International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-7, Issue-6, March 2019

Exploitation of Cross-site Scripting (XSS) Vulnerabilities and their Prevention on the Server-side

K. Joylin Bala, E. Babu Raj

Abstract: ***Web applications actively replace native applications due to their flexible nature. They can be easily deployed and scaled, which require constant interaction with the user machine for software updates. Widespread use of cloud computing [10] has resulted in favoring web applications for easy deployment and scalability. Today the movement of software applications to the web has resulted to web application vulnerabilities [1]. Instead of targeting multiple operating systems or platforms, attackers can focus on exploiting web applications for compromising sensitive information. Web browsers act as the interface between the user and the web and are crucial for user security. The client-side attacks can result in the compromise of credentials and identity theft. In this paper, totally three models are developed namely Injection of code into un-sanitized parameters, Browser exploitation techniques and Manipulation of application registries which serve as the basis for exploiting and subsequently preventing cross-site scripting vulnerabilities [3]. By using these models as a foundation, the attacks are minimized in a large scale. In this work the results shows that, for the random sample of attack vectors 4, 2, 9, the vulnerability score is 0, which is considered to be minimum and forth random sample of attack vectors 2, 5, 7 the vulnerability score is 89.12 which is considered to be maximum. This work aims at developing a solution in web applications undergo rigorous testing by being a target to the engine and consequently finding flaws embedded within them.***

Keywords: ***client-side scripting, cross-site scripting, exploitation, server-side scripting, prevention, Web application.***

I. INTRODUCTION

Numerous research studies have been done on the execution of code by providing malformed input, usually known as payloads [2], through parameters accepting user input. It can be reckoned that these research studies focus on languages such as PHP and Java. Popular content management systems also written in these languages containing vulnerabilities are also covered in extensive detail. Providing client-side script input generally with the intent of compromising certain types of data to end users is known as a Cross-site scripting attack. A cross site scripting attack is a very specific type of attack on a web application. It is used by hackers to copy real sites and cheat people into providing personal data [16].XSS is a very common vulnerability found in Web applications and allows the attacker to inject malicious code [8], the reason of that is the developer trusts user inputs, then send back user input data to the client browser so the malicious code will execute.

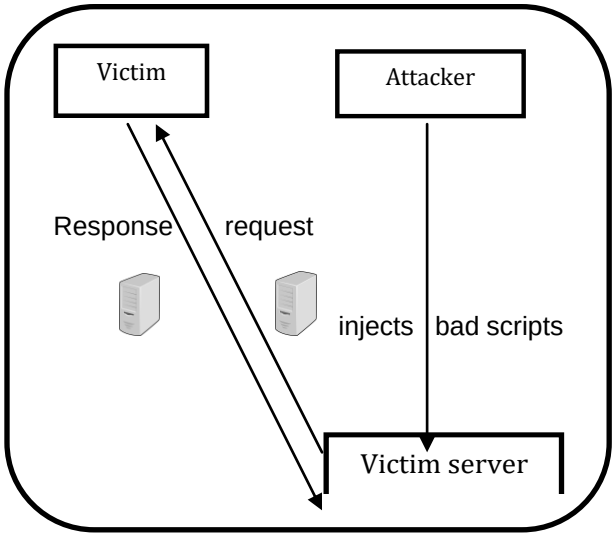

Figure 1: Architecture of XSS attack

Injection of code into un-sanitized parameters, Browser exploitation techniques and Manipulation of application registries which serve as the basis for exploiting and subsequently preventing cross-site scripting vulnerabilities [3]. By using these models as a foundation, the attacks are minimized in a large scale. In this work the results shows that, for the random sample of attack vectors 4, 2, 9, the vulnerability score is 0, which is considered to be minimum and forth random sample of attack vectors 2, 5, 7 the vulnerability score is 89.12 which is considered to be maximum. This work aims at developing a solution in web applications undergo rigorous testing by being a target to the engine and consequently finding flaws embedded within them. Rapid application deployment is an important characteristic of web applications along with rapid application development. Deploying web applications is easier than distributing executable of native applications. Native applications require compilation for different architectures and operating systems. Libraries supporting an operating system may not be compatible with another

