Security Analysis of Online Bibliography System – iBib

Owen Yang, Alice Ho Yu Au-Yeung, Houtan Ziyaeimatin, Florence Tabamo, University of British Columbia

Abstract— **The iBib, an online bibliography system, requires the confidentiality, integrity, and availability (CIA) policies to be satisfied. In order to properly function as a trusted web application, iBib needs to be more secure. This paper addresses the problems associated with the security of iBib. Two vulnerabilities found were SQL injection and Cross Site Scripting (XSS). These security holes compromise the CIA policies when executed. In SQL injection, attackers are able to log in successfully as any user using an altered password. In XSS, simply knowing the structure of iBib can allow severe manipulation of the application. XSS attacks are also done by code injection. Solutions for these two vulnerabilities are presented, as well as recommendations for future works in iBib.**

Index Terms— **Cross Site Scripting (XSS), iBib, SQL Injection**

I. Introduction

Handling a huge amount of materials can be time-consuming as the number of paper works increases. A database designed to manage these information in an organized and systematic way would greatly decrease the effort needed to search for any information. By putting this system online would further increase its usefulness since access becomes much more convenient. iBib is designed to achieve all these goals—it is an online bibliography database system used to store and handle many different types of materials including papers, journals and other types of publications [1]. It provides many other useful functionalities such as searching for particular publication or display statistical values such as number of times a paper has been downloaded.

With many functionalities comes with many potential security flaws. Threats regarding iBib are mostly related to the languages used in development. iBib is mainly composed of two different components: PHP and MySQL. MySQL is used to query the database regarding information on papers and other publishing, while PHP presents this information in a human readable format at the same time allowing the information to be accessible via web browsers. The database and website are all susceptible to attacks. The following lists the potential vulnerabilities of iBib result from the nature of MySQL and PHP:

1. The database is vulnerable since MySQL has a weak authentication system in which an unauthorized user is allowed to gain full access to the database [2].
2. iBib is developed using PHP, where common mistakes can be exploited by a malicious user and compromise the application [3].

These vulnerabilities need to be addressed in order to improve the security of iBib. For example, we do not want to sacrifice confidentiality—invalid users should not be able to login to our accounts; we also want to maintain integrity—we do not want attackers to maliciously modify our code. Therefore, finding security vulnerabilities and invent ways to prevent these vulnerabilities are important.

In this report, we present two vulnerabilities we have discovered in iBib—SQL Injection Attack and Cross Site Scripting. Solutions to these vulnerabilities are also recommended.

II. Approach to Analysis

The first step was to get iBib running. The application requires older versions of PHP, Apache and MySQL which are known to contain bugs which makes them vulnerable on their own. Once installation was completed, research was done to learn more about web applications and their common vulnerabilities. From being more familiar (with great help from WebGoat [1]) with attacks on web applications, two vulnerabilities were found in iBib:

- SQL Injection
- Cross Site Scripting (XSS)

After testing these vulnerabilities in iBib, an analysis was conducted to figure out which solutions would best serve the security holes. The solutions were realized after studying the effects of SQL injection and Cross Site Scripting on the source codes, MySQL database, and application itself.

[1] Webgoat is a J2EE web application made to teach about the insecurities using intentionally vulnerable lessons [5].

III. SQL INJECTION

A. What is SQL Injection

SQL Injection is a serious threat to any database-driven system. It allows attackers to execute unauthorized Structured Query Language (SQL) by manipulating SQL queries to a Web form input box to gain access to resources or make changes to data [4]. For example, the following SQL query is used to authenticate all staff during login:

```
SELECT *
FROM staff
WHERE username = uname AND password = pwd
```

When a user tries to login, the username and password entered in the input fields will be used to execute the query. If the expected result is returned, the user is authenticated. However, most database system provides no mechanisms to validate inputs. As a result, attackers can use the input fields to manipulate the SQL query to achieve their purposes. The next section is going to illustrate how an SQL Injection attack is executed on iBib.

