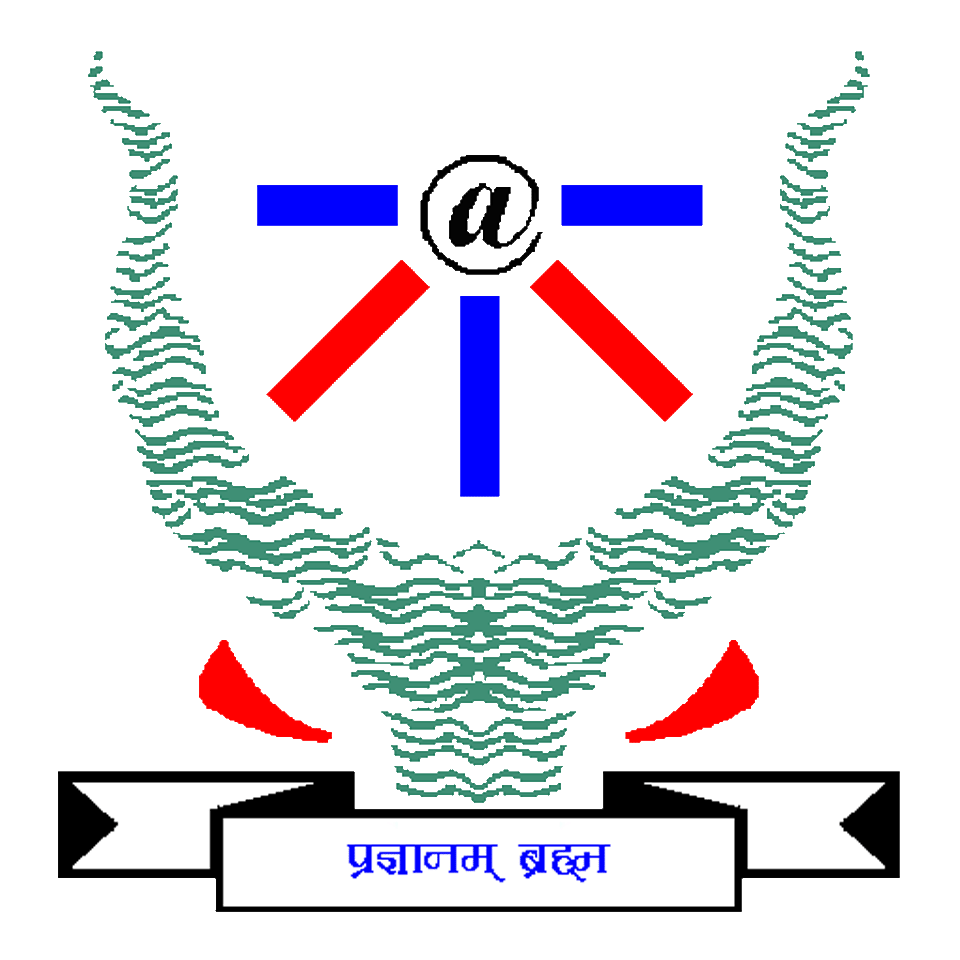
**Project Report**

**Cross Site Scripting Detection**

**( Using Penetration Testing )**

Under Guidance of

Dr. S.Venkateshan



Submitted By:-

Akash Yadav (iit2013207)

Dikshant Devkota (iit2013002)

Sachin Pokharel (iit2013009)

Department of Information Technology.

Indian Institute of Information Technology - Allahabad

**Acknowledgement**

We would like to express our sincere gratitude to “Dr. S. Venkatesan” without whose guidance the on-going proceedings of the project would have been a mammoth task. Throughout this semester the efforts we made was solely because of the guidance our mentor provided.

**Dr. S Venkatesan**

**IT Department. IIITA**

**Candidate Declaration**

We hereby declare that the work presented in this project entitled “Cross-Site-Scripting Detection using penetration testing”.

