VOL. 10, NO. 10, JUNE 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

A DYNAMIC TOOL FOR DETECTION OF XSS ATTACKS IN A REAL-TIME ENVIRONMENT

K. G. Maheswari[1] and R. Anita[2]
[1]Department of MCA, Institute of Road and Transport Technology, Anna University, Erode, Tamil Nadu, India
[2]Department of Electrical and Electronics Engineering, Institute of Road and Transport Technology, Anna University, Erode, Tamil Nadu, India
E-Mail: kgmaheswari@gmail.com

ABSTRACT

With the wide spread application of internet, the web application have become a focal target for the attackers. The cross site scripting attack popularly known as XSS attack takes advantage of the web browser rather than the application itself. This poses serious threat to the developers who got to ensure the security of the web services. Web intrusion detection systems are security programs that help the developers as well as the customers to evaluate whether events and activities occurring in a Web Application are legitimate. The objective of Web IDS is to identify intrusions with high false alarms and low detection rate while consuming minor properties. The proposed work presents an intrusion detection system that analyzes web requests looking for evidence of malicious behaviour and provides a sophisticated query analysis. The injection of vulnerabilities and attacks is indeed an effective way to evaluate security mechanisms and to point out not only their weaknesses but also ways for their improvement. This work tried a solution for the vulnerable web attacks through real time tools such as DVWA and XSS Me.

Keywords: cross-site scripting attack, XSS Me, web IDS, filter, attacker.

1. INTRODUCTION

The well-established popularity of internet has paved way for the glory of web applications in every field especially banking, social networking, health services, finance etc. Woefully the vulnerabilities in the software are evolving as a serious threat to the web applications. The recent statistical reports [8] state that 63 percent of websites accessed poses an average of 6 unsolved defects. The Open Web Application Security Project (OWASP) [6] and Common Vulnerabilities and Exposures (CWE) [7] in the year 2013 projected that XSS attack or the cross-site scripting attack as the most serious threat to web applications. The cross site scripting attack [1] involves the injection of malicious and invalidated client-side scripts into the web applications therefore imposing serious threat to the user. The XSS attacks can be easily executed but they may be difficult to be detected and prevented. This can be explained with the help of the following scenario.

A user may like the comment of another user in a web application. If the victims likes the comment in which the attacker has injected malicious JavaScript, then the malicious scripts is executed in the victim's browser whenever the victim visits the comments. The victim is unaware of the malicious script being executed in his/her browser.

The JavaScript is widely employed to enrich the display of web pages from the client side. JavaScript language [11] was initially introduced by Netscape to enhance client-side utilities. It was loaded with the capabilities of the object oriented programming language. It was later or standardized by ECMA. The java script can be easily downloaded into the browser both knowingly and unknowingly by the user. The sand boxing mechanism [12] enables the secure execution of the JavaScript. The sand box mechanism is designed in such a way as to approve only a predefined set of operations. To make it clear only a limited number of resources can be accessed by the java script within the browser. The faults in the origin policy are taken as advantage by the attackers.

The XSS attacks can be broadly classified into three types as shown in the Figure-2. In the DOM-based attack the vulnerability instead of being a part of the html appears in the DOM that is the Document Object Model. Research shows that more than 50% of the websites present today are vulnerable to the DOM-based attacks [13]. The stored XSS attack [14] is carried out by storing a default malicious JavaScript on the target server database. The reflected XSS attack [15] is also known as the non-persistent XSS attack. It takes place when the browser executable code is injected by the attacker embedded within a single HTTP response. The injected script is never stored in the application as in the case of the stored XSS attack.

The XSS attack scenario can be clearly explained with the help of the following scenario. Suppose that a user logs on a website www.foreverbanking.com to perform online transactions. The cookies in the web-based application from the foreverbanking.com store session information in the browser of the user. The cookie can be accessed only by the JavaScript code downloaded from the foreverbanking.com web server due to the imposition of the origin policy. Yet there are chances that the user may browse some malicious website, say devil.com, and can be lured into clicking the below link:

<a href="http://foreverbanking.com/
<script>
document.location=
'http://devil.com/steal-cookie.php?';
+document.cookie
</script>">
Click here to collect your prize.

