 sites out of 5 that they were tested. And After performing a link analysis, it was able to print that the site is vulnerable. And If there is no vulnerability, a message appears stating

Table 2 The detection XSS in the 5 sites

	Site 1	Site 2	Site 3	Site 4	Site 5
XSSFilter	✓	✓	✗	✓	✗
Xspear	✓	✗	✗	✗	✗

that the site is secure. However, a variety of factors may have influenced performance testing. These variables, which may have had a greater effect on the results, are the variations in the local Internet speed of the test machine and the fluctuations in web traffic received by the websites examined at the time of the test. These factors typically differ during the day, depending on the moment.

5.2 Filter Efficiency

Without waiting for websites to repair them, client-side filter methods defend users from XSS vulnerabilities. Internet Explorer 8(12) comes with XSS security built in the approach of IE8, the uses the concept of balancing inputs to outputs, IE8 aims to provide ordinary users with security. It has been shown that their regular expressions are inadequate for security.

XSSAuditor (14) is the name of a Google Chrome-integrated XSS filter. This filter aims at a modern architecture that can only be used for browser defenses. This technique has many features, the most important being interposing on all script evaluation requests, defeating browser quirks, and uncommon attack vectors. XSSFilt has too enjoys features. However, XSSAuditor relies on full string matching, unlike XSSFilt, and can therefore skip risks because of the application-specific sanitization. But the XSSAuditor does not detect partial script injections. For XSSFilt, XSSAuditor is 95 percent versus 99.75 percent. Meaning, however, skewed towards simple vulnerabilities adding a script tag. The XSS filter of NoScript performed well against both datasets.

Specifically, however, NoScript is looking for JavaScript syntax and simple Java-Script functions including a warning. NoScript's method suffers from a higher rate of false positives, since NoScript sanitizes outgoing requests rather than incoming responses. It cannot confirm if the offending content is present in the response, which leads to JavaScript code execution. It cannot confirm if the offending content is present in the response, let alone if it leads to JavaScript code execution. But in our experiments, we were able to add built-in XSS defenses to our system. Much like IE8. The XSS filter uses the matching input and output theory to identify vulnerabilities in XSS, and put a test word to compare whether it is reflected in the response or not. The efficiency of the filter revolves based that after the process of analyzing the code, the filter makes some corrections, modifications, and substitutions of contexts with other codes.

5.3 Usability

Usability refers to the consistency of user experience when communicating with goods or systems, like websites, apps, devices, or applications. Usability is about performance, efficiency, and overall user satisfaction. In XSS Parser, the filter searches for malicious strings in outbound requests and relies on a regular expression corresponding to that string. The same pattern is then searched in the answer of the server. XSS filters depend on the completeness of regular expressions that need to strike a complicated balance between usability and security.

Aspects of our framework are not to block the entire web page. User experience will be significantly impacted by a lot of discovered attacks, which would allow consumers to use regular browsing behaviors without disrupting or delaying their duties.

XSSFilter system is easy to use and fast in detecting and stopping vulnerability in a short time. Our XSSFilter has the advantage of being fast in obtaining results to discover the vulnerability and it changes the order page with high accuracy.

5.4 Comparison Between All Filters

In research (Gupta, 2016), the performance analysis of existing client-side XSS filters (i.e. IE8, NoScript, and XSSAuditor) was also compared with XSS-immune. The result in the table shows that the XSS-immune is the best XSSFilter and they compare it with the current XSS defensive methodology.

Most current XSS filters are unable to detect partial injection of XSS worms. Besides, a lot of pre-processing is required in the current architecture of web applications for efficient deployment on various web browser platforms as well as web applications. Context-conscious sterilization is simply avoided by most of these sterilization techniques. While XSS attack vectors are sterilized in a way that is not context-sensitive, this form of conventional sterilization method is easily bypassed by attackers. As a consequence, an unacceptable rate of false-positive and false-negatives is found in current XSS filters.

False positives are misclassified security warnings, suggesting that there is a danger when there is no one. The benefit of the filter is dependent on the percentage of vulnerabilities protected by the filter and the rate of false-positive and false negatives. This table shows the number of false positives for each candidate and shows a comparison of several false-positive filters. As it is understood, false positives typically cause further disruption to users.

NoScript will result in a higher rate of false positives. As NoScript sanitizes outgoing requests rather than incoming responses, it cannot confirm whether the offending content occurs in the response. On the other hand, XSS Auditor only prevents the execution of the offending script but typically does not prohibit the display of the main content of the website.

6 Conclusions

When talking about web protection, which is an overly wide area, but security goals are critical when developing stable web applications. To be able to accommodate all of them, web applications need to defend against a variety of different attacks by malicious actors seeking to steal their data and the data of their users. This is not a simple job, since there are so many different forms of attacks targeted at all sorts of vulnerabilities that are often found in web applications. The most common security threat is cross-site scripting (XSS). This paper presented a thorough study and comparison of many popular XSS filters, NoScript and XSSAuditor, XSS-immune, XSSFilt, identifying their weaknesses and proposing a new filter, XSSFilter. This paper describe the implementation of the filter, XSSFilter filter compares the request with the response, after a raw HTML string is parsed and converted into an XML, the filter detect a vulnerability and then prevent it through by creating a new modified request without injection. The new result is displayed by modified requests.