“Cross-Site-Scripting Detection using penetration testing” submitted in partial fulfilment of the 8th semester of Bachelor of Technology (B.Tech) program, in Information Technology at Indian Institute of Information Technology, Allahabad, is an authentic record of our original work carried out under the guidance of Dr. “S. Venkatesan” due acknowledgements have been made in the text of the project to all other material used. This work was done in full compliance with the requirements and constraints of the prescribed curriculum.

The Bonafide Work Of -

Dikshant Devkota (IIT2013002)

Sachin Pokharel (IIT2013009)

Akash Yadav (IIT2013207)

**Place: Allahabad**

**Date: 28/04/2017**

**Abstract**

Web applications have become the dominant method for implementing and providing access to on­line services. As the use of web applications increases, a greater focus on web security is needed. In recent years, there has been a significant increase in the number of web­based attacks. Cross­site scripting (mostly called as XSS) is one of the most dangerous and most common website vulnerability on the internet. The attack is based on the possibility to insert malicious JavaScript code into web pages . Since Web browsers support the execution of commands embedded in web pages to enable dynamic web pages attackers can make use of this feature to enforce the execution of malicious code in a user’s Web browser.

We have implemented XSS detection technique using Penetration Testing in which the test can be done blind, without access to the source code, as an attacker would do (black box penetration test), or with more information about the architecture or source code (**white box penetration test**)

**Contents**

Abstract.............................................................................................................................4

1. Introduction.............................................................................................................6

* How Cross-Site Scripting works………………………………………...7
* Reasons Why XSS Vulnerabilities are Exploited……………………….8

2. Literature Survey…..............................................................................................9

3. Problem Statement & Objectives........................................................................10

4. Methodology….........................................................................................................11

* White Box Testing……………………………………………………….11
* Black Box Testing………………………………………………………..12
* Data Flow Diagram...………………………………………………….....13

5. Hardware & Software requirements..................................................................14

6. Activity Time Chart.......................................................................................15

.

7. Results and Discussion………...........................................................................16

8. Conclusion & Future Scope................................................................................19

.

9. References.................................................................................................................20

.

**Introduction**

Cross-site scripting is a type of computer security vulnerability that is found in web-based applications which allows code injection by web users into any webpage that is viewed by other users. XSS is made possible due to the fact that faulty coding causes XSS holes (vulnerabilities on websites that allows attackers to avoid security measures) in the client-side script that allows for insertion of malicious code. During an attack, “everything looks fine” to the end user, but in actuality they are subject to a wide variety of threats. XSS is a potentially dangerous vulnerability that is easy to execute and very long and arduous to repair. XSS holes exist in 7 out of every 10 websites. Many site owners do not consider an XSS hole to be a big threat, which is a commonly made mistake because the consequences of an XSS attack against web applications and its users have been proven to be extremely serious. The most frequent kinds of web applications that are victimized by XSS attacks are search engines, discussion boards, web-based emails, and posts. Even the most well-known websites in today’s world like Google, Yahoo!, MySpace, Facebook, PayPal, and WikiPedia were once victims.

The most commonly used programming languages during XSS attacks are HTML, XHTML, JavaScript, and Adobe’s Flash. However the most popular and potentially the most detrimental language used by malicious attackers is JavaScript.

**How Cross-Site Scripting works:**

A website is vulnerable if it accepts and subsequently return the same input back to a user. The most common example is when a user does a search and the Web server returns the same data the user typed in. As an example, a user does a search for “XSS” and the browser returns a message of, “Your search for XSS returned the following.”