Revised Manuscript Received on 30 March 2019.
***** Correspondence Author
K.Joylin Bala*, Department of Computer Science, Bharathiar University, India
E.Babu Raj, Department of Computer Science, Marian Engineering College, India

operating system which requires code rewrite and duplication. Web applications solve this problem by relying on the browser as the only environment to support. A popular example of this can be given by the node runtime environment which is built in top of the Chrome V8 engine. Applications can be written in JavaScript [11] by taking advantage of numerous frameworks available for the JavaScript ecosystem. Current user trends very much suggest that ecosystem of personal computers consumers revolve also around web applications along with native applications. A popular example would be the movement of word processing to the web with Google Docs in comparison with Microsoft Word. One of the major advantages of web applications is the possibility to make frequent changes which can result in additional features and more stability. This advantage has given rise to processes like continuous integration and continuous delivery CI/CD. CI/CD makes use of systems which automates the process of building, testing and deploying code. This results in increased code updates and faster release cycles which results in more stable systems and more features. The process of delivering the artifacts to the end user is eliminated and users can access the application just by navigating to an application endpoint URL [12]. Due to the prevalence of standards between web browsers, developers can spend more time in implementing features than checking for compatibility or porting the same functionality between different types of systems. Even though there have been cases of browsers not supporting web application features, the user statistics always skew to browsers implementing this features along with increased performance and more stability. Algorithms to mitigate this type of attacks by sanitizing inputs both client-side and server- side have been introduced and implemented in numerous web applications currently. We provide a detailed survey of this various techniques in a more concise and abstract way in the upcoming sections. In this paper, we introduce improved algorithms and frame models on them which we consequently implement as a program. Even though such traditional web applications are very much relevant, there has been an increased trend in the development of single page applications and progressive web applications. In the essence of these applications, most of the applications logic is applications are made possible by JavaScript frameworks such as React, AngularJS and Vue. In this paper, exploitation of cross-site scripting vulnerabilities will be focused on these single page applications to improve existing attack methodologies. This approach has a number of advantages since native applications in the desktop and mobile ecosystem are continuously being replaced by hybrid applications. In this paper, three models are developed, which serve as the basis for exploiting and subsequently preventing cross-site scripting vulnerabilities. By using these models as a foundation, we develop an engine that can be automated to overcome the attacks on a large scale. This paper aims at developing a solution in which web applications undergo rigorous testing by being a target to the engine and consequently finding flaws [13] embedded within them.The rest of this paper is structured as follows. In Section II, the review of several research papers are summarized. Section III depicts the steps to exploit XSS vulnerabilities. In Section IV, I have suggested the models which we have used as the basis for the prevention of XSS vulnerabilities. In Section V, the metrics data representation for a random sample is to be presented. The Section V explains the proposed work. In Section VI, the Experimental Result are Discussed and Section VII provides conclusion.

II. RELATED WORKS

Many algorithms were developed for the prevention of XSS vulnerabilities on both client-side and server-side. Here, we view a few researches related to XSS vulnerabilities and their prevention. XSS-GUARD by PrithviBisht and V.N. Venkatakrishnan insist the un-trusted input which favors XSS attacks. The filter cannot reject any script input. Proper filter can be used for filtering [15] the input and thus we can reduce the effect of XSS vulnerabilities. Cross-site Scripting Prevention with Dynamic Data Tainting and Static Analysis by Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, Giovanni Vigna suggest that by restricted use of web browser [4] we can limit XSS vulnerabilities especially Firefox. But a web based application must be applicable to most of the browsers. Amit Klein in DOM based XSS insists firewall which establishes a barrier between a trusted internal network and un-trusted external network. The Firewalls fail to filter malicious code. Firewall is a network security device that grants or rejects network access to traffic flows between an un-trusted zone and a trusted zone. Engin Kirda, NenadJ ovanovic, Christopher Kruegel, Giovanni Vigna tells about Noxes which is a client-side solution for mitigating XSS attacks [5]. It is limited to Internet Explorer and Mozilla. But it must be applicable to most of the browsers. Joaquin Garcia-Alfaro and Guillermo Navarro-Arribas say filtering- and analysis-based proposals are the standard defense mechanism and the most deployed technique for Ajax based applications which is a high client-side processing [6]. But it is limited to web browsers and any user whose browser does not support JavaScript or any related language, this functionality disabled, will not be able to properly use pages that depend on Ajax.XSS classifier by ShailendraRathore, Pradip Kumar Sharma, and Jong Hyuk Park suggest that their study is based on machine. They have suggested ten different machine learning classifiers which are used on a prepared dataset to classify wepages into two categories: XSS or non-XSS. It is inadequate because it must be independent to most of the machine learning classifiers which is impossible to many cases. We can't limit any extent of the usage of machine learning classifiers. Also, it is limited to SNS (Social Networking Site) [7] environment and it is unsuited to all web applications. The rapid detection [10] method was suggested based on Skip List, rapid detection of cross-site scripting attacks by using some steps like creating a skip list signature, detecting suspicious strings, marking attack vectors etc. is enough to detect or prevent XSS attacks. It is unsatisfactory because it focuses on a few steps.