When the user clicks the link it sends a HTTP request from the user's web browser to the foreverbanking.com web server requesting the below script

<script>
document. location=
'http://devil.com/steal-cookie.php?';
+document. cookie
</script>

The foreverbanking.com web server obtains the request and verifies whether it has the requested resource. When the resource is unavailable it sends an error message. The error message may include the requested resource name (a script) in the error message. The malicious website gains access to the cookie set by foreverbanking.com web server on the execution of the script.

This scenario can be simply explained in four steps.

1. User logs into the attacker's website.
2. User clicks on a malicious link and an HTTP request containing JavaScript code is sent to the trusted server.
3. The trusted server returns error message containing the name of the resource (a JavaScript code).
4. The JavaScript code is executed and the user's cookie associated with the trusted server is sent to the attacker's server.

The prominent reason behind the XSS attack is the deficiency of the source code. The deficiencies may arise due to the short comings of the programming languages, inefficiency of the developers, and negligence in input validation [2]. The attackers make use of these vulnerabilities and inject malicious scripts there succeed in manipulating web contents, session hijacking and theft of cookies. Hence it is essential for the developers to eliminate these kinds of vulnerabilities during the development stage. Detection of vulnerabilities in the early stage of software development eliminates the risk of failure and also maintenance costs in the later stage of the software lifecycle. The software security focuses on the elimination as well as minimisation of the software vulnerabilities in early stage of software development so that the software does not crash even on the instance of malicious attack.

The software security approaches [3] can be divided into three namely guidelines for secure coding, prevention of real time attacks and detection of vulnerabilities. They are depicted in Figure-1. The guidelines for secure coding approach provide the developers with a set of guidelines for secure development of the software. They are formed based on several standards and libraries. The risk of vulnerabilities shall be reduced if the developers are well trained on the usage of secure libraries and reside on the prescribed guidelines during the development of the software. The prevention of real time attacks approach [4] aims to deploy run time observer on both client and server side in order to monitor and detect the real time attacks [9]. The real time prevention approach interprets the real time data and also validates the incoming and outgoing traffic against vulnerable scripts [5]. Detection of vulnerabilities approach [10] examines the applications right from the early stage of development in order to detect the vulnerabilities in the software.

2. RELATED WORKS

Several works have been proposed and implemented to prevent the XSS attacks both on the server and the client side. The method proposed in [16] aims to implement certain changes in the Firefox browser and thereby intercept the calls to the JavaScript spider-monkey and keeps a record of all the code executed. It employs the use of an intrusion detection system to validate the behaviour of the script being executed with of the existing malicious script. The system only employs the use of certain malicious scripts. It proves to be inefficient when new kinds of attacks are deployed on the system as they do not have the signature for new kind of malicious scripts and attacks. Futher researches [18] were also conducted to ensure the data is sanitized [17] before it is given as output by employing the technique of separation of the malicious data or the untrusted data from the HTML output [19]. In order to give out client-side solutions web proxies were used to alleviate the effects of XSS attacks [12]. The requests are filtered by the proxy server before they are being sent to the original server. The system has to ensure that the user's private data are given protection. It employs the use of black listed URLs. The proxy server allows requests sent only from the white listed URLs. The min drawback behind this approach is that it cannot find solution to control the false positive rates. Another client side proxy approach [21] aims to mitigate the XSS attack by inspecting the special characters and ensures that the same characters occur in the response. The sure may be unaware of the vulnerabilities and fail to install this kind of defence mechanisms. The attackers take advantage of the user's negligence and try to impose vulnerable attacks. Other works were proposed that aim to detect the flaws in security strategies.

3. PROPOSED WORK

Websites rely completely on complex web applications to deliver content to all users according to set preferences and specific needs. In this manner organizations provide better value to their customers and prospects. Dynamic websites suffer from various

vulnerabilities rendering organizations helpless and prone to cross site scripting attacks. Cross Site Scripting attacks are difficult to detect because they are executed as a background process. Cross Site Scripting is the most common web vulnerabilities in existence today which is most exploited issue .This paper presents various approaches used by clients and Server to prevent XSS attacks.