A cross-site scripting attack can be done rather easily to a Web server that is not properly protected. Web servers generate both text and HTML markup on their web pages. The client’s browser then interprets the web pages. HTML uses special characters to distinguish text from markup. Different characters are special at different points in the document, depending on the grammar. The less-than sign “<” usually indicates the beginning of an HTML tag. An HTML tag can affect the formatting of the page or introduce a program that will be executed by the browser. If the Web server creates pages by inserting dynamic data into a template, it should be checked to ensure that the data to be inserted does not contain any special characters. The user’s Web browser could mistake any special characters as HTML markup. This would result in the browser mistaking some data values as HTML tags or script instead of displaying them as text. An attacker can choose the data that the Web server inserts into the web page, thereby tricking the user’s browser into running a malicious script or program. The program will run in the browser’s security context. The attacker can use this to run the program in an inappropriate security context.

**Reasons Why XSS Vulnerabilities are Exploited :**

* Account Hijacking for identity theft
* Cookie theft/poisoning to acquire sensitive information
* Conduct phishing attacks
* Gain free access to otherwise paid for content
* False advertising.
* Spread web worms.
* Access browser history and clipboard contents.
* Control the browser remotely.
* Scan and exploit intranet appliances and applications.

**Literature Survey**

**Cross Site Scripting-Latest developments and solutions: A survey**

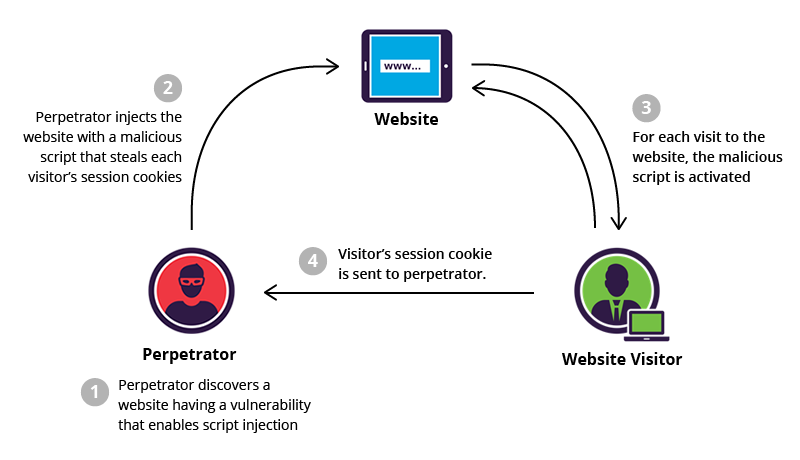
By: Jayamsakthi Shanmugam, Dr. M. Ponnavaikko

Research Student, BITS, Pilani, India

Research reports indicate that more than 80% of the web applications are vulnerable to XSS threats. User friendly web applications are developed to increase the customer base and hackers utilize the features provided by the web applications. Research report shows that there is a shift in the focus of the cyber criminals and cyber spies to evade the counter measures built within the web applications. The authors have collected around 2800 vulnerable Cross Site Scripting (XSS) web applications which formed the basis for drawing conclusions along with the other researchers report on this problem. Recent trend in the growth of XSS attacks indicate that worms are planted in the web application using XSS mechanisms. This paper surveys such vulnerabilities with the current solutions. Categories of solutions are based on the location (client side or server side), analysis type (static, dynamic, taint, alias, data flow, source code, control flow graph), technique (crawling, reverse engineering, black box testing, proxy server) and intrusion detection type (anomaly, misuse, automatic, multimodal). The strengths and weaknesses of all approaches are discussed. In this article, the authors propose the future line of research based on the gaps in the existing solutions proposed by earlier research work.

**Problem Statement & Objectives**

Often attackers will inject JavaScript, VBScript, ActiveX, HTML, or Flash into a vulnerable application to fool a user (Read below for further details) in order to gather data from them. Everything from account hijacking, changing of user settings, cookie theft/poisoning, or false advertising is possible. New malicious uses are being found every day for XSS attacks.