III. EXPLOITATION OF XSS VULNERABILITIES

Exploitation is the first and foremost step to prevent XSS attacks. Exploitation can be done in many ways. Here three steps are followed to exploit XSS vulnerabilities. They are Injection of code into un-sanitized parameters, browser exploitation techniques and manipulation of application registries.

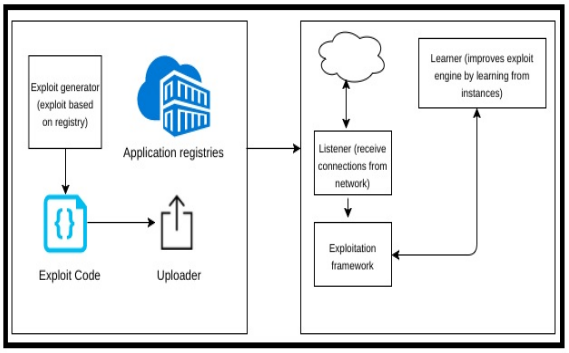

Figure 2. Exploitation of XSS vulnerabilities

Web applications receive input from the user to provide expected results. Such input derived from the user may be trusted or un-trusted. There are various methods to get the required input from the user, but the control of the input flow to the application should be restricted based on trustworthiness. But trustworthiness of an input is a qualitative measure which can't be precisely calculated. This opens the opportunity to send malicious or cleverly crafted code to the remote web server of which trustworthiness can't be determined. If the input received by the web server is not checked properly by the web server, then it is possible for the web server to execute the code send in the request. In the context of sanitizing data there are a lot of ways one can follow. Sanitizing the Uniform Resource Locator (URL), HTTP (Hyper Text Transfer Protocol) referrer objects, GET and POST parameters [9], various properties like document.location, document.referrer, cookies, headers and database data. Encoding [14] HTML [20] tags and user input along with special characters with the PHP functions strip_tags() and htmlspecialchars() can be effective in most contexts. Finding vulnerable points in the web application can be done either by inspecting the source code of the web application or by sending requests and inspecting the output of the web application. Our model consists of finding vulnerable points in a web application by randomly Fuzzing it with payloads. Fuzzing is the process of sending a request after crafting it with the necessary parameters. Source code of web applications may not be available in most cases and thus Fuzzing technique is used more often to find vulnerable web applications. Payload is the code, which want to be executed on the target web application. The payloads are executed using following steps,

Step 1: Craft various types of payload and collect it in a readable information exchange format.

Step 2: Create an interface which uses HTTP to send the crafted payloads.

Step 3:Connect the interface with the Fuzzing framework to handle requests

Step 4: Automate the process and collect output information to pass it to other models.

A. Browser exploitation techniques

The code runs primarily in the web browser. By exploiting the web browser, user code can be run on the user's operating system. Browsers provide various security measures to prevent remote code execution by implementing features like sandboxing, but it is still possible to execute code on the user's machine by using code vulnerabilities in the browser [17]. We demonstrate a case in which a browser exploitation framework can be used to try exploiting the vulnerabilities present in the system. The known vulnerabilities are included as a module in the framework which can be used in the exploiter. The framework also provides an API which can be used to extend functionalities from our engine.

B. Manipulation of application registries

Applications registries provide users and developers to collaborate by allowing developers to upload packages to them. Uploaded packages can be then used by other users and developers in their projects. Validation of packages is generally carried out by peer review of other developers using the packages. By following a packing structure code can be uploaded to application registries and allowed to be used by other developers. Since the validation of code is solely done by other developers it is possible to upload a malicious package to the registry. We develop an approach by which we collect various popular use cases of these packages and create corresponding packages with our code. By carrying out this approach in large scale, our code may get into applications which use these packages. This attack can be made worse by submitting pull requests to existing repositories with our package in the changes. For our code to create impact we can collect sensitive information from web pages including our package. One of the most used methods to submit information is through HTML forms. We develop an algorithm to get our target information from the page.

Algorithm 1: Finding Sensitive Data

Step 1: Check for <form> tags by iterating through the elements of the page using DOM.