B. How SQL Injection Attack Exploits the Security of iBib

Users need to provide their username and password to authenticate themselves to iBib. The username and password that a user inputs in the login page are processed by a function called `verifySignon`, which is located in the file `classes/staffQuery.php`.

```
function verifySignon($username, $pwd) {
    $sql = "select * from staff";
    $sql = $sql." where username =
lower('".$username."') and pwd =
md5(lower('".$pwd."'))";
    $result = $this->_conn->exec($sql);
    if ($result == false) {
      $this->_errorOccurred = true;
      $this->_error = "Error verifying
username and password.";
      $this->_dbErrno =
$this->_conn->getDbErrno();
      $this->_dbError =
$this->_conn->getDbError();
      $this->_SQL = $sql;
      return false;
    }
    return $result;
}
```

The function executes an SQL query to verify the user. The query returns a row from the table `staff`, where the information of all staff is stored, when condition evaluates to true. The variables $username and $pwd are first replaced with the username and password entered by the user. Then the SQL query will be executed. For example, if the user enters “admin” as the username and “123456” as the password, the resulting SQL query is:

```
Select  *
From    staff
Where   username = admin AND pwd =
123456
```

After replacing the variables, the SQL query will be executed. If the condition evaluates to `false`, an appropriate error message is returned by the function.

When the `verifySignon` function was implemented, it assumes that user will always provide the valid username and password. However, it is not always the case. By noticing that the `verifySignon` function makes no effort to validate user inputs before replacing the variables in the SQL query, we attempt to login by entering the following in the password field:

```
wrong_pwd')) OR (('1'='1
```

as shown in figure 1.

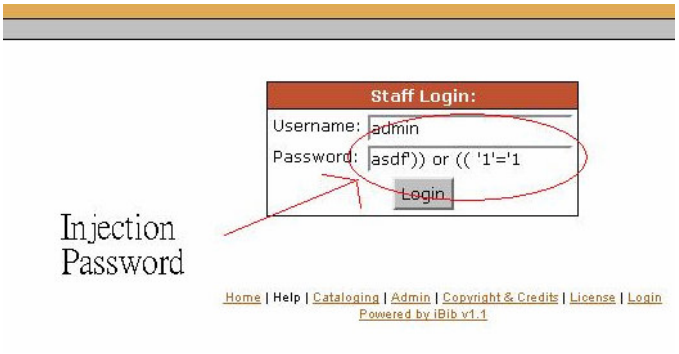

Fig 1. The input field where SQL injection takes place

Since the `verifySignon` function directly replaces the variables in the query with the user inputs, what we have entered manipulates the SQL query as follows:

```
SELECT  *
FROM    staff
WHERE (username = username_entered) AND
pwd=MD5(lower('wrong_pwd'))OR(('1'='1'
))
```

Note that this query will always evaluate to `true`, even though we have entered an obviously incorrect password. We have successfully log into iBib by maliciously manipulating the SQL query using the input field.

C. Solutions to SQL Injection

PHP provides a setting in its initialization file called “magic_quotes_gpc” (as shown in figure 2) to tackle the problem of SQL Injection.

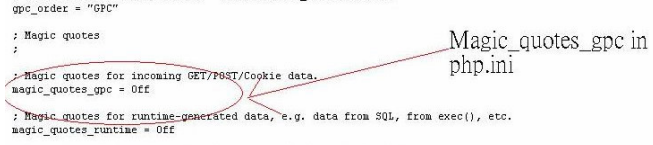

Fig 2. Line where magic_quotes_gpc is turned on or off.

However, this design violates the design principle of "Psychological Acceptability" because users tend to turn this function off since it automatically replaces all quotes with backslashes and makes the input texts difficult to comprehend. With "magic_quotes_gpc" turned on, the attack will return the following as demonstrated in figure 3:

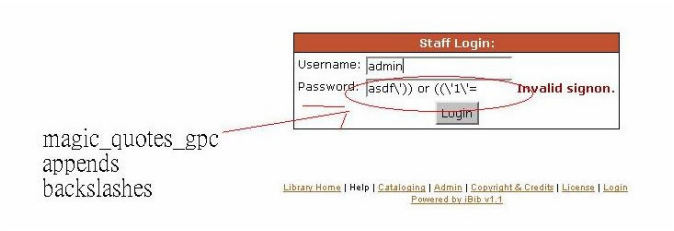

Fig 3. Effects of turning magic_quotes_gpc on.

Instead, we recommend the following solutions:

1) Use the `addslashes()` function.
   The addslashes() function can be used to manipulate user inputs before they are used in the SQL query. It returns a string with backslashes added in front of single quote ('), double quote ("), backslash (\) and NUL (the **`NULL`** byte). Use of addslashes() fucntion solves the problem introduced by "magic_quotes_gpc" since all manipulation of inputs is invisible to users.
2) Disallow single and double quotes as well as parentheses in the password text area.
   We are employing the design principle of "least privilege"—since users cannot include quotes and parentheses in their passwords, they do not need to enter these characters in the password field.
3) Parse user inputs for quotes and parentheses.
   We are employing the design principle of "complete mediation"—password is checked every time it is entered.