**Objectives : -**

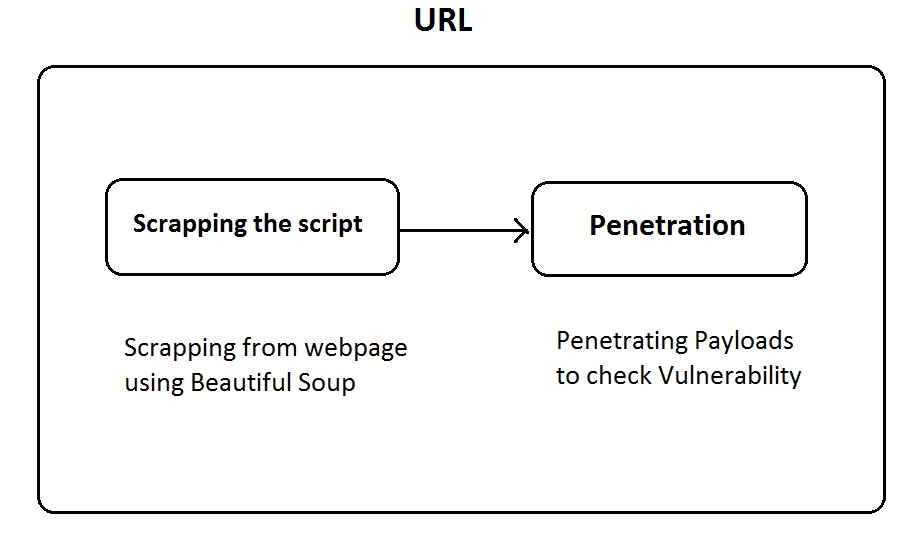
Existing web browsers cannot be trusted to make script identification decisions in untrusted HTML due to their unreliable parsing behavior. So the objective is to parse out the field and areas present in the site to check whether they are vulnerable to external script or not, those vulnerable parts are generally called XSS hole.

To detect XSS hole we inject different Payloads to the different user input required fields in the site. If the payload script respond then the particular site is vulnerable to XSS

**Methodology**

**White Box Testing :-**

As name suggest white box Penetration Testing we needed pre known data of the website inorder to check the vulnerability. So to fetch the fields and user input required area, we did scrapping using beautiful soup ( Python Library). Then added different payloads to the extracted forms and checked by seeing the output whether the given script is responding or not. If the scripts or payloads reacted through a given field or form then that particular form in that particular webpage is vulnerable to XSS attack.