Step 2: Check for sensitive element names and collect them into a list.

Step 3: Check the page for sensitive keywords and if present make the page a target.

Step 4: If any of the two conditions are true for any given page, then do.

a. Grab the data from the fields of the forms

b. Grab the cookie for the web page

c. Encode the given data to send it to a remote server

d. Send the encoded data to a remote server controlled or owned Trainable

Exploitation Engine (TrExEn).

IV. PREVENTION OF CROSS-SITE SCRIPTING ATTACKS

The models which we developed for exploitation of XSS are used as the basis for the prevention of XSS vulnerabilities. Our models for exploiting web applications can be automated to carry out the process in a large scale. To enclose all the models into a manageable system we develop an engine (TrExEn) which can execute the models. The overall architecture contains four models. They are Active Injector, Application manipulator, Browser exploiter and Intelligent learner.

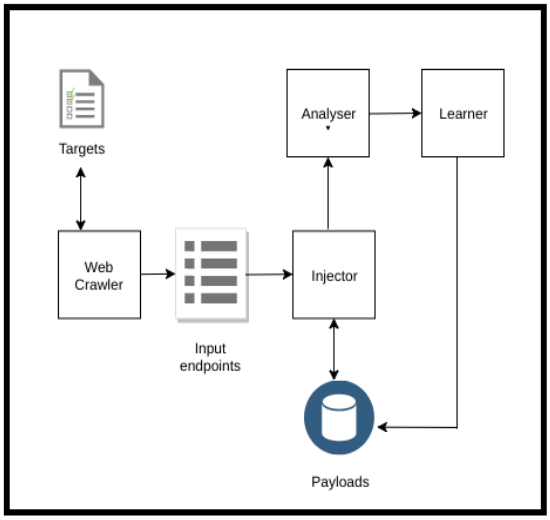

Figure 3: Overall Architecture models

A. Active injector

To find the vulnerable points in the web application we continuously fuzz the pages present in the web application. As described in our code injection model, we use our crafted code strings to inject into various input points in the web application. Responses from the pages are collected and then analyzed manually or through specifications in the module code. We iterate through all the input methods in the page and co Algorithm. Select a method and execute as step from the available models.

B. Application manipulator

The success of this module entirely depends on the ability to convince other developers to include the package into their web applications. To increase the chances of including the package into a web application the use cases should be well researched and analyzed. This module can accept a use case and necessary code to generate a package which contains the code to be executed. Directory structure and uploading to the application registry is automatically taken care by this module.

C. Browser exploiter

This module uses the browser exploitation model to check for available vulnerabilities in the user's browser. The findings are then categorized and arranged to find the available exploits for the vulnerabilities. Browser Exploitation Frameworks are extended and then interfaced through the module to automate the process. Browser exploitation frameworks have features to select versions and operating systems of the target and create a hook. A hook is a piece of code that when ran on the user's browser allows us to obtain code execution possibilities. If the exploit fails then the framework tries for another exploit and repeats till it finds one which satisfies the code execution requirement.

D. Intelligent learner

Performance of the automated models present in the engine can be enhanced by using a learning system. Machine learning techniques are used to carry out the learning task from the data gained from running the models in various web applications. This data can be cleaned and optimized to create cases that would increase the success of the system. Mathematical models can be used to model the training process. Patterns present in applications are recorded in the model to make finding vulnerable points easier.

V. PROPOSED WORK

XSS is most ordinarily associated with execution of malicious javascript through web applications. This vulnerability is also utilized as a platform for launching other types of attacks. The other types of attacks include Session Hijacking, Denial of Service(DoS), Defacement, Account Hijacking and Cross Site Request Forgery(CSRF). The proposed work will identify the vulnerabilities on the web page using following proposed algorithm.

Algorithm 2: Calculating vulnerability and EO

Step 1: Crawl through the website Wi (for i = 1,2,3...,n) by following hyperlinks WiHj (for j = 1,2,3...,n) in the pages

Step 2: Generate a list Swb by adding each element of the webpage Swbk (for k = 1,2,3...,n) in a set data type after performing a filter operation

Step 3: Perform two or more HTTP requests with both GET and POST as methods and specified request body for every element in the set Swb

a) Request body with only required parameters

b) Request body with a random payload rp selected from a list of payloads P

Step 4:Record the responses Rl (for l = 1,2,3...,n) in a data directory with the feature k (for 1,2,3...,n), hyperlink (for j = 1,2,3...,n) and website (for i = 1,2,3...,n) specified as the filename Fijkl