Early detection of XSS vulnerabilities during the development process helps in protecting a web application from insignificant flaws. XSS-Me is an extension of Mozilla Firefox browser and comes under Exploit- Me tool that tests for reflected XSS vulnerabilities. The appearance of XSS-Me tool is portrayed in Figure-5. XSS-Me submits two requests: one request is where exploitation is identified using Firefox's JavaScript engine, known as "error"; another request is where we identify exploitation by matching for the attack vector in the HTML response.

The tool works as shown in Figure-3 by scanning for potential loop holes in the web application. First, it locates HTML forms in the web page being tested. The corresponding values are replaced with strings that represent an XSS attack. If the resulting HTML page sets a particular JavaScript value (document. vulnerable = true) then the given XSS string is marked as susceptible by the tool. The attack strings in "warning" may result in being a successful attack on different browser because they seem to have come from a server completely uuencoded. For example, when we start the XSS-Me tool, all the forms in current web page is listed in a series of tabs including all visible and hidden fields.

By default, the field is set to its current value, or if nothing is stated then the string "Change this to the value you want tested" is exhibited. Only if we check the box next to a field name, it will get tested for XSS one at a time. For example, the tool attempts to manually verify the "txtSearch" field for the parameter inserted as desired JavaScript (e.g.; <SCRIPT>alert ("Test for XSS") </SCRIPT> and to test "Go" field for all the XSS attack strings. The proposed system architecture in this research work is shown in the Figure-4. It consists of eight major components namely User Interface, Data Collection Manager, Decision Manager, Attack Analyzer Module, and Temporal Rule Manager.

3.1 User interface

The user interface component of the proposed system is responsible for accepting data requests and receiving responses from the user through web applications and registration module. It also accepts queries in the form of keywords and returns result in the form of rules. It also provides special menus and forms through which query processing can be performed.

In this module a user registers itself for a specific web application by storing personal information like name, date of birth, password, email address, phone number and other details relevant to application in the server database. Before storing the information to database the pre-processing is done.

3.2 Preprocessing module

In the Pre-processing module, optimal subsets of attributes that influence a particular intrusion are obtained as the important features. This module uses a pattern matching approach for effective feature selection. This module helps to select thousands of records from the huge dataset and converts them into relational database table format and then performs data cleaning. Records of incorrect and missing data are also deleted from the dataset.

The decision manager monitors the overall process of this proposed system. It acts as an intermediate between modules. One of the major responsibilities of the decision manager is its functionality in providing an expert decision support by interacting with pre-processing model and inference system which is available in rule manager module. It takes the decisions based upon the information provided from the rule manager. It instructs the new facts and rules in to the rule manager at the time of expert decision making process.

3.3 Temporal rule manager

The rule manager consists of query processor. It is responsible for firing the rules which are provided by the domain expert. The rules are prioritized and scheduled by the rule manager for execution. The expert advice is used for performing knowledge acquisition from domain experts. For this purpose, the knowledge engineer develops a questionnaire and posts and queries and problems related to the intrusion domain. The domain expert provides solutions and suggestions in the form of natural language sentences and rules. Moreover, they provide expert advice for analyzing the results obtained by the prediction.

3.4 Attack analyzer module

Fourth module is, attack analyzer module which analyzes the attack based on the response from the decision manager by comparing with corresponding data stored in database. If data does not match then that login attempt is analyzed as attack, input validation library is updated and control goes to invalid tag and knowledge base.

The knowledge base contains rules for making decisions on heart disease prediction. This rule base is created by storing rules obtained from domain experts and also using the rules obtained from the training performed on the data set. The rules are extracted based upon the temporal relations. The rules from the rule manager can be used to process the queries of individual patient record. The queries specify the severity of the disease in each patient. These rules are stored the form IF….THEN statements which are used in the inference process for effective decision making.

4. RESULT AND CONCLUSIONS

Cross Site Scripting is the most common web vulnerabilities in existence today which is most exploited issue .This paper presents various approaches used by clients and Server to prevent XSS attacks.

Early detection of XSS vulnerabilities during the development process helps in protecting a web application from insignificant flaws. XSS-Me is an extension of Mozilla Firefox browser and comes under Exploit- Me tool that tests for reflected XSS vulnerabilities. XSS-Me submits two requests: one request is where exploitation is identified using Firefox's JavaScript engine, known as "error"; another request is where we identify exploitation by matching for the attack vector in the HTML response.