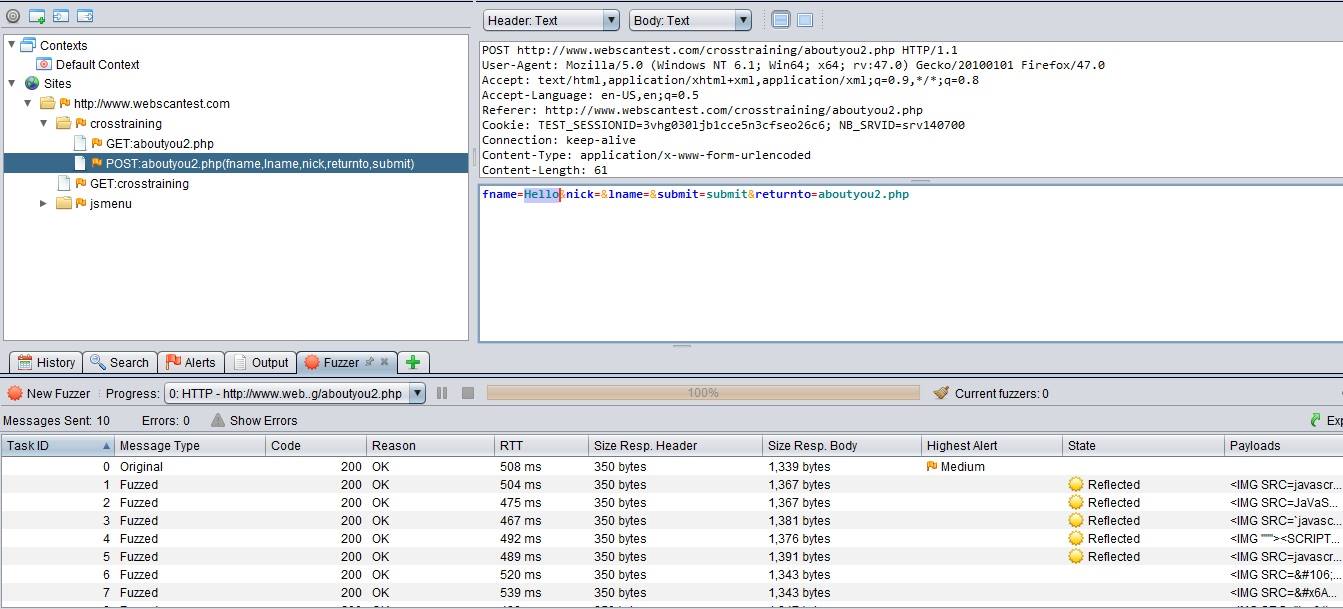


White Box Penetration testing can be done when no webpage data is provided or given. So as to penetrate different payloads for testing the webpage & finding the vulnerable fields within we need to do Scrapping. It is not that all a field respond to or react to all the payloads given. If a Proper Source code is provided by the owner of the webpage then we can do Black Box Testing (details next page)