Step 5:For each file in the directory, calculate variation by finding the difference of responses and Vulnerability Score VS as numerical measures and insert into a database

a) Variation is found by performing a search operation to find the expected response due to the execution of the payload on the page

b) Vulnerability Score is calculated by performing a linear search on a set of key value pairs with payload and criticality score

Step 6:Calculate Expected Outcome (EO) of the website is calculated by

EO = WSL * UIP * CoS / LUY

Table 1.The terms used in the calculation of Expected Outcome (EO)

Term	Explanation
EO	Expected Outcome
WN	Website Number
AV	Attack Vectors
WSL	Web Server Language
UIP	User Input Possibilities
CoS	Complexity of Score
LUY	Last Updated Year
Swb	Sensitive Web Features
VS	Vulnerability Score

Websites are selected by taking random samples from a larger sample of websites. Each random sample has the least amount of variation among them but the variation between each sample can be higher. Each sample is assigned a score which gives the Complexity of Score (CoS) of the sample. The vulnerability Score increases with data responses which confirm the presence of a cross-site scripting vulnerability. Last Updated Year is the number of years from the year 2000 which can be determined by reading the footer of the page or using an online website information tool.

Table 2.Metrics data representation for a random sample

WN(i)	AV (ii)	VS (iii)	Scope (iv)	EO (v)
1	3,4,2	23.34	Large	3.4
2	4, 2, 9	0	Small	1.5
3	5, 3, 2	78.34	Medium	6.5
4	2, 5, 7	89.12	Medium	8.1

Payload ID is specified in the attack vector. We design metrics to c. The reason for including the web server language in the expected outcome metric is the presence of comparatively large number of vulnerabilities in out-of-date languages. Here the word out-of-date is used to mention languages which are being used for a longer time, not languages which are not updated. From the work it has been inferred that websites running PHP [18] have the large amount of vulnerabilities present in them due to the absence of inbuilt models which escape characters. Languages such as Python and Ruby have template engines as models in popular web frameworks such as Flask and Django. For example, Flask implements the jinja2 template engine which provides out of the box character escaping as follows:

```
defcalculate_uip(website):
uip = 0
elements = crawl(website)
for el in elements:
if el == 'input': uip += 1
defexpected_outcome(wsl, uip, cos, luy):
wsl_score = wsl_scores[wsl]
return (wsl * uip * cos)/luy
```

The proposed system was conducted based on different experiments. The expected outcome metrics were based on the web server language, input, site, updated year and sensitive web features.

VI. EXPERIMENTAL RESULT AND DISCUSSION

The proposed technique is evaluated using Python. Python is more powerful than any other languages. It is very simple and can be used in any environment. We tested for different web applications and found it is more successful. It can scan, filter and sanitize [19] XSS attack. The escape user input can prevent XSS attack. The results obtained from Metrics data representation for a random sample is summarized in Table-2.We have shown the relationship between Vulnerability score and Scope in Figure- 4 and we have shown the relationship between Vulnerability score and Expected Outcome.

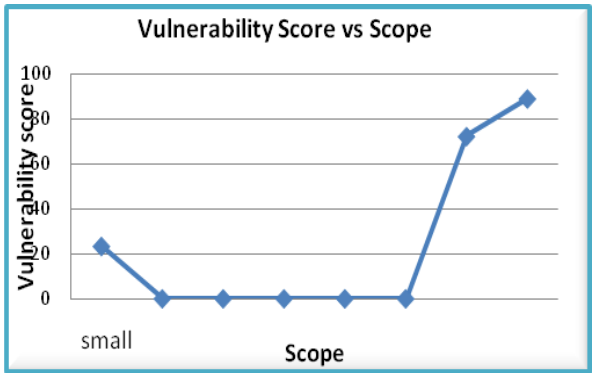

Figure 4. Vulnerability score and Scope

Figure 5. Vulnerability Score and Expected Outcome

VII. CONCLUSION

The requests are send after fixing the bugs in the application. To protect web applications against un-trusted input or which accepts any arbitrary data, checks it against known instances of possible attacks and then forwards it to the suitable web resource. It requires a lot of work in the server-side which can't be successfully implemented for a simple web application.