IV. Cross Site Scripting

A. *What is Cross Site Scripting?*

Cross site scripting (XSS) is a type of attack that can severely interfere with a web page's integrity. XSS attacks can also interfere with a web application's availability and confidentiality. XSS works with an attacker inserting malicious HTML, Java, and other scripts to have the web site, an assumed trusted source, to steal sensitive information and to crash servers. For example, the script:

```
<script> alert(document.cookie) </script>
```

will cause the web page's cookie to pop up on the internet browser. The reason XSS remains a problem is because the task of validating all user inputs is tedious and error-prone, especially for large applications [7].

B. *Cross Site Scripting in iBib*

Since iBib is an open source application, its directory structure and installation scripts are available to anybody who wishes to view them. The address in which the XSS attack takes place is http://localhost/webbiblio/install/index.php. The attacker can know where to look for the installation scripts and which fields can be modified. In this case, the "database name" field is where the code is injected. The steps for XSS in iBib are as follows:

1. The attacker opens the webpage `index.php` in the install folder. Same file name is used, except under different directory.

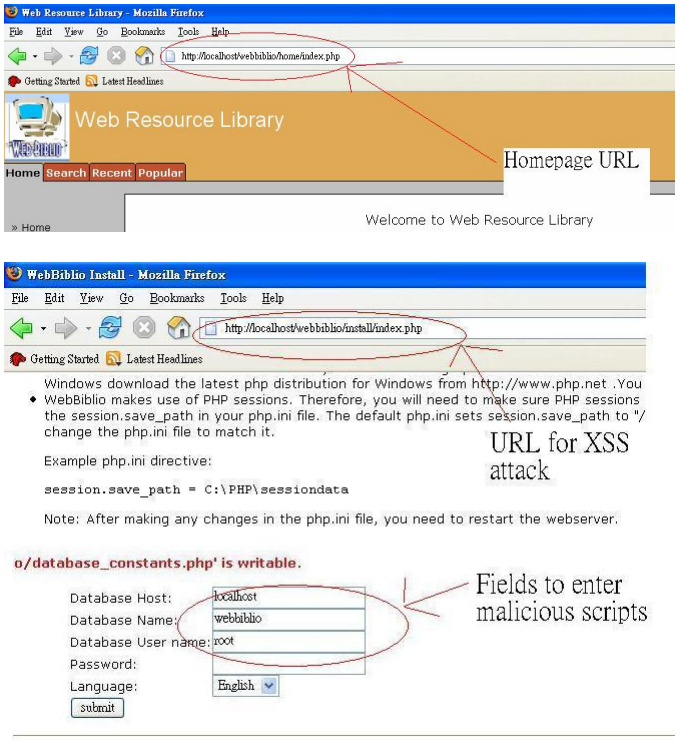

Fig 4. Input field where XSS attack can take place.

2. Once the webpage is open, the attacker can enter the database name as follows:

```
database_name");echo ("I CRACKED
your system.
```

Fig 5. Injected XSS code in iBib.

When attacker enters the above string, the phrase "`I CRACKED your system`" will be shown in all pages.

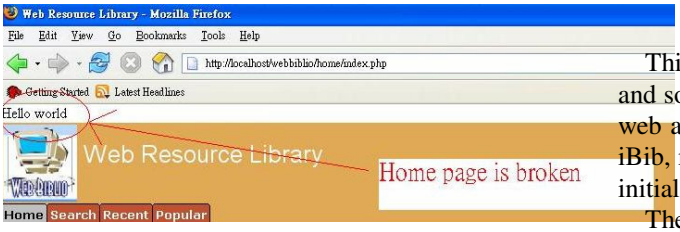

Fig 6. Result of XSS example.