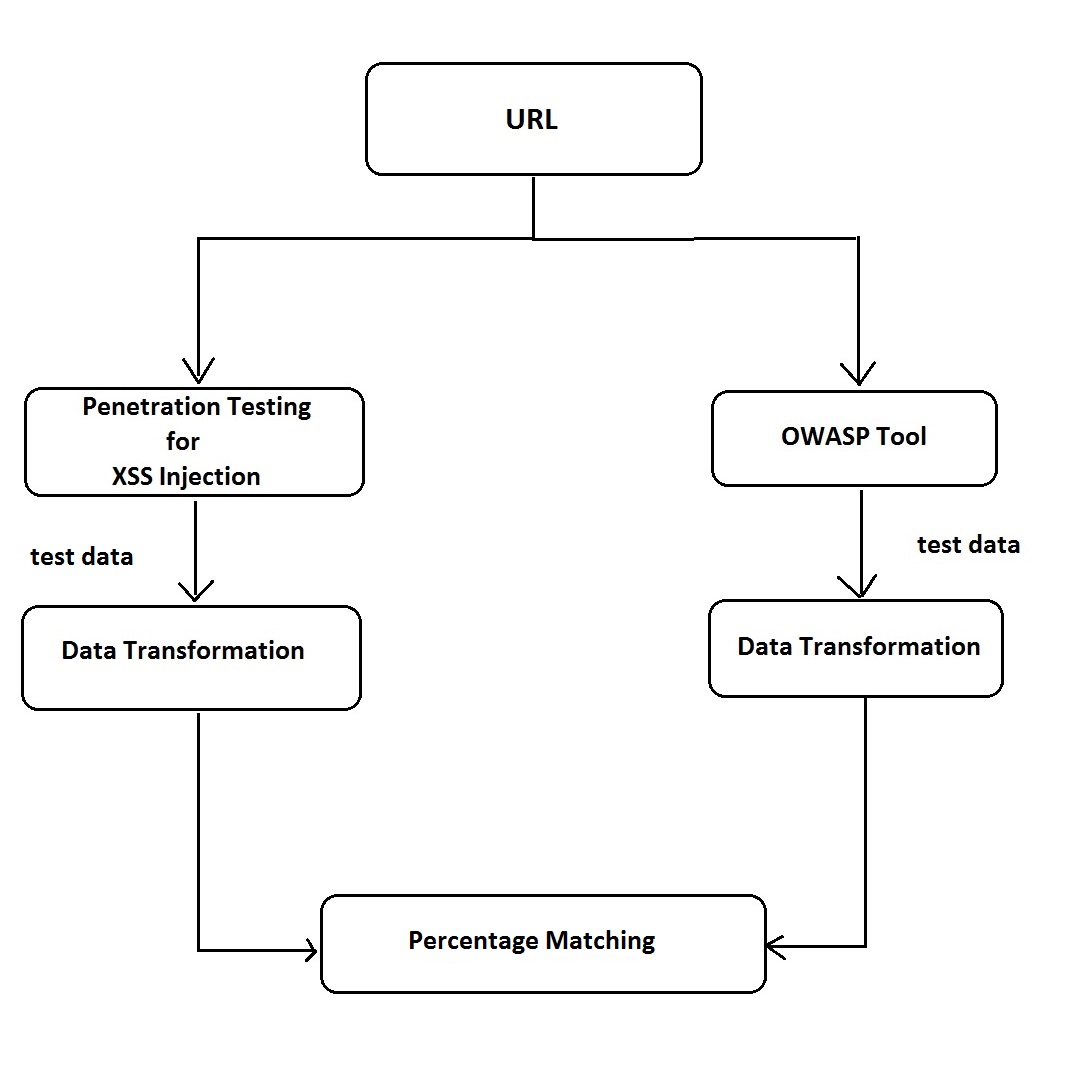
**Black Box Testing :-**

Testing in which the test can be done blind, without access to the source code, as an attacker would do (black box penetration testing). Testing without knowing the source code is a bit difficult & time consuming task, since each and every user input fields & forms needs to be checked manually for each and every domain and sub domain. we used the tool like OWASP for Black Box Testing by penetrating and fuzzing the given webpage.

Screenshots of the Tool Used (OSWAP) :-



**Data Flow Diagram**



**Hardware & Software required**

**Software Requirement:-**

* Anaconda Navigator
* MS - excel
* OWASP tool
* Python library :-

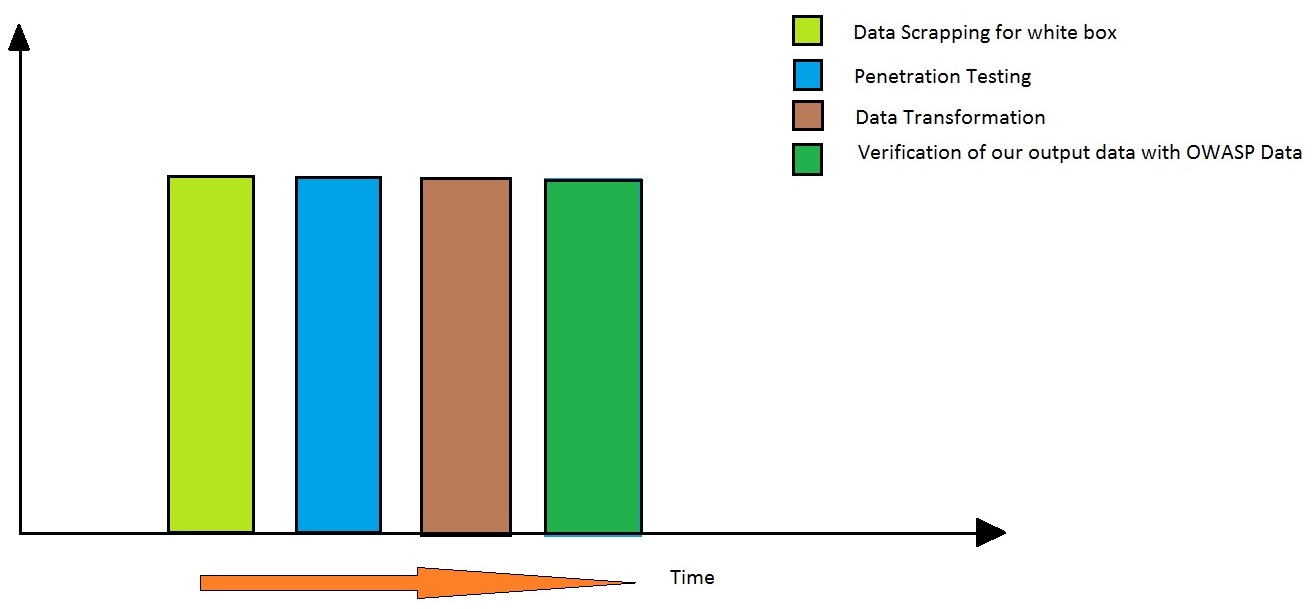
1. Urllib2csv
2. Mechanize
3. Beautiful Soup
4. Panda