Other methods such as encoding, escaping and validating are easier to implement and maintain in web applications. Allow the user to select default and often used data: This restricts the ability of an attacker to input data
and allows better control over web applications. Data type check can strictly restrict the type of data which can be sent into a web application. This is done by comparing the language level data type and encoding characters such as html special chars. In this paper we have concluded how XSS attacks can be exploited and prevented by these models. Finally the outcome is when the vulnerability score is high, the scope is medium and the expected outcome is high.

REFERENCES

1. Isatou Hydara and et. al. “Current state of research on cross-site scripting (XSS) – A systematic literature review”, ELSEVIER Information and Software Technology, pp. 170–186, 2015.
2. Amit Singh and S Sathappan “A Survey on XSS webattack and Defense Mechanisms”, IJARCSSE, Vol 3 issue 4, March 2014.
3. Chavan, S.B and Meshram,B.B., Classification of Web Application Vulnerabilities, International Journal of Engineering Science and Innovative Technology (IJESIT),Volume 2, issue 2, 2013.
4. Garg, A. and Singh, S., A Review on Web application Security Vulnerabilities, International Journal oF Advance Research in Computer Science and Software Engineering, Volume 3, Issue1, 2013.
5. Singh, A. and Sthappan,S. , A survey on XSS web-attack and Defence Mechanisms, International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE), Volume 4, Issue 3, ISSN:2277 128X, 2014.
6. DafyddStuttard and Marcus Pinto,The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws.Second edition. pp-443, 2011.
7. Ahmed Elhady Mohamed. Complete Cross-site Scripting Walkthrough. pp-1.
8. DafyddStuttard and Marcus Pinto. 2011. The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. Third edition. pp- 6.
9. Philipp Vogt, Florian Nentwich , Nenad Jovanovic , Engin Kirda , Christopher Kruegel , and Giovanni Vigna, Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis.pp-3, 2007.
10. Engin Kirda , Christopher Kruegel , Giovanni Vigna , Nenad Jovanovic, Client-side cross-site scripting protection, JournalComputers and Security. pp-592-604, 2009.
11. Joaquin Garcia-Alfaro Guillermo Navarro-Arribas, Prevention of Cross-Site Scripting Attacks on Current Web Applications.pp 1770-1784, 2007.
12. Shailendra Rathore, Pradip Kumar Sharma and Jong Hyuk Park, XSSClassifier: An Efficient XSS Attack Detection Approach Based on Machine Learning Classifier on SNSs J Inf Process Syst, Vol.13, No.4, pp.1014-1028, 2017.
13. Mahmoud Mohammadi, Bill Chu, Heather Richter Lipford. Detecting Cross-Site Scripting Vulnerabilities through Automated Unit Testing.pp.1.
14. Cross-Site-Scripting Attacks and Their Prevention during Development, Ms. Daljit Kaur1 , Dr. Parminder Kaur. 2017. pp-154.
15. Symantec Internet Security Threat Report , 2012, pp-32.
16. Shashank Gupta, Shashank Gupta, Exploitation of Cross-Site Scripting (XSS) Vulnerability on Real World Web Applications and its Defense,pp. 30, 2016.
17. Jakob Kallin and Irene Lobo Valbuena.Excess XSS by Attacks: Cross Site Scripting Exploits and Defense Paperback,pp.10, 2007.
18. Jeremiah Grossman Robert “RSnake” Hansen Petko “pdp” D. Petkov Anton Rager Seth Fogie, XSS Attacks scripting exploits and defense. pp. 398, 2007.
19. E Kritzinger SH von Solms Cyber Security for home users: A New Way of Protection through Awareness Enforcement, pp-2, 2018.
20. Shashank Gupta. 12017. Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art, pp 512–530.
21. Abdalla Wasef Marashdih and Zarul Fitri Zaaba. Cross Site Scripting: Detection Approaches in Web Application. 2016, pp.159.
22. Leena Jacob,Madhumita Chatterjeer, Virginia Mary Nadar. Detection Model For XSS Attack, pp.66, 2016.
23. Andrea Hauser, DOM based Cross-site Scripting Client-side attacks on browsers. 2017.pp-2.
24. Rahul Johari, Pankaj Sharma, "A Survey On Web Application Vulnerabilities (SQLIA,XSS)Exploitationand Security Engine for SQL Injection", InternationalConference on Communication Systems andNetwork Technologies, pp. 453-458, 2012.
25. Satou Hydara, Abu Bakar Md. Sultan, Hazura Zulzalil, Novia Admodisastro, Current state ofresearch on cross-site scripting (XSS) – A systematic literature review, Information and Software Technology, Vol. 58, pp.170–186, 2015.