The reason for this is in the file index.php, the following line exists:

```
<tr><td><font  class="primary">Database
Name:</font></td><td><input type="text"
name="db"
value="webbiblio"></td></tr>
```

index.php calls the file write_config.php that in turn, will write to database_constants.php. The user input will then be inserted within the file and so the result in database_constants.php is :

```
define("OBIB_DATABASE",   "webbiblio");
echo (“I CRACKED your system.”);
```

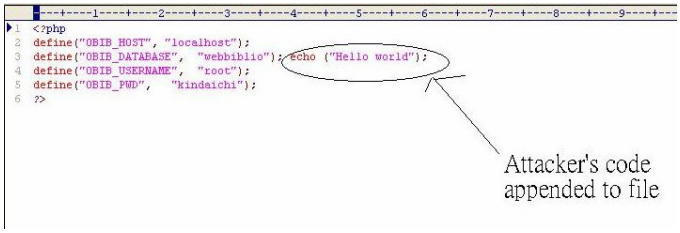

Fig 7. Changes shown if database_contants.php file.

This attack can be done in all the text fields provided in the address http://localhost/webbiblio/install/index.php. The example provided is a simple and harmless piece of code. If in fact, an attacker was to compromise the integrity of iBib, they would be able to inject their malicious script quite easily since there is no limit in the size of the string.

C. *Solution to XSS*

The solutions to the Cross Site Scripting problem are simple. The first solution is to make the file database_constants.php a read-only file. The reason for this is to disallow changes to be made to this file. The input field should only allow a limited number of characters so that long lines of code cannot be inserted. Also, the input should be parsed for illegal characters since this would be an indication of XSS. This will allows XSS attacks to be filtered out [8].

The second solution is to remove the installation folder to prevent anybody from accessing and maliciously modifying this folder. Once there is no access to the install folder, nobody can have access to the address http://localhost/webbiblio/install/index.php, where the XSS attack takes place.

In both solutions, database_constants.php is prevented from being modified since this is the file which is manipulated and where the injected code runs.

V. Discussion

This project was successful in that vulnerabilities were found and solutions were given. These vulnerabilities are common in web applications. If one was to further analyze the security of iBib, more security holes would be found because iBib is in its initial version.

The confidentiality, availability, and integrity (CIA) security policies would be compromised if any of SQL injection or XSS attacks were to be successful. This report presents solutions to the problems addressed. Using SQL injection, unauthorized log in was allowed – a feat which compromises the confidentiality of iBib. Only authorized users should be able to access their own accounts. Academic work could be submitted using another hacked account – resulting in deception of the integrity of the system. By doing an XSS attack, all of the CIA policies can be dishonored, depending on the degree of the attack. An XSS attack in iBib can range from simply appending a string onto the application, to inserting spy ware or crashing iBib itself.

VI. Recommendations

These two attacks are both of the same type which is code injection. One major recommendation is to validate user’s inputs as to prevent an attacker to inject a script, whether it be SQL injection or Cross Site Scripting. If the user’s inputs are checked for possible scripts, then the likelihood of the said attacks can be decreased or even eliminated completely.

The files containing crucial constants should not be modifiable.

Application of “least privilege” design principle should be enforced so that no access to the install folder is allowed after installation.

These recommendations brings iBib one step closer to being a more secure web application.

Acknowledgement

We would like to thank Konstantin (Kosta) Beznosov for taking the time to meet and help with queries regarding iBib.

References

[1] K. Beznosov. "Project_EECE496-IBib." LERSSE: Project_EECE496-IBib. 20 Apr. 2006. 4 Feb. 2007 https://lersse.ece.ubc.ca/tiki-index.php?page=Project_EECE496-iBib

[2] N. Davies. MySQL Vulnerabilities. 5 February 2007.http://www.linuxdevcenter.com/pub/a/linux/2002/12/16/insecurities.html

[3] P. Dickinson. "Top 7 PHP Security Blunders." Site Point. 21 Dec. 2005. 4 Feb. 2007http://www.sitepoint.com/article/php-security-blunders

[4] "Part IV. Security." PHP. 2001. The PHP Group. 4 Feb. 2007. http://www.php.net/manual/en/security.php

[5] "WebGoat Provides a Safe Place to Learn Application Security." Security. 3 Apr. 2006. ITworld.com. 4 Feb. 2007. http://security.itworld.com/4367/nls_security_webgoat_060404/page_1.html

[6] What is SQL Injection? http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci1003024,00.html

[7] D. Scott. and R. Sharp. Developing Secure Web Applications. *IEEE Internet Computing*. Vol. 6, Issue 6, Page 38, 2002.

[8] A. Nguyen-Tuong, S. Guarnieri, D. Greene, J. Shirley, and D. Evans. Automatically Hardening Web Applications Using Precise Tainting. *20th IFIP International Information Security Conference*. May 30 - June 1, 2005, Makuhari-Messe, Chiba, Japan.