The tool works by scanning for potential loop holes in the web application. First, it locates HTML forms in the web page being tested. The corresponding values are replaced with strings that represent an XSS attack. If the resulting HTML page sets a particular JavaScript value (document. vulnerable = true) then the given XSS string is marked as susceptible by the tool. The attack strings in "warning" may result in being a successful attack on different browser because they seem to have come from a server completely unencoded. For example, when we start the XSS-Me tool, all the forms in current web page is listed in a series of tabs including all visible and hidden fields. The result evaluation of a few web pages is described in Figure-6.

Mostly browser defences are looking for the scripts and only intercepting JavaScript calls to the engine and ignoring the other possible kind of vectors that does not involve JavaScript which otherwise can execute sophisticated attacks (e.g.; injection into HTML header can cause credential stealing, cross-site request forgery). Further, defences are unable to identify the malicious scripts which are spread over more than one parameter (multiple point of injection) as looking for the source code in the combination of different inputs is not practically feasible. The Figure-7 gives a summary of number of tests passed and failed by few web pages in the form of a pie chart. It also gives the number of warnings and the total number of tests executed.

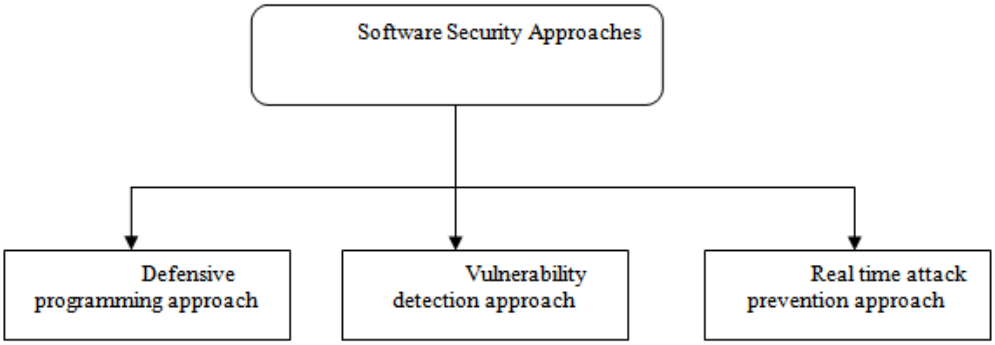

Figure-1. Types of software security approaches.

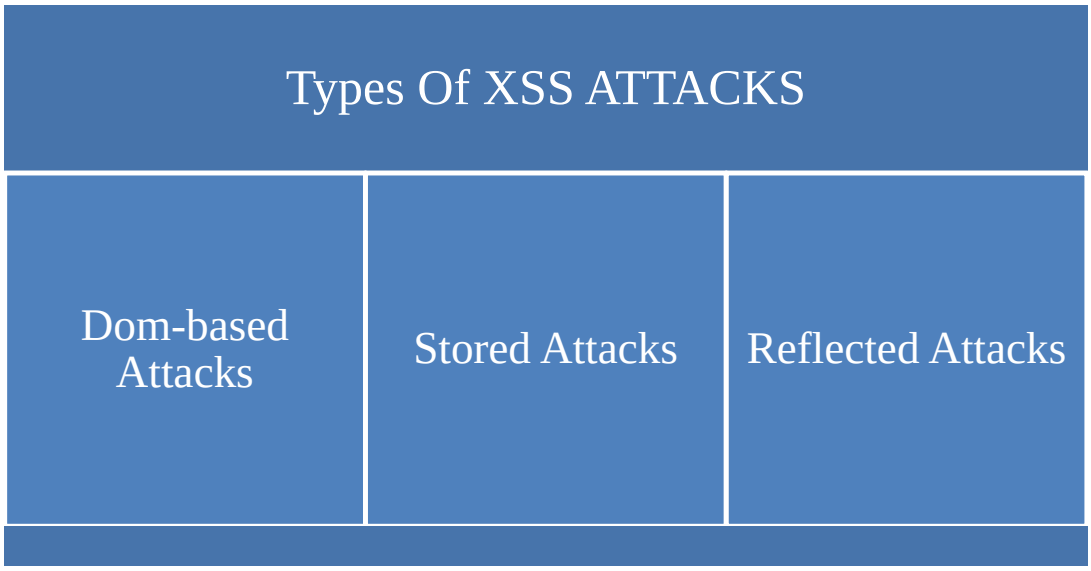

Figure-2. Types of XSS attacks.