We showed that our proposed XSSFilter is easy to use and fast in detecting and stopping vulnerability in a short time. Our system in XSS Filter has the advantage of not being late in obtaining results to discover the vulnerability and it changes the order page with high accuracy by creates a correction and modification of the HTML request page before submitting and displaying it, to avoid reflected XSS vulnerabilities by altering the HTML request.

References

[1] Gupta, S. (2016). "XSS-immune: a Google chrome extension-based XSS defensive framework for contemporary platforms of web applications," Secur. Commun. Networks, vol. 9, no. 17, pp. 3966–3986.

[2] acunetix. (n.d.). https://www.acunetix.com/vulnerability-scanner/. Retrieved from acunetix.

[3] al, B. e. (2010). Mozilla Developer Network. Recuperado el, 1. Bates. (2010).

[4] Bates, D., Barth, A., and Jackson, C. (2010). Regular expressions considered harmful in client-side XSS filters. Paper presented at the Proceedings of the 19th international conference on World wide web.

[5] Christey, S., and Martin, R. A. (2007). Vulnerability type distributions in CVE Mitre report. OWASP Foundation.

[6] Grossman, J. (2007). Whitehat website security statistics report. White-Hat Security.

[7] Hydara, I., Sultan, A. M., Zulzalil, H., and Admodiasaso, A. (2015). Current state of research on cross-site scripting (XSS) – A systematic literature review. Information and Software Technology.

[8] Internet. (2015). https://www.alexa.com/topsites. Retrieved April 2020, from Alexa.

[9] Introducing Content Security Policy. (2013). Retrieved March 2020, from https://developer.mozilla.org/en/.

[10] lxml. (n.d.). https://lxml.de/. Retrieved from lxml.

[11] lxmlpath. (n.d.). https://lxml.de/xpathxslt.html. Retrieved from lxmlpath.

[12] Maone, G. (2012). NoScript-JavaScript/Java/Flash blocker for a safer Firefox experience. In.

[13] Mewara, B., Bairwa, S., and Gajrani, J. (2014). Browser's defenses against reflected cross-site scripting attacks. Paper presented at the 2014 International Conference on Signal Propagation and Computer Technology (ICSPCT 2014).

[14] Mozilla. (n.d.). https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept. Retrieved from Mozilla Developers.

[15] Nava. (2010).

[16] Nava, E. V., and Lindsay, D. (2009). Our favorite XSS filters/IDS and how to attack them. Black Hat USA.

[17] Network. (2014).

[18] OWASP, T. (2017). The Ten Most Critical Web Application Security Risks. OWASP Foundation.

[19] Pelizz̀i. (2012).

[20] Pelizzi, R., and Sekar, R. (2012). Protection, usability and improvements in reflected XSS filters. Paper presented at the proceedings of the 7th ACM Symposium on Information, Computer and Communications Security.

[21] Rodríguez, G. E., Torres, J. G., Flores, P., and Benavides, D. E. (2020). Cross-site scripting (XSS) attacks and mitigation: A survey. Computer Networks.
[22] Rodríguez, Torres, Flores, and Benavides. (2020).
[23] Stock, B. (2014). "Precise client-side protection against DOM-based cross-site scripting," in 23rd {USENIX} Security Symposium ({USENIX} Security 14), pp. 655–670.
[24] testPHP. (n.d.). http://testphp.vulnweb.com. Retrieved from TestPHP.
[25] Vigna, Jovanovic, Kirda, E., Kruegel, C., Vigna, G., and Jovanovic, N. (2006). 1Noxes: a client-side solution for mitigating cross-site scripting attacks. Paper presented at the Proceedings of the 2006 ACM symposium on Applied computing.
[26] Vikne, A., and Ellingsen, P. (2018). Client-Side XSS Filtering in Firefox. In: SOFTENG.
[27] Vogt, P., Nentwich, F., Jovanovic, N., and Kirda, E. (2007). Cross Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. Paper presented at the NDSS.
[28] vulnerability report. (2014). Retrieved March 2020, from https://www.infopoint-security.de/medien/cenzic-vulnerability-report-2014.pdf.
[29] Wichers, D. (2013). OWASP TOP 10-2013. OWASP Foundation.

Biographies

Khulud Fisal Alenzi has received her BS and MSc degrees from University of Tabuk in 2015, 2020 respectively, College of Computers and Information Technology. Her major research interests include Cyber Security and Web Applications.

Onytra Abbas Bashir Abbas received her PhD in AI (text summarization and caching in mobile web application) from Sudan University of Science and Technology, Sudan (SUST) 2012. She is currently Assistance Professor in the College of Computers and Information Technology, University of Tabuk. Her major research interests include Cyber Security, Machine Learning and Web applications.