* System Running OS – Win XP/7/8/10

**Hardware Requirements**:-

* Running Win OS higher than win XP
* 128 MB & higher RAM
* Internet connection
* 128 GB HDD
* Processor Core 2 Duo & above

**Activity Time Chart**



Activity Time Chart

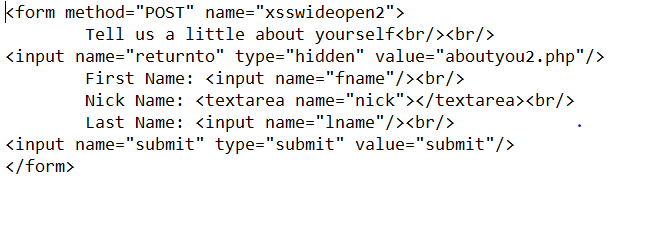
**Results and Discussion**

Scrapping out the script of URL :- [www.webscantest.com/crosstraining/aboutyou2.php](http://www.webscantest.com/crosstraining/aboutyou2.php) and forms in the URL as per the requirement pre data for white box testing

* **Web\_script.txt**



* **Form\_script.txt**

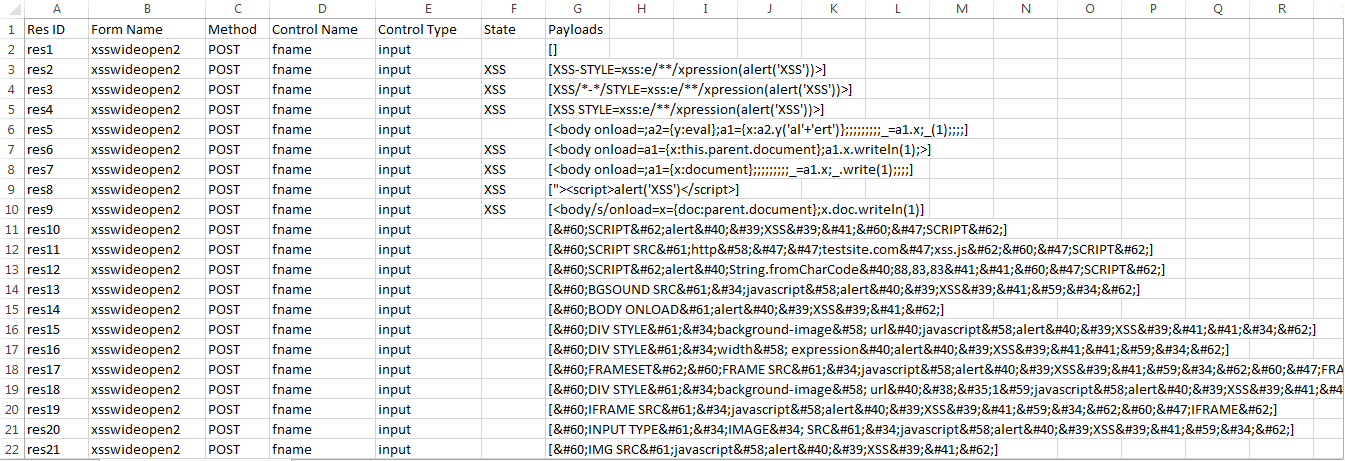


**Formdetail.csv**



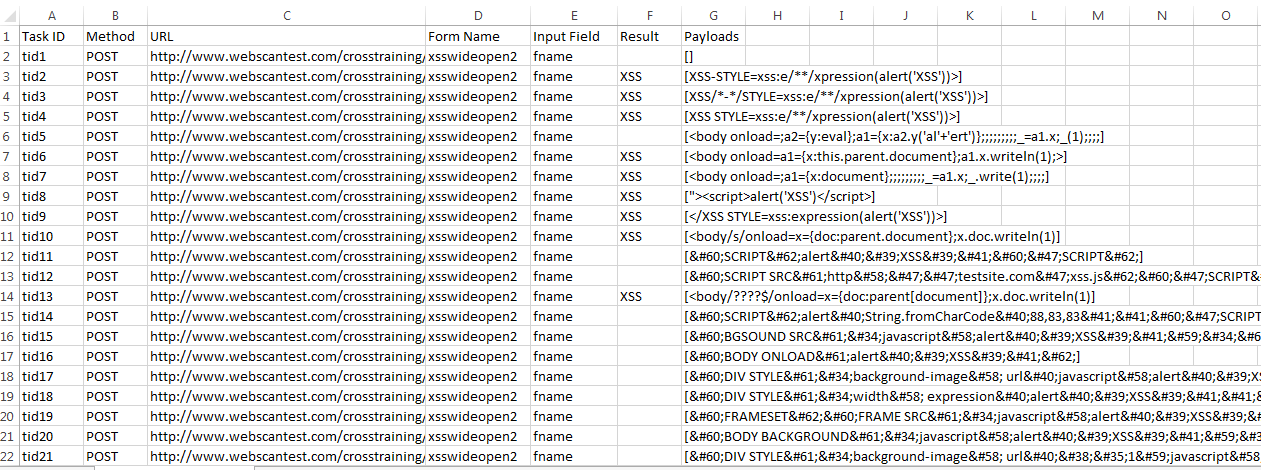
These form details were used as the pre data about the user inputs available at these site. The payloads scripts were injected in the forms and the responses were checked for every input section and for each script. If the browser response to that particular script at that particular input control, than it is suspected that part is XSS vulnerable.

**Penetration-testing dataset.csv**



Black box testing was then done using OWASP Tool for testing different web vulnerabilities with similar set of payloads.

**Black-box-testing dataset.csv**



**Percentage matching**

Through, white-box-penetration testing it was found out that for the given set of payloads, 32% of the responses were executing the script in the payload, concluding that this URL is vulnerable for Cross-Site-Scripting.