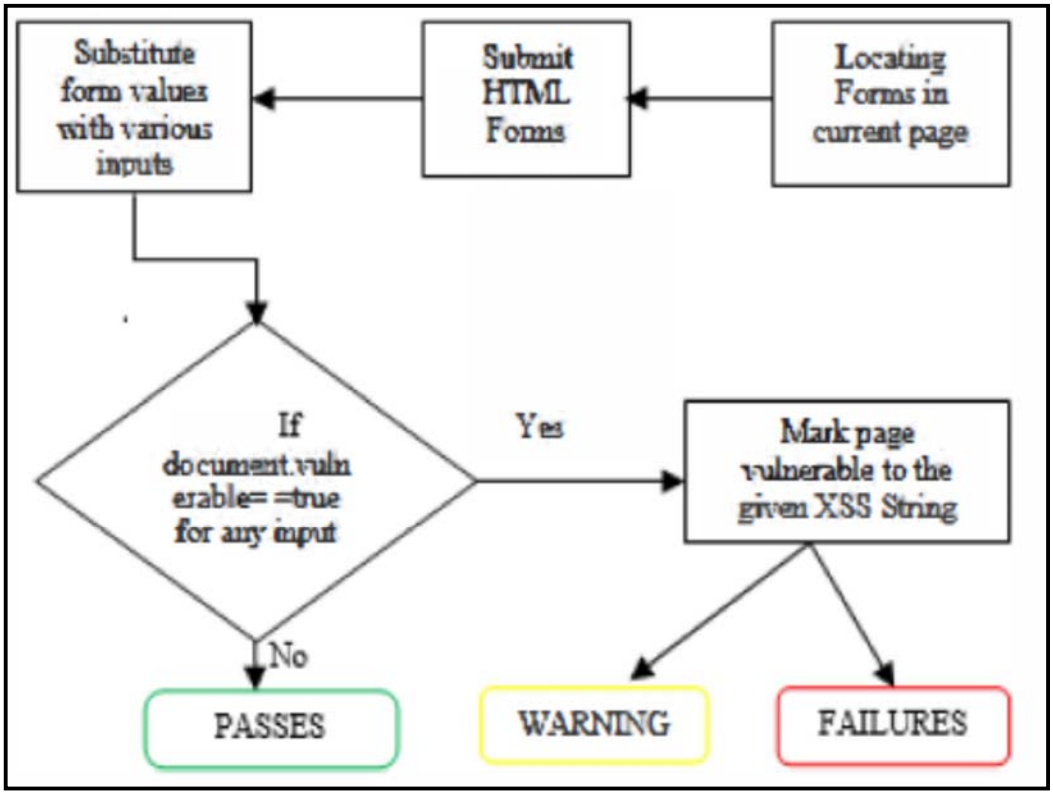

Figure-3. Working of XSS-Me tool.

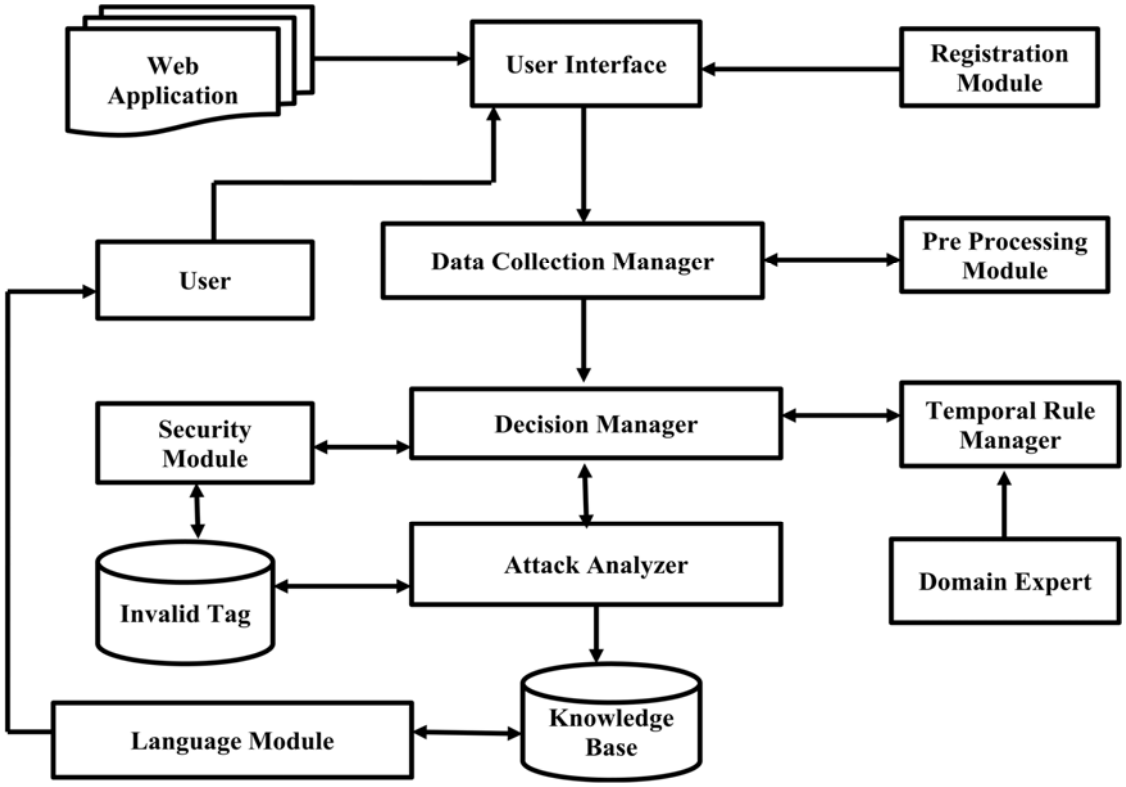

Figure-4. Proposed system architecture.

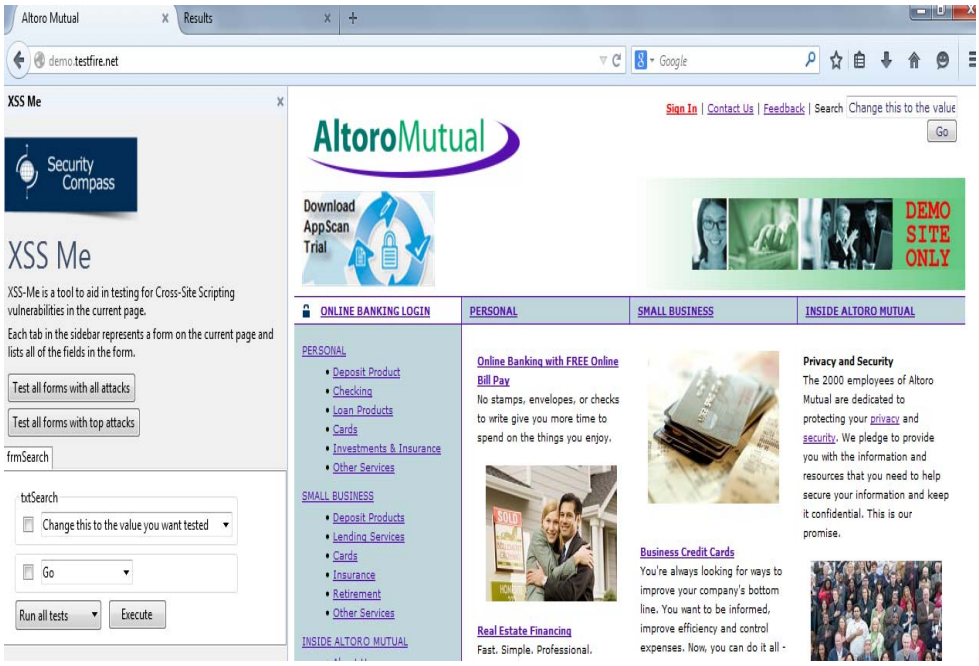

Figure-5. Appearance of XSS Me.