Unlike, white-box-penetration testing, black-box-testing through OWASP showed that there were more no. of responses than white-box-penetration and 34.5% of the responses were executing the scripts in the payload.

We can conclude that our white-box-penetration was able to detect the vulanerablity with ±5% vary in detecting the responses than done by OWASP tool.

**Conclusion & Future Scope**

Cross Site Scripting detection using penetration testing is a robust XSS prevention approach that was demonstrably effective on existing web browsers and most of the vulnerable websites. It lets us know which part or field where the user inputs are to be given in a given webpage is vulnerable to XSS or not. It is a good option for users to check using our tool whether their websites are XSS Vulnerable or not unlike the browser based techniques which takes long time for prevention.

In future our project can work dynamically across multiple sites checking multiple fields and other user input areas within domain as well as sub domains finding out the vulnerable parts exposed to XSS attack .Our project can work as standard tool for finding out the XSS vulnerability by integrating it to extension based browsers like Mozilla, Chrome, Safari so that functioning and surfing will be much more easier and safer.

**References**

* <https://technet.microsoft.com/en-us/library/cc512662.aspx>
* <http://ieeexplore.ieee.org/document/4276491/>
* <https://www.veracode.com/security/xss>
* <https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29>
* <https://www.incapsula.com/web-application-security/cross-site-scripting-xss-attacks.html>
* <https://excess-xss.com/>