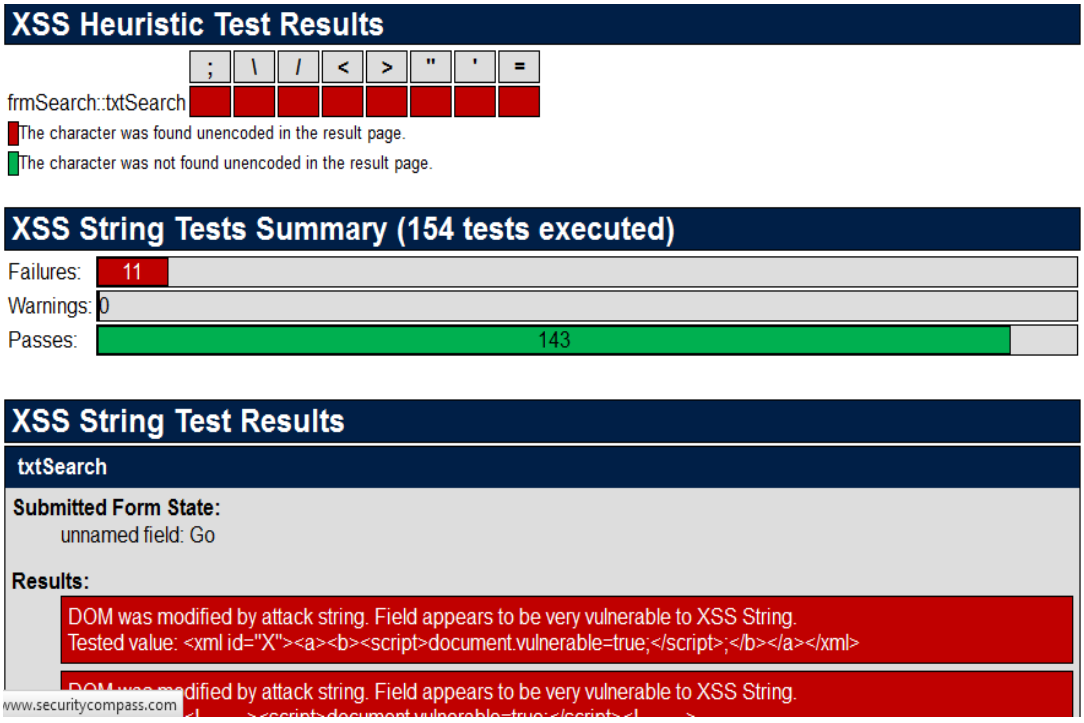

Figure-6. XSS-Me result evaluation.

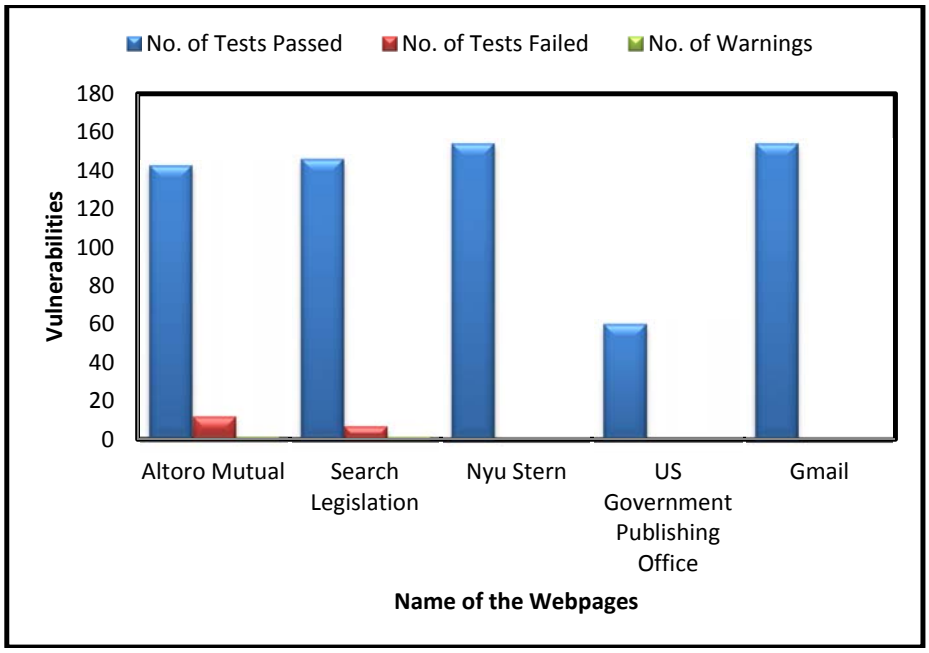

Figure-7. Pie chart representation of result evaluation.

REFERENCES

[1] Adam Kie˙zun, Philip J. Guo, Karthick Jayaraman, Michael D. Ernst, "Automatic Creation of SQL Injection and Cross-Site Scripting Attacks, Published in IEEE Conference , Vol.11, 2011.

[2] Bhawna Mewara, Sheetal Bairwa, Jyoti Gajrani and Vinesh Jain, "Enhanced Browser Defense for Reflected Cross-Site Scripting", IEEE, Vol. 978 , pp. 4799-6896, 2014.

[3] N. Jovanovic, C. Kruegel, and E. Kirda, "Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities," Proc. of the IEEE Symposium on Security and Privacy, May 2006.

[4] D. Bates, A. Barth, C. Jackson, Regular Expressions Considered Harmful in Client-side XSS Filters, In: Proc. 19th Int. Conf. World Wide Web - WWW, 2010.

[5] H. Shahriar, M. Zulkernine, "Injecting Comments to Detect JavaScript Code Injection Attacks", IEEE 35th Annual Computer Software and Applications Conference Workshops, 104-109, 2011.

[6] Common Vulnerabilities and Exposures (The Standard for Information Security Vulnerability Names) http://cwe.mitre.org/.

[7] https://www.owasp.org/index.php/Top_10.

[8] www.whitehatsec.com/home/resource/stats.html.

[9] P. Sharma, R. Johari, S.S. Sarma, "Integrated approach to prevent SQL injection attack and reflected cross site scripting_attack", pp. 343-351 , 2012.

[10] N. Jovanovic, C. Kruegel, E. Kirda, " Precise Alias analysis for Static Detection of Web Application Vulnerabilities", Proceedings of the 2006 workshop on Programming_languages and analysis for security, pp. 27-36, 2006.

[11] D. Flanagan, "Java Script: The Definitive Guide", December 2001, 4th edition.

[12] Engin Kirda, Christopher Kruegel, Giovanni Vigna, and Nenad Jovanovic, "Noxes: A Client-Side Solution for Mitigating Cross-Site Scripting Attacks", published in SAC'06 ACM Vol. 108, No. 2, 2006.

[13] Dwen-Ren Tsai, Hsuan-Chang Chen, Peichi Liu, Allen Y. Chang, "Optimum Tuning of Defense Settings for Common Attacks on the Web Applications", published in IEEE conference, Vol.978, No. 1, pp. 4244-4170, 2009 IEEE.

[14] Guowei Dong, Yan Zhang, Xin Wang, Peng Wang, Liang Kun Liu, "Detecting Cross Site Scripting Vulnerabilities Introduced by HTML5", 11th International Joint Conference on Computer Science

and Software Engineering, Vol. 978, No. 1, pp. 4799-5822, 2014.

[15] S. Cook. "A web developer's guide to cross-site scripting", Technical report, SANS Institute, 2003.

[16] O. Hallaraker and G. Vigna, "Detecting Malicious JavaScript Code in Mozilla," In: Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), Shanghai, China. pp. 85-94, June 2005.

[17] Chin. E, Wagner.D, "Efficient character-level taint tracking for java" In: Proceedings of the 2009 ACM Workshop on Secure Web Service, pp. 3-12.

[18] P. Vogt, F.Nentwich, N.Jovanovic, E.Kirda, C.Krugel, and G.Vigna "Cross site scripting prevention with dynamic data tainting and static analysis", In NDSS, 2007.

[19] V. Haldar, D.Chandra, and M.Franz, "Dynamic Taint Propagation for Java", In: Twenty-First annual Computer Security Applications Conference (ACSAC), 2005.

[20] Imran Yusof, Al-Sakib Khan Pathan, "Preventing Persistent Cross-Site Scripting (XSS) Attack By Applying Pattern Filtering Approach", supported by Networking and Distributed Computing Laboratory, 2014.

[21] S. YAMAGUCHI, O. ISMAlL, M. ETOH and Y. KADOBA Y ASHI, "A proposal and implementation of automatic detection/collection system for cross-site scripting vulnerability," In: Proceedings of the International Conference on Advanced Information Networking and Application (AlNA04), 2010.